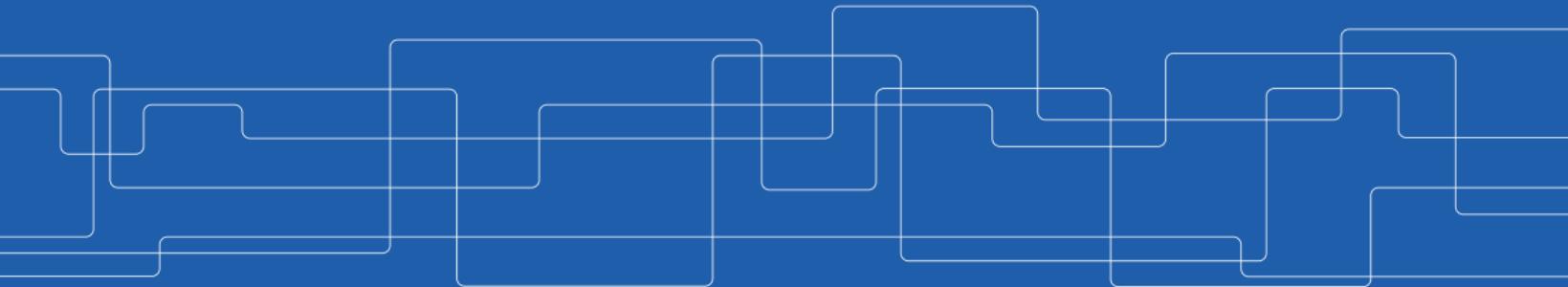




Introduction

Amir H. Payberah
payberah@kth.se
2021-11-03





Course Information



Course Objective

- ▶ This course has a **system-based** focus.



Course Objective

- ▶ This course has a **system-based** focus.
- ▶ Learn the theory of **machine learning** and **deep learning**.

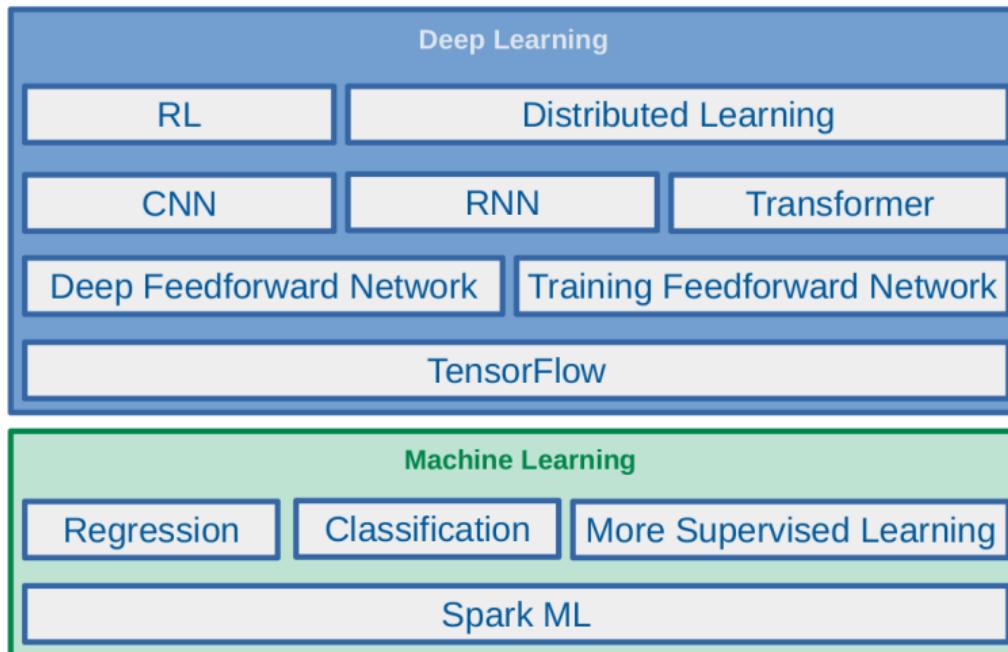


Course Objective

- ▶ This course has a **system-based** focus.
- ▶ Learn the theory of **machine learning** and **deep learning**.
- ▶ Learn the practical aspects of building **machine learning** and **deep learning** algorithms using data parallel programming platforms, such as **Spark** and **TensorFlow**.



Topics of Study





Intended Learning Outcomes (ILOs)

- ▶ **ILO1:** explain the **principles** of **ML/DL algorithms** and apply their techniques to solve problems.



Intended Learning Outcomes (ILOs)

- ▶ ILO1: explain the **principles** of **ML/DL algorithms** and apply their techniques to solve problems.
- ▶ ILO2: explain different **DNN architectures**, such as CNN, RNN, etc., and know how to build and train such networks.



Intended Learning Outcomes (ILOs)

- ▶ ILO1: explain the **principles** of **ML/DL algorithms** and apply their techniques to solve problems.
- ▶ ILO2: explain different **DNN architectures**, such as CNN, RNN, etc., and know how to build and train such networks.
- ▶ ILO3: explain the **principles** of distributed learning.



Intended Learning Outcomes (ILOs)

- ▶ ILO1: explain the **principles** of **ML/DL algorithms** and apply their techniques to solve problems.
- ▶ ILO2: explain different **DNN architectures**, such as CNN, RNN, etc., and know how to build and train such networks.
- ▶ ILO3: explain the **principles** of **distributed learning**.
- ▶ ILO4: implement **ML/DL algorithms** using **Spark** and **TensorFlow**.





The Course Assessment

- ▶ Task1: the review questions (P/F)



The Course Assessment

- ▶ Task1: the review questions (P/F)
- ▶ Task2: the lab assignments (A-F)



The Course Assessment

- ▶ Task1: the review questions (P/F)
- ▶ Task2: the lab assignments (A-F)
- ▶ Task3: the final project (A-F)



How Each ILO is Assessed?

	Task1	Task2	Task3
ILO1	x		
ILO2	x		
ILO3	x		
ILO4	x	x	x



Task1: The Review Questions (A-F)

- ▶ One review question **per week**.
- ▶ Questions about the **lectures**.
- ▶ The review questions are **graded (A-F)**.



Task2: The Lab Assignments (A-F)

- ▶ Two lab assignments: source code and oral presentation.



Task2: The Lab Assignments (A-F)

- ▶ Two lab assignments: source code and oral presentation.
- ▶ E: source code



Task2: The Lab Assignments (A-F)

- ▶ Two lab assignments: source code and oral presentation.
- ▶ E: source code
- ▶ D: source code + half questions (basic)



Task2: The Lab Assignments (A-F)

- ▶ Two lab assignments: source code and oral presentation.
- ▶ E: source code
- ▶ D: source code + half questions (basic)
- ▶ C: source code + all questions (basic)



Task2: The Lab Assignments (A-F)

- ▶ Two lab assignments: source code and oral presentation.
- ▶ E: source code
- ▶ D: source code + half questions (basic)
- ▶ C: source code + all questions (basic)
- ▶ B: source code + half questions (basic and advanced)



Task2: The Lab Assignments (A-F)

- ▶ Two lab assignments: source code and oral presentation.
- ▶ E: source code
- ▶ D: source code + half questions (basic)
- ▶ C: source code + all questions (basic)
- ▶ B: source code + half questions (basic and advanced)
- ▶ A: source code + all questions (basic and advanced)



Task3: The Final Project (A-F)

- ▶ One final project: source code and oral presentation.
- ▶ Proposed by students and confirmed by the teacher: A-level or C-level proposals.



Task3: The Final Project (A-F)

- ▶ One final project: source code and oral presentation.
- ▶ Proposed by students and confirmed by the teacher: A-level or C-level proposals.
- ▶ E: C-level source code



Task3: The Final Project (A-F)

- ▶ One final project: source code and oral presentation.
- ▶ Proposed by students and confirmed by the teacher: A-level or C-level proposals.
- ▶ E: C-level source code
- ▶ D: C-level source code + half questions (basic and advanced)



Task3: The Final Project (A-F)

- ▶ One final project: source code and oral presentation.
- ▶ Proposed by students and confirmed by the teacher: A-level or C-level proposals.
- ▶ E: C-level source code
- ▶ D: C-level source code + half questions (basic and advanced)
- ▶ C: C-level source code + all questions (basic and advanced) or A-level source code + all questions (basic)



Task3: The Final Project (A-F)

- ▶ One final project: source code and oral presentation.
- ▶ Proposed by students and confirmed by the teacher: A-level or C-level proposals.
- ▶ E: C-level source code
- ▶ D: C-level source code + half questions (basic and advanced)
- ▶ C: C-level source code + all questions (basic and advanced) or A-level source code + all questions (basic)
- ▶ B: A-level source code + half questions (basic and advanced)



Task3: The Final Project (A-F)

- ▶ One final project: source code and oral presentation.
- ▶ Proposed by students and confirmed by the teacher: A-level or C-level proposals.
- ▶ E: C-level source code
- ▶ D: C-level source code + half questions (basic and advanced)
- ▶ C: C-level source code + all questions (basic and advanced) or A-level source code + all questions (basic)
- ▶ B: A-level source code + half questions (basic and advanced)
- ▶ A: A-level source code + all questions (basic and advanced)



The Final Grade

- ▶ The **final grade** is the **weighted average** of the **review questions** (0.2), **two labs** (0.25 each), and the **final project** (0.3).



The Final Grade

- ▶ The **final grade** is the **weighted average** of the **review questions** (0.2), **two labs** (0.25 each), and the **final project** (0.3).
- ▶ To compute it, map **A-E** to **5-1**, and take the average.



The Final Grade

- ▶ The **final grade** is the **weighted average** of the **review questions** (0.2), **two labs** (0.25 each), and the **final project** (0.3).
- ▶ To compute it, map **A-E** to **5-1**, and take the average.
- ▶ The floating values are **rounded up**, if they are **more than half**, otherwise they are **rounded down**.
 - E.g., **3.6** will be rounded to **4**, and **4.5** will be rounded to **4**.



The Final Grade

- ▶ The **final grade** is the **weighted average** of the **review questions** (0.2), **two labs** (0.25 each), and the **final project** (0.3).
- ▶ To compute it, map **A-E** to **5-1**, and take the average.
- ▶ The floating values are **rounded up**, if they are **more than half**, otherwise they are **rounded down**.
 - E.g., **3.6** will be rounded to **4**, and **4.5** will be rounded to **4**.
- ▶ A **late** submission will **reduce you grade level by one**. That is, A will become B, B will become C, and so on.



The Final Grade

- ▶ The **final grade** is the **weighted average** of the **review questions** (0.2), **two labs** (0.25 each), and the **final project** (0.3).
- ▶ To compute it, map **A-E** to **5-1**, and take the average.
- ▶ The floating values are **rounded up**, if they are **more than half**, otherwise they are **rounded down**.
 - E.g., 3.6 will be rounded to 4, and 4.5 will be rounded to 4.
- ▶ A **late** submission will **reduce you grade level by one**. That is, A will become B, B will become C, and so on.
- ▶ To pass the course, you need to take at least **E** in all the assignments.

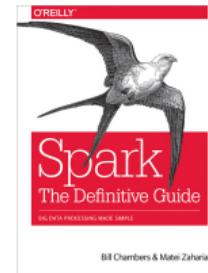
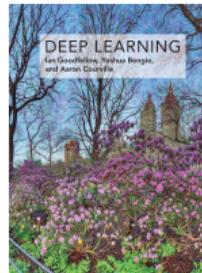
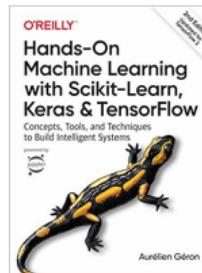
How to Submit the Assignments?

- ▶ Through the [Canvas](#) site.
- ▶ Students will work in **groups of two** on all the [Tasks 1-4](#).



The Course Material

- ▶ [Hands-on machine learning with Scikit-Learn and TensorFlow, 2nd Edition](#), A. Geron, O'Reilly Media, 2019
- ▶ [Deep learning](#), I. Goodfellow et al., Cambridge: MIT press, 2016
- ▶ [Spark - The Definitive Guide](#), M. Zaharia et al., O'Reilly Media, 2018.





The Course Web Page

<https://id2223kth.github.io>



The Questions-Answers Page

<https://tinyurl.com/6s5jy46a>



The Course Overview

Sheepdog or Mop





Chihuahua or Muffin



@teenybiscuit

Barn Owl or Apple





Artificial Intelligence Challenge

- ▶ Artificial intelligence (AI) can solve problems that can be described by a list of formal mathematical rules.



Artificial Intelligence Challenge

- ▶ Artificial intelligence (AI) can solve problems that can be described by a list of formal mathematical rules.
- ▶ The challenge is to solve the tasks that are hard for people to describe formally.



Artificial Intelligence Challenge

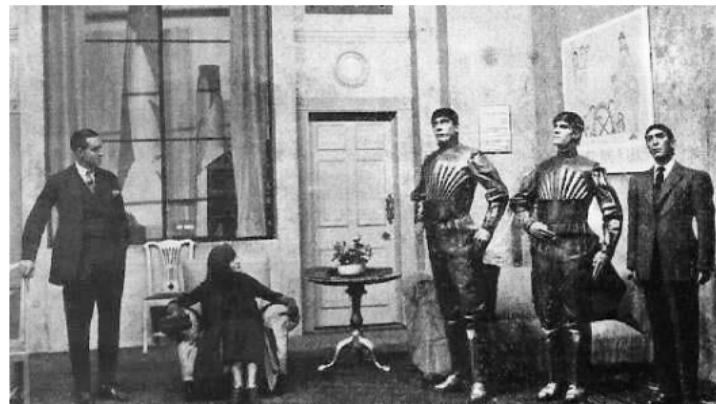
- ▶ Artificial intelligence (AI) can solve problems that can be described by a list of formal mathematical rules.
- ▶ The challenge is to solve the tasks that are hard for people to describe formally.
- ▶ Let computers to learn from experience.



History of AI

1920: Rossum's Universal Robots (R.U.R.)

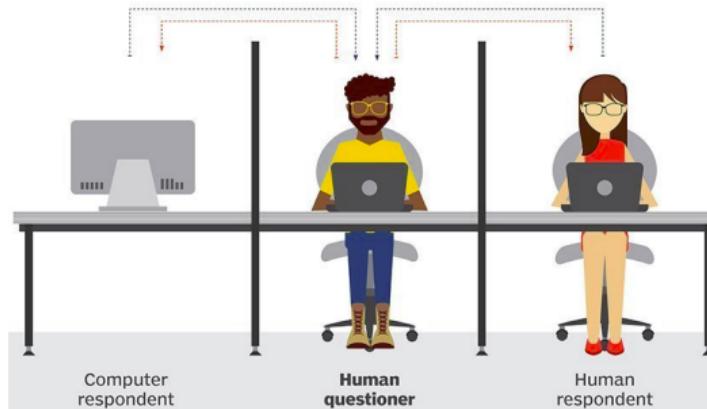
- ▶ A science fiction play by Karel Čapek, in 1920.
- ▶ A factory that creates artificial people named robots.



[<https://dev.to/lshultebraucks/a-short-history-of-artificial-intelligence-7hm>]

1950: Turing Test

- ▶ In 1950, **Turing** introduced the **Turing test**.
- ▶ An attempt to define **machine intelligence**.



[<https://searchenterpriseai.techtarget.com/definition/Turing-test>]



1956: The Dartmouth Workshop

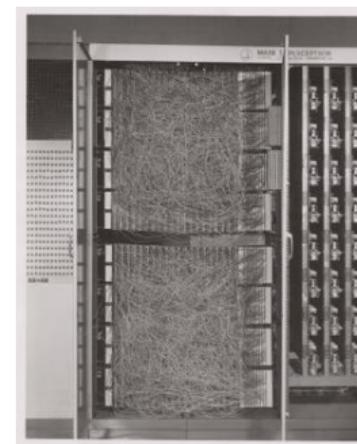
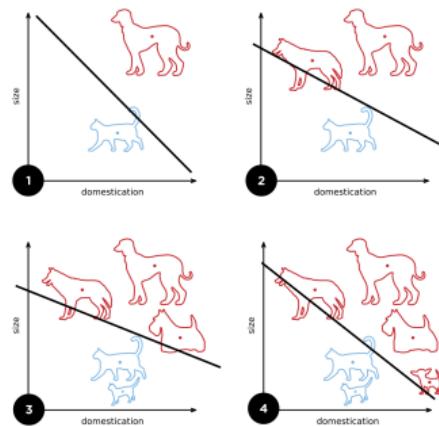
- ▶ Probably the **first workshop of AI**.
- ▶ Researchers from **CMU, MIT, IBM** met together and founded the **AI research**.



[<https://twitter.com/lordsaicom/status/898139880441696257>]

1958: Perceptron

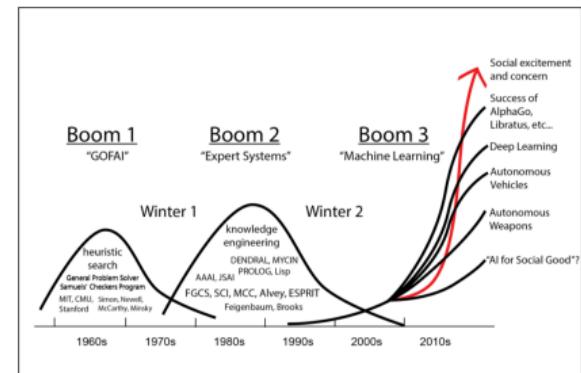
- ▶ A **supervised learning** algorithm for **binary classifiers**.
- ▶ Implemented in custom-built hardware as the **Mark 1 perceptron**.



[<https://en.wikipedia.org/wiki/Perceptron>]

1974–1980: The First AI Winter

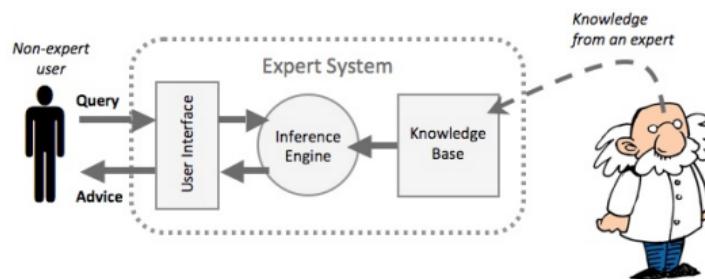
- ▶ The over **optimistic settings**, which were not occurred
- ▶ The **problems**:
 - Limited **computer power**
 - Lack of **data**
 - Intractability and the **combinatorial explosion**



[<http://www.technologystories.org/ai-evolution>]

1980's: Expert systems

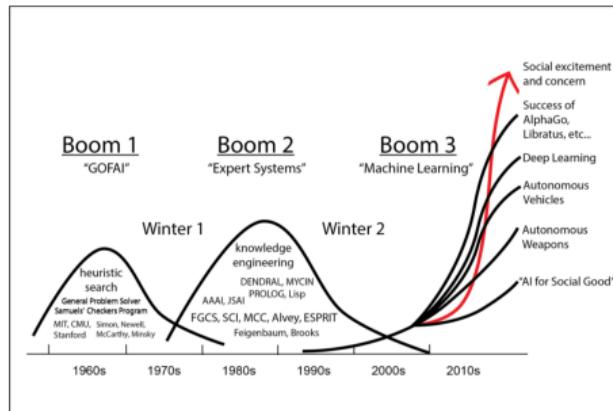
- ▶ The programs that solve problems in a **specific domain**.
- ▶ **Two** engines:
 - Knowledge engine: **represents** the **facts and rules** about a specific topic.
 - Inference engine: **applies** the **facts and rules** from the knowledge engine to new facts.



[https://www.igcseict.info/theory/7_2/expert]

1987–1993: The Second AI Winter

- ▶ After a series of financial setbacks.
- ▶ The fall of **expert systems** and hardware companies.



[<http://www.technologystories.org/ai-evolution>]



1997: IBM Deep Blue

- ▶ The first chess computer to beat a world chess champion Garry Kasparov.



[<http://marksist.org/icerik/Tarihte-Bugun/1757/11-Mayis-1997-Deep-Blue-adli-bilgisayar>]



2012: AlexNet - Image Recognition

- ▶ The [ImageNet competition](#) in image classification.
- ▶ The [AlexNet Convolutional Neural Network \(CNN\)](#) won the challenge by a large margin.

IM_{AG}ENET

2016: DeepMind AlphaGo

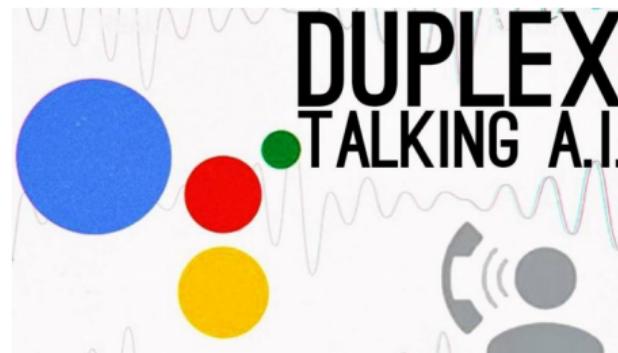
- ▶ DeepMind **AlphaGo** won **Lee Sedol**, one of the best players at Go.
- ▶ In 2017, DeepMind published **AlphaGo Zero**.
 - The **next generation** of AlphaGo.
 - It learned Go by playing **against itself**.



[<https://www.zdnet.com/article/google-alphago-caps-victory-by-winning-final-historic-go-match>]

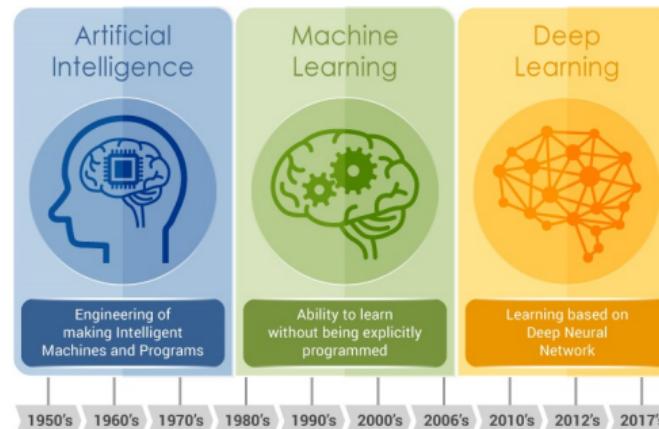
2018: Google Duplex

- ▶ An AI system for accomplishing real-world tasks over the phone.
- ▶ A Recurrent Neural Network (RNN) built using TensorFlow.



AI Generations

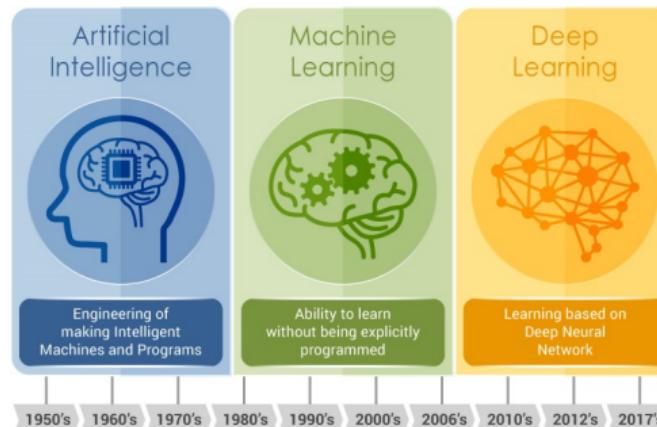
- ▶ Rule-based AI
- ▶ Machine learning
- ▶ Deep learning



[<https://bit.ly/2woLEzs>]

AI Generations - Rule-based AI

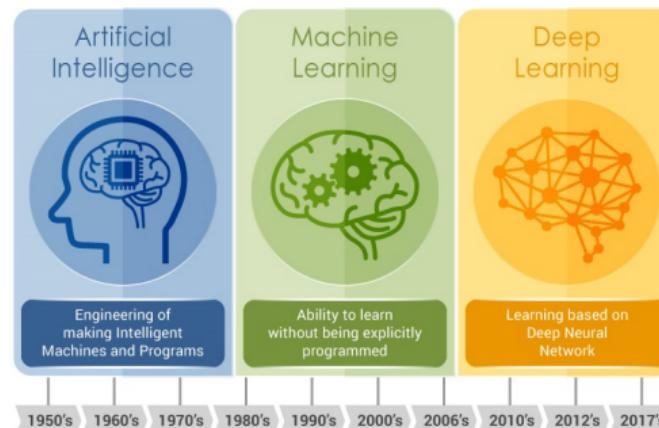
- ▶ Hard-code knowledge
- ▶ Computers reason using logical inference rules



[<https://bit.ly/2woLEzs>]

AI Generations - Machine Learning

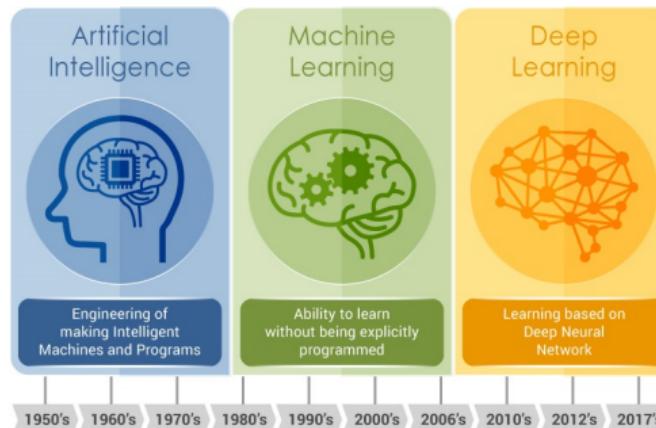
- ▶ If AI systems acquire **their own knowledge**
- ▶ **Learn from data without being explicitly programmed**



[<https://bit.ly/2woLEzs>]

AI Generations - Deep Learning

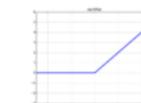
- ▶ For many tasks, it is **difficult to know what features** should be extracted
- ▶ Use **machine learning** to **discover** the mapping from **representation to output**



[<https://bit.ly/2woLEzs>]

Why Does Deep Learning Work Now?

- ▶ Huge **quantity** of data
- ▶ Tremendous increase in **computing power**
- ▶ Better **training** algorithms



Non-Linearity



Machine Learning and Deep Learning



Learning Algorithms

- ▶ A **ML algorithm** is an algorithm that is able to **learn** from data.
- ▶ What is **learning**?

Learning Algorithms

- ▶ A **ML algorithm** is an algorithm that is able to **learn from data**.
- ▶ What is **learning**?
- ▶ A computer program is said to **learn** from **experience E** with respect to some class of **tasks T** and **performance measure P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**. (Tom M. Mitchell)



Learning Algorithms - Example 1

- ▶ A **spam filter** that can learn to flag **spam** given examples of **spam emails** and examples of **regular emails**.



[<https://bit.ly/2oiplYM>]

Learning Algorithms - Example 1

- ▶ A **spam filter** that can learn to flag **spam** given examples of **spam emails** and examples of **regular emails**.
- ▶ **Task T**: flag spam for new emails
- ▶ **Experience E**: the training data
- ▶ **Performance measure P**: the ratio of correctly classified emails



[<https://bit.ly/2oiplYM>]

Learning Algorithms - Example 2

- ▶ Given dataset of prices of 500 houses, how can we learn to **predict the prices** of other houses, as a **function of the size of their living areas**?



[<https://bit.ly/2MyiJUy>]

Learning Algorithms - Example 2

- ▶ Given dataset of prices of 500 houses, how can we learn to **predict the prices** of other houses, as a **function of the size of their living areas?**
- ▶ **Task T:** predict the price
- ▶ **Experience E:** the dataset of living areas and prices
- ▶ **Performance measure P:** the difference between the predicted price and the real price



[<https://bit.ly/2MyiJUy>]

Types of Machine Learning Algorithms

- ▶ Supervised learning
- ▶ Unsupervised learning



Types of Machine Learning Algorithms

► Supervised learning

- Input data is **labeled**, e.g., spam/not-spam or a stock price at a time.
- Regression vs. classification

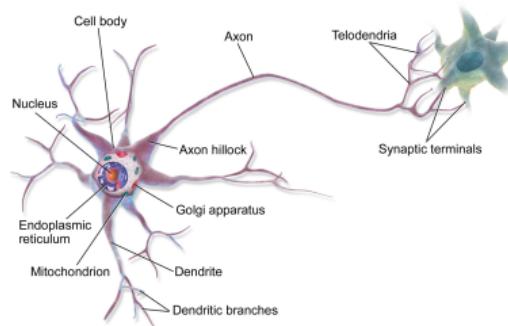
► Unsupervised learning

- Input data is **unlabeled**.
- Find **hidden structures** in data.



From Machine Learning to Deep Learning

- ▶ Deep Learning (DL) is part of ML methods based on learning data representations.
- ▶ Mimic the neural networks of our brain.



[A. Geron, O'Reilly Media, 2017]



Artificial Neural Networks

- ▶ Artificial Neural Network (ANN) is inspired by biological neurons.

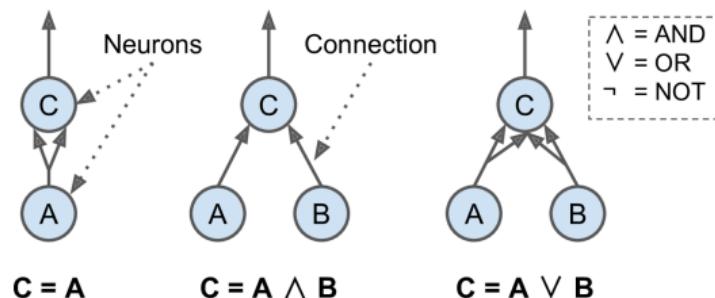


Artificial Neural Networks

- ▶ Artificial Neural Network (ANN) is inspired by biological neurons.
- ▶ One or more binary inputs and one binary output

Artificial Neural Networks

- ▶ Artificial Neural Network (ANN) is inspired by biological neurons.
- ▶ One or more binary inputs and one binary output
- ▶ Activates its output when more than a certain number of its inputs are active.



[A. Geron, O'Reilly Media, 2017]

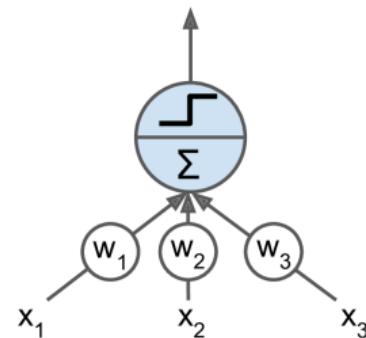


The Linear Threshold Unit (LTU)

- ▶ Inputs of a LTU are **numbers** (**not binary**).

The Linear Threshold Unit (LTU)

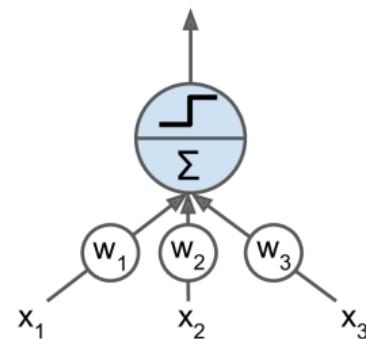
- ▶ Inputs of a LTU are **numbers** (not binary).
- ▶ Each **input connection** is associated with a **weight**.
- ▶ Computes a **weighted sum of its inputs** and applies a **step function** to that **sum**.



The Linear Threshold Unit (LTU)

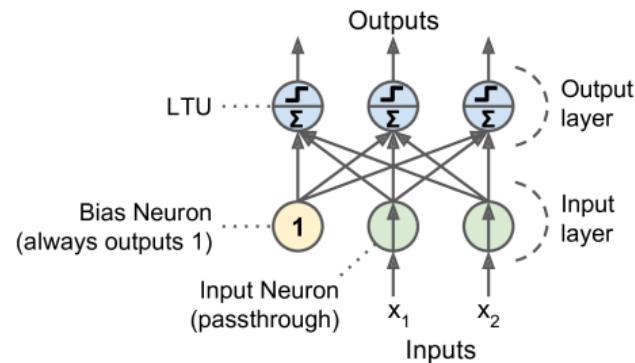
- ▶ Inputs of a LTU are **numbers** (not binary).
- ▶ Each **input connection** is associated with a **weight**.
- ▶ Computes a **weighted sum of its inputs** and applies a **step function** to that **sum**.

- ▶ $z = w_1x_1 + w_2x_2 + \dots + w_nx_n = w^T x$
- ▶ $\hat{y} = \text{step}(z) = \text{step}(w^T x)$



The Perceptron

- ▶ The **perceptron** is a **single layer** of LTUs.
- ▶ The **input neurons** output whatever **input** they are fed.
- ▶ A **bias neuron**, which just **outputs 1** all the time.





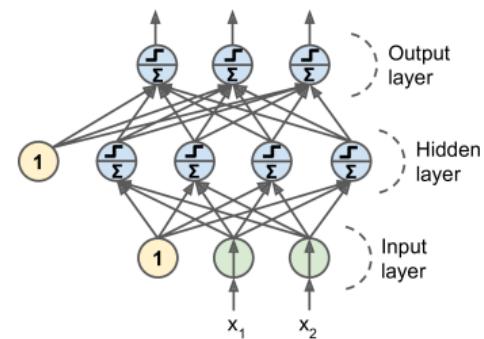
Deep Learning Models

- ▶ Deep Neural Network ([DNN](#))
- ▶ Convolutional Neural Network ([CNN](#))
- ▶ Recurrent Neural Network ([RNN](#))
- ▶ Transformer
- ▶ Deep Reinforcement Learning

Deep Neural Networks

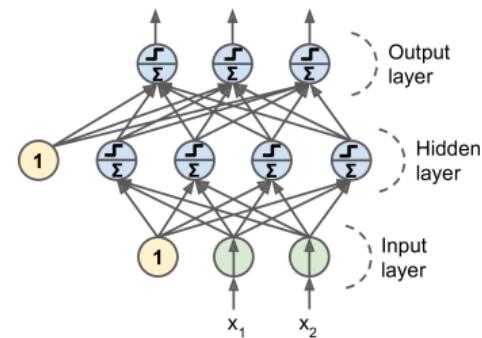
► Multi-Layer Perceptron (MLP)

- One input layer.
- One or more layers of LTUs (hidden layers).
- One final layer of LTUs (output layer).



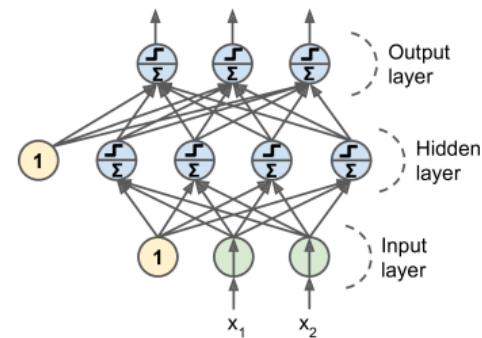
Deep Neural Networks

- ▶ Multi-Layer Perceptron (MLP)
 - One input layer.
 - One or more layers of LTUs (hidden layers).
 - One final layer of LTUs (output layer).
- ▶ Deep Neural Network (DNN) is an ANN with two or more hidden layers.



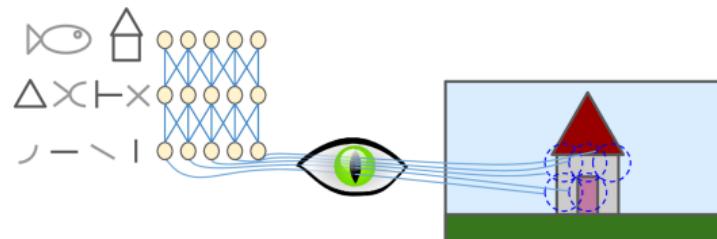
Deep Neural Networks

- ▶ Multi-Layer Perceptron (MLP)
 - One input layer.
 - One or more layers of LTUs (hidden layers).
 - One final layer of LTUs (output layer).
- ▶ Deep Neural Network (DNN) is an ANN with two or more hidden layers.
- ▶ Backpropagation training algorithm.



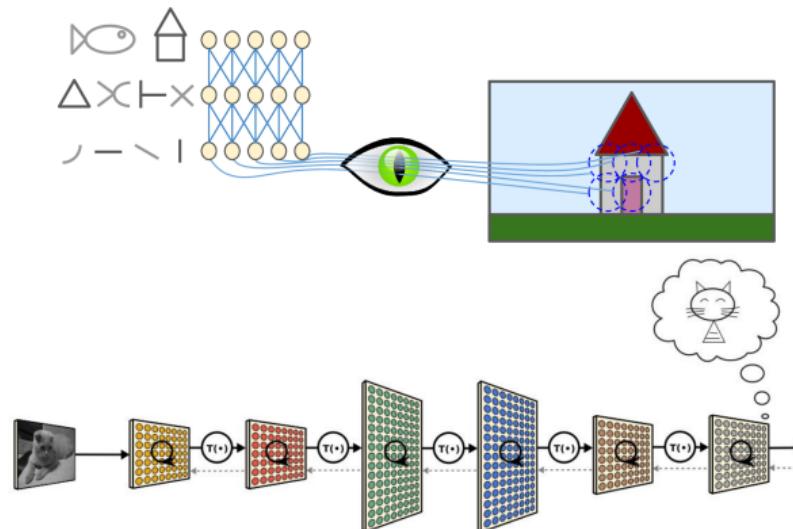
Convolutional Neural Networks

- ▶ Many neurons in the **visual cortex** react only to a **limited region** of the visual field.



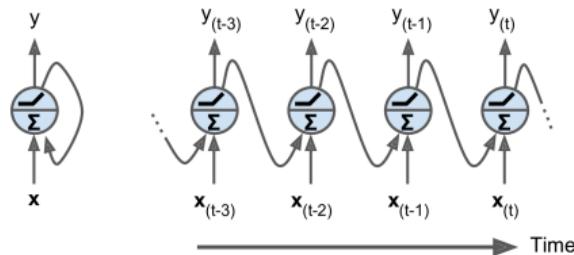
Convolutional Neural Networks

- ▶ Many neurons in the **visual cortex** react only to a **limited region** of the visual field.
- ▶ The **higher-level** neurons are based on the outputs of **neighboring lower-level** neurons.



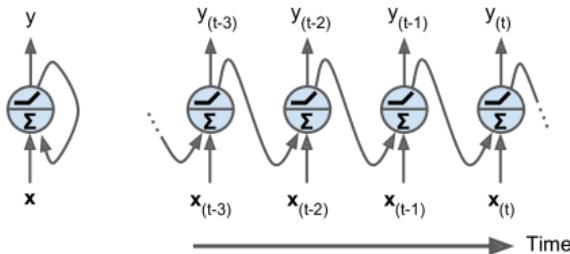
Recurrent Neural Networks

- ▶ The **output** depends on the **input** and the **previous computations**.



Recurrent Neural Networks

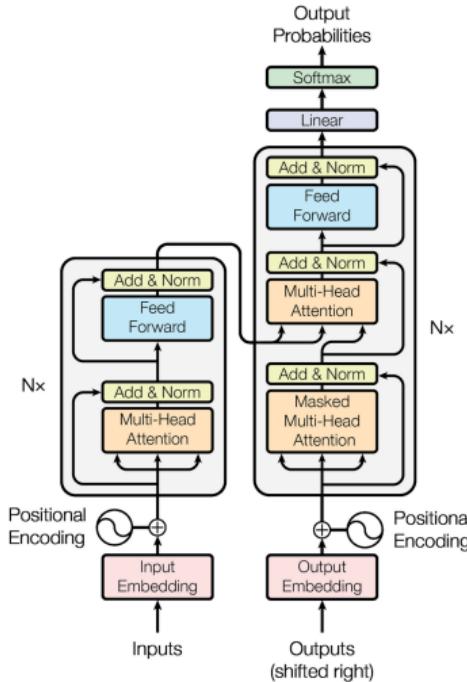
- ▶ The **output** depends on the **input** and the **previous computations**.



- ▶ Analyze **time series data**, e.g., stock market, and autonomous driving systems.
- ▶ Work on sequences of **arbitrary lengths**, rather than on **fixed-sized inputs**.



Transformer



[A. Vaswani et al., Attention Is All You Need, 2017]



Linear Algebra Review

Vector

- ▶ A **vector** is an **array of numbers**.
- ▶ **Notation:**
 - Denoted by **bold lowercase letters**, e.g., \mathbf{x} .
 - \mathbf{x}_i denotes the **i**th entry.

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}$$

Matrix and Tensor

- ▶ A **matrix** is a 2-D array of numbers.
- ▶ A **tensor** is an array with more than two axes.
- ▶ Notation:
 - Denoted by **bold uppercase letters**, e.g., \mathbf{A} .
 - a_{ij} denotes the entry in i th row and j th column.
 - If \mathbf{A} is $m \times n$, it has m rows and n columns.

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \dots & a_{m,n} \end{bmatrix}$$



Matrix Addition and Subtraction

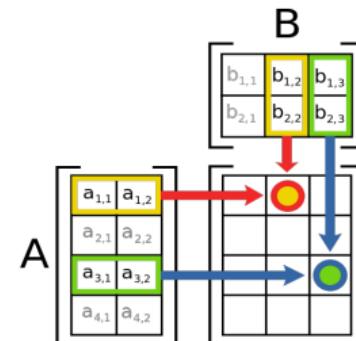
- The **matrices** must have the **same dimensions**.

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}$$

Matrix Product

- ▶ The **matrix product** of matrices **A** and **B** is a third matrix **C**, where $\mathbf{C} = \mathbf{AB}$.
- ▶ If **A** is of shape $m \times n$ and **B** is of shape $n \times p$, then **C** is of shape $m \times p$.

$$c_{ij} = \sum_k a_{ik} b_{kj}$$



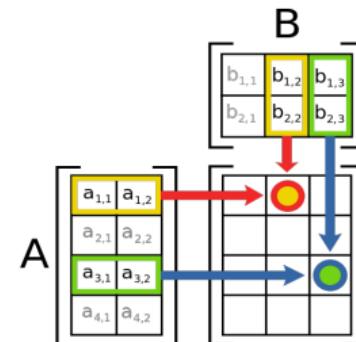
[https://en.wikipedia.org/wiki/Matrix_multiplication]

Matrix Product

- ▶ The **matrix product** of matrices **A** and **B** is a third matrix **C**, where $\mathbf{C} = \mathbf{AB}$.
- ▶ If **A** is of shape $m \times n$ and **B** is of shape $n \times p$, then **C** is of shape $m \times p$.

$$c_{ij} = \sum_k a_{ik} b_{kj}$$

- ▶ Properties
 - Associative: $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$
 - Not commutative: $\mathbf{AB} \neq \mathbf{BA}$



[https://en.wikipedia.org/wiki/Matrix_multiplication]



Matrix Transpose

- ▶ Swap the rows and columns of a matrix.

$$A = \begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} a & c & e \\ b & d & f \end{bmatrix}$$

Matrix Transpose

- ▶ Swap the rows and columns of a matrix.

$$A = \begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} a & c & e \\ b & d & f \end{bmatrix}$$

- ▶ Properties

- $A_{ij} = A_{ji}^T$
- If A is $m \times n$, then A^T is $n \times m$
- $(A + B)^T = A^T + B^T$
- $(AB)^T = B^T A^T$



Inverse of a Matrix

- If A is a **square** matrix, its **inverse** is called A^{-1} .

$$AA^{-1} = A^{-1}A = I$$

- Where I , the **identity** matrix, is a **diagonal matrix** with all **1's** on the diagonal.

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



L^p Norm for Vectors

- ▶ We can measure the **size of vectors** using a **norm** function.
- ▶ Norms are functions **mapping vectors to non-negative values**.
- ▶ L¹ norm

$$\|x\|_1 = \sum_i |x_i|$$



L^p Norm for Vectors

- We can measure the **size of vectors** using a **norm** function.
- Norms are functions **mapping vectors to non-negative values**.
- L¹ norm

$$\|x\|_1 = \sum_i |x_i|$$

- L² norm

$$\|x\|_2 = \left(\sum_i |x_i|^2 \right)^{\frac{1}{2}} = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$$



L^p Norm for Vectors

- ▶ We can measure the **size of vectors** using a **norm** function.
- ▶ Norms are functions **mapping vectors to non-negative values**.
- ▶ L¹ norm

$$\|x\|_1 = \sum_i |x_i|$$

- ▶ L² norm

$$\|x\|_2 = \left(\sum_i |x_i|^2 \right)^{\frac{1}{2}} = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$$

- ▶ L^p norm

$$\|x\|_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}}$$



Probability Review



Random Variables

- ▶ **Random variable:** a **variable** that can take on **different values randomly**.
- ▶ Random variables may be **discrete** or **continuous**.



Random Variables

- ▶ Random variable: a **variable** that can take on **different values randomly**.
- ▶ Random variables may be **discrete** or **continuous**.
 - Discrete random variable: **finite** or **countably infinite number of states**



Random Variables

- ▶ Random variable: a **variable** that can take on **different values randomly**.
- ▶ Random variables may be **discrete** or **continuous**.
 - Discrete random variable: **finite or countably infinite number of states**
 - Continuous random variable: **real value**



Random Variables

- ▶ Random variable: a **variable** that can take on **different values randomly**.
- ▶ Random variables may be **discrete** or **continuous**.
 - Discrete random variable: **finite or countably infinite number of states**
 - Continuous random variable: **real value**
- ▶ Notation:
 - Denoted by an **upper case letter**, e.g., **X**
 - Values of a random variable **X** are denoted by **lower case letters**, e.g., **x** and **y**.



Probability Distributions

- ▶ **Probability distribution:** how likely a **random variable** is to take on each of its **possible states**.



Probability Distributions

- ▶ **Probability distribution:** how likely a **random variable** is to take on each of its **possible states**.
 - E.g., the **random variable** **X** denotes the outcome of a **coin toss**.



Probability Distributions

- ▶ **Probability distribution:** how likely a **random variable** is to take on each of its **possible states**.
 - E.g., the **random variable** **X** denotes the outcome of a **coin toss**.
 - The **probability distribution** of **X** would take the value **0.5** for **X = head**, and **0.5** for **Y = tail** (assuming the coin is **fair**).



Probability Distributions

- ▶ **Probability distribution:** how likely a **random variable** is to take on each of its **possible states**.
 - E.g., the **random variable** **X** denotes the outcome of a **coin toss**.
 - The **probability distribution** of **X** would take the value **0.5** for **X = head**, and **0.5** for **Y = tail** (assuming the coin is **fair**).
- ▶ The way we **describe probability distributions** depends on whether the variables are **discrete** or **continuous**.



Discrete Variables

- ▶ Probability mass function (PMF): the probability distribution of a discrete random variable X .
- ▶ Notation: denoted by a lowercase p .



Discrete Variables

- ▶ Probability mass function (PMF): the probability distribution of a discrete random variable X .
- ▶ Notation: denoted by a lowercase p .
 - E.g., $p(x) = 1$ indicates that $X = x$ is certain
 - E.g., $p(x) = 0$ indicates that $X = x$ is impossible



Discrete Variables

- ▶ Probability mass function (PMF): the probability distribution of a discrete random variable X .
- ▶ Notation: denoted by a lowercase p .
 - E.g., $p(x) = 1$ indicates that $X = x$ is certain
 - E.g., $p(x) = 0$ indicates that $X = x$ is impossible
- ▶ Properties:
 - The domain D of p must be the set of all possible states of X
 - $\forall x \in D(X), 0 \leq p(x) \leq 1$
 - $\sum_{x \in D(X)} p(x) = 1$



Independence

- ▶ Two random variables X and Y are **independent**, if their **probability distribution** can be expressed as their **products**.

$$\forall x \in D(X), y \in D(Y), p(X = x, Y = y) = p(X = x)p(Y = y)$$



Independence

- ▶ Two random variables X and Y are **independent**, if their **probability distribution** can be expressed as their **products**.

$$\forall x \in D(X), y \in D(Y), p(X = x, Y = y) = p(X = x)p(Y = y)$$

- ▶ E.g., if a **coin is tossed** and a single **6-sided die is rolled**, then the probability of landing on the **head** side of the coin and **rolling a 3** on the die is:

Independence

- ▶ Two random variables X and Y are **independent**, if their **probability distribution** can be expressed as their **products**.

$$\forall x \in D(X), y \in D(Y), p(X = x, Y = y) = p(X = x)p(Y = y)$$

- ▶ E.g., if a **coin is tossed** and a single **6-sided die is rolled**, then the probability of landing on the **head** side of the coin and **rolling a 3** on the die is:

$$p(X = \text{head}, Y = 3) = p(X = \text{head})p(Y = 3) = \frac{1}{2} \times \frac{1}{6} = \frac{1}{12}$$



Conditional Probability

- ▶ **Conditional probability:** the probability of an event given that another event has occurred.

$$p(Y = y \mid X = x) = \frac{p(Y = y, X = x)}{p(X = x)}$$



Conditional Probability

- ▶ **Conditional probability:** the probability of an event given that another event has occurred.

$$p(Y = y \mid X = x) = \frac{p(Y = y, X = x)}{p(X = x)}$$

- ▶ E.g., if 60% of the class passed both labs and 80% of the class passed the first labs, then what percent of those who passed the first lab also passed the second lab?



Conditional Probability

- ▶ **Conditional probability:** the probability of an event given that another event has occurred.

$$p(Y = y \mid X = x) = \frac{p(Y = y, X = x)}{p(X = x)}$$

- ▶ E.g., if 60% of the class passed both labs and 80% of the class passed the first labs, then what percent of those who passed the first lab also passed the second lab?
 - E.g., X and Y random variables for the first and the second labs, respectively.

$$p(Y = \text{lab2} \mid X = \text{lab1}) = \frac{p(Y = \text{lab2}, X = \text{lab1})}{p(X = \text{lab1})} = \frac{0.6}{0.8} = \frac{3}{4}$$



Expectation

- ▶ The **expected value** of a random variable X with respect to a probability distribution $p(x)$ is the **average** value that X takes on when it is drawn from $p(x)$.

$$E_{x \sim p}[X] = \sum_x p(x)x$$



Expectation

- ▶ The **expected value** of a random variable X with respect to a probability distribution $p(x)$ is the **average** value that X takes on when it is drawn from $p(x)$.

$$E_{x \sim p}[X] = \sum_x p(x)x$$

- ▶ E.g., If $X : \{1, 2, 3\}$, and $p(X = 1) = 0.3$, $p(X = 2) = 0.5$, $p(X = 3) = 0.2$

Expectation

- ▶ The **expected value** of a random variable X with respect to a probability distribution $p(x)$ is the **average** value that X takes on when it is drawn from $p(x)$.

$$E_{x \sim p}[X] = \sum_x p(x)x$$

- ▶ E.g., If $X : \{1, 2, 3\}$, and $p(X = 1) = 0.3$, $p(X = 2) = 0.5$, $p(X = 3) = 0.2$
 - $E[X] = 0.3 \times 1 + 0.5 \times 2 + 0.2 \times 3 = 1.9$



Variance and Standard Deviation

- ▶ The **variance** gives a measure of how much the **values** of a random variable **X** vary as we sample it from its **probability distribution p(X)**.

$$\text{Var}(X) = E[(X - E[X])^2]$$

$$\text{Var}(X) = \sum_x p(x)(x - E[X])^2$$



Variance and Standard Deviation

- ▶ The **variance** gives a measure of how much the **values** of a random variable **X** vary as we sample it from its **probability distribution p(X)**.

$$\text{Var}(X) = E[(X - E[X])^2]$$

$$\text{Var}(X) = \sum_x p(x)(x - E[X])^2$$

- ▶ E.g., If $X : \{1, 2, 3\}$, and $p(X = 1) = 0.3$, $p(X = 2) = 0.5$, $p(X = 3) = 0.2$

Variance and Standard Deviation

- ▶ The **variance** gives a measure of how much the **values** of a random variable **X** vary as we sample it from its **probability distribution p(X)**.

$$\text{Var}(X) = E[(X - E[X])^2]$$

$$\text{Var}(X) = \sum_x p(x)(x - E[X])^2$$

- ▶ E.g., If $X : \{1, 2, 3\}$, and $p(X = 1) = 0.3$, $p(X = 2) = 0.5$, $p(X = 3) = 0.2$
 - $E[X] = 0.3 \times 1 + 0.5 \times 2 + 0.2 \times 3 = 1.9$
 - $\text{Var}(X) = 0.3(1 - 1.9)^2 + 0.5(2 - 1.9)^2 + 0.2(3 - 1.9)^2 = 0.49$

Variance and Standard Deviation

- ▶ The **variance** gives a measure of how much the **values** of a random variable **X** vary as we sample it from its **probability distribution p(X)**.

$$\text{Var}(X) = E[(X - E[X])^2]$$

$$\text{Var}(X) = \sum_x p(x)(x - E[X])^2$$

- ▶ E.g., If $X : \{1, 2, 3\}$, and $p(X = 1) = 0.3$, $p(X = 2) = 0.5$, $p(X = 3) = 0.2$
 - $E[X] = 0.3 \times 1 + 0.5 \times 2 + 0.2 \times 3 = 1.9$
 - $\text{Var}(X) = 0.3(1 - 1.9)^2 + 0.5(2 - 1.9)^2 + 0.2(3 - 1.9)^2 = 0.49$
- ▶ The **standard deviation**, shown by σ , is the **square root** of the variance.

Covariance (1/2)

- ▶ The **covariance** gives some sense of **how much two values are linearly related** to each other.

$$\text{Cov}(X, Y) = E[(X - E[X])(Y - E[Y])]$$

$$\text{Cov}(X, Y) = \sum_{(x,y)} p(x, y)(x - E[X])(y - E[Y])$$

Covariance (2/2)

			Y		
	p(X, Y)	1	2	3	p(X)
	1	1/4	1/4	0	1/2
X	2	0	1/4	1/4	1/2
	p(Y)	1/4	1/2	1/4	1

Covariance (2/2)

			Y		
	p(X, Y)	1	2	3	p(X)
	1	1/4	1/4	0	1/2
X	2	0	1/4	1/4	1/2
	p(Y)	1/4	1/2	1/4	1

$$E[X] = \frac{1}{2} \times 1 + \frac{1}{2} \times 2 = \frac{3}{2} \quad E[Y] = \frac{1}{4} \times 1 + \frac{1}{2} \times 2 + \frac{1}{4} \times 3 = 2$$

Covariance (2/2)

			Y		
	p(X, Y)	1	2	3	p(X)
	1	1/4	1/4	0	1/2
X	2	0	1/4	1/4	1/2
	p(Y)	1/4	1/2	1/4	1

$$E[X] = \frac{1}{2} \times 1 + \frac{1}{2} \times 2 = \frac{3}{2} \quad E[Y] = \frac{1}{4} \times 1 + \frac{1}{2} \times 2 + \frac{1}{4} \times 3 = 2$$

$$\text{Cov}(X, Y) = \sum_{(x,y)} p(x,y)(x - E[X])(y - E[Y])$$

Covariance (2/2)

			Y		
	p(X, Y)	1	2	3	p(X)
	1	1/4	1/4	0	1/2
X	2	0	1/4	1/4	1/2
	p(Y)	1/4	1/2	1/4	1

$$E[X] = \frac{1}{2} \times 1 + \frac{1}{2} \times 2 = \frac{3}{2} \quad E[Y] = \frac{1}{4} \times 1 + \frac{1}{2} \times 2 + \frac{1}{4} \times 3 = 2$$

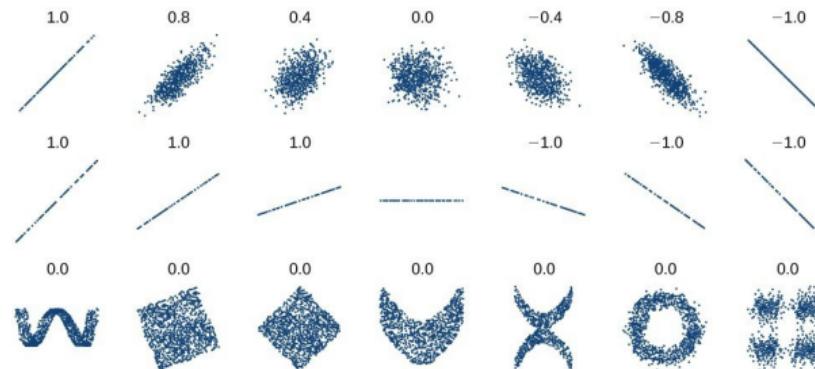
$$\text{Cov}(X, Y) = \sum_{(x,y)} p(x,y)(x - E[X])(y - E[Y])$$

$$\begin{aligned}
&= \frac{1}{4}\left(1 - \frac{3}{2}\right)(1 - 2) + \frac{1}{4}\left(1 - \frac{3}{2}\right)(2 - 2) + 0\left(1 - \frac{3}{2}\right)(3 - 2) \\
&+ 0\left(2 - \frac{3}{2}\right)(1 - 2) + \frac{1}{4}\left(2 - \frac{3}{2}\right)(2 - 2) + \frac{1}{4}\left(2 - \frac{3}{2}\right)(3 - 2) = \frac{1}{4}
\end{aligned}$$

Correlation Coefficient

- The **Correlation coefficient** is a quantity that measures the **strength** of the **association** (or dependence) between two random variables, e.g., **X** and **Y**.

$$\rho(X, Y) = \frac{\text{Cov}(X, Y)}{\sigma(X)\sigma(Y)}$$





Probability and Likelihood (1/2)

- ▶ Let $X : \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ be a discrete random variable drawn independently from a distribution probability p depending on a parameter θ .



Probability and Likelihood (1/2)

- ▶ Let $X : \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ be a **discrete random variable** drawn independently from a **distribution probability p** depending on a **parameter θ** .
 - For six tosses of a coin, $X : \{h, t, t, t, h, t\}$, **h**: head, and **t**: tail.
 - Suppose you have a **coin** with probability θ to land heads and $(1 - \theta)$ to land tails.



Probability and Likelihood (1/2)

- ▶ Let $X : \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ be a **discrete random variable** drawn independently from a **distribution probability p** depending on a **parameter θ** .
 - For six tosses of a coin, $X : \{h, t, t, t, h, t\}$, **h**: head, and **t**: tail.
 - Suppose you have a **coin** with probability θ to land heads and $(1 - \theta)$ to land tails.
- ▶ $p(X | \theta = \frac{2}{3})$ is the **probability** of X given $\theta = \frac{2}{3}$.



Probability and Likelihood (1/2)

- ▶ Let $X : \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ be a **discrete random variable** drawn independently from a **distribution probability p** depending on a **parameter θ** .
 - For six tosses of a coin, $X : \{h, t, t, t, h, t\}$, h : head, and t : tail.
 - Suppose you have a **coin** with probability θ to land heads and $(1 - \theta)$ to land tails.
- ▶ $p(X | \theta = \frac{2}{3})$ is the **probability** of X given $\theta = \frac{2}{3}$.
- ▶ $p(X = h | \theta)$ is the **likelihood** of θ given $X = h$.



Probability and Likelihood (1/2)

- ▶ Let $X : \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ be a **discrete random variable** drawn independently from a **distribution probability p** depending on a **parameter θ** .
 - For six tosses of a coin, $X : \{h, t, t, t, h, t\}$, **h**: head, and **t**: tail.
 - Suppose you have a **coin** with probability θ to land heads and $(1 - \theta)$ to land tails.
- ▶ $p(X | \theta = \frac{2}{3})$ is the **probability** of X given $\theta = \frac{2}{3}$.
- ▶ $p(X = h | \theta)$ is the **likelihood** of θ given $X = h$.
- ▶ **Likelihood (L)**: a function of the **parameters (θ)** of a probability model, given **specific observed data**, e.g., $X = h$.

$$L(\theta | X) = p(X | \theta)$$



Probability and Likelihood (2/2)

- ▶ The **likelihood** differs from that of a **probability**.
- ▶ A **probability** $p(X | \theta)$ refers to the occurrence of **future events**.
- ▶ A **likelihood** $L(\theta | X)$ refers to **past events** with known outcomes.



Maximum Likelihood Estimator

- If samples in \mathbf{X} are **independent** we have:

$$\begin{aligned} L(\theta \mid \mathbf{X}) &= p(\mathbf{X} \mid \theta) = p(x^{(1)}, x^{(2)}, \dots, x^{(m)} \mid \theta) \\ &= p(x^{(1)} \mid \theta)p(x^{(2)} \mid \theta) \cdots p(x^{(m)} \mid \theta) = \prod_{i=1}^m p(x^{(i)} \mid \theta) \end{aligned}$$

Maximum Likelihood Estimator

- If samples in X are **independent** we have:

$$\begin{aligned} L(\theta \mid X) &= p(X \mid \theta) = p(x^{(1)}, x^{(2)}, \dots, x^{(m)} \mid \theta) \\ &= p(x^{(1)} \mid \theta)p(x^{(2)} \mid \theta) \cdots p(x^{(m)} \mid \theta) = \prod_{i=1}^m p(x^{(i)} \mid \theta) \end{aligned}$$

- The **maximum likelihood estimator (MLE)**: what is the **most likely value** of θ given the training set?

$$\hat{\theta}_{\text{MLE}} = \arg \max_{\theta} L(\theta \mid X) = \arg \max_{\theta} \prod_{i=1}^m p(x^{(i)} \mid \theta)$$



Maximum Likelihood Estimator - Example

- ▶ Six tosses of a coin, with the following model:
 - Possible outcomes: **h** with probability of θ , and **t** with probability $(1 - \theta)$.
 - Results of coin tosses are **independent of one another**.
- ▶ Data: **X** : {**h, t, t, t, h, t**}

Maximum Likelihood Estimator - Example

- ▶ Six tosses of a coin, with the following model:
 - Possible outcomes: **h** with probability of θ , and **t** with probability $(1 - \theta)$.
 - Results of coin tosses are independent of one another.
- ▶ Data: $X : \{h, t, t, t, h, t\}$
- ▶ The likelihood is

$$\begin{aligned}L(\theta | X) &= p(X | \theta) \\&= p(X = h | \theta)p(X = t | \theta)p(X = t | \theta)p(X = t | \theta)p(X = h | \theta)p(X = t | \theta) \\&= \theta(1 - \theta)(1 - \theta)(1 - \theta)\theta(1 - \theta) \\&= \theta^2(1 - \theta)^4\end{aligned}$$



Maximum Likelihood Estimator - Example

- ▶ Six tosses of a coin, with the following model:
 - Possible outcomes: **h** with probability of θ , and **t** with probability $(1 - \theta)$.
 - Results of coin tosses are independent of one another.
- ▶ Data: $X : \{h, t, t, t, h, t\}$
- ▶ The likelihood is

$$\begin{aligned}L(\theta | X) &= p(X | \theta) \\&= p(X = h | \theta)p(X = t | \theta)p(X = t | \theta)p(X = t | \theta)p(X = h | \theta)p(X = t | \theta) \\&= \theta(1 - \theta)(1 - \theta)(1 - \theta)\theta(1 - \theta) \\&= \theta^2(1 - \theta)^4\end{aligned}$$

- ▶ $\hat{\theta}$ is the value of θ that maximizes the likelihood:

$$\hat{\theta}_{MLE} = \arg \max_{\theta} L(\theta | X) = \frac{2}{2 + 4}$$



Log-Likelihood

- ▶ The MLE product is prone to numerical underflow.

$$\hat{\theta}_{\text{MLE}} = \arg \max_{\theta} L(\theta \mid X) = \arg \max_{\theta} \prod_{i=1}^m p(x^{(i)} \mid \theta)$$



Log-Likelihood

- The MLE product is prone to numerical underflow.

$$\hat{\theta}_{\text{MLE}} = \arg \max_{\theta} L(\theta | X) = \arg \max_{\theta} \prod_{i=1}^m p(x^{(i)} | \theta)$$

- To overcome this problem we can use the logarithm of the likelihood.
 - It does not change its arg max, but transforms a product into a sum.

$$\hat{\theta}_{\text{MLE}} = \arg \max_{\theta} \sum_{i=1}^m \log p(x^{(i)} | \theta)$$



Negative Log-Likelihood

- ▶ Likelihood: $L(\theta \mid X) = \prod_{i=1}^m p(x^{(i)} \mid \theta)$



Negative Log-Likelihood

- ▶ Likelihood: $L(\theta | X) = \prod_{i=1}^m p(x^{(i)} | \theta)$
- ▶ Log-Likelihood: $\log L(\theta | X) = \log \prod_{i=1}^m p(x^{(i)} | \theta) = \sum_{i=1}^m \log p(x^{(i)} | \theta)$



Negative Log-Likelihood

- ▶ Likelihood: $L(\theta | X) = \prod_{i=1}^m p(x^{(i)} | \theta)$
- ▶ Log-Likelihood: $\log L(\theta | X) = \log \prod_{i=1}^m p(x^{(i)} | \theta) = \sum_{i=1}^m \log p(x^{(i)} | \theta)$
- ▶ Negative Log-Likelihood: $-\log L(\theta | X) = -\sum_{i=1}^m \log p(x^{(i)} | \theta)$

Negative Log-Likelihood

- ▶ Likelihood: $L(\theta | X) = \prod_{i=1}^m p(x^{(i)} | \theta)$
- ▶ Log-Likelihood: $\log L(\theta | X) = \log \prod_{i=1}^m p(x^{(i)} | \theta) = \sum_{i=1}^m \log p(x^{(i)} | \theta)$
- ▶ Negative Log-Likelihood: $-\log L(\theta | X) = -\sum_{i=1}^m \log p(x^{(i)} | \theta)$
- ▶ Negative log-likelihood is also called the **cross-entropy**



Cross-Entropy

- ▶ Cross-entropy: quantify the difference (error) between two probability distributions.
- ▶ How close is the predicted distribution to the true distribution?

$$H(p, q) = - \sum_x p(x) \log(q(x))$$

- ▶ Where p is the true distribution, and q the predicted distribution.



Cross-Entropy - Example

- ▶ Six tosses of a coin: $X : \{h, t, t, t, h, t\}$
- ▶ The true distribution p : $p(h) = \frac{2}{6}$ and $p(t) = \frac{4}{6}$
- ▶ The predicted distribution q : h with probability of θ , and t with probability $(1 - \theta)$.



Cross-Entropy - Example

- ▶ Six tosses of a coin: $X : \{h, t, t, t, h, t\}$
- ▶ The **true distribution** p : $p(h) = \frac{2}{6}$ and $p(t) = \frac{4}{6}$
- ▶ The **predicted distribution** q : h with probability of θ , and t with probability $(1 - \theta)$.
- ▶ Cross entropy: $H(p, q) = -\sum_x p(x)\log(q(x))$
 $= -p(h)\log(q(h)) - p(t)\log(q(t)) = -\frac{2}{6}\log(\theta) - \frac{4}{6}\log(1 - \theta)$



Cross-Entropy - Example

- ▶ Six tosses of a coin: $X : \{h, t, t, t, h, t\}$
- ▶ The true distribution p : $p(h) = \frac{2}{6}$ and $p(t) = \frac{4}{6}$
- ▶ The predicted distribution q : h with probability of θ , and t with probability $(1 - \theta)$.
- ▶ Cross entropy: $H(p, q) = -\sum_x p(x)\log(q(x))$
 $= -p(h)\log(q(h)) - p(t)\log(q(t)) = -\frac{2}{6}\log(\theta) - \frac{4}{6}\log(1 - \theta)$
- ▶ Likelihood: $\theta^2(1 - \theta)^4$

Cross-Entropy - Example

- ▶ Six tosses of a coin: $X : \{h, t, t, t, h, t\}$
- ▶ The **true distribution** p : $p(h) = \frac{2}{6}$ and $p(t) = \frac{4}{6}$
- ▶ The **predicted distribution** q : h with probability of θ , and t with probability $(1 - \theta)$.
- ▶ Cross entropy: $H(p, q) = -\sum_x p(x)\log(q(x))$
 $= -p(h)\log(q(h)) - p(t)\log(q(t)) = -\frac{2}{6}\log(\theta) - \frac{4}{6}\log(1 - \theta)$
- ▶ Likelihood: $\theta^2(1 - \theta)^4$
- ▶ Negative log likelihood: $-\log(\theta^2(1 - \theta)^4) = -2\log(\theta) - 4\log(1 - \theta)$



Summary



Summary

- ▶ Logic-based AI, Machine Learning, Deep Learning
- ▶ Deep Learning models
 - Deep Feed Forward
 - Convolutional Neural Network (CNN)
 - Recurrent Neural Network (RNN)
 - Transformer
- ▶ Linear algebra and probability
 - Random variables
 - Probability distribution
 - Likelihood
 - Negative log-likelihood and cross-entropy



References

- ▶ Ian Goodfellow et al., Deep Learning (Ch. 1, 2, 3)



Questions?

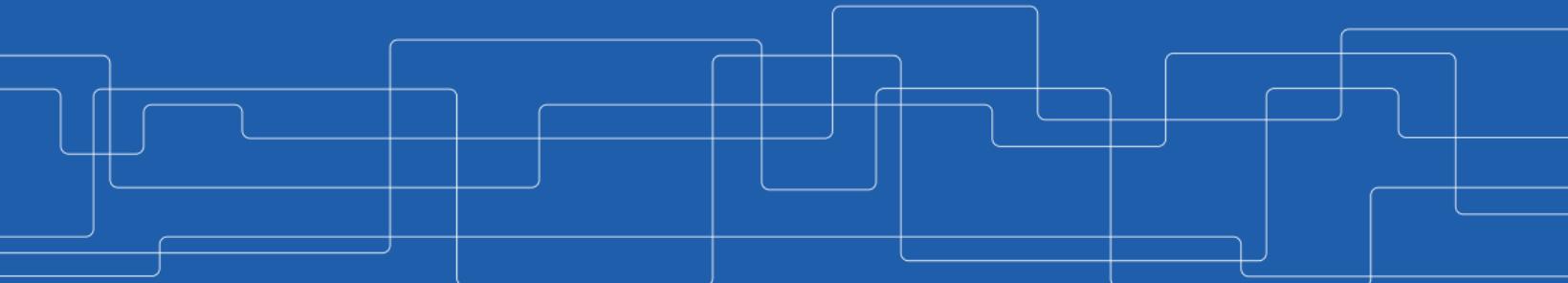
Acknowledgements

Some of the pictures were copied from the book Hands-On Machine Learning with Scikit-Learn and TensorFlow, Aurelien Geron, O'Reilly Media, 2017.



Machine Learning with Spark

Amir H. Payberah
payberah@kth.se
2021-11-04



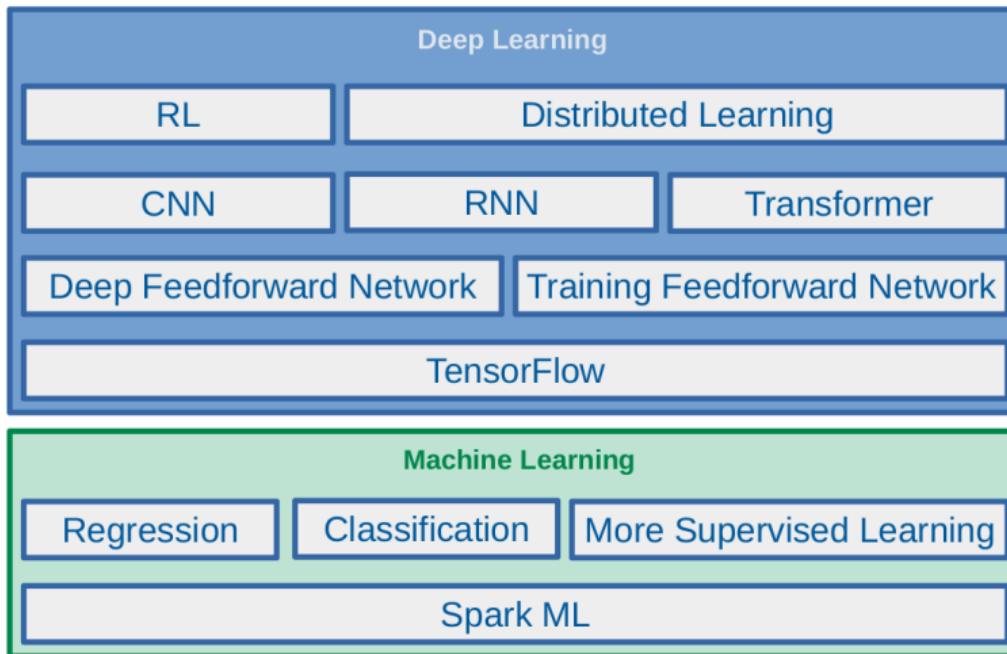


The Course Web Page

<https://id2223kth.github.io>
<https://tinyurl.com/6s5jy46a>

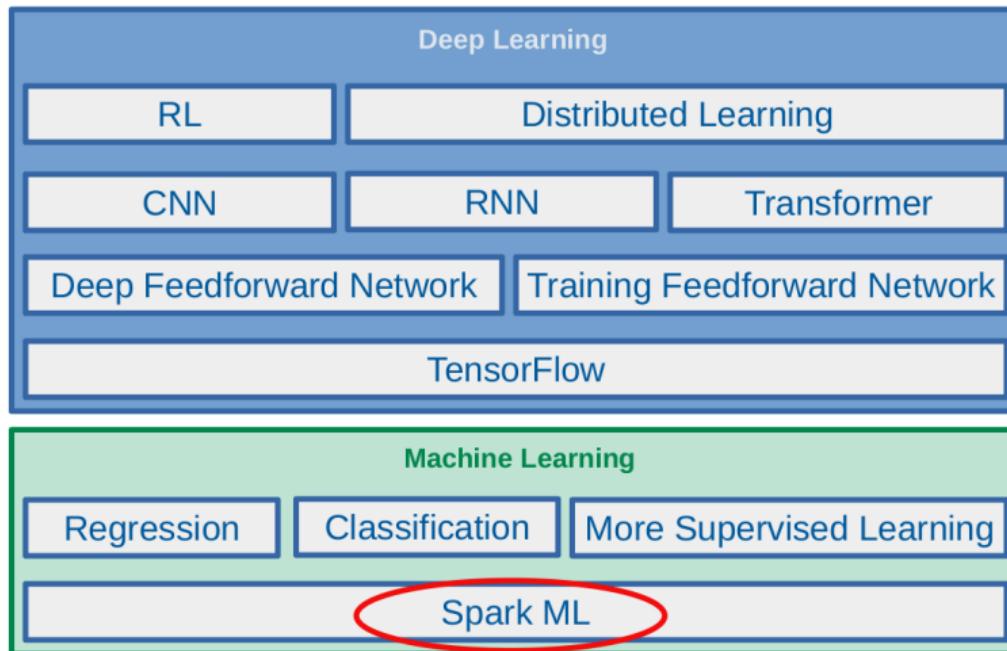


Where Are We?





Where Are We?



Big Data





Problem

- ▶ Traditional platforms **fail** to show the expected performance.
- ▶ Need **new systems** to **store and process** large-scale data

Scale Up vs. Scale Out

- ▶ Scale **up** or scale **vertically**
- ▶ Scale **out** or scale **horizontally**



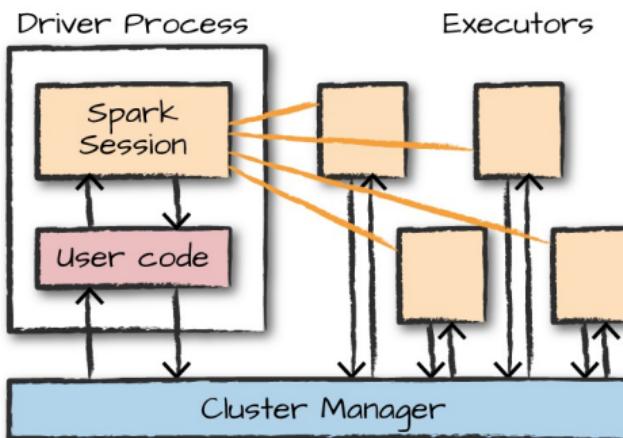


Spark

Spark Execution Model (1/3)

- ▶ Spark applications consist of

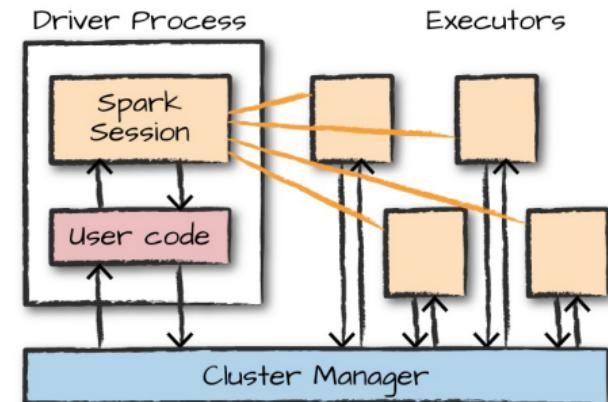
- A **driver** process
- A **set of executor** processes



[M. Zaharia et al., Spark: The Definitive Guide, O'Reilly Media, 2018]

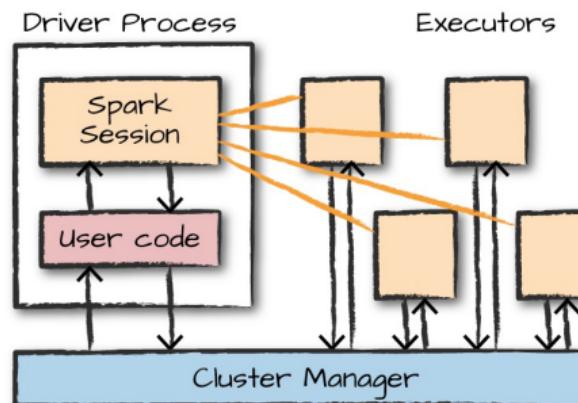
Spark Execution Model (2/3)

- ▶ The **driver process** is the **heart** of a **Spark application**
- ▶ Sits on a **node** in the cluster
- ▶ Runs the **main()** function



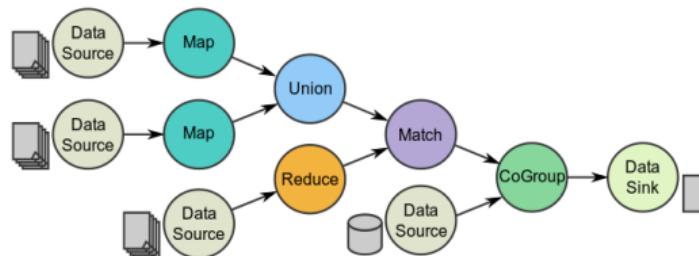
Spark Execution Model (3/3)

- ▶ Executors execute codes assigned to them by the driver.



Spark Programming Model

- ▶ Job description based on [directed acyclic graphs \(DAG\)](#).
- ▶ There are two types of RDD operators: [transformations](#) and [actions](#).



Resilient Distributed Datasets (RDD) (1/2)

- ▶ A **distributed memory** abstraction.
- ▶ **Immutable collections** of **objects** spread across a cluster.
 - Like a `LinkedList <MyObjects>`



Resilient Distributed Datasets (RDD) (2/2)

- ▶ An **RDD** is divided into a number of **partitions**, which are **atomic** pieces of information.
- ▶ Partitions of an RDD can be stored on different **nodes** of a cluster.





Creating RDDs

- ▶ Turn a collection into an RDD.

```
val a = sc.parallelize(Array(1, 2, 3))
```



Creating RDDs

- ▶ Turn a collection into an RDD.

```
val a = sc.parallelize(Array(1, 2, 3))
```

- ▶ Load text file from local FS, HDFS, or S3.

```
val a = sc.textFile("file.txt")
val b = sc.textFile("directory/*.txt")
val c = sc.textFile("hdfs://namenode:9000/path/file")
```

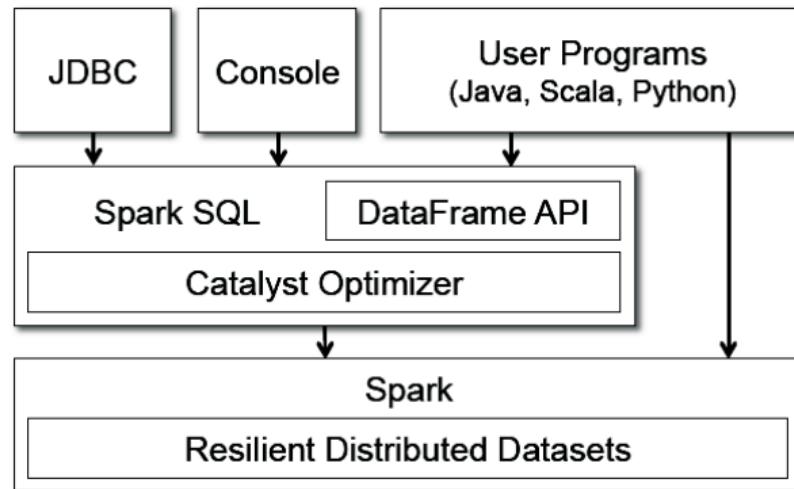


RDD Operations

- ▶ **Transformations:** lazy operators that create new RDDs.
- ▶ **Actions:** launch a computation and return a value to the program or write data to the external storage.



Spark and Spark SQL



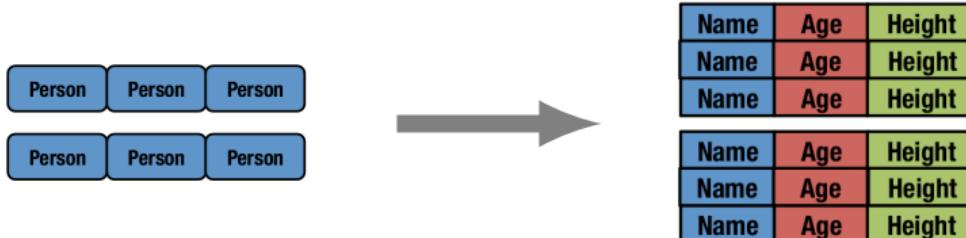


DataFrame

- ▶ A **DataFrame** is a **distributed collection of rows** with a **homogeneous schema**.
- ▶ It is equivalent to a **table** in a relational database.
- ▶ It can also be manipulated in similar ways to **RDDs**.

Adding Schema to RDDs

- ▶ Spark + RDD: functional transformations on partitioned collections of opaque objects.
- ▶ SQL + DataFrame: declarative transformations on partitioned collections of tuples.





Creating a DataFrame - From an RDD

- ▶ You can use `toDF` to convert an RDD to DataFrame.

```
val tupleRDD = sc.parallelize(Array(("seif", 65, 0), ("amir", 40, 1)))
val tupleDF = tupleRDD.toDF("name", "age", "id")
```



Creating a DataFrame - From an RDD

- ▶ You can use `toDF` to convert an RDD to DataFrame.

```
val tupleRDD = sc.parallelize(Array(("seif", 65, 0), ("amir", 40, 1)))
val tupleDF = tupleRDD.toDF("name", "age", "id")
```

- ▶ If RDD contains `case` class instances, Spark infers the attributes from it.

```
case class Person(name: String, age: Int, id: Int)

val peopleRDD = sc.parallelize(Array(Person("seif", 65, 0), Person("amir", 40, 1)))
val peopleDF = peopleRDD.toDF
```



Creating a DataFrame - From Data Source

► Data sources supported by Spark.

- CSV, JSON, Parquet, ORC, JDBC/ODBC connections, Plain-text files
- Cassandra, HBase, MongoDB, AWS Redshift, XML, etc.

```
val peopleJson = spark.read.format("json").load("people.json")

val peopleCsv = spark.read.format("csv")
  .option("sep", ";")
  .option("inferSchema", "true")
  .option("header", "true")
  .load("people.csv")
```



Column

- ▶ Different ways to refer to a column.

```
val people = spark.read.format("json").load("people.json")

people.col("name")

col("name")

column("name")

'name

$"name"

expr("name")
```



DataFrame Transformations (1/6)

- ▶ `select` allows to do the DataFrame equivalent of SQL queries on a table of data.

```
people.select("name", "age", "id").show(2)
people.select(col("name"), expr("age + 3")).show()
people.select(expr("name AS username")).show(2)
```



DataFrame Transformations (1/6)

- ▶ `select` allows to do the DataFrame equivalent of SQL queries on a table of data.

```
people.select("name", "age", "id").show(2)
people.select(col("name"), expr("age + 3")).show()
people.select(expr("name AS username")).show(2)
```

- ▶ `filter` and `where` both filter rows.

```
people.filter(col("age") < 20).show()

people.where("age < 20").show()
```



DataFrame Transformations (1/6)

- ▶ `select` allows to do the DataFrame equivalent of SQL queries on a table of data.

```
people.select("name", "age", "id").show(2)
people.select(col("name"), expr("age + 3")).show()
people.select(expr("name AS username")).show(2)
```

- ▶ `filter` and `where` both filter rows.

```
people.filter(col("age") < 20).show()

people.where("age < 20").show()
```

- ▶ `distinct` can be used to extract unique rows.

```
people.select("name").distinct().count()
```



DataFrame Transformations (2/6)

- ▶ `withColumn` adds a new column to a DataFrame.

```
people.withColumn("teenager", expr("age < 20")).show()
```



DataFrame Transformations (2/6)

- ▶ `withColumn` adds a new column to a DataFrame.

```
people.withColumn("teenager", expr("age < 20")).show()
```

- ▶ `withColumnRenamed` renames a column.

```
people.withColumnRenamed("name", "username").columns
```



DataFrame Transformations (2/6)

- ▶ `withColumn` adds a new column to a DataFrame.

```
people.withColumn("teenager", expr("age < 20")).show()
```

- ▶ `withColumnRenamed` renames a column.

```
people.withColumnRenamed("name", "username").columns
```

- ▶ `drop` removes a column.

```
people.drop("name").columns
```



DataFrame Transformations (3/6)

- ▶ `count` returns the total number of values.

```
people.select(count("age")).show()
```



DataFrame Transformations (3/6)

- ▶ `count` returns the total number of values.

```
people.select(count("age")).show()
```

- ▶ `countDistinct` returns the number of unique groups.

```
people.select(countDistinct("name")).show()
```



DataFrame Transformations (3/6)

- ▶ `count` returns the total number of values.

```
people.select(count("age")).show()
```

- ▶ `countDistinct` returns the number of unique groups.

```
people.select(countDistinct("name")).show()
```

- ▶ `first` and `last` return the first and last value of a DataFrame.

```
people.select(first("name"), last("age")).show()
```



DataFrame Transformations (4/6)

- ▶ `min` and `max` extract the minimum and maximum values from a DataFrame.

```
people.select(min("name"), max("age"), max("id")).show()
```



DataFrame Transformations (4/6)

- ▶ `min` and `max` extract the minimum and maximum values from a DataFrame.

```
people.select(min("name"), max("age"), max("id")).show()
```

- ▶ `sum` adds all the values in a column.

```
people.select(sum("age")).show()
```



DataFrame Transformations (4/6)

- ▶ `min` and `max` extract the minimum and maximum values from a DataFrame.

```
people.select(min("name"), max("age"), max("id")).show()
```

- ▶ `sum` adds all the values in a column.

```
people.select(sum("age")).show()
```

- ▶ `avg` calculates the average.

```
people.select(avg("age")).show()
```



DataFrame Transformations (5/6)

- ▶ `groupBy` and `agg` together perform aggregations on `groups`.

```
people.groupBy("name").agg(count("age")).show()
```



DataFrame Transformations (5/6)

- ▶ `groupBy` and `agg` together perform aggregations on **groups**.

```
people.groupBy("name").agg(count("age")).show()
```

- ▶ `join` performs the join operation between **two tables**.

```
val t1 = spark.createDataFrame(Seq((0, "a", 0), (1, "b", 1), (2, "c", 1)))
    .toDF("num", "name", "id")
val t2 = spark.createDataFrame(Seq((0, "x"), (1, "y"), (2, "z")))
    .toDF("id", "group")

val joinExpression = t1.col("id") === t2.col("id")
var joinType = "inner"

t1.join(t2, joinExpression, joinType).show()
```



DataFrame Transformations (6/6)

- ▶ You can use `udf` to define new column-based functions.

```
import org.apache.spark.sql.functions.udf

val df = spark.createDataFrame(Seq((0, "hello"), (1, "world"))).toDF("id", "text")

val upper: String => String = _.toUpperCase
val upperUDF = spark.udf.register("upper", upper)

df.withColumn("upper", upperUDF(col("text"))).show
```



DataFrame Actions

- ▶ Like RDDs, DataFrames also have their own set of actions.
- ▶ `collect`: returns an **array** that contains all the **rows** in this DataFrame.
- ▶ `count`: returns the **number of rows** in this DataFrame.
- ▶ `first` and `head`: returns the **first row** of the DataFrame.
- ▶ `show`: displays the **top 20 rows** of the DataFrame in a tabular form.
- ▶ `take`: returns the **first n rows** of the DataFrame.



Machine Learning



Machine Learning with Spark

- ▶ Spark provides support for **statistics** and **machine learning**.
 - Supervised learning
 - Unsupervised engines
 - Deep learning



Supervised Learning

- ▶ Using **labeled historical data** and **training a model** to **predict** the values of those labels based on **various features** of the data points.
- ▶ **Classification** (**categorical** values)
 - E.g., predicting disease, classifying images, ...
- ▶ **Regression** (**continuous** values)
 - E.g., predicting sales, predicting height, ...

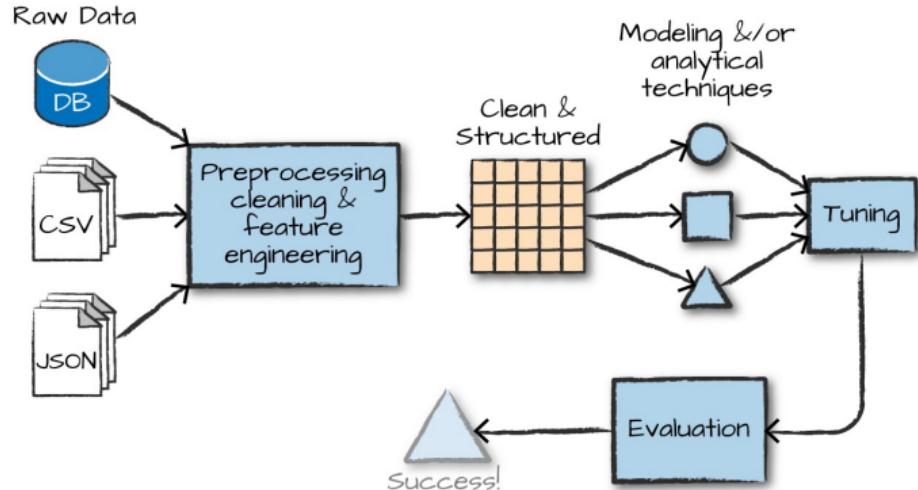


Unsupervised Learning

- ▶ No label to predict.
- ▶ Trying to find patterns or discover the underlying structure in a given set of data.
 - Clustering, anomaly detection, ...

The Advanced Analytic Process

- ▶ Data collection
- ▶ Data cleaning
- ▶ Feature engineering
- ▶ Training models
- ▶ Model tuning and evaluation





What is MLlib? (1/2)

- ▶ **MLlib** is a package built on **Spark**.
- ▶ It provides **interfaces** for:
 - **Gathering** and **cleaning** data
 - **Feature engineering** and feature selection
 - **Training** and **tuning** large-scale **supervised** and **unsupervised** machine learning models
 - Using those models in **production**



What is MLlib? (2/2)

- ▶ MLlib consists of **two packages**.



What is MLlib? (2/2)

- ▶ MLlib consists of **two packages**.
- ▶ **org.apache.spark.mllib**
 - Uses **RDDs**
 - It is in **maintenance mode** (only receives **bug fixes**, not new features)

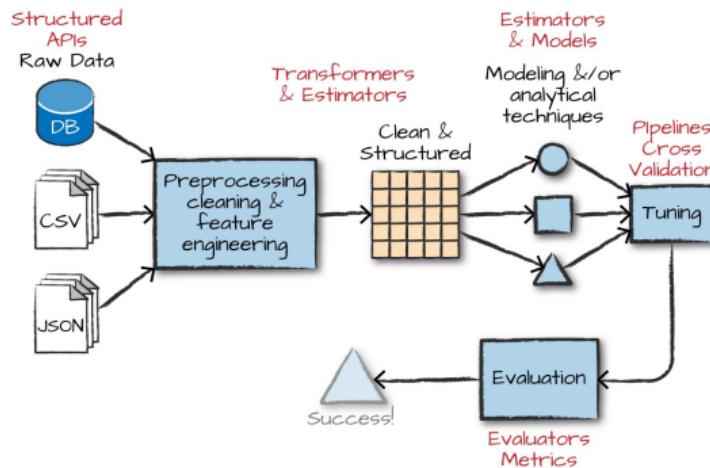


What is MLlib? (2/2)

- ▶ MLlib consists of two packages.
- ▶ `org.apache.spark.mllib`
 - Uses RDDs
 - It is in maintenance mode (only receives bug fixes, not new features)
- ▶ `org.apache.spark.ml`
 - Uses DataFrames
 - Offers a high-level interface for building machine learning pipelines

High-Level MLlib Concepts

- ▶ ML pipelines (`spark.ml`) provide a uniform set of high-level APIs built on top of `DataFrames` to create machine learning pipelines.





Pipeline

- ▶ Pipeline is a sequence of algorithms to **process** and **learn** from data.



Pipeline

- ▶ Pipeline is a sequence of algorithms to process and learn from data.
- ▶ E.g., a text document processing workflow might include several stages:



Pipeline

- ▶ Pipeline is a sequence of algorithms to process and learn from data.
- ▶ E.g., a text document processing workflow might include several stages:
 - Split each document's text into words.



Pipeline

- ▶ Pipeline is a sequence of algorithms to process and learn from data.
- ▶ E.g., a text document processing workflow might include several stages:
 - Split each document's text into words.
 - Convert each document's words into a numerical feature vector.



Pipeline

- ▶ Pipeline is a sequence of algorithms to process and learn from data.
- ▶ E.g., a text document processing workflow might include several stages:
 - Split each document's text into words.
 - Convert each document's words into a numerical feature vector.
 - Learn a prediction model using the feature vectors and labels.



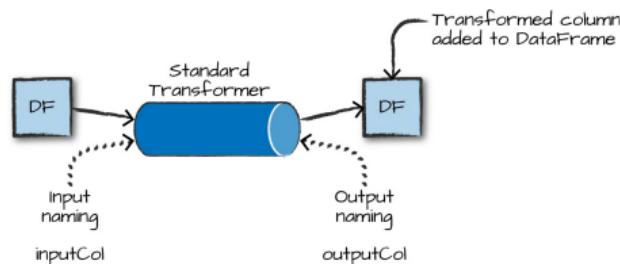
Pipeline

- ▶ Pipeline is a sequence of algorithms to process and learn from data.
- ▶ E.g., a text document processing workflow might include several stages:
 - Split each document's text into words.
 - Convert each document's words into a numerical feature vector.
 - Learn a prediction model using the feature vectors and labels.
- ▶ Main pipeline components: transformers and estimators

Transformers

- ▶ **Transformers** take a **DataFrame** as input and produce a new **DataFrame** as output.

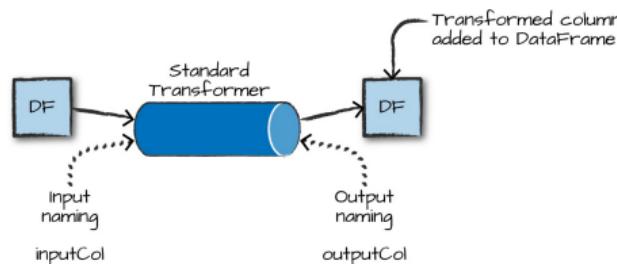
```
// transformer: DataFrame =[transform]> DataFrame  
  
transform(dataset: DataFrame): DataFrame
```



Transformers

- ▶ **Transformers** take a `DataFrame` as input and produce a new `DataFrame` as output.
- ▶ The class `Transformer` implements a method `transform()` that converts **one** `DataFrame` into another.

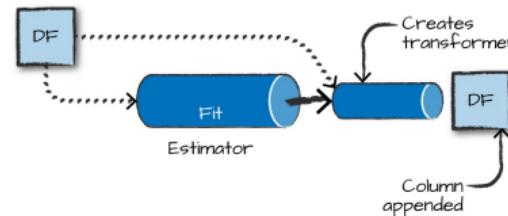
```
// transformer: DataFrame =[transform]=> DataFrame  
  
transform(dataset: DataFrame): DataFrame
```



Estimators

- ▶ Estimator is an abstraction of a learning algorithm that fits a **model** on a **dataset**.

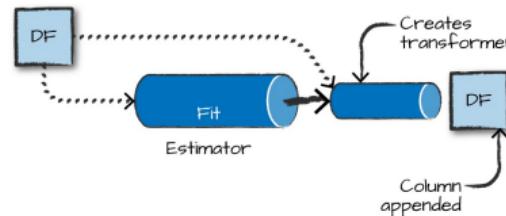
```
// estimator: DataFrame =[fit]=> Model  
fit(dataset: DataFrame): M
```



Estimators

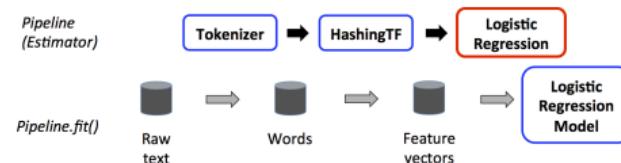
- ▶ Estimator is an abstraction of a learning algorithm that fits a model on a dataset.
- ▶ The class Estimator implements a method `fit()`, which accepts a DataFrame and produces a Model (Transformer).

```
// estimator: DataFrame =[fit]=> Model  
  
fit(dataset: DataFrame): M
```



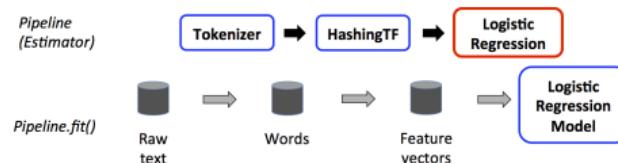
How Does Pipeline Work? (1/3)

- ▶ A pipeline is a **sequence** of stages.
- ▶ Stages of a pipeline **run in order**.



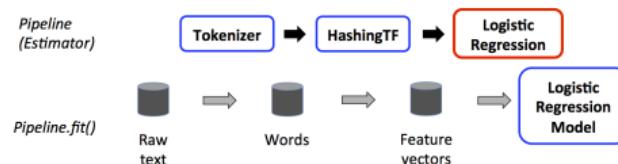
How Does Pipeline Work? (1/3)

- ▶ A pipeline is a **sequence** of stages.
- ▶ **Stages** of a pipeline **run in order**.
- ▶ The input **DataFrame** is transformed as it passes through each stage.
 - Each stage is either a **Transformer** or an **Estimator**.



How Does Pipeline Work? (1/3)

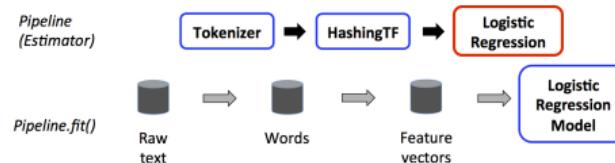
- ▶ A pipeline is a **sequence** of stages.
- ▶ **Stages** of a pipeline **run in order**.
- ▶ The input **DataFrame** is transformed as it passes through each stage.
 - Each stage is either a **Transformer** or an **Estimator**.
- ▶ E.g., a Pipeline with **three stages**: **Tokenizer** and **HashingTF** are **Transformers**, and **LogisticRegression** is an **Estimator**.





How Does Pipeline Work? (2/3)

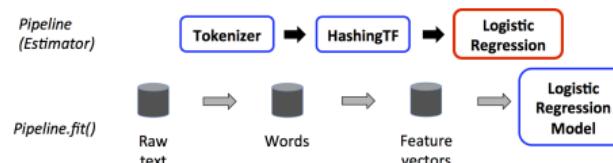
- ▶ `Pipeline.fit()`: is called on the original DataFrame
 - DataFrame with raw text documents and labels





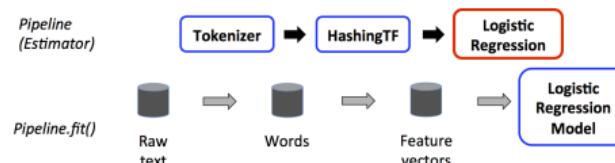
How Does Pipeline Work? (2/3)

- ▶ `Pipeline.fit()`: is called on the original DataFrame
 - DataFrame with raw text documents and labels
- ▶ `Tokenizer.transform()`: splits the raw text documents into words
 - Adds a new column with words to the DataFrame



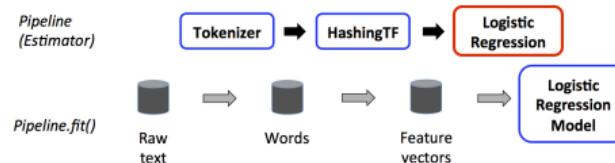
How Does Pipeline Work? (2/3)

- ▶ `Pipeline.fit()`: is called on the original DataFrame
 - DataFrame with raw text documents and labels
- ▶ `Tokenizer.transform()`: splits the raw text documents into words
 - Adds a new column with words to the DataFrame
- ▶ `HashingTF.transform()`: converts the words column into feature vectors
 - Adds new column with those vectors to the DataFrame



How Does Pipeline Work? (2/3)

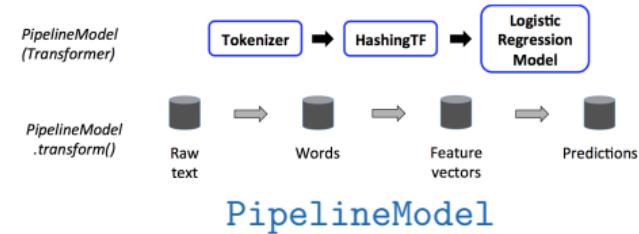
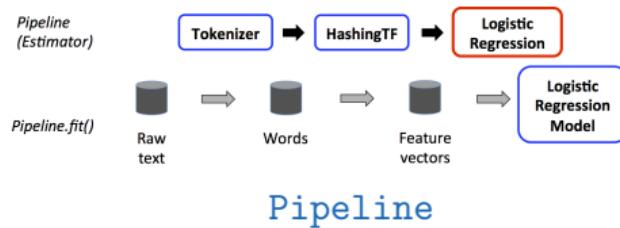
- ▶ `Pipeline.fit()`: is called on the original DataFrame
 - DataFrame with raw text documents and labels
- ▶ `Tokenizer.transform()`: splits the raw text documents into words
 - Adds a new column with words to the DataFrame
- ▶ `HashingTF.transform()`: converts the words column into feature vectors
 - Adds new column with those vectors to the DataFrame
- ▶ `LogisticRegression.fit()`: produces a model (`LogisticRegressionModel`).





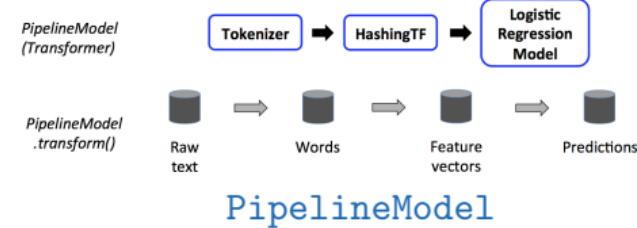
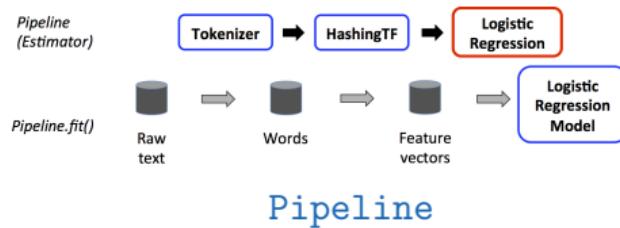
How Does Pipeline Work? (3/3)

- ▶ A Pipeline is an Estimator (`DataFrame = [fit] => Model`).



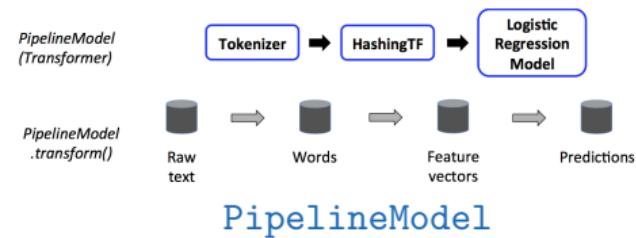
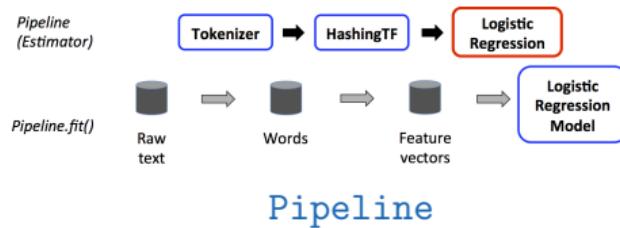
How Does Pipeline Work? (3/3)

- ▶ A **Pipeline** is an **Estimator** (`DataFrame = [fit] => Model`).
- ▶ After a **Pipeline's fit()** runs, it **produces** a **PipelineModel**.



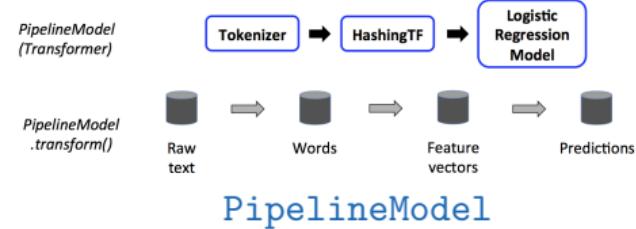
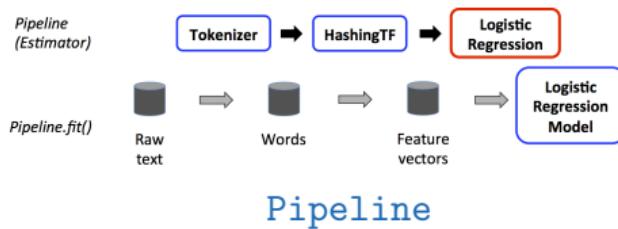
How Does Pipeline Work? (3/3)

- ▶ A Pipeline is an Estimator (`DataFrame =[fit]=> Model`).
- ▶ After a Pipeline's `fit()` runs, it produces a PipelineModel.
- ▶ PipelineModel is a Transformer (`DataFrame =[transform]=> DataFrame`).



How Does Pipeline Work? (3/3)

- ▶ A Pipeline is an Estimator (`DataFrame =[fit]=> Model`).
- ▶ After a Pipeline's `fit()` runs, it produces a PipelineModel.
- ▶ PipelineModel is a Transformer (`DataFrame =[transform]=> DataFrame`).
- ▶ The PipelineModel is used at test time.





Example - Input DataFrame (1/2)

- ▶ Make a DataFrame of the type `Article`.

```
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.linalg.{Vector, Vectors}
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.sql.Row

case class Article(id: Long, topic: String, text: String)

val articles = spark.createDataFrame(Seq(
    Article(0, "sci.math", "Hello, Math!"),
    Article(1, "alt.religion", "Hello, Religion!"),
    Article(2, "sci.physics", "Hello, Physics!"),
    Article(3, "sci.math", "Hello, Math Revised!"),
    Article(4, "sci.math", "Better Math"),
    Article(5, "alt.religion", "TGIF"))).toDF

articles.show
```



Example - Input DataFrame (2/2)

- ▶ Add a new column `label` to the DataFrame.
- ▶ `udf` is a feature of Spark SQL to define **new Column-based functions**.

```
val topic2Label: Boolean => Double = x => if (x) 1 else 0

val toLabel = spark.udf.register("topic2Label", topic2Label)

val labelled = articles.withColumn("label", toLabel($"topic".like("sci%"))).cache

labelled.show
```



Example - Transformers (1/2)

- ▶ Break each sentence into individual terms (words).

```
import org.apache.spark.ml.feature.Tokenizer
import org.apache.spark.ml.feature.RegexTokenizer

val tokenizer = new RegexTokenizer().setInputCol("text").setOutputCol("words")

val tokenized = tokenizer.transform(labelled)

tokenized.show(false)
```



Example - Transformers (2/2)

- ▶ Takes a set of words and converts them into **fixed-length feature vector**.
 - 5000 in our example
- ▶ Uses a **hash function** to map each word into an **index** in the feature vector.
- ▶ Then computes the **term frequencies** based on the mapped indices.

```
import org.apache.spark.ml.feature.HashingTF

val hashingTF = new HashingTF().setInputCol(tokenizer.getOutputCol)
                           .setOutputCol("features")
                           .setNumFeatures(5000)

val hashed = hashingTF.transform(tokenized)

hashed.show(false)
```



Example - Estimator

```
val Array(trainDF, testDF) = hashed.randomSplit(Array(0.8, 0.2))

trainDF.show

testDF.show
```

```
import org.apache.spark.ml.classification.LogisticRegression

val lr = new LogisticRegression().setMaxIter(20).setRegParam(0.01)

val model = lr.fit(trainDF)

val pred = model.transform(testDF).select("topic", "label", "prediction")

pred.show
```



Example - Pipeline

```
val Array(trainDF2, testDF2) = labelled.randomSplit(Array(0.8, 0.2))

trainDF2.show

testDF2.show
```

```
import org.apache.spark.ml.{Pipeline, PipelineModel}

val pipeline = new Pipeline().setStages(Array(tokenizer, hashingTF, lr))

val model2 = pipeline.fit(trainDF2)

val pred = model2.transform(testDF2).select("topic", "label", "prediction")

pred.show
```



Parameters

- ▶ MLlib **Estimators** and **Transformers** use a **uniform API** for specifying parameters.



Parameters

- ▶ MLlib `Estimators` and `Transformers` use a `uniform API` for specifying parameters.
- ▶ `Param`: a named parameter
- ▶ `ParamMap`: a set of `(parameter, value)` pairs



Parameters

- ▶ MLlib **Estimators** and **Transformers** use a **uniform API** for specifying parameters.
- ▶ **Param**: a **named parameter**
- ▶ **ParamMap**: a set of **(parameter, value)** pairs
- ▶ Two ways to **pass parameters** to an algorithm:
 1. Set parameters for an instance, e.g., `lr.setMaxIter(10)`
 2. Pass a **ParamMap** to `fit()` or `transform()`.



Example - ParamMap

```
// set parameters using setter methods.  
val lr = new LogisticRegression()  
  
lr.setMaxIter(10).setRegParam(0.01)
```

```
// specify parameters using a ParamMap  
val lr = new LogisticRegression()  
  
val paramMap = ParamMap(lr.maxIter -> 20)  
  .put(lr.maxIter, 30) // specify one Param  
  .put(lr.regParam -> 0.1, lr.threshold -> 0.55) // specify multiple Params  
  
val model = lr.fit(training, paramMap)
```



Low-Level Data Types - Local Vector

- ▶ Stored on a **single** machine
- ▶ **Dense** and **sparse**
 - **Dense** (1.0, 0.0, 3.0): [1.0, 0.0, 3.0]
 - **Sparse** (1.0, 0.0, 3.0): (3, [0, 2], [1.0, 3.0])

```
import org.apache.spark.mllib.linalg.{Vector, Vectors}

val dv: Vector = Vectors.dense(1.0, 0.0, 3.0)

val sv1: Vector = Vectors.sparse(3, Array(0, 2), Array(1.0, 3.0))
val sv2: Vector = Vectors.sparse(3, Seq((0, 1.0), (2, 3.0)))
```



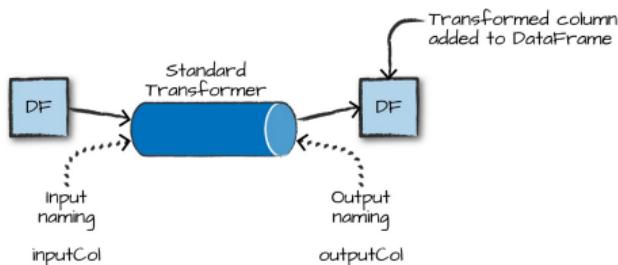
Preprocessing and Feature Engineering



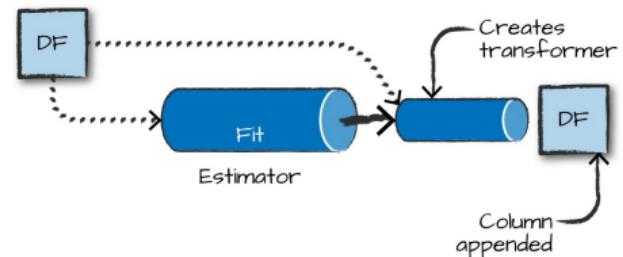
Formatting Models

- ▶ In most of **classification and regression** algorithms, we want to get the **data**.
 - A **column** to represent the **label** (**Double**).
 - A **column** to represent the **features** (**Vector**)

Transformers and Estimators



Transformer



Estimator



Transformer Properties

- ▶ All transformers require you to specify the `input` and `output` columns.
- ▶ We can set these with `setInputCol` and `setOutputCol`.

```
val tokenizer = new RegexTokenizer().setInputCol("text").setOutputCol("words")
```



Vector Assembler

- ▶ Concatenate all your features into one vector.

```
import org.apache.spark.ml.feature.VectorAssembler

case class Nums(val1: Long, val2: Long, val3: Long)

val numsDF = spark.createDataFrame(Seq(Nums(1, 2, 3), Nums(4, 5, 6), Nums(7, 8, 9))).toDF

val va = new VectorAssembler().setInputCols(Array("val1", "val2", "val3"))
                           .setOutputCol("features")

va.transform(numsDF).show()
```



MLlib Transformers

- ▶ Continuous features
- ▶ Categorical features
- ▶ Text data



MLlib Transformers

- ▶ Continuous features
- ▶ Categorical features
- ▶ Text data



Continuous Features - Bucketing

- ▶ Convert **continuous features** into **categorical features**.

```
import org.apache.spark.ml.feature.Bucketizer

val contDF = spark.range(20).selectExpr("cast(id as double)")
val bucketBorders = Array(-1.0, 5.0, 10.0, 15.0, 20.0)

val bucketer = new Bucketizer().setSplits(bucketBorders).setInputCol("id")

bucketer.transform(contDF).show()
```



Continuous Features - Scaling and Normalization

- ▶ To scale and normalize continuous data.

```
import org.apache.spark.ml.feature.VectorAssembler

case class Nums(val1: Long, val2: Long, val3: Long)
val numsDF = spark.createDataFrame(Seq(Nums(1, 2, 3), Nums(4, 5, 6), Nums(7, 8, 9))).toDF
val va = new VectorAssembler().setInputCols(Array("val1", "val2", "val3"))
                           .setOutputCol("features")
val nums = va.transform(numsDF)
```

```
import org.apache.spark.ml.feature.StandardScaler

val scaler = new StandardScaler().setInputCol("features").setOutputCol("scaled")
scaler.fit(nums).transform(nums).show()
```



Continuous Features - Maximum Absolute Scaler

- ▶ Scales the data by dividing each feature by the maximum absolute value in this feature (column).

```
import org.apache.spark.ml.feature.VectorAssembler

case class Nums(val1: Long, val2: Long, val3: Long)
val numsDF = spark.createDataFrame(Seq(Nums(1, 2, 3), Nums(4, 5, 6), Nums(7, 8, 9))).toDF
val va = new VectorAssembler().setInputCols(Array("val1", "val2", "val3"))
                           .setOutputCol("features")
val nums = va.transform(numsDF)
```

```
import org.apache.spark.ml.feature.MaxAbsScaler

val maScaler = new MaxAbsScaler().setInputCol("features").setOutputCol("mas")
maScaler.fit(nums).transform(nums).show()
```



MLlib Transformers

- ▶ Continuous features
- ▶ Categorical features
- ▶ Text data



Categorical Features - String Indexer

- ▶ Maps **strings** to different **numerical IDs**.

```
val simpleDF = spark.read.json("simple-ml.json")
```

```
import org.apache.spark.ml.feature.StringIndexer  
  
val lblIndxr = new StringIndexer().setInputCol("lab").setOutputCol("labelInd")  
val idxRes = lblIndxr.fit(simpleDF).transform(simpleDF)  
  
idxRes.show()
```



Categorical Features - Converting Indexed Values Back to Text

- ▶ Maps back to the original values.

```
import org.apache.spark.ml.feature.IndexToString

val labelReverse = new IndexToString().setInputCol("labelInd").setOutputCol("original")

labelReverse.transform(idxRes).show()
```



Categorical Features - One-Hot Encoding

- ▶ Converts each **distinct value** to a **boolean flag** as a component in a **vector**.

```
val simpleDF = spark.read.json("simple-ml.json")
```

```
import org.apache.spark.ml.feature.OneHotEncoder

val lblIndxr = new StringIndexer().setInputCol("color").setOutputCol("colorInd")
val colorLab = lblIndxr.fit(simpleDF).transform(simpleDF.select("color"))
val ohe = new OneHotEncoder().setInputCol("colorInd").setOutputCol("one-hot")
ohe.transform(colorLab).show()

// Since there are three values, the vector is of length 2 and the mapping is as follows:
// 0 -> 10, (2,[0],[1.0])
// 1 -> 01, (2,[1],[1.0])
// 2 -> 00, (2,[],[])
// (2,[0],[1.0]) means a vector of length 2 with 1.0 at position 0 and 0 elsewhere.
```



MLlib Transformers

- ▶ Continuous features
- ▶ Categorical features
- ▶ Text data



Text Data - Tokenizing Text

- ▶ Converting **free-form text** into a list of **tokens** or individual words.

```
val sales = spark.read.format("csv").option("header", "true").load("sales.csv")
    .where("Description IS NOT NULL")

sales.show(false)
```

```
import org.apache.spark.ml.feature.Tokenizer

val tkn = new Tokenizer().setInputCol("Description").setOutputCol("DescOut")
val tokenized = tkn.transform(sales.select("Description"))
tokenized.show(false)
```



Text Data - Removing Common Words

- ▶ Filters **stop words**, such as "the", "and", and "but".

```
import org.apache.spark.ml.feature.StopWordsRemover

val df = spark.createDataFrame(Seq((0, Seq("I", "saw", "the", "red", "balloon")),
  (1, Seq("Mary", "had", "a", "little", "lamb")))).toDF("id", "raw")

val englishStopWords = StopWordsRemover.loadDefaultStopWords("english")

val stops = new StopWordsRemover().setStopWords(englishStopWords)
  .setInputCol("raw").setOutputCol("WithoutStops")

stops.transform(df).show(false)
```



Text Data - Converting Words into Numerical Representations

- ▶ Counts instances of words in word features.
- ▶ Treats every row as a document, every word as a term, and the total collection of all terms as the vocabulary.

```
import org.apache.spark.ml.feature.CountVectorizer

val df = spark.createDataFrame(Seq((0, Array("a", "b", "c")),
  (1, Array("a", "b", "b", "c", "a)))).toDF("id", "words")

val cvModel = new CountVectorizer().setInputCol("words").setOutputCol("features")
  .setVocabSize(3).setMinDF(2)

val fittedCV = cvModel.fit(df)

fittedCV.transform(df).show(false)
```



Summary



Summary

- ▶ Spark: RDD
- ▶ Spark SQL: DataFrame
- ▶ MLlib
 - Transformers and Estimators
 - Pipeline
 - Feature engineering



References

- ▶ Matei Zaharia et al., Spark - The Definitive Guide, (Ch. 24 and 25)

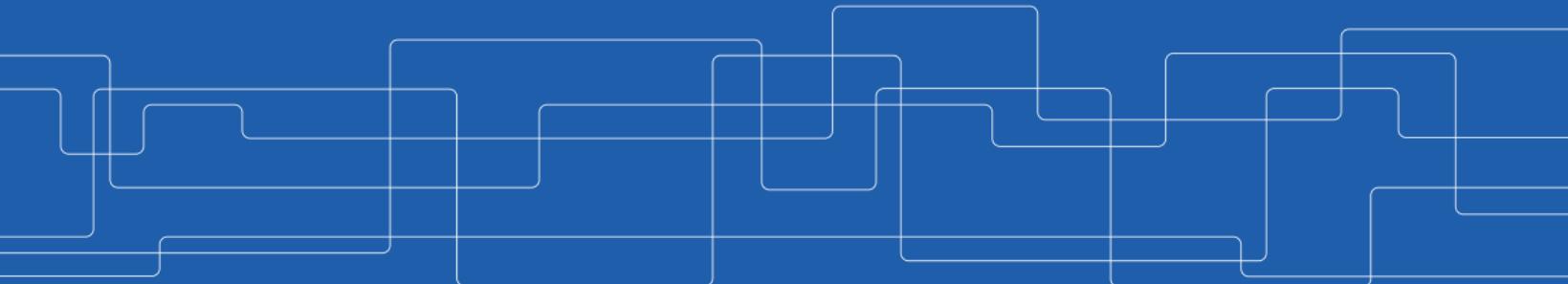


Questions?



Machine Learning - Regressions

Amir H. Payberah
payberah@kth.se
2021-11-09



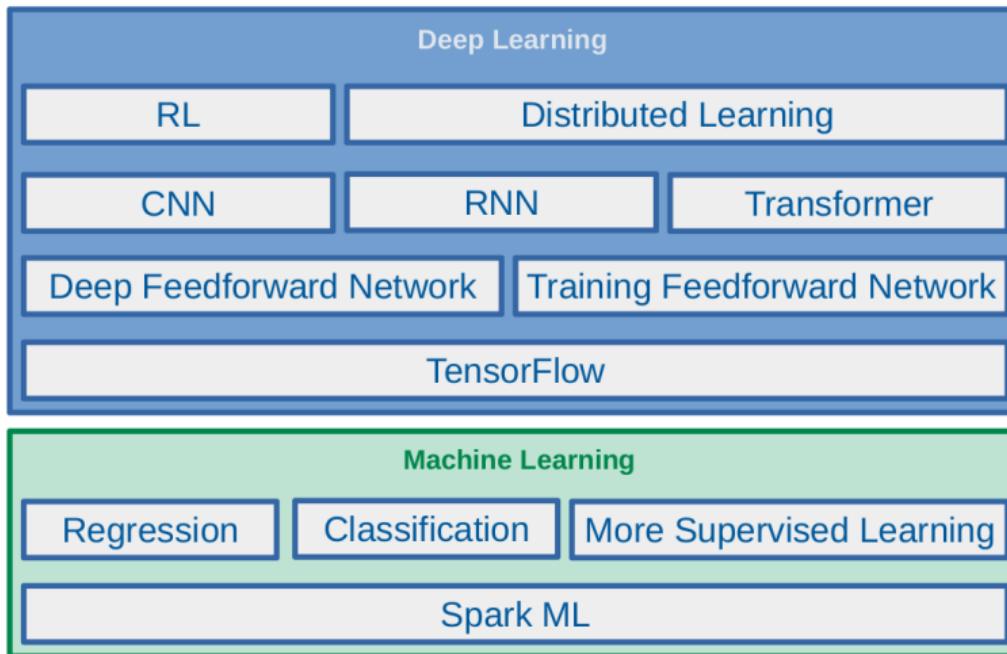


The Course Web Page

<https://id2223kth.github.io>
<https://tinyurl.com/6s5jy46a>

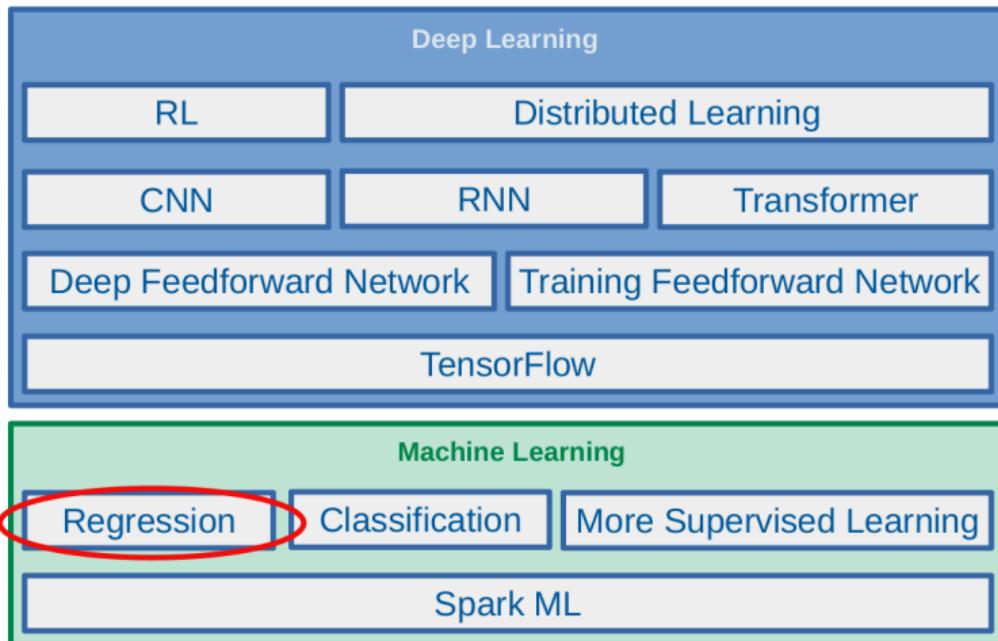


Where Are We?





Where Are We?





Let's Start with an Example



THIS IS MY DREAM HOUSE ALRIGHT, EXCEPT
IN MY DREAM IT WAS ABOUT HALF THIS PRICE.



The Housing Price Example (1/3)

- ▶ Given the dataset of m houses.

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:



The Housing Price Example (1/3)

- Given the dataset of m houses.

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:

- Predict the prices of other houses, as a function of the size of living area and number of bedrooms?



The Housing Price Example (2/3)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:
:	:	:



The Housing Price Example (2/3)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:

$$x^{(1)} = \begin{bmatrix} 2104 \\ 3 \end{bmatrix} \quad y^{(1)} = 400 \quad x^{(2)} = \begin{bmatrix} 1600 \\ 3 \end{bmatrix} \quad y^{(2)} = 330 \quad x^{(3)} = \begin{bmatrix} 2400 \\ 3 \end{bmatrix} \quad y^{(3)} = 369$$



The Housing Price Example (2/3)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:

$$x^{(1)} = \begin{bmatrix} 2104 \\ 3 \end{bmatrix} \quad y^{(1)} = 400 \quad x^{(2)} = \begin{bmatrix} 1600 \\ 3 \end{bmatrix} \quad y^{(2)} = 330 \quad x^{(3)} = \begin{bmatrix} 2400 \\ 3 \end{bmatrix} \quad y^{(3)} = 369$$

$$X = \begin{bmatrix} x^{(1)\top} \\ x^{(2)\top} \\ x^{(3)\top} \\ \vdots \end{bmatrix} = \begin{bmatrix} 2104 & 3 \\ 1600 & 3 \\ 2400 & 3 \\ \vdots & \vdots \end{bmatrix} \quad y = \begin{bmatrix} 400 \\ 330 \\ 369 \\ \vdots \end{bmatrix}$$

The Housing Price Example (2/3)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:

$$x^{(1)} = \begin{bmatrix} 2104 \\ 3 \end{bmatrix} \quad y^{(1)} = 400 \quad x^{(2)} = \begin{bmatrix} 1600 \\ 3 \end{bmatrix} \quad y^{(2)} = 330 \quad x^{(3)} = \begin{bmatrix} 2400 \\ 3 \end{bmatrix} \quad y^{(3)} = 369$$

$$X = \begin{bmatrix} x^{(1)\top} \\ x^{(2)\top} \\ x^{(3)\top} \\ \vdots \end{bmatrix} = \begin{bmatrix} 2104 & 3 \\ 1600 & 3 \\ 2400 & 3 \\ \vdots & \vdots \end{bmatrix} \quad y = \begin{bmatrix} 400 \\ 330 \\ 369 \\ \vdots \end{bmatrix}$$

- $x^{(i)} \in \mathbb{R}^2$: $x_1^{(i)}$ is the living area, and $x_2^{(i)}$ is the number of bedrooms of the i th house in the training set.

The Housing Price Example (3/3)

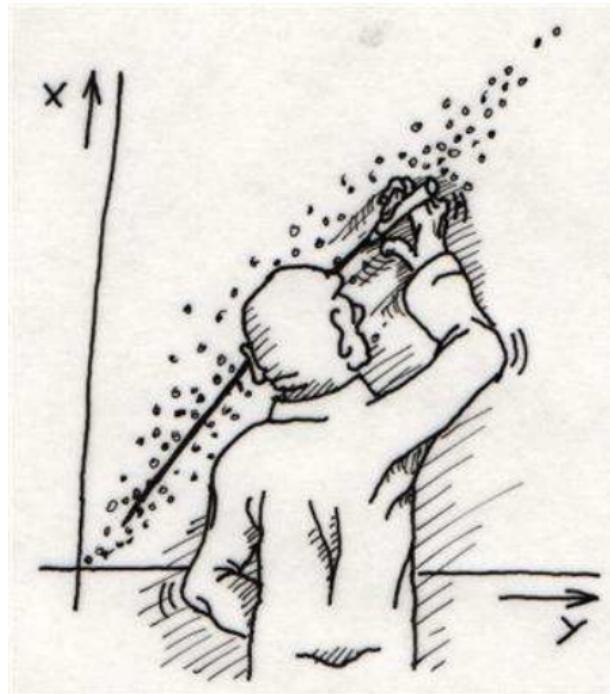
Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:

- ▶ Predict the prices of other houses \hat{y} as a function of the size of their living areas x_1 , and number of bedrooms x_2 , i.e., $\hat{y} = f(x_1, x_2)$
- ▶ E.g., what is \hat{y} , if $x_1 = 4000$ and $x_2 = 4$?

The Housing Price Example (3/3)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
:	:	:

- ▶ Predict the prices of other houses \hat{y} as a function of the size of their living areas x_1 , and number of bedrooms x_2 , i.e., $\hat{y} = f(x_1, x_2)$
- ▶ E.g., what is \hat{y} , if $x_1 = 4000$ and $x_2 = 4$?
- ▶ As an initial choice: $\hat{y} = f_w(x) = w_1x_1 + w_2x_2$



[<http://www.vias.org/science.cartoons/regression.html>]



Linear Regression



Linear Regression (1/2)

- ▶ Our goal: to build a system that takes input $x \in \mathbb{R}^n$ and predicts output $\hat{y} \in \mathbb{R}$.



Linear Regression (1/2)

- ▶ Our goal: to build a system that takes input $x \in \mathbb{R}^n$ and predicts output $\hat{y} \in \mathbb{R}$.
- ▶ In linear regression, the output \hat{y} is a linear function of the input x .

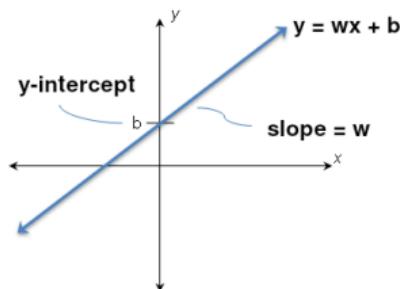
$$\begin{aligned}\hat{y} &= f_w(x) = w_1x_1 + w_2x_2 + \cdots + w_nx_n \\ \hat{y} &= w^T x\end{aligned}$$

- \hat{y} : the predicted value
- n : the number of features
- x_i : the i th feature value
- w_j : the j th model parameter ($w \in \mathbb{R}^n$)

Linear Regression (2/2)

- ▶ Linear regression often has one additional parameter, called **intercept b**:

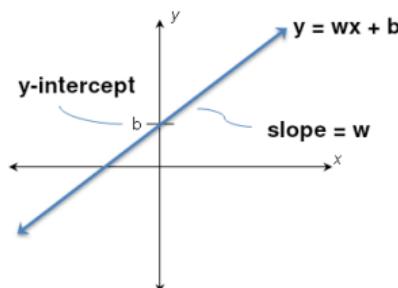
$$\hat{y} = w^T x + b$$



Linear Regression (2/2)

- ▶ Linear regression often has one additional parameter, called **intercept b**:

$$\hat{y} = \mathbf{w}^T \mathbf{x} + b$$



- ▶ Instead of adding the bias parameter **b**, we can augment **x** with an **extra entry** that is **always set to 1**.

$$\hat{y} = f_w(x) = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n, \text{ where } x_0 = 1$$



Linear Regression - Model Parameters

- ▶ Parameters $w \in \mathbb{R}^n$ are values that control the behavior of the model.



Linear Regression - Model Parameters

- ▶ Parameters $w \in \mathbb{R}^n$ are values that control the behavior of the model.
- ▶ w are a set of weights that determine how each feature affects the prediction.



Linear Regression - Model Parameters

- ▶ Parameters $w \in \mathbb{R}^n$ are values that control the behavior of the model.
- ▶ w are a set of weights that determine how each feature affects the prediction.
 - $w_i > 0$: increasing the value of the feature x_i , increases the value of our prediction \hat{y} .



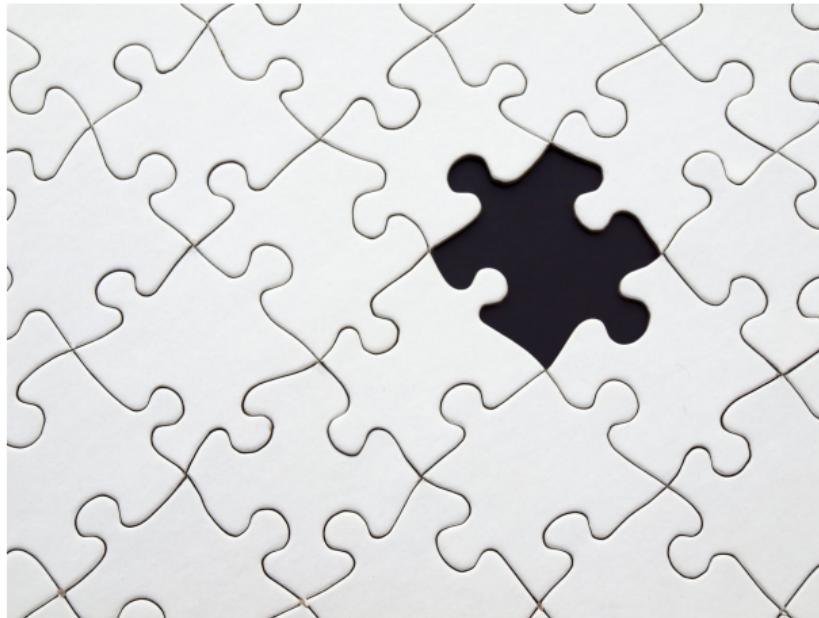
Linear Regression - Model Parameters

- ▶ Parameters $w \in \mathbb{R}^n$ are values that control the behavior of the model.
- ▶ w are a set of weights that determine how each feature affects the prediction.
 - $w_i > 0$: increasing the value of the feature x_i , increases the value of our prediction \hat{y} .
 - $w_i < 0$: increasing the value of the feature x_i , decreases the value of our prediction \hat{y} .



Linear Regression - Model Parameters

- ▶ Parameters $w \in \mathbb{R}^n$ are values that control the behavior of the model.
- ▶ w are a set of weights that determine how each feature affects the prediction.
 - $w_i > 0$: increasing the value of the feature x_i , increases the value of our prediction \hat{y} .
 - $w_i < 0$: increasing the value of the feature x_i , decreases the value of our prediction \hat{y} .
 - $w_i = 0$: the value of the feature x_i , has no effect on the prediction \hat{y} .

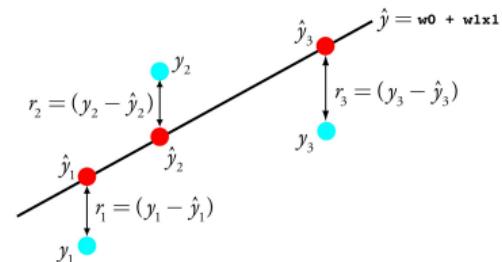
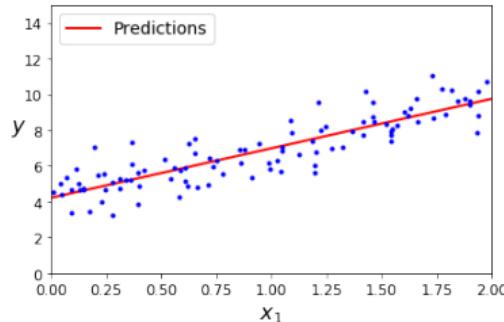


$$\hat{y} = f_w(x) = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$



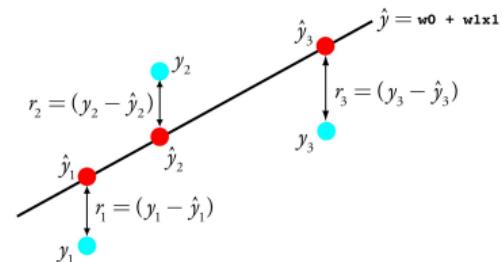
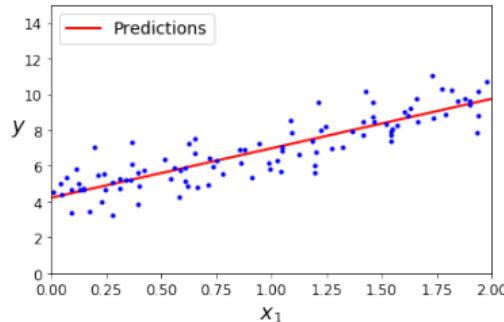
How to Learn Model Parameters w ?

Linear Regression - Cost Function (1/2)



- One reasonable model should make \hat{y} close to y , at least for the training dataset.

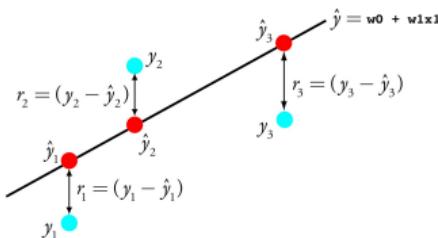
Linear Regression - Cost Function (1/2)



- One reasonable model should make \hat{y} close to y , at least for the training dataset.
- Residual: the difference between the dependent variable y and the predicted value \hat{y} .

$$r^{(i)} = y^{(i)} - \hat{y}^{(i)}$$

Linear Regression - Cost Function (2/2)

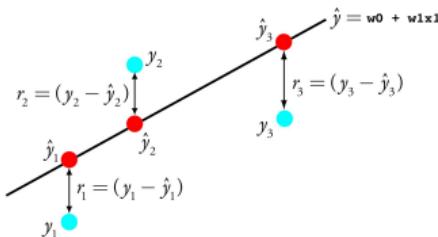


► Cost function $J(w)$

- For each value of the w , it measures how close the $\hat{y}^{(i)}$ is to the corresponding $y^{(i)}$.
- We can define $J(w)$ as the mean squared error (MSE):

$$J(w) = \text{MSE}(w) = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$

Linear Regression - Cost Function (2/2)



► Cost function $J(w)$

- For each value of the w , it measures how close the $\hat{y}^{(i)}$ is to the corresponding $y^{(i)}$.
- We can define $J(w)$ as the mean squared error (MSE):

$$\begin{aligned} J(w) &= \text{MSE}(w) = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2 \\ &= E[(\hat{y} - y)^2] = \frac{1}{m} \|\hat{y} - y\|_2^2 \end{aligned}$$



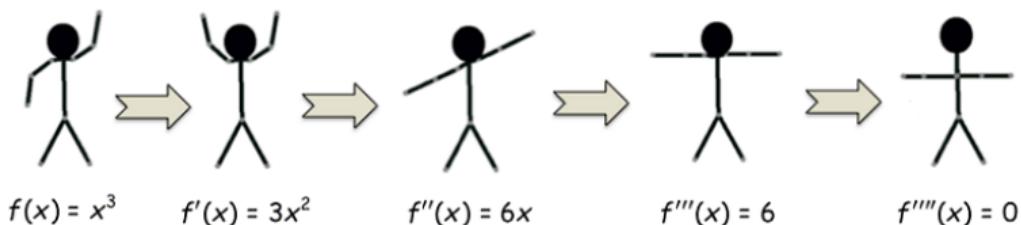
How to Learn Model Parameters?

- ▶ We want to choose w so as to minimize $J(w)$.
- ▶ Two approaches to find w :
 - Normal equation
 - Gradient descent



Normal Equation

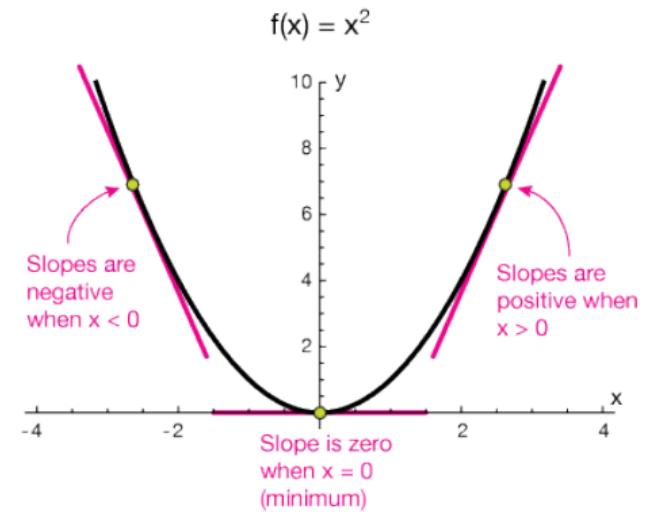
Derivatives and Gradient (1/4)



[<https://mathequality.wordpress.com/2012/09/26/derivative-dance-gangnam-style/>]

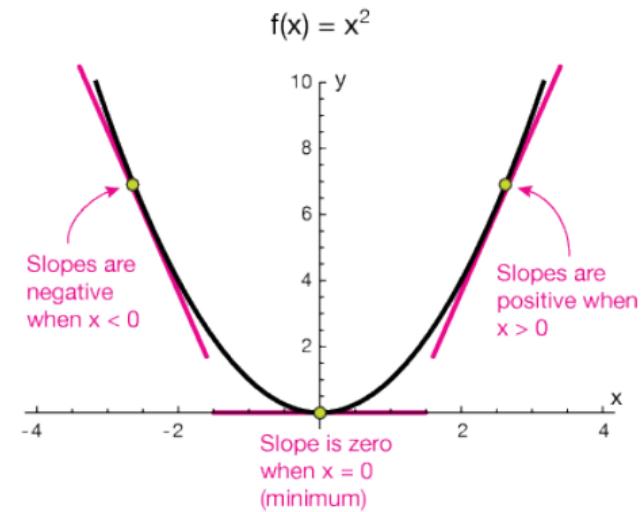
Derivatives and Gradient (2/4)

- The **first derivative** of $f(x)$, shown as $f'(x)$, shows the **slope** of the **tangent line** to the function at the point x .



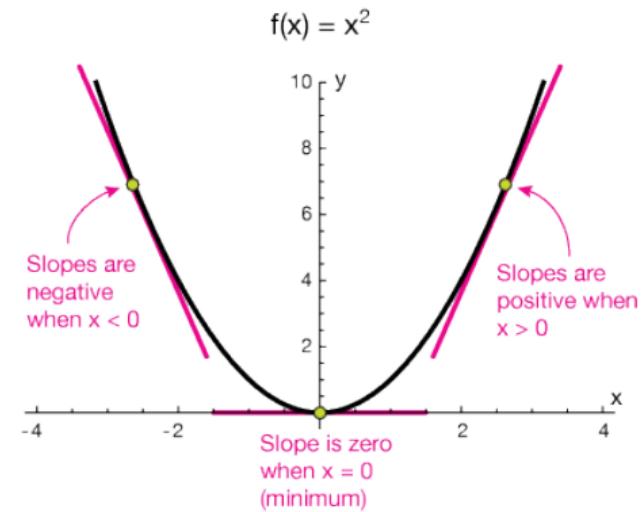
Derivatives and Gradient (2/4)

- ▶ The **first derivative** of $f(x)$, shown as $f'(x)$, shows the **slope** of the **tangent line** to the function at the point x .
- ▶ $f(x) = x^2 \Rightarrow f'(x) = 2x$



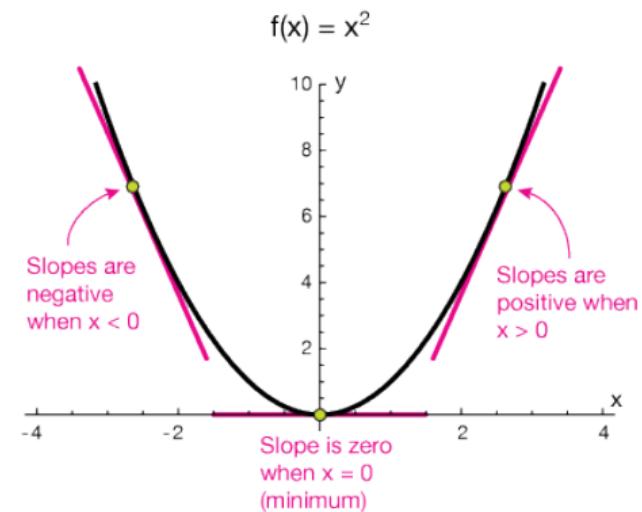
Derivatives and Gradient (2/4)

- ▶ The **first derivative** of $f(x)$, shown as $f'(x)$, shows the **slope** of the **tangent line** to the function at the point x .
- ▶ $f(x) = x^2 \Rightarrow f'(x) = 2x$
- ▶ If $f(x)$ is increasing, then $f'(x) > 0$



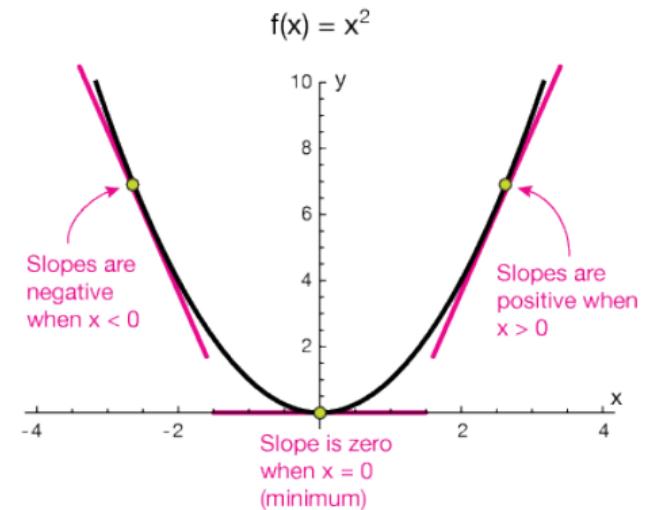
Derivatives and Gradient (2/4)

- ▶ The **first derivative** of $f(x)$, shown as $f'(x)$, shows the **slope** of the **tangent line** to the function at the point x .
- ▶ $f(x) = x^2 \Rightarrow f'(x) = 2x$
- ▶ If $f(x)$ is increasing, then $f'(x) > 0$
- ▶ If $f(x)$ is decreasing, then $f'(x) < 0$



Derivatives and Gradient (2/4)

- ▶ The **first derivative** of $f(x)$, shown as $f'(x)$, shows the **slope** of the **tangent line** to the function at the point x .
- ▶ $f(x) = x^2 \Rightarrow f'(x) = 2x$
- ▶ If $f(x)$ is increasing, then $f'(x) > 0$
- ▶ If $f(x)$ is decreasing, then $f'(x) < 0$
- ▶ If $f(x)$ is at local minimum/maximum, then $f'(x) = 0$





Derivatives and Gradient (3/4)

- ▶ What if a function has multiple arguments, e.g., $f(x_1, x_2, \dots, x_n)$
- ▶ **Partial derivatives:** the derivative with respect to a particular argument.
 - $\frac{\partial f}{\partial x_1}$, the derivative with respect to x_1
 - $\frac{\partial f}{\partial x_2}$, the derivative with respect to x_2



Derivatives and Gradient (3/4)

- ▶ What if a function has multiple arguments, e.g., $f(x_1, x_2, \dots, x_n)$
- ▶ **Partial derivatives:** the derivative with respect to a particular argument.
 - $\frac{\partial f}{\partial x_1}$, the derivative with respect to x_1
 - $\frac{\partial f}{\partial x_2}$, the derivative with respect to x_2
- ▶ $\frac{\partial f}{\partial x_i}$: shows how much the function f will change, if we change x_i .



Derivatives and Gradient (3/4)

- ▶ What if a function has multiple arguments, e.g., $f(x_1, x_2, \dots, x_n)$
- ▶ **Partial derivatives:** the derivative with respect to a particular argument.
 - $\frac{\partial f}{\partial x_1}$, the derivative with respect to x_1
 - $\frac{\partial f}{\partial x_2}$, the derivative with respect to x_2
- ▶ $\frac{\partial f}{\partial x_i}$: shows how much the function f will change, if we change x_i .
- ▶ **Gradient:** the vector of all partial derivatives for a function f .

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$



Derivatives and Gradient (4/4)

- ▶ What is the gradient of $f(x_1, x_2, x_3) = x_1 - x_1x_2 + x_3^2$?

Derivatives and Gradient (4/4)

- ▶ What is the gradient of $f(x_1, x_2, x_3) = x_1 - x_1x_2 + x_3^2$?

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial}{\partial x_1}(x_1 - x_1x_2 + x_3^2) \\ \frac{\partial}{\partial x_2}(x_1 - x_1x_2 + x_3^2) \\ \frac{\partial}{\partial x_3}(x_1 - x_1x_2 + x_3^2) \end{bmatrix} = \begin{bmatrix} 1 - x_2 \\ -x_1 \\ 2x_3 \end{bmatrix}$$



Normal Equation (1/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$\hat{y} = w^T x$$

Normal Equation (1/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$\hat{y} = w^T x$$

$$X = \begin{bmatrix} [x_1^{(1)}, x_2^{(1)}, \dots, x_n^{(1)}] \\ [x_1^{(2)}, x_2^{(2)}, \dots, x_n^{(2)}] \\ \vdots \\ [x_1^{(m)}, x_2^{(m)}, \dots, x_n^{(m)}] \end{bmatrix} = \begin{bmatrix} x^{(1)\top} \\ x^{(2)\top} \\ \vdots \\ x^{(m)\top} \end{bmatrix} \quad \hat{y} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{bmatrix}$$

Normal Equation (1/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$\hat{y} = w^T x$$

$$X = \begin{bmatrix} [x_1^{(1)}, x_2^{(1)}, \dots, x_n^{(1)}] \\ [x_1^{(2)}, x_2^{(2)}, \dots, x_n^{(2)}] \\ \vdots \\ [x_1^{(m)}, x_2^{(m)}, \dots, x_n^{(m)}] \end{bmatrix} = \begin{bmatrix} x^{(1)\top} \\ x^{(2)\top} \\ \vdots \\ x^{(m)\top} \end{bmatrix} \quad \hat{y} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{bmatrix}$$

$$\hat{y} = w^T X^T \text{ or } \hat{y} = Xw$$



Normal Equation (2/2)

- ▶ To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

Normal Equation (2/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$J(w) = \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2, \nabla_w J(w) = 0$$

Normal Equation (2/2)

- ▶ To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$\begin{aligned}J(w) &= \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2, \quad \nabla_w J(w) = 0 \\&\Rightarrow \nabla_w \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 = 0\end{aligned}$$

Normal Equation (2/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$\begin{aligned}J(w) &= \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2, \quad \nabla_w J(w) = 0 \\&\Rightarrow \nabla_w \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 = 0 \\&\Rightarrow \nabla_w \frac{1}{m} \|\mathbf{Xw} - \mathbf{y}\|_2^2 = 0\end{aligned}$$

Normal Equation (2/2)

- ▶ To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$\begin{aligned}J(w) &= \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2, \quad \nabla_w J(w) = 0 \\&\Rightarrow \nabla_w \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 = 0 \\&\Rightarrow \nabla_w \frac{1}{m} \|\mathbf{X}w - \mathbf{y}\|_2^2 = 0 \\&\Rightarrow \nabla_w (\mathbf{X}w - \mathbf{y})^\top (\mathbf{X}w - \mathbf{y}) = 0\end{aligned}$$

Normal Equation (2/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$J(w) = \frac{1}{m} \|\hat{y} - y\|_2^2, \nabla_w J(w) = 0$$

$$\Rightarrow \nabla_w \frac{1}{m} \|\hat{y} - y\|_2^2 = 0$$

$$\Rightarrow \nabla_w \frac{1}{m} \|Xw - y\|_2^2 = 0$$

$$\Rightarrow \nabla_w (Xw - y)^T (Xw - y) = 0$$

$$\Rightarrow \nabla_w (w^T X^T X w - 2w^T X^T y + y^T y) = 0$$

Normal Equation (2/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$J(w) = \frac{1}{m} \|\hat{y} - y\|_2^2, \nabla_w J(w) = 0$$

$$\Rightarrow \nabla_w \frac{1}{m} \|\hat{y} - y\|_2^2 = 0$$

$$\Rightarrow \nabla_w \frac{1}{m} \|Xw - y\|_2^2 = 0$$

$$\Rightarrow \nabla_w (Xw - y)^T (Xw - y) = 0$$

$$\Rightarrow \nabla_w (w^T X^T X w - 2w^T X^T y + y^T y) = 0$$

$$\Rightarrow 2X^T X w - 2X^T y = 0$$

Normal Equation (2/2)

- To minimize $J(w)$, we can simply solve for where its gradient is 0: $\nabla_w J(w) = 0$

$$J(w) = \frac{1}{m} \|\hat{y} - y\|_2^2, \nabla_w J(w) = 0$$

$$\Rightarrow \nabla_w \frac{1}{m} \|\hat{y} - y\|_2^2 = 0$$

$$\Rightarrow \nabla_w \frac{1}{m} \|Xw - y\|_2^2 = 0$$

$$\Rightarrow \nabla_w (Xw - y)^T (Xw - y) = 0$$

$$\Rightarrow \nabla_w (w^T X^T X w - 2w^T X^T y + y^T y) = 0$$

$$\Rightarrow 2X^T X w - 2X^T y = 0$$

$$\Rightarrow w = (X^T X)^{-1} X^T y$$



Normal Equation - Example (1/7)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540

- ▶ Predict the value of \hat{y} , when $x_1 = 4000$ and $x_2 = 4$.

Normal Equation - Example (1/7)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540

- ▶ Predict the value of \hat{y} , when $x_1 = 4000$ and $x_2 = 4$.
- ▶ We should find w_0 , w_1 , and w_2 in $\hat{y} = w_0 + w_1x_1 + w_2x_2$.
- ▶ $w = (X^T X)^{-1} X^T y$.

Normal Equation - Example (2/7)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540

$$X = \left[\begin{array}{c|ccc} 1 & 2104 & 3 \\ 1 & 1600 & 3 \\ 1 & 2400 & 3 \\ 1 & 1416 & 2 \\ 1 & 3000 & 4 \end{array} \right] \quad y = \begin{bmatrix} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{bmatrix}$$

Normal Equation - Example (3/7)

$$X^T X = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2104 & 1600 & 2400 & 1416 & 3000 \\ 3 & 3 & 3 & 2 & 4 \end{bmatrix} \begin{bmatrix} 1 & 2104 & 3 \\ 1 & 1600 & 3 \\ 1 & 2400 & 3 \\ 1 & 1416 & 2 \\ 1 & 3000 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 10520 & 23751872 & 33144 \\ 15 & 33144 & 47 \end{bmatrix}$$



Normal Equation - Example (4/7)

$$(X^T X)^{-1} = \begin{bmatrix} 4.90366455e + 00 & 7.48766737e - 04 & -2.09302326e + 00 \\ 7.48766737e - 04 & 2.75281889e - 06 & -2.18023256e - 03 \\ -2.09302326e + 00 & -2.18023256e - 03 & 2.22674419e + 00 \end{bmatrix}$$

Normal Equation - Example (5/7)

$$X^T y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2104 & 1600 & 2400 & 1416 & 3000 \\ 3 & 3 & 3 & 2 & 4 \end{bmatrix} \begin{bmatrix} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{bmatrix} = \begin{bmatrix} 1871 \\ 4203712 \\ 5921 \end{bmatrix}$$

Normal Equation - Example (6/7)

$$\begin{aligned} w = (X^T X)^{-1} X^T y &= \begin{bmatrix} 4.90366455e + 00 & 7.48766737e - 04 & -2.09302326e + 00 \\ 7.48766737e - 04 & 2.75281889e - 06 & -2.18023256e - 03 \\ -2.09302326e + 00 & -2.18023256e - 03 & 2.22674419e + 00 \end{bmatrix} \begin{bmatrix} 1871 \\ 4203712 \\ 5921 \end{bmatrix} \\ &= \begin{bmatrix} -7.04346018e + 01 \\ 6.38433756e - 02 \\ 1.03436047e + 02 \end{bmatrix} \end{aligned}$$



Normal Equation - Example (7/7)

- ▶ Predict the value of y , when $x_1 = 4000$ and $x_2 = 4$.

$$\hat{y} = -7.04346018e + 01 + 6.38433756e - 02 \times 4000 + 1.03436047e + 02 \times 4 \approx 599$$



Normal Equation in Spark

```
case class house(x1: Long, x2: Long, y: Long)

val trainData = Seq(house(2104, 3, 400), house(1600, 3, 330), house(2400, 3, 369),
                    house(1416, 2, 232), house(3000, 4, 540)).toDF

val testData = Seq(house(4000, 4, 0)).toDF
```



Normal Equation in Spark

```
case class house(x1: Long, x2: Long, y: Long)

val trainData = Seq(house(2104, 3, 400), house(1600, 3, 330), house(2400, 3, 369),
                    house(1416, 2, 232), house(3000, 4, 540)).toDF

val testData = Seq(house(4000, 4, 0)).toDF
```

```
import org.apache.spark.ml.feature.VectorAssembler

val va = new VectorAssembler().setInputCols(Array("x1", "x2")).setOutputCol("features")

val train = va.transform(trainData)
val test = va.transform(testData)
```



Normal Equation in Spark

```
case class house(x1: Long, x2: Long, y: Long)

val trainData = Seq(house(2104, 3, 400), house(1600, 3, 330), house(2400, 3, 369),
                    house(1416, 2, 232), house(3000, 4, 540)).toDF

val testData = Seq(house(4000, 4, 0)).toDF
```

```
import org.apache.spark.ml.feature.VectorAssembler

val va = new VectorAssembler().setInputCols(Array("x1", "x2")).setOutputCol("features")

val train = va.transform(trainData)
val test = va.transform(testData)
```

```
import org.apache.spark.ml.regression.LinearRegression

val lr = new LinearRegression().setFeaturesCol("features").setLabelCol("y").setSolver("normal")
val lrModel = lr.fit(train)
lrModel.transform(test).show
```



Normal Equation - Computational Complexity

- ▶ The computational complexity of inverting $X^T X$ is $O(n^3)$.
 - For an $m \times n$ matrix (where n is the number of features).



Normal Equation - Computational Complexity

- ▶ The computational complexity of inverting $X^T X$ is $O(n^3)$.
 - For an $m \times n$ matrix (where n is the number of features).
- ▶ But, this equation is linear with regards to the number of instances in the training set (it is $O(m)$).
 - It handles large training sets efficiently, provided they can fit in memory.



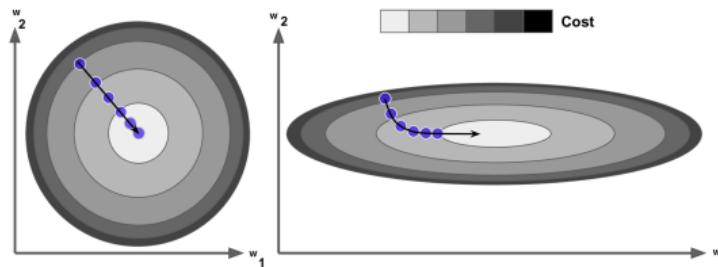
[<https://dailyfintech.com/2017/03/13/now-all-we-need-is-for-blockchain-to-become-technologically-boring>]



Gradient Descent

Gradient Descent (1/2)

- ▶ Gradient descent is a generic optimization algorithm capable of finding optimal solutions to a wide range of problems.
- ▶ The idea: to tweak parameters iteratively in order to minimize a cost function.



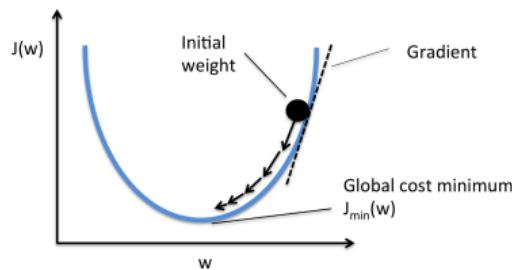
Gradient Descent (2/2)

- ▶ Suppose you are **lost** in the **mountains** in a dense fog.
- ▶ You can only feel the **slope** of the ground below your feet.
- ▶ A strategy to **get to the bottom** of the valley is to **go downhill** in the **direction of the steepest slope**.



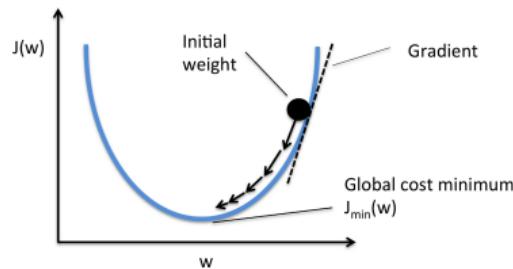
Gradient Descent - Iterative Optimization Algorithm

- ▶ Choose a starting point, e.g., filling w with random values.



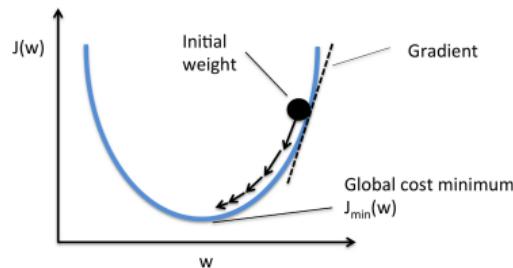
Gradient Descent - Iterative Optimization Algorithm

- ▶ Choose a starting point, e.g., filling w with random values.
- ▶ If the **stopping criterion** is true return the **current solution**, otherwise continue.



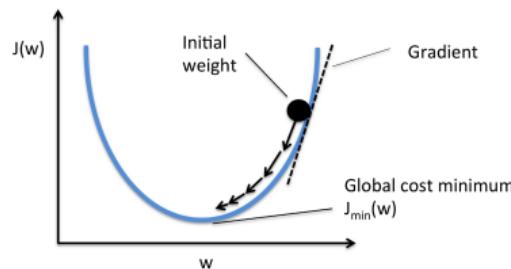
Gradient Descent - Iterative Optimization Algorithm

- ▶ Choose a starting point, e.g., filling w with random values.
- ▶ If the **stopping criterion** is true return the **current solution**, otherwise continue.
- ▶ Find a **descent direction**, a **direction in which the function value decreases** near the current point.



Gradient Descent - Iterative Optimization Algorithm

- ▶ Choose a **starting point**, e.g., filling w with **random values**.
- ▶ If the **stopping criterion** is true return the **current solution**, otherwise continue.
- ▶ Find a **descent direction**, a **direction in which the function value decreases** near the current point.
- ▶ Determine the **step size**, the **length of a step** in the given direction.





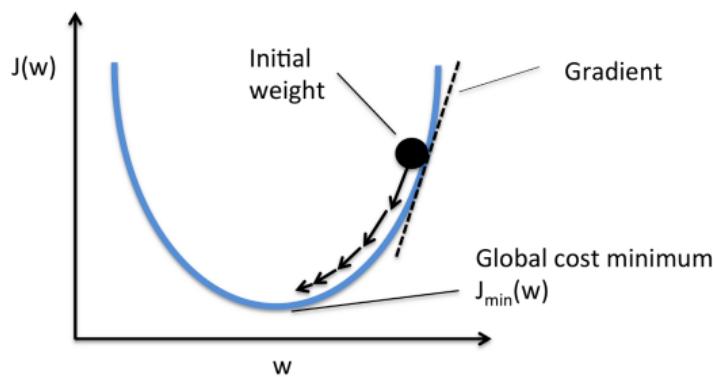
Gradient Descent - Key Points

- ▶ Stopping criterion
- ▶ Descent direction
- ▶ Step size (learning rate)

Gradient Descent - Stopping Criterion

- ▶ The **cost function minimum** property: the **gradient** has to be **zero**.

$$\nabla_w J(w) = 0$$





Gradient Descent - Descent Direction (1/2)

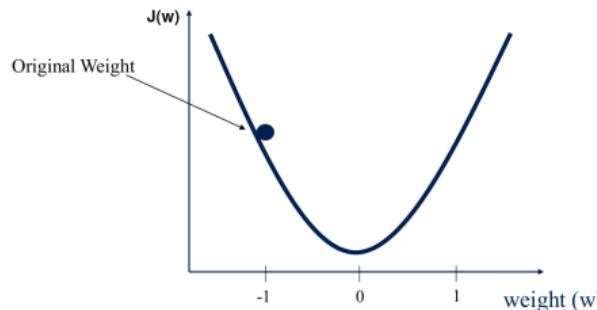
- ▶ Direction in which the **function value decreases** near the current point.
- ▶ Find the **direction of descent (slope)**.

Gradient Descent - Descent Direction (1/2)

- ▶ Direction in which the **function value decreases** near the current point.
- ▶ Find the **direction of descent (slope)**.
- ▶ Example:

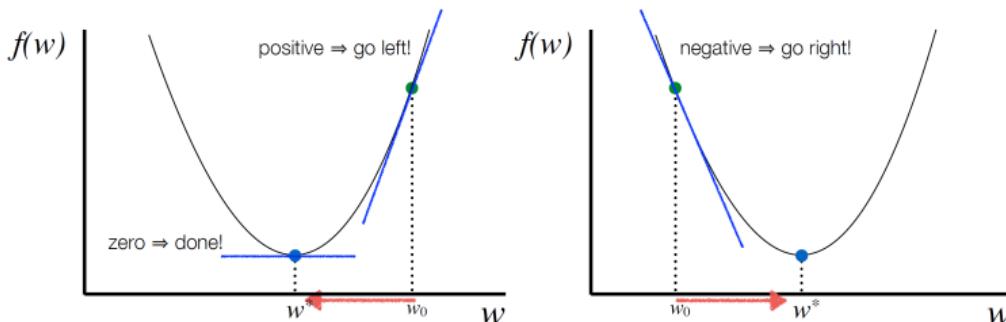
$$J(w) = w^2$$

$$\frac{\partial J(w)}{\partial w} = 2w = -2 \text{ at } w = -1$$



Gradient Descent - Descent Direction (2/2)

- Follow the opposite direction of the slope.



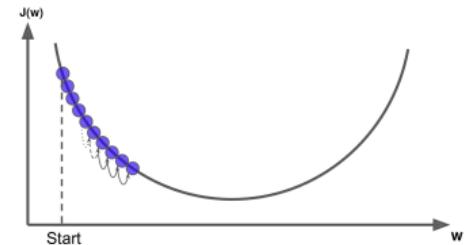


Gradient Descent - Learning Rate

- ▶ **Learning rate**: the length of steps.

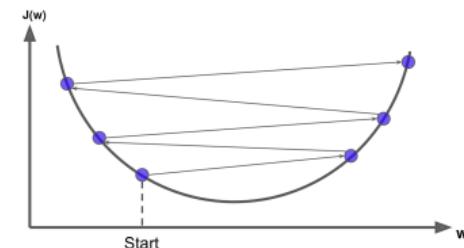
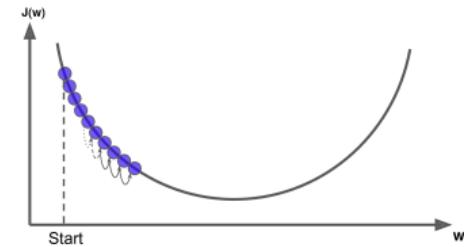
Gradient Descent - Learning Rate

- ▶ **Learning rate**: the length of steps.
- ▶ If it is **too small**: many iterations to converge.



Gradient Descent - Learning Rate

- ▶ **Learning rate**: the length of steps.
- ▶ If it is **too small**: many iterations to converge.
- ▶ If it is **too high**: the algorithm might diverge.



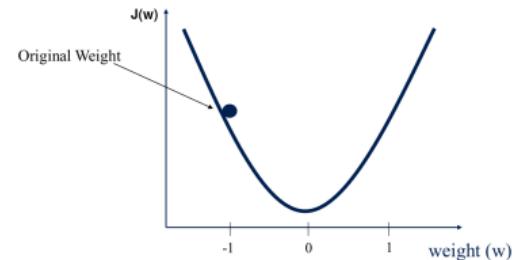


Gradient Descent - How to Learn Model Parameters w?

- **Goal:** find w that minimizes $J(w) = \sum_{i=1}^m (w^T x^{(i)} - y^{(i)})^2$.

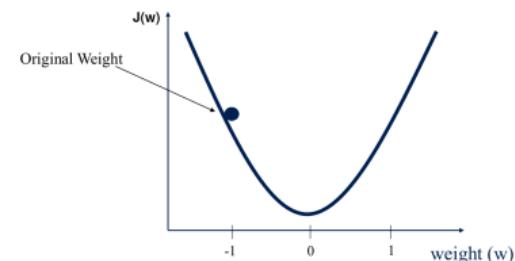
Gradient Descent - How to Learn Model Parameters w ?

- ▶ **Goal:** find w that minimizes $J(w) = \sum_{i=1}^m (w^T x^{(i)} - y^{(i)})^2$.
- ▶ Start at a **random point**, and repeat the following **steps**, until the **stopping criterion** is satisfied:



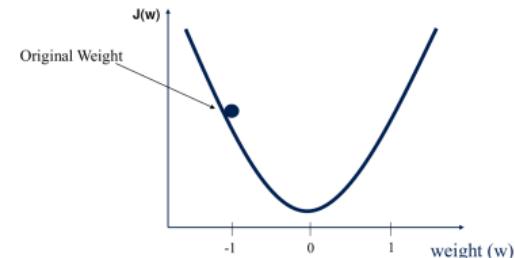
Gradient Descent - How to Learn Model Parameters w ?

- ▶ **Goal:** find w that minimizes $J(w) = \sum_{i=1}^m (w^T x^{(i)} - y^{(i)})^2$.
- ▶ Start at a random point, and repeat the following steps, until the stopping criterion is satisfied:
 1. Determine a descent direction $\frac{\partial J(w)}{\partial w}$



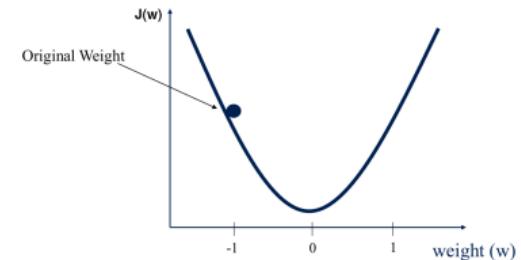
Gradient Descent - How to Learn Model Parameters w ?

- ▶ **Goal:** find w that minimizes $J(w) = \sum_{i=1}^m (w^T x^{(i)} - y^{(i)})^2$.
- ▶ Start at a random point, and repeat the following steps, until the stopping criterion is satisfied:
 1. Determine a descent direction $\frac{\partial J(w)}{\partial w}$
 2. Choose a step size η



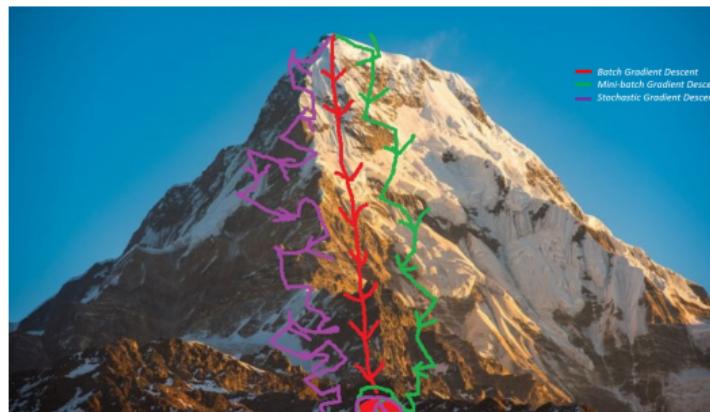
Gradient Descent - How to Learn Model Parameters w ?

- ▶ **Goal:** find w that minimizes $J(w) = \sum_{i=1}^m (w^T x^{(i)} - y^{(i)})^2$.
- ▶ Start at a **random point**, and repeat the following **steps**, until the **stopping criterion** is satisfied:
 1. Determine a **descent direction** $\frac{\partial J(w)}{\partial w}$
 2. Choose a **step size** η
 3. **Update** the parameters: $w^{(\text{next})} = w - \eta \frac{\partial J(w)}{\partial w}$
(should be done for **all parameters simultaneously**)



Gradient Descent - Different Algorithms

- ▶ Batch gradient descent
- ▶ Stochastic gradient descent
- ▶ Mini-batch gradient descent



[<https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3>]



Batch Gradient Descent



Batch Gradient Descent (1/2)

- ▶ Repeat the following **steps**, until the **stopping criterion** is satisfied:

1. Determine a **descent direction** $\frac{\partial J(w)}{\partial w}$ for all parameters w .

$$J(w) = \sum_{i=1}^m (w^\top x^{(i)} - y^{(i)})^2$$



Batch Gradient Descent (1/2)

- ▶ Repeat the following **steps**, until the stopping criterion is satisfied:

1. Determine a **descent direction** $\frac{\partial J(w)}{\partial w}$ for all parameters w .

$$J(w) = \sum_{i=1}^m (w^T x^{(i)} - y^{(i)})^2$$

$$\frac{\partial J(w)}{\partial w_j} = \frac{2}{m} \sum_{i=1}^m (w^T x^{(i)} - y^{(i)}) x_j^{(i)}$$
$$\nabla_w J(w) = \begin{bmatrix} \frac{\partial J(w)}{\partial w_0} \\ \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{bmatrix} = \frac{2}{m} X^T (Xw - y)$$

2. Choose a **step size** η



Batch Gradient Descent (1/2)

- ▶ Repeat the following **steps**, until the stopping criterion is satisfied:

1. Determine a **descent direction** $\frac{\partial J(w)}{\partial w}$ for all parameters w .

$$J(w) = \sum_{i=1}^m (w^T x^{(i)} - y^{(i)})^2$$

$$\frac{\partial J(w)}{\partial w_j} = \frac{2}{m} \sum_{i=1}^m (w^T x^{(i)} - y^{(i)}) x_j^{(i)}$$
$$\nabla_w J(w) = \begin{bmatrix} \frac{\partial J(w)}{\partial w_0} \\ \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{bmatrix} = \frac{2}{m} X^T (Xw - y)$$

2. Choose a **step size** η
3. **Update** the parameters: $w^{(\text{next})} = w - \eta \nabla_w J(w)$



Batch Gradient Descent (2/2)

- ▶ The algorithm is called **Batch Gradient Descent**, because at each step, calculations are over the **full training set X** .
- ▶ As a result it is **slow on very large training sets**, i.e., large m .
- ▶ But, it **scales well** with the **number of features n** .

Batch Gradient Descent - Example (1/5)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2$$

$$X = \left[\begin{array}{c|ccc} 1 & 2104 & 3 \\ 1 & 1600 & 3 \\ 1 & 2400 & 3 \\ 1 & 1416 & 2 \\ 1 & 3000 & 4 \end{array} \right] \quad y = \left[\begin{array}{c} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{array} \right]$$

Batch Gradient Descent - Example (2/5)

$$X = \left[\begin{array}{c|ccc} 1 & 2104 & 3 \\ 1 & 1600 & 3 \\ 1 & 2400 & 3 \\ 1 & 1416 & 2 \\ 1 & 3000 & 4 \end{array} \right] \quad y = \left[\begin{array}{c} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{array} \right]$$

$$\begin{aligned}\frac{\partial J(w)}{\partial w_0} &= \frac{2}{m} \sum_{i=1}^m (w^T x^{(i)} - y^{(i)}) x_0^{(i)} \\ &= \frac{2}{5} [(w_0 + 2104w_1 + 3w_2 - 400) + (w_0 + 1600w_1 + 3w_2 - 330) + \\ &\quad (w_0 + 2400w_1 + 3w_2 - 369) + (w_0 + 1416w_1 + 2w_2 - 232) + (w_0 + 3000w_1 + 4w_2 - 540)]\end{aligned}$$

Batch Gradient Descent - Example (3/5)

$$X = \left[\begin{array}{c|ccc} 1 & 2104 & 3 \\ 1 & 1600 & 3 \\ 1 & 2400 & 3 \\ 1 & 1416 & 2 \\ 1 & 3000 & 4 \end{array} \right] \quad y = \left[\begin{array}{c} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{array} \right]$$

$$\begin{aligned}\frac{\partial J(w)}{\partial w_1} &= \frac{2}{m} \sum_{i=1}^m (w^\top x^{(i)} - y^{(i)}) x_1^{(i)} \\ &= \frac{2}{5} [2104(w_0 + 2104w_1 + 3w_2 - 400) + 1600(w_0 + 1600w_1 + 3w_2 - 330) + \\ &\quad 2400(w_0 + 2400w_1 + 3w_2 - 369) + 1416(w_0 + 1416w_1 + 2w_2 - 232) + 3000(w_0 + 3000w_1 + 4w_2 - 540)]\end{aligned}$$

Batch Gradient Descent - Example (4/5)

$$X = \left[\begin{array}{c|ccc} 1 & 2104 & 3 \\ 1 & 1600 & 3 \\ 1 & 2400 & 3 \\ 1 & 1416 & 2 \\ 1 & 3000 & 4 \end{array} \right] \quad y = \left[\begin{array}{c} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{array} \right]$$

$$\begin{aligned}\frac{\partial J(w)}{\partial w_2} &= \frac{2}{m} \sum_{i=1}^m (w^T x^{(i)} - y^{(i)}) x_2^{(i)} \\ &= \frac{2}{5} [3(w_0 + 2104w_1 + 3w_2 - 400) + 3(w_0 + 1600w_1 + 3w_2 - 330) + \\ &\quad 3(w_0 + 2400w_1 + 3w_2 - 369) + 2(w_0 + 1416w_1 + 2w_2 - 232) + 4(w_0 + 3000w_1 + 4w_2 - 540)]\end{aligned}$$



Batch Gradient Descent - Example (5/5)

$$w_0^{(\text{next})} = w_0 - \eta \frac{\partial J(w)}{\partial w_0}$$

$$w_1^{(\text{next})} = w_1 - \eta \frac{\partial J(w)}{\partial w_1}$$

$$w_2^{(\text{next})} = w_2 - \eta \frac{\partial J(w)}{\partial w_2}$$



Stochastic Gradient Descent



Stochastic Gradient Descent (1/3)

- ▶ Batch gradient descent problem: it's slow, because it uses the whole training set to compute the gradients at every step.

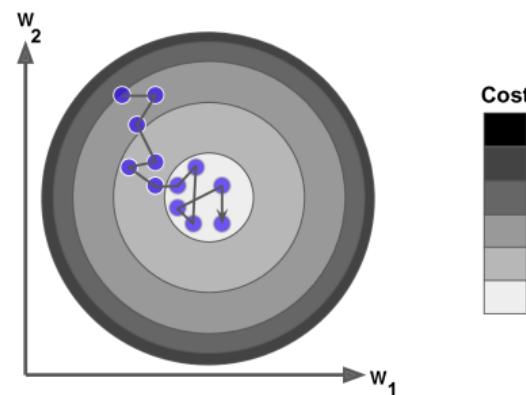


Stochastic Gradient Descent (1/3)

- ▶ **Batch gradient descent problem:** it's **slow**, because it uses the **whole training set** to compute the gradients at **every step**.
- ▶ **Stochastic gradient descent** computes the gradients based on only a **single instance**.
 - It picks a **random instance** in the **training set** at **every step**.

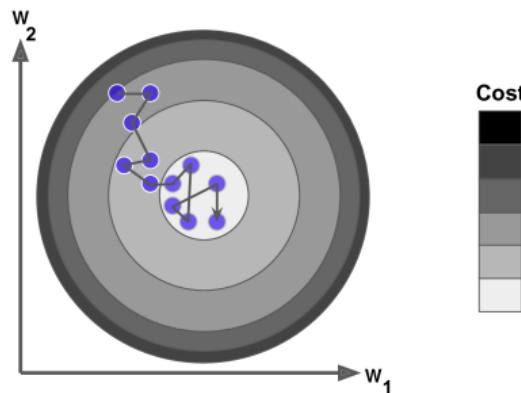
Stochastic Gradient Descent (2/3)

- ▶ The algorithm is much **faster**, but **less regular** than batch gradient descent.



Stochastic Gradient Descent (2/3)

- ▶ The algorithm is much **faster**, but **less regular** than batch gradient descent.
 - Instead of decreasing until it reaches the minimum, the **cost function will bounce up and down**.
 - It **never settles down**.





Stochastic Gradient Descent (3/3)

- ▶ With randomness the algorithm **can never settle at the minimum**.
- ▶ One solution is **simulated annealing**: start with **large learning rate**, then make it **smaller and smaller**.



Stochastic Gradient Descent (3/3)

- ▶ With randomness the algorithm **can never settle at the minimum**.
- ▶ One solution is **simulated annealing**: start with **large learning rate**, then make it **smaller and smaller**.
- ▶ **Learning schedule**: the function that **determines the learning rate** at each step.

Stochastic Gradient Descent - Example (1/3)

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2$$

$$X = \left[\begin{array}{ccc|c} 1 & 2104 & 3 & 400 \\ 1 & 1600 & 3 & 330 \\ 1 & 2400 & 3 & 369 \\ 1 & 1416 & 2 & 232 \\ 1 & 3000 & 4 & 540 \end{array} \right] \quad y = \left[\begin{array}{c} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{array} \right]$$

Stochastic Gradient Descent - Example (2/3)

$$X = \left[\begin{array}{c|ccc} 1 & 2104 & 3 \\ 1 & 1600 & 3 \\ 1 & 2400 & 3 \\ 1 & 1416 & 2 \\ 1 & 3000 & 4 \end{array} \right] \quad y = \left[\begin{array}{c} 400 \\ 330 \\ 369 \\ 232 \\ 540 \end{array} \right]$$

$$\frac{\partial J(w)}{\partial w_0} = \frac{2}{m}(w^T x^{(i)} - y^{(i)})x_0^{(i)} = \frac{2}{5}[(w_0 + 1600w_1 + 3w_2 - 330)]$$

$$\frac{\partial J(w)}{\partial w_1} = \frac{2}{m}(w^T x^{(i)} - y^{(i)})x_1^{(i)} = \frac{2}{5}[1600(w_0 + 1600w_1 + 3w_2 - 330)]$$

$$\frac{\partial J(w)}{\partial w_2} = \frac{2}{m}(w^T x^{(i)} - y^{(i)})x_2^{(i)} = \frac{2}{5}[3(w_0 + 1600w_1 + 3w_2 - 330)]$$



Stochastic Gradient Descent - Example (3/3)

$$w_0^{(\text{next})} = w_0 - \eta \frac{\partial J(w)}{\partial w_0}$$

$$w_1^{(\text{next})} = w_1 - \eta \frac{\partial J(w)}{\partial w_1}$$

$$w_2^{(\text{next})} = w_2 - \eta \frac{\partial J(w)}{\partial w_2}$$



Mini-Batch Gradient Descent

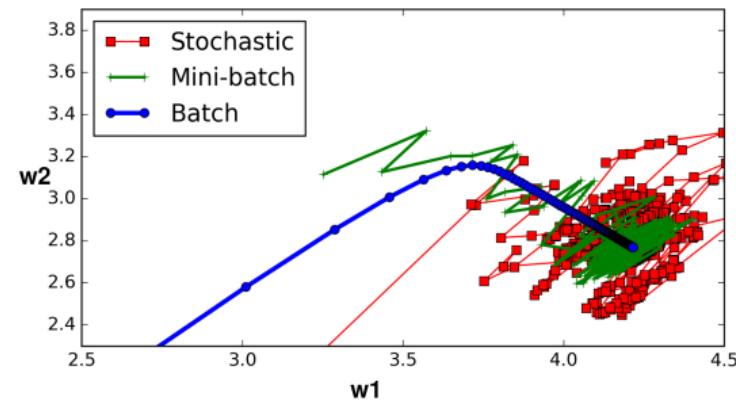


Mini-Batch Gradient Descent

- ▶ **Batch gradient descent**: at each step, it computes the gradients based on the **full training set**.
- ▶ **Stochastic gradient descent**: at each step, it computes the gradients based on **just one instance**.
- ▶ **Mini-batch gradient descent**: at each step, it computes the gradients based on small random sets of instances called **mini-batches**.

Comparison of Algorithms for Linear Regression

Algorithm	Large m	Large n
Normal Equation	Fast	Slow
Batch GD	Slow	Fast
Stochastic GD	Fast	Fast
Mini-batch GD	Fast	Fast





Gradient Descent in Spark

```
val data = spark.read.format("libsvm").load("data.txt")
```



Gradient Descent in Spark

```
val data = spark.read.format("libsvm").load("data.txt")
```

```
import org.apache.spark.ml.regression.LinearRegression
```

```
val lr = new LinearRegression().setMaxIter(10)
```

```
val lrModel = lr.fit(data)
```



Gradient Descent in Spark

```
val data = spark.read.format("libsvm").load("data.txt")

import org.apache.spark.ml.regression.LinearRegression

val lr = new LinearRegression().setMaxIter(10)

val lrModel = lr.fit(data)

println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

val trainingSummary = lrModel.summary
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
```



Generalization



Training Data and Test Data

- ▶ Split data into a **training set** and a **test set**.

```
val data = spark.read.format("libsvm").load("data.txt")  
  
val Array(trainDF, testDF) = data.randomSplit(Array(0.8, 0.2))
```

Full Dataset:

Training Data	Test Data
---------------	-----------



Training Data and Test Data

- ▶ Split data into a **training set** and a **test set**.
- ▶ Use **training set** when **training a machine learning model**.
 - Compute **training error** on the training set.
 - Try to **reduce** this training error.

```
val data = spark.read.format("libsvm").load("data.txt")  
  
val Array(trainDF, testDF) = data.randomSplit(Array(0.8, 0.2))
```

Full Dataset:

Training Data	Test Data
---------------	-----------



Training Data and Test Data

- ▶ Split data into a **training set** and a **test set**.
- ▶ Use **training set** when **training a machine learning model**.
 - Compute **training error** on the training set.
 - Try to **reduce** this training error.
- ▶ Use **test set** to **measure the accuracy of the model**.
 - **Test error** is the error when you run the **trained model** on **test data (new data)**.

```
val data = spark.read.format("libsvm").load("data.txt")  
  
val Array(trainDF, testDF) = data.randomSplit(Array(0.8, 0.2))
```

Full Dataset:

Training Data	Test Data
---------------	-----------



Generalization

- ▶ **Generalization:** make a model that performs **well** on **test data**.
 - Have a **small test error**.



Generalization

- ▶ **Generalization:** make a model that performs **well** on **test data**.
 - Have a **small test error**.
- ▶ **Challenges**
 1. Make the **training error small**.
 2. Make the **gap** between **training** and **test error small**.



More About The Test Error

- ▶ The **test error** is defined as the **expected value** of the **error** on test set.

$$\text{MSE} = \frac{1}{k} \sum_i^k (\hat{y}^{(i)} - y^{(i)})^2, \text{ k: the num. of instances in the test set}$$
$$= E[(\hat{y} - y)^2]$$



More About The Test Error

- The **test error** is defined as the **expected value** of the **error** on test set.

$$\begin{aligned} \text{MSE} &= \frac{1}{k} \sum_i^k (\hat{y}^{(i)} - y^{(i)})^2, \quad k: \text{the num. of instances in the test set} \\ &= E[(\hat{y} - y)^2] \end{aligned}$$

- A model's **test error** can be expressed as the **sum** of **bias** and **variance**.

$$E[(\hat{y} - y)^2] = \text{Bias}[\hat{y}, y]^2 + \text{Var}[\hat{y}] + \varepsilon^2$$





Bias and Underfitting

- ▶ Bias: the expected deviation from the true value of the function.

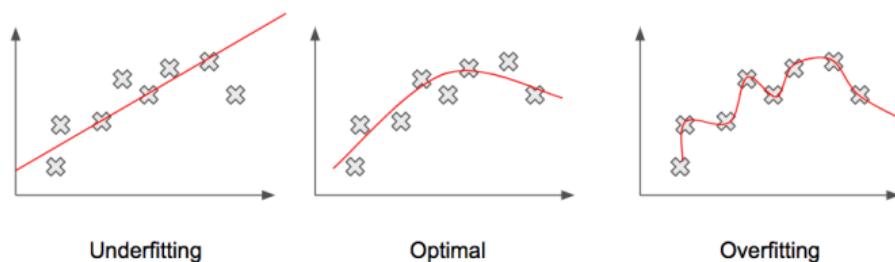
$$\text{Bias}[\hat{y}, y] = E[\hat{y}] - y$$

Bias and Underfitting

- ▶ Bias: the expected deviation from the true value of the function.

$$\text{Bias}[\hat{y}, y] = E[\hat{y}] - y$$

- ▶ A high-bias model is most likely to underfit the training data.
 - High error value on the training set.

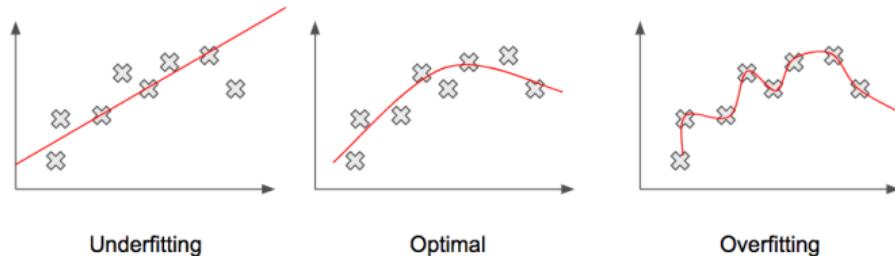


Bias and Underfitting

- ▶ Bias: the expected deviation from the true value of the function.

$$\text{Bias}[\hat{y}, y] = E[\hat{y}] - y$$

- ▶ A high-bias model is most likely to underfit the training data.
 - High error value on the training set.
- ▶ Underfitting happens when the model is too simple to learn the underlying structure of the data.





Variance and Overfitting

- ▶ **Variance:** how much a model changes if you train it on a different training set.

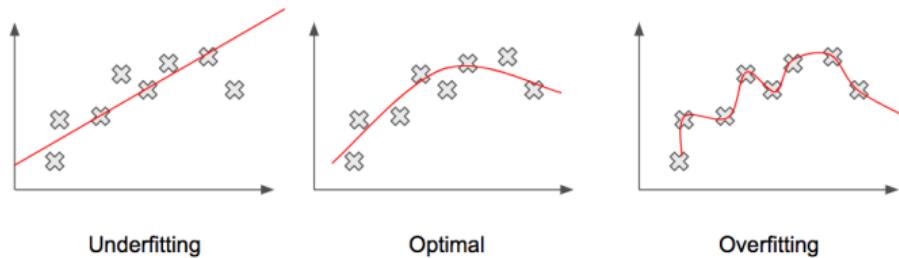
$$\text{Var}[\hat{y}] = E[(\hat{y} - E[\hat{y}])^2]$$

Variance and Overfitting

- ▶ **Variance**: how much a model changes if you train it on a different training set.

$$\text{Var}[\hat{y}] = E[(\hat{y} - E[\hat{y}])^2]$$

- ▶ A **high-variance** model is most likely to **overfit** the training data.
 - The **gap** between the **training error** and **test error** is **too large**.

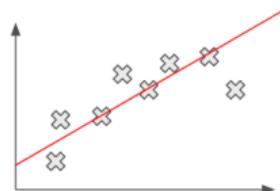


Variance and Overfitting

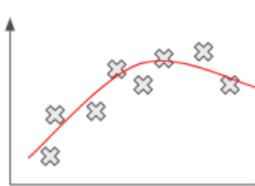
- ▶ **Variance**: how much a model changes if you train it on a different training set.

$$\text{Var}[\hat{y}] = E[(\hat{y} - E[\hat{y}])^2]$$

- ▶ A **high-variance** model is most likely to **overfit** the training data.
 - The **gap** between the **training error** and **test error** is **too large**.
- ▶ **Overfitting** happens when the **model is too complex** relative to the amount and noisiness of the training data.



Underfitting



Optimal



Overfitting



The Bias/Variance Tradeoff (1/2)

- ▶ Assume a model with two parameters w_0 (intercept) and w_1 (slope): $\hat{y} = w_0 + w_1 x$



The Bias/Variance Tradeoff (1/2)

- ▶ Assume a model with **two parameters** w_0 (**intercept**) and w_1 (**slope**): $\hat{y} = w_0 + w_1 x$
- ▶ They give the learning algorithm **two degrees of freedom**.



The Bias/Variance Tradeoff (1/2)

- ▶ Assume a model with two parameters w_0 (intercept) and w_1 (slope): $\hat{y} = w_0 + w_1x$
- ▶ They give the learning algorithm two degrees of freedom.
- ▶ We tweak both the w_0 and w_1 to adapt the model to the training data.

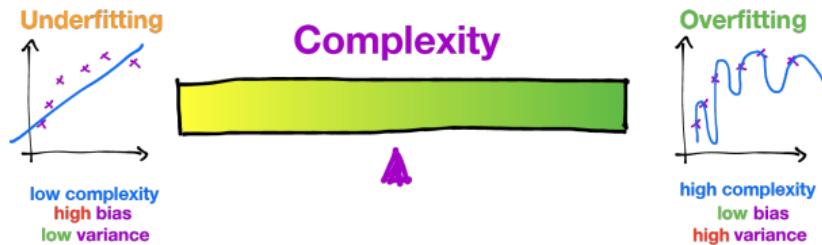


The Bias/Variance Tradeoff (1/2)

- ▶ Assume a model with two parameters w_0 (intercept) and w_1 (slope): $\hat{y} = w_0 + w_1x$
- ▶ They give the learning algorithm two degrees of freedom.
- ▶ We tweak both the w_0 and w_1 to adapt the model to the training data.
- ▶ If we forced $w_0 = 0$, the algorithm would have only one degree of freedom and would have a much harder time fitting the data properly.

The Bias/Variance Tradeoff (2/2)

- ▶ Increasing degrees of freedom will typically increase its variance and reduce its bias.
- ▶ Decreasing degrees of freedom increases its bias and reduces its variance.
- ▶ This is why it is called a **tradeoff**.



© Machine Learning @ Berkeley

[<https://ml.berkeley.edu/blog/2017/07/13/tutorial-4>]



Regularization (1/2)

- ▶ One way to reduce the risk of overfitting is to have fewer degrees of freedom.
- ▶ Regularization is a technique to reduce the risk of overfitting.
- ▶ For a linear model, regularization is achieved by constraining the weights of the model.

$$J(w) = \text{MSE}(w) + \lambda R(w)$$



Regularization (2/2)

- ▶ Lasso regression (1): $R(w) = \lambda \sum_{i=1}^n |w_i|$ is added to the cost function:

$$J(w) = \text{MSE}(w) + \lambda \sum_{i=1}^n |w_i|$$



Regularization (2/2)

- ▶ Lasso regression (1): $R(w) = \lambda \sum_{i=1}^n |w_i|$ is added to the cost function:

$$J(w) = \text{MSE}(w) + \lambda \sum_{i=1}^n |w_i|$$

- ▶ Ridge regression (2): $R(w) = \lambda \sum_{i=1}^n w_i^2$ is added to the cost function.

$$J(w) = \text{MSE}(w) + \lambda \sum_{i=1}^n w_i^2$$



Regularization (2/2)

- ▶ Lasso regression (/1): $R(w) = \lambda \sum_{i=1}^n |w_i|$ is added to the cost function:

$$J(w) = \text{MSE}(w) + \lambda \sum_{i=1}^n |w_i|$$

- ▶ Ridge regression (/2): $R(w) = \lambda \sum_{i=1}^n w_i^2$ is added to the cost function.

$$J(w) = \text{MSE}(w) + \lambda \sum_{i=1}^n w_i^2$$

- ▶ ElasticNet: a middle ground between /1 and /2 regularization.

$$J(w) = \text{MSE}(w) + \alpha \lambda \sum_{i=1}^n |w_i| + (1 - \alpha) \lambda \sum_{i=1}^n w_i^2$$



Regularization in Spark

$$J(w) = \text{MSE}(w) + \alpha \lambda \sum_{i=1}^n |w_i| + (1 - \alpha) \lambda \sum_{i=1}^n w_i^2$$

- ▶ If $\alpha = 0$: L_2 regularization
- ▶ If $\alpha = 1$: L_1 regularization
- ▶ For α in $(0, 1)$: a combination of L_1 and L_2 regularizations

```
import org.apache.spark.ml.regression.LinearRegression

val lr = new LinearRegression().setElasticNetParam(0.8)

val lrModel = lr.fit(data)
```



Hyperparameters



Hyperparameters and Validation Sets (1/2)

- ▶ Hyperparameters are settings that we can use to control the behavior of a learning algorithm.



Hyperparameters and Validation Sets (1/2)

- ▶ Hyperparameters are settings that we can use to control the behavior of a learning algorithm.
- ▶ The values of hyperparameters are not adapted by the learning algorithm itself.
 - E.g., the α and λ values for regularization.



Hyperparameters and Validation Sets (1/2)

- ▶ Hyperparameters are settings that we can use to control the behavior of a learning algorithm.
- ▶ The values of hyperparameters are not adapted by the learning algorithm itself.
 - E.g., the α and λ values for regularization.
- ▶ We do not learn the hyperparameter.
 - It is not appropriate to learn that hyperparameter on the training set.
 - If learned on the training set, such hyperparameters would always result in overfitting.



Hyperparameters and Validation Sets (2/2)

- ▶ To find **hyperparameters**, we need a **validation set** of examples that the **training algorithm does not observe**.



Hyperparameters and Validation Sets (2/2)

- ▶ To find **hyperparameters**, we need a **validation set** of examples that the **training algorithm does not observe**.
- ▶ We construct the **validation set** from the **training data** (**not the test data**).

Hyperparameters and Validation Sets (2/2)

- ▶ To find **hyperparameters**, we need a **validation set** of examples that the **training algorithm does not observe**.
- ▶ We construct the **validation set** from the **training data** (**not the test data**).
- ▶ We split the **training data** into two disjoint subsets:
 1. One is used to **learn the parameters**.
 2. The other one (the **validation set**) is used to **estimate the test error** during or after training, allowing for the **hyperparameters** to be updated accordingly.

Full Dataset:

Training Data	Validation Data	Test Data

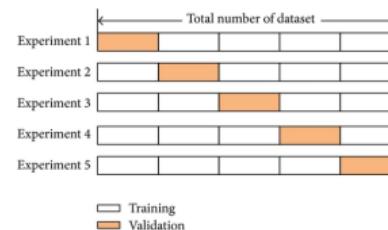
Cross-Validation

- ▶ **Cross-validation:** a technique to avoid **wasting too much training data in validation sets.**



Cross-Validation

- ▶ **Cross-validation:** a technique to avoid **wasting too much training data** in **validation sets**.
- ▶ The **training set** is split into **complementary subsets**.



Cross-Validation

- ▶ **Cross-validation:** a technique to avoid **wasting too much training data** in **validation sets**.
- ▶ The **training set** is split into **complementary subsets**.
- ▶ Each model is **trained** against a different **combination** of these subsets and **validated** against the **remaining parts**.



Cross-Validation

- ▶ **Cross-validation:** a technique to avoid **wasting too much training data** in **validation sets**.
- ▶ The **training set** is split into **complementary subsets**.
- ▶ Each model is **trained** against a different **combination** of these subsets and **validated** against the **remaining parts**.
- ▶ Once the model type and hyperparameters have been selected, a **final model** is trained using these hyperparameters on the **full training set**, and the test error is measured on the **test set**.





Hyperparameters and Cross-Validation in Spark (1/2)

- ▶ **CrossValidator** to optimize hyperparameters in algorithms and model selection.
- ▶ It requires the following items:
 - **Estimator**: algorithm or Pipeline to tune.
 - Set of **ParamMaps**: parameters to choose from (also called a **parameter grid**).
 - **Evaluator**: metric to measure **how well a fitted Model does on held-out test data**.



Hyperparameters and Cross-Validation in Spark (2/2)

```
// construct a grid of parameters to search over.  
// this grid has 2 x 2 = 4 parameter settings for CrossValidator to choose from.  
val paramGrid = new ParamGridBuilder()  
  .addGrid(lr.regParam, Array(0.1, 0.01))  
  .addGrid(lr.elasticNetParam, Array(0.0, 1.0))  
  .build()
```



Hyperparameters and Cross-Validation in Spark (2/2)

```
// construct a grid of parameters to search over.  
// this grid has 2 x 2 = 4 parameter settings for CrossValidator to choose from.  
val paramGrid = new ParamGridBuilder()  
  .addGrid(lr.regParam, Array(0.1, 0.01))  
  .addGrid(lr.elasticNetParam, Array(0.0, 1.0))  
  .build()
```

```
val lr = new LinearRegression()  
  
// num folds = 3 => (2 x 2) x 3 = 12 different models being trained  
val cv = new CrossValidator()  
  .setEstimator(lr)  
  .setEvaluator(new RegressionEvaluator())  
  .setEstimatorParamMaps(paramGrid)  
  .setNumFolds(3)  
  
val cvModel = cv.fit(trainDF)
```



Summary

Summary

- ▶ Linear regression model $\hat{y} = w^T x$
 - Learning parameters w
 - Cost function $J(w)$
 - Learn parameters: normal equation, gradient descent (batch, stochastic, mini-batch)
- ▶ Generalization
 - Overfitting vs. underfitting
 - Bias vs. variance
 - Regularization: Lasso regression, Ridge regression, ElasticNet
- ▶ Hyperparameters and cross-validation



Reference

- ▶ Ian Goodfellow et al., Deep Learning (Ch. 4, 5)
- ▶ Aurélien Géron, Hands-On Machine Learning (Ch. 2, 4)
- ▶ Matei Zaharia et al., Spark - The Definitive Guide (Ch. 27)

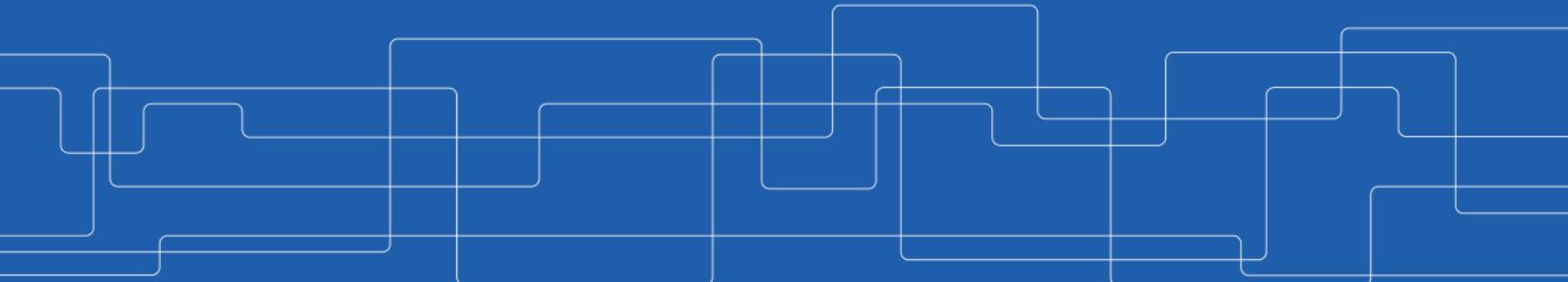


Questions?



Machine Learning - Classification

Amir H. Payberah
payberah@kth.se
2021-11-10



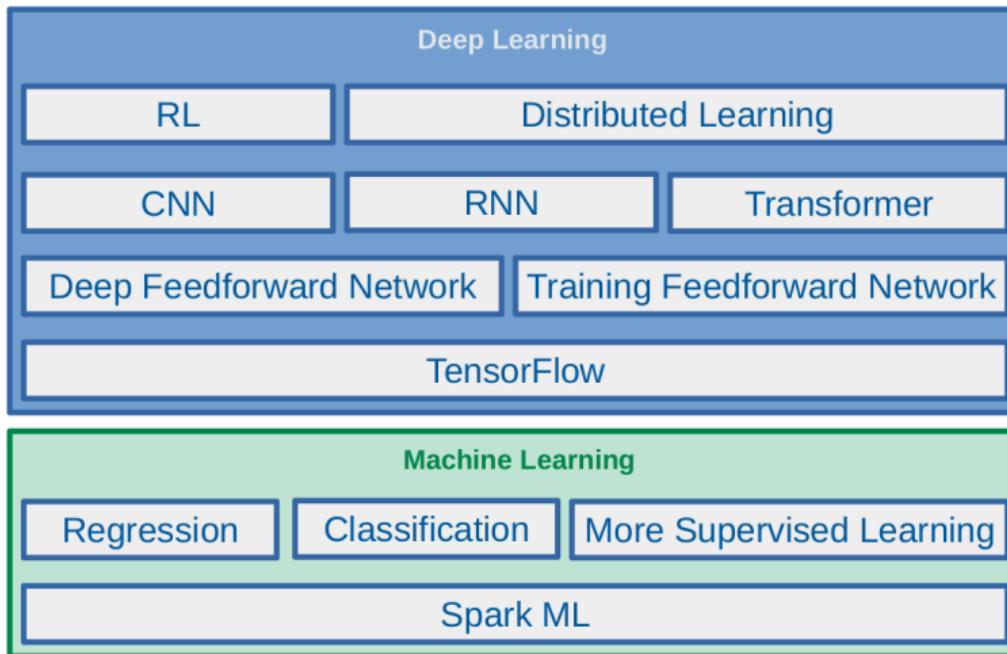


The Course Web Page

<https://id2223kth.github.io>
<https://tinyurl.com/6s5jy46a>

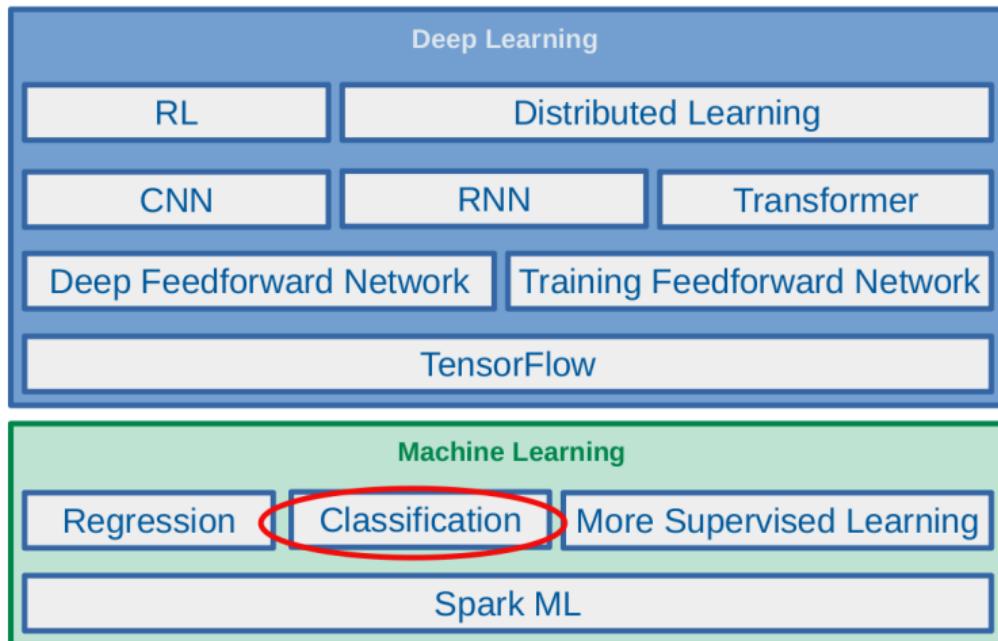


Where Are We?



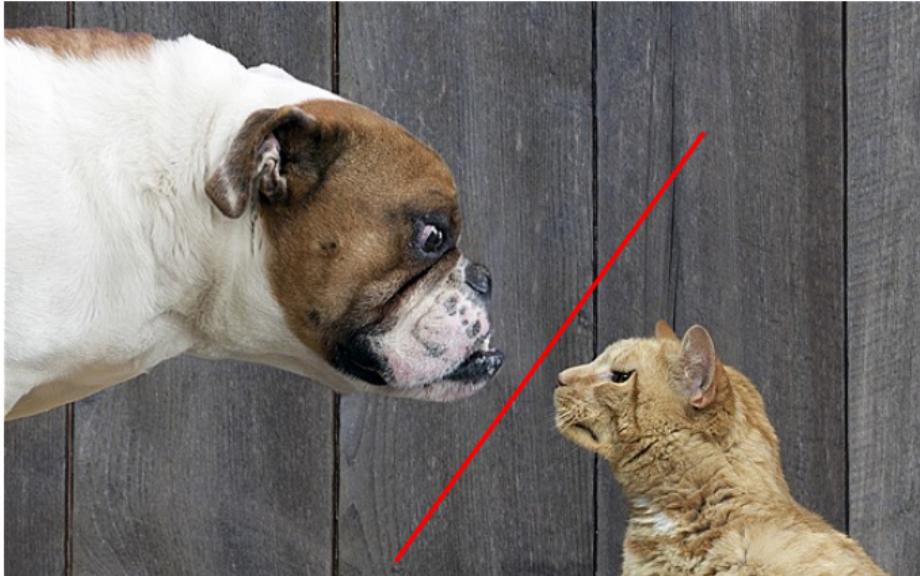


Where Are We?





Let's Start with an Example



[<https://www.telegraph.co.uk/lifestyle/pets/8151921/Dogs-are-smarter-than-cats-feline-friends-disagree.html>]

Example (1/4)

- ▶ Given the dataset of m cancer tests.

Tumor size	Cancer
330	1
120	0
400	1
:	:

Example (1/4)

- ▶ Given the dataset of m cancer tests.

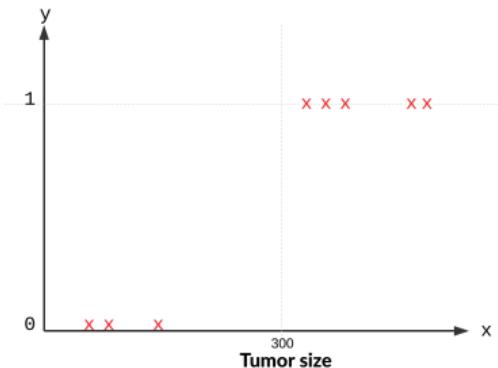
Tumor size	Cancer
330	1
120	0
400	1
:	:

- ▶ Predict the risk of cancer, as a function of the tumor size?

Example (2/4)

Tumor size	Cancer
330	1
120	0
400	1
:	:
:	:

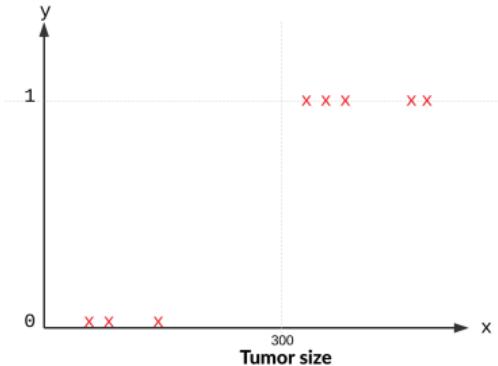
$$x = \begin{bmatrix} 330 \\ 120 \\ 400 \\ \vdots \\ \vdots \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ \vdots \end{bmatrix}$$



Example (2/4)

Tumor size	Cancer
330	1
120	0
400	1
:	:
:	:

$$x = \begin{bmatrix} 330 \\ 120 \\ 400 \\ \vdots \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

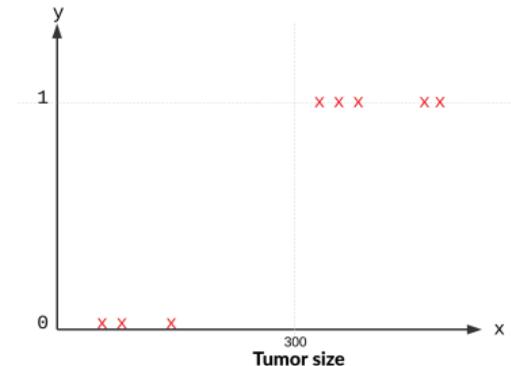


- $x^{(i)} \in \mathbb{R}$: $x_1^{(i)}$ is the tumor size of the i th instance in the training set.

Example (3/4)

Tumor size	Cancer
330	1
120	0
400	1
⋮	⋮

$$x = \begin{bmatrix} 330 \\ 120 \\ 400 \\ \vdots \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

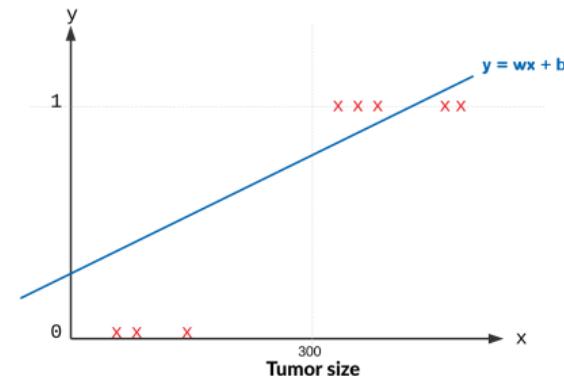


- ▶ Predict the risk of cancer \hat{y} as a function of the tumor sizes x_1 , i.e., $\hat{y} = f(x_1)$
- ▶ E.g., what is \hat{y} , if $x_1 = 500$?

Example (3/4)

Tumor size	Cancer
330	1
120	0
400	1
⋮	⋮

$$x = \begin{bmatrix} 330 \\ 120 \\ 400 \\ \vdots \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

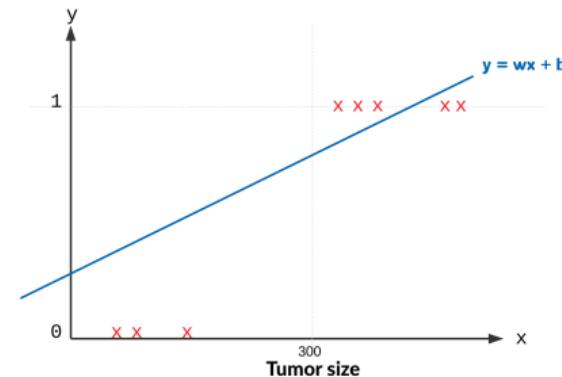


- ▶ Predict the risk of cancer \hat{y} as a function of the tumor sizes x_1 , i.e., $\hat{y} = f(x_1)$
- ▶ E.g., what is \hat{y} , if $x_1 = 500$?
- ▶ As an initial choice: $\hat{y} = f_w(x) = w_0 + w_1 x_1$

Example (3/4)

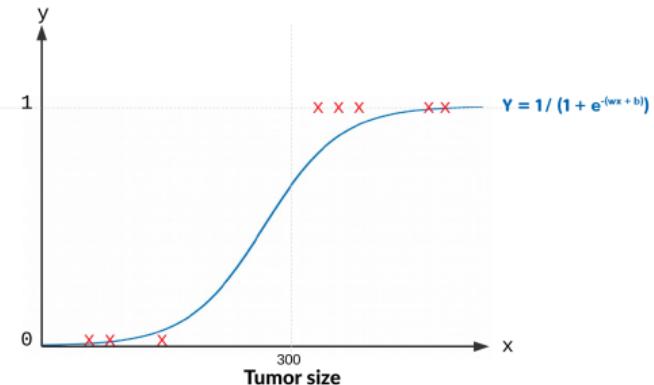
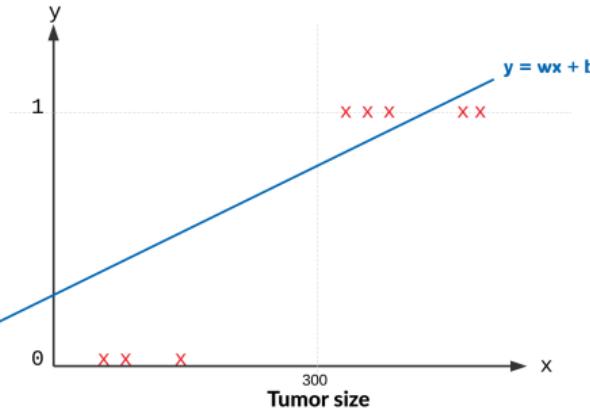
Tumor size	Cancer
330	1
120	0
400	1
⋮	⋮

$$x = \begin{bmatrix} 330 \\ 120 \\ 400 \\ \vdots \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$



- ▶ Predict the risk of cancer \hat{y} as a function of the tumor sizes x_1 , i.e., $\hat{y} = f(x_1)$
- ▶ E.g., what is \hat{y} , if $x_1 = 500$?
- ▶ As an initial choice: $\hat{y} = f_w(x) = w_0 + w_1 x_1$
- ▶ Bad model!

Example (4/4)

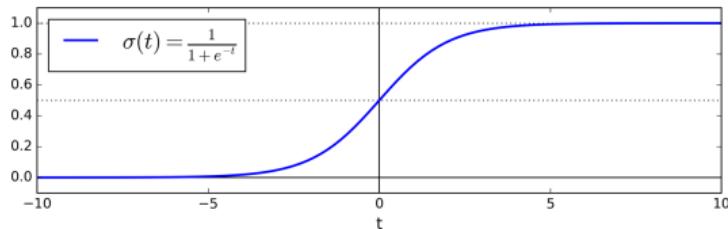


- ▶ A better model $\hat{y} = \frac{1}{1+e^{-(w_0+w_1x_1)}}$

Sigmoid Function

- The **sigmoid function**, denoted by $\sigma(\cdot)$, outputs a number **between 0 and 1**.

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



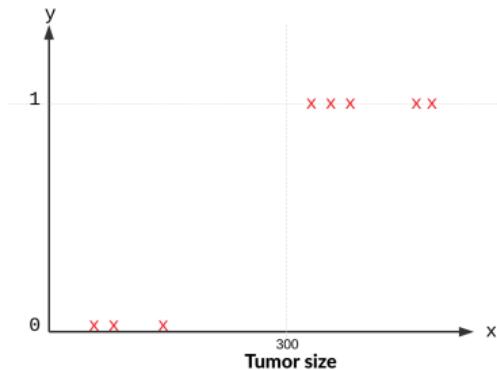
- When $t < 0$, then $\sigma(t) < 0.5$
- when $t \geq 0$, then $\sigma(t) \geq 0.5$



Binomial Logistic Regression

Binomial Logistic Regression (1/2)

- ▶ Our goal: to build a system that takes input $x \in \mathbb{R}^n$ and predicts output $\hat{y} \in \{0, 1\}$.
- ▶ To specify which of 2 categories an input x belongs to.



Binomial Logistic Regression (2/2)

- ▶ Linear regression: the model computes the weighted sum of the input features (plus a bias term).

$$\hat{y} = w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n = w^T x$$



Binomial Logistic Regression (2/2)

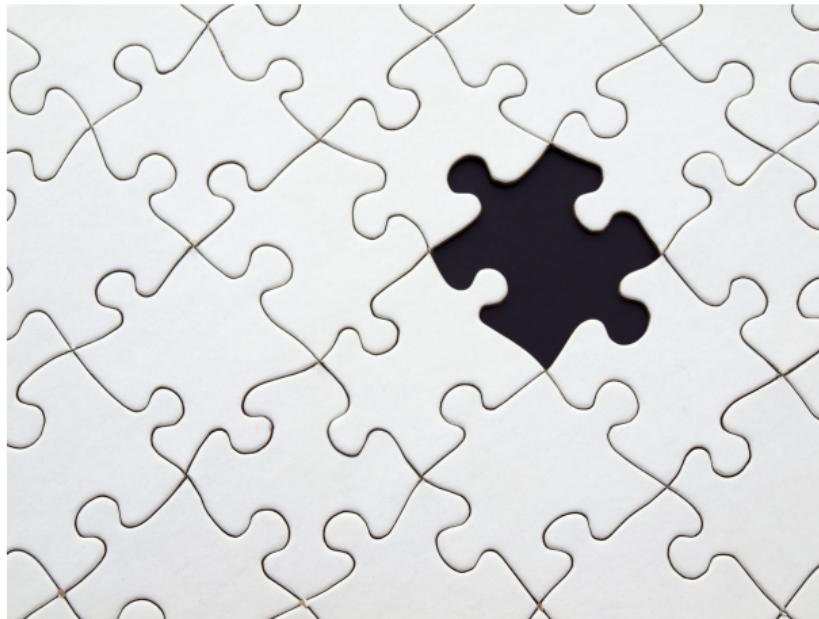
- ▶ **Linear regression:** the model computes the **weighted sum of the input features** (plus a bias term).

$$\hat{y} = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = w^T x$$

- ▶ **Binomial logistic regression:** the model computes a **weighted sum of the input features** (plus a bias term), but it **outputs the logistic of this result**.

$$z = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = w^T x$$

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



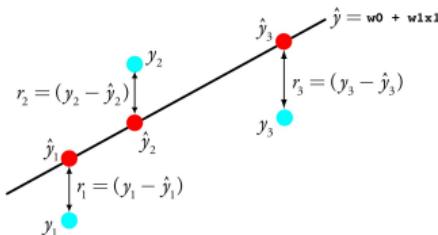
$$z = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = w^T x$$

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



How to Learn Model Parameters w ?

Linear Regression - Cost Function



- One reasonable model should make \hat{y} close to y , at least for the training dataset.
- Cost function $J(w)$: the mean squared error (MSE)

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

$$J(w) = \frac{1}{m} \sum_{i=1}^m \text{cost}(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$



Binomial Logistic Regression - Cost Function (1/5)

- ▶ Naive idea: minimizing the Mean Squared Error (MSE)

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

$$J(w) = \frac{1}{m} \sum_{i=1}^m \text{cost}(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$

Binomial Logistic Regression - Cost Function (1/5)

- Naive idea: minimizing the Mean Squared Error (MSE)

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

$$J(w) = \frac{1}{m} \sum_{i=1}^m \text{cost}(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$

$$J(w) = \text{MSE}(w) = \frac{1}{m} \sum_{i=1}^m \left(\frac{1}{1 + e^{-w^T x^{(i)}}} - y^{(i)} \right)^2$$



Binomial Logistic Regression - Cost Function (1/5)

- ▶ Naive idea: minimizing the Mean Squared Error (MSE)

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

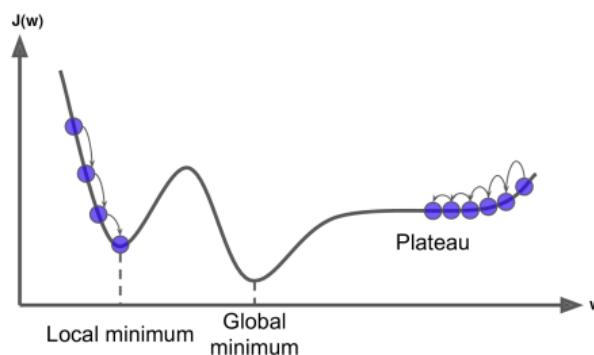
$$J(w) = \frac{1}{m} \sum_{i=1}^m \text{cost}(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$

$$J(w) = \text{MSE}(w) = \frac{1}{m} \sum_{i=1}^m \left(\frac{1}{1 + e^{-w^\top x^{(i)}}} - y^{(i)} \right)^2$$

- ▶ This cost function is a non-convex function for parameter optimization.

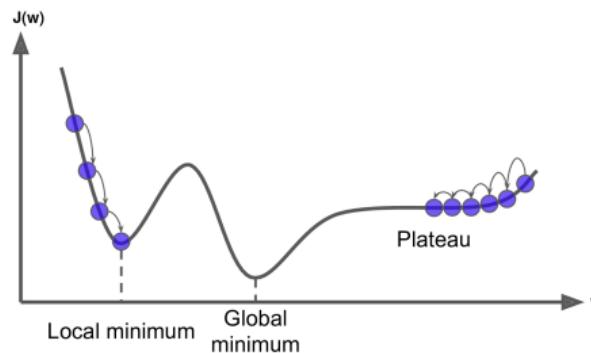
Binomial Logistic Regression - Cost Function (2/5)

- ▶ What do we mean by **non-convex**?
- ▶ If a line joining two points on the curve, **crosses the curve**.
- ▶ The algorithm may converge to a **local minimum**.



Binomial Logistic Regression - Cost Function (2/5)

- ▶ What do we mean by **non-convex**?
- ▶ If a line joining two points on the curve, **crosses the curve**.
- ▶ The algorithm may converge to a **local minimum**.
- ▶ We want a **convex** logistic regression **cost function $J(w)$** .





Binomial Logistic Regression - Cost Function (3/5)

- ▶ The predicted value $\hat{y} = \sigma(w^T x) = \frac{1}{1+e^{-w^T x}}$
- ▶ $\text{cost}(\hat{y}^{(i)}, y^{(i)}) = ?$



Binomial Logistic Regression - Cost Function (3/5)

- ▶ The predicted value $\hat{y} = \sigma(w^T x) = \frac{1}{1+e^{-w^T x}}$
- ▶ $\text{cost}(\hat{y}^{(i)}, y^{(i)}) = ?$
- ▶ The $\text{cost}(\hat{y}^{(i)}, y^{(i)})$ should be
 - Close to 0, if the predicted value \hat{y} will be close to true value y .
 - Large, if the predicted value \hat{y} will be far from the true value y .

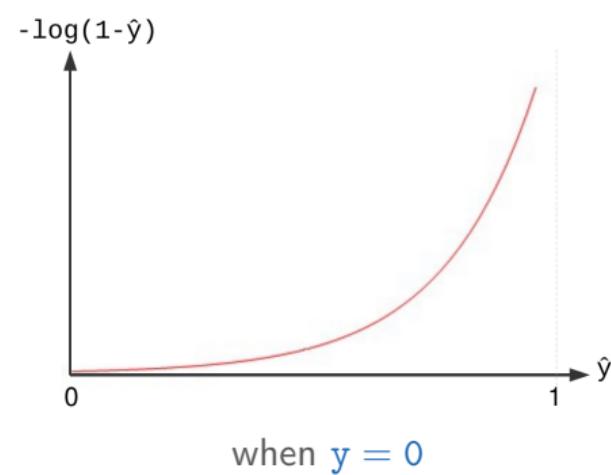
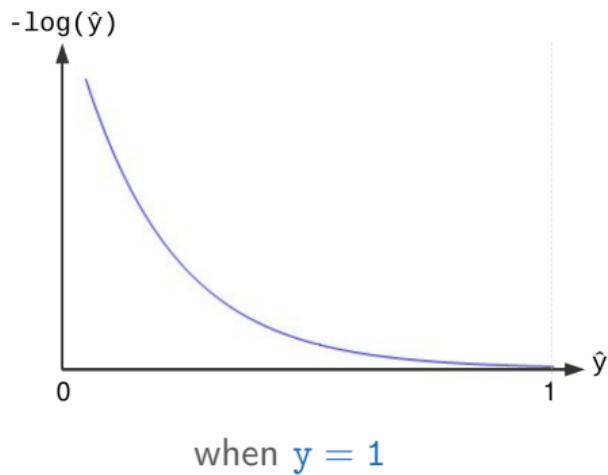
Binomial Logistic Regression - Cost Function (3/5)

- ▶ The predicted value $\hat{y} = \sigma(w^T x) = \frac{1}{1+e^{-w^T x}}$
- ▶ $\text{cost}(\hat{y}^{(i)}, y^{(i)}) = ?$
- ▶ The $\text{cost}(\hat{y}^{(i)}, y^{(i)})$ should be
 - Close to 0, if the predicted value \hat{y} will be close to true value y .
 - Large, if the predicted value \hat{y} will be far from the true value y .

$$\text{cost}(\hat{y}^{(i)}, y^{(i)}) = \begin{cases} -\log(\hat{y}^{(i)}) & \text{if } y^{(i)} = 1 \\ -\log(1 - \hat{y}^{(i)}) & \text{if } y^{(i)} = 0 \end{cases}$$

Binomial Logistic Regression - Cost Function (4/5)

$$\text{cost}(\hat{y}^{(i)}, y^{(i)}) = \begin{cases} -\log(\hat{y}^{(i)}) & \text{if } y^{(i)} = 1 \\ -\log(1 - \hat{y}^{(i)}) & \text{if } y^{(i)} = 0 \end{cases}$$





Binomial Logistic Regression - Cost Function (5/5)

- We can define $J(w)$ as below

$$\text{cost}(\hat{y}^{(i)}, y^{(i)}) = \begin{cases} -\log(\hat{y}^{(i)}) & \text{if } y^{(i)} = 1 \\ -\log(1 - \hat{y}^{(i)}) & \text{if } y^{(i)} = 0 \end{cases}$$

Binomial Logistic Regression - Cost Function (5/5)

- We can define $J(w)$ as below

$$\text{cost}(\hat{y}^{(i)}, y^{(i)}) = \begin{cases} -\log(\hat{y}^{(i)}) & \text{if } y^{(i)} = 1 \\ -\log(1 - \hat{y}^{(i)}) & \text{if } y^{(i)} = 0 \end{cases}$$

$$J(w) = \frac{1}{m} \sum_{i=1}^m \text{cost}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$$



How to Learn Model Parameters w ?

- ▶ We want to choose w so as to minimize $J(w)$.
- ▶ An approach to find w : gradient descent
 - Batch gradient descent
 - Stochastic gradient descent
 - Mini-batch gradient descent



Binomial Logistic Regression Gradient Descent (1/3)

- ▶ **Goal:** find w that minimizes $J(w) = -\frac{1}{m} \sum_i^m (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$.



Binomial Logistic Regression Gradient Descent (1/3)

- ▶ **Goal:** find w that minimizes $J(w) = -\frac{1}{m} \sum_i^m (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$.
- ▶ Start at a **random point**, and repeat the following **steps**, until the **stopping criterion** is satisfied:



Binomial Logistic Regression Gradient Descent (1/3)

- ▶ **Goal:** find w that minimizes $J(w) = -\frac{1}{m} \sum_i^m (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$.
- ▶ Start at a **random point**, and repeat the following **steps**, until the **stopping criterion** is satisfied:
 1. Determine a **descent direction** $\frac{\partial J(w)}{\partial w}$



Binomial Logistic Regression Gradient Descent (1/3)

- ▶ Goal: find w that minimizes $J(w) = -\frac{1}{m} \sum_i^m (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$.
- ▶ Start at a random point, and repeat the following steps, until the stopping criterion is satisfied:
 1. Determine a descent direction $\frac{\partial J(w)}{\partial w}$
 2. Choose a step size η



Binomial Logistic Regression Gradient Descent (1/3)

- ▶ Goal: find w that minimizes $J(w) = -\frac{1}{m} \sum_i^m (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$.
- ▶ Start at a random point, and repeat the following steps, until the stopping criterion is satisfied:
 1. Determine a descent direction $\frac{\partial J(w)}{\partial w}$
 2. Choose a step size η
 3. Update the parameters: $w^{(\text{next})} = w - \eta \frac{\partial J(w)}{\partial w}$ (simultaneously for all parameters)



Binomial Logistic Regression Gradient Descent (2/3)

- ▶ 1. Determine a **descent direction** $\frac{\partial J(w)}{\partial w}$.

$$\hat{y} = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

$$J(w) = \frac{1}{m} \sum_{i=1}^m \text{cost}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$$



Binomial Logistic Regression Gradient Descent (2/3)

- ▶ 1. Determine a **descent direction** $\frac{\partial J(w)}{\partial w}$.

$$\hat{y} = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

$$J(w) = \frac{1}{m} \sum_i^m \text{cost}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_i^m (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$$

$$\begin{aligned}\frac{\partial J(w)}{\partial w_j} &= \frac{1}{m} \sum_i^m -\left(y^{(i)} \frac{1}{\hat{y}^{(i)}} - (1 - y^{(i)}) \frac{1}{1 - \hat{y}^{(i)}}\right) \frac{\partial \hat{y}^{(i)}}{\partial w_j} \\ &= \frac{1}{m} \sum_i^m -\left(y^{(i)} \frac{1}{\hat{y}^{(i)}} - (1 - y^{(i)}) \frac{1}{1 - \hat{y}^{(i)}}\right) \hat{y}^{(i)} (1 - \hat{y}^{(i)}) \frac{\partial w^T x}{\partial w_j} \\ &= \frac{1}{m} \sum_i^m -(y^{(i)}(1 - \hat{y}^{(i)}) - (1 - y^{(i)})\hat{y}^{(i)})x_j \\ &= \frac{1}{m} \sum_i^m (\hat{y}^{(i)} - y^{(i)})x_j\end{aligned}$$



Binomial Logistic Regression Gradient Descent (3/3)

- ▶ 2. Choose a step size η
- ▶ 3. Update the parameters: $w_j^{(\text{next})} = w_j - \eta \frac{\partial J(w)}{\partial w_j}$
 - $0 \leq j \leq n$, where n is the number of features.

Binomial Logistic Regression Gradient Descent - Example (1/4)

Tumor size	Cancer
330	1
120	0
400	1

$$X = \begin{bmatrix} 1 & | & 330 \\ 1 & | & 120 \\ 1 & | & 400 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

- ▶ Predict the risk of cancer \hat{y} as a function of the tumor sizes x_1 .
- ▶ E.g., what is \hat{y} , if $x_1 = 500$?

Binomial Logistic Regression Gradient Descent - Example (2/4)

$$X = \left[\begin{array}{c|c} 1 & 330 \\ 1 & 120 \\ 1 & 400 \end{array} \right] \quad y = \left[\begin{array}{c} 1 \\ 0 \\ 1 \end{array} \right]$$

$$\hat{y} = \sigma(w_0 + w_1 x_1) = \frac{1}{1 + e^{-(w_0 + w_1 x_1)}}$$

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

Binomial Logistic Regression Gradient Descent - Example (2/4)

$$X = \left[\begin{array}{c|c} 1 & 330 \\ 1 & 120 \\ 1 & 400 \end{array} \right] \quad y = \left[\begin{array}{c} 1 \\ 0 \\ 1 \end{array} \right]$$

$$\hat{y} = \sigma(w_0 + w_1 x_1) = \frac{1}{1 + e^{-(w_0 + w_1 x_1)}}$$

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

$$\begin{aligned} \frac{\partial J(w)}{\partial w_0} &= \frac{1}{3} \sum_{i=1}^3 (\hat{y}^{(i)} - y^{(i)}) x_0 \\ &= \frac{1}{3} \left[\left(\frac{1}{1 + e^{-(w_0 + 330w_1)}} - 1 \right) + \left(\frac{1}{1 + e^{-(w_0 + 120w_1)}} - 0 \right) + \left(\frac{1}{1 + e^{-(w_0 + 400w_1)}} - 1 \right) \right] \end{aligned}$$

Binomial Logistic Regression Gradient Descent - Example (3/4)

$$X = \left[\begin{array}{c|c} 1 & 330 \\ 1 & 120 \\ 1 & 400 \end{array} \right] \quad y = \left[\begin{array}{c} 1 \\ 0 \\ 1 \end{array} \right]$$

$$\hat{y} = \sigma(w_0 + w_1 x_1) = \frac{1}{1 + e^{-(w_0 + w_1 x_1)}}$$

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

Binomial Logistic Regression Gradient Descent - Example (3/4)

$$X = \left[\begin{array}{c|c} 1 & 330 \\ 1 & 120 \\ 1 & 400 \end{array} \right] \quad y = \left[\begin{array}{c} 1 \\ 0 \\ 1 \end{array} \right]$$

$$\hat{y} = \sigma(w_0 + w_1 x_1) = \frac{1}{1 + e^{-(w_0 + w_1 x_1)}}$$

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

$$\begin{aligned} \frac{\partial J(w)}{\partial w_1} &= \frac{1}{3} \sum_{i=1}^3 (\hat{y}^{(i)} - y^{(i)}) x_1 \\ &= \frac{1}{3} [330 \left(\frac{1}{1 + e^{-(w_0 + 330w_1)}} - 1 \right) + 120 \left(\frac{1}{1 + e^{-(w_0 + 120w_1)}} - 0 \right) + 400 \left(\frac{1}{1 + e^{-(w_0 + 400w_1)}} - 1 \right)] \end{aligned}$$



Binomial Logistic Regression Gradient Descent - Example (4/4)

$$w_0^{(\text{next})} = w_0 - \eta \frac{\partial J(w)}{\partial w_0}$$

$$w_1^{(\text{next})} = w_1 - \eta \frac{\partial J(w)}{\partial w_1}$$



Binomial Logistic Regression in Spark

```
case class cancer(x1: Long, y: Long)

val trainData = spark.createDataFrame(Seq(cancer(330, 1), cancer(120, 0), cancer(400, 1))).toDF
val testData = spark.createDataFrame(Seq(cancer(500, 0))).toDF
```



Binomial Logistic Regression in Spark

```
case class cancer(x1: Long, y: Long)

val trainData = spark.createDataFrame(Seq(cancer(330, 1), cancer(120, 0), cancer(400, 1))).toDF
val testData = spark.createDataFrame(Seq(cancer(500, 0))).toDF

import org.apache.spark.ml.feature.VectorAssembler

val va = new VectorAssembler().setInputCols(Array("x1")).setOutputCol("features")

val train = va.transform(trainData)
val test = va.transform(testData)
```



Binomial Logistic Regression in Spark

```
case class cancer(x1: Long, y: Long)

val trainData = spark.createDataFrame(Seq(cancer(330, 1), cancer(120, 0), cancer(400, 1))).toDF
val testData = spark.createDataFrame(Seq(cancer(500, 0))).toDF

import org.apache.spark.ml.feature.VectorAssembler

val va = new VectorAssembler().setInputCols(Array("x1")).setOutputCol("features")

val train = va.transform(trainData)
val test = va.transform(testData)

import org.apache.spark.ml.classification.LogisticRegression

val lr = new LogisticRegression().setFeaturesCol("features").setLabelCol("y")
  .setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)

val lrModel = lr.fit(train)
lrModel.transform(test).show
```



Binomial Logistic Regression

Probabilistic Interpretation



Probability and Likelihood (1/2)

- ▶ Let $X : \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ be a discrete random variable drawn independently from a distribution probability p depending on a parameter θ .



Probability and Likelihood (1/2)

- ▶ Let $X : \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ be a discrete random variable drawn independently from a distribution probability p depending on a parameter θ .
 - For six tosses of a coin, $X : \{h, t, t, t, h, t\}$, h : head, and t : tail.
 - Suppose you have a coin with probability θ to land heads and $(1 - \theta)$ to land tails.



Probability and Likelihood (1/2)

- ▶ Let $X : \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ be a **discrete random variable** drawn independently from a **distribution probability p** depending on a **parameter θ** .
 - For six tosses of a coin, $X : \{h, t, t, t, h, t\}$, **h**: head, and **t**: tail.
 - Suppose you have a **coin** with probability θ to land heads and $(1 - \theta)$ to land tails.
- ▶ $p(X | \theta = \frac{2}{3})$ is the **probability** of X given $\theta = \frac{2}{3}$.



Probability and Likelihood (1/2)

- ▶ Let $X : \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ be a **discrete random variable** drawn independently from a **distribution probability p** depending on a **parameter θ** .
 - For six tosses of a coin, $X : \{h, t, t, t, h, t\}$, h : head, and t : tail.
 - Suppose you have a **coin** with probability θ to land heads and $(1 - \theta)$ to land tails.
- ▶ $p(X | \theta = \frac{2}{3})$ is the **probability** of X given $\theta = \frac{2}{3}$.
- ▶ $p(X = h | \theta)$ is the **likelihood** of θ given $X = h$.



Probability and Likelihood (1/2)

- ▶ Let $X : \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ be a **discrete random variable** drawn independently from a **distribution probability p** depending on a **parameter θ** .
 - For six tosses of a coin, $X : \{h, t, t, t, h, t\}$, **h**: head, and **t**: tail.
 - Suppose you have a **coin** with probability θ to land heads and $(1 - \theta)$ to land tails.
- ▶ $p(X | \theta = \frac{2}{3})$ is the **probability** of X given $\theta = \frac{2}{3}$.
- ▶ $p(X = h | \theta)$ is the **likelihood** of θ given $X = h$.
- ▶ **Likelihood (L)**: a function of the **parameters (θ)** of a probability model, given **specific observed data**, e.g., $X = h$.

$$L(\theta) = p(X | \theta)$$

Probability and Likelihood (2/2)

- If samples in \mathbf{X} are **independent** we have:

$$\begin{aligned} L(\theta) &= p(\mathbf{X} \mid \theta) = p(x^{(1)}, x^{(2)}, \dots, x^{(m)} \mid \theta) \\ &= p(x^{(1)} \mid \theta)p(x^{(2)} \mid \theta) \cdots p(x^{(m)} \mid \theta) = \prod_{i=1}^m p(x^{(i)} \mid \theta) \end{aligned}$$



Likelihood and Log-Likelihood

- ▶ The Likelihood product is prone to numerical underflow.

$$L(\theta) = p(x \mid \theta) = \prod_{i=1}^m p(x^{(i)} \mid \theta)$$



Likelihood and Log-Likelihood

- ▶ The Likelihood product is prone to numerical underflow.

$$L(\theta) = p(x \mid \theta) = \prod_{i=1}^m p(x^{(i)} \mid \theta)$$

- ▶ To overcome this problem we can use the logarithm of the likelihood.
 - Transforms a product into a sum.

$$\log(L(\theta)) = \log(p(x \mid \theta)) = \sum_{i=1}^m \log p(x^{(i)} \mid \theta)$$

Likelihood and Log-Likelihood

- ▶ The Likelihood product is prone to numerical underflow.

$$L(\theta) = p(x \mid \theta) = \prod_{i=1}^m p(x^{(i)} \mid \theta)$$

- ▶ To overcome this problem we can use the logarithm of the likelihood.
 - Transforms a product into a sum.

$$\log(L(\theta)) = \log(p(x \mid \theta)) = \sum_{i=1}^m \log p(x^{(i)} \mid \theta)$$

- ▶ Negative Log-Likelihood: $-\log L(\theta) = -\sum_{i=1}^m \log p(x^{(i)} \mid \theta)$

Binomial Logistic Regression and Log-Likelihood (1/2)

- ▶ Let's consider the value of $\hat{y}^{(i)}$ as the **probability**:

$$\begin{cases} p(y^{(i)} = 1 \mid x^{(i)}; w) = \hat{y}^{(i)} \\ p(y^{(i)} = 0 \mid x^{(i)}; w) = 1 - \hat{y}^{(i)} \end{cases} \Rightarrow p(y^{(i)} \mid x^{(i)}; w) = (\hat{y}^{(i)})^{y^{(i)}} (1 - \hat{y}^{(i)})^{(1-y^{(i)})}$$

Binomial Logistic Regression and Log-Likelihood (1/2)

- ▶ Let's consider the value of $\hat{y}^{(i)}$ as the **probability**:

$$\begin{cases} p(y^{(i)} = 1 \mid x^{(i)}; w) = \hat{y}^{(i)} \\ p(y^{(i)} = 0 \mid x^{(i)}; w) = 1 - \hat{y}^{(i)} \end{cases} \Rightarrow p(y^{(i)} \mid x^{(i)}; w) = (\hat{y}^{(i)})^{y^{(i)}} (1 - \hat{y}^{(i)})^{(1-y^{(i)})}$$

- ▶ So the **likelihood** is:

$$L(w) = p(y \mid x; w) = \prod_{i=1}^m p(y^{(i)} \mid x^{(i)}; w) = \prod_{i=1}^m (\hat{y}^{(i)})^{y^{(i)}} (1 - \hat{y}^{(i)})^{(1-y^{(i)})}$$

Binomial Logistic Regression and Log-Likelihood (1/2)

- ▶ Let's consider the value of $\hat{y}^{(i)}$ as the **probability**:

$$\begin{cases} p(y^{(i)} = 1 \mid x^{(i)}; w) = \hat{y}^{(i)} \\ p(y^{(i)} = 0 \mid x^{(i)}; w) = 1 - \hat{y}^{(i)} \end{cases} \Rightarrow p(y^{(i)} \mid x^{(i)}; w) = (\hat{y}^{(i)})^{y^{(i)}} (1 - \hat{y}^{(i)})^{(1-y^{(i)})}$$

- ▶ So the **likelihood** is:

$$L(w) = p(y \mid x; w) = \prod_{i=1}^m p(y^{(i)} \mid x^{(i)}; w) = \prod_{i=1}^m (\hat{y}^{(i)})^{y^{(i)}} (1 - \hat{y}^{(i)})^{(1-y^{(i)})}$$

- ▶ And the **negative log-likelihood**:

$$-\log(L(w)) = -\sum_{i=1}^m \log(p(y^{(i)} \mid x^{(i)}; w)) = -\sum_{i=1}^m y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

Binomial Logistic Regression and Log-Likelihood (2/2)

- ▶ The negative log-likelihood:

$$-\log(L(w)) = -\sum_{i=1}^m \log p(y^{(i)} | x^{(i)}; w) = -\sum_{i=1}^m y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

Binomial Logistic Regression and Log-Likelihood (2/2)

- ▶ The **negative log-likelihood**:

$$-\log(L(w)) = -\sum_{i=1}^m \log p(y^{(i)} | x^{(i)}; w) = -\sum_{i=1}^m y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

- ▶ This equation is the same as the **logistic regression cost function**.

$$J(w) = -\frac{1}{m} \sum_i (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$$

Binomial Logistic Regression and Log-Likelihood (2/2)

- ▶ The negative log-likelihood:

$$-\log(L(w)) = -\sum_{i=1}^m \log p(y^{(i)} | x^{(i)}; w) = -\sum_{i=1}^m y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

- ▶ This equation is the same as the logistic regression cost function.

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

- ▶ Minimize the negative log-likelihood to minimize the cost function $J(w)$.



Binomial Logistic Regression and Cross-Entropy (1/2)

- ▶ Negative log-likelihood is also called the **cross-entropy**
- ▶ **Cross-entropy**: quantify the **difference (error)** between **two probability distributions**.
- ▶ How close is the **predicted distribution** to the **true distribution**?

$$H(p, q) = - \sum_j p_j \log(q_j)$$

- ▶ Where **p** is the **true distribution**, and **q** is the **predicted distribution**.



Binomial Logistic Regression and Cross-Entropy (2/2)

$$H(p, q) = - \sum_j p_j \log(q_j)$$

Binomial Logistic Regression and Cross-Entropy (2/2)

$$H(p, q) = - \sum_j p_j \log(q_j)$$

- ▶ The **true probability** distribution: $p(y=1) = y$ and $p(y=0) = 1 - y$



Binomial Logistic Regression and Cross-Entropy (2/2)

$$H(p, q) = - \sum_j p_j \log(q_j)$$

- ▶ The **true probability** distribution: $p(y=1) = y$ and $p(y=0) = 1 - y$
- ▶ The **predicted probability** distribution: $q(y=1) = \hat{y}$ and $q(y=0) = 1 - \hat{y}$



Binomial Logistic Regression and Cross-Entropy (2/2)

$$H(p, q) = - \sum_j p_j \log(q_j)$$

- ▶ The **true probability** distribution: $p(y=1) = y$ and $p(y=0) = 1 - y$
- ▶ The **predicted probability** distribution: $q(y=1) = \hat{y}$ and $q(y=0) = 1 - \hat{y}$
- ▶ $p \in \{y, 1 - y\}$ and $q \in \{\hat{y}, 1 - \hat{y}\}$

Binomial Logistic Regression and Cross-Entropy (2/2)

$$H(p, q) = - \sum_j p_j \log(q_j)$$

- ▶ The **true probability** distribution: $p(y=1) = y$ and $p(y=0) = 1 - y$
- ▶ The **predicted probability** distribution: $q(y=1) = \hat{y}$ and $q(y=0) = 1 - \hat{y}$
- ▶ $p \in \{y, 1 - y\}$ and $q \in \{\hat{y}, 1 - \hat{y}\}$
- ▶ So, the **cross-entropy** of p and q is nothing but the **logistic cost function**.

$$H(p, q) = - \sum_j p_j \log(q_j) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})) = \text{cost}(y, \hat{y})$$

Binomial Logistic Regression and Cross-Entropy (2/2)

$$H(p, q) = - \sum_j p_j \log(q_j)$$

- ▶ The **true probability** distribution: $p(y=1) = y$ and $p(y=0) = 1 - y$
- ▶ The **predicted probability** distribution: $q(y=1) = \hat{y}$ and $q(y=0) = 1 - \hat{y}$
- ▶ $p \in \{y, 1 - y\}$ and $q \in \{\hat{y}, 1 - \hat{y}\}$
- ▶ So, the **cross-entropy** of p and q is nothing but the **logistic cost function**.

$$H(p, q) = - \sum_j p_j \log(q_j) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})) = \text{cost}(y, \hat{y})$$

$$J(w) = \frac{1}{m} \sum_i^m \text{cost}(y, \hat{y}) = \frac{1}{m} \sum_i^m H(p, q) = -\frac{1}{m} \sum_i^m (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$$

- ▶ Minimize the cross-entropy to minimize the cost function $J(w)$.



Multinomial Logistic Regression



Multinomial Logistic Regression

- ▶ Multinomial classifiers can distinguish between more than two classes.
- ▶ Instead of $y \in \{0, 1\}$, we have $y \in \{1, 2, \dots, k\}$.



Binomial vs. Multinomial Logistic Regression (1/2)

- ▶ In a **binomial classifier**, $y \in \{0, 1\}$, the **estimator** is $\hat{y} = p(y = 1 | x; w)$.
 - We find **one** set of parameters **w**.

$$w^T = [w_0, w_1, \dots, w_n]$$



Binomial vs. Multinomial Logistic Regression (1/2)

- ▶ In a **binomial classifier**, $y \in \{0, 1\}$, the **estimator** is $\hat{y} = p(y = 1 | x; w)$.
 - We find **one** set of parameters w .

$$w^T = [w_0, w_1, \dots, w_n]$$

- ▶ In **multinomial classifier**, $y \in \{1, 2, \dots, k\}$, we need to estimate the result for each **individual label**, i.e., $\hat{y}_j = p(y = j | x; w)$.

Binomial vs. Multinomial Logistic Regression (1/2)

- ▶ In a **binomial classifier**, $y \in \{0, 1\}$, the **estimator** is $\hat{y} = p(y = 1 | x; w)$.
 - We find **one** set of parameters w .

$$w^T = [w_0, w_1, \dots, w_n]$$

- ▶ In **multinomial classifier**, $y \in \{1, 2, \dots, k\}$, we need to estimate the result for each **individual label**, i.e., $\hat{y}_j = p(y = j | x; w)$.
 - We find **k** set of parameters W .

$$W = \begin{bmatrix} [w_{0,1}, w_{1,1}, \dots, w_{n,1}] \\ [w_{0,2}, w_{1,2}, \dots, w_{n,2}] \\ \vdots \\ [w_{0,k}, w_{1,k}, \dots, w_{n,k}] \end{bmatrix} = \begin{bmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_k^T \end{bmatrix}$$

Binomial vs. Multinomial Logistic Regression (2/2)

- ▶ In a **binary class**, $y \in \{0, 1\}$, we use the **sigmoid** function.

$$w^T x = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n$$

$$\hat{y} = p(y = 1 \mid x; w) = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

Binomial vs. Multinomial Logistic Regression (2/2)

- ▶ In a **binary class**, $y \in \{0, 1\}$, we use the **sigmoid** function.

$$w^T x = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n$$

$$\hat{y} = p(y = 1 \mid x; w) = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

- ▶ In **multiclasses**, $y \in \{1, 2, \dots, k\}$, we use the **softmax** function.

$$w_j^T x = w_{0,j} x_0 + w_{1,j} x_1 + \cdots + w_{n,j} x_n, 1 \leq j \leq k$$

$$\hat{y}_j = p(y = j \mid x; w_j) = \sigma(w_j^T x) = \frac{e^{w_j^T x}}{\sum_{i=1}^k e^{w_i^T x}}$$

Sigmoid vs. Softmax

- ▶ Sigmoid function: $\sigma(w^T x) = \frac{1}{1+e^{-w^T x}}$
- ▶ Softmax function: $\sigma(w_j^T x) = \frac{e^{w_j^T x}}{\sum_{i=1}^k e^{w_i^T x}}$
 - Calculate the probabilities of each target class over all possible target classes.
 - The softmax function for two classes is equivalent the sigmoid function.





How Does Softmax Work? - Step 1

- ▶ For each instance $x^{(i)}$, computes the score $w_j^T x^{(i)}$ for each class j .

$$w_j^T x^{(i)} = w_{0,j} x_0^{(i)} + w_{1,j} x_1^{(i)} + \cdots + w_{n_j} x_n^{(i)}$$

- ▶ Note that each class j has its own dedicated parameter vector w_j .

$$W = \begin{bmatrix} [w_{0,1}, w_{1,1}, \dots, w_{n,1}] \\ [w_{0,2}, w_{1,2}, \dots, w_{n,2}] \\ \vdots \\ [w_{0,k}, w_{1,k}, \dots, w_{n,k}] \end{bmatrix} = \begin{bmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_k^T \end{bmatrix}$$



How Does Softmax Work? - Step 2

- ▶ For each instance $x^{(i)}$, apply the softmax function on its scores: $w_1^T x^{(i)}, \dots, w_k^T x^{(i)}$
- ▶ Estimates the probability that the instance $x^{(i)}$ belongs to class j .

$$\hat{y}_j^{(i)} = p(y^{(i)} = j \mid x^{(i)}; w_j) = \sigma(w_j^T x^{(i)}) = \frac{e^{w_j^T x^{(i)}}}{\sum_{l=1}^k e^{w_l^T x^{(i)}}}$$

- ▶ k : the number of classes.
- ▶ $w_j^T x^{(i)}$: the scores of class j for the instance $x^{(i)}$.
- ▶ $\sigma(w_j^T x^{(i)})$: the estimated probability that $x^{(i)}$ belongs to class j .



How Does Softmax Work? - Step 3

- ▶ Predicts the class with the highest estimated probability.

Softmax Model Estimation and Prediction - Example (1/2)

- ▶ Assume we have a **training set** consisting of $m = 4$ instances from $k = 3$ **classes**.

$$x^{(1)} \rightarrow \text{class1}, y^{(1)\top} = [1 \ 0 \ 0]$$

$$x^{(2)} \rightarrow \text{class2}, y^{(2)\top} = [0 \ 1 \ 0]$$

$$x^{(3)} \rightarrow \text{class3}, y^{(3)\top} = [0 \ 0 \ 1]$$

$$x^{(4)} \rightarrow \text{class3}, y^{(4)\top} = [0 \ 0 \ 1]$$

$$Y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Softmax Model Estimation and Prediction - Example (1/2)

- ▶ Assume we have a **training set** consisting of $m = 4$ instances from $k = 3$ **classes**.

$$x^{(1)} \rightarrow \text{class1}, y^{(1)\top} = [1 \ 0 \ 0]$$

$$x^{(2)} \rightarrow \text{class2}, y^{(2)\top} = [0 \ 1 \ 0]$$

$$x^{(3)} \rightarrow \text{class3}, y^{(3)\top} = [0 \ 0 \ 1]$$

$$x^{(4)} \rightarrow \text{class3}, y^{(4)\top} = [0 \ 0 \ 1]$$

$$Y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

- ▶ Assume **training set** X and random parameters W are as below:

$$X = \left[\begin{array}{c|ccc} 1 & 0.1 & 0.5 \\ 1 & 1.1 & 2.3 \\ 1 & -1.1 & -2.3 \\ 1 & -1.5 & -2.5 \end{array} \right] \quad W = \begin{bmatrix} 0.01 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.3 \\ 0.1 & 0.2 & 0.3 \end{bmatrix}$$

Softmax Model Estimation and Prediction - Example (2/2)

- Now, let's compute the softmax activation:

$$\hat{y}_j^{(i)} = p(y^{(i)} = j \mid x^{(i)}; w_j) = \sigma(w_j^T x^{(i)}) = \frac{e^{w_j^T x^{(i)}}}{\sum_{l=1}^k e^{w_l^T x^{(i)}}}$$

$$\hat{Y} = \begin{bmatrix} \hat{y}^{(1)\top} \\ \hat{y}^{(2)\top} \\ \hat{y}^{(3)\top} \\ \hat{y}^{(4)\top} \end{bmatrix} = \begin{bmatrix} 0.29 & 0.34 & 0.36 \\ 0.21 & 0.33 & 0.46 \\ 0.43 & 0.33 & 0.24 \\ 0.45 & 0.33 & 0.22 \end{bmatrix} \quad \text{the predicted classes} = \begin{bmatrix} 3 \\ 3 \\ 1 \\ 1 \end{bmatrix} \quad \text{The correct classes} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 3 \end{bmatrix}$$

Softmax Model Estimation and Prediction - Example (2/2)

- Now, let's compute the softmax activation:

$$\hat{y}_j^{(i)} = p(y^{(i)} = j \mid x^{(i)}; w_j) = \sigma(w_j^T x^{(i)}) = \frac{e^{w_j^T x^{(i)}}}{\sum_{l=1}^k e^{w_l^T x^{(i)}}}$$

$$\hat{Y} = \begin{bmatrix} \hat{y}^{(1)\top} \\ \hat{y}^{(2)\top} \\ \hat{y}^{(3)\top} \\ \hat{y}^{(4)\top} \end{bmatrix} = \begin{bmatrix} 0.29 & 0.34 & 0.36 \\ 0.21 & 0.33 & 0.46 \\ 0.43 & 0.33 & 0.24 \\ 0.45 & 0.33 & 0.22 \end{bmatrix} \quad \text{the predicted classes} = \begin{bmatrix} 3 \\ 3 \\ 1 \\ 1 \end{bmatrix} \quad \text{The correct classes} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 3 \end{bmatrix}$$

- They are terribly wrong.

Softmax Model Estimation and Prediction - Example (2/2)

- Now, let's compute the softmax activation:

$$\hat{y}_j^{(i)} = p(y^{(i)} = j \mid x^{(i)}; w_j) = \sigma(w_j^T x^{(i)}) = \frac{e^{w_j^T x^{(i)}}}{\sum_{l=1}^k e^{w_l^T x^{(i)}}}$$

$$\hat{Y} = \begin{bmatrix} \hat{y}^{(1)\top} \\ \hat{y}^{(2)\top} \\ \hat{y}^{(3)\top} \\ \hat{y}^{(4)\top} \end{bmatrix} = \begin{bmatrix} 0.29 & 0.34 & 0.36 \\ 0.21 & 0.33 & 0.46 \\ 0.43 & 0.33 & 0.24 \\ 0.45 & 0.33 & 0.22 \end{bmatrix} \quad \text{the predicted classes} = \begin{bmatrix} 3 \\ 3 \\ 1 \\ 1 \end{bmatrix} \quad \text{The correct classes} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 3 \end{bmatrix}$$

- They are **terribly wrong**.
- We need to **update the weights** based on the cost function.

Softmax Model Estimation and Prediction - Example (2/2)

- Now, let's compute the softmax activation:

$$\hat{y}_j^{(i)} = p(y^{(i)} = j \mid x^{(i)}; w_j) = \sigma(w_j^T x^{(i)}) = \frac{e^{w_j^T x^{(i)}}}{\sum_{l=1}^k e^{w_l^T x^{(i)}}}$$

$$\hat{Y} = \begin{bmatrix} \hat{y}^{(1)\top} \\ \hat{y}^{(2)\top} \\ \hat{y}^{(3)\top} \\ \hat{y}^{(4)\top} \end{bmatrix} = \begin{bmatrix} 0.29 & 0.34 & 0.36 \\ 0.21 & 0.33 & 0.46 \\ 0.43 & 0.33 & 0.24 \\ 0.45 & 0.33 & 0.22 \end{bmatrix} \quad \text{the predicted classes} = \begin{bmatrix} 3 \\ 3 \\ 1 \\ 1 \end{bmatrix} \quad \text{The correct classes} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 3 \end{bmatrix}$$

- They are **terribly wrong**.
- We need to **update the weights** based on the cost function.
- What is the cost function?**



Multinomial Logistic Regression - Cost Function (1/2)

- ▶ The **objective** is to have a model that estimates a **high probability** for the target class, and consequently a **low probability** for the other classes.

Multinomial Logistic Regression - Cost Function (1/2)

- ▶ The **objective** is to have a model that estimates a **high probability** for the target class, and consequently a **low probability** for the other classes.
- ▶ **Cost function:** the **cross-entropy** between the **correct classes** and **predicted class** for all classes.

$$J(\mathbf{w}_j) = -\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^k y_j^{(i)} \log(\hat{y}_j^{(i)})$$

Multinomial Logistic Regression - Cost Function (1/2)

- ▶ The **objective** is to have a model that estimates a **high probability** for the target class, and consequently a **low probability** for the other classes.
- ▶ **Cost function:** the **cross-entropy** between the **correct classes** and **predicted class** for all classes.

$$J(\mathbf{w}_j) = -\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^k y_j^{(i)} \log(\hat{y}_j^{(i)})$$

- ▶ $y_j^{(i)}$ is 1 if the target class for the i th instance is j , otherwise, it is 0.

Multinomial Logistic Regression - Cost Function (2/2)

$$J(\mathbf{w}_j) = -\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^k y_j^{(i)} \log(\hat{y}_j^{(i)})$$

- ▶ $y_j^{(i)}$ is 1 if the target class for the i th instance is j , otherwise, it is 0.

Multinomial Logistic Regression - Cost Function (2/2)

$$J(w_j) = -\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^k y_j^{(i)} \log(\hat{y}_j^{(i)})$$

- ▶ $y_j^{(i)}$ is 1 if the target class for the i th instance is j , otherwise, it is 0.
- ▶ If there are two classes ($k = 2$), this cost function is equivalent to the logistic regression's cost function.

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



How to Learn Model Parameters W ?

- ▶ Goal: find W that minimizes $J(W)$.



How to Learn Model Parameters W ?

- ▶ Goal: find W that minimizes $J(W)$.
- ▶ Start at a random point, and repeat the following steps, until the stopping criterion is satisfied:



How to Learn Model Parameters W ?

- ▶ Goal: find W that minimizes $J(W)$.
- ▶ Start at a random point, and repeat the following steps, until the stopping criterion is satisfied:
 1. Determine a descent direction $\frac{\partial J(W)}{\partial w}$



How to Learn Model Parameters W ?

- ▶ Goal: find W that minimizes $J(W)$.
- ▶ Start at a random point, and repeat the following steps, until the stopping criterion is satisfied:
 1. Determine a descent direction $\frac{\partial J(W)}{\partial w}$
 2. Choose a step size η



How to Learn Model Parameters W ?

- ▶ Goal: find W that minimizes $J(W)$.
- ▶ Start at a random point, and repeat the following steps, until the stopping criterion is satisfied:
 1. Determine a descent direction $\frac{\partial J(W)}{\partial w}$
 2. Choose a step size η
 3. Update the parameters: $w^{(\text{next})} = w - \eta \frac{\partial J(W)}{\partial w}$ (simultaneously for all parameters)



Multinomial Logistic Regression in Spark

```
val training = spark.read.format("libsvm").load("multiclass_data.txt")
```



Multinomial Logistic Regression in Spark

```
val training = spark.read.format("libsvm").load("multiclass_data.txt")
```

```
import org.apache.spark.ml.classification.LogisticRegression  
  
val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)  
val lrModel = lr.fit(training)
```



Multinomial Logistic Regression in Spark

```
val training = spark.read.format("libsvm").load("multiclass_data.txt")
```

```
import org.apache.spark.ml.classification.LogisticRegression  
  
val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)  
val lrModel = lr.fit(training)
```

```
println(s"Coefficients: \n${lrModel.coefficientMatrix}")  
println(s"Intercepts: \n${lrModel.interceptVector}")
```



Performance Measures



Copyright © 2012, ReadyToManage

[<http://blog.readytomanage.com/performance-management-cartoon>]



Performance Measures

- ▶ Evaluate the performance of a model.
- ▶ Depends on the application and its requirements.
- ▶ There are many different types of classification algorithms, but the evaluation of them share similar principles.

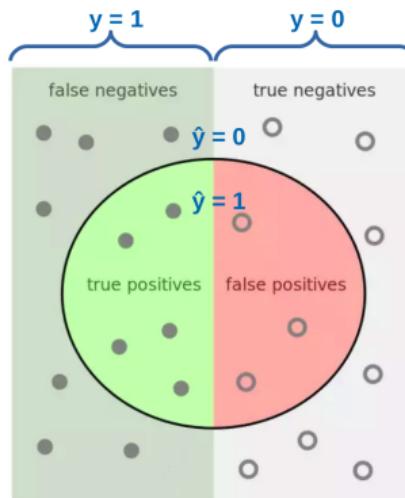


Evaluation of Classification Models (1/3)

- ▶ In a **classification problem**, there exists a **true output y** and a **model-generated predicted output \hat{y}** for each data point.
- ▶ The results for each instance point can be assigned to one of **four categories**:
 - **True Positive (TP)**
 - **True Negative (TN)**
 - **False Positive (FP)**
 - **False Negative (FN)**

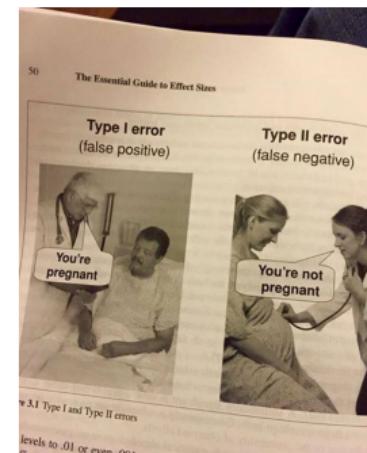
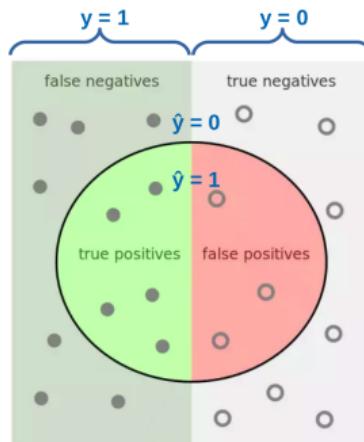
Evaluation of Classification Models (2/3)

- ▶ True Positive (TP): the label y is positive and prediction \hat{y} is also positive.
- ▶ True Negative (TN): the label y is negative and prediction \hat{y} is also negative.



Evaluation of Classification Models (3/3)

- ▶ False Positive (FP): the label y is negative but prediction \hat{y} is positive (type I error).
- ▶ False Negative (FN): the label y is positive but prediction \hat{y} is negative (type II error).





Why Pure Accuracy Is Not A Good Metric?

- ▶ **Accuracy**: how **close** the **prediction** is to the **true value**.



Why Pure Accuracy Is Not A Good Metric?

- ▶ **Accuracy**: how **close** the **prediction** is to the **true value**.
- ▶ Assume a highly **unbalanced dataset**
- ▶ E.g., a dataset where **95%** of the data points are **not fraud** and **5%** of the data points are **fraud**.



Why Pure Accuracy Is Not A Good Metric?

- ▶ **Accuracy**: how **close** the **prediction** is to the **true value**.
- ▶ Assume a highly **unbalanced dataset**
- ▶ E.g., a dataset where **95%** of the data points are **not fraud** and **5%** of the data points are **fraud**.
- ▶ A **naive classifier** that **predicts not fraud**, regardless of input, will be **95% accurate**.



Why Pure Accuracy Is Not A Good Metric?

- ▶ **Accuracy**: how **close** the **prediction** is to the **true value**.
- ▶ Assume a highly **unbalanced dataset**
- ▶ E.g., a dataset where **95%** of the data points are **not fraud** and **5%** of the data points are **fraud**.
- ▶ A **naive classifier** that **predicts not fraud**, regardless of input, will be **95% accurate**.
- ▶ For this reason, metrics like **precision** and **recall** are typically used.

Precision

- ▶ It is the **accuracy** of the **positive predictions**.

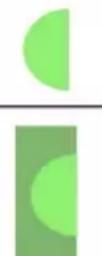
$$\text{Precision} = p(y = 1 \mid \hat{y} = 1) = \frac{\text{TP}}{\text{TP} + \text{FP}}$$



Recall

- ▶ Is the **ratio** of positive instances that are correctly detected by the classifier.
- ▶ Also called **sensitivity** or **true positive rate (TPR)**.

$$\text{Recall} = p(\hat{y} = 1 \mid y = 1) = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{Recall} = \frac{\text{Green Circle}}{\text{Green Square}}$$




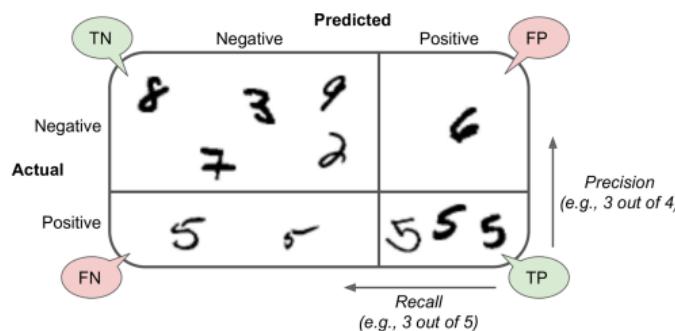
F1 Score

- ▶ *F1 score*: combine precision and recall into a single metric.
- ▶ The *F1 score* is the harmonic mean of precision and recall.
- ▶ Whereas the regular mean treats all values equally, the harmonic mean gives much more weight to low values.
- ▶ *F1* only gets high score if both recall and precision are high.

$$F1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

Confusion Matrix

- ▶ The **confusion matrix** is $K \times K$, where K is the **number of classes**.
- ▶ It shows the **number of correct and incorrect predictions** made by the classification model **compared to the actual outcomes** in the data.



Confusion Matrix - Example

		Predicted		
		Negative		Positive
		TN	FP	
Actual	Negative	8	3	9
	Positive	7	2	6
		5	5	5
		Precision (e.g., 3 out of 4)		TP
		Recall (e.g., 3 out of 5)		FN

$$TP = 3, TN = 5, FP = 1, FN = 2$$

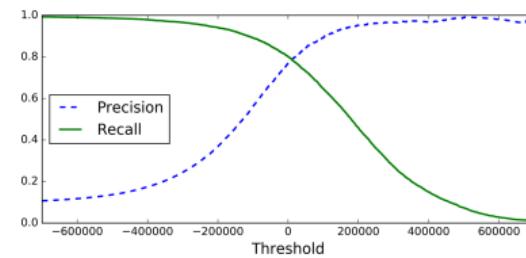
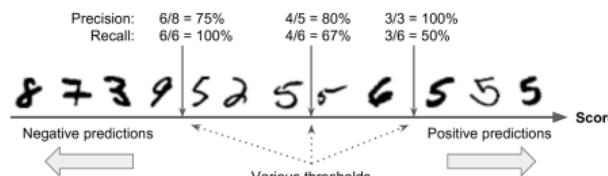
$$\text{Precision} = \frac{TP}{TP + FP} = \frac{3}{3 + 1} = \frac{3}{4}$$

$$\text{Recall (TPR)} = \frac{TP}{TP + FN} = \frac{3}{3 + 2} = \frac{3}{5}$$

$$\text{FPR} = \frac{FP}{TN + FP} = \frac{1}{5 + 1} = \frac{5}{6}$$

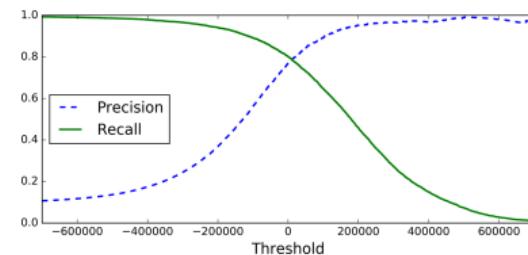
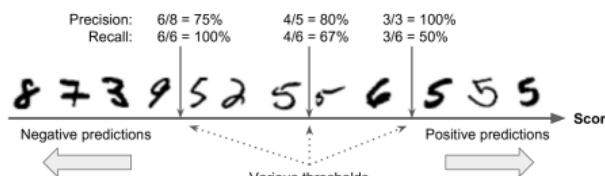
Precision-Recall Tradeoff

- ▶ Precision-recall tradeoff: increasing precision **reduces** recall, and vice versa.



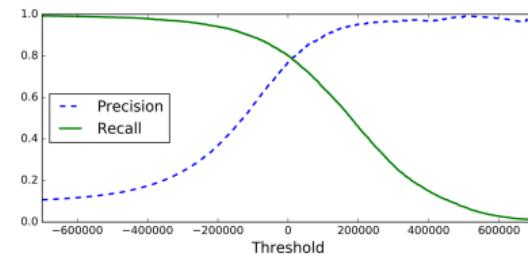
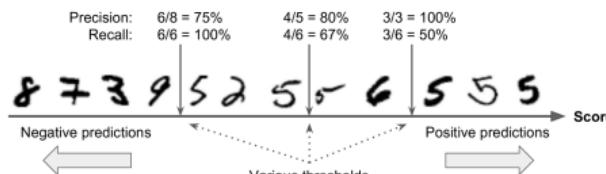
Precision-Recall Tradeoff

- ▶ Precision-recall tradeoff: increasing precision **reduces** recall, and vice versa.
- ▶ Assume a classifier that **detects number 5** from the other digits.
 - If an instance score is **greater than a threshold**, it assigns it to the **positive class**, otherwise to the **negative class**.



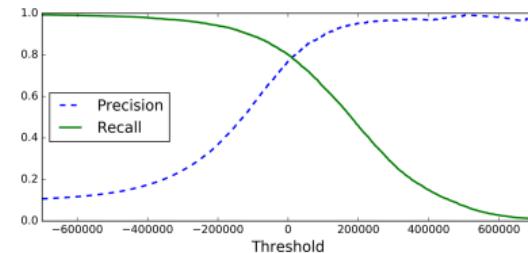
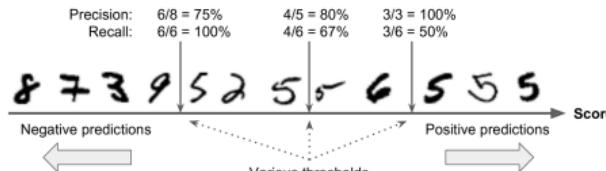
Precision-Recall Tradeoff

- ▶ Precision-recall tradeoff: increasing precision **reduces** recall, and vice versa.
- ▶ Assume a classifier that **detects number 5** from the other digits.
 - If an instance score is **greater than a threshold**, it assigns it to the **positive class**, otherwise to the **negative class**.
- ▶ Raising the threshold (move it to the arrow on the right), the **false positive** (the 6) becomes a **true negative**, thereby **increasing precision**.



Precision-Recall Tradeoff

- ▶ Precision-recall tradeoff: increasing precision **reduces** recall, and vice versa.
- ▶ Assume a classifier that **detects number 5** from the other digits.
 - If an instance score is **greater than a threshold**, it assigns it to the **positive class**, otherwise to the **negative class**.
- ▶ Raising the threshold (move it to the arrow on the right), the **false positive** (the 6) becomes a **true negative**, thereby **increasing precision**.
- ▶ Lowering the threshold **increases recall** and **reduces precision**.

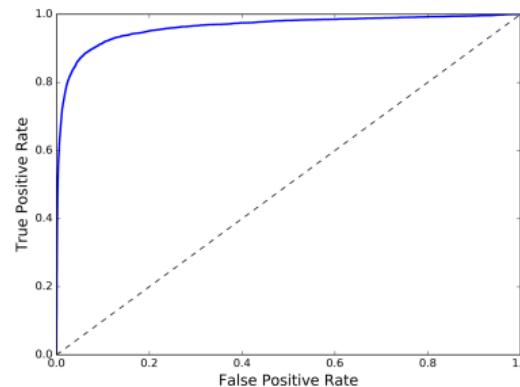


The ROC Curve (1/2)

- ▶ True positive rate (TPR) (recall): $p(\hat{y} = 1 \mid y = 1)$
- ▶ False positive rate (FPR): $p(\hat{y} = 1 \mid y = 0)$

$$\text{Recall} = \frac{\text{Green}}{\text{Total}}$$

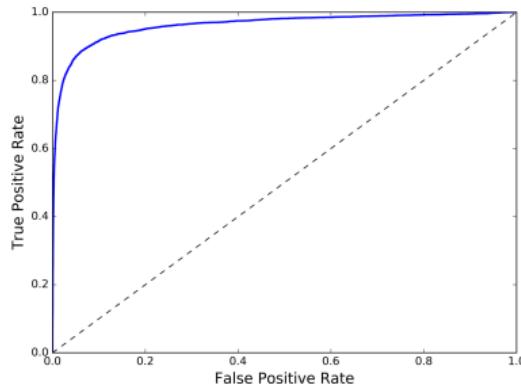
$$\text{FPR} = \frac{\text{Red}}{\text{Total}}$$



The ROC Curve (1/2)

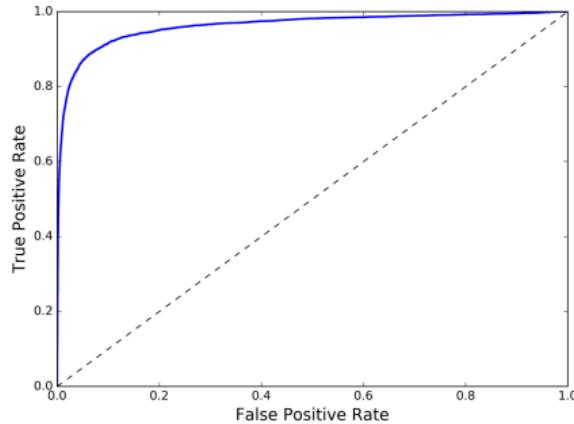
- ▶ True positive rate (TPR) (recall): $p(\hat{y} = 1 \mid y = 1)$
- ▶ False positive rate (FPR): $p(\hat{y} = 1 \mid y = 0)$
- ▶ The **receiver operating characteristic (ROC)** curves summarize the **trade-off** between the TPR and FPR for a model using different probability **thresholds**.

$$\text{Recall} = \frac{\text{Green}}{\text{Total}}$$
$$\text{FPR} = \frac{\text{Red}}{\text{Total}}$$



The ROC Curve (2/2)

- ▶ Here is a **tradeoff**: the **higher** the **TPR**, the **more FPR** the classifier produces.
- ▶ The **dotted line** represents the ROC curve of a **purely random** classifier.
- ▶ A **good classifier** moves toward the **top-left corner**.
- ▶ **Area under the curve (AUC)**





Binomial Logistic Regression Measurements in Spark

```
val lr = new LogisticRegression()  
val lrModel = lr.fit(training)
```



Binomial Logistic Regression Measurements in Spark

```
val lr = new LogisticRegression()
val lrModel = lr.fit(training)

val trainingSummary = lrModel.binarySummary

// obtain the objective per iteration.
val objectiveHistory = trainingSummary.objectiveHistory
objectiveHistory.foreach(loss => println(loss))

// obtain the ROC as a dataframe and areaUnderROC.
val roc = trainingSummary.roc
roc.show()
println(s"areaUnderROC: ${trainingSummary.areaUnderROC}")

// set the model threshold to maximize F-Measure
val fMeasure = trainingSummary.fMeasureByThreshold
val maxFMeasure = fMeasure.select(max("F-Measure")).head().getDouble(0)
val bestThreshold = fMeasure.where($"F-Measure" === maxFMeasure)
    .select("threshold").head().getDouble(0)
lrModel.setThreshold(bestThreshold)
```



Multinomial Logistic Regression in Spark (1/2)

```
val trainingSummary = lrModel.summary

// for multiclass, we can inspect metrics on a per-label basis
println("False positive rate by label:")
trainingSummary.falsePositiveRateByLabel.zipWithIndex.foreach { case (rate, label) =>
  println(s"label $label: $rate")
}

println("True positive rate by label:")
trainingSummary.truePositiveRateByLabel.zipWithIndex.foreach { case (rate, label) =>
  println(s"label $label: $rate")
}
```

Multinomial Logistic Regression in Spark (2/2)

```
println("Precision by label:")
trainingSummary.precisionByLabel.zipWithIndex.foreach { case (prec, label) =>
    println(s"label $label: $prec")
}

println("Recall by label:")
trainingSummary.recallByLabel.zipWithIndex.foreach { case (rec, label) =>
    println(s"label $label: $rec")
}

val accuracy = trainingSummary.accuracy
val falsePositiveRate = trainingSummary.weightedFalsePositiveRate
val truePositiveRate = trainingSummary.weightedTruePositiveRate
val fMeasure = trainingSummary.weightedFMeasure
val precision = trainingSummary.weightedPrecision
val recall = trainingSummary.weightedRecall
```



Summary



Summary

- ▶ Binomial logistic regression
 - $y \in \{0, 1\}$
 - Sigmoid function
 - Minimize the cross-entropy
- ▶ Multinomial logistic regression
 - $y \in \{1, 2, \dots, k\}$
 - Softmax function
 - Minimize the cross-entropy
- ▶ Performance measurements
 - TP, TF, FP, FN
 - Precision, recall, F1
 - Threshold and ROC



Reference

- ▶ Ian Goodfellow et al., Deep Learning (Ch. 4, 5)
- ▶ Aurélien Géron, Hands-On Machine Learning (Ch. 3)
- ▶ Matei Zaharia et al., Spark - The Definitive Guide (Ch. 26)

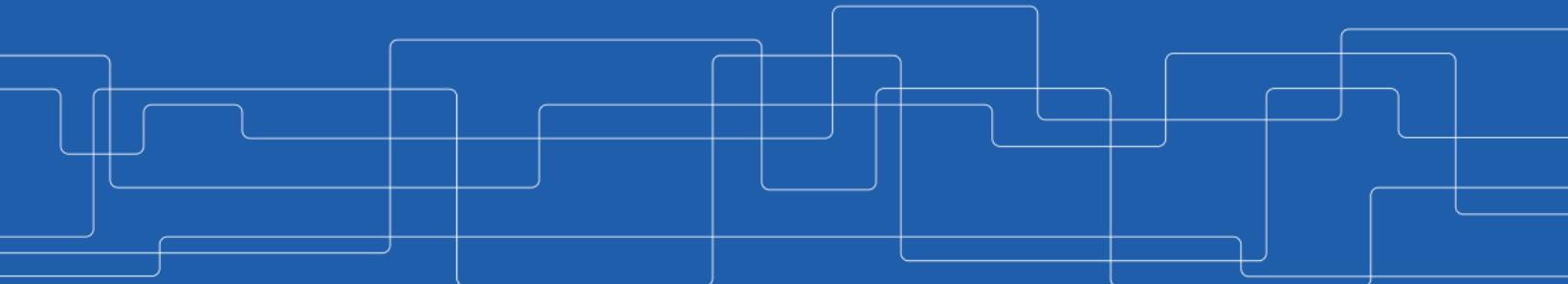


Questions?



More on Supervised Learning

Amir H. Payberah
payberah@kth.se
2021-11-17



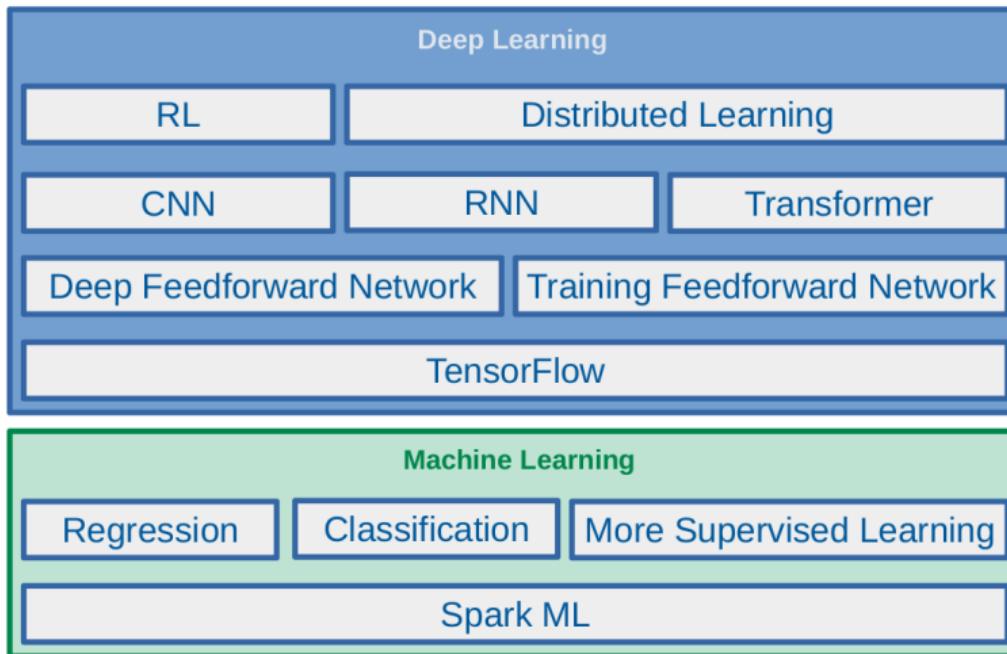


The Course Web Page

<https://id2223kth.github.io>
<https://tinyurl.com/6s5jy46a>

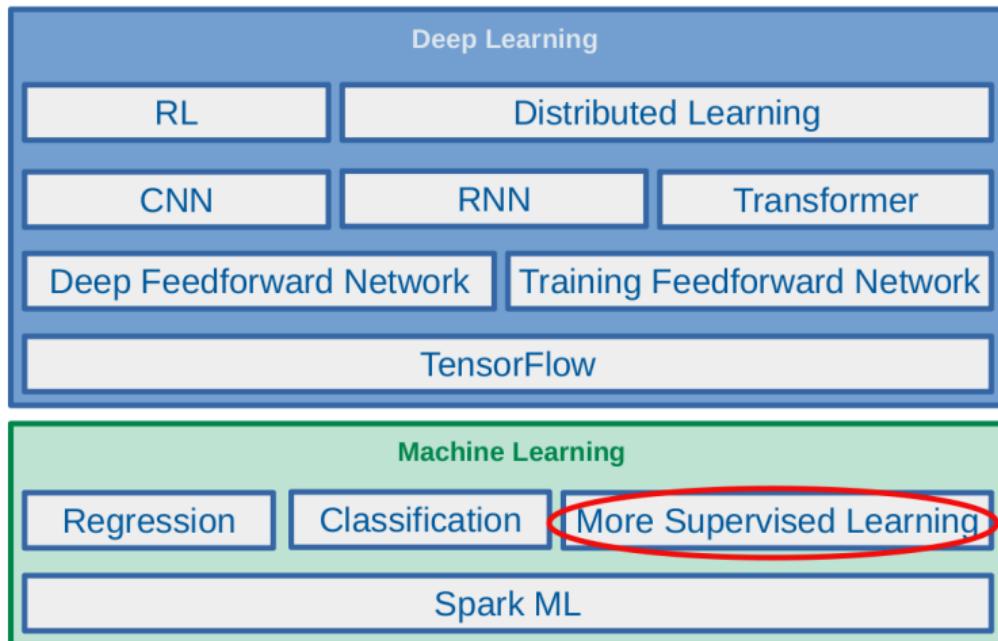


Where Are We?





Where Are We?





Let's Start with an Example



Buying Computer Example (1/3)

- Given the dataset of m people.

id	age	income	student	credit rating	buys computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middleage	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
:	:	:	:	:	:



Buying Computer Example (1/3)

- Given the dataset of m people.

id	age	income	student	credit rating	buys computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middleage	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
:	:	:	:	:	:

- Predict if a new person buys a computer?



Buying Computer Example (1/3)

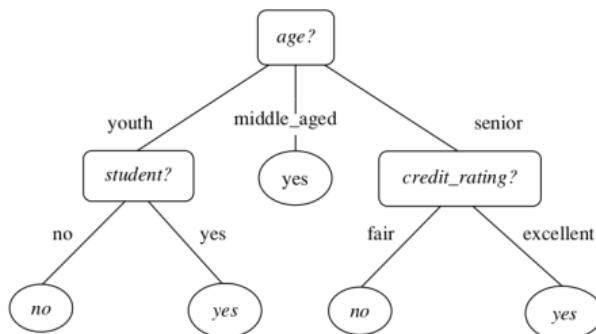
- Given the dataset of m people.

id	age	income	student	credit rating	buys computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middleage	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
:	:	:	:	:	:

- Predict if a new person buys a computer?
- Given an instance $x^{(i)}$, e.g., $x_1^{(i)} = \text{senior}$, $x_2^{(i)} = \text{medium}$, $x_3^{(i)} = \text{no}$, and $x_4^{(i)} = \text{fair}$, then $y^{(i)} = ?$

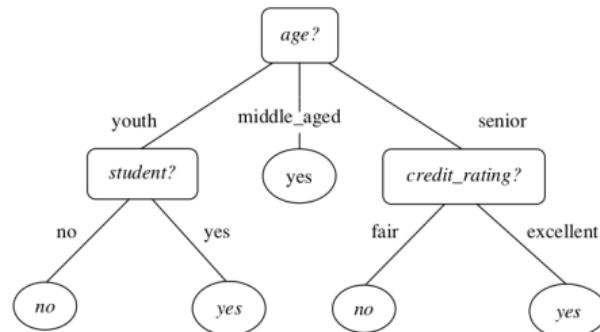
Buying Computer Example (2/3)

id	age	income	student	credit rating	buys computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middleage	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
:	:	:	:	:	:



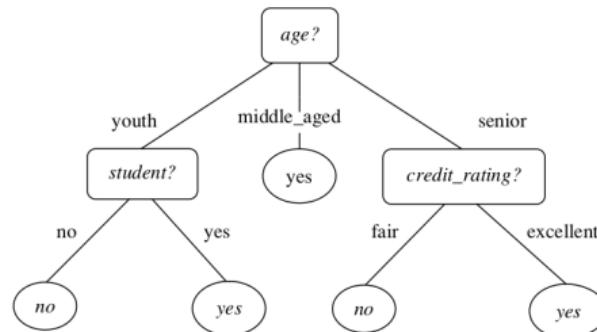
Buying Computer Example (3/3)

- Given an input instance $x^{(i)}$, for which the class label $y^{(i)}$ is unknown.



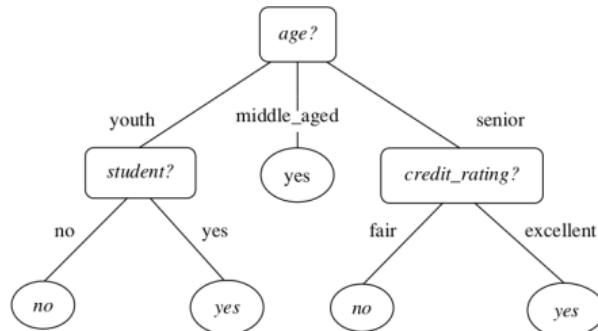
Buying Computer Example (3/3)

- ▶ Given an input instance $x^{(i)}$, for which the class label $y^{(i)}$ is unknown.
- ▶ The attribute values of the input (e.g., `age` or `income`) are tested.



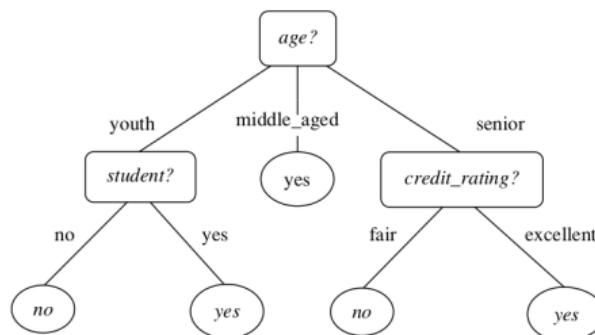
Buying Computer Example (3/3)

- ▶ Given an input instance $x^{(i)}$, for which the **class label** $y^{(i)}$ is **unknown**.
- ▶ The **attribute values** of the input (e.g., **age** or **income**) are **tested**.
- ▶ A **path** is traced from the **root** to a **leaf** node, which holds the **class prediction** for that input.



Buying Computer Example (3/3)

- ▶ Given an input instance $x^{(i)}$, for which the **class label** $y^{(i)}$ is **unknown**.
- ▶ The **attribute values** of the input (e.g., **age** or **income**) are **tested**.
- ▶ A **path** is traced from the **root** to a **leaf** node, which holds the **class prediction** for that input.
- ▶ E.g., input $x^{(i)}$ with $x_1^{(i)} = \text{senior}$, $x_2^{(i)} = \text{medium}$, $x_3^{(i)} = \text{no}$, and $x_4^{(i)} = \text{fair}$.

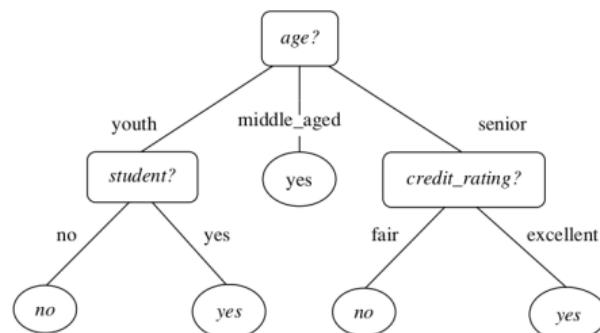




Decision Tree

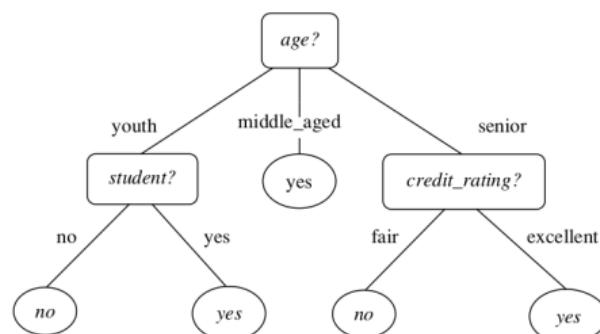
Decision Tree

- ▶ A **decision tree** is a flowchart-like tree structure.



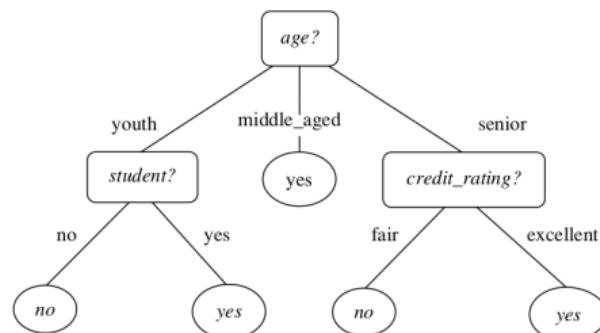
Decision Tree

- ▶ A **decision tree** is a flowchart-like tree structure.
 - The **topmost node**: represents the **root**



Decision Tree

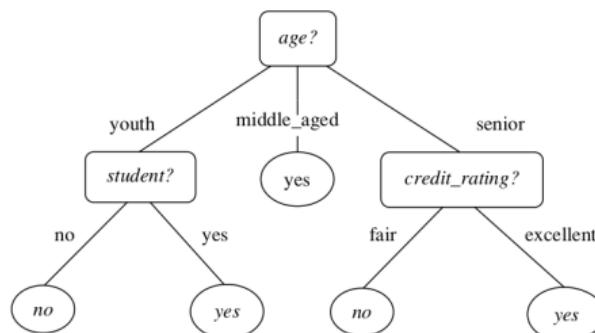
- ▶ A **decision tree** is a flowchart-like tree structure.
 - The **topmost node**: represents the **root**
 - Each **internal node**: denotes a **test** on an attribute



Decision Tree

- ▶ A **decision tree** is a flowchart-like tree structure.

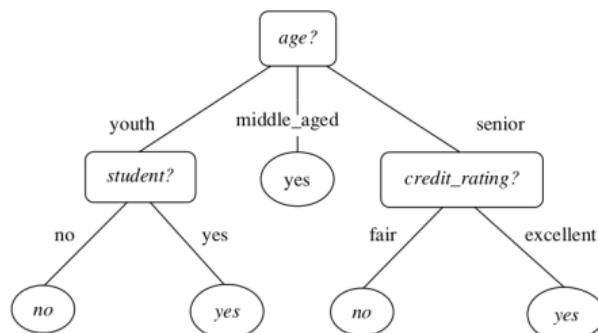
- The **topmost node**: represents the **root**
- Each **internal node**: denotes a **test** on an **attribute**
- Each **branch**: represents an **outcome** of the test



Decision Tree

- ▶ A **decision tree** is a flowchart-like tree structure.

- The **topmost node**: represents the **root**
- Each **internal node**: denotes a **test** on an **attribute**
- Each **branch**: represents an **outcome** of the test
- Each **leaf**: holds a **class label**





Training Algorithm (1/2)

- ▶ Decision trees are constructed in a **top-down recursive divide-and-conquer** manner.



Training Algorithm (1/2)

- ▶ Decision trees are constructed in a top-down recursive divide-and-conquer manner.
- ▶ The algorithm is called with the following parameters.



Training Algorithm (1/2)

- ▶ Decision trees are constructed in a top-down recursive divide-and-conquer manner.
- ▶ The algorithm is called with the following parameters.
 - Data partition D : initially the complete set of training data and labels $D = (X, y)$.



Training Algorithm (1/2)

- ▶ Decision trees are constructed in a **top-down recursive divide-and-conquer** manner.
- ▶ The algorithm is called with the following **parameters**.
 - **Data partition D**: initially the **complete set of training data and labels** $D = (X, y)$.
 - **Feature list**: list of features $\{x_1^{(i)}, \dots, x_n^{(i)}\}$ of each data instance $x^{(i)}$.



Training Algorithm (1/2)

- ▶ Decision trees are constructed in a top-down recursive divide-and-conquer manner.
- ▶ The algorithm is called with the following parameters.
 - Data partition D : initially the complete set of training data and labels $D = (X, y)$.
 - Feature list: list of features $\{x_1^{(i)}, \dots, x_n^{(i)}\}$ of each data instance $x^{(i)}$.
 - Feature selection method: determines the splitting criterion.



Training Algorithm (2/2)

- ▶ 1. The tree starts as a **single node**, N , representing the **training data instances** D .



Training Algorithm (2/2)

- ▶ 1. The tree starts as a **single node**, **N**, representing the **training data instances D**.
- ▶ 2. If all instances **x** in **D** are all of the **same class**, then node **N** becomes a **leaf**.



Training Algorithm (2/2)

- ▶ 1. The tree starts as a **single node**, **N**, representing the **training data instances D**.
- ▶ 2. If all instances **x** in **D** are all of the **same class**, then node **N** becomes a **leaf**.
- ▶ 3. The algorithm calls **feature selection method** to determine the **splitting criterion**.



Training Algorithm (2/2)

- ▶ 1. The tree starts as a **single node**, N , representing the **training data instances** D .
- ▶ 2. If all instances x in D are all of the **same class**, then node N becomes a **leaf**.
- ▶ 3. The algorithm calls **feature selection method** to determine the **splitting criterion**.
 - Indicates (i) the **splitting feature** x_k , and (ii) a **split-point** or a **splitting subset**.
 - The instances in D are partitioned accordingly.



Training Algorithm (2/2)

- ▶ 1. The tree starts as a **single node**, N , representing the **training data instances** D .
- ▶ 2. If all instances x in D are all of the **same class**, then node N becomes a **leaf**.
- ▶ 3. The algorithm calls **feature selection method** to determine the **splitting criterion**.
 - Indicates (i) the **splitting feature** x_k , and (ii) a **split-point** or a **splitting subset**.
 - The instances in D are partitioned accordingly.
- ▶ 4. The algorithm repeats the same process **recursively** to form a decision tree.



Training Algorithm - Termination Conditions

- ▶ The training algorithm **stops** only when any one of the following **conditions** is true.



Training Algorithm - Termination Conditions

- ▶ The training algorithm **stops** only when any one of the following **conditions** is true.
 - ▶ 1. All the **instances** in partition **D** at a node **N** belong to **the same class**.
 - It is **labeled with that class**.



Training Algorithm - Termination Conditions

- ▶ The training algorithm **stops** only when any one of the following **conditions** is true.
 - ▶ 1. All the **instances** in partition **D** at a node **N** belong to **the same class**.
 - It is **labeled with that class**.
 - ▶ 2. **No remaining features** on which the instances may be **further partitioned**.



Training Algorithm - Termination Conditions

- ▶ The training algorithm **stops** only when any one of the following **conditions** is true.
 - ▶ 1. All the **instances** in partition **D** at a node **N** belong to **the same class**.
 - It is **labeled with that class**.
 - ▶ 2. **No remaining features** on which the instances may be **further partitioned**.
 - ▶ 3. There are **no instances** for a **given branch**, that is, a partition **D_j** is **empty**.



Training Algorithm - Termination Conditions

- ▶ The training algorithm **stops** only when any one of the following **conditions** is true.
 - ▶ 1. All the **instances** in partition **D** at a node **N** belong to **the same class**.
 - It is **labeled with that class**.
 - ▶ 2. No remaining **features** on which the instances may be **further partitioned**.
 - ▶ 3. There are **no instances** for a **given branch**, that is, a partition **D_j** is **empty**.
- ▶ In **conditions 2 and 3**:
 - Convert node **N** into a **leaf**.
 - Label it either with the **most common class** in **D**.
 - Or, the **class distribution** of the node tuples may be stored.

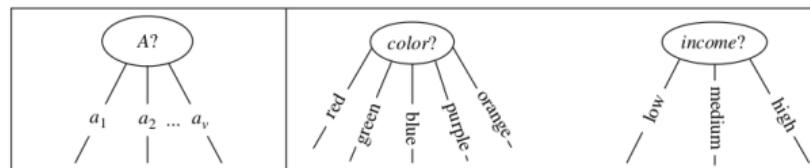


Training Algorithm - Partitioning Instances (1/3)

- ▶ Assume **A** is the **splitting feature**
- ▶ **Three** possibilities to **partition instances** in **D** based on the feature **A**.
- ▶ 1. **A** is **discrete-valued**

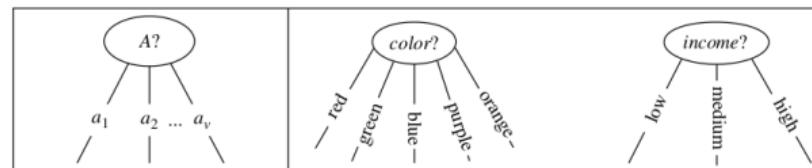
Training Algorithm - Partitioning Instances (1/3)

- ▶ Assume A is the **splitting feature**
- ▶ **Three** possibilities to **partition instances** in D based on the feature A .
- ▶ 1. A is **discrete-valued**
 - Assume A has v distinct values $\{a_1, a_2, \dots, a_v\}$



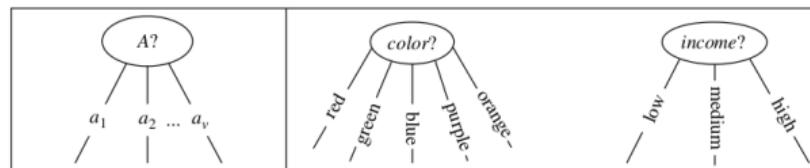
Training Algorithm - Partitioning Instances (1/3)

- ▶ Assume A is the **splitting feature**
- ▶ **Three** possibilities to **partition instances** in D based on the feature A .
- ▶ 1. A is **discrete-valued**
 - Assume A has v distinct values $\{a_1, a_2, \dots, a_v\}$
 - A **branch** is created for **each known value** a_j of A and **labeled with that value**.



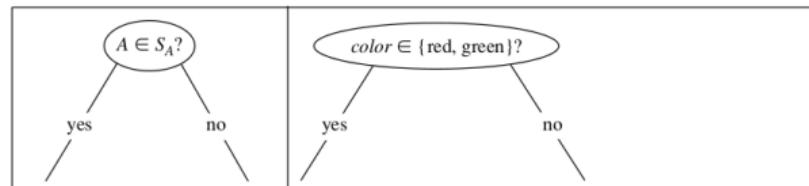
Training Algorithm - Partitioning Instances (1/3)

- ▶ Assume A is the **splitting feature**
- ▶ **Three** possibilities to **partition instances** in D based on the feature A .
- ▶ 1. A is **discrete-valued**
 - Assume A has v distinct values $\{a_1, a_2, \dots, a_v\}$
 - A **branch** is created for **each known value** a_j of A and **labeled with that value**.
 - Partition D_j is the **subset of tuples** in D having value a_j of A .



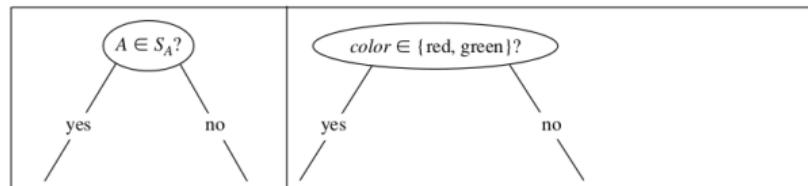
Training Algorithm - Partitioning Instances (2/3)

- ▶ 2. A is discrete-valued



Training Algorithm - Partitioning Instances (2/3)

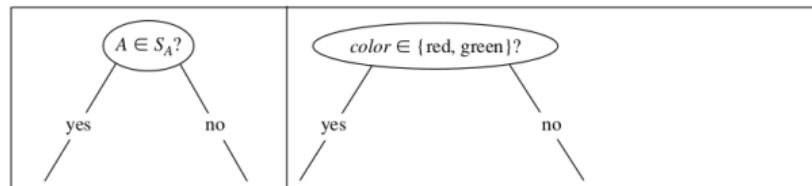
- ▶ 2. A is discrete-valued
 - A binary tree must be produced.



Training Algorithm - Partitioning Instances (2/3)

► 2. A is discrete-valued

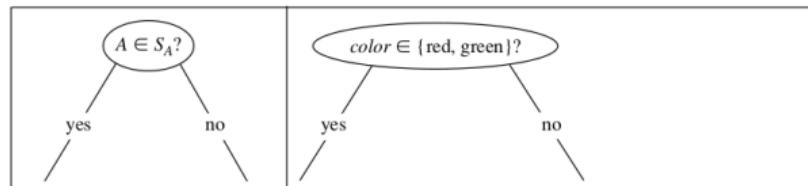
- A binary tree must be produced.
- The test at node N is of the form $A \in S_A?$, where S_A is the splitting subset for A .



Training Algorithm - Partitioning Instances (2/3)

► 2. A is discrete-valued

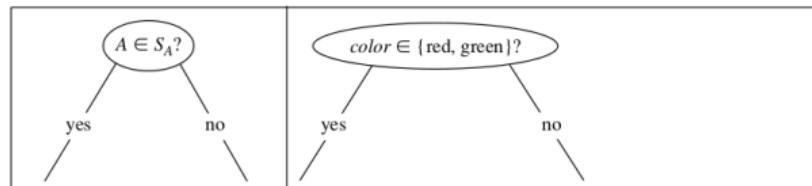
- A binary tree must be produced.
- The test at node N is of the form $A \in S_A ?$, where S_A is the splitting subset for A .
- The left branch out of N corresponds to the instances in D that satisfy the test.



Training Algorithm - Partitioning Instances (2/3)

► 2. A is discrete-valued

- A binary tree must be produced.
- The test at node N is of the form $A \in S_A ?$, where S_A is the splitting subset for A .
- The left branch out of N corresponds to the instances in D that satisfy the test.
- The right branch out of N corresponds to the instances in D that do not satisfy the test.



Training Algorithm - Partitioning Instances (3/3)



Training Algorithm - Partitioning Instances (3/3)

- ▶ 3. A is continuous-valued



Training Algorithm - Partitioning Instances (3/3)

► 3. A is continuous-valued

- A test at node N has two possible outcomes: corresponds to $A \leq s$ or $A > s$, with s as the split point.



Training Algorithm - Partitioning Instances (3/3)

► 3. A is continuous-valued

- A test at node N has two possible outcomes: corresponds to $A \leq s$ or $A > s$, with s as the split point.
- The instances are partitioned such that D_1 holds the instances in D for which $A \leq s$, while D_2 holds the rest.



Training Algorithm - Partitioning Instances (3/3)

► 3. A is continuous-valued

- A test at node N has two possible outcomes: corresponds to $A \leq s$ or $A > s$, with s as the split point.
- The instances are partitioned such that D_1 holds the instances in D for which $A \leq s$, while D_2 holds the rest.
- Two branches are labeled according to the previous outcomes.





Training Algorithm - Feature Selection Measures (1/2)

- ▶ Feature selection measure: how to **split instances** at a node **N**.



Training Algorithm - Feature Selection Measures (1/2)

- ▶ Feature selection measure: how to split instances at a node N .
- ▶ Pure partition: if all instances in a partition belong to the same class.



Training Algorithm - Feature Selection Measures (1/2)

- ▶ Feature selection measure: how to split instances at a node N .
- ▶ Pure partition: if all instances in a partition belong to the same class.
- ▶ The best splitting criterion is the one that most closely results in a pure scenario.



Training Algorithm - Feature Selection Measures (2/2)

- ▶ It provides a **ranking** for each feature describing the **given training instances**.



Training Algorithm - Feature Selection Measures (2/2)

- ▶ It provides a **ranking** for each feature describing the **given training instances**.
- ▶ The **feature** having the **best score** for the measure is chosen as the **splitting feature** for the **given instances**.



Training Algorithm - Feature Selection Measures (2/2)

- ▶ It provides a **ranking** for each feature describing the **given training instances**.
- ▶ The **feature** having the **best score** for the measure is chosen as the **splitting feature** for the **given instances**.
- ▶ **Two** popular feature selection measures are:
 - **Information gain (ID3)**
 - **Gini index (CART)**



Information Gain (Entropy)



ID3 (1/7)

- ▶ ID3 (Iterative Dichotomiser 3) uses **information gain** as its feature selection measure.



ID3 (1/7)

- ▶ ID3 (Iterative Dichotomiser 3) uses **information gain** as its **feature selection measure**.
- ▶ The feature with the **highest information gain** is chosen as the **splitting feature** for node **N**.



ID3 (1/7)

- ▶ ID3 (Iterative Dichotomiser 3) uses **information gain** as its **feature selection measure**.
- ▶ The feature with the **highest information gain** is chosen as the **splitting feature** for node **N**.
- ▶ The **information gain** is based on the **decrease in entropy** after a dataset is split on a feature.



ID3 (2/7)

- ▶ What's entropy?



ID3 (2/7)

- ▶ What's entropy?
- ▶ The average information needed to identify the class label of an instance in D.

ID3 (2/7)

- ▶ What's entropy?
- ▶ The average information needed to identify the class label of an instance in D .

$$\text{entropy}(D) = - \sum_{i=1}^m p_i \log_2(p_i)$$

- ▶ p_i is the probability that an instance in D belongs to class i , with m distinct classes.

ID3 (2/7)

- ▶ What's entropy?
- ▶ The average information needed to identify the class label of an instance in D .

$$\text{entropy}(D) = - \sum_{i=1}^m p_i \log_2(p_i)$$

- ▶ p_i is the probability that an instance in D belongs to class i , with m distinct classes.
- ▶ D 's entropy is zero when it contains instances of only one class (pure partition).

ID3 (3/7)

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle.aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle.aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle.aged	medium	no	excellent	yes
13	middle.aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$\text{entropy}(D) = - \sum_{i=1}^m p_i \log_2(p_i)$$

ID3 (3/7)

<i>RID</i>	<i>age</i>	<i>income</i>	<i>student</i>	<i>credit_rating</i>	<i>Class: buys_computer</i>
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle.aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle.aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle.aged	medium	no	excellent	yes
13	middle.aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$\text{entropy}(D) = - \sum_{i=1}^m p_i \log_2(p_i)$$

label = buys_computer $\Rightarrow m = 2$

$$\text{entropy}(D) = - \frac{9}{14} \log_2\left(\frac{9}{14}\right) - \frac{5}{14} \log_2\left(\frac{5}{14}\right) = 0.94$$



ID3 (4/7)

- ▶ Suppose we want to **partition** instances in **D** on some feature **A** with **v** distinct values, $\{a_1, a_2, \dots, a_v\}$.

ID3 (4/7)

- ▶ Suppose we want to **partition** instances in D on some feature A with v distinct values, $\{a_1, a_2, \dots, a_v\}$.
- ▶ A can split D into v partitions $\{D_1, D_2, \dots, D_v\}$.

ID3 (4/7)

- ▶ Suppose we want to **partition** instances in D on some feature A with v distinct values, $\{a_1, a_2, \dots, a_v\}$.
- ▶ A can split D into v partitions $\{D_1, D_2, \dots, D_v\}$.
- ▶ The **expected information** required to **classify an instance** from D based on the partitioning by A is:

$$\text{entropy}(A, D) = \sum_{j=1}^v \frac{|D_j|}{|D|} \text{entropy}(D_j)$$

ID3 (4/7)

- ▶ Suppose we want to **partition** instances in D on some feature A with v distinct values, $\{a_1, a_2, \dots, a_v\}$.
- ▶ A can split D into v partitions $\{D_1, D_2, \dots, D_v\}$.
- ▶ The **expected information** required to **classify** an instance from D based on the partitioning by A is:

$$\text{entropy}(A, D) = \sum_{j=1}^v \frac{|D_j|}{|D|} \text{entropy}(D_j)$$

- ▶ $\frac{|D_j|}{|D|}$ is the **weight** of the j th partition.

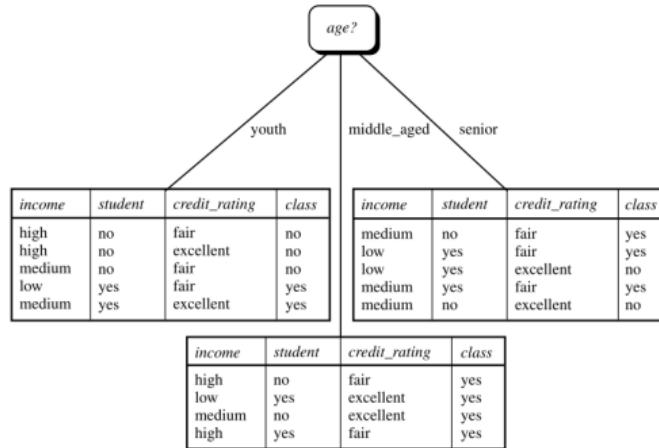
ID3 (4/7)

- ▶ Suppose we want to **partition** instances in D on some feature A with v distinct values, $\{a_1, a_2, \dots, a_v\}$.
- ▶ A can split D into v partitions $\{D_1, D_2, \dots, D_v\}$.
- ▶ The **expected information** required to **classify** an instance from D based on the partitioning by A is:

$$\text{entropy}(A, D) = \sum_{j=1}^v \frac{|D_j|}{|D|} \text{entropy}(D_j)$$

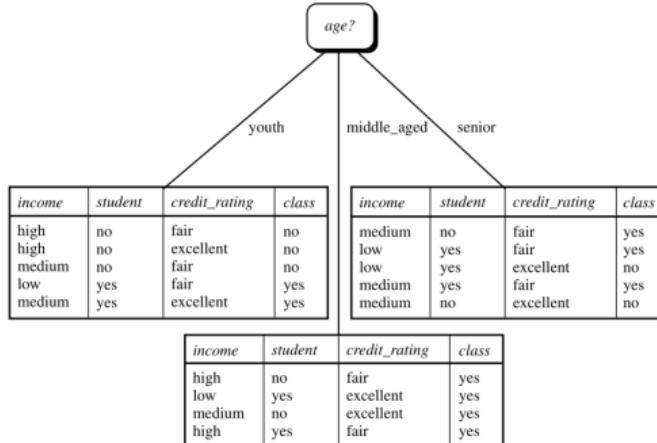
- ▶ $\frac{|D_j|}{|D|}$ is the **weight** of the j th partition.
- ▶ The **smaller** the **expected information** required, the **greater** the **purity** of the partitions.

ID3 (5/7)



$$\text{entropy}(A, D) = \sum_{j=1}^v \frac{|D_j|}{|D|} \text{entropy}(D_j)$$

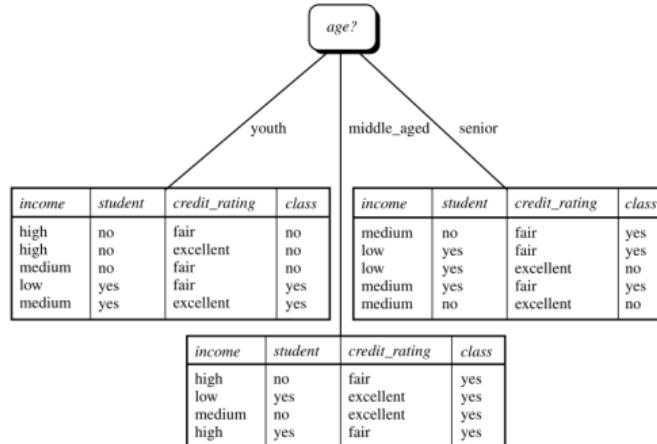
ID3 (5/7)



$$\text{entropy}(A, D) = \sum_{j=1}^v \frac{|D_j|}{|D|} \text{entropy}(D_j)$$

$$\text{entropy}(\text{age}, D) = \frac{5}{14} \text{entropy}(D_{\text{youth}}) + \frac{4}{14} \text{entropy}(D_{\text{middle_aged}}) + \frac{5}{14} \text{entropy}(D_{\text{senior}})$$

ID3 (5/7)



$$\text{entropy}(A, D) = \sum_{j=1}^v \frac{|D_j|}{|D|} \text{entropy}(D_j)$$

$$\text{entropy}(\text{age}, D) = \frac{5}{14} \text{entropy}(D_{\text{youth}}) + \frac{4}{14} \text{entropy}(D_{\text{middle_aged}}) + \frac{5}{14} \text{entropy}(D_{\text{senior}})$$

$$\text{entropy}(\text{age}, D) = \frac{5}{14} \left(-\frac{2}{5} \log_2 \left(\frac{2}{5} \right) - \frac{3}{5} \log_2 \left(\frac{3}{5} \right) \right) + \frac{4}{14} \left(-\frac{4}{4} \log_2 \left(\frac{4}{4} \right) \right) + \frac{5}{14} \left(-\frac{3}{5} \log_2 \left(\frac{3}{5} \right) - \frac{2}{5} \log_2 \left(\frac{2}{5} \right) \right) = 0.694$$



ID3 (6/7)

- ▶ The information gain $\text{Gain}(A, D)$ is defined as:

$$\text{Gain}(A, D) = \text{entropy}(D) - \text{entropy}(A, D)$$



ID3 (6/7)

- ▶ The information gain $\text{Gain}(A, D)$ is defined as:

$$\text{Gain}(A, D) = \text{entropy}(D) - \text{entropy}(A, D)$$

- ▶ It shows how much would be gained by branching on A .

ID3 (6/7)

- ▶ The information gain $\text{Gain}(A, D)$ is defined as:

$$\text{Gain}(A, D) = \text{entropy}(D) - \text{entropy}(A, D)$$

- ▶ It shows how much would be gained by branching on A .
- ▶ The feature A with the highest $\text{Gain}(A, D)$ is chosen as the splitting feature at node N .



ID3 (7/7)

- ▶ Now, we can compute the **information gain** $\text{Gain}(A)$ for the feature $A = \text{age}$.

$$\text{Gain}(\text{age}, D) = \text{entropy}(D) - \text{entropy}(\text{age}, D) = 0.940 - 0.694 = 0.246$$

ID3 (7/7)

- ▶ Now, we can compute the information gain $\text{Gain}(A)$ for the feature $A = \text{age}$.

$$\text{Gain}(\text{age}, D) = \text{entropy}(D) - \text{entropy}(\text{age}, D) = 0.940 - 0.694 = 0.246$$

- ▶ Similarly we have:

- $\text{Gain}(\text{income}, D) = 0.029$
- $\text{Gain}(\text{student}, D) = 0.151$
- $\text{Gain}(\text{credit_rating}, D) = 0.048$

ID3 (7/7)

- ▶ Now, we can compute the information gain $\text{Gain}(A)$ for the feature $A = \text{age}$.

$$\text{Gain}(\text{age}, D) = \text{entropy}(D) - \text{entropy}(\text{age}, D) = 0.940 - 0.694 = 0.246$$

- ▶ Similarly we have:
 - $\text{Gain}(\text{income}, D) = 0.029$
 - $\text{Gain}(\text{student}, D) = 0.151$
 - $\text{Gain}(\text{credit_rating}, D) = 0.048$
- ▶ The age has the highest information gain among the attributes, it is selected as the splitting feature.



Gini Impurity



CART (1/8)

- ▶ CART (Classification And Regression Tree) considers a **binary split** for each **feature**.



CART (1/8)

- ▶ CART (Classification And Regression Tree) considers a **binary split** for each **feature**.
- ▶ It uses the **Gini index** to measure the **misclassification** (**impurity of D**).

$$\text{Gini}(D) = 1 - \sum_{i=1}^m p_i^2$$



CART (1/8)

- ▶ CART (Classification And Regression Tree) considers a **binary split** for each **feature**.
- ▶ It uses the **Gini index** to measure the **misclassification** (**impurity of D**).

$$\text{Gini}(D) = 1 - \sum_{i=1}^m p_i^2$$

- ▶ p_i is the **probability** that an instance in **D** belongs to **class i**, with **m** distinct classes.



CART (1/8)

- ▶ CART (Classification And Regression Tree) considers a **binary split** for each **feature**.
- ▶ It uses the **Gini index** to measure the **misclassification** (**impurity of D**).

$$\text{Gini}(D) = 1 - \sum_{i=1}^m p_i^2$$

- ▶ p_i is the **probability** that an instance in **D** belongs to **class i**, with **m** distinct classes.
- ▶ It will be **zero** if all partitions are **pure**. **Why?**



CART (1/8)

- ▶ CART (Classification And Regression Tree) considers a **binary split** for each **feature**.
- ▶ It uses the **Gini index** to measure the **misclassification** (**impurity of D**).

$$\text{Gini}(D) = 1 - \sum_{i=1}^m p_i^2$$

- ▶ p_i is the **probability** that an instance in **D** belongs to **class i**, with **m** distinct classes.
- ▶ It will be **zero** if all partitions are **pure**. **Why?**
- ▶ We need to determine the **splitting criterion**: **splitting feature + splitting subset**.



CART (2/8)

- ▶ Assume A is a discrete-valued feature with v distinct values, $\{a_1, a_2, \dots, a_v\}$, occurring in D .

CART (2/8)

- ▶ Assume A is a discrete-valued feature with v distinct values, $\{a_1, a_2, \dots, a_v\}$, occurring in D .
- ▶ S_A will be all possible subsets of A .

CART (2/8)

- ▶ Assume A is a discrete-valued feature with v distinct values, $\{a_1, a_2, \dots, a_v\}$, occurring in D .
- ▶ S_A will be all possible subsets of A .
 - E.g., $A = \text{income} = \{\text{low}, \text{medium}, \text{high}\}$
 - $S_A = \{\{\text{low}, \text{medium}, \text{high}\}, \{\text{low}, \text{medium}\}, \{\text{medium}, \text{high}\}, \{\text{low}, \text{high}\}, \{\text{low}\}, \{\text{medium}\}, \{\text{high}\}, \{\}\}$

CART (2/8)

- ▶ Assume A is a discrete-valued feature with v distinct values, $\{a_1, a_2, \dots, a_v\}$, occurring in D .
- ▶ S_A will be all possible subsets of A .
 - E.g., $A = \text{income} = \{\text{low}, \text{medium}, \text{high}\}$
 - $S_A = \{\{\text{low}, \text{medium}, \text{high}\}, \{\text{low}, \text{medium}\}, \{\text{medium}, \text{high}\}, \{\text{low}, \text{high}\}, \{\text{low}\}, \{\text{medium}\}, \{\text{high}\}, \{\}\}$
 - The test is of the form $D_1 \in s_A?$, where s_A is a subset of S_A , e.g., $s_A = \{\text{low}, \text{high}\}$.

CART (3/8)

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$\text{Gini}(D) = 1 - \sum_{i=1}^m p_i^2$$

CART (3/8)

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$\text{Gini}(D) = 1 - \sum_{i=1}^m p_i^2$$

label = buys_computer $\Rightarrow m = 2$

$$\text{Gini}(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$

CART (4/8)

- ▶ If a binary split on A partitions D into D_1 and D_2 , the **Gini index** of D given that partitioning is:

$$\text{Gini}(A, D) = \frac{|D_1|}{D} \text{Gini}(D_1) + \frac{|D_2|}{D} \text{Gini}(D_2)$$

CART (4/8)

- ▶ If a binary split on A partitions D into D_1 and D_2 , the **Gini index** of D given that partitioning is:

$$\text{Gini}(A, D) = \frac{|D_1|}{D} \text{Gini}(D_1) + \frac{|D_2|}{D} \text{Gini}(D_2)$$

- ▶ The subset that gives the **minimum Gini index** is selected as its **splitting subset**.



CART (5/8)

- ▶ For a **feature** $A = \text{income}$, we consider each of the possible **splitting subsets**.

CART (5/8)

- ▶ For a **feature** $A = \text{income}$, we consider each of the possible **splitting subsets**.
 - $S_A = \{\{\text{low}, \text{medium}, \text{high}\}, \{\text{low}, \text{medium}\}, \{\text{medium}, \text{high}\}, \{\text{low}, \text{high}\}, \{\text{low}\}, \{\text{medium}\}, \{\text{high}\}, \{\}\}$



CART (5/8)

- ▶ For a **feature** $A = \text{income}$, we consider each of the possible **splitting subsets**.
 - $S_A = \{\{\text{low}, \text{medium}, \text{high}\}, \{\text{low}, \text{medium}\}, \{\text{medium}, \text{high}\}, \{\text{low}, \text{high}\}, \{\text{low}\}, \{\text{medium}\}, \{\text{high}\}, \{\}\}$
- ▶ Assume, we choose the **splitting subset** $s_A = \{\text{low}, \text{medium}\}$.

CART (5/8)

- ▶ For a **feature** $A = \text{income}$, we consider each of the possible **splitting subsets**.
 - $S_A = \{\{\text{low}, \text{medium}, \text{high}\}, \{\text{low}, \text{medium}\}, \{\text{medium}, \text{high}\}, \{\text{low}, \text{high}\}, \{\text{low}\}, \{\text{medium}\}, \{\text{high}\}, \{\}\}$
- ▶ Assume, we choose the **splitting subset** $s_A = \{\text{low}, \text{medium}\}$.
- ▶ Consider partition D_1 satisfies the condition $D_1 \in s_A$, and D_2 does not.

$$\begin{aligned} \text{Gini}_{\text{income} \in \{\text{low}, \text{medium}\}}(A, D) &= \frac{10}{14} \text{Gini}(D_1) + \frac{4}{14} \text{Gini}(D_2) \\ &= \frac{10}{14} \text{Gini}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14} \left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right) = 0.443 \end{aligned}$$

CART (6/8)

- ▶ Similarly, we calculate the Gini index values for splits on the remaining subsets.

$$\text{Gini}_{\text{income} \in \{\text{low, medium}\}}(A, D) = \text{Gini}_{\text{income} \in \{\text{high}\}}(A, D) = 0.443$$

$$\text{Gini}_{\text{income} \in \{\text{low, high}\}}(A, D) = \text{Gini}_{\text{income} \in \{\text{medium}\}}(A, D) = 0.458$$

$$\text{Gini}_{\text{income} \in \{\text{medium, high}\}}(A, D) = \text{Gini}_{\text{income} \in \{\text{low}\}}(A, D) = 0.450$$

CART (6/8)

- ▶ Similarly, we calculate the Gini index values for splits on the remaining subsets.

$$\text{Gini}_{\text{income} \in \{\text{low}, \text{medium}\}}(A, D) = \text{Gini}_{\text{income} \in \{\text{high}\}}(A, D) = 0.443$$

$$\text{Gini}_{\text{income} \in \{\text{low}, \text{high}\}}(A, D) = \text{Gini}_{\text{income} \in \{\text{medium}\}}(A, D) = 0.458$$

$$\text{Gini}_{\text{income} \in \{\text{medium}, \text{high}\}}(A, D) = \text{Gini}_{\text{income} \in \{\text{low}\}}(A, D) = 0.450$$

- ▶ The best binary split for attribute $A = \text{income}$ is on $s_A = \{\text{low}, \text{medium}\}$ because it minimizes the Gini index.



CART (7/8)

- ▶ But, which feature?



CART (7/8)

- ▶ But, which feature?
- ▶ The reduction in impurity that would be incurred by a binary split on feature A is:

$$\Delta \text{Gini}(A) = \text{Gini}(D) - \text{Gini}(A, D)$$



CART (7/8)

- ▶ But, which feature?
- ▶ The reduction in impurity that would be incurred by a binary split on feature A is:

$$\Delta \text{Gini}(A) = \text{Gini}(D) - \text{Gini}(A, D)$$

- ▶ The feature that maximizes the reduction in impurity (has the minimum Gini index) is selected as the splitting feature.

CART (8/8)

- ▶ Now, we can compute the information gain $\text{Gain}(A)$ for different features.
 - $\Delta\text{Gini}(\text{income}) = 0.459 - 0.443 = 0.016$
 - $\Delta\text{Gini}(\text{age}) = 0.459 - 0.357 = 0.102$
 - $\Delta\text{Gini}(\text{student}) = 0.459 - 0.367 = 0.092$
 - $\Delta\text{Gini}(\text{credit_rating}) = 0.459 - 0.429 = 0.03$

CART (8/8)

- ▶ Now, we can compute the **information gain** $\text{Gain}(A)$ for different features.
 - $\Delta\text{Gini}(\text{income}) = 0.459 - 0.443 = 0.016$
 - $\Delta\text{Gini}(\text{age}) = 0.459 - 0.357 = 0.102$
 - $\Delta\text{Gini}(\text{student}) = 0.459 - 0.367 = 0.092$
 - $\Delta\text{Gini}(\text{credit_rating}) = 0.459 - 0.429 = 0.03$
- ▶ The feature $A = \text{age}$ and splitting subset $s_A = \{\text{youth}, \text{senior}\}$ gives the **minimum Gini index** overall.



Decision Tree in Spark (1/4)

- ▶ Two classes in `spark.ml`.
- ▶ Regression: `DecisionTreeRegressor`

```
val dt_regressor = new DecisionTreeRegressor().setLabelCol("label").setFeaturesCol("features")
val model = dt_regressor.fit(trainingData)
val predictions = model.transform(testData)
predictions.select("prediction", "rawPrediction", "probability", "label", "features").show(5)
```



Decision Tree in Spark (1/4)

- ▶ Two classes in `spark.ml`.
- ▶ Regression: `DecisionTreeRegressor`

```
val dt_regressor = new DecisionTreeRegressor().setLabelCol("label").setFeaturesCol("features")
val model = dt_regressor.fit(trainingData)
val predictions = model.transform(testData)
predictions.select("prediction", "rawPrediction", "probability", "label", "features").show(5)
```

- ▶ Classifier: `DecisionTreeClassifier`

```
val dt_classifier = new DecisionTreeClassifier().setLabelCol("label").setFeaturesCol("features")
val model = dt_classifier.fit(trainingData)
val predictions = model.transform(testData)
predictions.select("prediction", "rawPrediction", "probability", "label", "features").show(5)
```



Decision Tree in Spark (2/4)

- ▶ Input and output columns



Decision Tree in Spark (2/4)

- ▶ Input and output columns
- ▶ `labelCol` and `featuresCol` identify `label` and `features` column's names.



Decision Tree in Spark (2/4)

- ▶ Input and output columns
- ▶ `labelCol` and `featuresCol` identify label and features column's names.
- ▶ `predictionCol` indicates the predicted label.



Decision Tree in Spark (2/4)

- ▶ Input and output columns
- ▶ `labelCol` and `featuresCol` identify label and features column's names.
- ▶ `predictionCol` indicates the predicted label.
- ▶ `rawPredictionCol` is a vector of length of number of classes, with the counts of training instance labels at the tree node which makes the prediction.



Decision Tree in Spark (2/4)

- ▶ Input and output columns
- ▶ `labelCol` and `featuresCol` identify label and features column's names.
- ▶ `predictionCol` indicates the predicted label.
- ▶ `rawPredictionCol` is a vector of length of number of classes, with the counts of training instance labels at the tree node which makes the prediction.
- ▶ `probabilityCol` is a vector of length of number of classes equal to `rawPrediction` normalized to a multinomial distribution.



Decision Tree in Spark (3/4)

- ▶ Tunable parameters



Decision Tree in Spark (3/4)

- ▶ Tunable parameters
- ▶ `maxBins`: number of bins used when discretizing continuous features.



Decision Tree in Spark (3/4)

- ▶ Tunable parameters
- ▶ `maxBins`: number of bins used when discretizing continuous features.
- ▶ `impurity`: impurity measure used to choose between candidate splits, e.g., `entropy` and `gini`.

```
val maxBins = ...  
val dt_classifier = new DecisionTreeClassifier().setMaxBins(maxBins).setImpurity("gini")
```



Decision Tree in Spark (4/4)

- ▶ Stopping criteria that determines when the tree stops building.



Decision Tree in Spark (4/4)

- ▶ Stopping criteria that determines when the tree stops building.
- ▶ `maxDepth`: maximum depth of a tree.



Decision Tree in Spark (4/4)

- ▶ Stopping criteria that determines when the tree stops building.
- ▶ `maxDepth`: maximum depth of a tree.
- ▶ `minInstancesPerNode`: for a node to be split further, each of its children must receive at least this number of training instances.



Decision Tree in Spark (4/4)

- ▶ Stopping criteria that determines when the tree stops building.
- ▶ `maxDepth`: maximum depth of a tree.
- ▶ `minInstancesPerNode`: for a node to be split further, each of its children must receive at least this number of training instances.
- ▶ `minInfoGain`: for a node to be split further, the split must improve at least this much (in terms of information gain).

```
val maxDepth = ...
val minInstancesPerNode = ...
val minInfoGain = ...
val dt_classifier = new DecisionTreeClassifier()
    .setMaxDepth(maxDepth)
    .setMinInstancesPerNode(minInstancesPerNode)
    .setMinInfoGain(minInfoGain)
```



Ensemble Methods



Wisdom of the Crowd

- ▶ Ask a complex question to thousands of random people, then aggregate their answers.
- ▶ In many cases, this aggregated answer is better than an expert's answer.



Wisdom of the Crowd

- ▶ Ask a complex question to thousands of random people, then aggregate their answers.
- ▶ In many cases, this aggregated answer is better than an expert's answer.
- ▶ This is called the wisdom of the crowd.



Wisdom of the Crowd

- ▶ Ask a **complex question** to **thousands of random people**, then aggregate their answers.
- ▶ In many cases, this **aggregated answer** is **better** than an **expert's answer**.
- ▶ This is called the **wisdom of the crowd**.
- ▶ Similarly, the aggregated estimations of a **group of estimators** (e.g., **classifiers** or **regressors**), often gets **better estimations** than with the best individual estimator.
- ▶ A **group of estimators** is an **ensemble**, and this technique is called **Ensemble Learning**.



Ensemble Learning

- ▶ Two main categories of **ensemble learning** algorithms.



Ensemble Learning

- ▶ Two main categories of **ensemble learning** algorithms.
- ▶ **Bagging**
 - Use the **same training algorithm** for **every estimator**, but to train them on **different random subsets** of the training set.
 - E.g., **random forest**

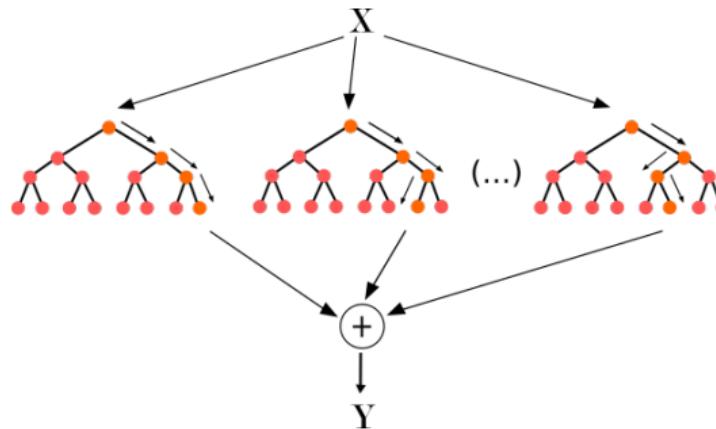


Ensemble Learning

- ▶ Two main categories of **ensemble learning** algorithms.
- ▶ **Bagging**
 - Use the **same training algorithm** for **every estimator**, but to train them on **different random subsets** of the training set.
 - E.g., **random forest**
- ▶ **Boosting**
 - Train estimators **sequentially**, each trying to **correct its predecessor**.
 - E.g., **adaboost** and **gradient boosting**

Random Forest

- ▶ **Random forest** builds **multiple decision trees** that are most of the time trained with the **bagging** method.
- ▶ It, then, merges the trees together to get a more **accurate and stable prediction**.





Random Forest in Spark (1/2)

- ▶ Two classes in `spark.ml`.
- ▶ Regression: `RandomForestRegressor`

```
val rf_regressor = new RandomForestRegressor().setLabelCol("label")
                                              .setFeaturesCol("features").setNumTrees(10)
val model = rf_regressor.fit(trainingData)
val predictions = model.transform(testData)
predictions.select("prediction", "label", "features").show(5)
```



Random Forest in Spark (1/2)

- ▶ Two classes in `spark.ml`.
- ▶ Regression: `RandomForestRegressor`

```
val rf_regressor = new RandomForestRegressor().setLabelCol("label")
                                              .setFeaturesCol("features").setNumTrees(10)
val model = rf_regressor.fit(trainingData)
val predictions = model.transform(testData)
predictions.select("prediction", "label", "features").show(5)
```

- ▶ Classifier: `RandomForestClassifier`

```
val rf_classifier = new RandomForestClassifier().setLabelCol("label")
                                              .setFeaturesCol("features").setNumTrees(10)
val model = rf_classifier.fit(trainingData)
val predictions = model.transform(testData)
predictions.select("prediction", "label", "features").show(5)
```



Random Forest in Spark (2/2)

- ▶ `numTrees`: number of trees in the forest.



Random Forest in Spark (2/2)

- ▶ `numTrees`: number of trees in the forest.
- ▶ `subsamplingRate`: specifies the size of the dataset used for training each tree in the forest, as a fraction of the size of the original dataset.
 - Default is 1.0 and decreasing it can speed up training.

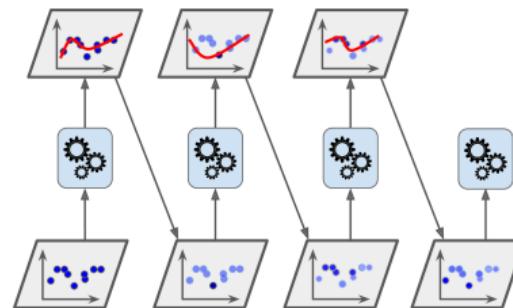


Random Forest in Spark (2/2)

- ▶ `numTrees`: number of trees in the forest.
- ▶ `subsamplingRate`: specifies the size of the dataset used for training each tree in the forest, as a fraction of the size of the original dataset.
 - Default is 1.0 and decreasing it can speed up training.
- ▶ `featureSubsetStrategy`: number of features to use as candidates for splitting at each tree node, as a fraction of the total number of features.
 - Possible values: `auto`, `all`, `onethird`, `sqrt`, `log2`, `n`

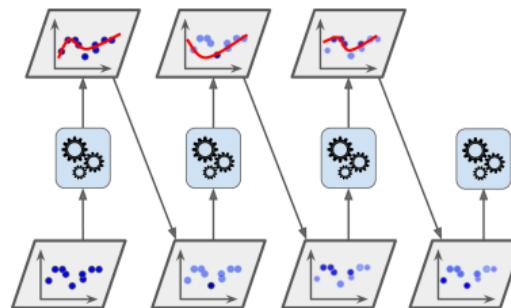
AdaBoost

- ▶ AdaBoost: train a **new estimator** by paying more attention to the training instances that the **predecessor underfitted**.



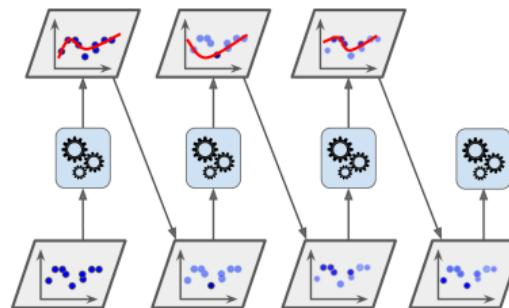
AdaBoost

- ▶ AdaBoost: train a **new estimator** by paying more attention to the training instances that the **predecessor underfitted**.
- ▶ Each **estimator** is trained on a **random subset** of the **total training set**.



AdaBoost

- ▶ AdaBoost: train a **new estimator** by paying more attention to the training instances that the **predecessor underfitted**.
- ▶ Each **estimator** is trained on a **random subset** of the **total training set**.
- ▶ AdaBoost assigns a **weight** to each **training instance**, which determines the **probability** that each instance should **appear in the training set**.





Gradient Boosting (1/3)

- ▶ Just like AdaBoost, Gradient Boosting works by sequentially adding estimators to an ensemble, each one correcting its predecessor.
- ▶ However, instead of tweaking the instance weights at every iteration, this method tries to fit the new estimator to the residual errors made by the previous estimator.



Gradient Boosting (2/3)

- ▶ Let's go through a regression example using Gradient Boosted Regression Trees.
- ▶ Fit the first estimator on the training set.

```
tree_reg1 = DecisionTreeRegressor(max_depth=2)
tree_reg1.fit(X, y)
```

- ▶ Now train the second estimator on the residual errors made by the first estimator.

```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)
```



Gradient Boosting (3/3)

- ▶ Then we train the **third estimator** on the **residual errors** made by the **second estimator**.

```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2)
tree_reg3.fit(X, y3)
```

- ▶ Now we have an **ensemble** containing three trees.
- ▶ It can **make predictions** on a new instance simply by adding up the predictions of all the trees.

```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```



Gradient Boosting in Spark

- ▶ Two classes in `spark.ml`.
- ▶ Regression: `GBTRegressor`

```
val gbt = new GBTRegressor().setLabelCol("label").setFeaturesCol("features")
                           .setMaxIter(10).setFeatureSubsetStrategy("auto")

val model = gbt.fit(trainingData)
val predictions = model.transform(testData)
```



Gradient Boosting in Spark

- ▶ Two classes in `spark.ml`.
- ▶ Regression: `GBTRegressor`

```
val gbt = new GBTRegressor().setLabelCol("label").setFeaturesCol("features")
                           .setMaxIter(10).setFeatureSubsetStrategy("auto")

val model = gbt.fit(trainingData)
val predictions = model.transform(testData)
```

- ▶ Classifier: `GBTClassifier`

```
val gbt = new GBTClassifier().setLabelCol("label").setFeaturesCol("features")
                           .setMaxIter(10).setFeatureSubsetStrategy("auto")

val model = gbt.fit(trainingData)
val predictions = model.transform(testData)
```



Summary



Summary

- ▶ Decision tree
 - Top-down training algorithm
 - Termination condition
 - Feature selection: entropy, gini
- ▶ Ensemble models
 - Bagging: random forest
 - Boosting: AdaBoost, Gradient Boosting



Reference

- ▶ Aurélien Géron, Hands-On Machine Learning (Ch. 5, 6, 7)
- ▶ Matei Zaharia et al., Spark - The Definitive Guide (Ch. 27)

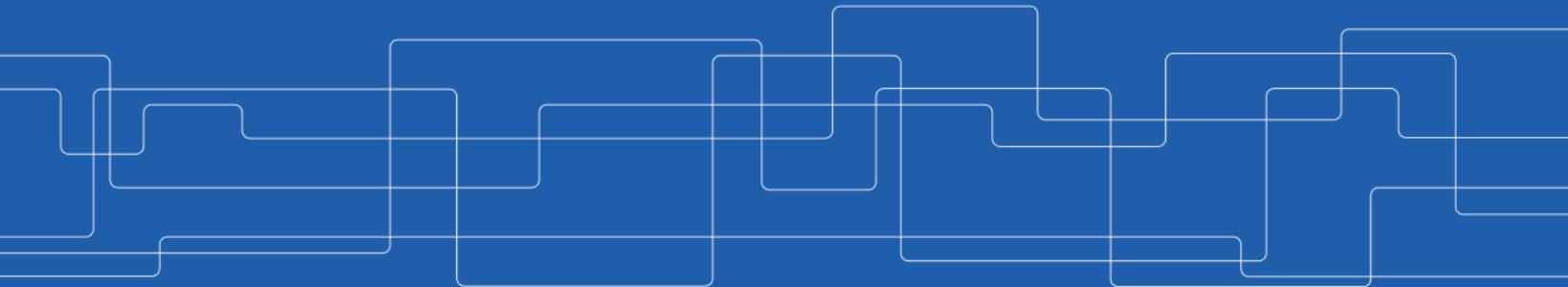


Questions?



Deep Feedforwards Networks

Amir H. Payberah
payberah@kth.se
2021-11-18



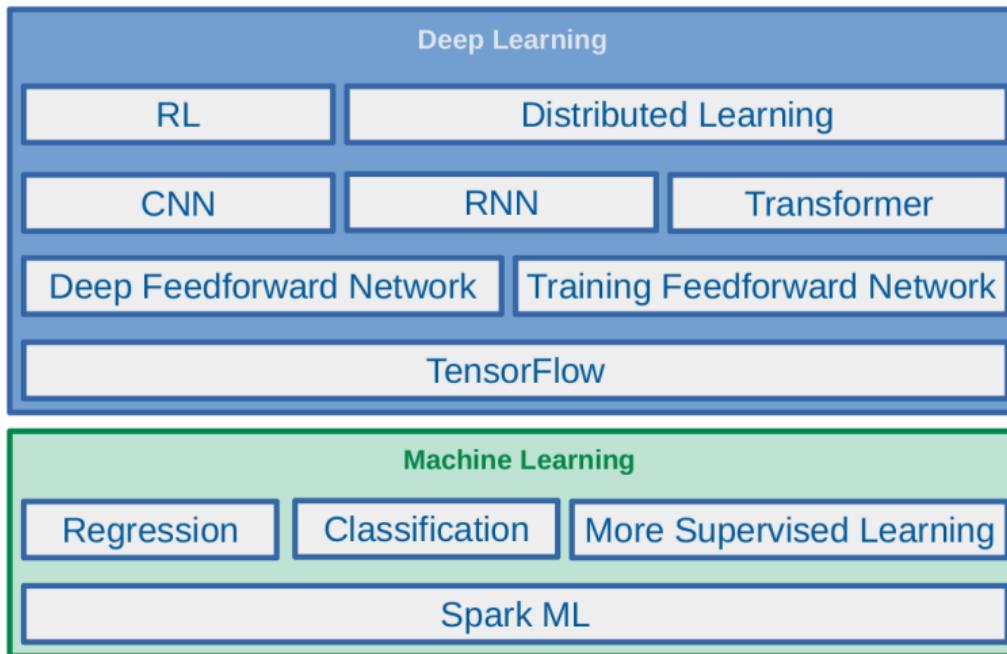


The Course Web Page

<https://id2223kth.github.io>
<https://tinyurl.com/6s5jy46a>

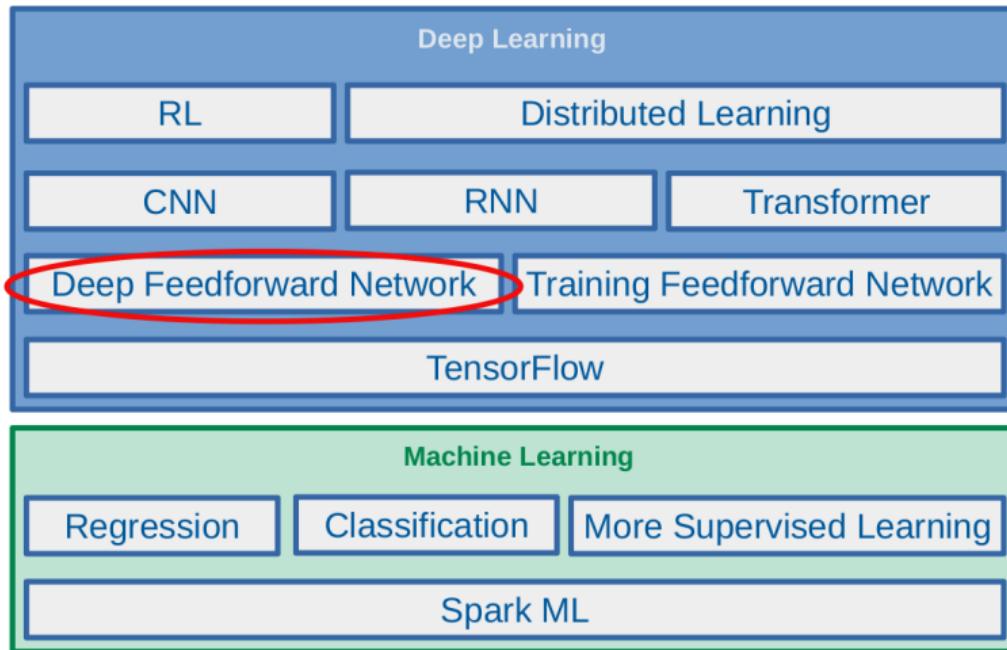


Where Are We?



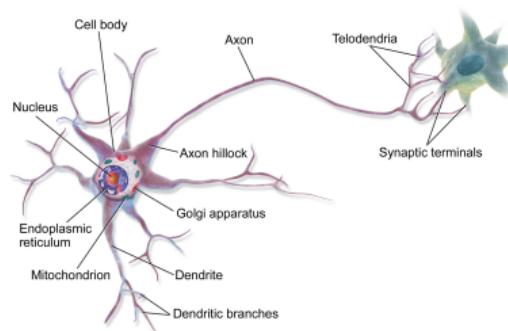


Where Are We?



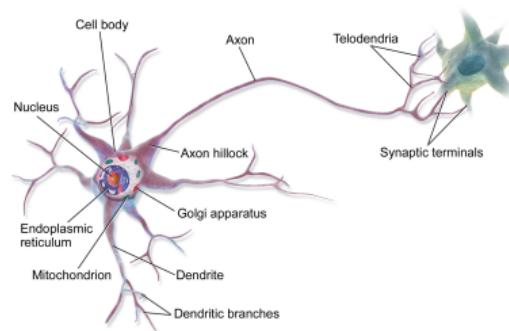
Biological Neurons (1/2)

- ▶ Brain architecture has inspired artificial neural networks.



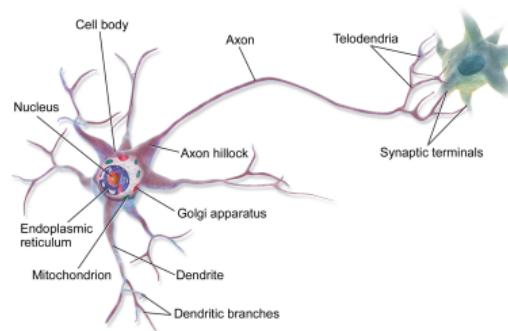
Biological Neurons (1/2)

- ▶ Brain architecture has inspired artificial neural networks.
- ▶ A biological neuron is composed of
 - Cell body, many dendrites (branching extensions), one axon (long extension), synapses



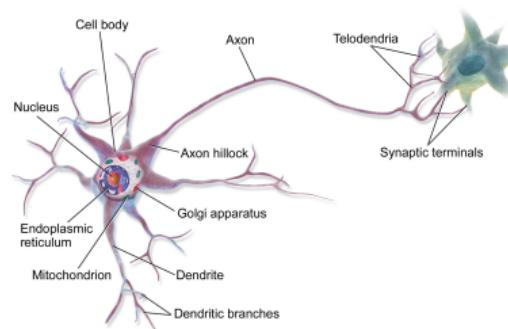
Biological Neurons (1/2)

- ▶ Brain architecture has inspired artificial neural networks.
- ▶ A biological neuron is composed of
 - Cell body, many dendrites (branching extensions), one axon (long extension), synapses
- ▶ Biological neurons receive signals from other neurons via these synapses.



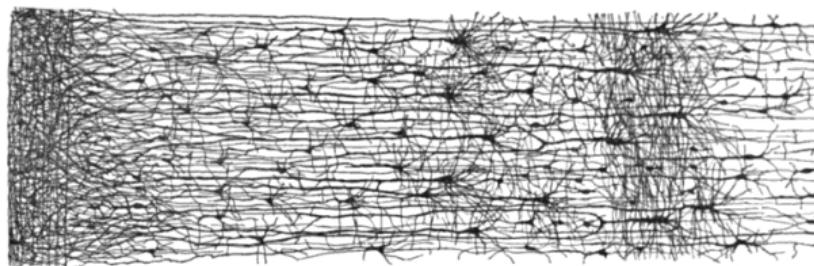
Biological Neurons (1/2)

- ▶ Brain architecture has inspired artificial neural networks.
- ▶ A biological neuron is composed of
 - Cell body, many dendrites (branching extensions), one axon (long extension), synapses
- ▶ Biological neurons receive signals from other neurons via these synapses.
- ▶ When a neuron receives a sufficient number of signals within a few milliseconds, it fires its own signals.



Biological Neurons (2/2)

- ▶ Biological neurons are organized in a vast **network of billions of neurons**.
- ▶ Each neuron typically is **connected to thousands of other neurons**.



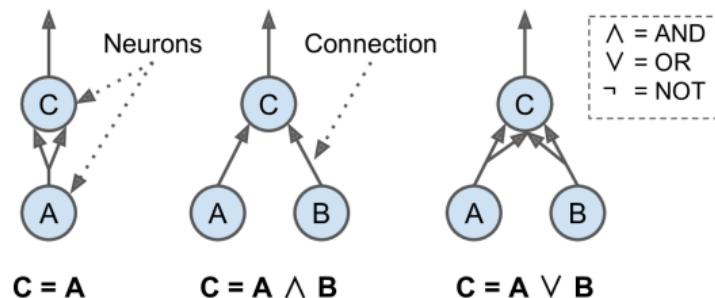


A Simple Artificial Neural Network

- ▶ One or more **binary inputs** and **one binary output**
- ▶ Activates its **output** when more than a **certain number of its inputs** are active.

A Simple Artificial Neural Network

- ▶ One or more **binary inputs** and **one binary output**
- ▶ Activates its **output** when more than a **certain number of its inputs are active**.





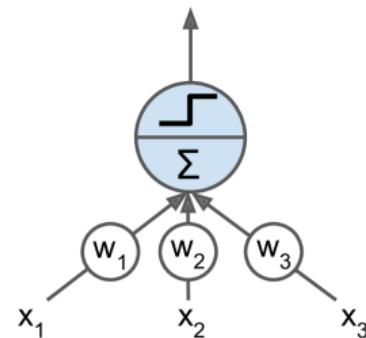
The Linear Threshold Unit (LTU)

- ▶ Inputs of a LTU are **numbers** (**not binary**).

The Linear Threshold Unit (LTU)

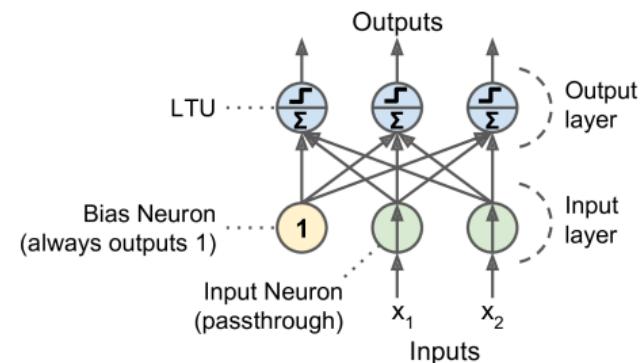
- ▶ Inputs of a LTU are **numbers** (not binary).
- ▶ Each **input connection** is associated with a **weight**.
- ▶ Computes a **weighted sum of its inputs** and applies a **step function** to that **sum**.

- ▶ $z = w_1x_1 + w_2x_2 + \dots + w_nx_n = w^T x$
- ▶ $\hat{y} = \text{step}(z) = \text{step}(w^T x)$



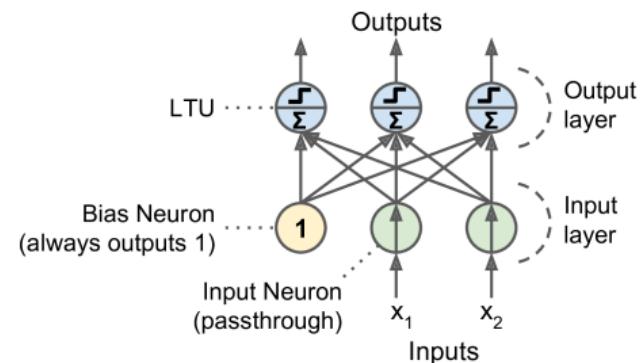
The Perceptron

- The **perceptron** is a **single layer** of LTUs.



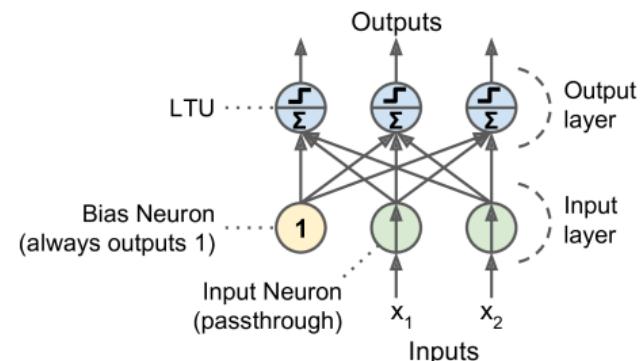
The Perceptron

- ▶ The **perceptron** is a **single layer** of LTUs.
- ▶ The **input neurons** output whatever **input** they are fed.



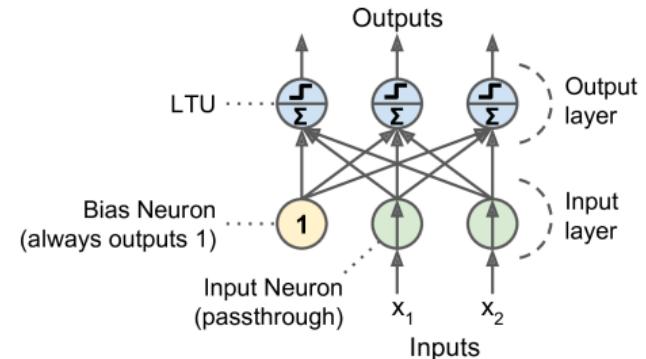
The Perceptron

- ▶ The **perceptron** is a **single layer** of LTUs.
- ▶ The **input neurons** output whatever **input** they are fed.
- ▶ A **bias neuron**, which just **outputs 1** all the time.



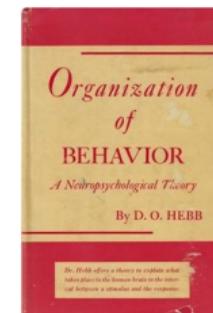
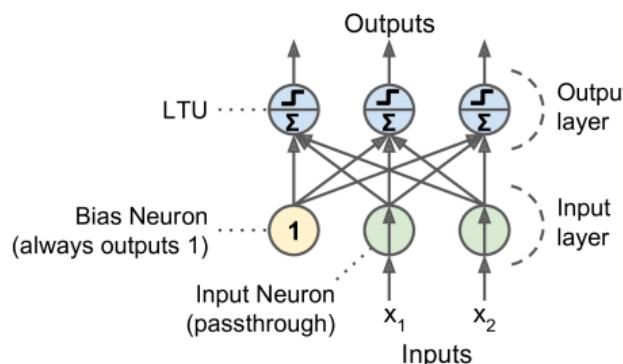
The Perceptron

- ▶ The **perceptron** is a **single layer** of LTUs.
- ▶ The **input neurons** output whatever **input** they are fed.
- ▶ A **bias neuron**, which just **outputs 1** all the time.
- ▶ If we use **logistic function (sigmoid)** instead of a **step** function, it computes a **continuous** output.



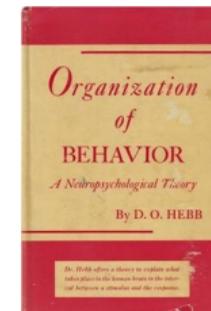
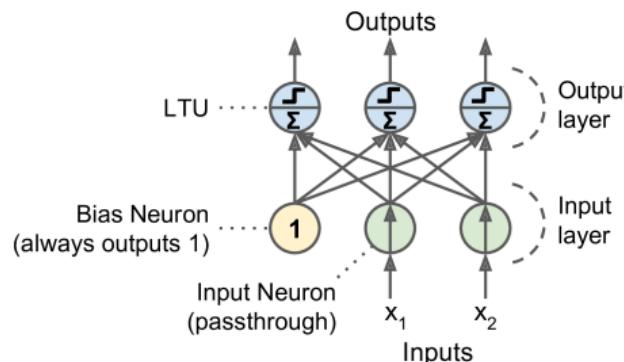
How is a Perceptron Trained? (1/2)

- ▶ The **Perceptron training algorithm** is inspired by **Hebb's rule**.



How is a Perceptron Trained? (1/2)

- ▶ The **Perceptron training algorithm** is inspired by **Hebb's rule**.
- ▶ When a **biological neuron** often **triggers another neuron**, the **connection** between these two neurons grows **stronger**.





How is a Perceptron Trained? (2/2)

- ▶ Feed **one training instance x** to each neuron j at a time and make its **prediction \hat{y}** .
- ▶ Update the **connection weights**.

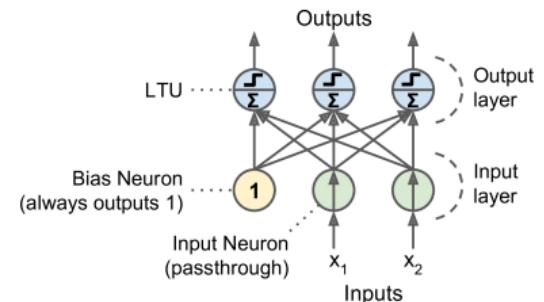
How is a Perceptron Trained? (2/2)

- ▶ Feed **one training instance x** to each neuron j at a time and make its **prediction \hat{y}** .
- ▶ Update the **connection weights**.

$$\hat{y}_j = \sigma(\mathbf{w}_j^T \mathbf{x} + b)$$

$$J(w_j) = \text{cross_entropy}(y_j, \hat{y}_j)$$

$$w_{i,j}^{(\text{next})} = w_{i,j} - \eta \frac{\partial J(w_j)}{w_i}$$



How is a Perceptron Trained? (2/2)

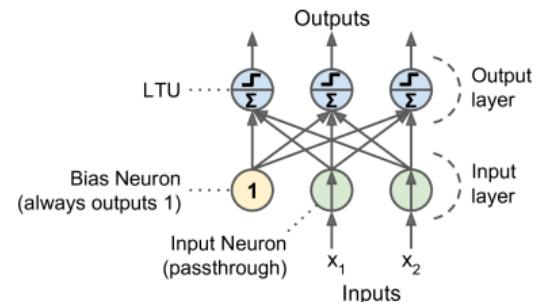
- ▶ Feed **one training instance x** to each neuron j at a time and make its **prediction \hat{y}** .
- ▶ Update the **connection weights**.

$$\hat{y}_j = \sigma(\mathbf{w}_j^T \mathbf{x} + b)$$

$$J(w_j) = \text{cross_entropy}(y_j, \hat{y}_j)$$

$$w_{i,j}^{(\text{next})} = w_{i,j} - \eta \frac{\partial J(w_j)}{w_i}$$

- ▶ $w_{i,j}$: the **weight** between neurons i and j .
- ▶ x_i : the **i th input value**.
- ▶ \hat{y}_j : the **j th predicted output value**.
- ▶ y_j : the **j th true output value**.
- ▶ η : the **learning rate**.

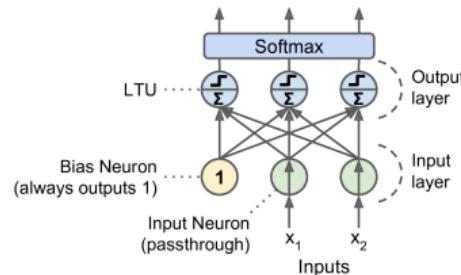




Perceptron in TensorFlow



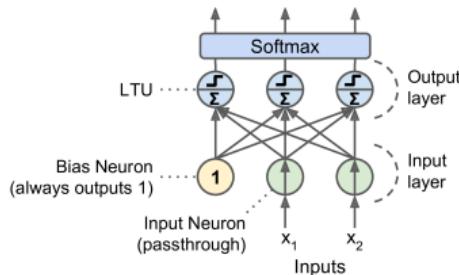
Perceptron in TensorFlow



```
n_neurons = 3
n_features = 2

model = keras.models.Sequential()
model.add(keras.layers.Dense(n_neurons, input_shape=(n_features,), activation="softmax"))
```

Perceptron in TensorFlow



```
n_neurons = 3
n_features = 2

model = keras.models.Sequential()
model.add(keras.layers.Dense(n_neurons, input_shape=(n_features,), activation="softmax"))

model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd", metrics=["accuracy"])
model.fit(X_train, y_train, epochs=30)
```



Multi-Layer Perceptron (MLP)



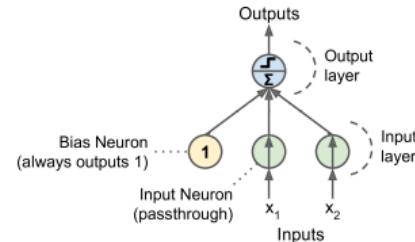
Perceptron Weakness (1/2)

- ▶ Incapable of solving some **trivial problems**, e.g., **XOR** classification problem. **Why?**

Perceptron Weakness (1/2)

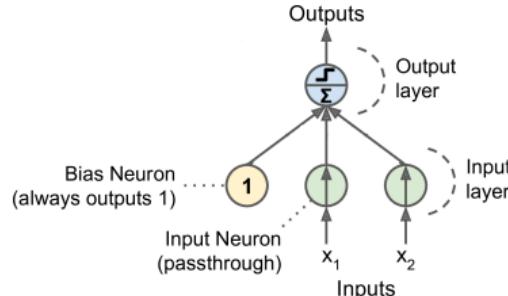
- Incapable of solving some **trivial problems**, e.g., **XOR** classification problem. **Why?**

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

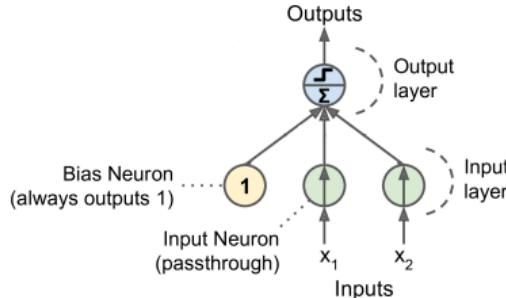
Perceptron Weakness (2/2)



$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad \hat{y} = \text{step}(z), z = w_1x_1 + w_2x_2 + b$$

$$J(w) = \frac{1}{4} \sum_{x \in X} (\hat{y}(x) - y(x))^2$$

Perceptron Weakness (2/2)

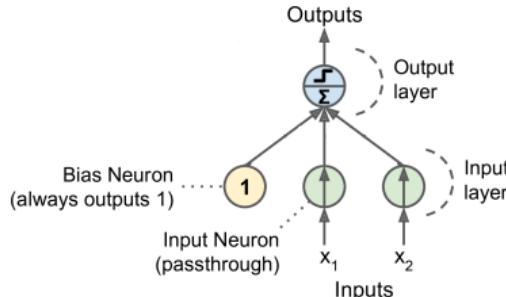


$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad \hat{y} = \text{step}(z), z = w_1x_1 + w_2x_2 + b$$

$$J(w) = \frac{1}{4} \sum_{x \in X} (\hat{y}(x) - y(x))^2$$

- If we minimize $J(w)$, we obtain $w_1 = 0$, $w_2 = 0$, and $b = \frac{1}{2}$.

Perceptron Weakness (2/2)



$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad \hat{y} = \text{step}(z), z = w_1x_1 + w_2x_2 + b$$

$$J(w) = \frac{1}{4} \sum_{x \in X} (\hat{y}(x) - y(x))^2$$

- ▶ If we minimize $J(w)$, we obtain $w_1 = 0$, $w_2 = 0$, and $b = \frac{1}{2}$.
- ▶ But, the model outputs 0.5 everywhere.



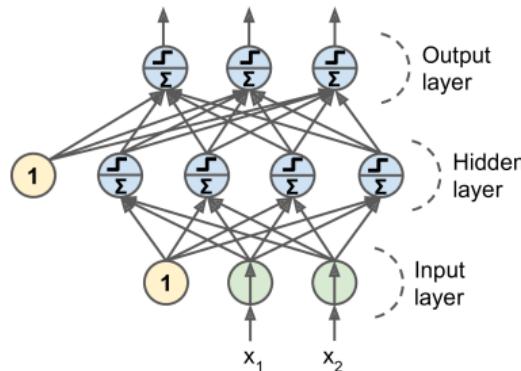
Multi-Layer Perceptron (MLP)

- ▶ The **limitations** of Perceptrons can be eliminated by **stacking multiple Perceptrons**.
- ▶ The resulting network is called a **Multi-Layer Perceptron (MLP)** or **deep feedforward neural network**.

Feedforward Neural Network Architecture

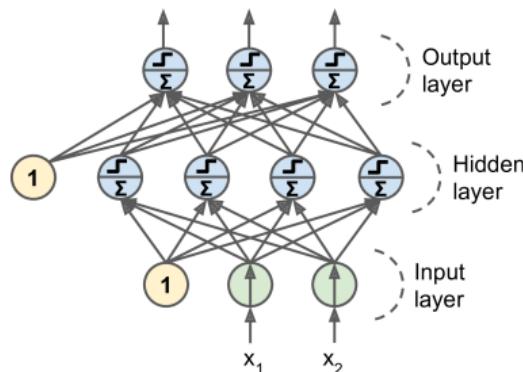
- ▶ A feedforward neural network is composed of:

- One **input layer**
- One or more **hidden layers**
- One final **output layer**



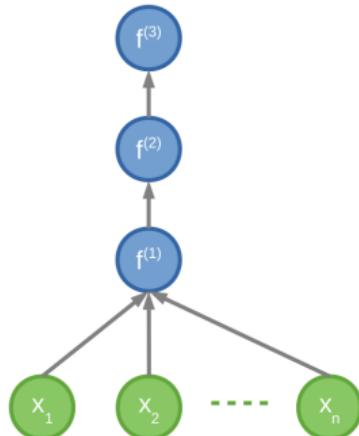
Feedforward Neural Network Architecture

- ▶ A feedforward neural network is composed of:
 - One **input layer**
 - One or more **hidden layers**
 - One final **output layer**
- ▶ Every layer except the output layer includes a **bias neuron** and is **fully connected** to the **next layer**.



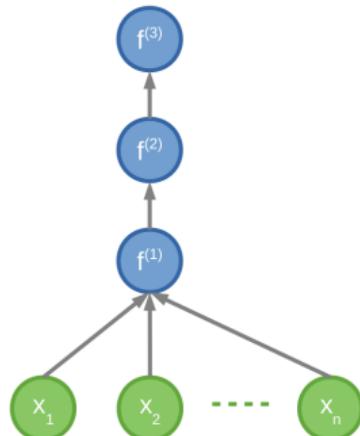
How Does it Work?

- ▶ The model is associated with a directed acyclic graph describing how the functions are composed together.



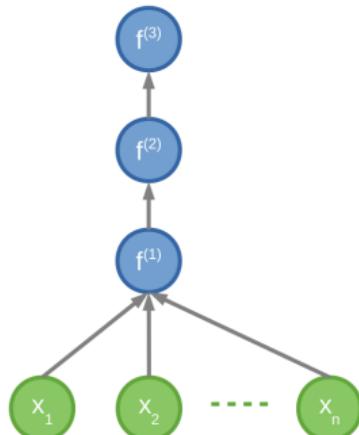
How Does it Work?

- ▶ The model is associated with a directed acyclic graph describing how the functions are composed together.
- ▶ E.g., assume a network with just a single neuron in each layer.
- ▶ Also assume we have three functions $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$ connected in a chain: $\hat{y} = f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$



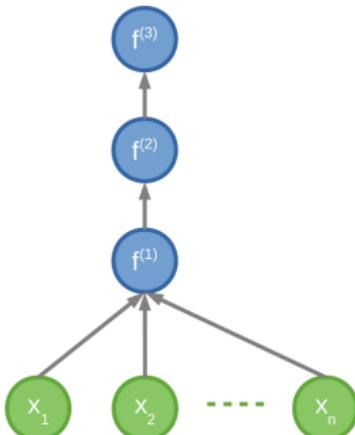
How Does it Work?

- ▶ The model is associated with a directed acyclic graph describing how the functions are composed together.
- ▶ E.g., assume a network with just a single neuron in each layer.
- ▶ Also assume we have three functions $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$ connected in a chain: $\hat{y} = f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$
- ▶ $f^{(1)}$ is called the first layer of the network.
- ▶ $f^{(2)}$ is called the second layer, and so on.

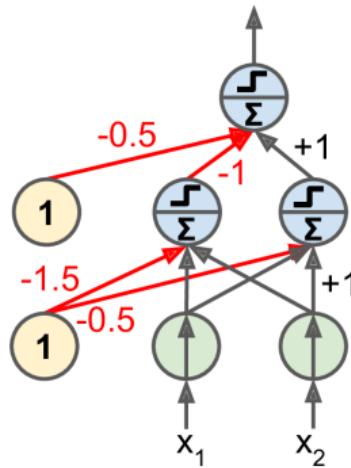


How Does it Work?

- ▶ The model is associated with a directed acyclic graph describing how the functions are composed together.
- ▶ E.g., assume a network with just a single neuron in each layer.
- ▶ Also assume we have three functions $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$ connected in a chain: $\hat{y} = f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$
- ▶ $f^{(1)}$ is called the first layer of the network.
- ▶ $f^{(2)}$ is called the second layer, and so on.
- ▶ The length of the chain gives the depth of the model.

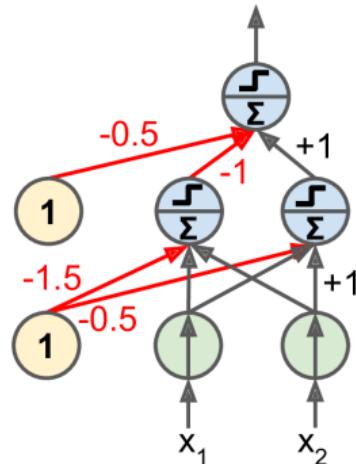


XOR with Feedforward Neural Network (1/3)



$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad W_x = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad b_x = \begin{bmatrix} -1.5 \\ -0.5 \end{bmatrix}$$

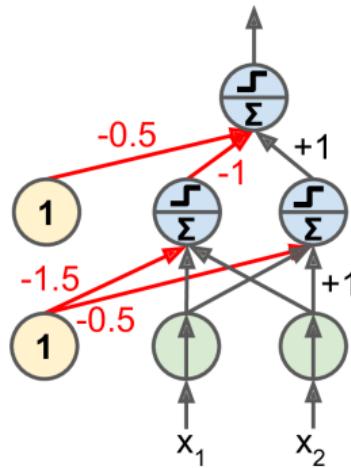
XOR with Feedforward Neural Network (2/3)



$$\text{out}_h = XW_x^T + b_x = \begin{bmatrix} -1.5 & -0.5 \\ -0.5 & 0.5 \\ -0.5 & 0.5 \\ 0.5 & 1.5 \end{bmatrix} \quad h = \text{step}(\text{out}_h) = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

$$w_h = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad b_h = -0.5$$

XOR with Feedforward Neural Network (3/3)



$$\text{out} = \mathbf{w}_h^T \mathbf{h} + b_h = \begin{bmatrix} -0.5 \\ 0.5 \\ 0.5 \\ -0.5 \end{bmatrix} \quad \text{step}(\text{out}) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$



How to Learn Model Parameters W ?



Feedforward Neural Network - Cost Function

- ▶ We use the **cross-entropy** (minimizing the negative log-likelihood) between the training data y and the model's predictions \hat{y} as the **cost function**.

$$\text{cost}(y, \hat{y}) = - \sum_j y_j \log(\hat{y}_j)$$

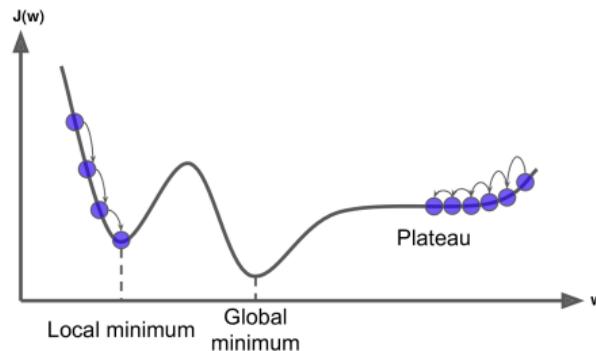


Gradient-Based Learning (1/2)

- ▶ The **most significant difference** between the **linear models** we have seen so far and **feedforward neural network**?

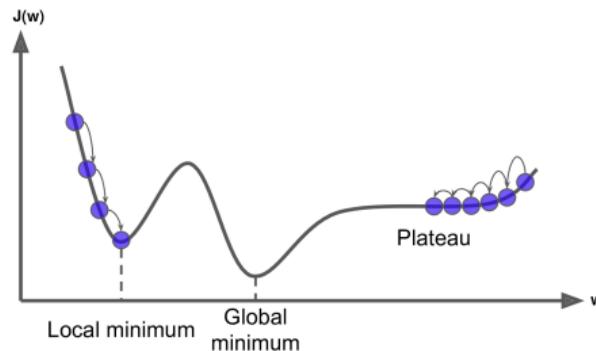
Gradient-Based Learning (1/2)

- ▶ The **most significant difference** between the **linear models** we have seen so far and **feedforward neural network**?
- ▶ The **non-linearity** of a neural network causes its **cost functions** to become **non-convex**.



Gradient-Based Learning (1/2)

- ▶ The **most significant difference** between the **linear models** we have seen so far and **feedforward neural network**?
- ▶ The **non-linearity** of a neural network causes its **cost functions** to become **non-convex**.
- ▶ Linear models, with **convex cost function**, **guarantee** to find **global minimum**.
 - Convex optimization converges starting from **any initial parameters**.





Gradient-Based Learning (2/2)

- ▶ Stochastic gradient descent applied to **non-convex cost functions** has no such convergence guarantee.



Gradient-Based Learning (2/2)

- ▶ Stochastic gradient descent applied to **non-convex cost functions** has no such convergence guarantee.
- ▶ It is **sensitive** to the values of the **initial parameters**.



Gradient-Based Learning (2/2)

- ▶ Stochastic gradient descent applied to **non-convex cost functions** has no such convergence guarantee.
- ▶ It is **sensitive** to the values of the **initial parameters**.
- ▶ For feedforward neural networks, it is important to **initialize** all **weights** to small random values.

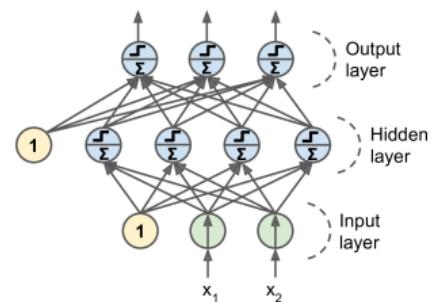


Gradient-Based Learning (2/2)

- ▶ Stochastic gradient descent applied to **non-convex cost functions** has no such convergence guarantee.
- ▶ It is **sensitive** to the values of the **initial parameters**.
- ▶ For feedforward neural networks, it is important to **initialize** all **weights** to small random values.
- ▶ The **biases** may be **initialized** to zero or to small positive values.

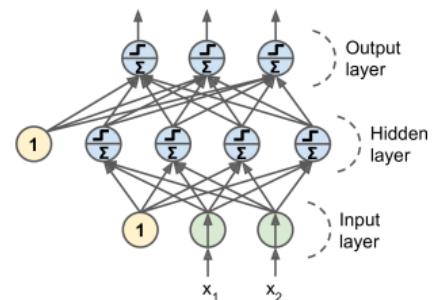
Training Feedforward Neural Networks

- ▶ How to **train** a **feedforward neural network**?



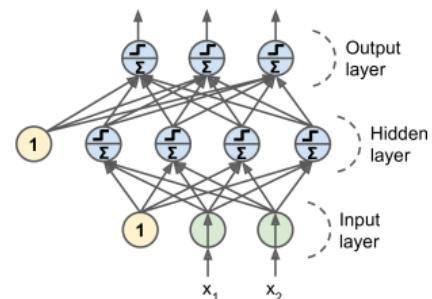
Training Feedforward Neural Networks

- ▶ How to **train** a **feedforward neural network**?
- ▶ For each training instance $x^{(i)}$ the algorithm does the following **steps**:



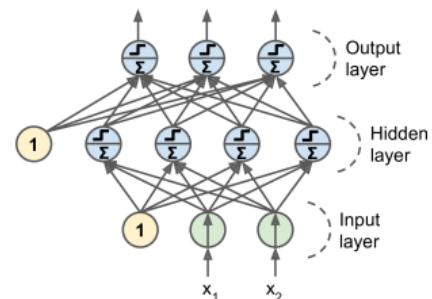
Training Feedforward Neural Networks

- ▶ How to **train** a **feedforward neural network**?
- ▶ For each training instance $x^{(i)}$ the algorithm does the following **steps**:
 1. **Forward pass:** make a **prediction** (compute $\hat{y}^{(i)} = f(x^{(i)})$).



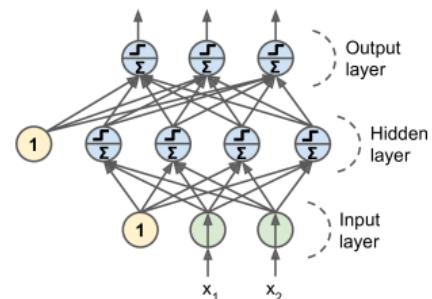
Training Feedforward Neural Networks

- ▶ How to **train** a **feedforward neural network**?
- ▶ For each training instance $x^{(i)}$ the algorithm does the following **steps**:
 1. **Forward pass:** make a **prediction** (compute $\hat{y}^{(i)} = f(x^{(i)})$).
 2. Measure the **error** (compute $\text{cost}(\hat{y}^{(i)}, y^{(i)})$).



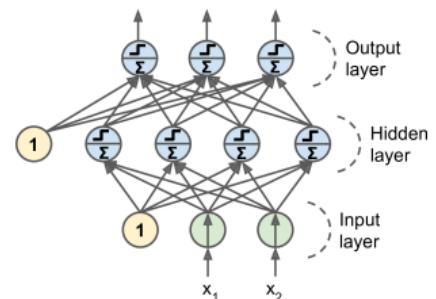
Training Feedforward Neural Networks

- ▶ How to **train** a **feedforward neural network**?
- ▶ For each training instance $x^{(i)}$ the algorithm does the following **steps**:
 1. **Forward pass**: make a **prediction** (compute $\hat{y}^{(i)} = f(x^{(i)})$).
 2. Measure the **error** (compute $\text{cost}(\hat{y}^{(i)}, y^{(i)})$).
 3. **Backward pass**: go through each layer in **reverse** to measure the **error contribution** from each connection.



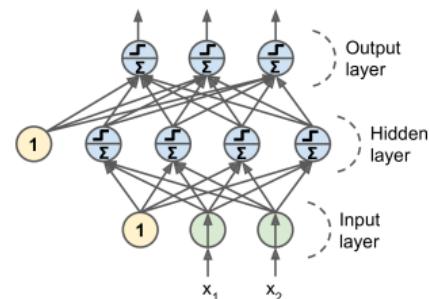
Training Feedforward Neural Networks

- ▶ How to **train** a **feedforward neural network**?
- ▶ For each training instance $x^{(i)}$ the algorithm does the following **steps**:
 1. **Forward pass**: make a **prediction** (compute $\hat{y}^{(i)} = f(x^{(i)})$).
 2. Measure the **error** (compute $\text{cost}(\hat{y}^{(i)}, y^{(i)})$).
 3. **Backward pass**: go through each layer in **reverse** to measure the **error contribution** from **each connection**.
 4. **Tweak the connection weights** to **reduce the error** (update W and b).



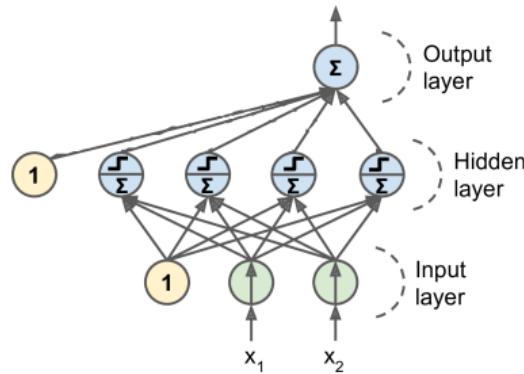
Training Feedforward Neural Networks

- ▶ How to **train** a **feedforward neural network**?
- ▶ For each training instance $x^{(i)}$ the algorithm does the following **steps**:
 1. **Forward pass**: make a **prediction** (compute $\hat{y}^{(i)} = f(x^{(i)})$).
 2. Measure the **error** (compute $\text{cost}(\hat{y}^{(i)}, y^{(i)})$).
 3. **Backward pass**: go through each layer in **reverse** to measure the **error contribution** from **each connection**.
 4. **Tweak the connection weights** to **reduce the error** (update W and b).
- ▶ It's called the **backpropagation** training algorithm



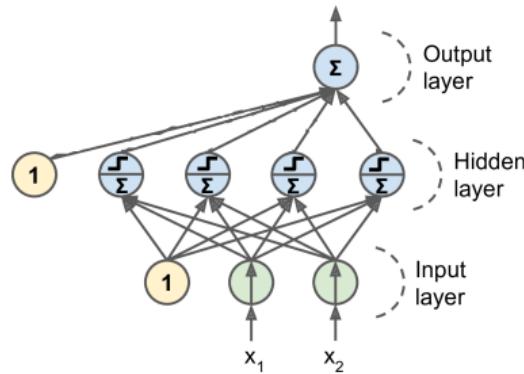
Output Unit (1/3)

- ▶ Linear units in neurons of the output layer.



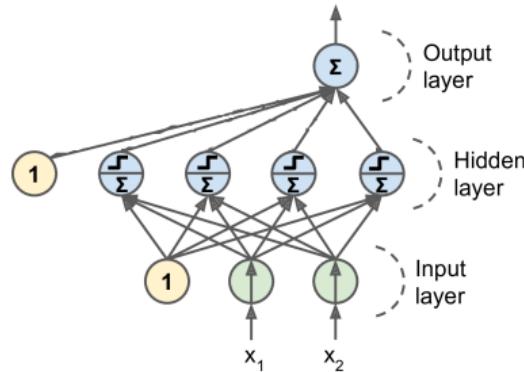
Output Unit (1/3)

- ▶ Linear units in neurons of the output layer.
- ▶ Output function: $\hat{y}_j = w_j^T h + b_j$.



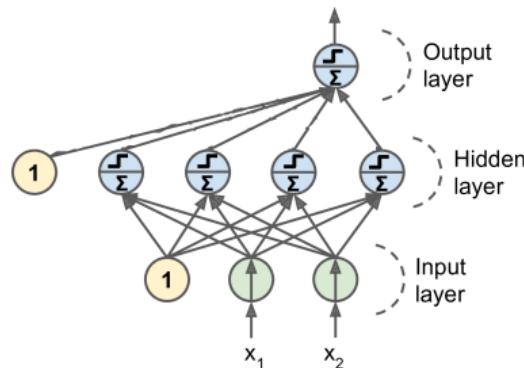
Output Unit (1/3)

- ▶ Linear units in neurons of the output layer.
- ▶ Output function: $\hat{y}_j = w_j^T h + b_j$.
- ▶ Cost function: minimizing the mean squared error.



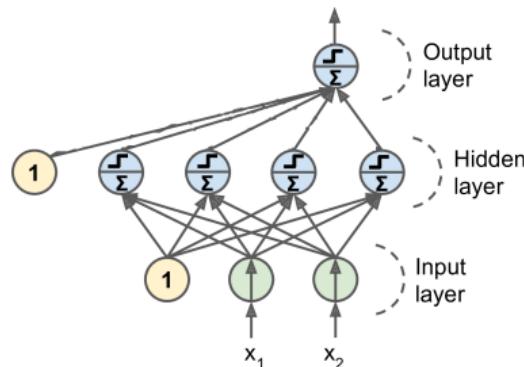
Output Unit (2/3)

- Sigmoid units in neurons of the output layer (binomial classification).



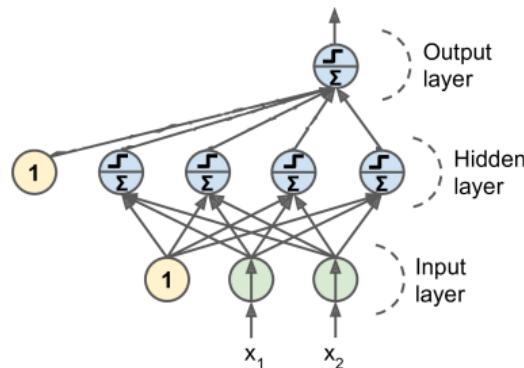
Output Unit (2/3)

- ▶ Sigmoid units in neurons of the output layer (binomial classification).
- ▶ Output function: $\hat{y}_j = \sigma(w_j^T h + b_j)$.



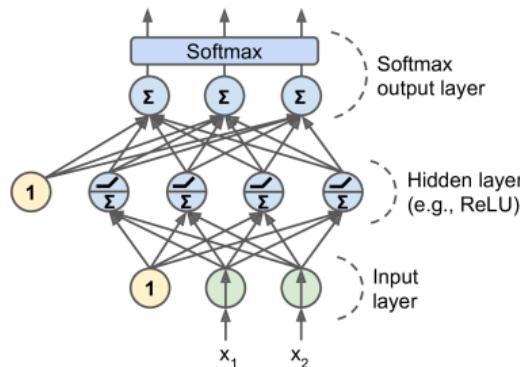
Output Unit (2/3)

- ▶ Sigmoid units in neurons of the output layer (binomial classification).
- ▶ Output function: $\hat{y}_j = \sigma(w_j^T h + b_j)$.
- ▶ Cost function: minimizing the cross-entropy.



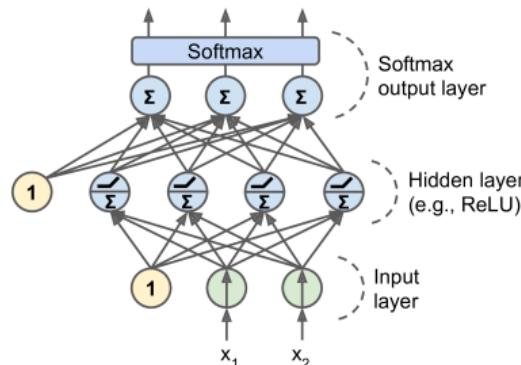
Output Unit (3/3)

- Softmax units in neurons of the output layer (multinomial classification).



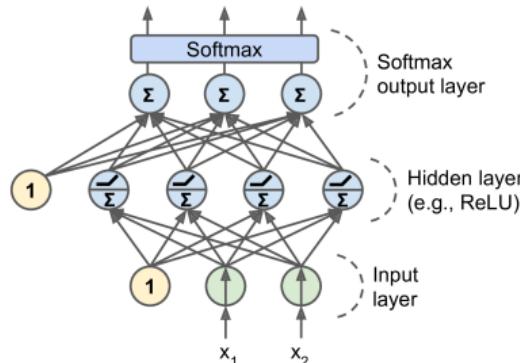
Output Unit (3/3)

- ▶ Softmax units in neurons of the output layer (multinomial classification).
- ▶ Output function: $\hat{y}_j = \text{softmax}(w_j^T h + b_j)$.



Output Unit (3/3)

- ▶ Softmax units in neurons of the output layer (multinomial classification).
- ▶ Output function: $\hat{y}_j = \text{softmax}(w_j^T h + b_j)$.
- ▶ Cost function: minimizing the cross-entropy.



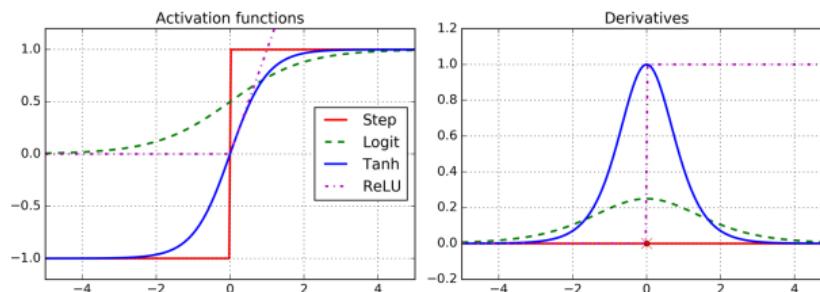


Hidden Units

- ▶ In order for the **backpropagation** algorithm to work properly, we need to **replace the step function** with **other activation functions**. **Why?**

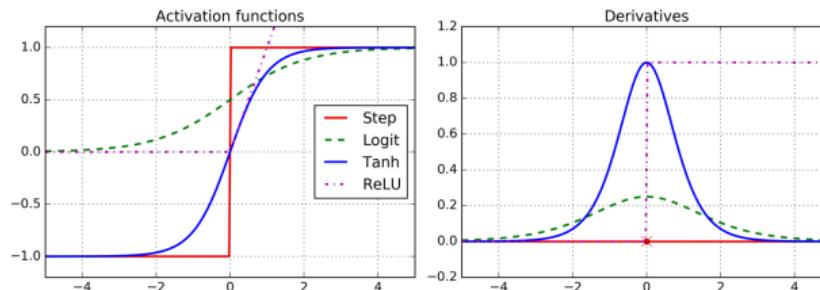
Hidden Units

- ▶ In order for the **backpropagation** algorithm to work properly, we need to **replace the step function** with **other activation functions**. **Why?**
- ▶ Alternative activation functions:



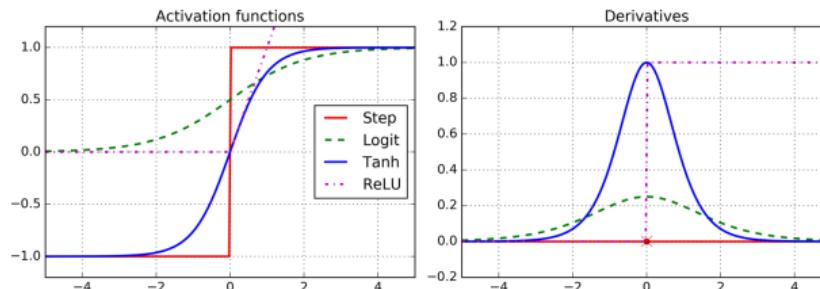
Hidden Units

- In order for the **backpropagation** algorithm to work properly, we need to **replace the step function** with **other activation functions**. **Why?**
- Alternative activation functions:
 - Logistic function (sigmoid): $\sigma(z) = \frac{1}{1+e^{-z}}$



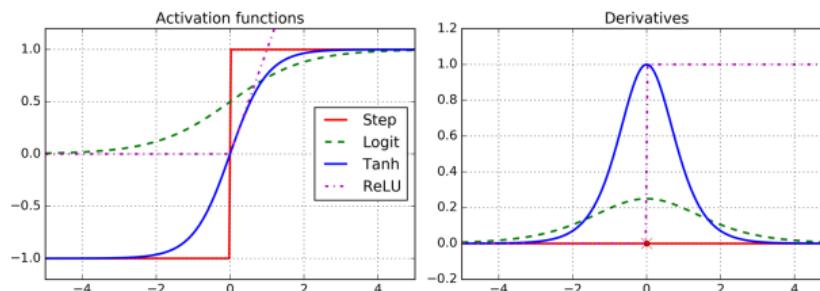
Hidden Units

- In order for the **backpropagation** algorithm to work properly, we need to **replace the step function** with **other activation functions**. **Why?**
- Alternative activation functions:
 - Logistic function (sigmoid): $\sigma(z) = \frac{1}{1+e^{-z}}$
 - Hyperbolic tangent function: $\tanh(z) = 2\sigma(2z) - 1$



Hidden Units

- ▶ In order for the **backpropagation** algorithm to work properly, we need to **replace the step function** with **other activation functions**. **Why?**
- ▶ Alternative activation functions:
 1. Logistic function (sigmoid): $\sigma(z) = \frac{1}{1+e^{-z}}$
 2. Hyperbolic tangent function: $\tanh(z) = 2\sigma(2z) - 1$
 3. Rectified linear units (ReLUs): $\text{ReLU}(z) = \max(0, z)$

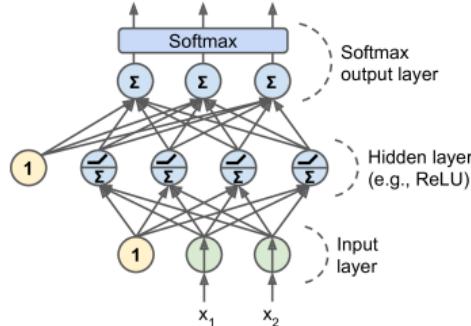




Feedforward Network in TensorFlow



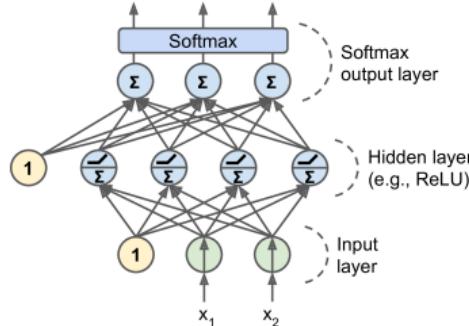
Feedforward Network in TensorFlow



```
n_output = 3
n_hidden = 4
n_features = 2

model = keras.models.Sequential()
model.add(keras.layers.Dense(n_hidden, input_shape=(n_features,), activation="relu"))
model.add(keras.layers.Dense(n_output, activation="softmax"))
```

Feedforward Network in TensorFlow



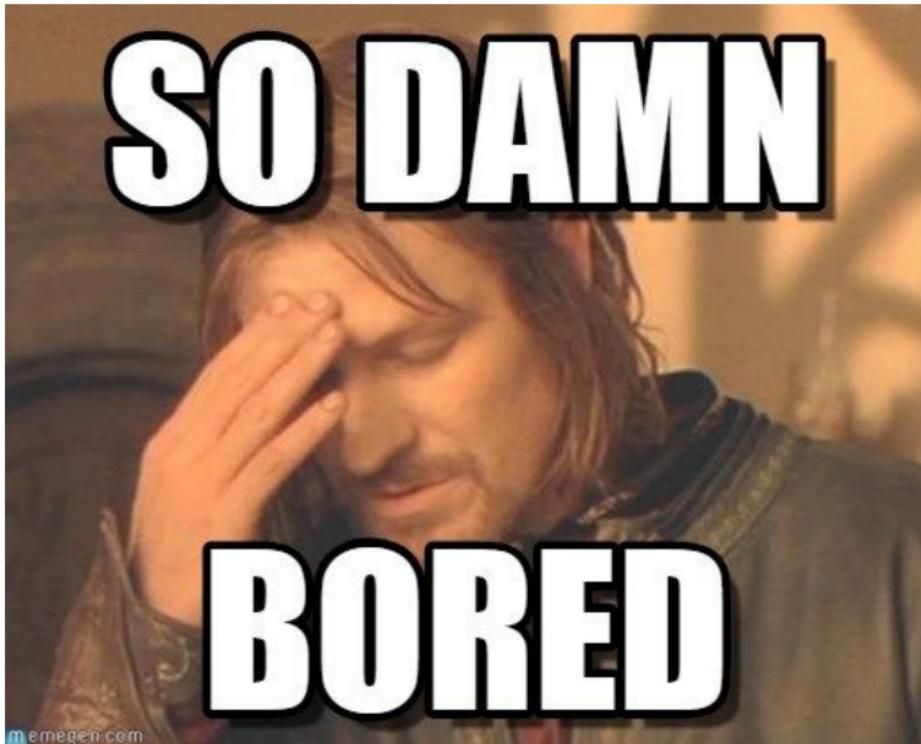
```
n_output = 3
n_hidden = 4
n_features = 2

model = keras.models.Sequential()
model.add(keras.layers.Dense(n_hidden, input_shape=(n_features,), activation="relu"))
model.add(keras.layers.Dense(n_output, activation="softmax"))
```

```
model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd", metrics=["accuracy"])
model.fit(X_train, y_train, epochs=30)
```



Dive into Backpropagation Algorithm



[<https://i.pinimg.com/originals/82/d9/2c/82d92c2c15c580c2b2fce65a83fe0b3f.jpg>]



Chain Rule of Calculus (1/2)

- ▶ Assume $x \in \mathbb{R}$, and two functions f and g , and also assume $y = g(x)$ and $z = f(y) = f(g(x))$.



Chain Rule of Calculus (1/2)

- ▶ Assume $x \in \mathbb{R}$, and two functions f and g , and also assume $y = g(x)$ and $z = f(y) = f(g(x))$.
- ▶ The **chain rule of calculus** is used to compute the **derivatives of functions**, e.g., z , formed by **composing other functions**, e.g., g .



Chain Rule of Calculus (1/2)

- ▶ Assume $x \in \mathbb{R}$, and two functions f and g , and also assume $y = g(x)$ and $z = f(y) = f(g(x))$.
- ▶ The **chain rule of calculus** is used to compute the **derivatives of functions**, e.g., z , formed by **composing other functions**, e.g., g .
- ▶ Then the **chain rule** states that $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$



Chain Rule of Calculus (1/2)

- ▶ Assume $x \in \mathbb{R}$, and two functions f and g , and also assume $y = g(x)$ and $z = f(y) = f(g(x))$.
- ▶ The **chain rule of calculus** is used to compute the **derivatives of functions**, e.g., z , formed by **composing other functions**, e.g., g .
- ▶ Then the **chain rule** states that $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$
- ▶ Example:

$$z = f(y) = 5y^4 \text{ and } y = g(x) = x^3 + 7$$



Chain Rule of Calculus (1/2)

- ▶ Assume $x \in \mathbb{R}$, and two functions f and g , and also assume $y = g(x)$ and $z = f(y) = f(g(x))$.
- ▶ The **chain rule of calculus** is used to compute the **derivatives of functions**, e.g., z , formed by **composing other functions**, e.g., g .
- ▶ Then the **chain rule** states that $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$
- ▶ Example:

$$z = f(y) = 5y^4 \text{ and } y = g(x) = x^3 + 7$$

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$



Chain Rule of Calculus (1/2)

- ▶ Assume $x \in \mathbb{R}$, and two functions f and g , and also assume $y = g(x)$ and $z = f(y) = f(g(x))$.
- ▶ The **chain rule of calculus** is used to compute the **derivatives of functions**, e.g., z , formed by **composing other functions**, e.g., g .
- ▶ Then the **chain rule** states that $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$
- ▶ Example:

$$z = f(y) = 5y^4 \text{ and } y = g(x) = x^3 + 7$$

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

$$\frac{dz}{dy} = 20y^3 \text{ and } \frac{dy}{dx} = 3x^2$$



Chain Rule of Calculus (1/2)

- ▶ Assume $x \in \mathbb{R}$, and two functions f and g , and also assume $y = g(x)$ and $z = f(y) = f(g(x))$.
- ▶ The **chain rule of calculus** is used to compute the **derivatives of functions**, e.g., z , formed by **composing other functions**, e.g., g .
- ▶ Then the **chain rule** states that $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$
- ▶ Example:

$$z = f(y) = 5y^4 \text{ and } y = g(x) = x^3 + 7$$

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

$$\frac{dz}{dy} = 20y^3 \text{ and } \frac{dy}{dx} = 3x^2$$

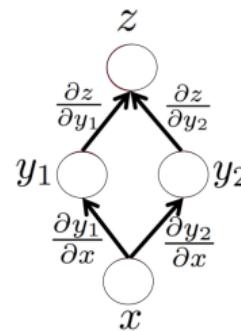
$$\frac{dz}{dx} = 20y^3 \times 3x^2 = 20(x^3 + 7) \times 3x^2$$

Chain Rule of Calculus (2/2)

- ▶ Two paths chain rule.

$z = f(y_1, y_2)$ where $y_1 = g(x)$ and $y_2 = h(x)$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2} \frac{\partial y_2}{\partial x}$$



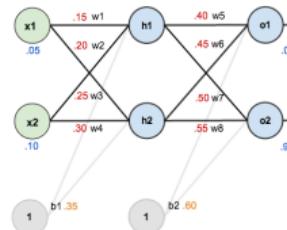


Backpropagation

- ▶ Backpropagation training algorithm for MLPs
- ▶ The algorithm repeats the following steps:
 1. Forward pass
 2. Backward pass

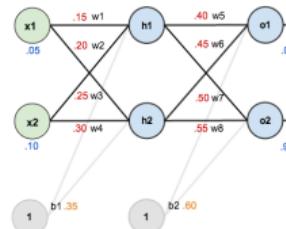
Backpropagation - Forward Pass

- ▶ Calculates outputs given input patterns.



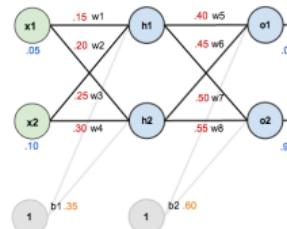
Backpropagation - Forward Pass

- ▶ Calculates outputs given input patterns.
- ▶ For each training instance



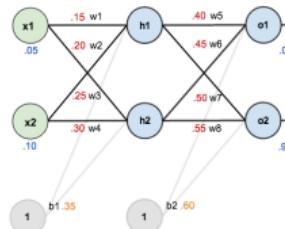
Backpropagation - Forward Pass

- ▶ Calculates outputs given input patterns.
- ▶ For each training instance
 - Feeds it to the network and computes the output of every neuron in each consecutive layer.



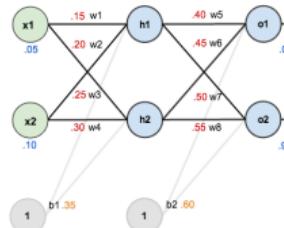
Backpropagation - Forward Pass

- ▶ Calculates outputs given input patterns.
- ▶ For each training instance
 - Feeds it to the network and computes the output of every neuron in each consecutive layer.
 - Measures the network's output error (i.e., the difference between the true and the predicted output of the network)



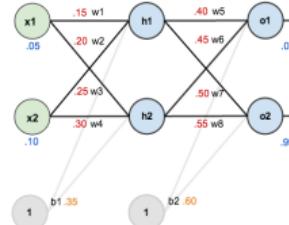
Backpropagation - Forward Pass

- ▶ Calculates outputs given input patterns.
- ▶ For each training instance
 - Feeds it to the network and computes the output of every neuron in each consecutive layer.
 - Measures the network's output error (i.e., the difference between the true and the predicted output of the network)
 - Computes how much each neuron in the last hidden layer contributed to each output neuron's error.



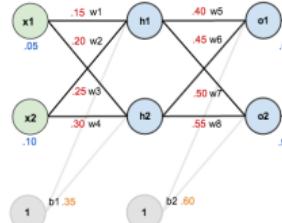
Backpropagation - Backward Pass

- ▶ Updates weights by calculating gradients.



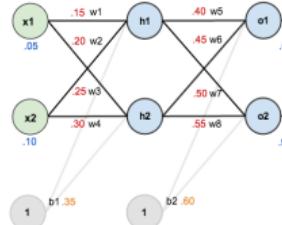
Backpropagation - Backward Pass

- ▶ Updates weights by calculating gradients.
- ▶ Measures how much of these error contributions came from each neuron in the previous hidden layer
 - Proceeds until the algorithm reaches the input layer.



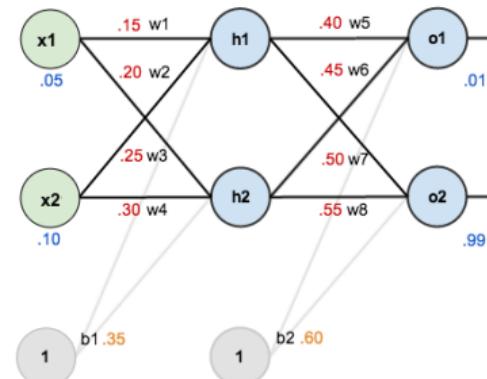
Backpropagation - Backward Pass

- ▶ Updates weights by calculating gradients.
- ▶ Measures how much of these error contributions came from each neuron in the previous hidden layer
 - Proceeds until the algorithm reaches the input layer.
- ▶ The last step is the gradient descent step on all the connection weights in the network, using the error gradients measured earlier.



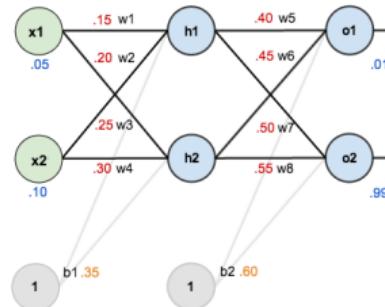
Backpropagation Example

- ▶ Two **inputs**, two **hidden**, and two **output** neurons.
- ▶ Bias in **hidden** and **output** neurons.
- ▶ Logistic activation in all the neurons.
- ▶ Squared error function as the cost function.



Backpropagation - Forward Pass (1/3)

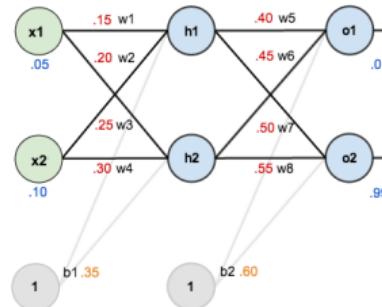
- ▶ Compute the **output** of the **hidden layer**



$$\text{net}_{h1} = w_1 x_1 + w_2 x_2 + b_1 = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 = 0.3775$$

Backpropagation - Forward Pass (1/3)

- ▶ Compute the **output** of the **hidden layer**



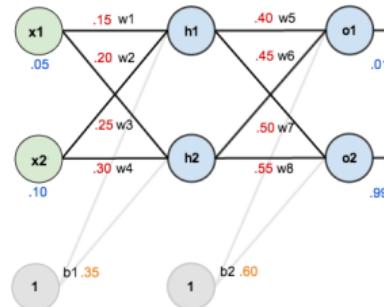
$$\text{net}_{h1} = w_1 x_1 + w_2 x_2 + b_1 = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 = 0.3775$$

$$\text{out}_{h1} = \frac{1}{1 + e^{\text{net}_{h1}}} = \frac{1}{1 + e^{0.3775}} = 0.59327$$

$$\text{out}_{h2} = 0.59688$$

Backpropagation - Forward Pass (2/3)

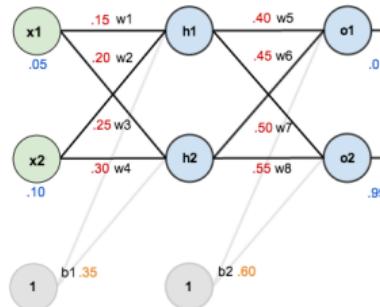
- ▶ Compute the **output** of the **output layer**



$$\text{net}_{o1} = w_5 \text{out}_{h1} + w_6 \text{out}_{h2} + b_2 = 0.4 \times 0.59327 + 0.45 \times 0.59688 + 0.6 = 1.1059$$

Backpropagation - Forward Pass (2/3)

- ▶ Compute the **output** of the **output layer**



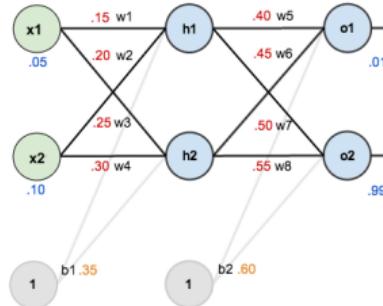
$$\text{net}_{o1} = w_5 \text{out}_{h1} + w_6 \text{out}_{h2} + b_2 = 0.4 \times 0.59327 + 0.45 \times 0.59688 + 0.6 = 1.1059$$

$$\text{out}_{o1} = \frac{1}{1 + e^{\text{net}_{o1}}} = \frac{1}{1 + e^{1.1059}} = 0.75136$$

$$\text{out}_{o2} = 0.77292$$

Backpropagation - Forward Pass (3/3)

- ▶ Calculate the **error** for each output



$$E_{o1} = \frac{1}{2}(\text{target}_{o1} - \text{output}_{o1})^2 = \frac{1}{2}(0.01 - 0.75136)^2 = 0.27481$$

$$E_{o2} = 0.02356$$

$$E_{\text{total}} = \sum \frac{1}{2}(\text{target} - \text{output})^2 = E_{o1} + E_{o2} = 0.27481 + 0.02356 = 0.29837$$

This class is boring...

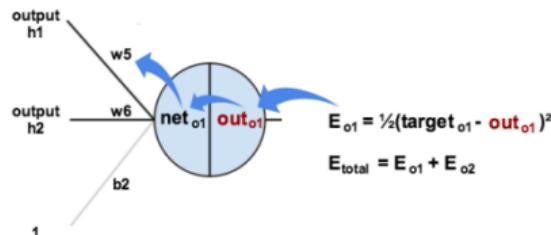


can we learn about dragons?

[<http://marimancusi.blogspot.com/2015/09/are-you-book-dragon.html>]

Backpropagation - Backward Pass - Output Layer (1/6)

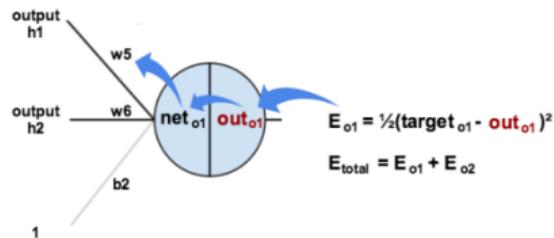
- ▶ Consider w_5
- ▶ We want to know how much a **change** in w_5 affects the **total error** ($\frac{\partial E_{\text{total}}}{\partial w_5}$)
- ▶ Applying the **chain rule**



$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial w_5}$$

Backpropagation - Backward Pass - Output Layer (2/6)

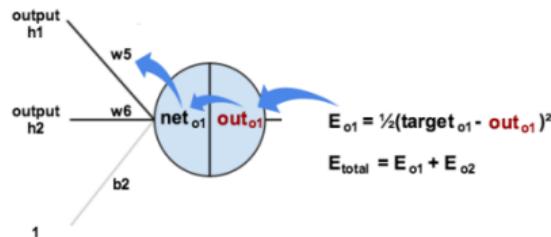
- ▶ First, how much does the total error change with respect to the output? ($\frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}}$)



$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial w_5}$$

Backpropagation - Backward Pass - Output Layer (2/6)

- First, how much does the total error change with respect to the output? ($\frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}}$)



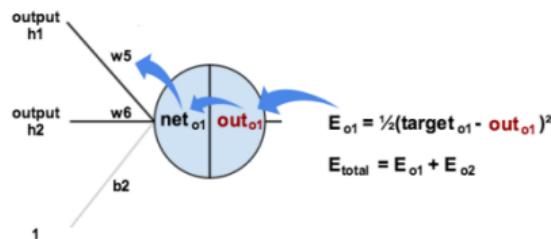
$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial w_5}$$

$$E_{\text{total}} = \frac{1}{2}(\text{target}_{o1} - \text{out}_{o1})^2 + \frac{1}{2}(\text{target}_{o2} - \text{out}_{o2})^2$$

$$\frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} = -2 \frac{1}{2}(\text{target}_{o1} - \text{out}_{o1}) = -(0.01 - 0.75136) = 0.74136$$

Backpropagation - Backward Pass - Output Layer (3/6)

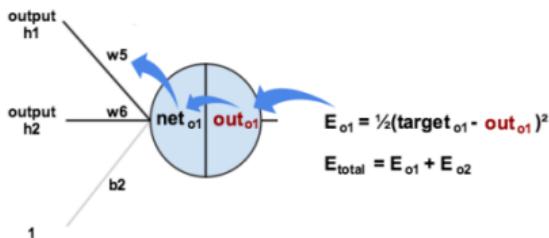
- ▶ Next, how much does the out_{o1} change with respect to its total input net_{o1} ?
 $(\frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}})$



$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial w_5}$$

Backpropagation - Backward Pass - Output Layer (3/6)

- ▶ Next, how much does the out_{o1} change with respect to its total input net_{o1} ?
 $(\frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}})$



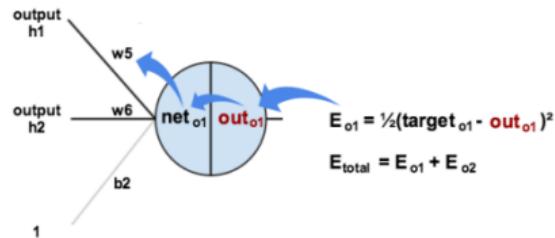
$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial w_5}$$

$$\text{out}_{o1} = \frac{1}{1 + e^{-\text{net}_{o1}}}$$

$$\frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} = \text{out}_{o1}(1 - \text{out}_{o1}) = 0.75136(1 - 0.75136) = 0.18681$$

Backpropagation - Backward Pass - Output Layer (4/6)

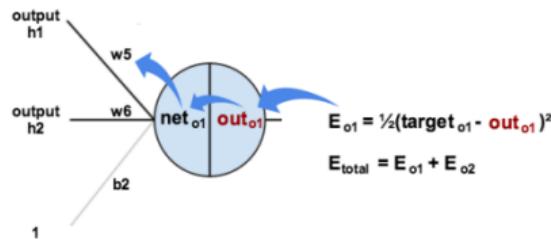
- ▶ Finally, how much does the total net_{o1} change with respect to w_5 ? ($\frac{\partial \text{net}_{o1}}{\partial w_5}$)



$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial w_5}$$

Backpropagation - Backward Pass - Output Layer (4/6)

- Finally, how much does the total net_{o1} change with respect to w_5 ? ($\frac{\partial \text{net}_{o1}}{\partial w_5}$)



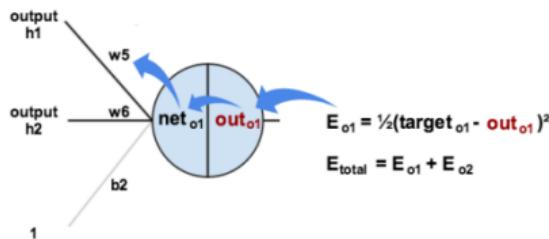
$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial w_5}$$

$$\text{net}_{o1} = w_5 \times \text{out}_{h1} + w_6 \times \text{out}_{h2} + b_2$$

$$\frac{\partial \text{net}_{o1}}{\partial w_5} = \text{out}_{h1} = 0.59327$$

Backpropagation - Backward Pass - Output Layer (5/6)

- ▶ Putting it all together:

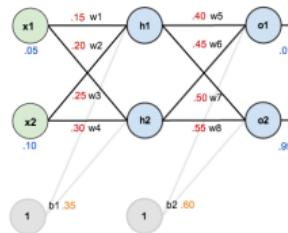


$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136 \times 0.18681 \times 0.59327 = 0.08216$$

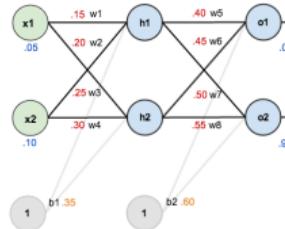
Backpropagation - Backward Pass - Output Layer (6/6)

- To decrease the error, we subtract this value from the current weight.



Backpropagation - Backward Pass - Output Layer (6/6)

- ▶ To decrease the error, we subtract this value from the **current weight**.
- ▶ We assume that the **learning rate** is $\eta = 0.5$.

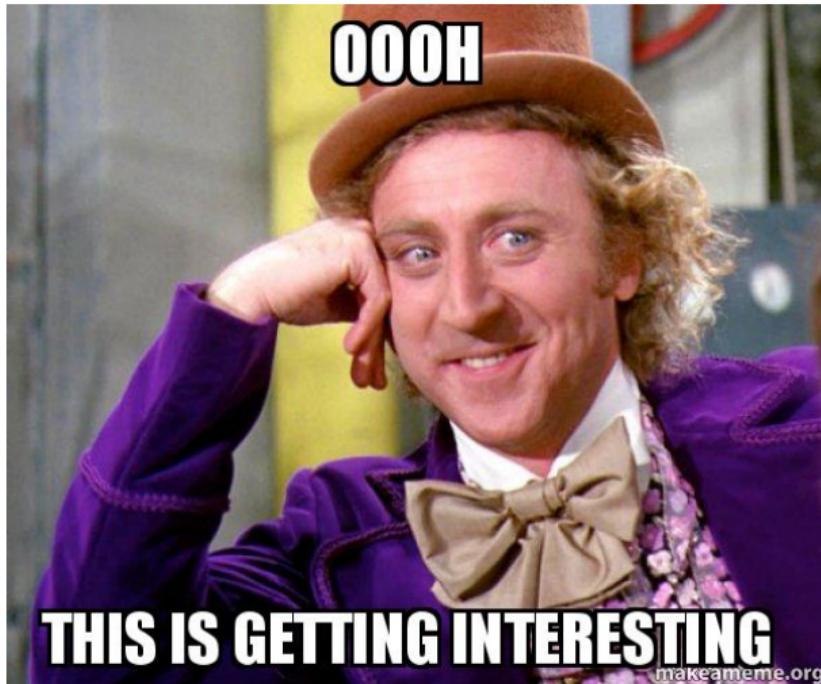


$$w_5^{(\text{next})} = w_5 - \eta \times \frac{\partial E_{\text{total}}}{\partial w_5} = 0.4 - 0.5 \times 0.08216 = 0.35891$$

$$w_6^{(\text{next})} = 0.40866$$

$$w_7^{(\text{next})} = 0.5113$$

$$w_8^{(\text{next})} = 0.56137$$

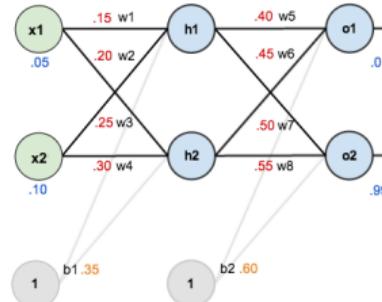


[<https://makeameme.org/meme/oooh-this>]

Backpropagation - Backward Pass - Hidden Layer (1/8)

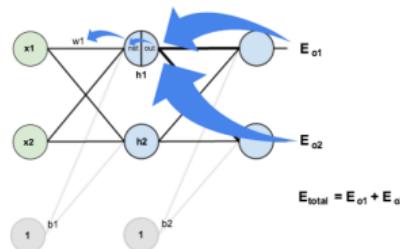
- ▶ Continue the **backwards pass** by calculating new values for w_1 , w_2 , w_3 , and w_4 .
- ▶ For w_1 we have:

$$\frac{\partial E_{\text{total}}}{\partial w_1} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} \times \frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} \times \frac{\partial \text{net}_{h1}}{\partial w_1}$$



Backpropagation - Backward Pass - Hidden Layer (2/8)

- Here, the output of each hidden layer neuron contributes to the output of multiple output neurons.
- E.g., out_{h1} affects both out_{o1} and out_{o2} , so $\frac{\partial E_{total}}{\partial out_{h1}}$ needs to take into consideration its effect on the both output neurons.

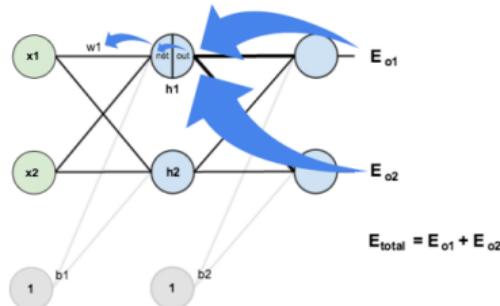


$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} \times \frac{\partial out_{h1}}{\partial net_{h1}} \times \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

Backpropagation - Backward Pass - Hidden Layer (3/8)

- ▶ Starting with $\frac{\partial E_{o1}}{\partial \text{out}_{h1}}$



$$\frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} = \frac{\partial E_{o1}}{\partial \text{out}_{h1}} + \frac{\partial E_{o2}}{\partial \text{out}_{h1}}$$

$$\frac{\partial E_{o1}}{\partial \text{out}_{h1}} = \frac{\partial E_{o1}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial \text{out}_{h1}}$$

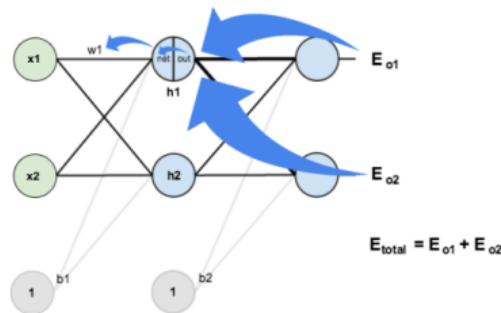
$$\frac{\partial E_{o1}}{\partial \text{out}_{o1}} = 0.74136, \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} = 0.18681$$

$$\text{net}_{o1} = w_5 \times \text{out}_{h1} + w_6 \times \text{out}_{h2} + b_2$$

$$\frac{\partial \text{net}_{o1}}{\partial \text{out}_{h1}} = w_5 = 0.40$$

Backpropagation - Backward Pass - Hidden Layer (4/8)

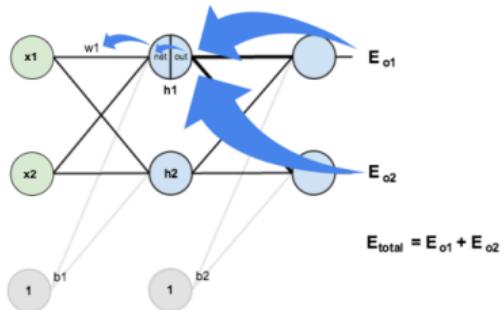
- ▶ Plugging them together.



$$\frac{\partial E_{o1}}{\partial \text{out}_{h1}} = \frac{\partial E_{o1}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial \text{out}_{h1}} = 0.74136 \times 0.18681 \times 0.40 = 0.0554$$
$$\frac{\partial E_{o2}}{\partial \text{out}_{h1}} = -0.01905$$

Backpropagation - Backward Pass - Hidden Layer (4/8)

- ▶ Plugging them together.



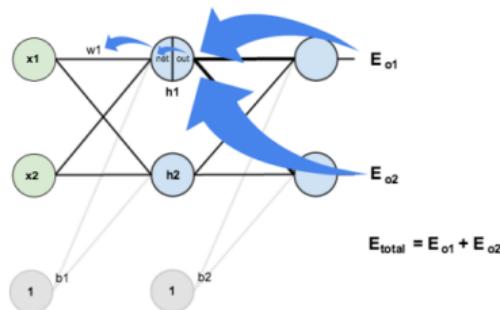
$$\frac{\partial E_{o1}}{\partial \text{out}_{h1}} = \frac{\partial E_{o1}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \times \frac{\partial \text{net}_{o1}}{\partial \text{out}_{h1}} = 0.74136 \times 0.18681 \times 0.40 = 0.0554$$

$$\frac{\partial E_{o2}}{\partial \text{out}_{h1}} = -0.01905$$

$$\frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} = \frac{\partial E_{o1}}{\partial \text{out}_{h1}} + \frac{\partial E_{o2}}{\partial \text{out}_{h1}} = 0.0554 + -0.01905 = 0.03635$$

Backpropagation - Backward Pass - Hidden Layer (5/8)

- Now we need to figure out $\frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}}$.



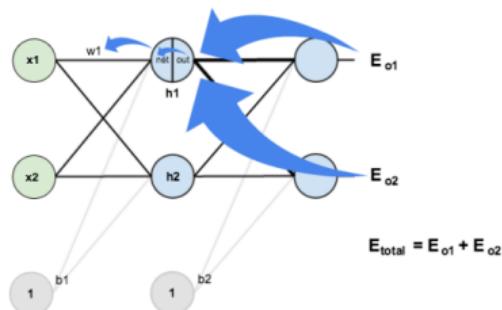
$$\frac{\partial E_{\text{total}}}{\partial w_1} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} \times \frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} \times \frac{\partial \text{net}_{h1}}{\partial w_1}$$

$$\text{out}_{h1} = \frac{1}{1 + e^{-\text{net}_{h1}}}$$

$$\frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} = \text{out}_{h1}(1 - \text{out}_{h1}) = 0.59327(1 - 0.59327) = 0.2413$$

Backpropagation - Backward Pass - Hidden Layer (6/8)

- And then $\frac{\partial \text{net}_{h1}}{\partial w_1}$.



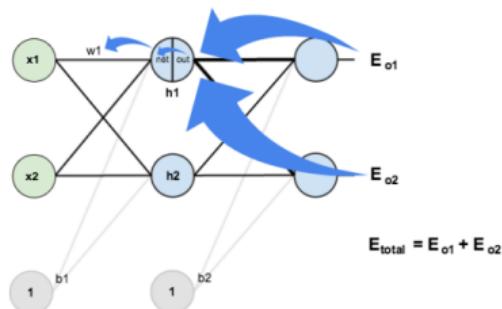
$$\frac{\partial E_{\text{total}}}{\partial w_1} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} \times \frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} \times \frac{\partial \text{net}_{h1}}{\partial w_1}$$

$$\text{net}_{h1} = w_1 x_1 + w_2 x_2 + b_1$$

$$\frac{\partial \text{net}_{h1}}{\partial w_1} = x_1 = 0.05$$

Backpropagation - Backward Pass - Hidden Layer (7/8)

- ▶ Putting it all together.

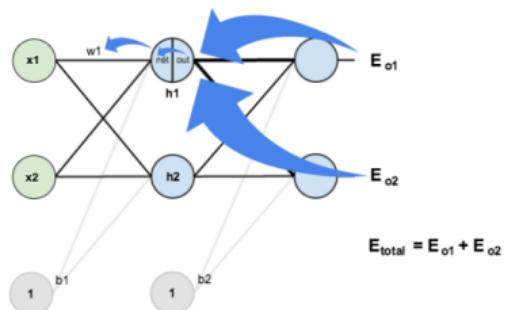


$$\frac{\partial E_{\text{total}}}{\partial w_1} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} \times \frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} \times \frac{\partial \text{net}_{h1}}{\partial w_1}$$

$$\frac{\partial E_{\text{total}}}{\partial w_1} = 0.03635 \times 0.2413 \times 0.05 = 0.00043$$

Backpropagation - Backward Pass - Hidden Layer (8/8)

- ▶ We can now update w_1 .
- ▶ Repeating this for w_2 , w_3 , and w_4 .



$$w_1^{(\text{next})} = w_1 - \eta \times \frac{\partial E_{\text{total}}}{\partial w_1} = 0.15 - 0.5 \times 0.00043 = 0.14978$$

$$w_2^{(\text{next})} = 0.19956$$

$$w_3^{(\text{next})} = 0.24975$$

$$w_4^{(\text{next})} = 0.2995$$



Summary



Summary

- ▶ LTU
- ▶ Perceptron
- ▶ Perceptron weakness
- ▶ MLP and feedforward neural network
- ▶ Gradient-based learning
- ▶ Backpropagation: forward pass and backward pass
- ▶ Output unit: linear, sigmoid, softmax
- ▶ Hidden units: sigmoid, tanh, relu



Reference

- ▶ Ian Goodfellow et al., Deep Learning (Ch. 6)
- ▶ Aurélien Géron, Hands-On Machine Learning (Ch. 10)



Questions?

Differentiable Programming With

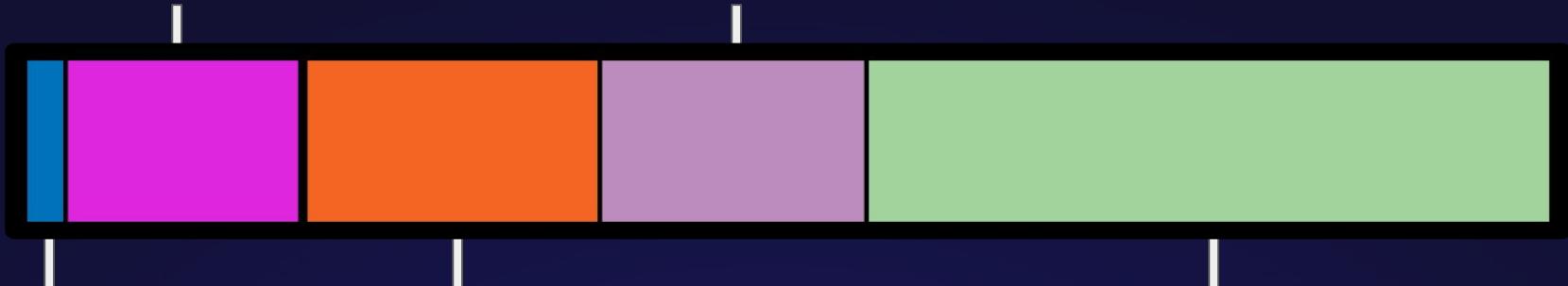




Agenda - 2019

Diff -Programming

Code Along



Intro

Tensorflow

Free Code Session



Agenda - 2020/2021 Corona edition

Diff -Programming

Code Along



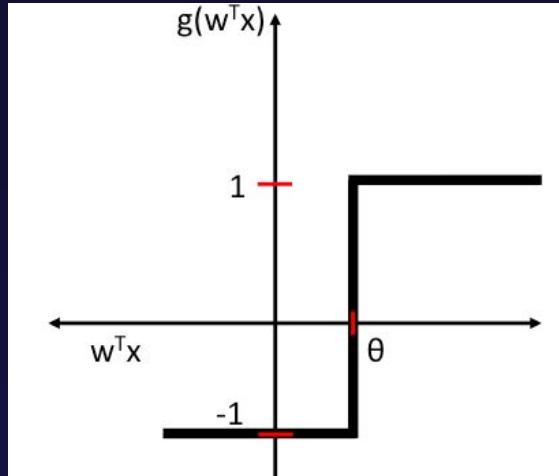
Intro

Tensorflow

~~Free Code Session~~

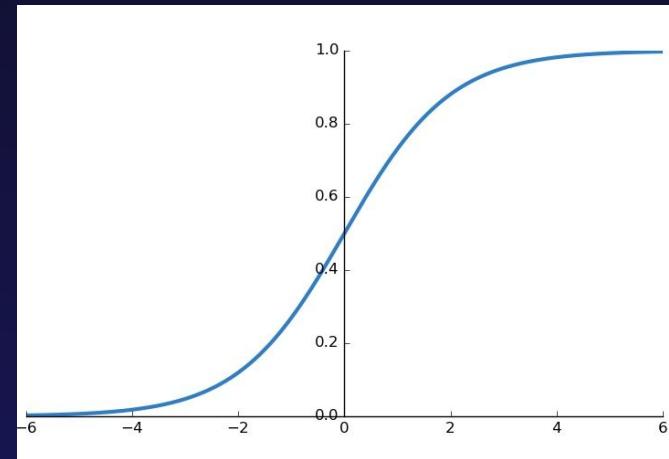
Differentiable Programming

Discrete



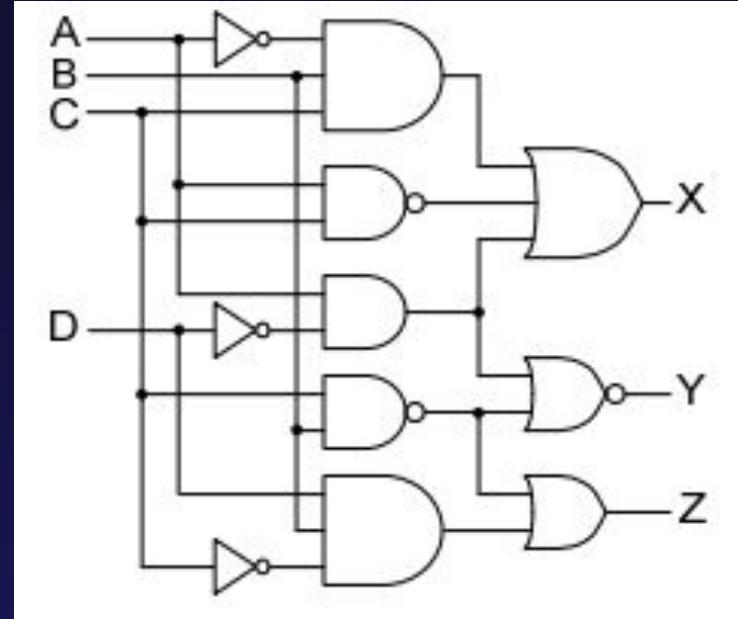
Continuous

VS



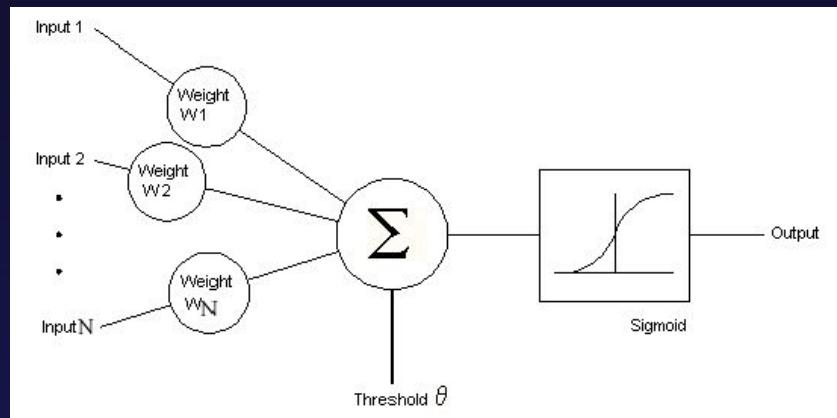
Discrete Circuits

- Discrete circuits are NOT differentiable
- “Butterfly effect”: Minor weight adjustments can have major output ramifications
- Early Neural Networks used discrete functions, but were hard to train effectively



Continuous Circuits

- Allows for differentiation
- Weight adjustments yield foreseeable changes
- Current training algorithms for continuous circuits are orders of magnitude faster than for discrete circuits



Differentiable Programming

Surprising amount of inherently discrete tasks can be approximately differentiated, for example:

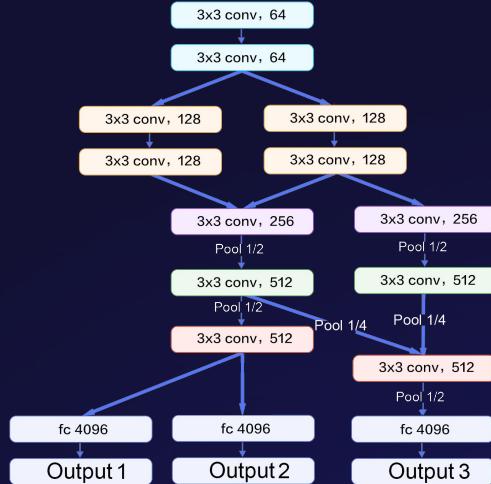


Searching and selecting files
from a file storage



Selecting what move to play in chess

Differentiable Programming



&

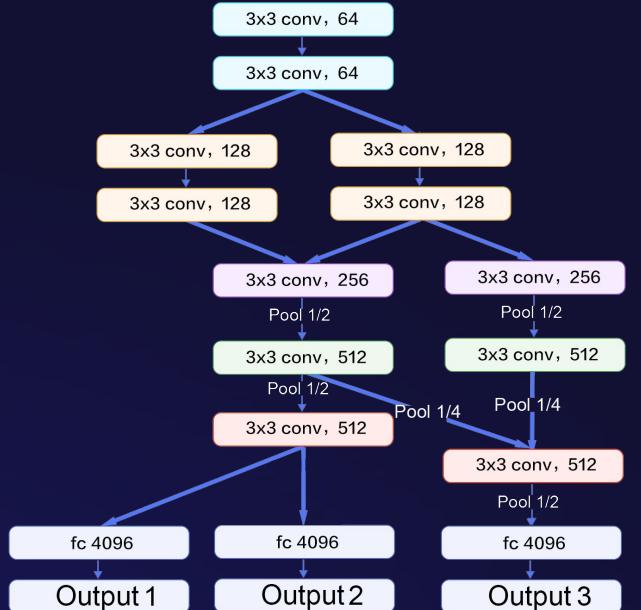


Model Architecture

Loss Function

Model Architecture

- Determines the expressibility of the function approximation
- Training and inference speed can vary widely between different model architectures
- Vast amount of different model possibilities can lead to protracted hyperparameter search



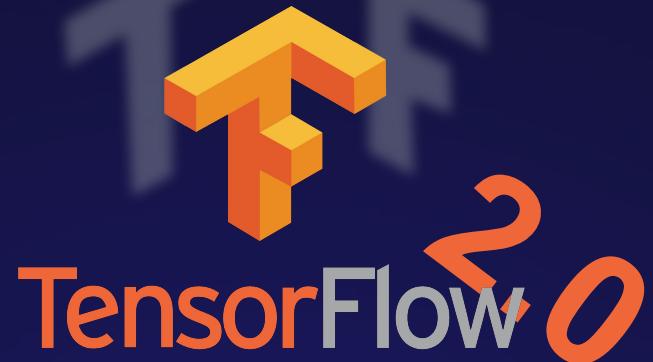
Loss Function

- The loss function acts as the learning target
- The loss must be defined so that a function that minimizes the loss also solves the desired problem
- A clever loss function is worth much more than a clever model architecture





TensorFlow^{2.0}





Tensorflow

Inference Engine

Heavy Optimization

Auto Differentiation

Distribution Strategies

Multi-Platform Support

etc...

Library

Optimizers

Layers

Activations

Standard Datasets

Pre-trained Models

etc...

Inference Engine

Eager Execution

- Express computations in pure Python
- Integrates nicely with your dynamic data structures
- Great for debugging and experimentation

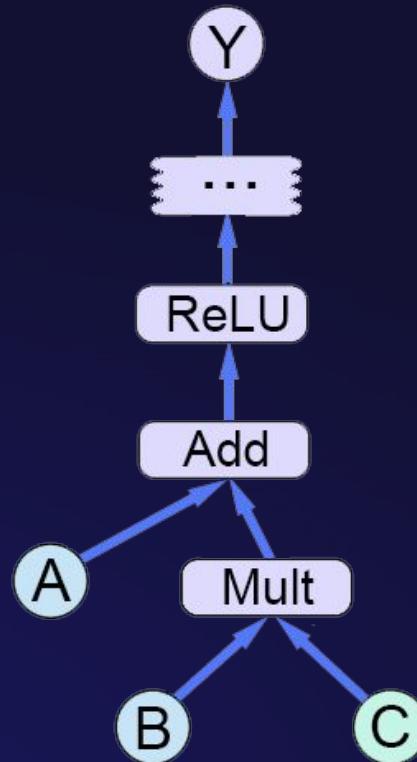
VS

Declarative Graphs

- Pre-define computations in form of a graph
- Allows for heavy optimizations
- Platform independent model structure

Computation Graphs

- Predefine computation in the form of graph
- Gives compiler apriori information, allowing for optimization:
common subexpression elimination, constant folding, etc...
- Hardware agnostic, allowing for easy deployment
- Intuitive for large models



Graph API Overview

Easy to use

Sequential

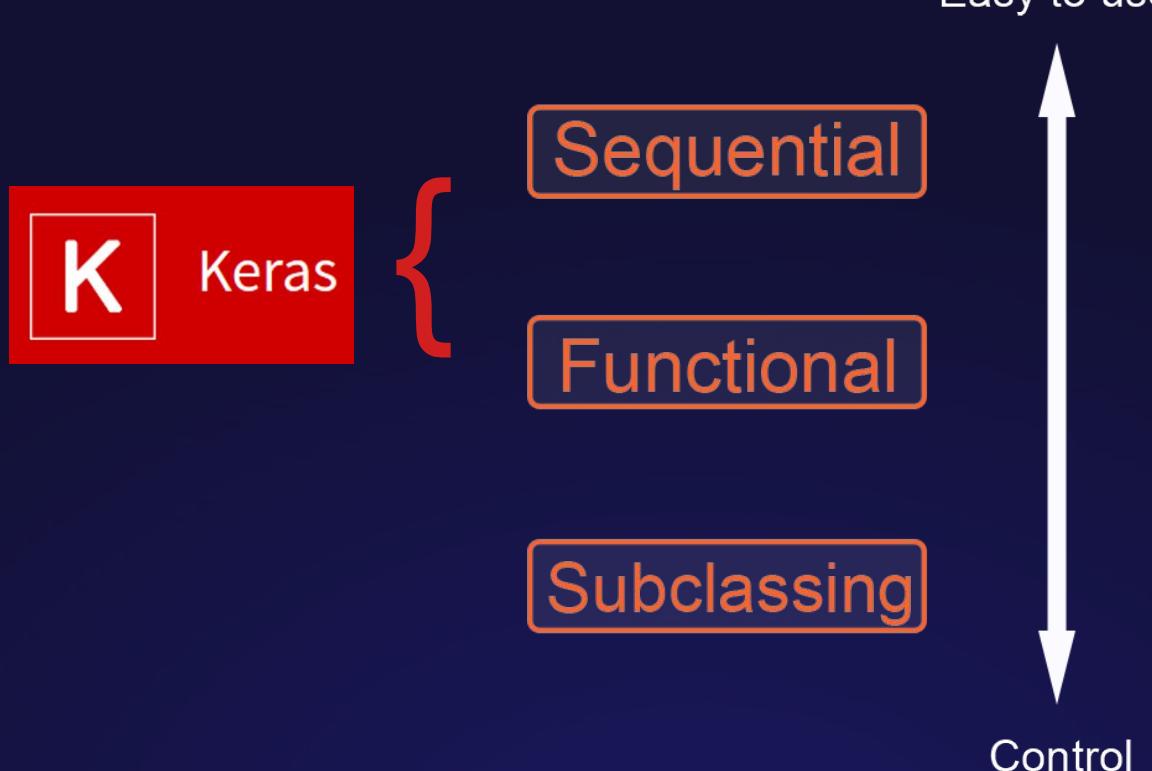
Functional

Subclassing



Control

Graph API Overview





Graph API Overview - Sequential

- Define sequentially stacked models
- Minimal code
- Great Overview



Easy to use

Sequential

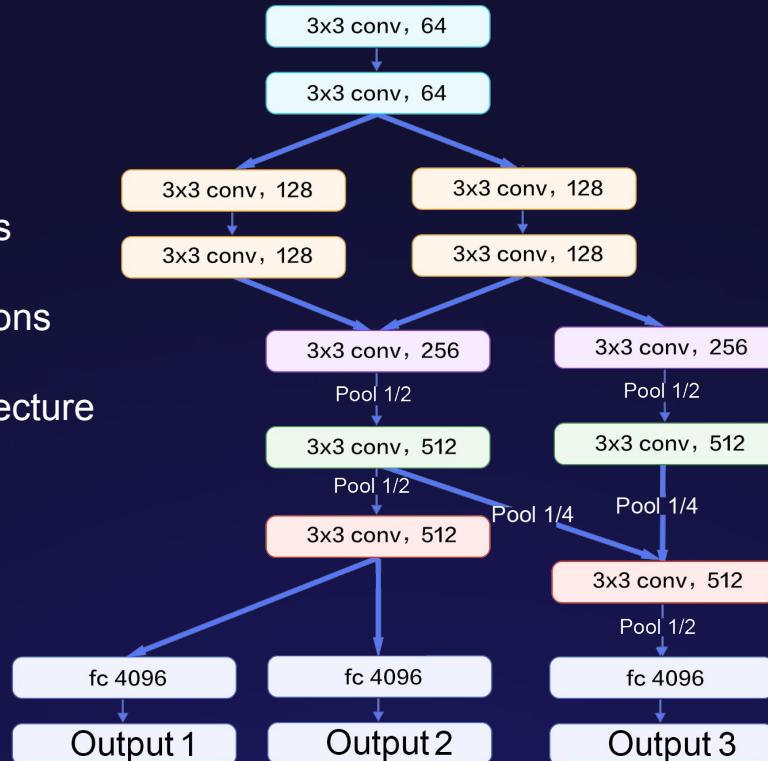
Functional

Subclassing

Control

Graph API Overview - Functional

- Non Sequential Models
- Layer-based Connections
- Good for simple architecture experimentation



Easy to use

Sequential

Functional

Subclassing

Control



Graph API Overview - Subclassing

- Full control over everything
- Custom Losses
- Custom Optimizers
- Custom Activations

Easy to use

```
class MyModel(tf.keras.Model):
    def __init__(self, num_classes=10):
        super(MyModel, self).__init__(name='my_model')
        self.dense_1 = layers.Dense(32, activation='relu')
        self.dense_2 = layers.Dense(num_classes, activation='softmax')

    def call(self, inputs):
        # Define your forward pass here
        x = self.dense_1(inputs)
        return self.dense_2(x)
```

Sequential

Functional

Subclassing



Control



Code Session

Code Session



- Explore eager execution:
 - Numerical Equation Solver
 - Function Approximation
- Solve a binary classification problem using:
 - Subclassing
 - Functional API
 - Sequential
- Using Pre-trained Models:
 - Image Classification
- Audience Choice

Easy to use

Sequential

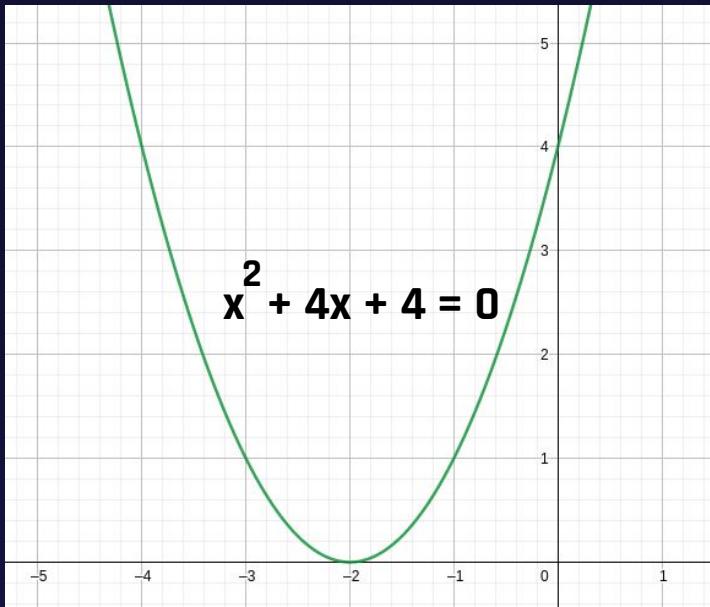
Functional

Subclassing



Control

Numerical Equation Solver

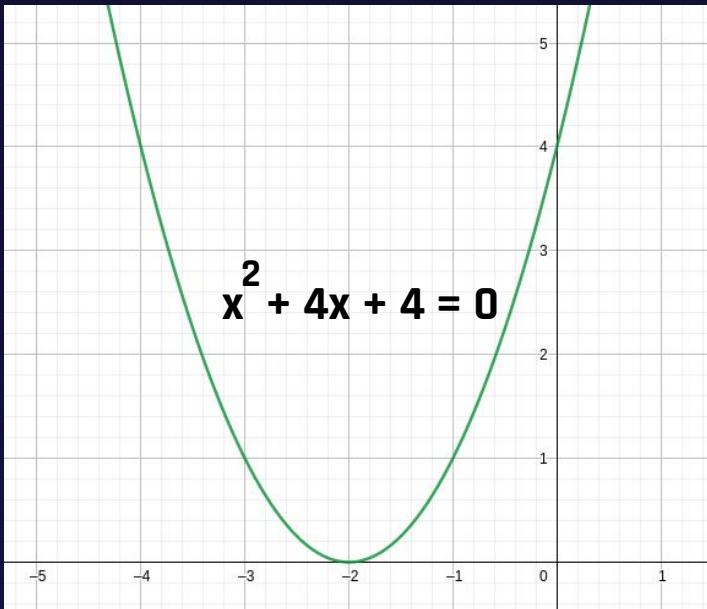


```
def func(x):
    return x**2 + 4*x + 4

myXVar = tf.Variable(0, dtype=tf.float32)
targetY = 0
lr = 0.01

for i in range(100): # 100 iterations Gradient Descent
    with tf.GradientTape() as tape:
        y = func(myXVar)
        loss = (y - targetY)**2 # Squared Error
        gradient = tape.gradient(loss, myXVar)
        myXVar.assign_sub(lr * gradient)
```

Function Approximation



```
def func(x):
    return x ** 2 + 4 * x + 4

X, Y = generateTrainingData(func)
print(X.shape, Y.shape)

# Create the model and Optimizer
denseLayer1 = tf.keras.layers.Dense(64, activation='relu')
denseLayer2 = tf.keras.layers.Dense(1, activation='linear')
optimizer = tf.optimizers.Adam(lr=0.001)

for i in range(10000): # 10K Gradient Descent Updates
    with tf.GradientTape() as tape:
        output1 = denseLayer1(X)
        y = denseLayer2(output1)

        loss = tf.reduce_mean((Y - y) ** 2) # MSE
    print("Loss:", loss)

    modelVars = denseLayer1.variables + denseLayer2.variables
    gradient = tape.gradient(loss, modelVars) # Calculate Gradient
    optimizer.apply_gradients(zip(gradient, modelVars)) # Update Model with Optimizer
```

Banknote Fraud Detection

Given preprocessed features of scanned banknotes
detect which notes are authentic.



Banknote Fraud Detection

Given preprocessed features of scanned banknotes
detect which notes are authentic.

- Binary classification problem
- 1372 data samples with 4 features
- [*** Download Link ***](#)



Banknote Fraud Detection

Given preprocessed features of scanned banknotes
detect which notes are authentic.

- Binary classification problem
- 1372 data samples with 4 features
- [*** Download Link ***](#)





Banknote Authentication Classification

```
class ModelClass(tf.keras.Model):

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self.d1 = tf.keras.layers.Dense(64, 'relu')
        self.d2 = tf.keras.layers.Dense(32, 'relu')
        self.d3 = tf.keras.layers.Dense(1, 'sigmoid')

    def call(self, inputs, training=None, mask=None):
        y1 = self.d1(inputs)
        y2 = self.d2(y1)
        y3 = self.d3(y2)
        return y3

trainX, trainY, testX, testY = LectureUtils.loadBanknotedata()
print(trainX.shape, trainY.shape, testX.shape, testY.shape)

model = ModelClass() # Create an instance of our model
optimizer = tf.optimizers.Adam()
# Specify the optimizer, loss and what metrics we would like to track
model.compile(optimizer, loss='binary_crossentropy', metrics=['acc'])

model.evaluate(testX, testY)
model.fit(trainX, trainY, batch_size=16, epochs=4)
model.evaluate(testX, testY)
```

Easy to use

Sequential

Functional

Subclassing

Control



Banknote Authentication Classification

```
# Pre-define the layers that will be used
inputLayer = tf.keras.layers.Input((4,))
d1 = tf.keras.layers.Dense(64, 'relu')
d2 = tf.keras.layers.Dense(32, 'relu')
d3 = tf.keras.layers.Dense(1, 'sigmoid')

# Pre define a computation graph using the functional API
y1 = d1(inputLayer)
y2 = d2(y1)
y3 = d3(y2)
model = tf.keras.Model(inputLayer, y3)

tf.keras.utils.plot_model(model, 'myModel.png') # Plot the model graph

optimizer = tf.optimizers.Adam()
# Specify the optimizer, loss and what metrics we would like to track
model.compile(optimizer, loss='binary_crossentropy', metrics=['acc'])

model.evaluate(testX, testY)
model.fit(trainX, trainY, batch_size=16, epochs=4)
model.evaluate(testX, testY)
```

Easy to use

Sequential

Functional

Subclassing

Control



Banknote Authentication Classification

```
# Pre define the layers that will be used
inputLayer = tf.keras.layers.Input((4,))
d1 = tf.keras.layers.Dense(64, 'relu')
d2 = tf.keras.layers.Dense(32, 'relu')
d3 = tf.keras.layers.Dense(1, 'sigmoid')

# Pre define a computation graph using the functional API
y1 = d1(inputLayer)
y2 = d2(y1)
y3 = d3(y2)
model = tf.keras.Model(inputLayer, y3)

tf.keras.utils.plot_model(model, 'myModel.png') # Plot the model graph

optimizer = tf.optimizers.Adam()
# Specify the optimizer, loss and what metrics we would like to track
model.compile(optimizer, loss='binary_crossentropy', metrics=['acc'])

model.evaluate(testX, testY)
model.fit(trainX, trainY, batch_size=16, epochs=4)
model.evaluate(testX, testY)
```

Easy to use

Sequential

Functional

Subclassing

Control

Banknote Authentication Classification

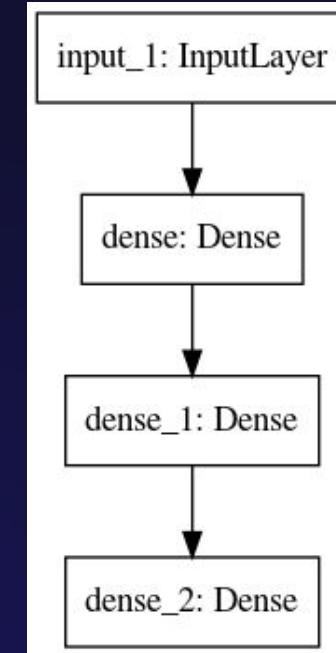
```
# Pre-define the layers that will be used
inputLayer = tf.keras.layers.Input((4,))
d1 = tf.keras.layers.Dense(64, 'relu')
d2 = tf.keras.layers.Dense(32, 'relu')
d3 = tf.keras.layers.Dense(1, 'sigmoid')

# Pre define a computation graph using the functional API
y1 = d1(inputLayer)
y2 = d2(y1)
y3 = d3(y2)
model = tf.keras.Model(inputLayer, y3)

tf.keras.utils.plot_model(model, 'myModel.png') # Plot the model graph

optimizer = tf.optimizers.Adam()
# Specify the optimizer, loss and what metrics we would like to track
model.compile(optimizer, loss='binary_crossentropy', metrics=['acc'])

model.evaluate(testX, testY)
model.fit(trainX, trainY, batch_size=16, epochs=4)
model.evaluate(testX, testY)
```



Banknote Authentication Classification

```
inputLayer = tf.keras.layers.Input((4,))

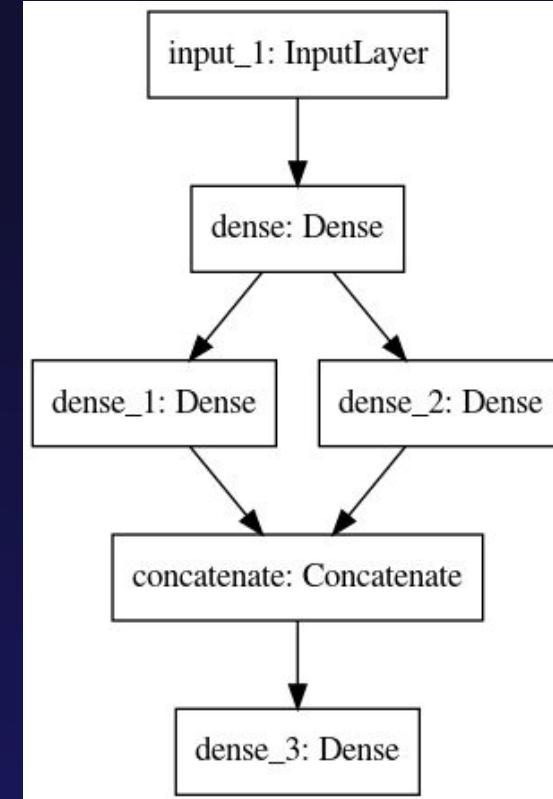
d1 = tf.keras.layers.Dense(64, 'relu')
d2 = tf.keras.layers.Dense(32, 'relu')
d3 = tf.keras.layers.Dense(32, 'sigmoid')
concLayer = tf.keras.layers.concatenate()
d4 = tf.keras.layers.Dense(1, 'sigmoid')

# Pre define a computation graph using the functional API
y1 = d1(inputLayer)
y2 = d2(y1)
y3 = d3(y1)
concY = concLayer([y2, y3])
y4 = d4(concY)
model = tf.keras.Model(inputLayer, y4)

tf.keras.utils.plot_model(model, 'myModel.png') # Plot the model graph

optimizer = tf.optimizers.Adam()
# Specify the optimizer, loss and what metrics we would like to track
model.compile(optimizer, loss='binary_crossentropy', metrics=['acc'])

model.evaluate(testX, testY)
model.fit(trainX, trainY, batch_size=16, epochs=4)
model.evaluate(testX, testY)
```





Banknote Authentication Classification

```
# Pre define the layers that will be used
inputLayer = tf.keras.layers.Input((4,))
d1 = tf.keras.layers.Dense(64, 'relu')
d2 = tf.keras.layers.Dense(32, 'relu')
d3 = tf.keras.layers.Dense(1, 'sigmoid')

# Pre define a computation graph using the functional API
y1 = d1(inputLayer)
y2 = d2(y1)
y3 = d3(y2)
model = tf.keras.Model(inputLayer, y3)

tf.keras.utils.plot_model(model, 'myModel.png') # Plot the model graph

optimizer = tf.optimizers.Adam()
# Specify the optimizer, loss and what metrics we would like to track
model.compile(optimizer, loss='binary_crossentropy', metrics=['acc'])

model.evaluate(testX, testY)
model.fit(trainX, trainY, batch_size=16, epochs=4)
model.evaluate(testX, testY)
```

Easy to use

Sequential

Functional

Subclassing

Control

Banknote Authentication Classification

```
# Pre-define the layers and the computation graph using the functional API
inputLayer = tf.keras.layers.Input((4,))
d1 = tf.keras.layers.Dense(64, 'relu')(inputLayer)
d2 = tf.keras.layers.Dense(32, 'relu')(d1)
d3 = tf.keras.layers.Dense(1, 'sigmoid')(d2)
model = tf.keras.Model(inputLayer, d3)

tf.keras.utils.plot_model(model, 'myModel.png') # Plot the model graph

optimizer = tf.optimizers.Adam()
# Specify the optimizer, loss and what metrics we would like to track
model.compile(optimizer, loss='binary_crossentropy', metrics=['acc'])

model.evaluate(testX, testY)
model.fit(trainX, trainY, batch_size=16, epochs=4)
model.evaluate(testX, testY)
```

Easy to use

Sequential

Functional

Subclassing

Control



Banknote Authentication Classification

```
# Pre-define a sequential computation graph
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(64, 'relu'))
model.add(tf.keras.layers.Dense(32, 'relu'))
model.add(tf.keras.layers.Dense(1, 'sigmoid'))

tf.keras.utils.plot_model(model, 'myModel.png') # Plot the model graph

optimizer = tf.optimizers.Adam()
# Specify the optimizer, loss and what metrics we would like to track
model.compile(optimizer, loss='binary_crossentropy', metrics=['acc'])

model.evaluate(testX, testY)
model.fit(trainX, trainY, batch_size=16, epochs=4)
model.evaluate(testX, testY)
```

Easy to use

Sequential

Functional

Subclassing

Control

Pretrained Models

Training big models is expensive!

Approximative Computational Training Burden:

- **BERT Large** (*NLP Model*)
 - 4 days training on 16 TPUs
 - Cost ~7k dollars
- **AlphaZero** (*RL Model*)
 - 40 days Training time
 - Cost ~35 Million dollars

Pretrained Models

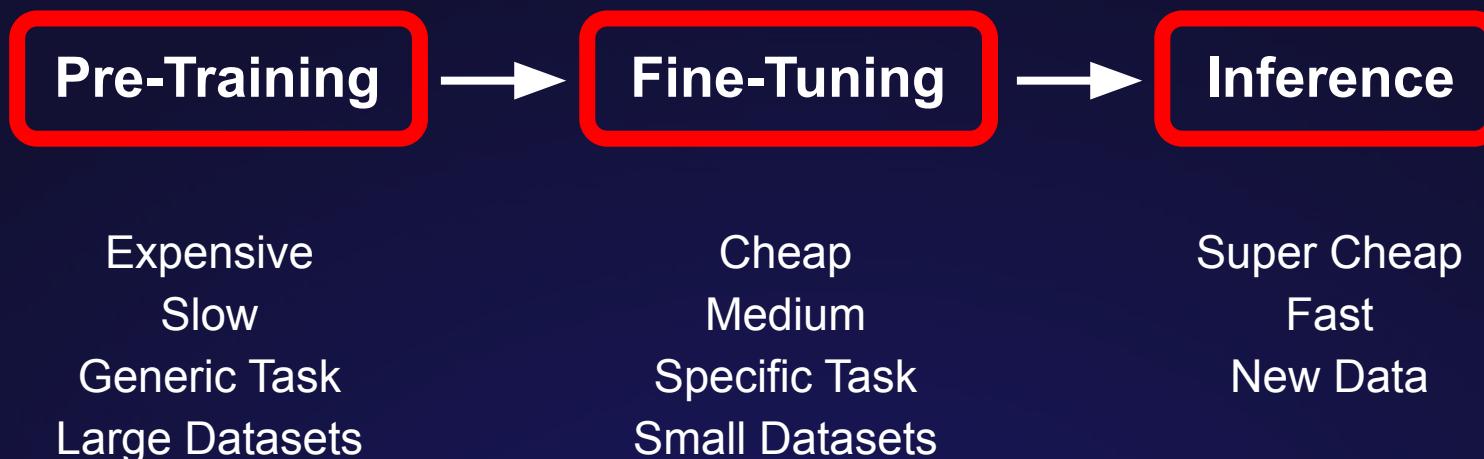
Training big models is expensive!

Approximative Computational Training Burden:

- **BERT Large** (*NLP Model*)
 - 4 days training on 16 TPUs
 - Cost ~7k dollars
- **AlphaZero** (*RL Model*)
 - 40 days Training time
 - Cost ~35 Million dollars

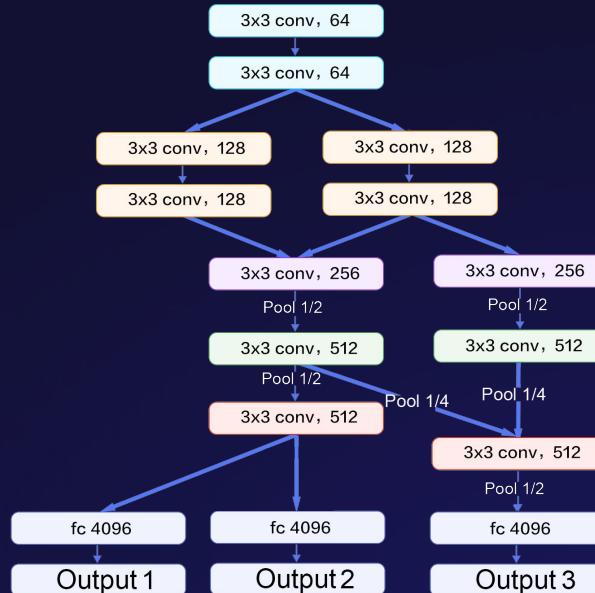
However, once a model is trained it can be used without great cost.

Pretrained Models



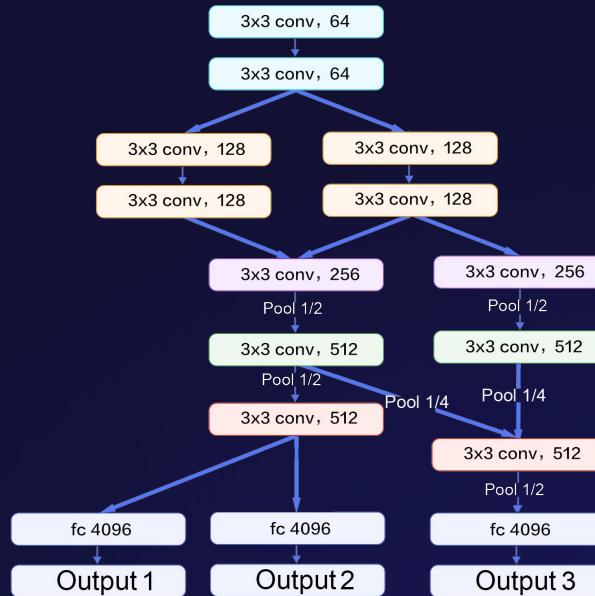
Pretrained Models - Image Classification

Pre-trained Convolutional Neural networks for images



- Over 14 Million labeled Images
- 20k Different classes
- Only naturally occurring images

Pretrained Models - Image Classification



```

import tensorflow as tf

# Load a jpg img and preprocess it for the ResNet50 model
def prepareImage(filePath, modelDims=(224, 224)):
    img = tf.io.read_file(filePath)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, modelDims)
    return tf.keras.applications.resnet50.preprocess_input(img)

imgs = []
for path in ["image1.jpg", "image2.jpg"]:
    imgs.append(prepareImage(path))

cnnModel = tf.keras.applications.ResNet50() # Create pretrained Model
tf.keras.utils.plot_model(cnnModel)

imgs = tf.convert_to_tensor(imgs) # Convert to tensor
print(imgs.shape)

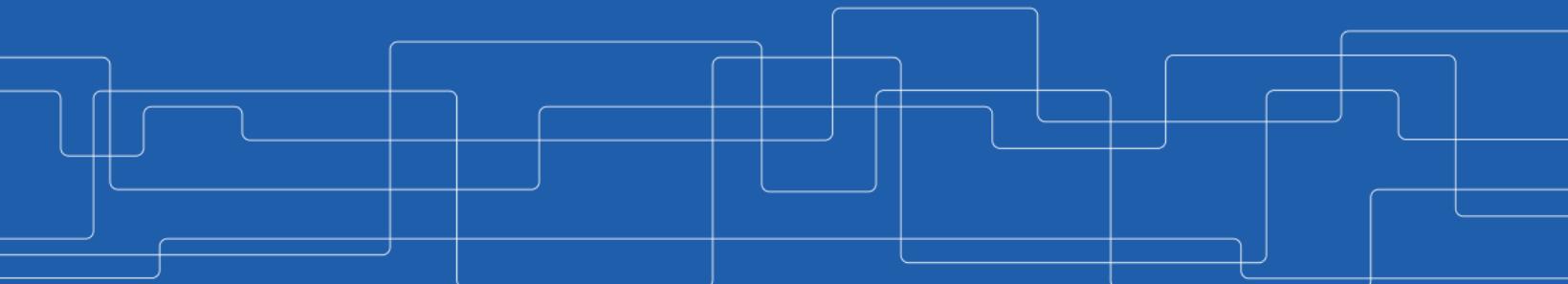
predictions = cnnModel.predict(imgs) # Make predictions
#Decode prediction into ImageNet classes
print(tf.keras.applications.resnet50.decode_predictions(predictions))
  
```

Words of Wisdom



Training Deep Feedforwards Networks

Amir H. Payberah
payberah@kth.se
2021-11-24



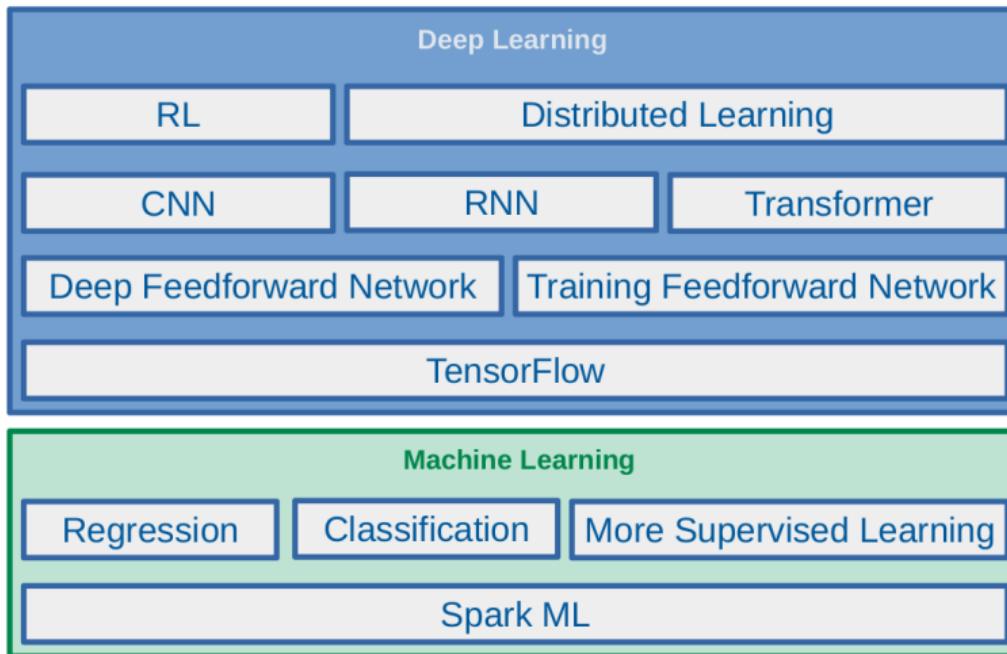


The Course Web Page

<https://id2223kth.github.io>
<https://tinyurl.com/6s5jy46a>

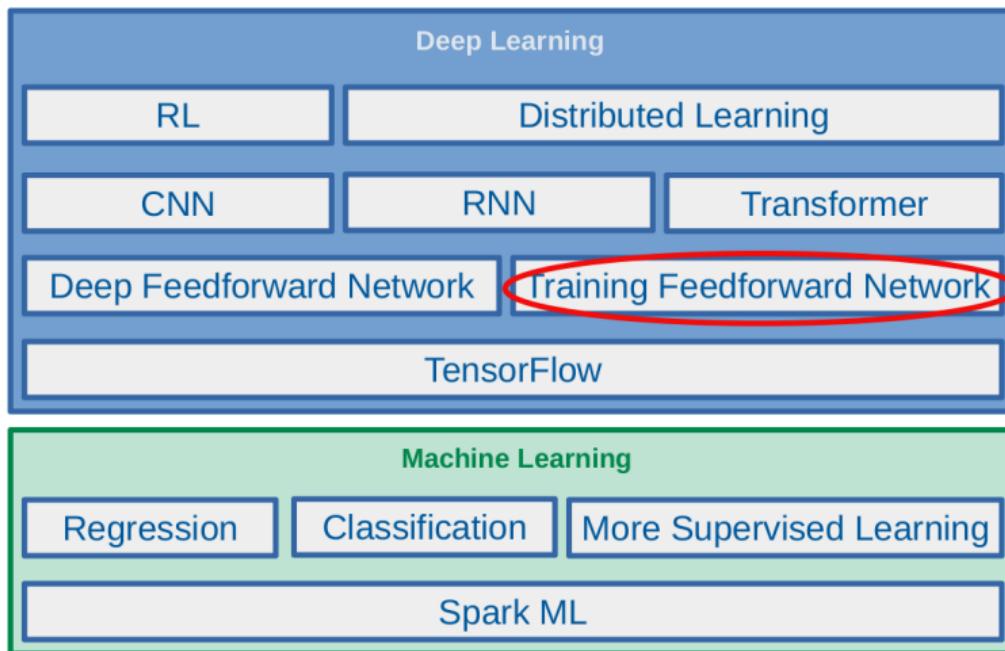


Where Are We?





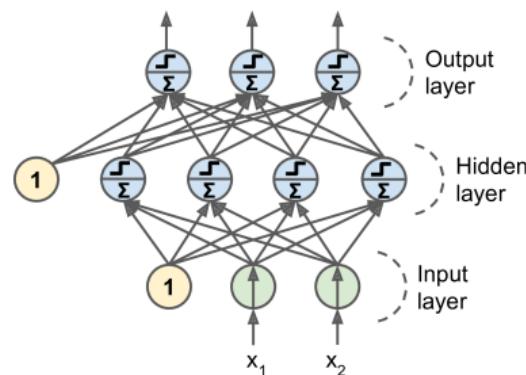
Where Are We?



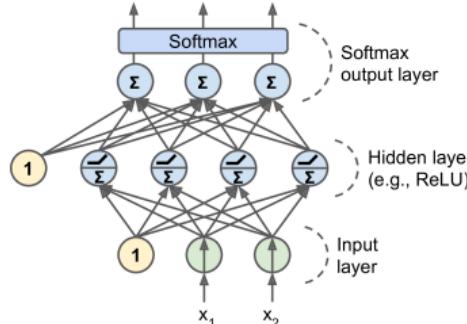
Feedforward Neural Network Architecture

- ▶ A **feedforward neural network** is composed of:

- One **input layer**
- One or more **hidden layers**
- One final **output layer**



Feedforward Network in TensorFlow



```
n_output = 3
n_hidden = 4
n_features = 2

model = keras.models.Sequential()
model.add(keras.layers.Dense(n_hidden, input_shape=(n_features,), activation="relu"))
model.add(keras.layers.Dense(n_output, activation="softmax"))

model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd", metrics=["accuracy"])
model.fit(X_train, y_train, epochs=30)
```



Challenges of Training Feedforward Neural Networks

- ▶ Challenges ...





Challenges of Training Feedforward Neural Networks

- ▶ Challenges ...
- ▶ Overfitting: risk of overfitting a model with large number of parameters.



Challenges of Training Feedforward Neural Networks

- ▶ Challenges ...
- ▶ Overfitting: risk of overfitting a model with large number of parameters.
- ▶ Vanishing/exploding gradients: hard to train lower layers.



Challenges of Training Feedforward Neural Networks

- ▶ Challenges ...
- ▶ Overfitting: risk of overfitting a model with large number of parameters.
- ▶ Vanishing/exploding gradients: hard to train lower layers.
- ▶ Training speed: slow training with large networks.



Overfitting



High Degree of Freedom and Overfitting Problem

- ▶ With **large number of parameters**, a network has a **high degree of freedom**.
- ▶ It can **fit** a huge variety of **complex datasets**.



High Degree of Freedom and Overfitting Problem

- ▶ With **large number of parameters**, a network has a **high degree of freedom**.
- ▶ It can **fit** a huge variety of **complex datasets**.
- ▶ This **flexibility** also means that it is **prone to overfitting on training set**.



High Degree of Freedom and Overfitting Problem

- ▶ With **large number of parameters**, a network has a **high degree of freedom**.
- ▶ It can **fit** a huge variety of **complex datasets**.
- ▶ This **flexibility** also means that it is **prone to overfitting on training set**.
- ▶ Let's **reduce** the degree of freedom a model.



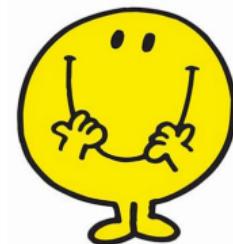
Avoiding Overfitting

- ▶ Early stopping
- ▶ ℓ_1 and ℓ_2 regularization
- ▶ Max-norm regularization
- ▶ Dropout
- ▶ Data augmentation



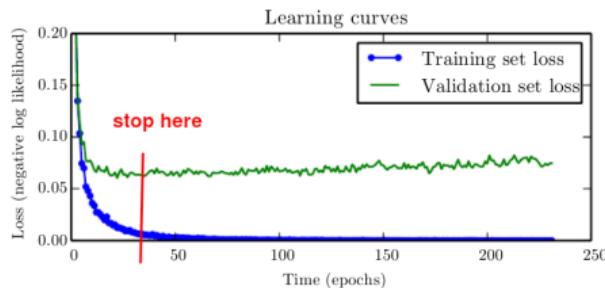
Avoiding Overfitting

- ▶ Early stopping
- ▶ ℓ_1 and ℓ_2 regularization
- ▶ Max-norm regularization
- ▶ Dropout
- ▶ Data augmentation



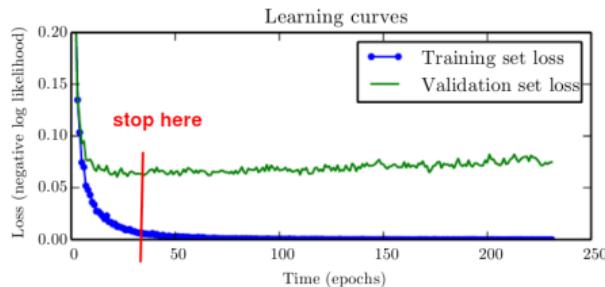
Early Stopping (1/2)

- ▶ As the **training steps go by**, its prediction error on the **training/validation set** naturally goes down.



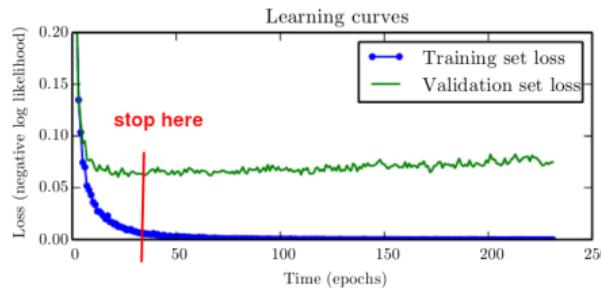
Early Stopping (1/2)

- ▶ As the **training steps go by**, its prediction error on the **training/validation set** naturally **goes down**.
- ▶ After a while the **validation error stops decreasing** and **starts to go back up**.
 - The model has started to **overfit the training data**.



Early Stopping (1/2)

- ▶ As the **training steps go by**, its prediction error on the **training/validation set** naturally **goes down**.
- ▶ After a while the **validation error stops decreasing** and **starts to go back up**.
 - The model has started to **overfit the training data**.
- ▶ In the **early stopping**, we **stop training** when the **validation error reaches a minimum**.





Early Stopping (2/2)

```
from tensorflow.keras.callbacks import EarlyStopping

model = tf.keras.models.Sequential(...)

model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

earlystop_callback = EarlyStopping(monitor='accuracy', min_delta=0.05, patience=1)

model.fit(x_train, y_train, epochs=500, callbacks=[earlystop_callback])
```

Avoiding Overfitting

- ▶ Early stopping
- ▶ ℓ_1 and ℓ_2 regularization
- ▶ Max-norm regularization
- ▶ Dropout
- ▶ Data augmentation





ℓ_1 and ℓ_2 Regularization (1/3)

- ▶ Penalize **large values** of weights w_j .

$$\tilde{J}(w) = J(w) + \lambda R(w)$$

- ▶ Two questions:
 1. How should we define $R(w)$?
 2. How do we determine λ ?



/1 and /2 Regularization (2/3)

- **/1 regression:** $R(w) = \lambda \sum_{i=1}^n |w_i|$ is added to the cost function.

$$\tilde{J}(w) = J(w) + \lambda \sum_{i=1}^n |w_i|$$

```
keras.layers.Dense(100, activation="relu", kernel_regularizer=keras.regularizers.l1(0.1))
```



/1 and /2 Regularization (3/3)

- **/2 regression:** $R(w) = \lambda \sum_{i=1}^n w_i^2$ is added to the **cost function**.

$$\tilde{J}(w) = J(w) + \lambda \sum_{i=1}^n w_i^2$$

```
keras.layers.Dense(100, activation="relu", kernel_regularizer=keras.regularizers.l2(0.01))
```

Avoiding Overfitting

- ▶ Early stopping
- ▶ ℓ_1 and ℓ_2 regularization
- ▶ Max-norm regularization
- ▶ Dropout
- ▶ Data augmentation





Max-Norm Regularization

- ▶ Max-norm regularization: constrains the weights w_j of the incoming connections for each neuron j .
 - Prevents them from getting too large.



Max-Norm Regularization

- ▶ Max-norm regularization: constrains the weights w_j of the incoming connections for each neuron j .
 - Prevents them from getting too large.
- ▶ After each training step, clip w_j as below, if $\|w_j\|_2 > r$:

$$w_j \leftarrow w_j \frac{r}{\|w_j\|_2}$$

- r is the max-norm hyperparameter
- $\|w_j\|_2 = (\sum_i w_{i,j}^2)^{\frac{1}{2}} = \sqrt{w_{1,j}^2 + w_{2,j}^2 + \dots + w_{n,j}^2}$



Max-Norm Regularization

- ▶ Max-norm regularization: constrains the weights w_j of the incoming connections for each neuron j .
 - Prevents them from getting too large.
- ▶ After each training step, clip w_j as below, if $\|w_j\|_2 > r$:

$$w_j \leftarrow w_j \frac{r}{\|w_j\|_2}$$

- r is the max-norm hyperparameter
- $\|w_j\|_2 = (\sum_i w_{i,j}^2)^{\frac{1}{2}} = \sqrt{w_{1,j}^2 + w_{2,j}^2 + \dots + w_{n,j}^2}$

```
keras.layers.Dense(100, activation="relu", kernel_constraint=keras.constraints.max_norm(1.))
```

Avoiding Overfitting

- ▶ Early stopping
- ▶ ℓ_1 and ℓ_2 regularization
- ▶ Max-norm regularization
- ▶ Dropout
- ▶ Data augmentation



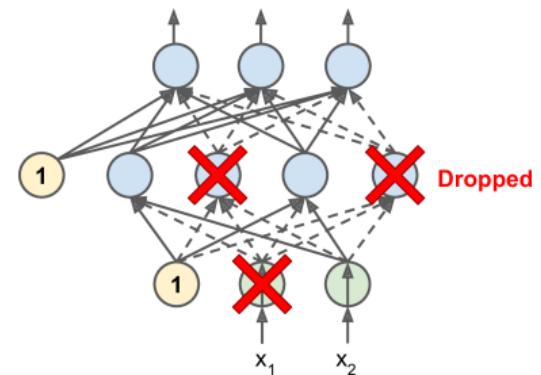
Dropout (1/4)

- ▶ Would a **company** perform better if its employees were told **to toss a coin** every morning to decide **whether or not to go to work**?



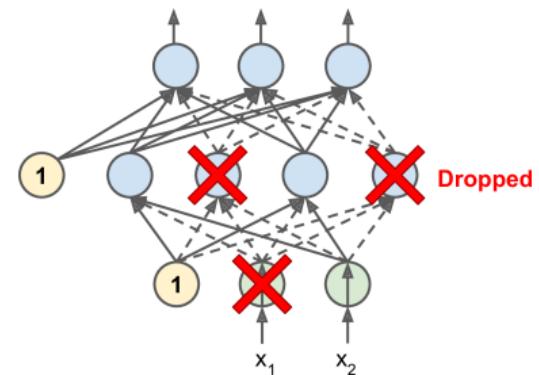
Dropout (2/4)

- ▶ At each **training step**, each neuron drops out temporarily with a **probability p** .



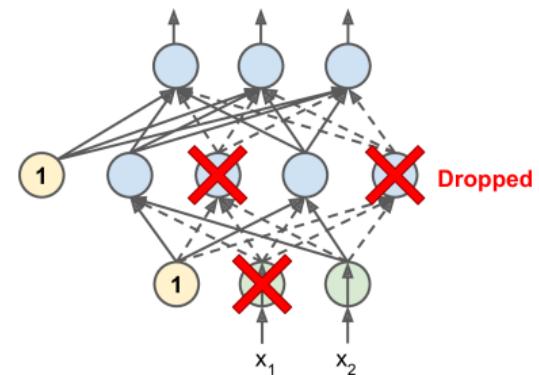
Dropout (2/4)

- ▶ At each **training step**, each neuron drops out temporarily with a **probability p** .
 - The **hyperparameter p** is called the **dropout rate**.



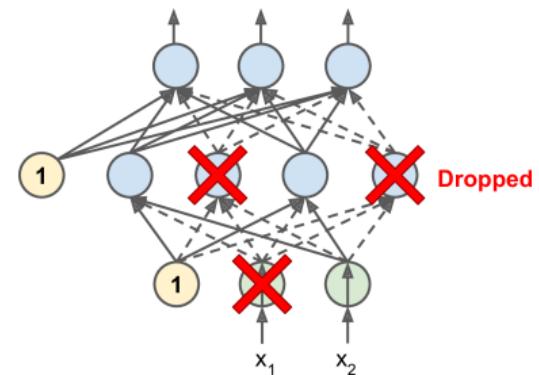
Dropout (2/4)

- ▶ At each **training step**, each neuron drops out temporarily with a probability p .
 - The **hyperparameter p** is called the **dropout rate**.
 - A neuron will be **entirely ignored** during **this training step**.



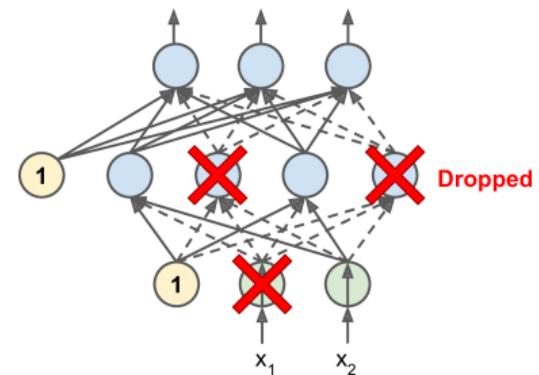
Dropout (2/4)

- ▶ At each **training step**, each neuron drops out temporarily with a probability p .
 - The **hyperparameter p** is called the **dropout rate**.
 - A neuron will be **entirely ignored** during **this training step**.
 - It may be **active** during the **next step**.



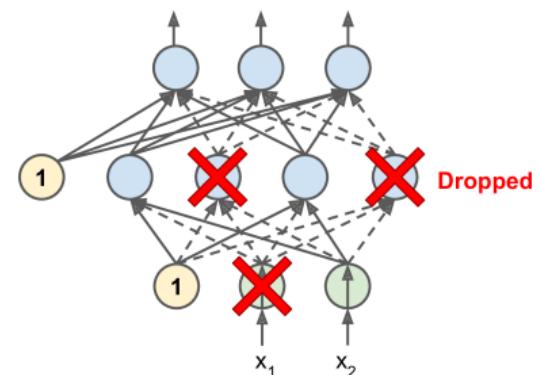
Dropout (2/4)

- ▶ At each **training step**, each neuron drops out temporarily with a probability p .
 - The **hyperparameter p** is called the **dropout rate**.
 - A neuron will be **entirely ignored** during **this training step**.
 - It may be **active** during the **next step**.
 - Exclude the **output neurons**.



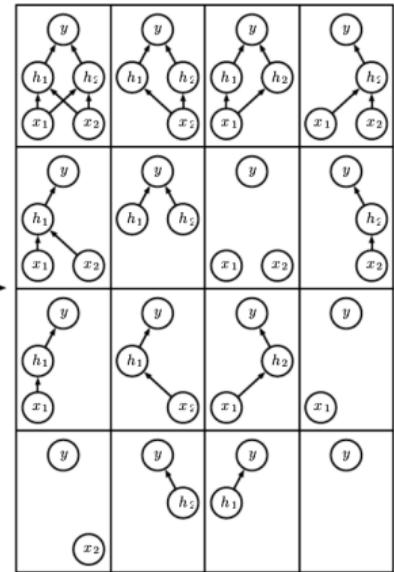
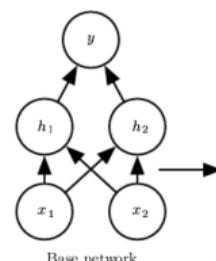
Dropout (2/4)

- ▶ At each **training step**, each neuron drops out temporarily with a probability p .
 - The **hyperparameter p** is called the **dropout rate**.
 - A neuron will be **entirely ignored** during **this training step**.
 - It may be **active** during the **next step**.
 - Exclude the **output neurons**.
- ▶ After training, neurons don't get dropped anymore.



Dropout (3/4)

- ▶ Each neuron can be either **present or absent**.
- ▶ **2^N possible networks**, where **N** is the total number of **droppable neurons**.
 - **N = 4** in this figure.



4



Dropout (4/4)

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dropout(rate=0.2),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dropout(rate=0.2),
    keras.layers.Dense(10, activation="softmax")
])
```

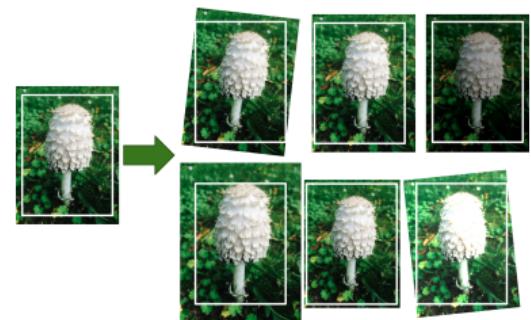
Avoiding Overfitting

- ▶ Early stopping
- ▶ ℓ_1 and ℓ_2 regularization
- ▶ Max-norm regularization
- ▶ Dropout
- ▶ Data augmentation



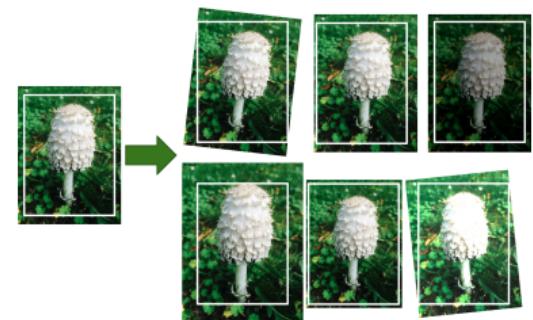
Data Augmentation

- ▶ One way to make a model **generalize better** is to **train it on more data**.
- ▶ This will **reduce overfitting**.



Data Augmentation

- ▶ One way to make a model **generalize better** is to **train it on more data**.
- ▶ This will **reduce overfitting**.
- ▶ Create **fake data** and add it to the **training set**.
 - E.g., in an **image classification** we can slightly shift, rotate and resize an image.
 - Add the resulting pictures to the **training set**.



Vanishing/Exploding Gradients





Vanishing/Exploding Gradients Problem (1/4)

- ▶ The **backpropagation** goes from **output** to **input** layer, and propagates the **error gradient** on the way.

$$w^{(\text{next})} = w - \eta \frac{\partial J(w)}{\partial w}$$



Vanishing/Exploding Gradients Problem (1/4)

- ▶ The **backpropagation** goes from **output** to **input** layer, and propagates the **error gradient** on the way.

$$w^{(\text{next})} = w - \eta \frac{\partial J(w)}{\partial w}$$

- ▶ Gradients often get **smaller and smaller** as the algorithm progresses **down to the lower layers**.
- ▶ As a result, the gradient descent update leaves the **lower layer connection weights** virtually **unchanged**.



Vanishing/Exploding Gradients Problem (1/4)

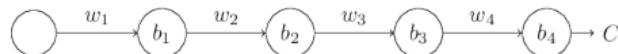
- ▶ The backpropagation goes from output to input layer, and propagates the error gradient on the way.

$$w^{(\text{next})} = w - \eta \frac{\partial J(w)}{\partial w}$$

- ▶ Gradients often get smaller and smaller as the algorithm progresses down to the lower layers.
- ▶ As a result, the gradient descent update leaves the lower layer connection weights virtually unchanged.
- ▶ This is called the vanishing gradients problem.

Vanishing/Exploding Gradients Problem (2/4)

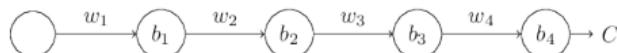
- ▶ Assume a network with just a single neuron in each layer.



- w_1, w_2, \dots are the weights
- b_1, b_2, \dots are the biases
- C is the cost function

Vanishing/Exploding Gradients Problem (2/4)

- ▶ Assume a network with just a single neuron in each layer.



- w_1, w_2, \dots are the **weights**
- b_1, b_2, \dots are the **biases**
- C is the **cost function**

- ▶ The output a_j from the j th neuron is $\sigma(z_j)$.

- σ is the **sigmoid** activation function
- $z_j = w_j a_{j-1} + b_j$
- E.g., $a_4 = \sigma(z_4) = \text{sigmoid}(w_4 a_3 + b_4)$

Vanishing/Exploding Gradients Problem (3/4)

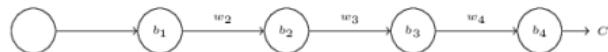
- ▶ Lets compute the **gradient** associated to the **first hidden neuron** ($\frac{\partial C}{\partial b_1}$).



$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times \frac{\partial z_4}{\partial a_3} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial z_3}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial z_1}{\partial b_1}$$

Vanishing/Exploding Gradients Problem (3/4)

- Lets compute the **gradient** associated to the **first hidden neuron** ($\frac{\partial C}{\partial b_1}$).



$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times \frac{\partial z_4}{\partial a_3} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial z_3}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial z_1}{\partial b_1}$$

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times \frac{\partial w_4 a_3 + b_4}{\partial a_3} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial w_3 a_2 + b_3}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial w_2 a_1 + b_2}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial w_1 a_0 + b_1}{\partial b_1}$$

Vanishing/Exploding Gradients Problem (3/4)

- Lets compute the **gradient** associated to the **first hidden neuron** ($\frac{\partial C}{\partial b_1}$).



$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times \frac{\partial z_4}{\partial a_3} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial z_3}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial z_1}{\partial b_1}$$

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times \frac{\partial w_4 a_3 + b_4}{\partial a_3} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial w_3 a_2 + b_3}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial w_2 a_1 + b_2}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial w_1 a_0 + b_1}{\partial b_1}$$

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times w_4 \times \frac{\partial a_3}{\partial z_3} \times w_3 \times \frac{\partial a_2}{\partial z_2} \times w_2 \times \frac{\partial a_1}{\partial z_1} \times 1$$

Vanishing/Exploding Gradients Problem (4/4)

- ▶ Now, consider $\frac{\partial C}{\partial b_3}$.



$$\frac{\partial C}{\partial b_3} = \frac{\partial C}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times w_4 \times \frac{\partial a_3}{\partial z_3}$$

Vanishing/Exploding Gradients Problem (4/4)

- ▶ Now, consider $\frac{\partial C}{\partial b_3}$.



$$\frac{\partial C}{\partial b_3} = \frac{\partial C}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times w_4 \times \frac{\partial a_3}{\partial z_3}$$

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times w_4 \times \frac{\partial a_3}{\partial z_3} \times w_3 \times \frac{\partial a_2}{\partial z_2} \times w_2 \times \frac{\partial a_1}{\partial z_1} \times 1$$

Vanishing/Exploding Gradients Problem (4/4)

- ▶ Now, consider $\frac{\partial C}{\partial b_3}$.



$$\frac{\partial C}{\partial b_3} = \frac{\partial C}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times w_4 \times \frac{\partial a_3}{\partial z_3}$$

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times w_4 \times \frac{\partial a_3}{\partial z_3} \times w_3 \times \frac{\partial a_2}{\partial z_2} \times w_2 \times \frac{\partial a_1}{\partial z_1} \times 1$$

- ▶ Assume $w_3 \times \frac{\partial a_2}{\partial z_2} < \frac{1}{4}$ and $w_2 \times \frac{\partial a_1}{\partial z_1} < \frac{1}{4}$
 - The gradient $\frac{\partial C}{\partial b_1}$ be a factor of 16 (or more) smaller than $\frac{\partial C}{\partial b_3}$.
 - This is the essential origin of the vanishing gradient problem.

Overcoming the Vanishing Gradient

- ▶ Parameter initialization strategies
- ▶ Nonsaturating activation function
- ▶ Batch normalization
- ▶ Gradient clipping





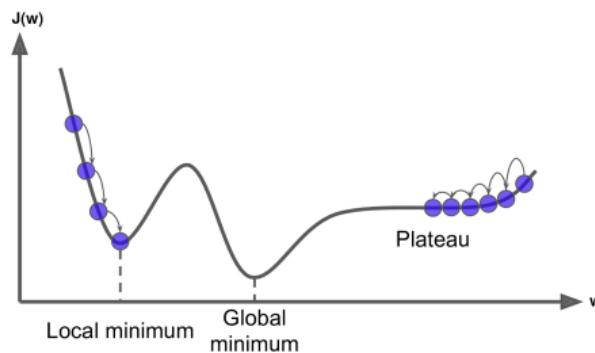
Overcoming the Vanishing Gradient

- ▶ Parameter initialization strategies
- ▶ Nonsaturating activation function
- ▶ Batch normalization
- ▶ Gradient clipping



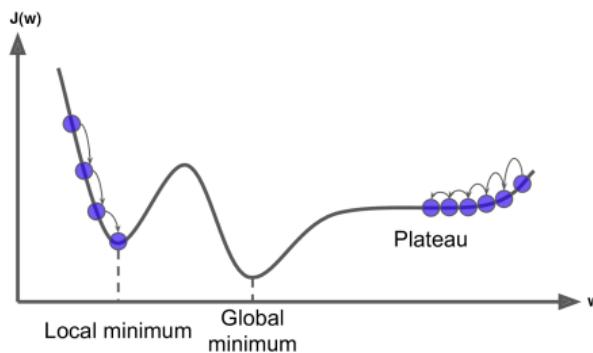
Parameter Initialization Strategies (1/4)

- The non-linearity of a neural network causes the cost functions to become non-convex.



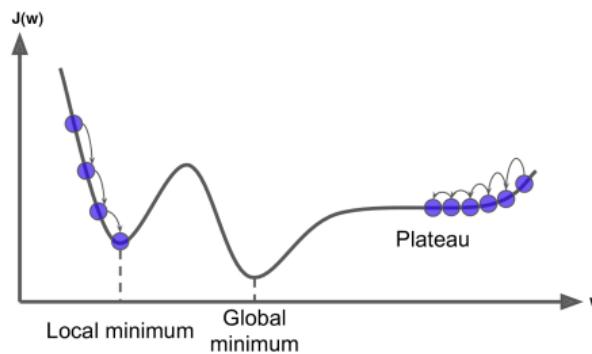
Parameter Initialization Strategies (1/4)

- ▶ The **non-linearity** of a neural network causes the **cost functions** to become **non-convex**.
- ▶ The stochastic gradient descent on **non-convex cost functions** performs is **sensitive** to the values of the **initial parameters**.



Parameter Initialization Strategies (1/4)

- ▶ The **non-linearity** of a neural network causes the **cost functions** to become **non-convex**.
- ▶ The stochastic gradient descent on **non-convex cost functions** performs is **sensitive** to the values of the **initial parameters**.
- ▶ Designing initialization strategies is a **difficult task**.





Parameter Initialization Strategies (2/4)

- ▶ The **initial parameters** need to **break symmetry** between **different units**.



Parameter Initialization Strategies (2/4)

- ▶ The **initial parameters** need to **break symmetry** between **different units**.
- ▶ **Two hidden units** with the **same activation function** connected to the **same inputs**, must have **different** initial parameters.



Parameter Initialization Strategies (2/4)

- ▶ The **initial parameters** need to **break symmetry** between **different units**.
- ▶ **Two hidden units** with the **same activation function** connected to the **same inputs**, must have **different** initial parameters.
 - The goal of having each unit **compute a different function**.



Parameter Initialization Strategies (2/4)

- ▶ The **initial parameters** need to **break symmetry** between **different units**.
- ▶ **Two hidden units** with the **same activation function** connected to the **same inputs**, must have **different** initial parameters.
 - The goal of having each unit **compute a different function**.
- ▶ It motivates **random initialization** of the parameters.
 - Typically, we set the **biases** to **constants**, and initialize only the **weights randomly**.



Parameter Initialization Strategies (3/4)

- We need the signals to flow properly in **both** directions.



Parameter Initialization Strategies (3/4)

- ▶ We need the signals to flow properly in **both** directions.
- ▶ The **Glorot and Bengio initialization** proposed that:



Parameter Initialization Strategies (3/4)

- ▶ We need the signals to flow properly in **both** directions.
- ▶ The **Glorot and Bengio initialization** proposed that:
 - The **variance** of the outputs of each layer to be **equal** to the **variance** of its inputs.



Parameter Initialization Strategies (3/4)

- ▶ We need the signals to flow properly in **both** directions.
- ▶ The **Glorot and Bengio initialization** proposed that:
 - The **variance** of the outputs of each layer to be **equal** to the **variance** of its inputs.
 - The **gradients** to have **equal variance before and after** flowing through a layer in the reverse direction.



Parameter Initialization Strategies (3/4)

- ▶ We need the signals to flow properly in **both** directions.
- ▶ The **Glorot and Bengio initialization** proposed that:
 - The **variance** of the outputs of each layer to be **equal** to the **variance** of its inputs.
 - The **gradients** to have **equal variance before and after** flowing through a layer in the reverse direction.
- ▶ It is not possible to guarantee both unless each layer has an **equal number of inputs and neurons**.



Parameter Initialization Strategies (3/4)

- ▶ We need the signals to flow properly in **both** directions.
- ▶ The **Glorot and Bengio initialization** proposed that:
 - The **variance** of the outputs of each layer to be **equal** to the **variance** of its inputs.
 - The **gradients** to have **equal variance before and after** flowing through a layer in the reverse direction.
- ▶ It is not possible to guarantee both unless each layer has an **equal number of inputs and neurons**.
- ▶ Based on the **Xavier initialization**, the weights are **initialized** using **normal distribution** with **mean 0** and the following **standard deviation**.



Parameter Initialization Strategies (4/4)

- ▶ fan_{in} and fan_{out} are the **number of inputs and neurons** for the layer whose weights are being initialized.
- ▶ $\text{fan}_{\text{avg}} = \frac{2}{\text{fan}_{\text{in}} + \text{fan}_{\text{out}}}$



Parameter Initialization Strategies (4/4)

- ▶ fan_{in} and fan_{out} are the **number of inputs and neurons** for the layer whose weights are being initialized.
- ▶ $\text{fan}_{\text{avg}} = \frac{2}{\text{fan}_{\text{in}} + \text{fan}_{\text{out}}}$
- ▶ **Glorot** initialization, for **none**, **logistic**, **sigmoid**, and **tanh**: $\sigma^2 = \frac{1}{\text{fan}_{\text{avg}}}$



Parameter Initialization Strategies (4/4)

- ▶ fan_{in} and fan_{out} are the **number of inputs and neurons** for the layer whose weights are being initialized.
- ▶ $\text{fan}_{\text{avg}} = \frac{2}{\text{fan}_{\text{in}} + \text{fan}_{\text{out}}}$
- ▶ **Glorot** initialization, for **none**, **logistic**, **sigmoid**, and **tanh**: $\sigma^2 = \frac{1}{\text{fan}_{\text{avg}}}$
- ▶ **He** initialization, for **ReLU**: $\sigma^2 = \frac{2}{\text{fan}_{\text{in}}}$



Parameter Initialization Strategies (4/4)

- ▶ fan_{in} and fan_{out} are the **number of inputs and neurons** for the layer whose weights are being initialized.
- ▶ $\text{fan}_{\text{avg}} = \frac{2}{\text{fan}_{\text{in}} + \text{fan}_{\text{out}}}$
- ▶ **Glorot** initialization, for **none**, **logistic**, **sigmoid**, and **tanh**: $\sigma^2 = \frac{1}{\text{fan}_{\text{avg}}}$
- ▶ **He** initialization, for **ReLU**: $\sigma^2 = \frac{2}{\text{fan}_{\text{in}}}$

```
keras.layers.Dense(10, activation="relu", kernel_initializer="he_normal")
```

Overcoming the Vanishing Gradient

- ▶ Parameter initialization strategies
- ▶ Nonsaturating activation function
- ▶ Batch normalization
- ▶ Gradient clipping

by Roger Hargreaves





Nonsaturating Activation Functions (1/4)

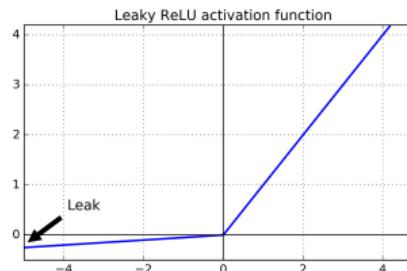
- ▶ $\text{ReLU}(z) = \max(0, z)$
- ▶ The **dying ReLUs** problem.

Nonsaturating Activation Functions (1/4)

- ▶ $\text{ReLU}(z) = \max(0, z)$
- ▶ The **dying ReLUs** problem.
 - During **training**, some neurons **stop outputting anything other than 0**.
 - E.g., when the **weighted sum of the neuron's inputs is negative**, it starts outputting 0.

Nonsaturating Activation Functions (1/4)

- ▶ $\text{ReLU}(z) = \max(0, z)$
- ▶ The **dying ReLUs** problem.
 - During **training**, some neurons **stop outputting anything other than 0**.
 - E.g., when the **weighted sum of the neuron's inputs is negative**, it starts outputting 0.
- ▶ Use **leaky ReLU** instead: $\text{LeakyReLU}_\alpha(z) = \max(\alpha z, z)$.
 - α is the **slope** of the function for $z < 0$.





Nonsaturating Activation Functions (2/4)

► Randomized Leaky ReLU (RReLU)

- α is picked **randomly** during training, and it is **fixed** during testing.



Nonsaturating Activation Functions (2/4)

- ▶ Randomized Leaky ReLU (RReLU)
 - α is picked **randomly** during training, and it is **fixed** during testing.
- ▶ Parametric Leaky ReLU (PReLU)
 - Learn α **during training** (instead of being a hyperparameter).

Nonsaturating Activation Functions (2/4)

► Randomized Leaky ReLU (RReLU)

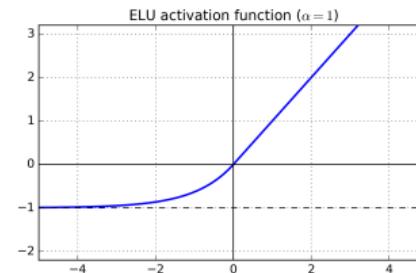
- α is picked **randomly** during training, and it is **fixed** during testing.

► Parametric Leaky ReLU (PReLU)

- Learn α **during training** (instead of being a hyperparameter).

► Exponential Linear Unit (ELU)

$$\text{ELU}_\alpha(z) = \begin{cases} \alpha(\exp(z) - 1) & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$



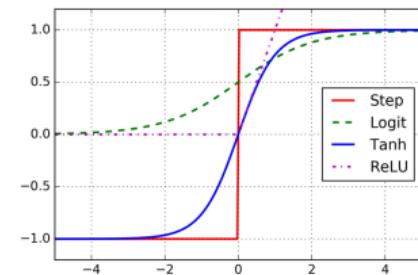
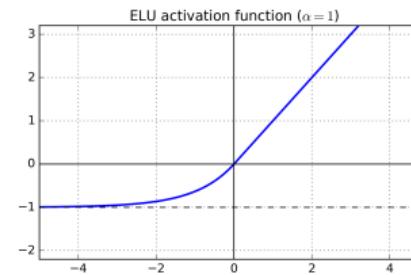
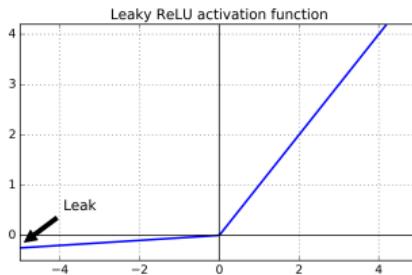


Nonsaturating Activation Functions (3/4)

- ▶ Which activation function should we use?

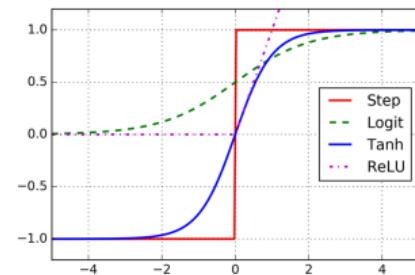
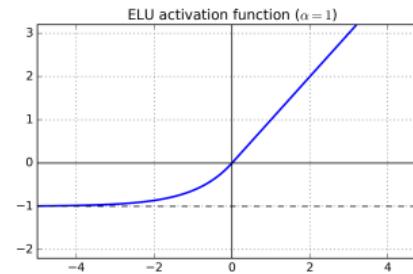
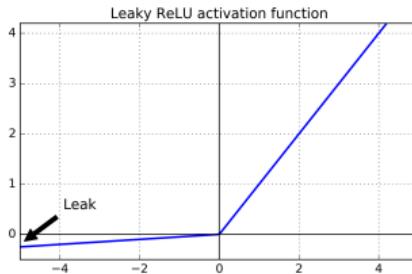
Nonsaturating Activation Functions (3/4)

- ▶ Which activation function should we use?
- ▶ In general logistic < tanh < ReLU < leaky ReLU (and its variants) < ELU



Nonsaturating Activation Functions (3/4)

- ▶ Which activation function should we use?
- ▶ In general logistic < tanh < ReLU < leaky ReLU (and its variants) < ELU
- ▶ If you care about runtime performance, then leaky ReLUs works better than ELUs.





Nonsaturating Activation Functions (4/4)

```
# elu
keras.layers.Dense(10, activation="elu")
```



Nonsaturating Activation Functions (4/4)

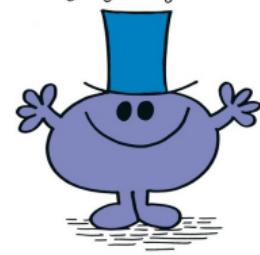
```
# elu
keras.layers.Dense(10, activation="elu")
```

```
# leaky relu
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(128, kernel_initializer="he_normal"),
    keras.layers.LeakyReLU(),
    keras.layers.Dense(10, activation="softmax")
])
```

Overcoming the Vanishing Gradient

- ▶ Parameter initialization strategies
- ▶ Nonsaturating activation function
- ▶ **Batch normalization**
- ▶ Gradient clipping

by Roger Hargreaves





Batch Normalization (1/4)

- ▶ The gradient tells how to update each parameter, under the assumption that the other layers do not change.



Batch Normalization (1/4)

- ▶ The gradient tells how to update each parameter, under the assumption that the other layers do not change.
 - In practice, we update all of the layers simultaneously.
 - However, unexpected results can happen.



Batch Normalization (1/4)

- ▶ The gradient tells how to update each parameter, under the assumption that the other layers do not change.
 - In practice, we update all of the layers simultaneously.
 - However, unexpected results can happen.
- ▶ Batch normalization makes the learning of layers in the network more independent of each other.



Batch Normalization (1/4)

- ▶ The gradient tells how to update each parameter, under the assumption that the other layers do not change.
 - In practice, we update all of the layers simultaneously.
 - However, unexpected results can happen.
- ▶ Batch normalization makes the learning of layers in the network more independent of each other.
 - It is a technique to address the problem that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change.



Batch Normalization (1/4)

- ▶ The gradient tells how to **update each parameter**, under the assumption that **the other layers do not change**.
 - In practice, we update all of the layers **simultaneously**.
 - However, **unexpected results can happen**.
- ▶ **Batch normalization** makes the **learning of layers** in the network more **independent of each other**.
 - It is a technique to address the problem that the **distribution of each layer's inputs** changes **during training**, as the parameters of the previous layers change.
- ▶ The technique consists of **adding an operation** in the model just **before the activation function** of each layer.



Batch Normalization (2/4)

- ▶ It's zero-centering and normalizing the inputs, then scaling and shifting the result.



Batch Normalization (2/4)

- ▶ It's zero-centering and normalizing the inputs, then scaling and shifting the result.
 - Estimates the inputs' mean and standard deviation of the current mini-batch.



Batch Normalization (2/4)

- ▶ It's zero-centering and normalizing the inputs, then scaling and shifting the result.
 - Estimates the inputs' mean and standard deviation of the current mini-batch.

$$\mu_B = \frac{1}{m_B} \sum_{i=1}^{m_B} x^{(i)}$$

$$\sigma_B^2 = \frac{1}{m_B} \sum_{i=1}^{m_B} (x^{(i)} - \mu_B)^2$$

- ▶ μ_B : the empirical mean, evaluated over the whole mini-batch B .
- ▶ σ_B : the empirical standard deviation, also evaluated over the whole mini-batch.
- ▶ m_B : the number of instances in the mini-batch.



Batch Normalization (3/4)

$$\hat{x}^{(i)} = \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$
$$z^{(i)} = \gamma \hat{x}^{(i)} + \beta$$

- ▶ $\hat{x}^{(i)}$: the zero-centered and normalized input.
- ▶ $z^{(i)}$: the output of the BN operation, which is a scaled and shifted version of the inputs.
- ▶ γ : the scaling parameter vector for the layer.
- ▶ β : the shifting parameter (offset) vector for the layer.
- ▶ ϵ : a tiny number to avoid division by zero.
- ▶ \otimes : represents the element-wise multiplication.



Batch Normalization (4/4)

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(10, activation="softmax")
])
```

Overcoming the Vanishing Gradient

- ▶ Parameter initialization strategies
- ▶ Nonsaturating activation function
- ▶ Batch normalization
- ▶ **Gradient clipping**

Roger Hargreaves





Gradient Clipping

- **Gradient clipping:** clip the gradients during **backpropagation** so that they **never exceed some threshold**.

```
optimizer = keras.optimizers.SGD(clipvalue=1.0)
model.compile(loss="mse", optimizer=optimizer)
```



Gradient Clipping

- ▶ **Gradient clipping:** clip the gradients during **backpropagation** so that they **never exceed some threshold**.

```
optimizer = keras.optimizers.SGD(clipvalue=1.0)
model.compile(loss="mse", optimizer=optimizer)
```

- ▶ Setting the **clipvalue** or **clipnorm** argument when creating an optimizer.



Gradient Clipping

- ▶ Gradient clipping: clip the gradients during backpropagation so that they never exceed some threshold.

```
optimizer = keras.optimizers.SGD(clipvalue=1.0)
model.compile(loss="mse", optimizer=optimizer)
```

- ▶ Setting the `clipvalue` or `clipnorm` argument when creating an optimizer.
- ▶ `clipvalue=1.0` and `clipnorm=1.0`: values between -1.0 and 1.0.



Gradient Clipping

- ▶ **Gradient clipping:** clip the gradients during **backpropagation** so that they **never exceed some threshold**.

```
optimizer = keras.optimizers.SGD(clipvalue=1.0)
model.compile(loss="mse", optimizer=optimizer)
```

- ▶ Setting the **clipvalue** or **clipnorm** argument when creating an optimizer.
- ▶ **clipvalue=1.0** and **clipnorm=1.0**: values between -1.0 and 1.0.
- ▶ **clipvalue=1.0**: $[0.9, 100.0] \Rightarrow [0.9, 1.0]$



Gradient Clipping

- ▶ **Gradient clipping:** clip the gradients during **backpropagation** so that they **never exceed some threshold**.

```
optimizer = keras.optimizers.SGD(clipvalue=1.0)
model.compile(loss="mse", optimizer=optimizer)
```

- ▶ Setting the **clipvalue** or **clipnorm** argument when creating an optimizer.
- ▶ **clipvalue=1.0** and **clipnorm=1.0**: values between -1.0 and 1.0.
- ▶ **clipvalue=1.0**: $[0.9, 100.0] \Rightarrow [0.9, 1.0]$
- ▶ **clipnorm=1.0**: $[0.9, 100.0] \Rightarrow [0.00899964, 0.9999595]$

Training Speed

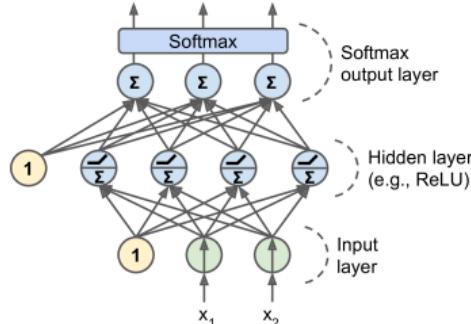




Regular Gradient Descent Optimization (1/2)

- ▶ Gradient descent optimization algorithm
- ▶ It updates the weights $w_i^{(\text{next})} = w_i - \eta \frac{\partial J(w)}{\partial w_i}$
- ▶ Better optimization algorithms to improve the training speed

Regular Gradient Descent Optimization (2/2)



```
n_output = 3
n_hidden = 4
n_features = 2

model = keras.models.Sequential()
model.add(keras.layers.Dense(n_hidden, input_shape=(n_features,), activation="relu"))
model.add(keras.layers.Dense(n_output, activation="softmax"))

model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd", metrics=["accuracy"])
model.fit(X_train, y_train, epochs=30)
```

Optimization Algorithms

- ▶ Momentum
- ▶ Nesterov momentum
- ▶ AdaGrad
- ▶ RMSProp
- ▶ Adam Optimization





Optimization Algorithms

- ▶ Momentum
- ▶ Nesterov momentum
- ▶ AdaGrad
- ▶ RMSProp
- ▶ Adam optimization



Roger Hargreaves

Momentum (1/3)

- ▶ Momentum is a concept from physics: an object in motion will have a tendency to keep moving.
- ▶ It measures the resistance to change in motion.
 - The higher momentum an object has, the harder it is to stop it.



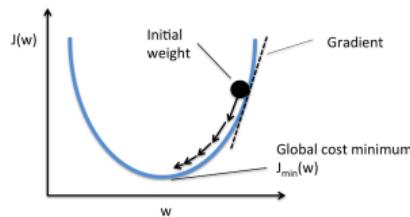


Momentum (2/3)

- ▶ This is the very simple idea behind **momentum optimization**.

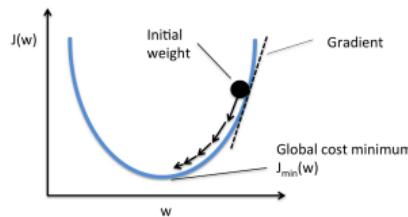
Momentum (2/3)

- ▶ This is the very simple idea behind **momentum optimization**.
- ▶ We can see the **change in the parameters w** as **motion**: $w_i^{(\text{next})} = w_i - \eta \frac{\partial J(w)}{\partial w_i}$



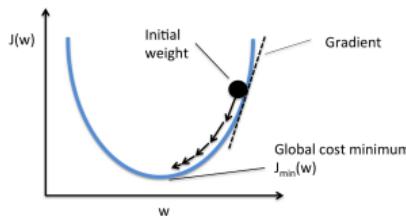
Momentum (2/3)

- ▶ This is the very simple idea behind **momentum optimization**.
- ▶ We can see the **change in the parameters w** as **motion**: $w_i^{(\text{next})} = w_i - \eta \frac{\partial J(w)}{\partial w_i}$
- ▶ We can thus use the concept of momentum to give the update process a **tendency to keep moving** in the same direction.



Momentum (2/3)

- ▶ This is the very simple idea behind **momentum optimization**.
- ▶ We can see the **change in the parameters w** as **motion**: $w_i^{(\text{next})} = w_i - \eta \frac{\partial J(w)}{\partial w_i}$
- ▶ We can thus use the concept of momentum to give the update process a **tendency to keep moving** in the same direction.
- ▶ It can help to **escape from bad local minima pits**.





Momentum (3/3)

- ▶ Regular gradient descent optimization: $w_i^{(\text{next})} = w_i - \eta \frac{\partial J(w)}{\partial w_i}$



Momentum (3/3)

- ▶ Regular gradient descent optimization: $w_i^{(\text{next})} = w_i - \eta \frac{\partial J(w)}{\partial w_i}$
- ▶ **Momentum optimization** cares about what previous gradients were.

Momentum (3/3)

- ▶ Regular gradient descent optimization: $w_i^{(\text{next})} = w_i - \eta \frac{\partial J(w)}{\partial w_i}$
- ▶ **Momentum optimization** cares about what previous gradients were.
- ▶ At each iteration, it adds the local gradient to the momentum vector m .

$$m_i = \beta m_i + \eta \frac{\partial J(w)}{\partial w_i}$$
$$w_i^{(\text{next})} = w_i - m_i$$



Momentum (3/3)

- ▶ Regular gradient descent optimization: $w_i^{(\text{next})} = w_i - \eta \frac{\partial J(w)}{\partial w_i}$
- ▶ **Momentum optimization** cares about what **previous gradients were**.
- ▶ At each iteration, it adds the **local gradient** to the **momentum vector m**.

$$m_i = \beta m_i + \eta \frac{\partial J(w)}{\partial w_i}$$
$$w_i^{(\text{next})} = w_i - m_i$$

- ▶ β is called **momentum**, and it is between 0 and 1.



Momentum (3/3)

- ▶ Regular gradient descent optimization: $w_i^{(\text{next})} = w_i - \eta \frac{\partial J(w)}{\partial w_i}$
- ▶ **Momentum optimization** cares about what previous gradients were.
- ▶ At each iteration, it adds the **local gradient** to the **momentum vector m**.

$$\begin{aligned}m_i &= \beta m_i + \eta \frac{\partial J(w)}{\partial w_i} \\w_i^{(\text{next})} &= w_i - m_i\end{aligned}$$

- ▶ β is called **momentum**, and it is between 0 and 1.

```
optimizer = keras.optimizers.SGD(lr=0.001, momentum=0.9)
model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
```

Optimization Algorithms

- ▶ Momentum
- ▶ Nesterov momentum
- ▶ AdaGrad
- ▶ RMSProp
- ▶ Adam optimization

By Roger Hargreaves



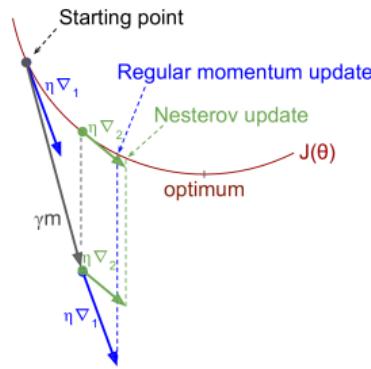


Nesterov Momentum (1/2)

- ▶ Nesterov Momentum is a small variant to Momentum optimization.
- ▶ Faster than vanilla Momentum optimization.

Nesterov Momentum (1/2)

- ▶ Nesterov Momentum is a small variant to Momentum optimization.
- ▶ Faster than vanilla Momentum optimization.
- ▶ ∇_1 represents the gradient of the cost function measured at the starting point w , and ∇_2 represents the gradient at the point located at $w + \beta m$.



Nesterov Momentum (2/2)

- ▶ Measure the gradient of the cost function slightly ahead in the direction of the momentum (not at the local position).

$$\begin{aligned}m_i &= \beta m_i + \eta \frac{\partial J(w + \beta m)}{\partial w_i} \\w_i^{(\text{next})} &= w_i - m_i\end{aligned}$$



Nesterov Momentum (2/2)

- ▶ Measure the gradient of the cost function slightly ahead in the direction of the momentum (not at the local position).

$$\begin{aligned} m_i &= \beta m_i + \eta \frac{\partial J(w + \beta m)}{\partial w_i} \\ w_i^{(\text{next})} &= w_i - m_i \end{aligned}$$

```
optimizer = keras.optimizers.SGD(lr=0.001, momentum=0.9, nesterov=True)
model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
```



Optimization Algorithms

- ▶ Momentum
- ▶ Nesterov momentum
- ▶ AdaGrad
- ▶ RMSProp
- ▶ Adam optimization





AdaGrad (1/2)

- ▶ AdaGrad keeps track of a learning rate for each parameter.
- ▶ Adapts the learning rate over time (adaptive learning rate).
- ▶ Decays the learning rate faster for steep dimensions than for dimensions with gentler slopes.



AdaGrad (2/2)

- ▶ For each feature w_i , we do the following steps:

$$s_i = s_i + \left(\frac{\partial J(w)}{\partial w_i} \right)^2$$

$$w_i^{(\text{next})} = w_i - \frac{\eta}{\sqrt{s_i + \epsilon}} \frac{\partial J(w)}{\partial w_i}$$



AdaGrad (2/2)

- ▶ For each feature w_i , we do the following steps:

$$s_i = s_i + \left(\frac{\partial J(w)}{\partial w_i} \right)^2$$

$$w_i^{(\text{next})} = w_i - \frac{\eta}{\sqrt{s_i + \epsilon}} \frac{\partial J(w)}{\partial w_i}$$

```
optimizer = keras.optimizers.Adagrad(lr=0.001)
model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
```

Optimization Algorithms

- ▶ Momentum
- ▶ Nesterov momentum
- ▶ AdaGrad
- ▶ RMSProp
- ▶ Adam optimization





RMSProp (1/2)

- ▶ AdaGrad often stops too early when training neural networks.
- ▶ The learning rate gets scaled down so much that the algorithm ends up stopping entirely before reaching the global optimum.



RMSProp (1/2)

- ▶ AdaGrad often stops too early when training neural networks.
- ▶ The learning rate gets scaled down so much that the algorithm ends up stopping entirely before reaching the global optimum.
- ▶ The RMSProp fixed the AdaGrad problem.
- ▶ It is like the AdaGrad problem, but accumulates only the gradients from the most recent iterations (not from the beginning of training).



RMSProp (2/2)

- ▶ For each feature w_i , we do the following steps:

$$\begin{aligned}s_i &= \beta s_i + (1 - \beta) \left(\frac{\partial J(w)}{\partial w_i} \right)^2 \\w_i^{(\text{next})} &= w_i - \frac{\eta}{\sqrt{s_i + \epsilon}} \frac{\partial J(w)}{\partial w_i}\end{aligned}$$



RMSProp (2/2)

- ▶ For each feature w_i , we do the following steps:

$$s_i = \beta s_i + (1 - \beta) \left(\frac{\partial J(w)}{\partial w_i} \right)^2$$
$$w_i^{(\text{next})} = w_i - \frac{\eta}{\sqrt{s_i + \epsilon}} \frac{\partial J(w)}{\partial w_i}$$

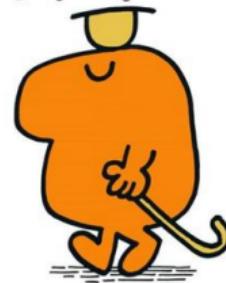
```
optimizer = keras.optimizers.RMSprop(lr=0.001, rho=0.9)
model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
```



Optimization Algorithms

- ▶ Momentum
- ▶ Nesterov momentum
- ▶ AdaGrad
- ▶ RMSProp
- ▶ Adam optimization

by Roger Hargreaves





Adam Optimization (1/3)

- ▶ Adam (Adaptive moment estimation) combines the ideas of Momentum optimization and RMSProp.



Adam Optimization (1/3)

- ▶ Adam (Adaptive moment estimation) combines the ideas of Momentum optimization and RMSProp.
- ▶ Like Momentum optimization, it keeps track of an exponentially decaying average of past gradients.



Adam Optimization (1/3)

- ▶ Adam (Adaptive moment estimation) combines the ideas of Momentum optimization and RMSProp.
- ▶ Like Momentum optimization, it keeps track of an exponentially decaying average of past gradients.
- ▶ Like RMSProp, it keeps track of an exponentially decaying average of past squared gradients.



Adam Optimization (2/3)

$$1. \quad m^{(\text{next})} = \beta_1 m + (1 - \beta_1) \nabla_w J(w)$$

$$2. \quad s^{(\text{next})} = \beta_2 s + (1 - \beta_2) \nabla_w J(w) \otimes \nabla_w J(w)$$

$$3. \quad m^{(\text{next})} = \frac{m}{1 - \beta_1^T}$$

$$4. \quad s^{(\text{next})} = \frac{s}{1 - \beta_2^T}$$

$$5. \quad w^{(\text{next})} = w - \eta m \oslash \sqrt{s + \epsilon}$$

- ▶ \otimes and \oslash represent the element-wise multiplication and division.
- ▶ Steps 1, 2, and 5: similar to both Momentum optimization and RMSProp.
- ▶ Steps 3 and 4: since m and s are initialized at 0, they will be biased toward 0 at the beginning of training, so these two steps will help boost m and s at the beginning of training.



Adam Optimization (3/3)

```
optimizer = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
```



Summary

Summary

- ▶ Overfitting
 - Early stopping, ℓ_1 and ℓ_2 regularization, max-norm regularization
 - Dropout, data augmentation
- ▶ Vanishing gradient
 - Parameter initialization, nonsaturating activation functions
 - Batch normalization, gradient clipping
- ▶ Training speed
 - Momentum, nesterov momentum, AdaGrad
 - RMSProp, Adam optimization





Reference

- ▶ Ian Goodfellow et al., Deep Learning (Ch. 7, 8)
- ▶ Aurélien Géron, Hands-On Machine Learning (Ch. 11)

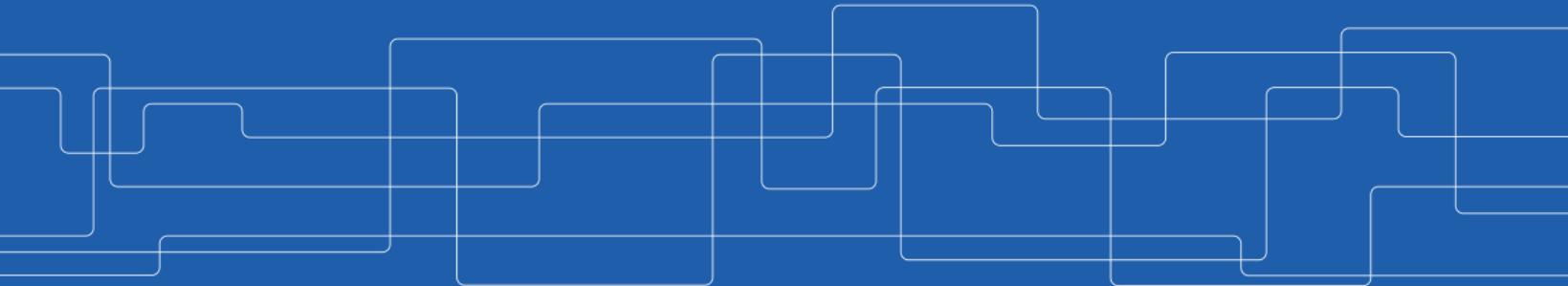


Questions?



Convolutional Neural Networks

Amir H. Payberah
payberah@kth.se
2021-11-25



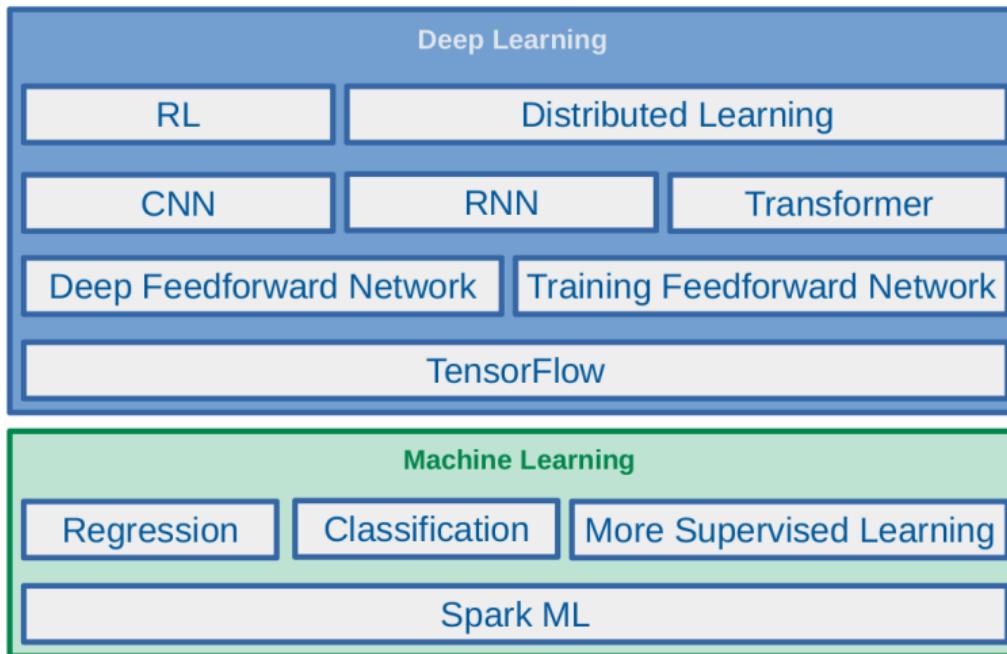


The Course Web Page

<https://id2223kth.github.io>
<https://tinyurl.com/6s5jy46a>

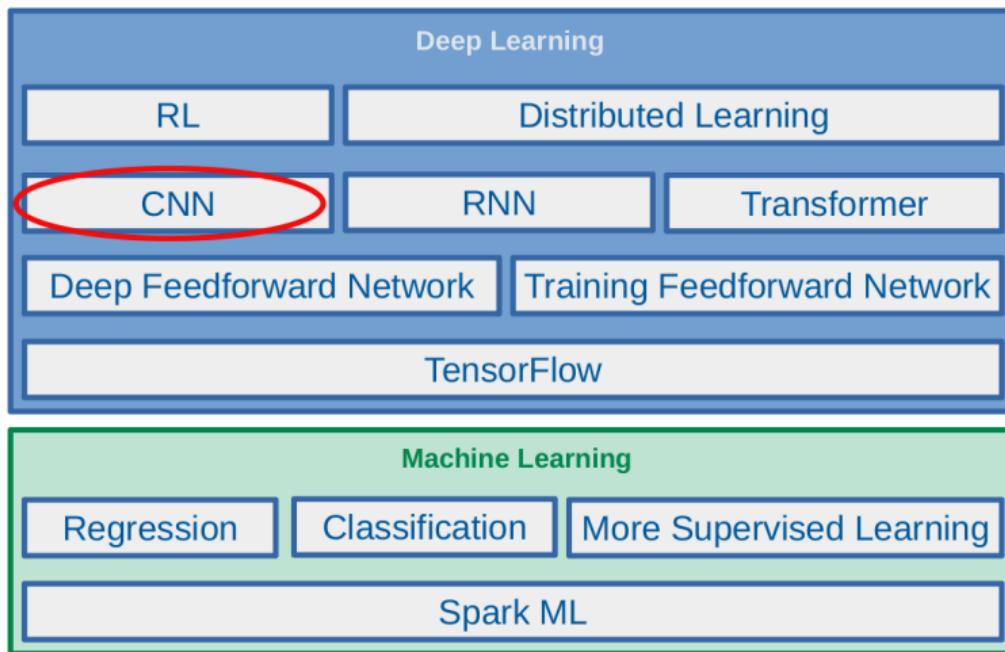


Where Are We?





Where Are We?





Let's Start With An Example



MNIST Dataset

- ▶ Handwritten digits in the [MNIST](#) dataset are 28x28 pixel greyscale images.

A 10x10 grid of handwritten digits, likely from the MNIST dataset, displayed in a black font on a white background. The digits are arranged in ten rows and ten columns. Each row contains a different digit, starting with '0' at the top and ending with '9' at the bottom. The digits are somewhat stylized and vary slightly in orientation and stroke thickness.

0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9



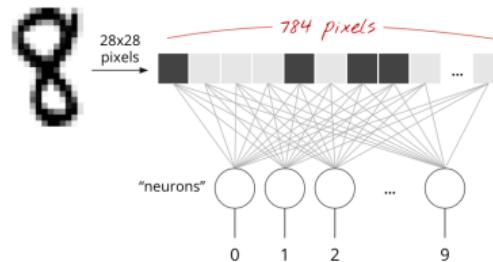
One-Layer Network For Classifying MNIST (1/4)



[<https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd>]

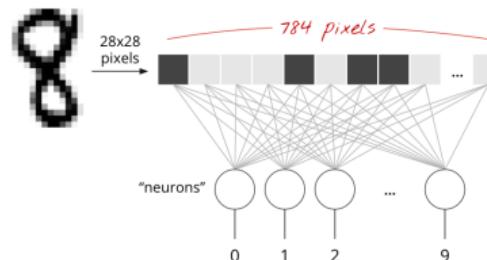
One-Layer Network For Classifying MNIST (2/4)

- ▶ Let's make a **one-layer** neural network for **classifying digits**.



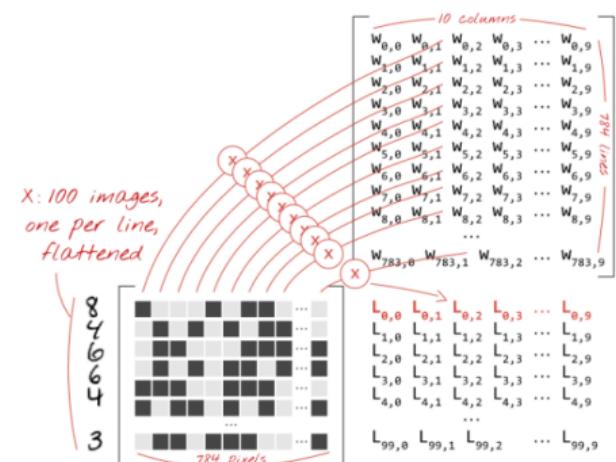
One-Layer Network For Classifying MNIST (2/4)

- ▶ Let's make a **one-layer** neural network for **classifying digits**.
- ▶ Each **neuron** in a neural network:
 - Does a **weighted sum** of all of its inputs
 - Adds a **bias**
 - Feeds the result through some **non-linear activation** function, e.g., **softmax**.



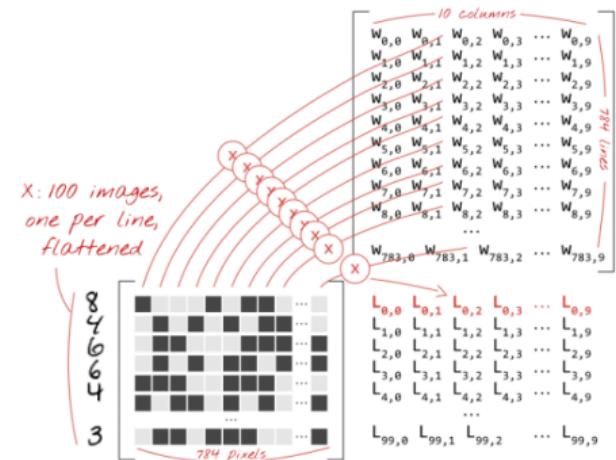
One-Layer Network For Classifying MNIST (3/4)

- ▶ Assume we have a **batch of 100 images** as the **input**.



One-Layer Network For Classifying MNIST (3/4)

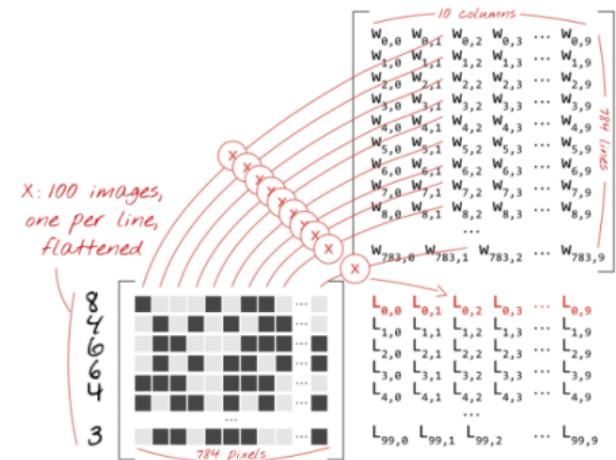
- ▶ Assume we have a **batch of 100 images** as the **input**.
- ▶ Using the **first column** of the **weights matrix W** , we compute the **weighted sum** of all the **pixels** of the **first image**.



One-Layer Network For Classifying MNIST (3/4)

- ▶ Assume we have a **batch of 100 images** as the **input**.
- ▶ Using the **first column** of the **weights matrix W** , we compute the **weighted sum** of all the **pixels** of the **first image**.
 - The **first neuron**:

$$L_{0,0} = w_{0,0}x_0^{(1)} + w_{1,0}x_1^{(1)} + \dots + w_{783,0}x_{783}^{(1)}$$



One-Layer Network For Classifying MNIST (3/4)

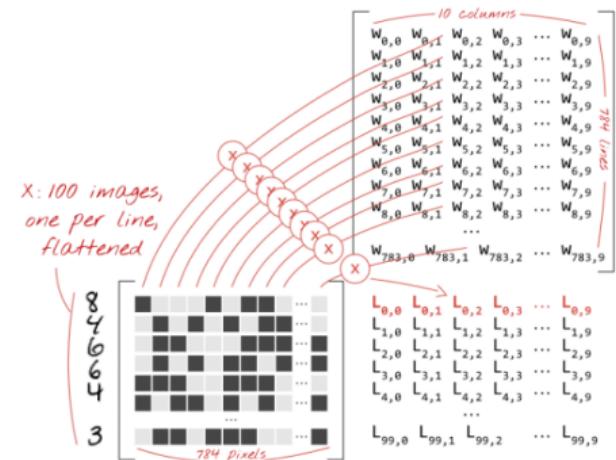
- ▶ Assume we have a **batch of 100 images** as the **input**.
- ▶ Using the **first column** of the **weights matrix W** , we compute the **weighted sum** of all the **pixels** of the **first image**.
 - The **first neuron**:

$$L_{0,0} = w_{0,0}x_0^{(1)} + w_{1,0}x_1^{(1)} + \dots + w_{783,0}x_{783}^{(1)}$$
 - The **2nd neuron until the 10th**:

$$L_{0,1} = w_{0,1}x_0^{(1)} + w_{1,1}x_1^{(1)} + \dots + w_{783,1}x_{783}^{(1)}$$

$$\dots$$

$$L_{0,9} = w_{0,9}x_0^{(1)} + w_{1,9}x_1^{(1)} + \dots + w_{783,9}x_{783}^{(1)}$$



One-Layer Network For Classifying MNIST (3/4)

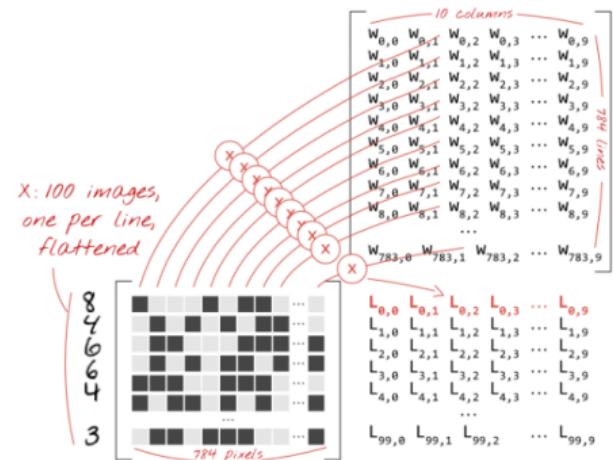
- ▶ Assume we have a **batch of 100 images** as the **input**.
- ▶ Using the **first column** of the **weights matrix W** , we compute the **weighted sum** of all the **pixels** of the **first image**.
 - The **first neuron**:

$$L_{0,0} = w_{0,0}x_0^{(1)} + w_{1,0}x_1^{(1)} + \dots + w_{783,0}x_{783}^{(1)}$$
 - The **2nd neuron until the 10th**:

$$L_{0,1} = w_{0,1}x_0^{(1)} + w_{1,1}x_1^{(1)} + \dots + w_{783,1}x_{783}^{(1)}$$

 \dots

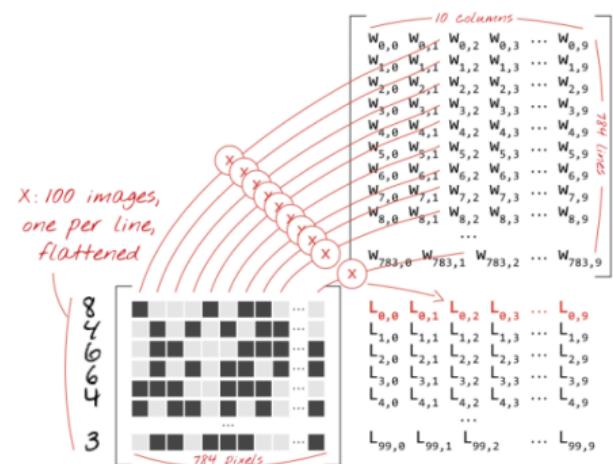
$$L_{0,9} = w_{0,9}x_0^{(1)} + w_{1,9}x_1^{(1)} + \dots + w_{783,9}x_{783}^{(1)}$$
 - Repeat the operation for the **other 99 images**, i.e., $x^{(2)} \dots x^{(100)}$



One-Layer Network For Classifying MNIST (4/4)

- ▶ Each neuron must now add its **bias**.
- ▶ Apply the **softmax activation function** for each instance $x^{(i)}$.

▶ For each input instance $x^{(i)}$: $L_i = \begin{bmatrix} L_{i,0} \\ L_{i,1} \\ \vdots \\ L_{i,9} \end{bmatrix}$

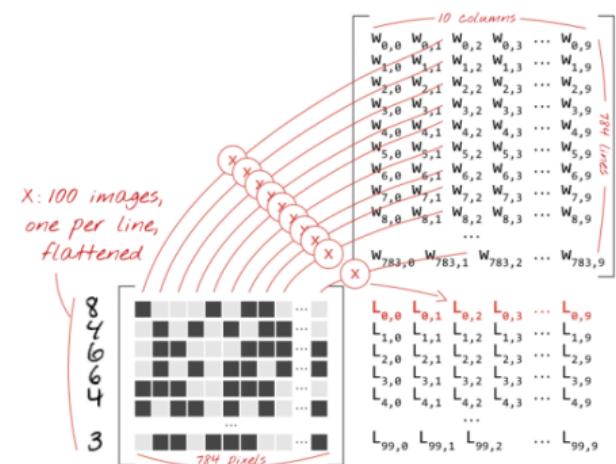


One-Layer Network For Classifying MNIST (4/4)

- ▶ Each neuron must now add its **bias**.
- ▶ Apply the **softmax activation function** for each instance $x^{(i)}$.

- ▶ For each input instance $x^{(i)}$: $L_i = \begin{bmatrix} L_{i,0} \\ L_{i,1} \\ \vdots \\ L_{i,9} \end{bmatrix}$

- ▶ $\hat{y}_i = \text{softmax}(L_i + b)$



One-Layer Network For Classifying MNIST (4/4)

- ▶ Each neuron must now add its **bias**.
- ▶ Apply the **softmax activation function** for each instance $x^{(i)}$.

▶ For each input instance $x^{(i)}$: $L_i = \begin{bmatrix} L_{i,0} \\ L_{i,1} \\ \vdots \\ L_{i,9} \end{bmatrix}$

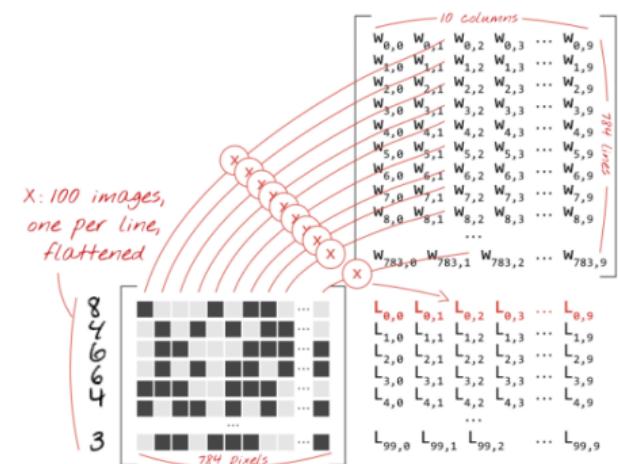
▶ $\hat{y}_i = \text{softmax}(L_i + b)$

$$Y = \text{softmax}(X \cdot W + b)$$

Predictions $Y[100, 10]$
 Images $X[100, 784]$
 Weights $W[784, 10]$
 Biases $b[10]$

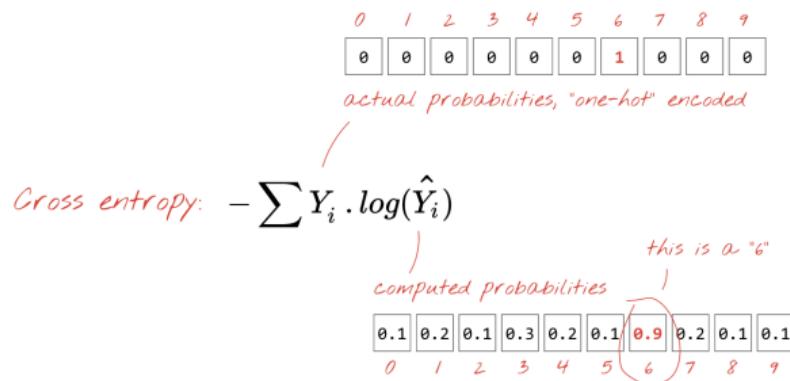
applied line by line
 matrix multiply
 broadcast on all lines

tensor shapes in []



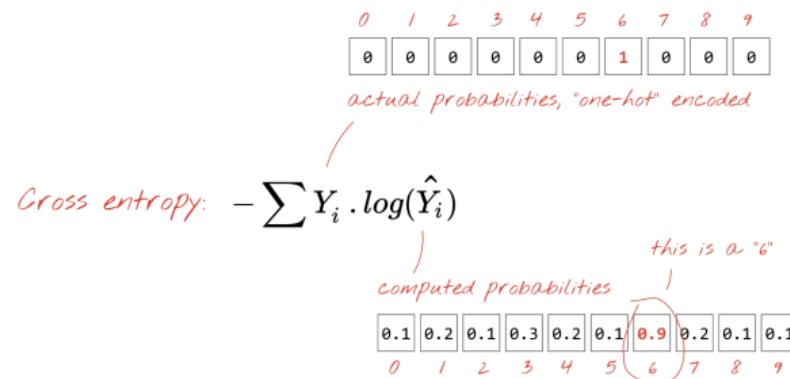
How Good the Predictions Are?

- ▶ Define the cost function $J(W)$ as the **cross-entropy** of what the network tells us (\hat{y}_i) and what we know to be the truth (y_i), for each instance $x^{(i)}$.



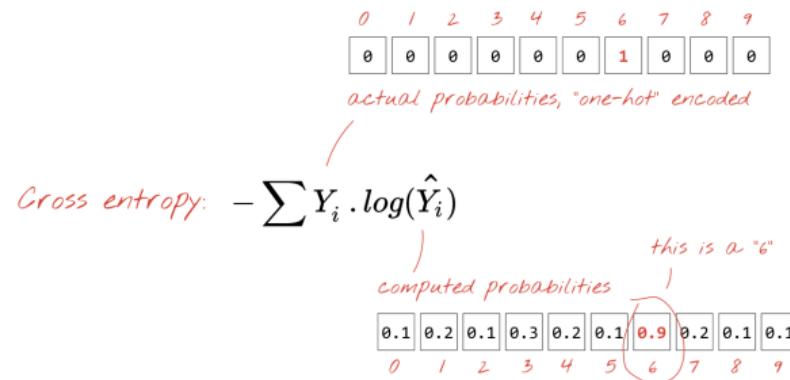
How Good the Predictions Are?

- ▶ Define the cost function $J(W)$ as the **cross-entropy** of what the network tells us (\hat{y}_i) and what we know to be the truth (y_i), for each instance $x^{(i)}$.
- ▶ Compute the **partial derivatives** of the cross-entropy with respect to all the **weights** and all the **biases**, $\nabla_W J(W)$.



How Good the Predictions Are?

- ▶ Define the cost function $J(W)$ as the **cross-entropy** of what the network tells us (\hat{y}_i) and what we know to be the truth (y_i), for each instance $x^{(i)}$.
- ▶ Compute the **partial derivatives** of the cross-entropy with respect to all the **weights** and all the **biases**, $\nabla_W J(W)$.
- ▶ Update weights and biases by a **fraction of the gradient** $W^{(\text{next})} = W - \eta \nabla_W J(W)$





```
mnist = tf.keras.datasets.mnist  
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```



```
mnist = tf.keras.datasets.mnist  
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
model = tf.keras.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28, 28)),  
    tf.keras.layers.Dense(10, activation='softmax')  
])
```



```
mnist = tf.keras.datasets.mnist  
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
model = tf.keras.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28, 28)),  
    tf.keras.layers.Dense(10, activation='softmax')  
])
```

```
model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```



```
mnist = tf.keras.datasets.mnist  
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
model = tf.keras.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28, 28)),  
    tf.keras.layers.Dense(10, activation='softmax')  
])
```

```
model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
model.fit(x_train, y_train, batch_size=100, epochs=10)  
model.evaluate(x_test, y_test)
```

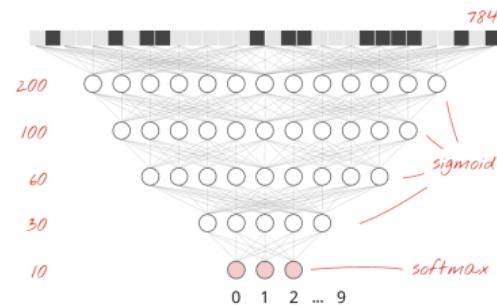


Some Improvement (1/5)

- ▶ Add more layers to **improve** the accuracy.
- ▶ On **intermediate layers** we will use the the **sigmoid** activation function.
- ▶ We keep **softmax** as the activation function on the **last layer**.



[<https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd>]



Some Improvement (2/5)

- ▶ Network **initialization**. e.g., using **He** initialization.
- ▶ Better **optimizer**, e.g., using **Adam** optimizer.



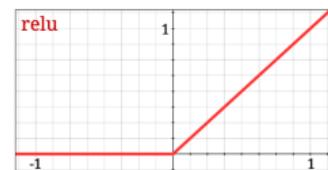
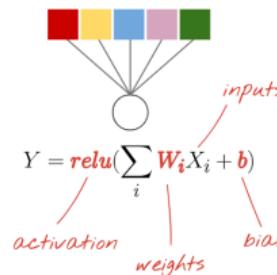
[<https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd>]

Some Improvement (3/5)

- ▶ Better activation function, e.g., using $\text{ReLU}(z) = \max(0, z)$.

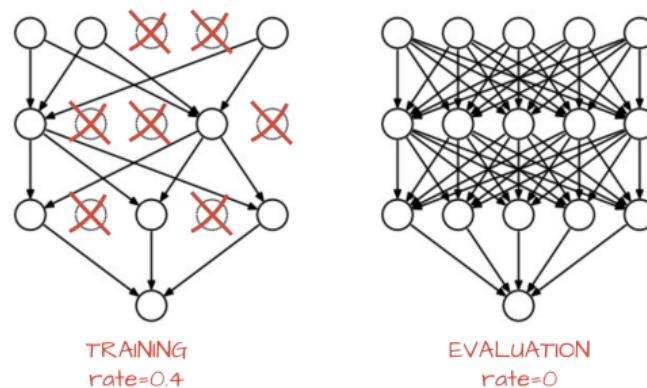


[<https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd>]



Some Improvement (4/5)

- ▶ Overcome **overfitting**, e.g., using **dropout**.



[<https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd>]

Some Improvement (5/5)

- ▶ Start fast and **decay** the learning rate exponentially.
- ▶ You can do this with the `tf.keras.callbacks.LearningRateScheduler` callback.



[<https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd>]



```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, kernel_initializer="he_normal", activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```



```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, kernel_initializer="he_normal", activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

```
# lr decay function
def lr_decay(epoch):
    return 0.01 * math.pow(0.6, epoch)

# lr schedule callback
lr_decay_callback = tf.keras.callbacks.LearningRateScheduler(lr_decay, verbose=True)

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'],
              callbacks=[lr_decay_callback])
```



```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, kernel_initializer="he_normal", activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

```
# lr decay function
def lr_decay(epoch):
    return 0.01 * math.pow(0.6, epoch)

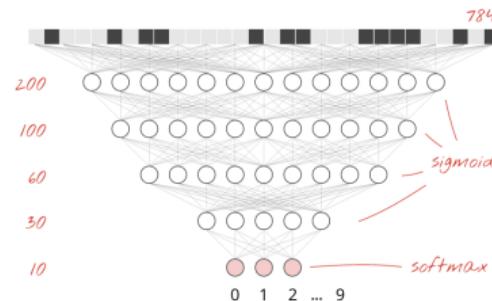
# lr schedule callback
lr_decay_callback = tf.keras.callbacks.LearningRateScheduler(lr_decay, verbose=True)

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'],
              callbacks=[lr_decay_callback])
```

```
model.fit(x_train, y_train, batch_size=100, epochs=10)
model.evaluate(x_test, y_test)
```

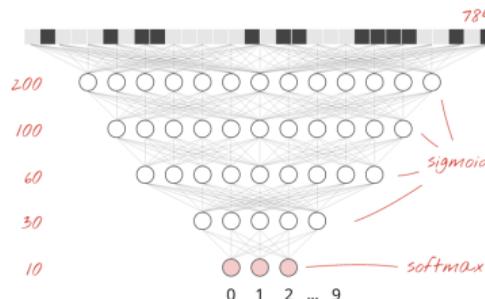
Vanilla Deep Neural Networks Challenges (1/2)

- ▶ Pixels of each image were flattened into a single vector (really **bad idea**).



Vanilla Deep Neural Networks Challenges (1/2)

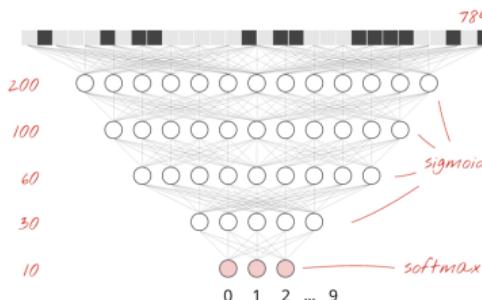
- ▶ Pixels of each image were flattened into a single vector (really **bad idea**).



- ▶ Vanilla deep neural networks **do not scale**.
 - In MNIST, images are black-and-white 28×28 pixel images: $28 \times 28 = 784$ weights.

Vanilla Deep Neural Networks Challenges (1/2)

- ▶ Pixels of each image were flattened into a single vector (really bad idea).



- ▶ Vanilla deep neural networks do not scale.
 - In MNIST, images are black-and-white 28×28 pixel images: $28 \times 28 = 784$ weights.
- ▶ Handwritten digits are made of shapes and we discarded the shape information when we flattened the pixels.

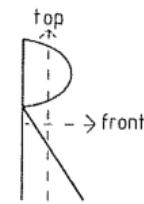
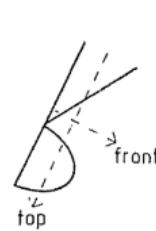


Vanilla Deep Neural Networks Challenges (2/2)

- ▶ Difficult to recognize objects.

Vanilla Deep Neural Networks Challenges (2/2)

- ▶ Difficult to **recognize** objects.
- ▶ **Rotation**
- ▶ **Lighting**: objects may **look different** depending on the level of **external lighting**.
- ▶ **Deformation**: objects can be deformed in a variety of **non-affine ways**.
- ▶ **Scale variation**: visual classes often exhibit **variation** in their size.
- ▶ **Viewpoint invariance**.





Tackle the Challenges

- ▶ Convolutional neural networks (CNN) can tackle the vanilla model challenges.
- ▶ CNN is a type of neural network that can take advantage of shape information.



Tackle the Challenges

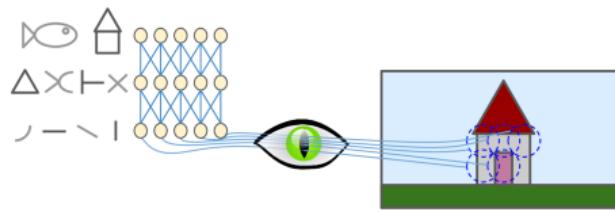
- ▶ Convolutional neural networks (CNN) can tackle the vanilla model challenges.
- ▶ CNN is a type of neural network that can take advantage of shape information.
- ▶ It applies a series of filters to the raw pixel data of an image to extract and learn higher-level features, which the model can then use for classification.



Filters and Convolution Operations

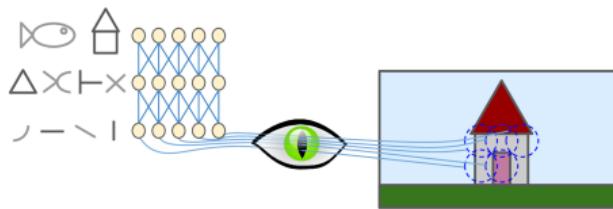
Brain Visual Cortex Inspired CNNs

- ▶ 1959, David H. Hubel and Torsten Wiesel.
- ▶ Many **neurons in the visual cortex** have a **small local receptive field**.



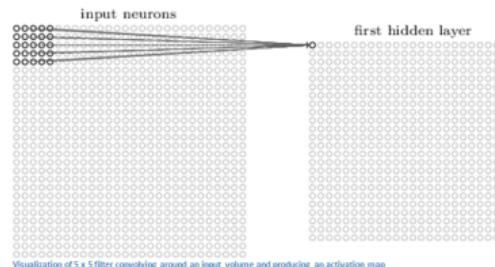
Brain Visual Cortex Inspired CNNs

- ▶ 1959, David H. Hubel and Torsten Wiesel.
- ▶ Many **neurons in the visual cortex** have a **small local receptive field**.
- ▶ They **react** only to visual stimuli located in a **limited region of the visual field**.



Receptive Fields and Filters

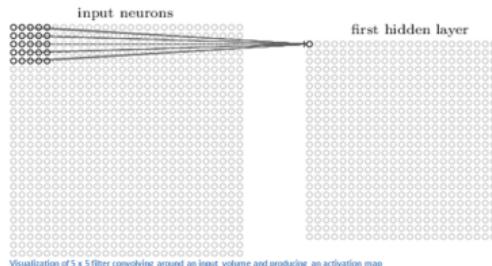
- ▶ Imagine a **flashlight** that is shining over the top left of the image.



[<https://adshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks>]

Receptive Fields and Filters

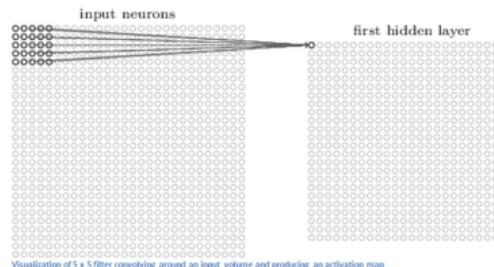
- ▶ Imagine a **flashlight** that is shining over the top left of the image.
- ▶ The **region that it is shining over** is called the **receptive field**.
- ▶ This **flashlight** is called a **filter**.



[<https://adshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks>]

Receptive Fields and Filters

- ▶ Imagine a **flashlight** that is shining over the top left of the image.
- ▶ The **region that it is shining over** is called the **receptive field**.
- ▶ This **flashlight** is called a **filter**.
- ▶ A filter is a **set of weights**.
- ▶ A **filter** is a **feature detector**, e.g., straight edges, simple colors, and curves.

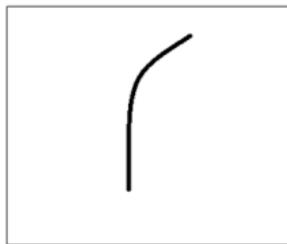


[<https://adshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks>]

Filters Example (1/3)

0	0	0	0	0	30	0	0
0	0	0	0	30	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter



Filters Example (1/3)

0	0	0	0	0	30	0	0
0	0	0	0	30	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	0	0	0	0	0

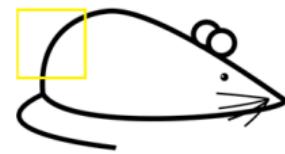
Pixel representation of filter



Visualization of a curve detector filter



Original image



Visualization of the filter on the image

[<https://adेशपांडे3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks>]

Filters Example (2/3)



Visualization of the receptive field

0	0	0	0	0	0	30	0
0	0	0	0	50	50	50	0
0	0	0	20	50	0	0	0
0	0	0	50	50	0	0	0
0	0	0	50	50	0	0	0
0	0	0	50	50	0	0	0
0	0	0	50	50	0	0	0
0	0	0	50	50	0	0	0

Pixel representation of the receptive field

*

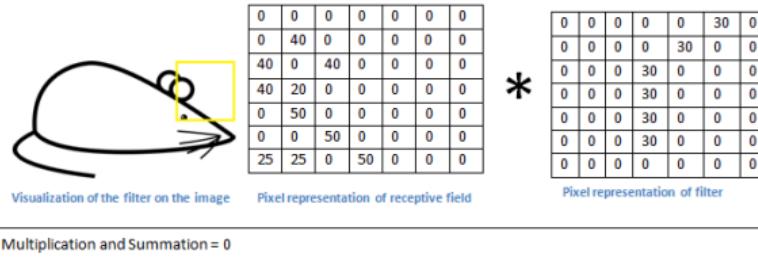
0	0	0	0	0	30	0	0
0	0	0	0	30	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Pixel representation of filter

$$\text{Multiplication and Summation} = (50 \cdot 30) + (50 \cdot 30) + (50 \cdot 30) + (20 \cdot 30) + (50 \cdot 30) = 6600 \text{ (A large number!)}$$

[<https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks>]

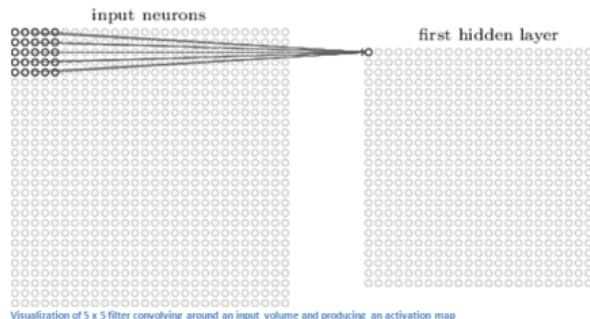
Filters Example (3/3)



[<https://adephante3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks>]

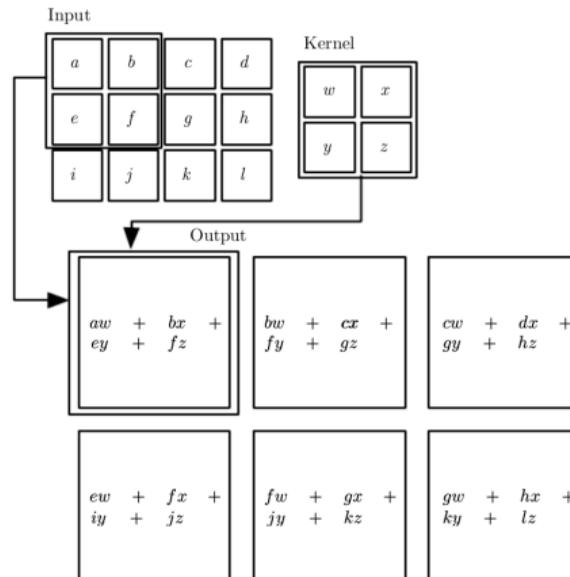
Convolution Operation

- ▶ Convolution takes a filter and multiplying it over the entire area of an input image.
- ▶ Imagine this flashlight (filter) sliding across all the areas of the input image.



[<https://adephante3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks>]

Convolution Operation - 2D Example

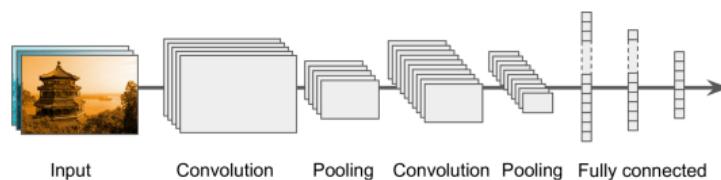




Convolutional Neural Network (CNN)

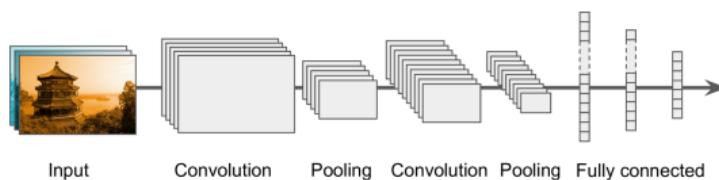
CNN Components (1/2)

- ▶ **Convolutional layers**: apply a specified number of **convolution filters** to the image.



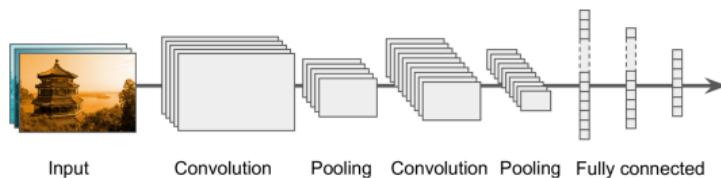
CNN Components (1/2)

- ▶ **Convolutional layers**: apply a specified number of **convolution filters** to the image.
- ▶ **Pooling layers**: **downsample the image** data extracted by the convolutional layers to **reduce the dimensionality** of the feature map in order to decrease processing time.



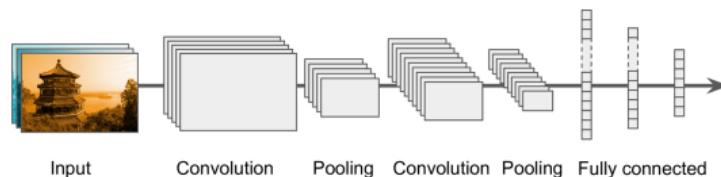
CNN Components (1/2)

- ▶ **Convolutional layers:** apply a specified number of **convolution filters** to the image.
- ▶ **Pooling layers:** **downsample the image** data extracted by the convolutional layers to **reduce the dimensionality** of the feature map in order to decrease processing time.
- ▶ **Dense layers:** a **fully connected layer** that performs **classification** on the features extracted by the convolutional layers and downsampled by the pooling layers.



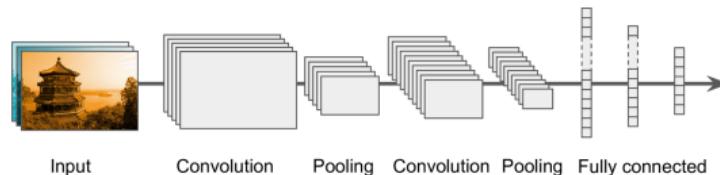
CNN Components (2/2)

- ▶ A CNN is composed of a stack of convolutional modules.



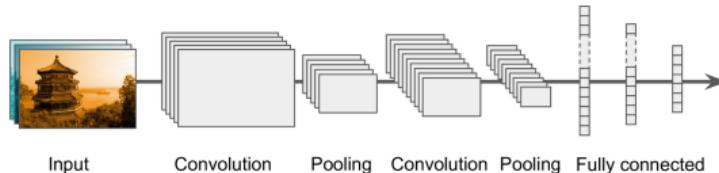
CNN Components (2/2)

- ▶ A **CNN** is composed of a **stack of convolutional modules**.
- ▶ Each **module** consists of a **convolutional layer** followed by a **pooling layer**.



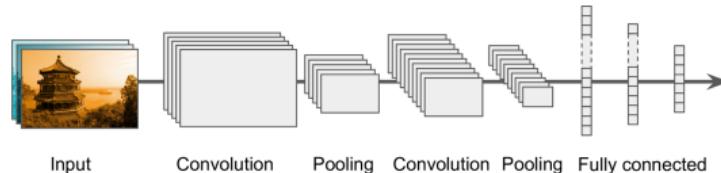
CNN Components (2/2)

- ▶ A **CNN** is composed of a **stack of convolutional modules**.
- ▶ Each **module** consists of a **convolutional layer** followed by a **pooling layer**.
- ▶ The **last module** is followed by **one or more dense layers** that perform **classification**.

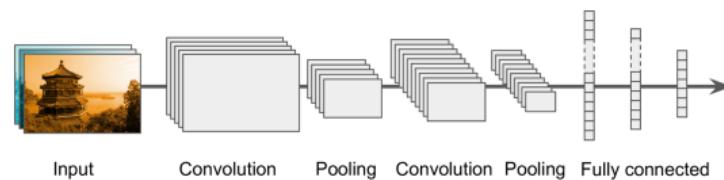


CNN Components (2/2)

- ▶ A **CNN** is composed of a **stack of convolutional modules**.
- ▶ Each **module** consists of a **convolutional layer** followed by a **pooling layer**.
- ▶ The **last module** is followed by **one or more dense layers** that perform **classification**.
- ▶ The **final dense layer** contains a **single node** for each target class in the model, with a **softmax** activation function.

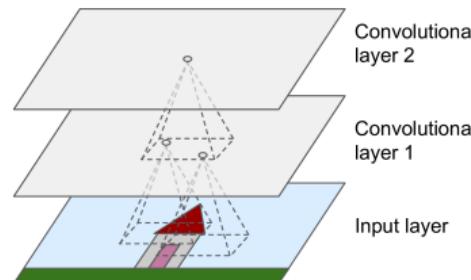


Convolutional Layer



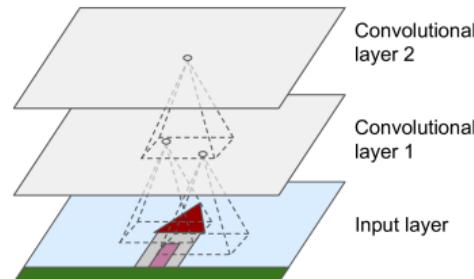
Convolutional Layer (1/4)

- ▶ Sparse interactions
- ▶ Each neuron in the convolutional layers is only connected to pixels in its receptive field (not every single pixel).



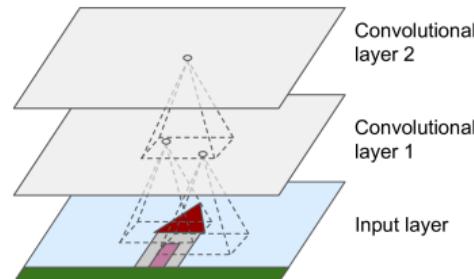
Convolutional Layer (2/4)

- ▶ Each neuron applies **filters** on its **receptive field**.



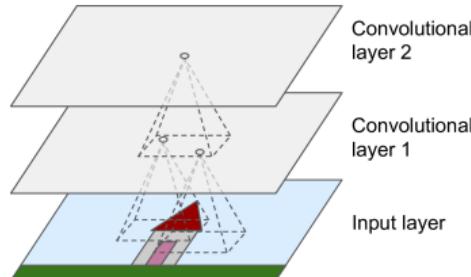
Convolutional Layer (2/4)

- ▶ Each neuron applies **filters** on its **receptive field**.
 - Calculates a **weighted sum** of the input pixels in the receptive field.



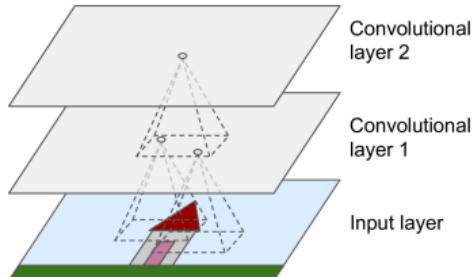
Convolutional Layer (2/4)

- ▶ Each neuron applies **filters** on its **receptive field**.
 - Calculates a **weighted sum** of the input pixels in the receptive field.
- ▶ Adds a **bias**, and feeds the result through its **activation function** to the next layer.



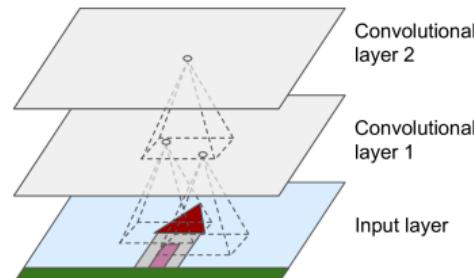
Convolutional Layer (2/4)

- ▶ Each neuron applies **filters** on its **receptive field**.
 - Calculates a **weighted sum** of the input pixels in the receptive field.
- ▶ Adds a **bias**, and feeds the result through its **activation function** to the next layer.
- ▶ The **output** of this layer is a **feature map (activation map)**



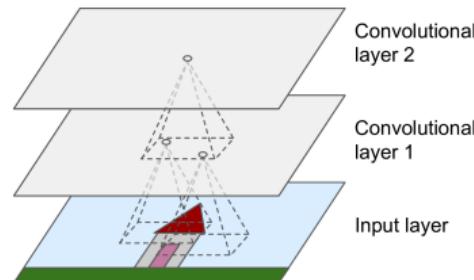
Convolutional Layer (3/4)

- ▶ Parameter sharing
- ▶ All neurons of a convolutional layer reuse the same weights.



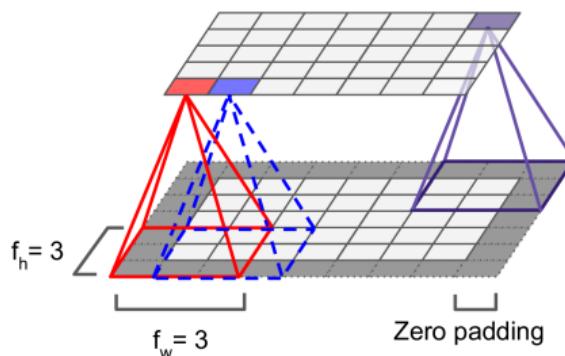
Convolutional Layer (3/4)

- ▶ Parameter sharing
- ▶ All neurons of a convolutional layer reuse the same weights.
- ▶ They apply the same filter in different positions.
- ▶ Whereas in a fully-connected network, each neuron had its own set of weights.



Convolutional Layer (4/4)

- ▶ Assume the filter size (kernel size) is $f_w \times f_h$.
 - f_h and f_w are the height and width of the receptive field, respectively.
- ▶ A neuron in row i and column j of a given layer is connected to the outputs of the neurons in the previous layer in rows i to $i + f_h - 1$, and columns j to $j + f_w - 1$.



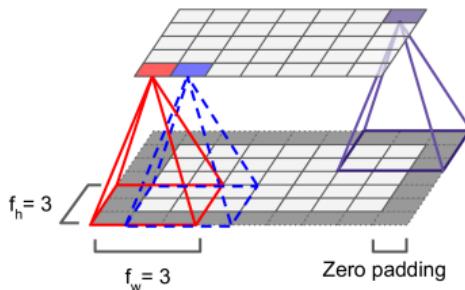


Padding

- ▶ What will happen if you apply a **5x5 filter** to a **32x32 input** volume?
 - The output volume would be **28x28**.
 - The spatial **dimensions decrease**.

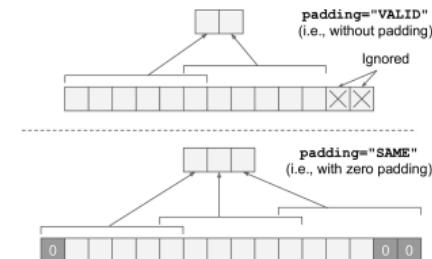
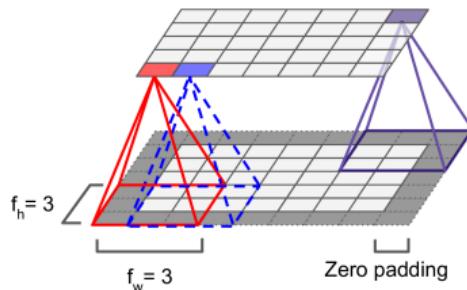
Padding

- ▶ What will happen if you apply a **5x5 filter** to a **32x32 input volume**?
 - The output volume would be **28x28**.
 - The spatial **dimensions decrease**.
- ▶ **Zero padding:** in order for a layer to have the **same height and width** as the previous layer, it is common to **add zeros around the inputs**.

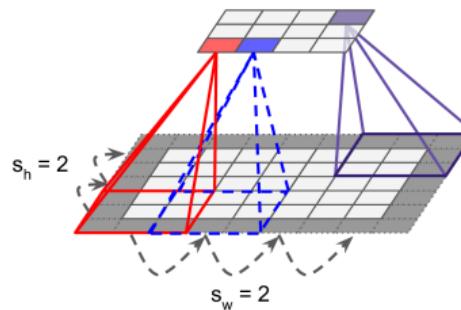


Padding

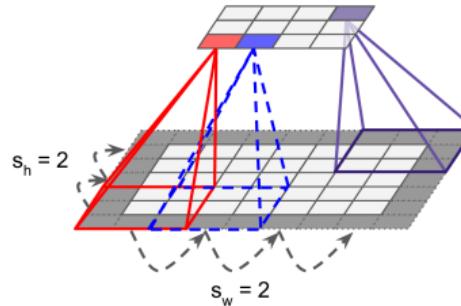
- ▶ What will happen if you apply a **5x5 filter** to a **32x32 input** volume?
 - The output volume would be **28x28**.
 - The spatial **dimensions decrease**.
- ▶ **Zero padding**: in order for a layer to have the **same height and width** as the previous layer, it is common to **add zeros around the inputs**.
- ▶ In **TensorFlow**, padding can be either **SAME** or **VALID** to have zero padding or not.



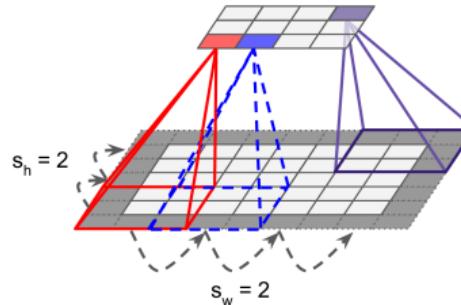
- ▶ The **distance** between two consecutive receptive fields is called the **stride**.



- ▶ The **distance** between two consecutive receptive fields is called the **stride**.
- ▶ The stride controls **how** the filter convolves around the input volume.

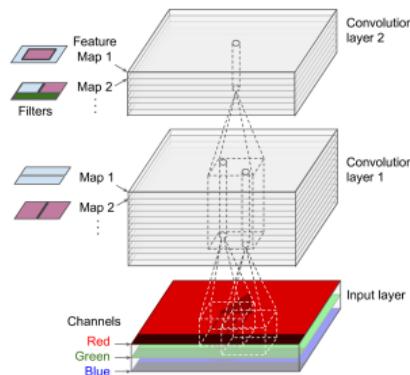


- ▶ The **distance** between two consecutive receptive fields is called the **stride**.
- ▶ The stride controls **how the filter convolves** around the input volume.
- ▶ Assume s_h and s_w are the **vertical and horizontal strides**, then, a neuron located in **row i** and **column j** in a layer is connected to the outputs of the neurons in the **previous layer** located in **rows $i \times s_h$** to $i \times s_h + f_h - 1$, and **columns $j \times s_w$** to $j \times s_w + f_w - 1$.



Stacking Multiple Feature Maps

- ▶ Up to now, we represented each convolutional layer with a **single feature map**.
- ▶ Each convolutional layer can be composed of **several feature maps** of equal sizes.
- ▶ Input images are also composed of **multiple sublayers**: **one per color channel**.
- ▶ A **convolutional layer simultaneously applies multiple filters** to its inputs.

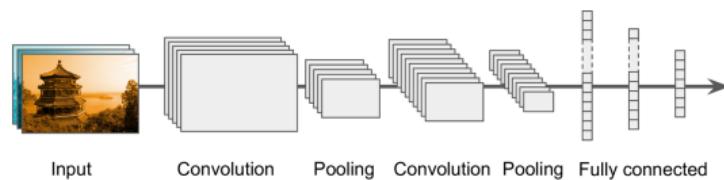




Activation Function

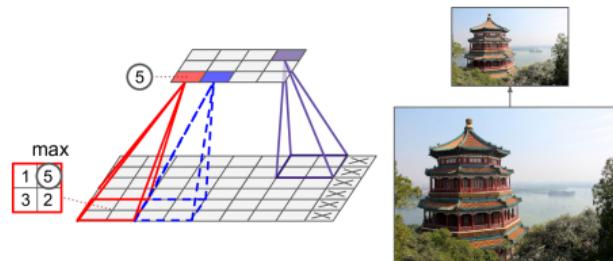
- ▶ After calculating a **weighted sum** of the input pixels in the **receptive fields**, and adding **biases**, each neuron feeds the result through its **ReLU activation function** to the next layer.
- ▶ The purpose of this activation function is to add **non linearity** to the system.

Pooling Layer



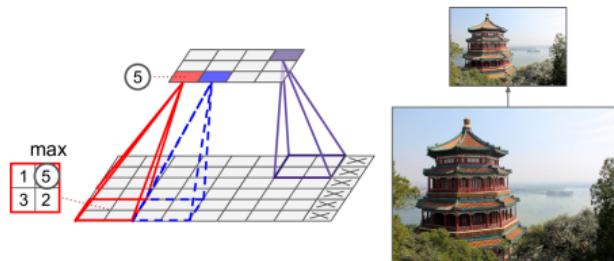
Pooling Layer (1/2)

- ▶ After the activation functions, we can apply a **pooling layer**.
- ▶ Its goal is to **subsample (shrink)** the input image.



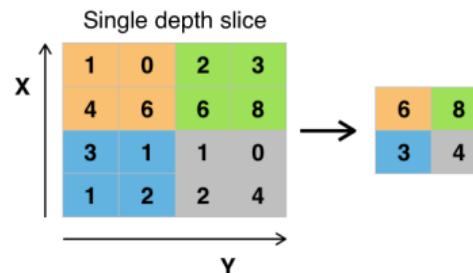
Pooling Layer (1/2)

- ▶ After the activation functions, we can apply a **pooling layer**.
- ▶ Its goal is to **subsample (shrink)** the input image.
 - To **reduce** the computational load, the memory usage, and the number of parameters.



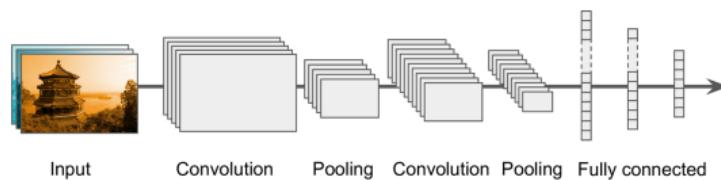
Pooling Layer (2/2)

- ▶ Each **neuron** in a pooling layer is connected to the outputs of a **receptive field** in the previous layer.
- ▶ A pooling neuron has **no weights**.
- ▶ It **aggregates the inputs** using an aggregation function such as the **max** or **mean**.



Example of Maxpool with a 2x2 filter and a stride of 2

Fully Connected Layer





Fully Connected Layer

- ▶ This layer takes an input from the **last convolution module**, and outputs an **N** dimensional vector.
 - **N** is the **number of classes** that the model has to choose from.



Fully Connected Layer

- ▶ This layer takes an input from the **last convolution module**, and outputs an **N** dimensional vector.
 - **N** is the **number of classes** that the model has to choose from.
- ▶ For example, if you wanted a **digit classification** model, **N** would be 10.



Fully Connected Layer

- ▶ This layer takes an input from the **last convolution module**, and outputs an **N** dimensional vector.
 - **N** is the **number of classes** that the model has to choose from.
- ▶ For example, if you wanted a **digit classification** model, **N would be 10**.
- ▶ Each number in this **N** dimensional vector represents the **probability of a certain class**.



Flattening

- ▶ We need to **convert the output** of the convolutional part of the CNN into a **1D feature vector**.
- ▶ This operation is called **flattening**.



Flattening

- ▶ We need to **convert the output** of the convolutional part of the CNN into a **1D feature vector**.
- ▶ This operation is called **flattening**.
- ▶ It gets the **output of the convolutional layers**, **flattens** all its structure to create a **single long feature vector** to be used by the **dense layer** for the final classification.



Example

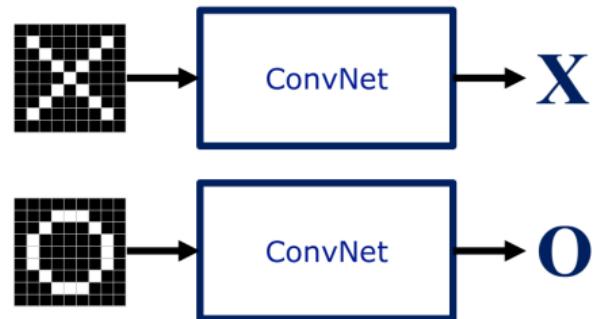


A Toy ConvNet: X's and O's

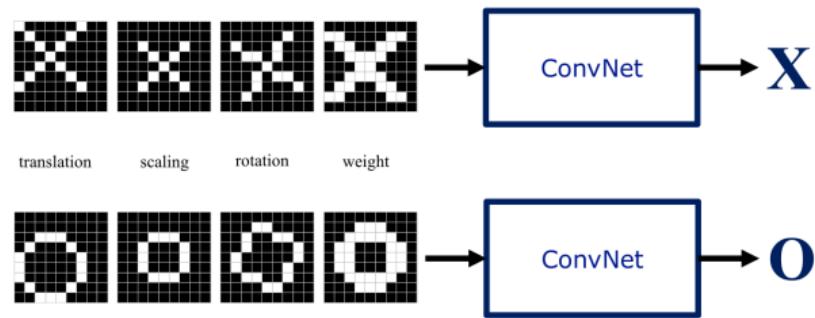
A two-dimensional
array of pixels



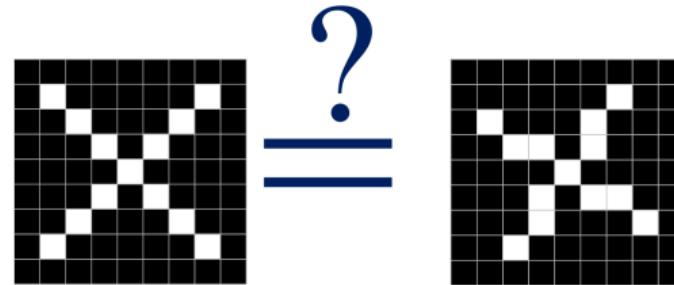
For Example



Trickier Cases



Deciding is Hard





What Computers See



-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	1	-1
-1	1	1	-1	-1	-1	1	-1	-1	
-1	-1	1	-1	-1	-1	1	-1	-1	
-1	-1	-1	1	-1	1	-1	-1	-1	
-1	-1	-1	-1	1	-1	-1	-1	-1	
-1	-1	-1	1	-1	1	-1	-1	-1	
-1	-1	1	-1	-1	-1	1	-1	-1	
-1	1	-1	-1	-1	-1	1	-1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

What Computers See



-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	1	1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	1	-1	-1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	X	-1	-1	-1	-1	X	X	-1
-1	X	X	-1	-1	X	X	-1	-1
-1	-1	X	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	X	-1	-1
-1	-1	X	X	-1	-1	X	X	-1
-1	X	X	-1	-1	-1	X	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

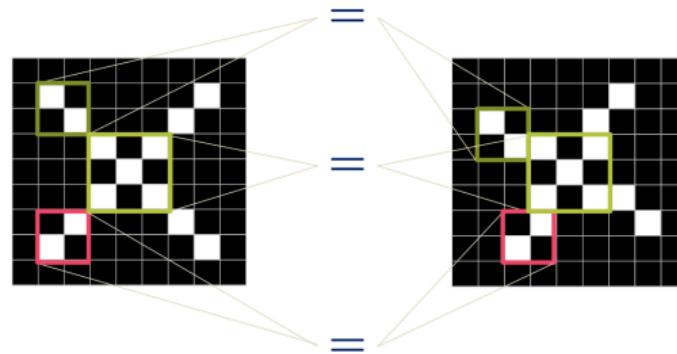
Computers are Literal

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	1	-1	-1



-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	1	1	-1	1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1	-1

ConvNets Match Pieces of the Image



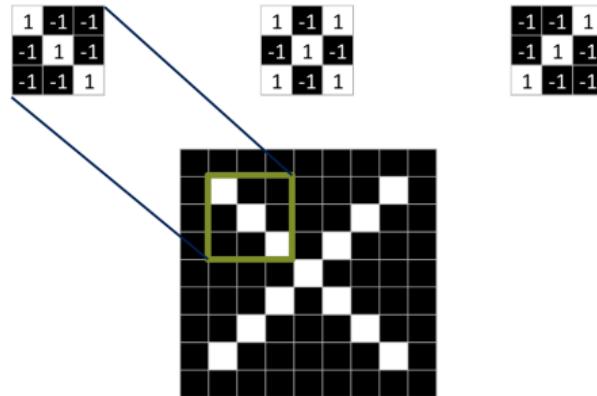
Filters Match Pieces of the Image

$$\begin{array}{|c|c|c|} \hline 1 & -1 & -1 \\ \hline -1 & 1 & -1 \\ \hline -1 & -1 & 1 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline 1 & -1 & 1 \\ \hline -1 & 1 & -1 \\ \hline 1 & -1 & 1 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline -1 & -1 & 1 \\ \hline -1 & 1 & -1 \\ \hline 1 & -1 & -1 \\ \hline \end{array}$$

Filters Match Pieces of the Image

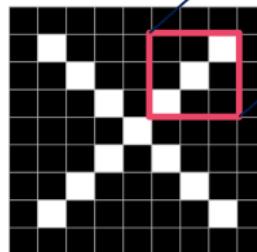


Filters Match Pieces of the Image

$$\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix}$$

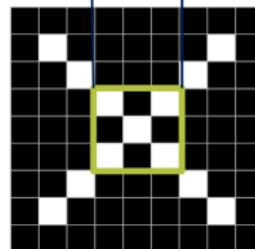


Filters Match Pieces of the Image

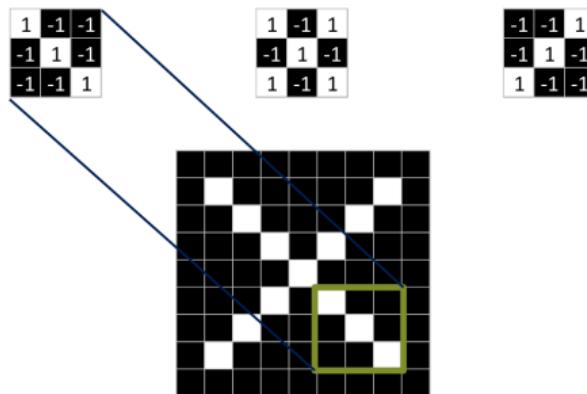
$$\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix}$$



Filters Match Pieces of the Image

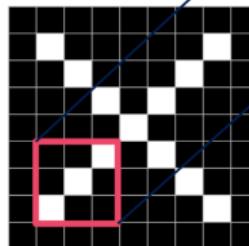


Filters Match Pieces of the Image

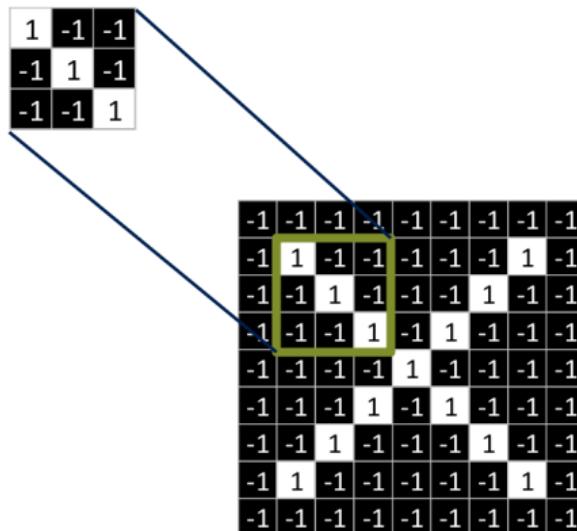
$$\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$

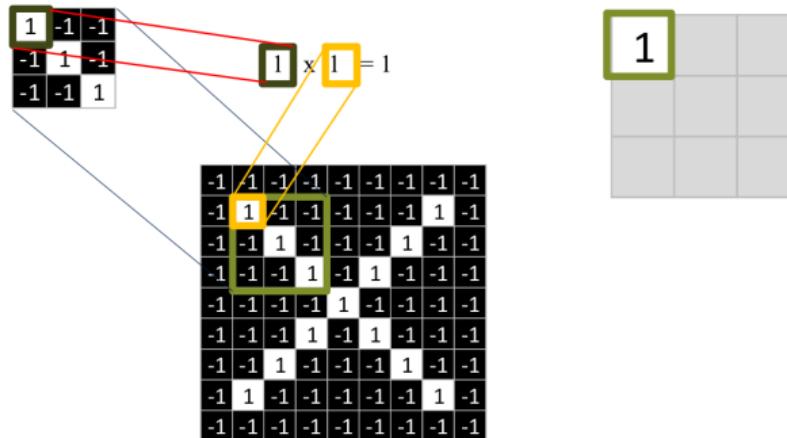
$$\begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix}$$



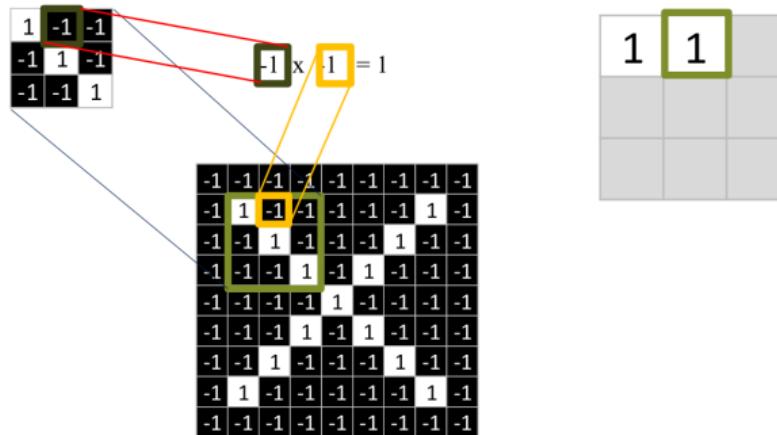
Filtering: The Math Behind the Match



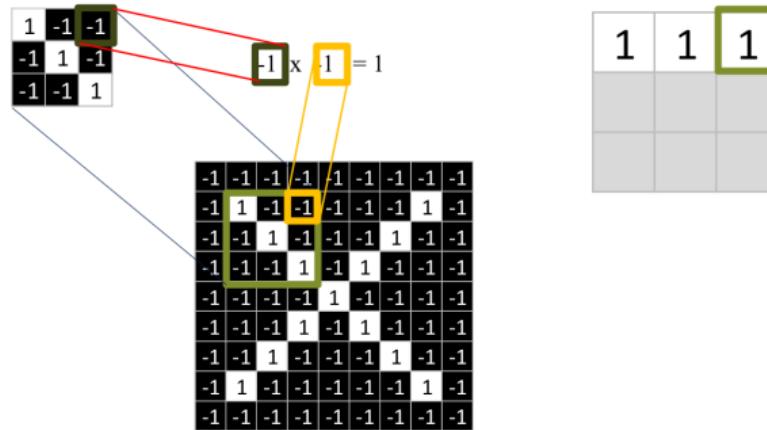
Filtering: The Math Behind the Match



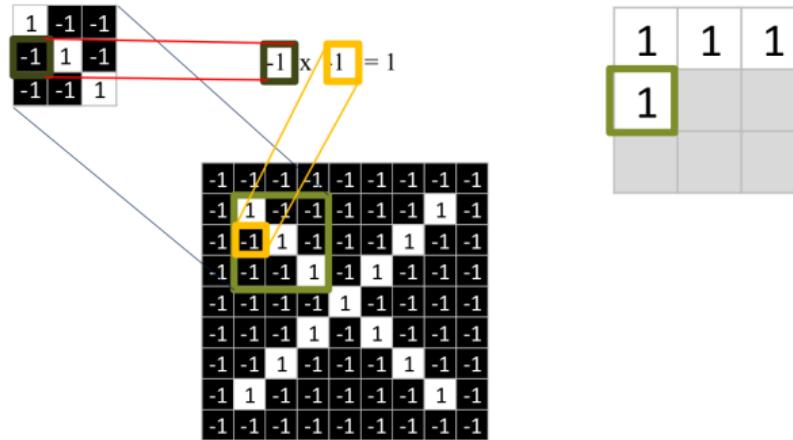
Filtering: The Math Behind the Match



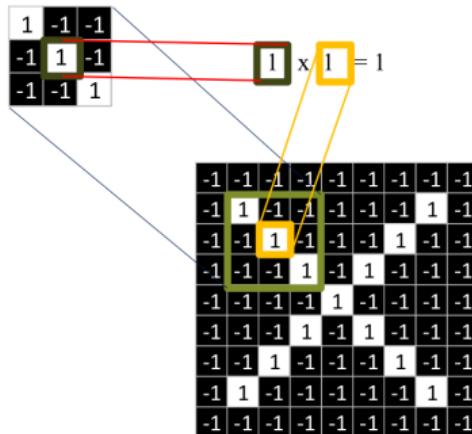
Filtering: The Math Behind the Match



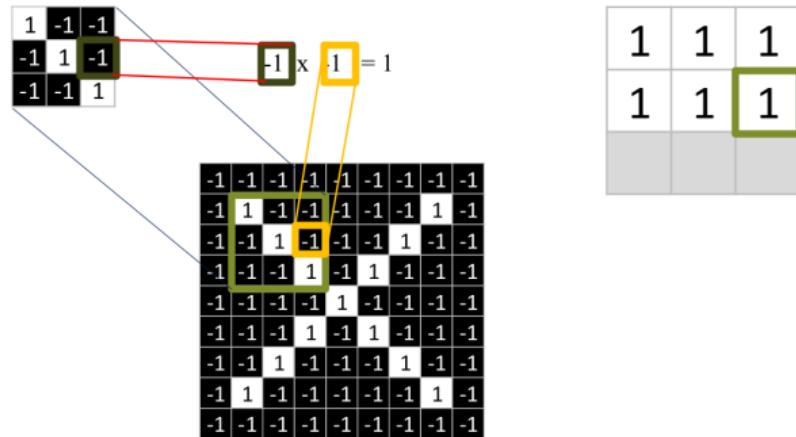
Filtering: The Math Behind the Match



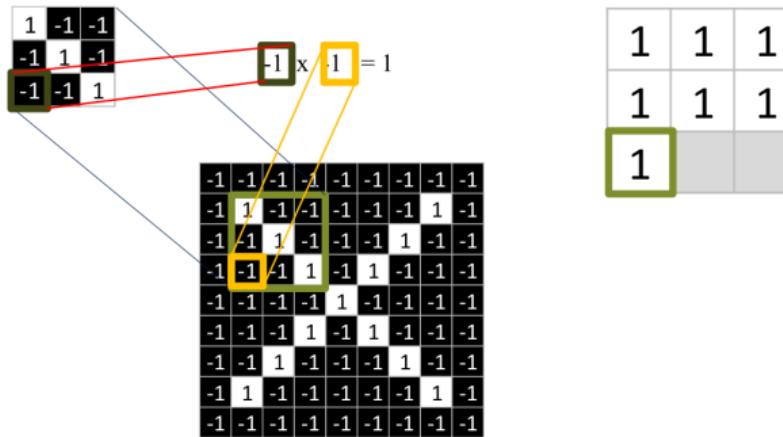
Filtering: The Math Behind the Match



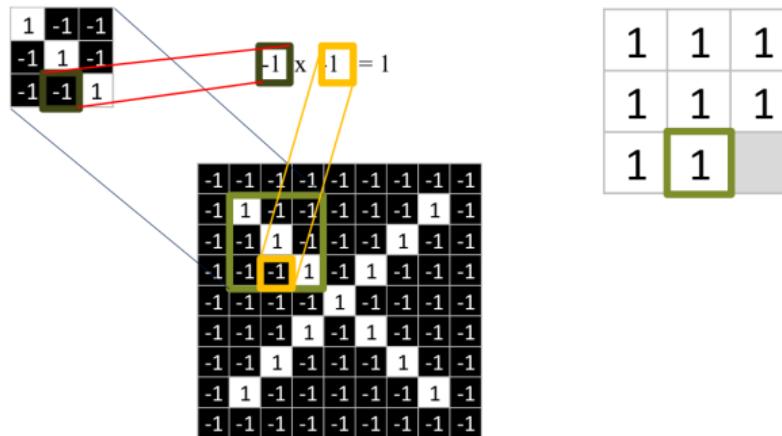
Filtering: The Math Behind the Match



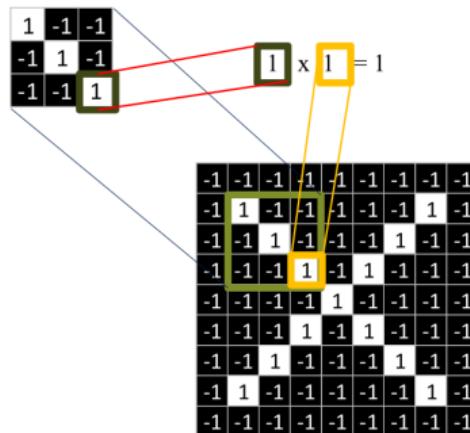
Filtering: The Math Behind the Match



Filtering: The Math Behind the Match

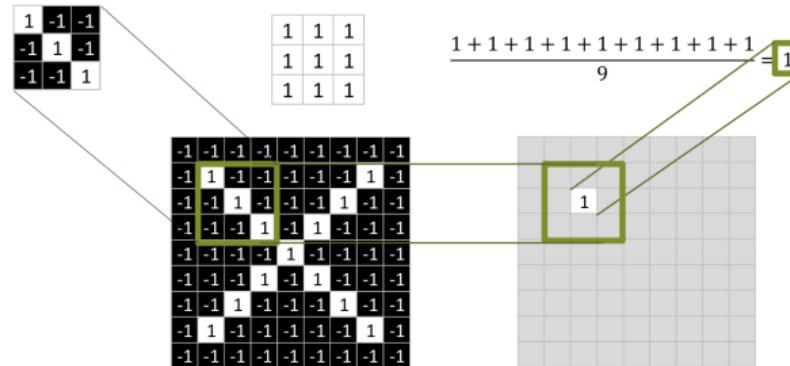


Filtering: The Math Behind the Match

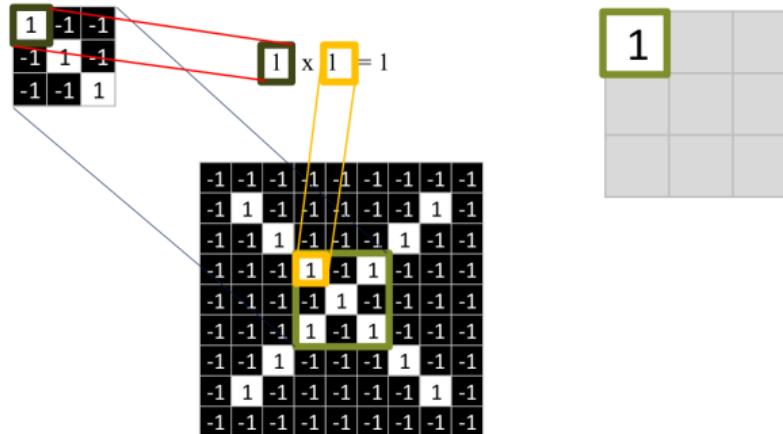


1	1	1
1	1	1
1	1	1

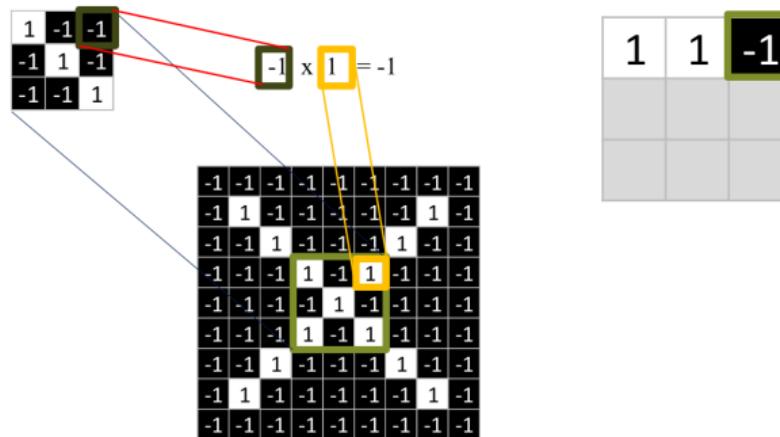
Filtering: The Math Behind the Match



Filtering: The Math Behind the Match

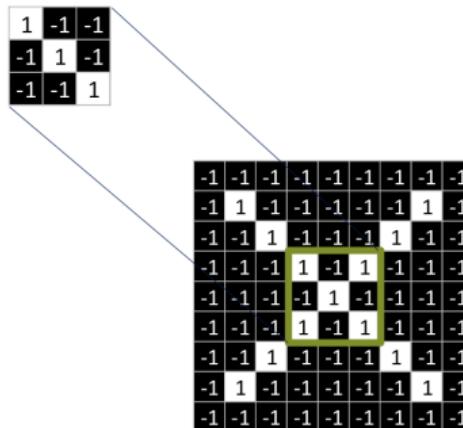


Filtering: The Math Behind the Match



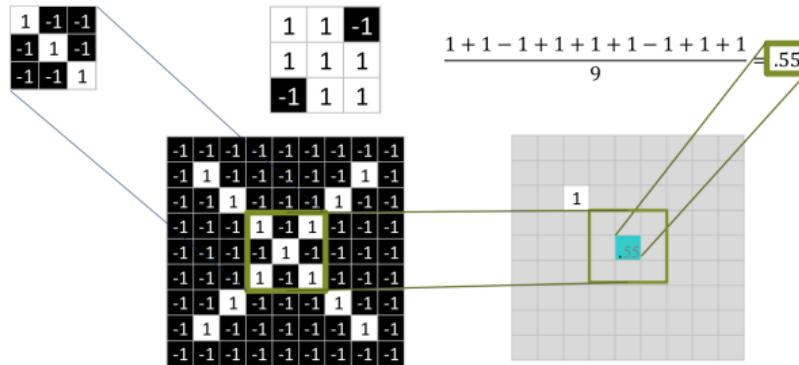
Filtering: The Math Behind the Match

$$\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}$$


$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & -1 \\ 1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix}$$

Filtering: The Math Behind the Match



Convolution: Trying Every Possible Match

$$\begin{array}{|c|c|c|c|c|c|c|c|c|} \hline -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ \hline -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ \hline -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ \hline -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ \hline -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & -1 \\ \hline -1 & 1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 \\ \hline -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 1 & -1 & -1 \\ \hline -1 & 1 & -1 \\ \hline -1 & -1 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 0.77 & -0.11 & 0.11 & 0.33 & 0.55 & -0.11 & 0.33 \\ \hline -0.11 & 1.00 & -0.11 & 0.33 & -0.11 & 0.11 & -0.11 \\ \hline 0.11 & -0.11 & 1.00 & -0.33 & 0.11 & -0.11 & 0.55 \\ \hline 0.33 & 0.33 & -0.33 & 0.55 & -0.33 & 0.33 & 0.33 \\ \hline 0.55 & -0.11 & 0.11 & -0.33 & 1.00 & -0.11 & 0.11 \\ \hline -0.11 & 0.11 & -0.11 & 0.33 & -0.11 & 1.00 & -0.11 \\ \hline 0.33 & -0.11 & 0.55 & 0.33 & 0.11 & -0.11 & 0.77 \\ \hline \end{array}$$

Three Filters Here, So Three Images Out

$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}$$

=

$$\begin{bmatrix} 0.77 & -0.11 & 0.11 & 0.33 & 0.55 & -0.11 & 0.33 \\ -0.11 & 1.00 & -0.11 & 0.33 & -0.11 & 0.11 & -0.11 \\ 0.11 & -0.11 & 1.00 & 0.33 & 0.33 & -0.11 & 0.33 \\ 0.33 & 0.11 & -0.33 & 0.55 & -0.33 & 0.33 & 0.33 \\ 0.55 & -0.11 & 0.11 & 0.33 & 1.00 & -0.11 & 0.11 \\ -0.11 & 0.11 & -0.11 & -0.33 & -0.11 & 1.00 & -0.11 \\ 0.33 & -0.11 & 0.55 & 0.33 & 0.33 & -0.11 & 0.77 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$

=

$$\begin{bmatrix} 0.33 & -0.55 & 0.11 & 0.11 & -0.55 & 0.33 \\ -0.55 & 0.55 & -0.55 & 0.33 & -0.55 & 0.55 & -0.55 \\ 0.11 & -0.55 & 0.55 & 0.77 & 0.55 & -0.55 & 0.11 \\ -0.11 & 0.33 & -0.77 & 1.00 & 0.77 & 0.33 & -0.33 \\ 0.11 & -0.55 & 0.55 & -0.77 & 0.55 & -0.55 & 0.11 \\ -0.55 & 0.55 & -0.55 & 0.11 & -0.55 & 0.55 & -0.55 \\ 0.33 & -0.55 & 0.11 & 0.11 & -0.55 & 0.33 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 & -1 & 1 & -1 \end{bmatrix}$$



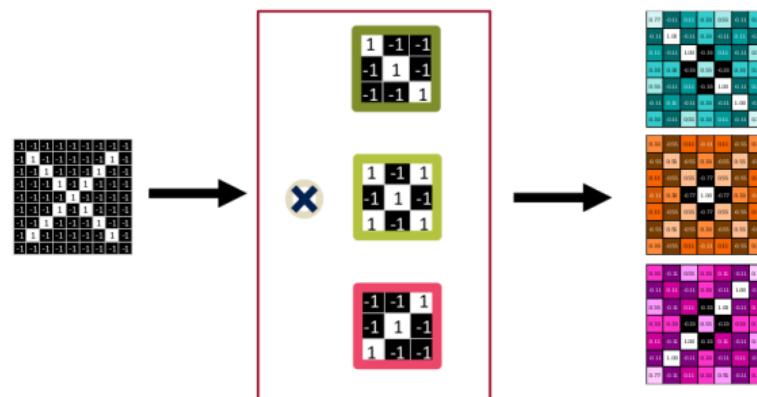
$$\begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix}$$

=

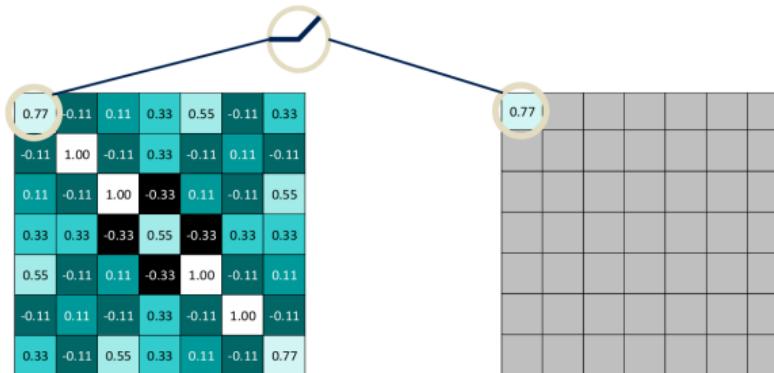
$$\begin{bmatrix} 0.33 & -0.11 & 0.55 & 0.33 & 0.11 & -0.11 & 0.77 \\ -0.11 & 0.11 & -0.11 & 0.33 & 0.11 & 1.00 & 0.11 \\ 0.55 & -0.11 & 0.11 & 0.33 & 1.00 & -0.11 & 0.11 \\ 0.33 & 0.11 & -0.33 & 0.55 & 0.33 & 0.33 & 0.33 \\ 0.11 & -0.11 & 1.00 & 0.33 & 0.11 & -0.11 & 0.55 \\ -0.11 & 1.00 & -0.11 & 0.33 & 0.11 & 0.11 & -0.11 \\ 0.77 & -0.11 & 0.11 & 0.33 & 0.55 & -0.11 & 0.33 \end{bmatrix}$$

Convolution Layer

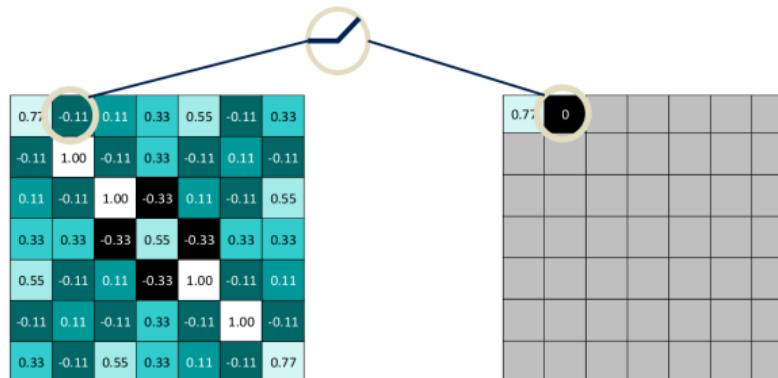
- One image becomes a stack of filtered images.



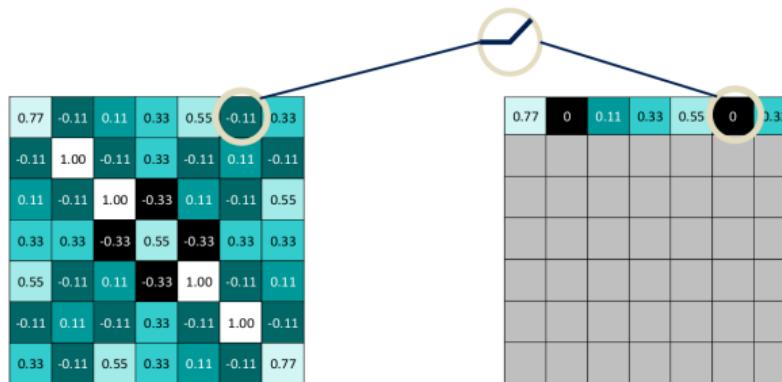
Rectified Linear Units (ReLUs)



Rectified Linear Units (ReLUs)



Rectified Linear Units (ReLUs)



Rectified Linear Units (ReLUs)

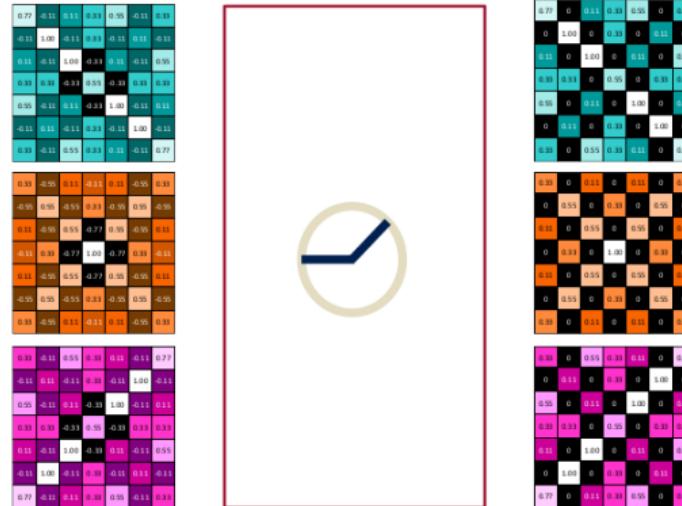
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77



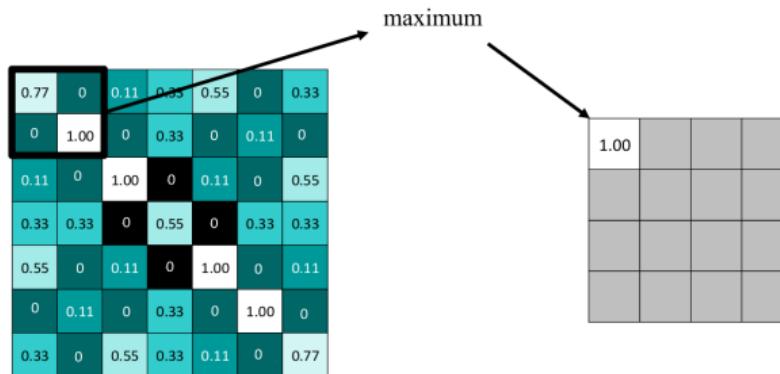
0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

ReLU Layer

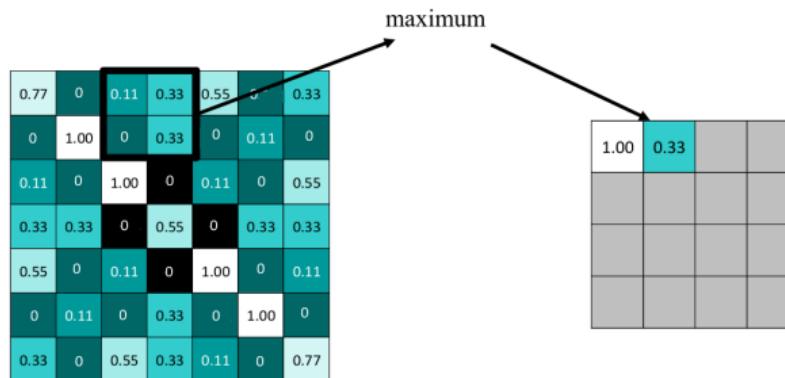
- ▶ A stack of images becomes a stack of images with no negative values.



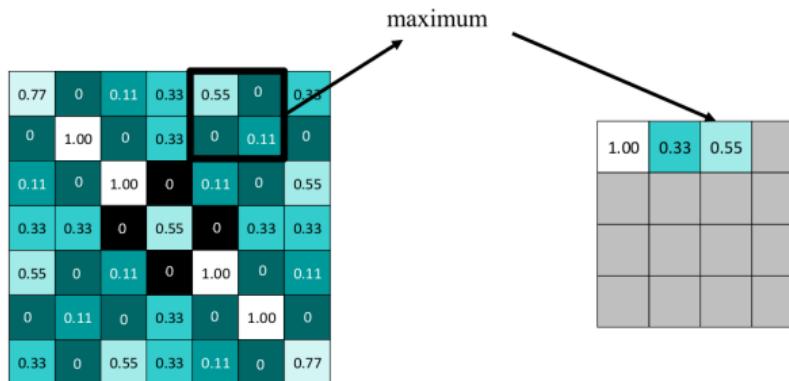
Pooling: Shrinking the Image Stack



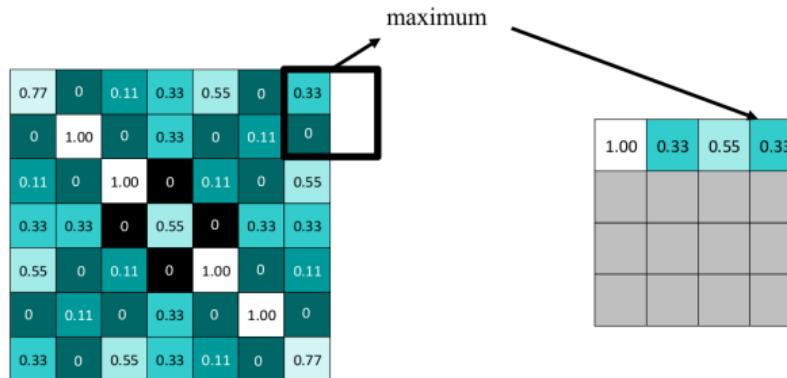
Pooling: Shrinking the Image Stack



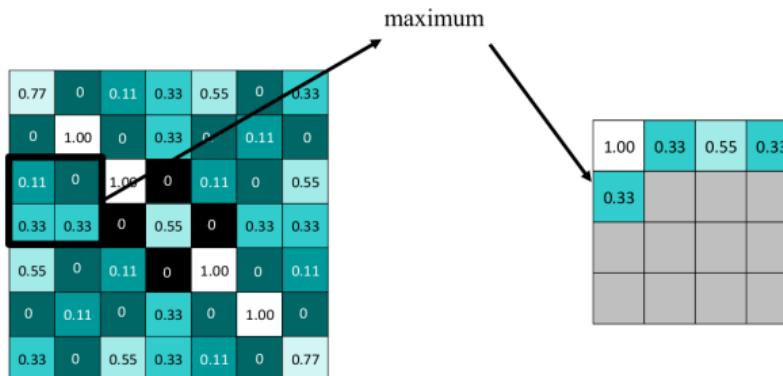
Pooling: Shrinking the Image Stack



Pooling: Shrinking the Image Stack



Pooling: Shrinking the Image Stack



Pooling: Shrinking the Image Stack

0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

max pooling

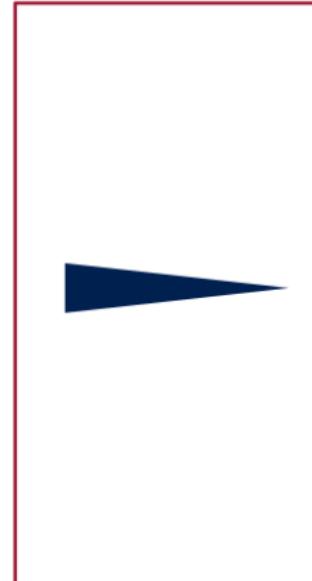
1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

Repeat For All the Filtered Images

0.77	0	0.33	0.33	0.33	0	0.33
0	1.00	0	0.33	0	0.33	0
0.33	0	1.00	0	0.33	0	0.33
0.33	0.33	0	0.33	0	0.33	0.33
0.33	0	0.33	0	1.00	0	0.33
0.33	0.33	0	0.33	0	1.00	0
0.33	0	0.33	0.33	0.33	0	0.77

0.33	0	0.33	0	0.33	0	0.33
0	0.55	0	0.33	0	0.55	0
0.33	0	0.33	0	0.33	0	0.33
0	0.33	0	1.00	0	0.33	0
0.33	0	0.33	0	0.33	0	0.33
0	0.55	0	0.33	0	0.55	0
0.33	0	0.33	0	0.33	0	0.33

0.33	0	0.33	0.33	0.33	0	0.77
0	0.33	0	0.33	0	1.00	0
0.33	0	0.33	0	1.00	0	0.33
0.33	0.33	0	0.33	0	0.33	0.33
0.33	0	1.00	0	0.33	0	0.33
0	1.00	0	0.33	0	0.33	0
0.33	0	0.33	0.33	0.33	0	0.33



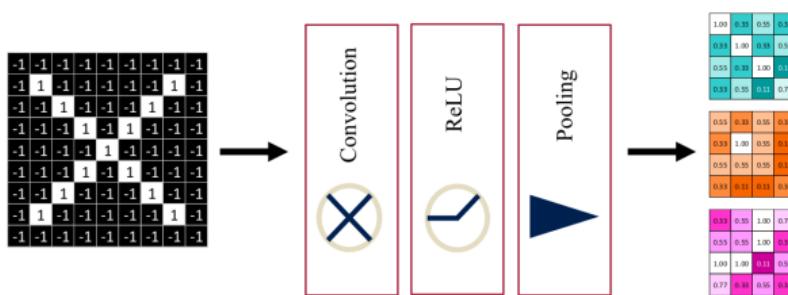
1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

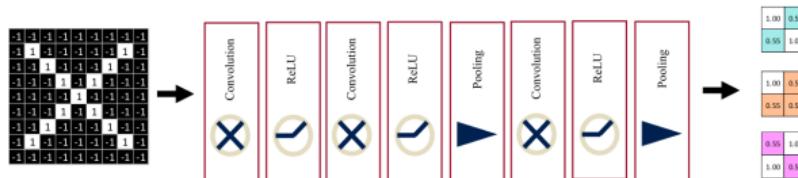
Layers Get Stacked

- ▶ The output of one becomes the input of the next.



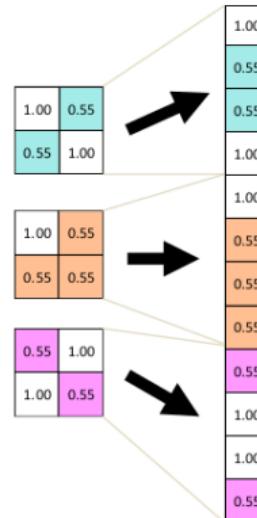


Deep Stacking



Fully Connected Layer

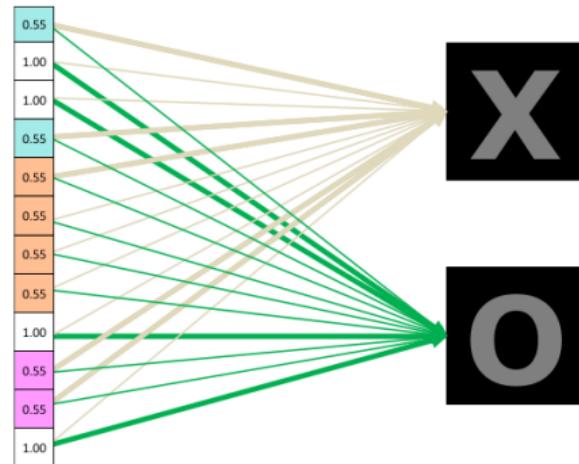
- ▶ Flattening the outputs before giving them to the **fully connected layer**.



Fully Connected Layer



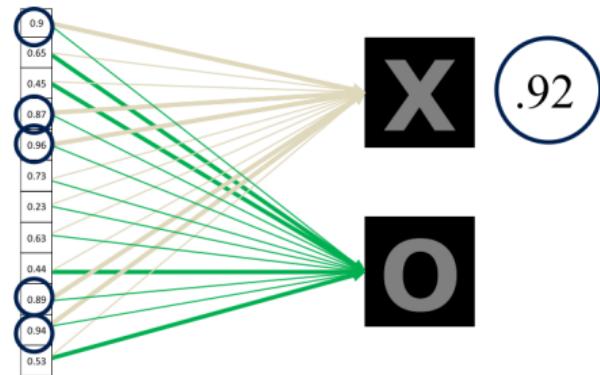
Fully Connected Layer



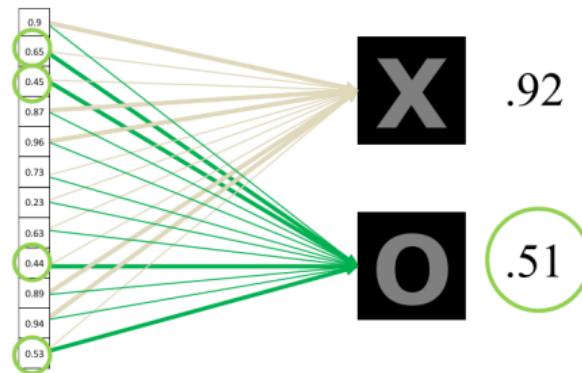
Fully Connected Layer



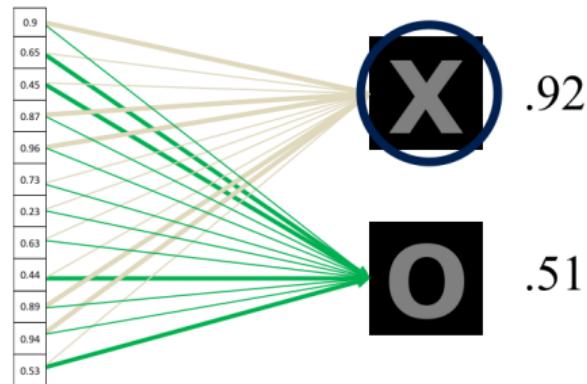
Fully Connected Layer



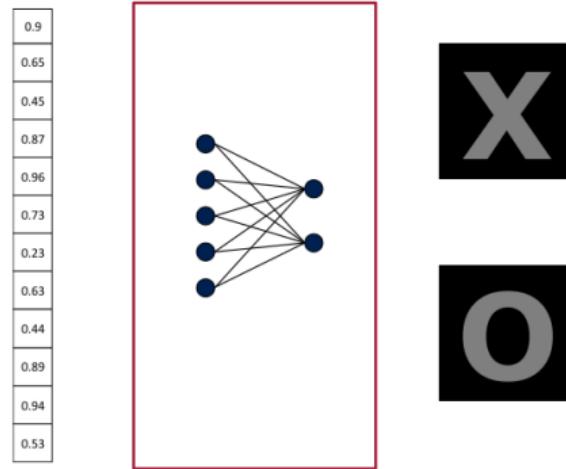
Fully Connected Layer



Fully Connected Layer

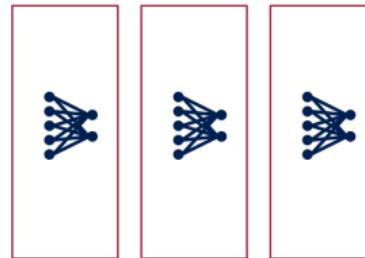


Fully Connected Layer

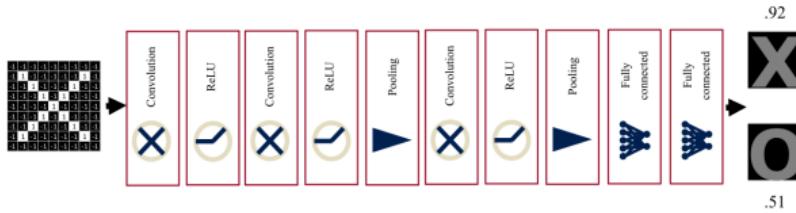


Fully Connected Layer

0.9
0.65
0.45
0.87
0.96
0.73
0.23
0.61
0.44
0.89
0.94
0.53



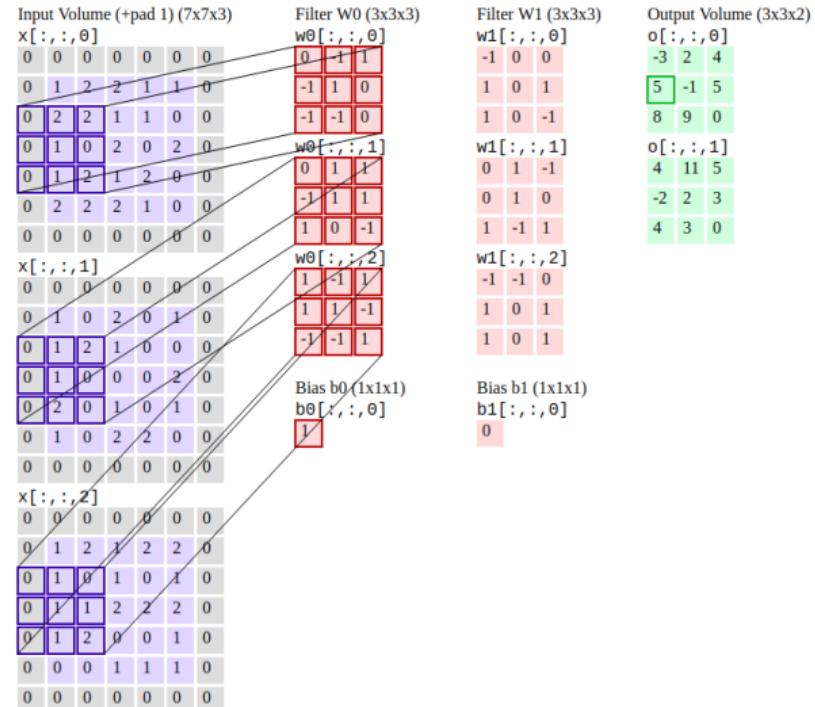
Putting It All Together





One more example

- ▶ A conv layer.
- ▶ Computes 2 feature maps.
- ▶ Filters: 3x3 with stride of 2.
- ▶ Input tensor shape: [7, 7, 3].
- ▶ Output tensor shape: [3, 3, 2].



[<http://cs231n.github.io/convolutional-networks>]



imgflip.com



CNN in TensorFlow



CNN in TensorFlow (1/7)

- ▶ A **CNN** for the **MNIST** dataset with the following network.



CNN in TensorFlow (1/7)

- ▶ A **CNN** for the **MNIST** dataset with the following network.
- ▶ Conv. layer 1: computes **32 feature maps** using a **5x5 filter** with ReLU activation.



CNN in TensorFlow (1/7)

- ▶ A CNN for the MNIST dataset with the following network.
- ▶ Conv. layer 1: computes 32 feature maps using a 5x5 filter with ReLU activation.
- ▶ Pooling layer 1: max pooling layer with a 2x2 filter and stride of 2.



CNN in TensorFlow (1/7)

- ▶ A CNN for the MNIST dataset with the following network.
- ▶ Conv. layer 1: computes 32 feature maps using a 5x5 filter with ReLU activation.
- ▶ Pooling layer 1: max pooling layer with a 2x2 filter and stride of 2.
- ▶ Conv. layer 2: computes 64 feature maps using a 5x5 filter.



CNN in TensorFlow (1/7)

- ▶ A CNN for the MNIST dataset with the following network.
- ▶ Conv. layer 1: computes 32 feature maps using a 5x5 filter with ReLU activation.
- ▶ Pooling layer 1: max pooling layer with a 2x2 filter and stride of 2.
- ▶ Conv. layer 2: computes 64 feature maps using a 5x5 filter.
- ▶ Pooling layer 2: max pooling layer with a 2x2 filter and stride of 2.



CNN in TensorFlow (1/7)

- ▶ A CNN for the MNIST dataset with the following network.
- ▶ Conv. layer 1: computes 32 feature maps using a 5x5 filter with ReLU activation.
- ▶ Pooling layer 1: max pooling layer with a 2x2 filter and stride of 2.
- ▶ Conv. layer 2: computes 64 feature maps using a 5x5 filter.
- ▶ Pooling layer 2: max pooling layer with a 2x2 filter and stride of 2.
- ▶ Dense layer: densely connected layer with 1024 neurons.



CNN in TensorFlow (1/7)

- ▶ A CNN for the MNIST dataset with the following network.
- ▶ Conv. layer 1: computes 32 feature maps using a 5x5 filter with ReLU activation.
- ▶ Pooling layer 1: max pooling layer with a 2x2 filter and stride of 2.
- ▶ Conv. layer 2: computes 64 feature maps using a 5x5 filter.
- ▶ Pooling layer 2: max pooling layer with a 2x2 filter and stride of 2.
- ▶ Dense layer: densely connected layer with 1024 neurons.
- ▶ Softmax layer



CNN in TensorFlow (2/7)

- ▶ Conv. layer 1: computes 32 feature maps using a 5x5 filter with ReLU activation.
- ▶ Padding same is added to preserve width and height.



CNN in TensorFlow (2/7)

- ▶ Conv. layer 1: computes 32 feature maps using a 5x5 filter with ReLU activation.
- ▶ Padding same is added to preserve width and height.
- ▶ Input tensor shape: [batch_size, 28, 28, 1]



CNN in TensorFlow (2/7)

- ▶ Conv. layer 1: computes 32 feature maps using a 5x5 filter with ReLU activation.
- ▶ Padding `same` is added to preserve width and height.
- ▶ Input tensor shape: `[batch_size, 28, 28, 1]`
- ▶ Output tensor shape: `[batch_size, 28, 28, 32]`

```
# MNIST images are 28x28 pixels, and have one color channel: [28, 28, 1]
```

```
tf.keras.layers.Conv2D(kernel_size=5, filters=32, activation='relu', padding='same',  
                      input_shape=[28, 28, 1])
```



CNN in TensorFlow (3/7)

- ▶ Pooling layer 1: max pooling layer with a 2x2 filter and stride of 2.



CNN in TensorFlow (3/7)

- ▶ Pooling layer 1: max pooling layer with a 2x2 filter and stride of 2.
- ▶ Input tensor shape: [batch_size, 28, 28, 32]



CNN in TensorFlow (3/7)

- ▶ Pooling layer 1: max pooling layer with a 2x2 filter and stride of 2.
- ▶ Input tensor shape: [batch_size, 28, 28, 32]
- ▶ Output tensor shape: [batch_size, 14, 14, 32]

```
tf.keras.layers.MaxPooling2D(pool_size=2, strides=2)
```



CNN in TensorFlow (4/7)

- ▶ Conv. layer 2: computes 64 feature maps using a 5x5 filter.
- ▶ Padding `same` is added to preserve width and height.



CNN in TensorFlow (4/7)

- ▶ Conv. layer 2: computes 64 feature maps using a 5x5 filter.
- ▶ Padding `same` is added to preserve width and height.
- ▶ Input tensor shape: `[batch_size, 14, 14, 32]`



CNN in TensorFlow (4/7)

- ▶ Conv. layer 2: computes 64 feature maps using a 5x5 filter.
- ▶ Padding `same` is added to preserve width and height.
- ▶ Input tensor shape: `[batch_size, 14, 14, 32]`
- ▶ Output tensor shape: `[batch_size, 14, 14, 64]`

```
tf.keras.layers.Conv2D(kernel_size=5, filters=64, activation='relu', padding='same')
```



CNN in TensorFlow (5/7)

- ▶ Pooling layer 2: max pooling layer with a 2x2 filter and stride of 2.



CNN in TensorFlow (5/7)

- ▶ Pooling layer 2: max pooling layer with a 2x2 filter and stride of 2.
- ▶ Input tensor shape: [batch_size, 14, 14, 64]



CNN in TensorFlow (5/7)

- ▶ Pooling layer 2: max pooling layer with a 2x2 filter and stride of 2.
- ▶ Input tensor shape: [batch_size, 14, 14, 64]
- ▶ Output tensor shape: [batch_size, 7, 7, 64]

```
tf.keras.layers.MaxPooling2D(pool_size=2, strides=2)
```



CNN in TensorFlow (6/7)

- ▶ **Flatten** tensor into a batch of vectors.



CNN in TensorFlow (6/7)

- ▶ **Flatten** tensor into a batch of vectors.
 - Input tensor shape: [batch_size, 7, 7, 64]



CNN in TensorFlow (6/7)

- ▶ **Flatten** tensor into a batch of vectors.
 - Input tensor shape: `[batch_size, 7, 7, 64]`
 - Output tensor shape: `[batch_size, 7 * 7 * 64]`

```
tf.keras.layers.Flatten()
```



CNN in TensorFlow (6/7)

- ▶ **Flatten** tensor into a batch of vectors.
 - Input tensor shape: `[batch_size, 7, 7, 64]`
 - Output tensor shape: `[batch_size, 7 * 7 * 64]`

```
tf.keras.layers.Flatten()
```

- ▶ **Dense layer:** densely connected layer with **1024 neurons**.



CNN in TensorFlow (6/7)

- ▶ **Flatten** tensor into a batch of vectors.
 - Input tensor shape: `[batch_size, 7, 7, 64]`
 - Output tensor shape: `[batch_size, 7 * 7 * 64]`

```
tf.keras.layers.Flatten()
```

- ▶ **Dense layer**: densely connected layer with **1024 neurons**.
 - Input tensor shape: `[batch_size, 7 * 7 * 64]`



CNN in TensorFlow (6/7)

- ▶ **Flatten** tensor into a batch of vectors.

- Input tensor shape: `[batch_size, 7, 7, 64]`
- Output tensor shape: `[batch_size, 7 * 7 * 64]`

```
tf.keras.layers.Flatten()
```

- ▶ **Dense layer**: densely connected layer with **1024 neurons**.

- Input tensor shape: `[batch_size, 7 * 7 * 64]`
- Output tensor shape: `[batch_size, 1024]`

```
tf.keras.layers.Dense(1024, activation='relu')
```



CNN in TensorFlow (6/7)

- ▶ **Flatten** tensor into a batch of vectors.
 - Input tensor shape: [batch_size, 7, 7, 64]
 - Output tensor shape: [batch_size, 7 * 7 * 64]

```
tf.keras.layers.Flatten()
```

- ▶ **Dense layer**: densely connected layer with 1024 neurons.
 - Input tensor shape: [batch_size, 7 * 7 * 64]
 - Output tensor shape: [batch_size, 1024]

```
tf.keras.layers.Dense(1024, activation='relu')
```

- ▶ **Softmax layer**: softmax layer with 10 neurons.



CNN in TensorFlow (6/7)

- ▶ **Flatten** tensor into a batch of vectors.
 - Input tensor shape: `[batch_size, 7, 7, 64]`
 - Output tensor shape: `[batch_size, 7 * 7 * 64]`

```
tf.keras.layers.Flatten()
```

- ▶ **Dense layer**: densely connected layer with **1024 neurons**.
 - Input tensor shape: `[batch_size, 7 * 7 * 64]`
 - Output tensor shape: `[batch_size, 1024]`

```
tf.keras.layers.Dense(1024, activation='relu')
```

- ▶ **Softmax layer**: softmax layer with **10 neurons**.
 - Input tensor shape: `[batch_size, 1024]`



CNN in TensorFlow (6/7)

- ▶ **Flatten** tensor into a batch of vectors.

- Input tensor shape: `[batch_size, 7, 7, 64]`
- Output tensor shape: `[batch_size, 7 * 7 * 64]`

```
tf.keras.layers.Flatten()
```

- ▶ **Dense layer**: densely connected layer with 1024 neurons.

- Input tensor shape: `[batch_size, 7 * 7 * 64]`
- Output tensor shape: `[batch_size, 1024]`

```
tf.keras.layers.Dense(1024, activation='relu')
```

- ▶ **Softmax layer**: softmax layer with 10 neurons.

- Input tensor shape: `[batch_size, 1024]`
- Output tensor shape: `[batch_size, 10]`

```
tf.keras.layers.Dense(10, activation='softmax')
```



CNN in TensorFlow (7/7)

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(kernel_size=5, filters=32, activation='relu', padding='same',
                          input_shape=[28, 28, 1]),
    tf.keras.layers.MaxPooling2D(pool_size=2, strides=2),
    tf.keras.layers.Conv2D(kernel_size=5, filters=64, activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D(pool_size=2, strides=2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```





Training CNNs

Training CNN (1/4)

- ▶ Let's see how to use **backpropagation** on a **single convolutional layer**.

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}



h_{11}	
h_{21}	h_{22}

Training CNN (1/4)

- ▶ Let's see how to use **backpropagation** on a **single convolutional layer**.
- ▶ Assume we have an input X of size **3×3** and a **single filter W** of size **2×2** .

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}



h_{11}	
h_{21}	h_{22}

Training CNN (1/4)

- ▶ Let's see how to use **backpropagation** on a **single convolutional layer**.
- ▶ Assume we have an input X of size **3×3** and a **single filter W** of size **2×2** .
- ▶ **No padding** and **stride = 1**.

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

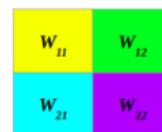


h_{11}	
h_{21}	h_{22}

Training CNN (1/4)

- ▶ Let's see how to use **backpropagation** on a **single convolutional layer**.
- ▶ Assume we have an input X of size **3×3** and a **single filter W** of size **2×2** .
- ▶ **No padding** and **stride = 1**.
- ▶ It generates an **output H** of size **2×2** .

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}



h_{11}	
h_{21}	h_{22}



Training CNN (2/4)

- ▶ Forward pass

Training CNN (2/4)

► Forward pass

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}



h_{11}	
h_{21}	h_{22}

$$h_{11} = W_{11}X_{11} + W_{12}X_{12} + W_{21}X_{21} + W_{22}X_{22}$$

Training CNN (2/4)

► Forward pass

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}



h_{11}	
h_{21}	h_{22}

$$h_{11} = W_{11}X_{11} + W_{12}X_{12} + W_{21}X_{21} + W_{22}X_{22}$$

$$h_{12} = W_{11}X_{12} + W_{12}X_{13} + W_{21}X_{22} + W_{22}X_{23}$$

Training CNN (2/4)

► Forward pass

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}



h_{11}	
	h_{12}
h_{21}	h_{22}

$$h_{11} = W_{11}X_{11} + W_{12}X_{12} + W_{21}X_{21} + W_{22}X_{22}$$

$$h_{12} = W_{11}X_{12} + W_{12}X_{13} + W_{21}X_{22} + W_{22}X_{23}$$

$$h_{21} = W_{11}X_{21} + W_{12}X_{22} + W_{21}X_{31} + W_{22}X_{32}$$

Training CNN (2/4)

► Forward pass



$$h_{11} = W_{11}X_{11} + W_{12}X_{12} + W_{21}X_{21} + W_{22}X_{22}$$

$$h_{12} = W_{11}X_{12} + W_{12}X_{13} + W_{21}X_{22} + W_{22}X_{23}$$

$$h_{21} = W_{11}X_{21} + W_{12}X_{22} + W_{21}X_{31} + W_{22}X_{32}$$

$$h_{22} = W_{11}X_{22} + W_{12}X_{23} + W_{21}X_{32} + W_{22}X_{33}$$

Training CNN (3/4)

- ▶ Backward pass
- ▶ E is the error: $E = E_{h_{11}} + E_{h_{12}} + E_{h_{21}} + E_{h_{22}}$

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

W_{11}	W_{12}
W_{21}	W_{22}

h_{11}	h_{12}
h_{21}	h_{22}

Training CNN (3/4)

- ▶ Backward pass
- ▶ E is the error: $E = E_{h_{11}} + E_{h_{12}} + E_{h_{21}} + E_{h_{22}}$

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

W_{11}	W_{12}
W_{21}	W_{22}

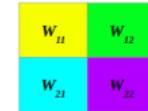
h_{11}	h_{12}
h_{21}	h_{22}

$$\frac{\partial E}{\partial W_{11}} = \frac{\partial E_{h_{11}}}{\partial h_{11}} \frac{\partial h_{11}}{\partial W_{11}} + \frac{\partial E_{h_{12}}}{\partial h_{12}} \frac{\partial h_{12}}{\partial W_{11}} + \frac{\partial E_{h_{21}}}{\partial h_{21}} \frac{\partial h_{21}}{\partial W_{11}} + \frac{\partial E_{h_{22}}}{\partial h_{22}} \frac{\partial h_{22}}{\partial W_{11}}$$

Training CNN (3/4)

- ▶ Backward pass
- ▶ E is the error: $E = E_{h_{11}} + E_{h_{12}} + E_{h_{21}} + E_{h_{22}}$

X_u	X_{i_2}	X_{i_3}
X_{z_1}	X_{z_2}	X_{z_3}
X_u	X_{i_2}	X_{i_3}



h_{ii}	h_{i_2}
h_{z_1}	h_{z_2}

$$\frac{\partial E}{\partial W_{11}} = \frac{\partial E_{h_{11}}}{\partial h_{11}} \frac{\partial h_{11}}{\partial W_{11}} + \frac{\partial E_{h_{12}}}{\partial h_{12}} \frac{\partial h_{12}}{\partial W_{11}} + \frac{\partial E_{h_{21}}}{\partial h_{21}} \frac{\partial h_{21}}{\partial W_{11}} + \frac{\partial E_{h_{22}}}{\partial h_{22}} \frac{\partial h_{22}}{\partial W_{11}}$$

$$\frac{\partial E}{\partial W_{12}} = \frac{\partial E_{h_{11}}}{\partial h_{11}} \frac{\partial h_{11}}{\partial W_{12}} + \frac{\partial E_{h_{12}}}{\partial h_{12}} \frac{\partial h_{12}}{\partial W_{12}} + \frac{\partial E_{h_{21}}}{\partial h_{21}} \frac{\partial h_{21}}{\partial W_{12}} + \frac{\partial E_{h_{22}}}{\partial h_{22}} \frac{\partial h_{22}}{\partial W_{12}}$$

Training CNN (3/4)

- ▶ Backward pass
- ▶ E is the error: $E = E_{h_{11}} + E_{h_{12}} + E_{h_{21}} + E_{h_{22}}$

X_u	X_{i_2}	X_{i_3}
X_{z_1}	X_{z_2}	X_{z_3}
X_u	X_{i_2}	X_{i_3}

w_{u_1}	w_{i_2}
w_{z_1}	w_{z_2}

h_{u_1}	h_{i_2}
h_{z_1}	h_{z_2}

$$\frac{\partial E}{\partial W_{11}} = \frac{\partial E_{h_{11}}}{\partial h_{11}} \frac{\partial h_{11}}{\partial W_{11}} + \frac{\partial E_{h_{12}}}{\partial h_{12}} \frac{\partial h_{12}}{\partial W_{11}} + \frac{\partial E_{h_{21}}}{\partial h_{21}} \frac{\partial h_{21}}{\partial W_{11}} + \frac{\partial E_{h_{22}}}{\partial h_{22}} \frac{\partial h_{22}}{\partial W_{11}}$$

$$\frac{\partial E}{\partial W_{12}} = \frac{\partial E_{h_{11}}}{\partial h_{11}} \frac{\partial h_{11}}{\partial W_{12}} + \frac{\partial E_{h_{12}}}{\partial h_{12}} \frac{\partial h_{12}}{\partial W_{12}} + \frac{\partial E_{h_{21}}}{\partial h_{21}} \frac{\partial h_{21}}{\partial W_{12}} + \frac{\partial E_{h_{22}}}{\partial h_{22}} \frac{\partial h_{22}}{\partial W_{12}}$$

$$\frac{\partial E}{\partial W_{21}} = \frac{\partial E_{h_{11}}}{\partial h_{11}} \frac{\partial h_{11}}{\partial W_{21}} + \frac{\partial E_{h_{12}}}{\partial h_{12}} \frac{\partial h_{12}}{\partial W_{21}} + \frac{\partial E_{h_{21}}}{\partial h_{21}} \frac{\partial h_{21}}{\partial W_{21}} + \frac{\partial E_{h_{22}}}{\partial h_{22}} \frac{\partial h_{22}}{\partial W_{21}}$$

Training CNN (3/4)

- ▶ Backward pass
- ▶ E is the error: $E = E_{h_{11}} + E_{h_{12}} + E_{h_{21}} + E_{h_{22}}$

X_u	X_{i_2}	X_{i_3}
X_{z_1}	X_{z_2}	X_{z_3}
X_u	X_{i_2}	X_{i_3}

w_{u_1}	w_{i_2}
w_{z_1}	w_{z_2}

h_{u_1}	h_{i_2}
h_{z_1}	h_{z_2}

$$\frac{\partial E}{\partial W_{11}} = \frac{\partial E_{h_{11}}}{\partial h_{11}} \frac{\partial h_{11}}{\partial W_{11}} + \frac{\partial E_{h_{12}}}{\partial h_{12}} \frac{\partial h_{12}}{\partial W_{11}} + \frac{\partial E_{h_{21}}}{\partial h_{21}} \frac{\partial h_{21}}{\partial W_{11}} + \frac{\partial E_{h_{22}}}{\partial h_{22}} \frac{\partial h_{22}}{\partial W_{11}}$$

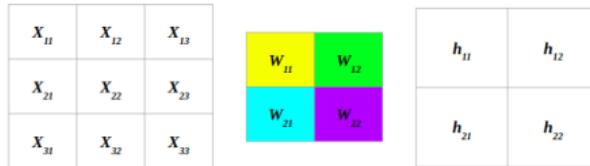
$$\frac{\partial E}{\partial W_{12}} = \frac{\partial E_{h_{11}}}{\partial h_{11}} \frac{\partial h_{11}}{\partial W_{12}} + \frac{\partial E_{h_{12}}}{\partial h_{12}} \frac{\partial h_{12}}{\partial W_{12}} + \frac{\partial E_{h_{21}}}{\partial h_{21}} \frac{\partial h_{21}}{\partial W_{12}} + \frac{\partial E_{h_{22}}}{\partial h_{22}} \frac{\partial h_{22}}{\partial W_{12}}$$

$$\frac{\partial E}{\partial W_{21}} = \frac{\partial E_{h_{11}}}{\partial h_{11}} \frac{\partial h_{11}}{\partial W_{21}} + \frac{\partial E_{h_{12}}}{\partial h_{12}} \frac{\partial h_{12}}{\partial W_{21}} + \frac{\partial E_{h_{21}}}{\partial h_{21}} \frac{\partial h_{21}}{\partial W_{21}} + \frac{\partial E_{h_{22}}}{\partial h_{22}} \frac{\partial h_{22}}{\partial W_{21}}$$

$$\frac{\partial E}{\partial W_{22}} = \frac{\partial E_{h_{11}}}{\partial h_{11}} \frac{\partial h_{11}}{\partial W_{22}} + \frac{\partial E_{h_{12}}}{\partial h_{12}} \frac{\partial h_{12}}{\partial W_{22}} + \frac{\partial E_{h_{21}}}{\partial h_{21}} \frac{\partial h_{21}}{\partial W_{22}} + \frac{\partial E_{h_{22}}}{\partial h_{22}} \frac{\partial h_{22}}{\partial W_{22}}$$

Training CNN (4/4)

- ▶ Update the weights W



$$W_{11}^{(\text{next})} = W_{11} - \eta \frac{\partial E}{\partial W_{11}}$$

$$W_{12}^{(\text{next})} = W_{12} - \eta \frac{\partial E}{\partial W_{12}}$$

$$W_{21}^{(\text{next})} = W_{21} - \eta \frac{\partial E}{\partial W_{21}}$$

$$W_{22}^{(\text{next})} = W_{22} - \eta \frac{\partial E}{\partial W_{22}}$$



Summary



Summary

- ▶ Receptive fields and filters
- ▶ Convolution operation
- ▶ Padding and strides
- ▶ Pooling layer
- ▶ Flattening, dropout, dense



Reference

- ▶ Tensorflow and Deep Learning without a PhD
<https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist>
- ▶ Ian Goodfellow et al., Deep Learning (Ch. 9)
- ▶ Aurélien Géron, Hands-On Machine Learning (Ch. 14)

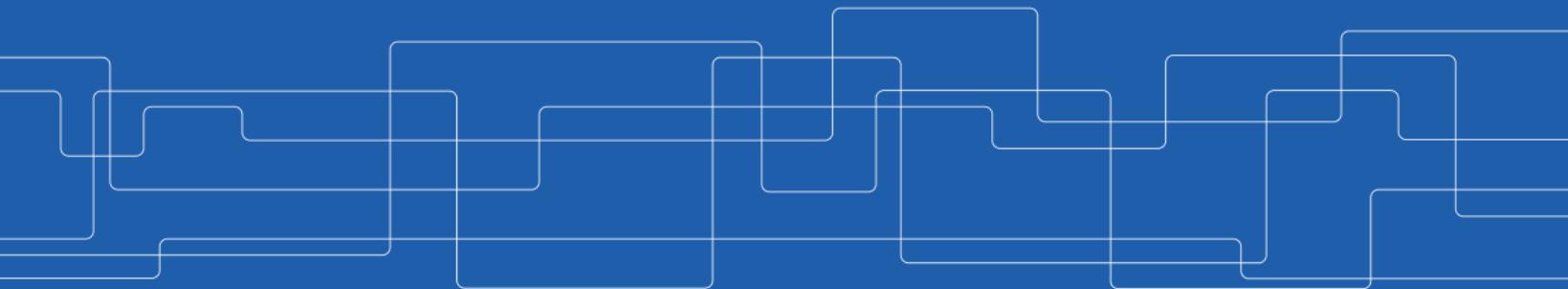


Questions?



Recurrent Neural Networks

Amir H. Payberah
payberah@kth.se
2021-12-01



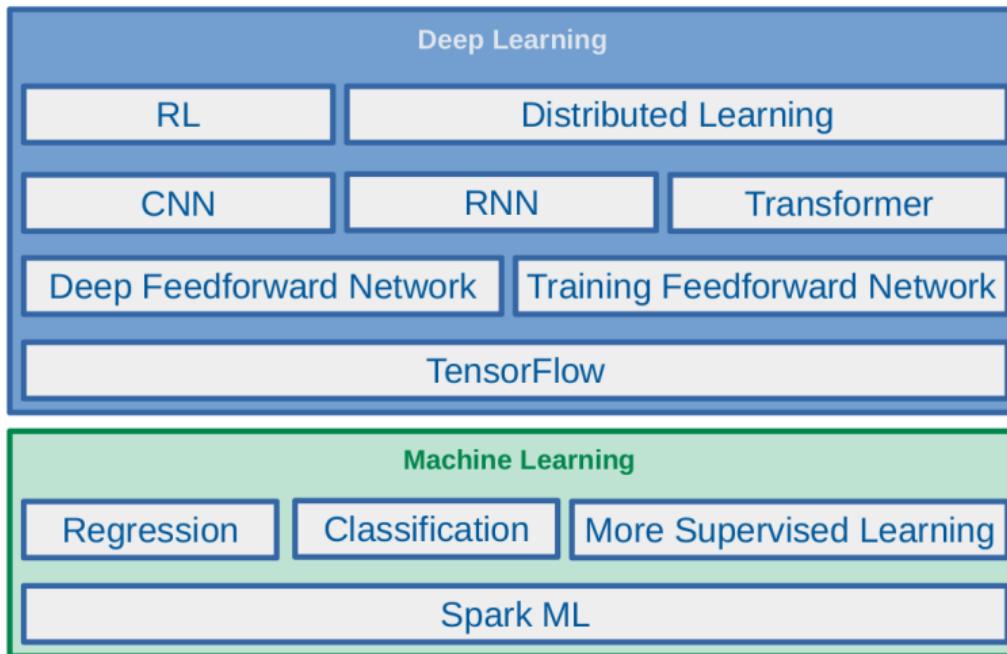


The Course Web Page

<https://id2223kth.github.io>
<https://tinyurl.com/6s5jy46a>

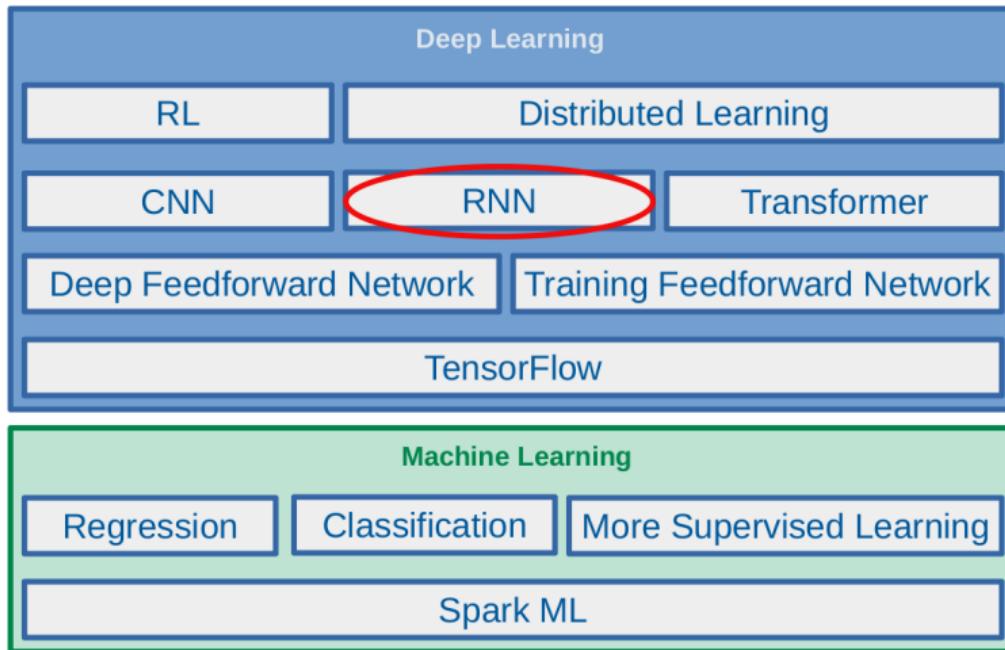


Where Are We?





Where Are We?





Let's Start With An Example



Google

the students opened their



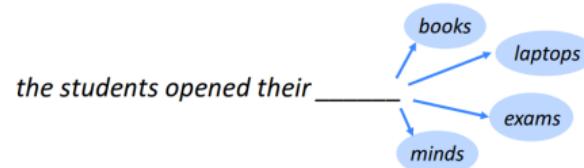
- their work
- their books
- their teachers
- their homework
- their lecturer
- their new lecturer

Feeling Lucky

venska

Language Modeling (1/2)

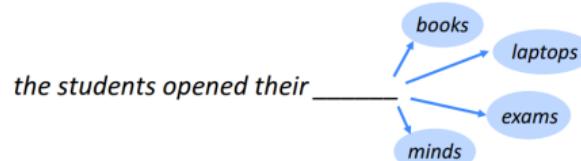
- ▶ Language modeling is the task of predicting what word comes next.



Language Modeling (2/2)

- More formally: given a sequence of words $x^{(1)}, x^{(2)}, \dots, x^{(t)}$, compute the probability distribution of the next word $x^{(t+1)}$:

$$p(x^{(t+1)} = w_j | x^{(t)}, \dots, x^{(1)})$$

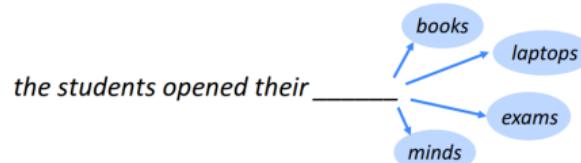


Language Modeling (2/2)

- More formally: given a sequence of words $x^{(1)}, x^{(2)}, \dots, x^{(t)}$, compute the probability distribution of the next word $x^{(t+1)}$:

$$p(x^{(t+1)} = w_j | x^{(t)}, \dots, x^{(1)})$$

- w_j is a word in vocabulary $V = \{w_1, \dots, w_v\}$.





n-gram Language Models

- ▶ the students opened their ___



n-gram Language Models

- ▶ the students opened their ___
- ▶ How to learn a Language Model?



n-gram Language Models

- ▶ the students opened their ___
- ▶ How to learn a Language Model?
- ▶ Learn a n-gram Language Model!



n-gram Language Models

- ▶ the students opened their ...
- ▶ How to learn a Language Model?
- ▶ Learn a n-gram Language Model!
- ▶ A n-gram is a chunk of n consecutive words.



n-gram Language Models

- ▶ the students opened their ...
- ▶ How to learn a Language Model?
- ▶ Learn a n-gram Language Model!
- ▶ A n-gram is a chunk of n consecutive words.
 - Unigrams: "the", "students", "opened", "their"



n-gram Language Models

- ▶ the students opened their ...
- ▶ How to learn a Language Model?
- ▶ Learn a n-gram Language Model!
- ▶ A n-gram is a chunk of n consecutive words.
 - Unigrams: "the", "students", "opened", "their"
 - Bigrams: "the students", "students opened", "opened their"



n-gram Language Models

- ▶ the students opened their ...
- ▶ How to learn a Language Model?
- ▶ Learn a n-gram Language Model!
- ▶ A n-gram is a chunk of n consecutive words.
 - Unigrams: "the", "students", "opened", "their"
 - Bigrams: "the students", "students opened", "opened their"
 - Trigrams: "the students opened", "students opened their"



n-gram Language Models

- ▶ the students opened their ...
- ▶ How to learn a Language Model?
- ▶ Learn a n-gram Language Model!
- ▶ A n-gram is a chunk of n consecutive words.
 - Unigrams: "the", "students", "opened", "their"
 - Bigrams: "the students", "students opened", "opened their"
 - Trigrams: "the students opened", "students opened their"
 - 4-grams: "the students opened their"



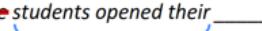
n-gram Language Models

- ▶ the students opened their ...
- ▶ How to learn a Language Model?
- ▶ Learn a n-gram Language Model!
- ▶ A n-gram is a chunk of n consecutive words.
 - Unigrams: "the", "students", "opened", "their"
 - Bigrams: "the students", "students opened", "opened their"
 - Trigrams: "the students opened", "students opened their"
 - 4-grams: "the students opened their"
- ▶ Collect statistics about how frequent different n-grams are, and use these to predict next word.



n-gram Language Models - Example

- ▶ Suppose we are learning a **4-gram** Language Model.
 - $x^{(t+1)}$ depends only on the preceding 3 words $\{x^{(t)}, x^{(t-1)}, x^{(t-2)}\}$.

~~as the proctor started the clock, the students opened their~~ _____
discard  condition on this



n-gram Language Models - Example

- ▶ Suppose we are learning a **4-gram** Language Model.
 - $x^{(t+1)}$ depends only on the preceding 3 words $\{x^{(t)}, x^{(t-1)}, x^{(t-2)}\}$.

~~as the proctor started the clock, the students opened their~~ _____
discard _____
condition on this

$$p(w_j | \text{students opened their}) = \frac{\text{students opened their } w_j}{\text{students opened their}}$$



n-gram Language Models - Example

- ▶ Suppose we are learning a **4-gram** Language Model.
 - $x^{(t+1)}$ depends only on the preceding 3 words $\{x^{(t)}, x^{(t-1)}, x^{(t-2)}\}$.

~~as the proctor started the clock, the students opened their~~ _____
discard condition on this

$$p(w_j | \text{students opened their}) = \frac{\text{students opened their } w_j}{\text{students opened their}}$$

- ▶ In the corpus:
 - "students opened their" occurred 1000 times



n-gram Language Models - Example

- ▶ Suppose we are learning a 4-gram Language Model.
 - $x^{(t+1)}$ depends only on the preceding 3 words $\{x^{(t)}, x^{(t-1)}, x^{(t-2)}\}$.

~~as the proctor started the clock, the students opened their~~
~~discard~~ the students opened their
condition on this

$$p(w_j | \text{students opened their}) = \frac{\text{students opened their } w_j}{\text{students opened their}}$$

- ▶ In the corpus:
 - "students opened their" occurred 1000 times
 - "students opened their books" occurred 400 times:
 $p(\text{books}|\text{students opened their}) = 0.4$



n-gram Language Models - Example

- ▶ Suppose we are learning a **4-gram Language Model**.
 - $x^{(t+1)}$ depends only on the preceding 3 words $\{x^{(t)}, x^{(t-1)}, x^{(t-2)}\}$.

~~as the proctor started the clock, the students opened their~~

discard

condition on this

$$p(w_j | \text{students opened their}) = \frac{\text{students opened their } w_j}{\text{students opened their}}$$

- ▶ In the corpus:
 - "students opened their" occurred 1000 times
 - "students opened their books" occurred 400 times:
 $p(\text{books}|\text{students opened their}) = 0.4$
 - "students opened their exams" occurred 100 times:
 $p(\text{exams}|\text{students opened their}) = 0.1$



Problems with n-gram Language Models - Sparsity

$$p(w_j | \text{students opened their}) = \frac{\text{students opened their } w_j}{\text{students opened their}}$$



Problems with n-gram Language Models - Sparsity

$$p(w_j | \text{students opened their}) = \frac{\text{students opened their } w_j}{\text{students opened their}}$$

- ▶ What if "students opened their w_j " never occurred in data? Then w_j has probability 0!



Problems with n-gram Language Models - Sparsity

$$p(w_j | \text{students opened their}) = \frac{\text{students opened their } w_j}{\text{students opened their}}$$

- ▶ What if "students opened their w_j " never occurred in data? Then w_j has probability 0!
- ▶ What if "students opened their" never occurred in data? Then we can't calculate probability for any w_j !



Problems with n-gram Language Models - Sparsity

$$p(w_j | \text{students opened their}) = \frac{\text{students opened their } w_j}{\text{students opened their}}$$

- ▶ What if "students opened their w_j " never occurred in data? Then w_j has probability 0!
- ▶ What if "students opened their" never occurred in data? Then we can't calculate probability for any w_j !
- ▶ Increasing n makes sparsity problems worse.
 - Typically we can't have n bigger than 5.



Problems with n-gram Language Models - Storage

$$p(w_j | \text{students opened their}) = \frac{\text{students opened their } w_j}{\text{students opened their}}$$



Problems with n-gram Language Models - Storage

$$p(w_j | \text{students opened their}) = \frac{\text{students opened their } w_j}{\text{students opened their}}$$

- ▶ For "students opened their w_j ", we need to store count for all possible 4-grams.
- ▶ The model size is in the order of $O(\exp(n))$.
- ▶ Increasing n makes model size huge.



Can We Build a Neural Language Model? (1/3)

► Recall the **Language Modeling** task:

- **Input:** sequence of words $x^{(1)}, x^{(2)}, \dots, x^{(t)}$
- **Output:** probability dist of the next word $p(x^{(t+1)} = w_j | x^{(t)}, \dots, x^{(1)})$

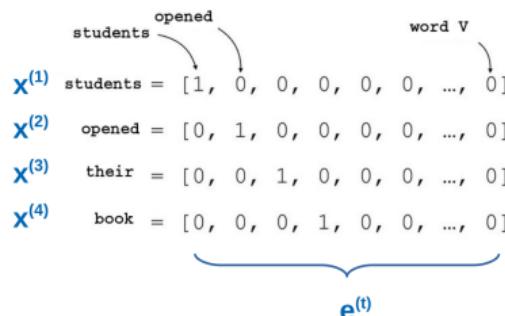
Can We Build a Neural Language Model? (1/3)

► Recall the **Language Modeling** task:

- **Input:** sequence of words $x^{(1)}, x^{(2)}, \dots, x^{(t)}$
- **Output:** probability dist of the next word $p(x^{(t+1)} = w_j | x^{(t)}, \dots, x^{(1)})$

► One-Hot encoding

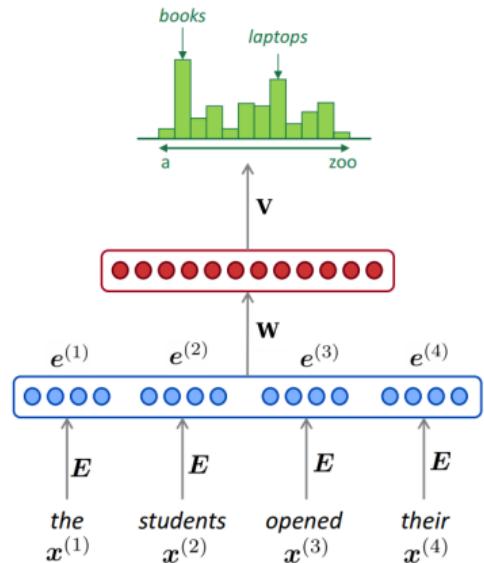
- Represent a **categorical variable** as a **binary vector**.
- All recodes are **zero**, except the index of the integer, which is **one**.
- Each embedded word $e^{(t)} = E^T x^{(t)}$ is a **one-hot vector** of size **vocabulary size**.



Can We Build a Neural Language Model? (2/3)

► A MLP model

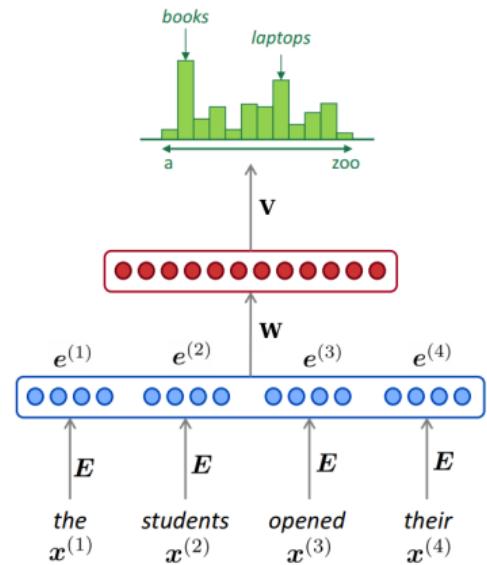
- Input: words $x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$
- Input layer: one-hot vectors $e^{(1)}, e^{(2)}, e^{(3)}, e^{(4)}$
- Hidden layer: $h = f(w^T e)$, f is an activation function.
- Output: $\hat{y} = \text{softmax}(v^T h)$



Can We Build a Neural Language Model? (3/3)

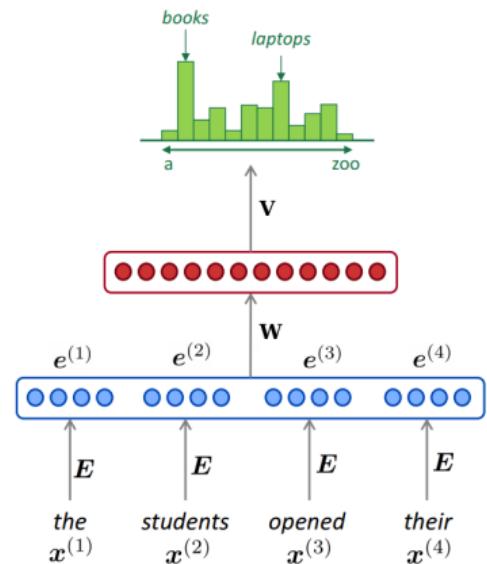
► Improvements over n-gram LM:

- No sparsity problem
- Model size is $O(n)$ not $O(\exp(n))$



Can We Build a Neural Language Model? (3/3)

- ▶ **Improvements** over n-gram LM:
 - No sparsity problem
 - Model size is $O(n)$ not $O(\exp(n))$
- ▶ Remaining problems:
 - It is **fixed 4** in our example, which is small
 - We need a neural architecture that can process any length input





Recurrent Neural Networks (RNN)



Recurrent Neural Networks (1/4)

- ▶ The idea behind Recurrent neural networks (RNN) is to make use of sequential data.



Recurrent Neural Networks (1/4)

- ▶ The idea behind Recurrent neural networks (RNN) is to make use of sequential data.
 - Until here, we assume that all inputs (and outputs) are independent of each other.



Recurrent Neural Networks (1/4)

- ▶ The idea behind Recurrent neural networks (RNN) is to make use of sequential data.
 - Until here, we assume that all inputs (and outputs) are independent of each other.
 - Independent input (output) is a bad idea for many tasks, e.g., predicting the next word in a sentence (it's better to know which words came before it).



Recurrent Neural Networks (1/4)

- ▶ The idea behind Recurrent neural networks (RNN) is to make use of sequential data.
 - Until here, we assume that all inputs (and outputs) are independent of each other.
 - Independent input (output) is a bad idea for many tasks, e.g., predicting the next word in a sentence (it's better to know which words came before it).
- ▶ They can analyze time series data and predict the future.

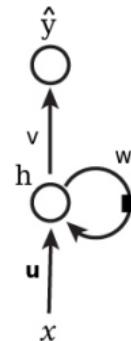


Recurrent Neural Networks (1/4)

- ▶ The idea behind Recurrent neural networks (RNN) is to make use of sequential data.
 - Until here, we assume that all inputs (and outputs) are independent of each other.
 - Independent input (output) is a bad idea for many tasks, e.g., predicting the next word in a sentence (it's better to know which words came before it).
- ▶ They can analyze time series data and predict the future.
- ▶ They can work on sequences of arbitrary lengths, rather than on fixed-sized inputs.

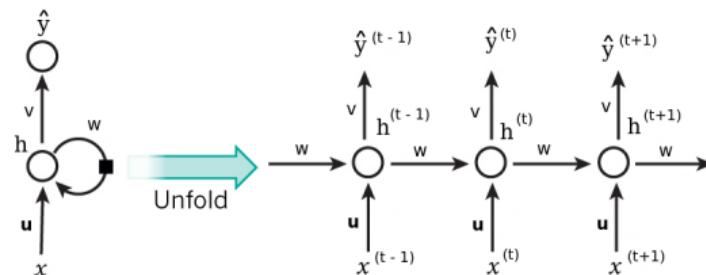
Recurrent Neural Networks (2/4)

- ▶ Neurons in an **RNN** have **connections pointing backward**.
- ▶ RNNs have **memory**, which captures **information** about what **has been calculated so far**.



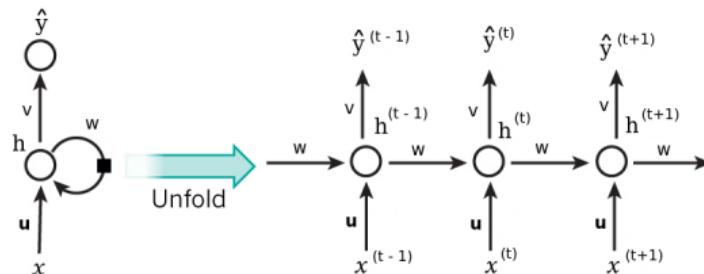
Recurrent Neural Networks (3/4)

- **Unfolding the network:** represent a network against the time axis.
 - We write out the network for the **complete sequence**.



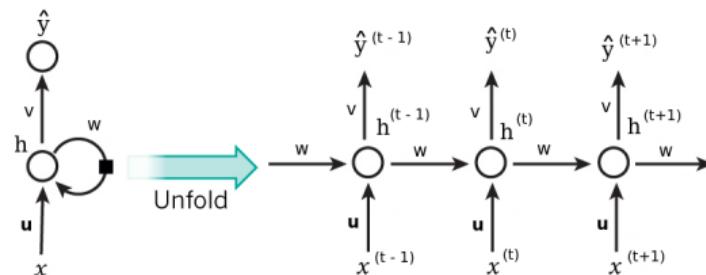
Recurrent Neural Networks (3/4)

- ▶ **Unfolding the network:** represent a network against the time axis.
 - We write out the network for the **complete sequence**.
- ▶ For example, if the sequence we care about is a **sentence of three words**, the network would be **unfolded** into a **3-layer** neural network.
 - One layer for each word.



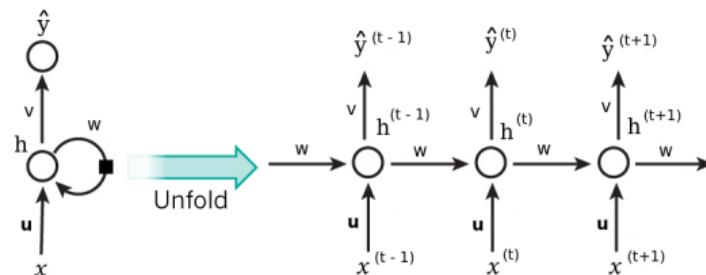
Recurrent Neural Networks (4/4)

- ▶ $h^{(t)} = f(u^T x^{(t)} + w h^{(t-1)})$, where f is an activation function, e.g., **tanh** or **ReLU**.



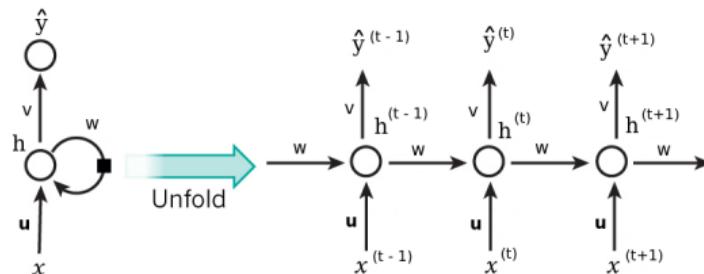
Recurrent Neural Networks (4/4)

- ▶ $h^{(t)} = f(u^T x^{(t)} + w h^{(t-1)})$, where f is an activation function, e.g., **tanh** or **ReLU**.
- ▶ $\hat{y}^{(t)} = g(vh^{(t)})$, where g can be the **softmax** function.



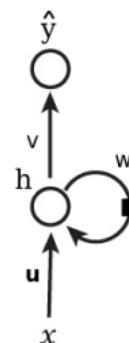
Recurrent Neural Networks (4/4)

- ▶ $h^{(t)} = f(u^T x^{(t)} + w h^{(t-1)})$, where f is an activation function, e.g., **tanh** or **ReLU**.
- ▶ $\hat{y}^{(t)} = g(v h^{(t)})$, where g can be the **softmax** function.
- ▶ $\text{cost}(y^{(t)}, \hat{y}^{(t)}) = \text{cross_entropy}(y^{(t)}, \hat{y}^{(t)}) = -\sum y^{(t)} \log \hat{y}^{(t)}$
- ▶ $y^{(t)}$ is the **correct** word at time step t , and $\hat{y}^{(t)}$ is the **prediction**.



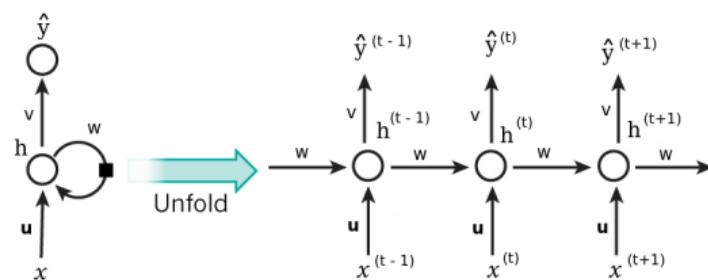
Recurrent Neurons - Weights (1/4)

- ▶ Each recurrent neuron has **three sets of weights**: **u**, **w**, and **v**.



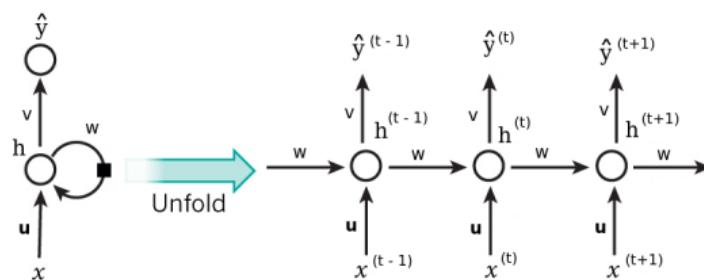
Recurrent Neurons - Weights (2/4)

- ▶ **u:** the weights for the inputs $x^{(t)}$.



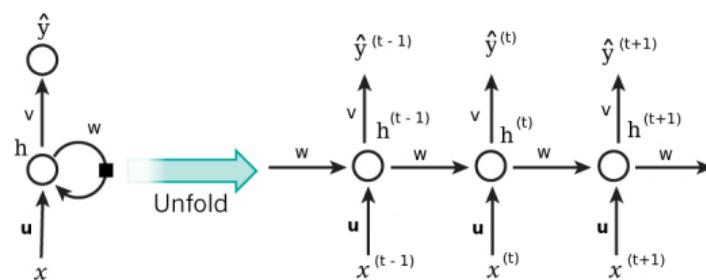
Recurrent Neurons - Weights (2/4)

- ▶ **u**: the **weights for the inputs** $x^{(t)}$.
- ▶ $x^{(t)}$: is the **input** at time step t.
- ▶ For example, $x^{(1)}$ could be a **one-hot vector** corresponding to the **first word of a sentence**.



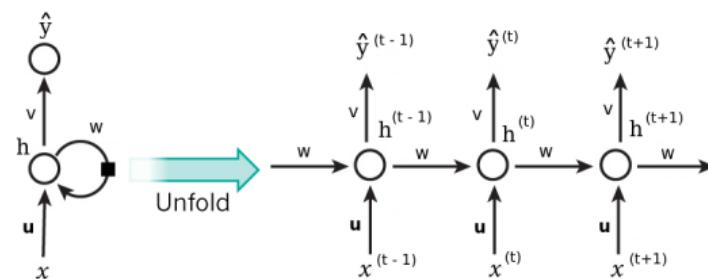
Recurrent Neurons - Weights (3/4)

- ▶ w : the weights for the hidden state of the previous time step $h^{(t-1)}$.



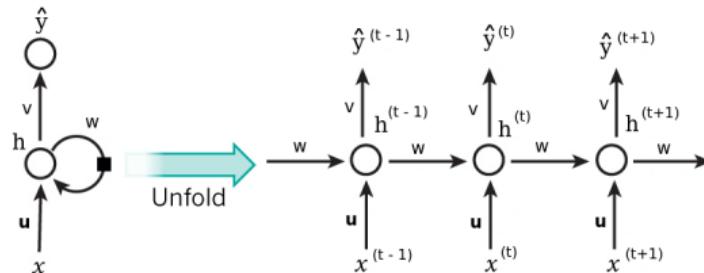
Recurrent Neurons - Weights (3/4)

- ▶ w : the **weights** for the **hidden state** of the **previous time step** $h^{(t-1)}$.
- ▶ $h^{(t)}$: is the **hidden state (memory)** at time step t .
 - $h^{(t)} = \tanh(u^T x^{(t)} + w h^{(t-1)})$
 - $h^{(0)}$ is the **initial hidden state**.



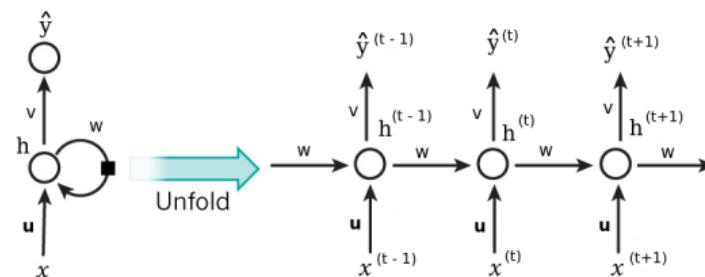
Recurrent Neurons - Weights (4/4)

- ▶ v : the weights for the hidden state of the current time step $h^{(t)}$.



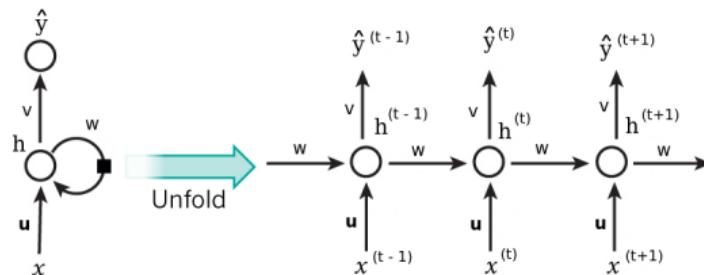
Recurrent Neurons - Weights (4/4)

- ▶ v : the **weights** for the **hidden state** of the **current time step** $h^{(t)}$.
- ▶ $\hat{y}^{(t)}$ is the **output** at step t .
- ▶ $\hat{y}^{(t)} = \text{softmax}(vh^{(t)})$



Recurrent Neurons - Weights (4/4)

- ▶ v : the **weights for the hidden state** of the **current time step** $h^{(t)}$.
- ▶ $\hat{y}^{(t)}$ is the **output** at step t .
- ▶ $\hat{y}^{(t)} = \text{softmax}(vh^{(t)})$
- ▶ For example, if we wanted to **predict the next word** in a sentence, it would be a **vector of probabilities** across our vocabulary.

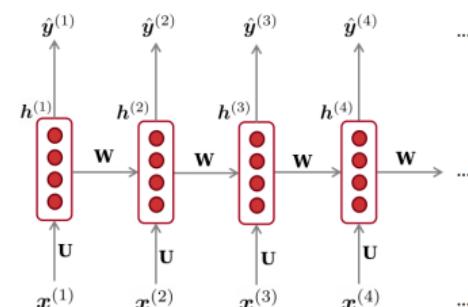
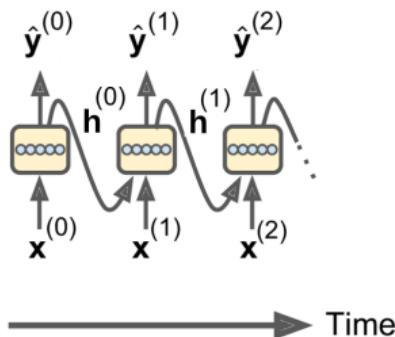


Layers of Recurrent Neurons

- ▶ At each time step t , every neuron of a layer receives both the input vector $x^{(t)}$ and the output vector from the previous time step $h^{(t-1)}$.

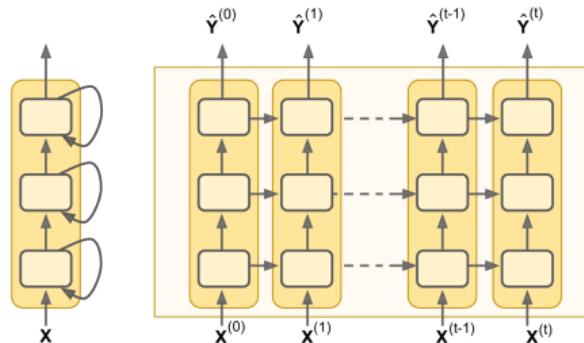
$$h^{(t)} = \tanh(u^T x^{(t)} + w^T h^{(t-1)})$$

$$y^{(t)} = \text{sigmoid}(v^T h^{(t)})$$



Deep RNN

- ▶ Stacking **multiple layers** of cells gives you a **deep RNN**.





Let's Back to Language Model Example

A RNN Neural Language Model (1/2)

- ▶ The input x will be a **sequence of words** (each $x^{(t)}$ is a **single word**).
- ▶ Each embedded word $e^{(t)} = E^T x^{(t)}$ is a **one-hot vector** of size **vocabulary size**.

students *opened* word V

$$x^{(1)} \text{ students} = [1, 0, 0, 0, 0, 0, 0, \dots, 0]$$

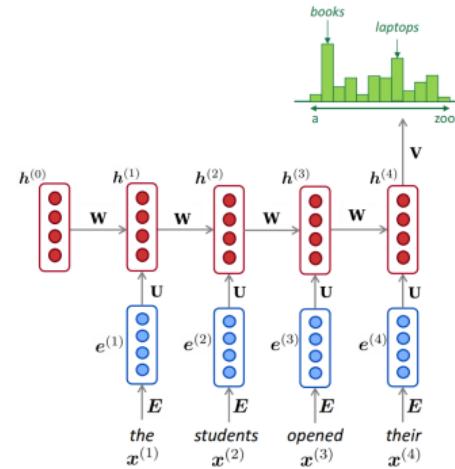
$$x^{(2)} \text{ opened} = [0, 1, 0, 0, 0, 0, 0, \dots, 0]$$

$$x^{(3)} \text{ their} = [0, 0, 1, 0, 0, 0, 0, \dots, 0]$$

$$x^{(4)} \text{ book} = [0, 0, 0, 1, 0, 0, 0, \dots, 0]$$

$\underbrace{\hspace{10em}}$

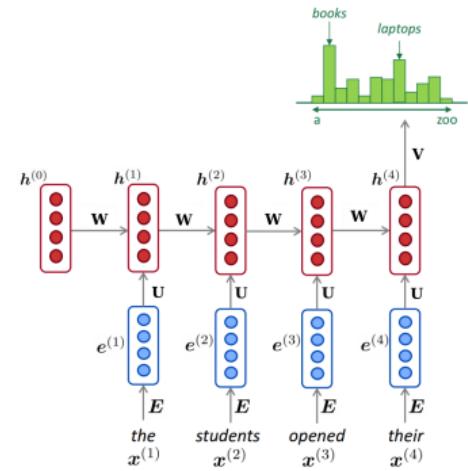
$e^{(t)}$



A RNN Neural Language Model (2/2)

► Let's recap the equations for the RNN:

- $h^{(t)} = \tanh(u^T e^{(t)} + w h^{(t-1)})$
- $\hat{y}^{(t)} = \text{softmax}(v h^{(t)})$

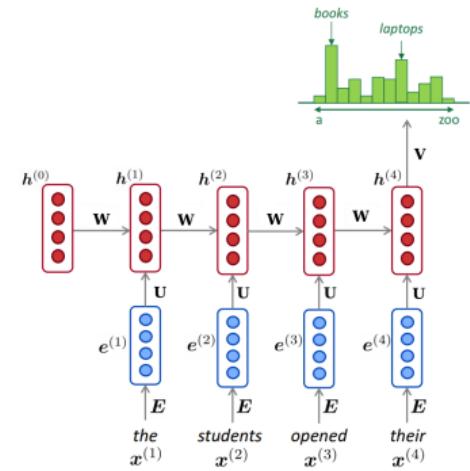


A RNN Neural Language Model (2/2)

- ▶ Let's recap the equations for the RNN:

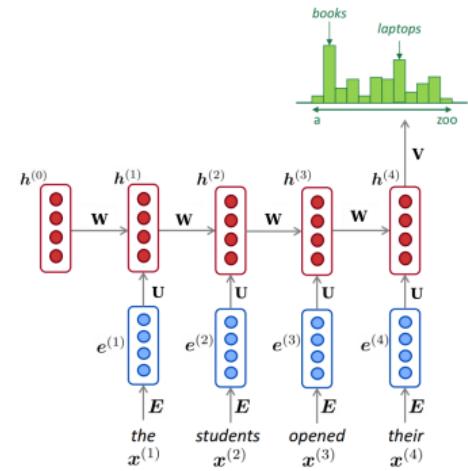
- $h^{(t)} = \tanh(u^T e^{(t)} + w h^{(t-1)})$
- $\hat{y}^{(t)} = \text{softmax}(v h^{(t)})$

- ▶ The output $\hat{y}^{(t)}$ is a vector of **vocabulary size** elements.



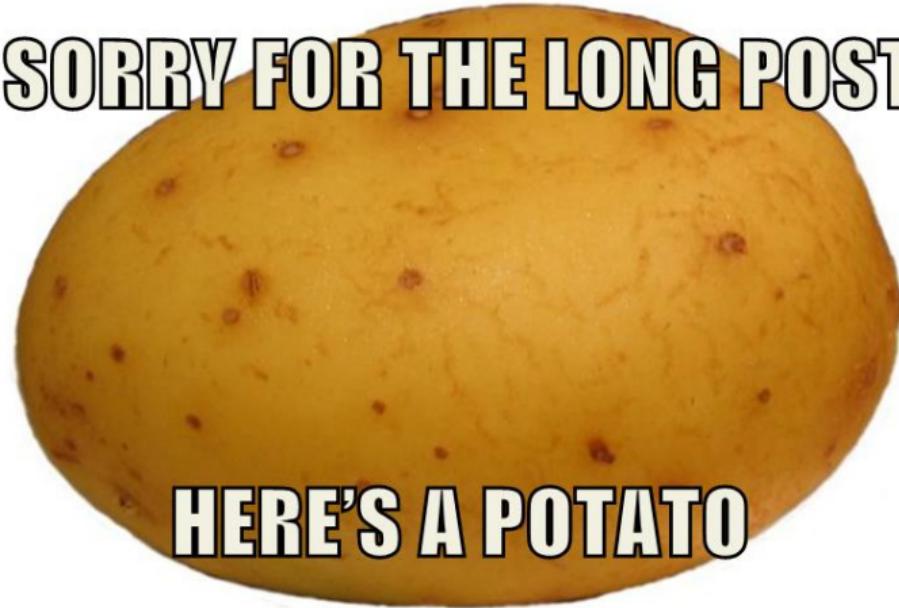
A RNN Neural Language Model (2/2)

- ▶ Let's recap the equations for the RNN:
 - $h^{(t)} = \tanh(u^T e^{(t)} + w h^{(t-1)})$
 - $\hat{y}^{(t)} = \text{softmax}(v h^{(t)})$
- ▶ The output $\hat{y}^{(t)}$ is a vector of **vocabulary size** elements.
- ▶ Each element of $\hat{y}^{(t)}$ represents the **probability** of that word being the **next word** in the sentence.





SORRY FOR THE LONG POST



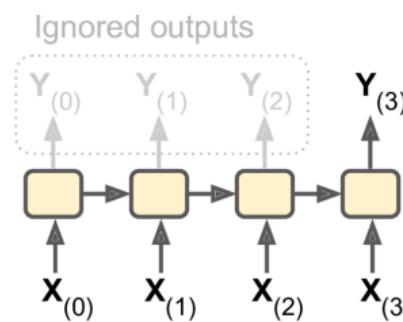
HERE'S A POTATO



RNN Design Patterns

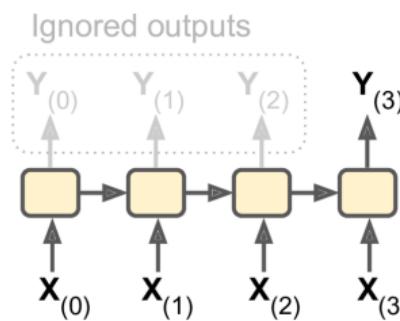
RNN Design Patterns - Sequence-to-Vector

- ▶ **Sequence-to-vector** network: takes a **sequence of inputs**, and ignore all outputs except for **the last one**.



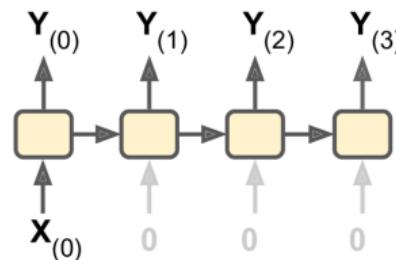
RNN Design Patterns - Sequence-to-Vector

- ▶ **Sequence-to-vector** network: takes a **sequence of inputs**, and ignore all outputs except for the last one.
- ▶ E.g., you could feed the network a **sequence of words** corresponding to a movie review, and the network would output a **sentiment score**.



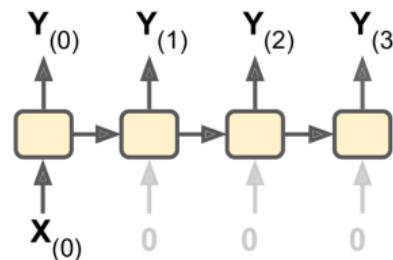
RNN Design Patterns - Vector-to-Sequence

- ▶ **Vector-to-sequence** network: takes a **single input** at the first time step, and let it **output a sequence**.



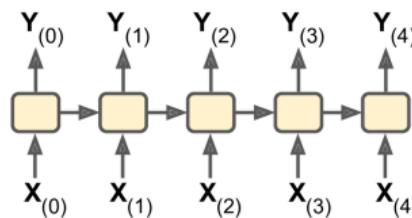
RNN Design Patterns - Vector-to-Sequence

- ▶ **Vector-to-sequence** network: takes a **single input** at the first time step, and let it **output a sequence**.
- ▶ E.g., the input could be an **image**, and the output could be a **caption** for that image.



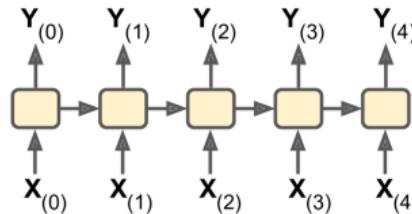
RNN Design Patterns - Sequence-to-Sequence

- ▶ **Sequence-to-sequence** network: takes a **sequence of inputs** and produce a **sequence of outputs**.



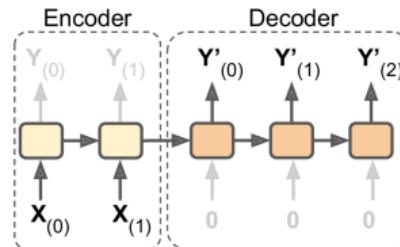
RNN Design Patterns - Sequence-to-Sequence

- ▶ **Sequence-to-sequence** network: takes a **sequence of inputs** and produce a **sequence of outputs**.
- ▶ Useful for **predicting time series such as stock prices**: you feed it the prices over the last N days, and it must output the prices shifted by one day into the future.
- ▶ Here, both input sequences and output sequences have the **same length**.



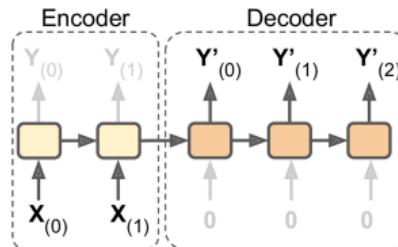
RNN Design Patterns - Encoder-Decoder

- ▶ **Encoder-decoder** network: a **sequence-to-vector** network (**encoder**), followed by a **vector-to-sequence** network (**decoder**).



RNN Design Patterns - Encoder-Decoder

- ▶ **Encoder-decoder** network: a **sequence-to-vector** network (**encoder**), followed by a **vector-to-sequence** network (**decoder**).
- ▶ E.g., **translating** a sentence from one language to another.
- ▶ You would feed the network **a sentence in one language**, the encoder would convert this sentence into a **single vector representation**, and then the decoder would decode this vector into a sentence in another language.

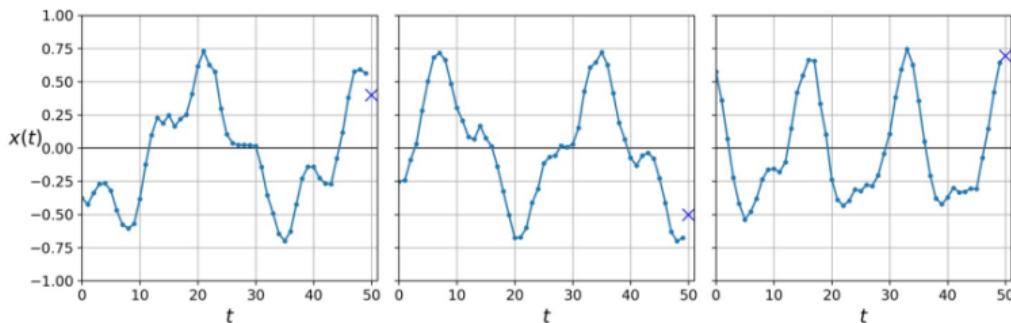




RNN in TensorFlow

RNN in TensorFlow (1/5)

- ▶ Forecasting a **time series**
- ▶ E.g., a dataset of 10000 time series, each of them **50 time steps long**.
- ▶ The goal here is to **forecast the value at the next time step** (represented by the X) for each of them.





RNN in TensorFlow (2/5)

- ▶ Use fully connected network

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[50, 1]),
    keras.layers.Dense(1)
])

model.compile(loss="mse", optimizer="adam")
history = model.fit(X_train, y_train, epochs=20)

model.evaluate(X_test, y_test, verbose=0)
# loss: 0.003993967570985357
```



RNN in TensorFlow (3/5)

► Simple RNN

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(1, input_shape=[None, 1])
])

model.compile(loss="mse", optimizer='adam')
history = model.fit(X_train, y_train, epochs=20)

model.evaluate(X_test, y_test, verbose=0)
# loss: 0.011026302369932333
```



RNN in TensorFlow (4/5)

► Deep RNN

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.SimpleRNN(1)
])

model.compile(loss="mse", optimizer="adam")
history = model.fit(X_train, y_train, epochs=20)

model.evaluate(X_test, y_test, verbose=0)
# loss: 0.003197280486735205
```



RNN in TensorFlow (5/5)

- ▶ Deep RNN (second implementation)
- ▶ Make the second layer return only the last output (no `return_sequences`)

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20),
    keras.layers.Dense(1)
])

model.compile(loss="mse", optimizer="adam")
history = model.fit(X_train, y_train, epochs=20)

model.evaluate(X_test, y_test, verbose=0)
# loss: 0.002757748544837038
```



Training RNNs

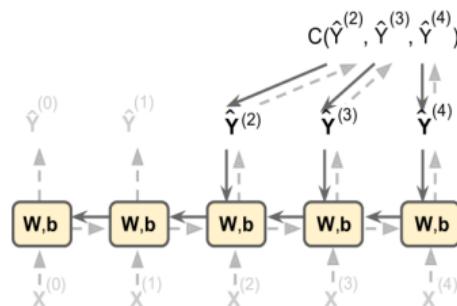


Training RNNs

- ▶ To train an RNN, we should unroll it through time and then simply use regular backpropagation.
- ▶ This strategy is called backpropagation through time (BPTT).

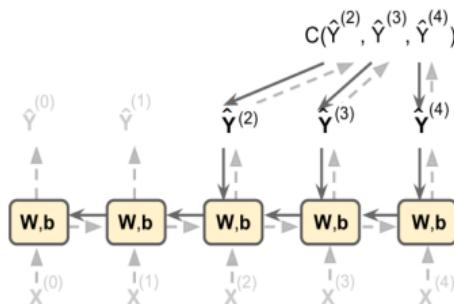
Backpropagation Through Time (1/3)

- ▶ To train the model using **BPTT**, we go through the following steps:
- ▶ 1. **Forward pass** through the **unrolled network** (represented by the dashed arrows).
- ▶ 2. The **cost function** is $C(\hat{y}^{t_{\min}}, \hat{y}^{t_{\min}+1}, \dots, \hat{y}^{t_{\max}})$, where t_{\min} and t_{\max} are the first and last output time steps, **not counting the ignored outputs**.



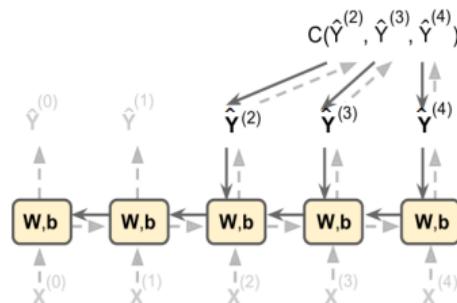
Backpropagation Through Time (2/3)

- ▶ 3. Propagate backward the gradients of that cost function through the unrolled network (represented by the solid arrows).
- ▶ 4. The model parameters are updated using the gradients computed during BPTT.

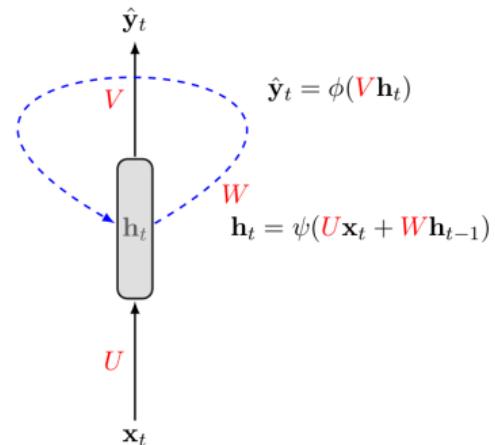


Backpropagation Through Time (3/3)

- ▶ The gradients **flow backward** through **all the outputs** used by the cost function, **not just through the final output**.
- ▶ For example, in the following figure:
 - The **cost function** is computed using the **last three outputs**, $\hat{y}^{(2)}$, $\hat{y}^{(3)}$, and $\hat{y}^{(4)}$.
 - Gradients flow through these three outputs, but **not through** $\hat{y}^{(0)}$ and $\hat{y}^{(1)}$.



BPTT Step by Step (1/20)

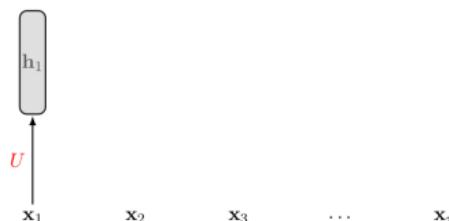




BPTT Step by Step (2/20)

x_1 x_2 x_3 \dots x_T

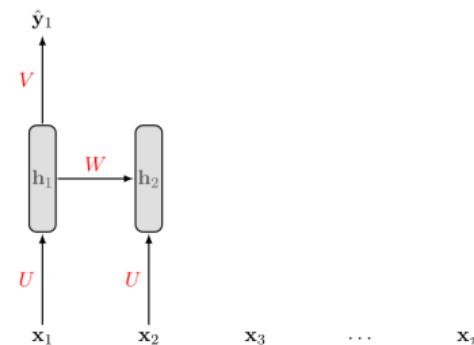
BPTT Step by Step (3/20)



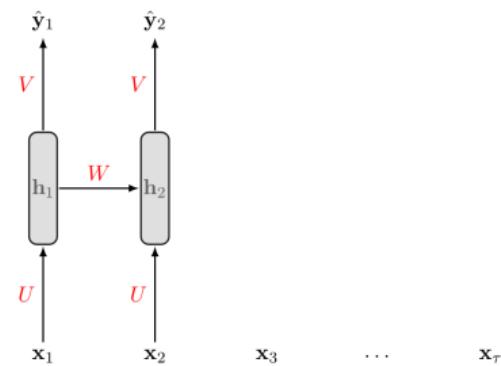
BPTT Step by Step (4/20)



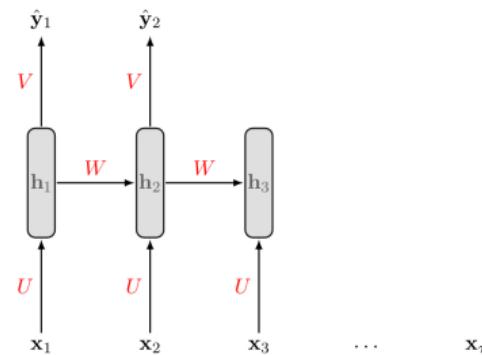
BPTT Step by Step (5/20)



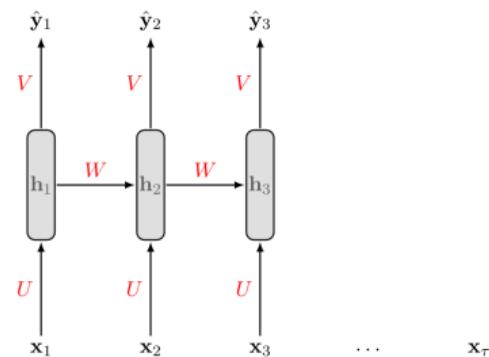
BPTT Step by Step (6/20)



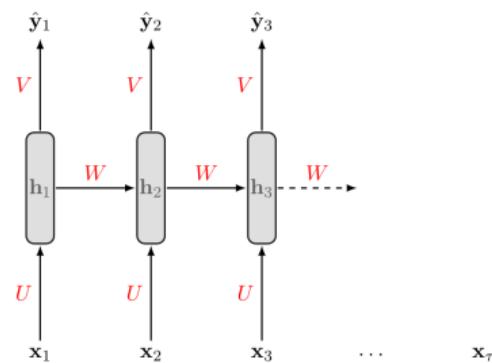
BPTT Step by Step (7/20)



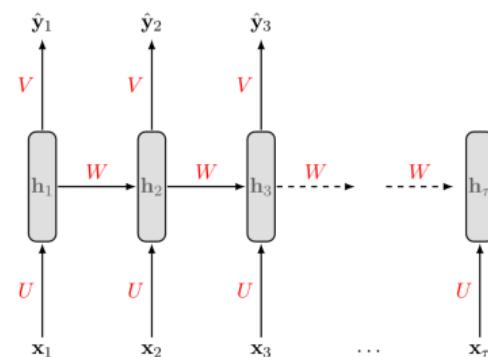
BPTT Step by Step (8/20)



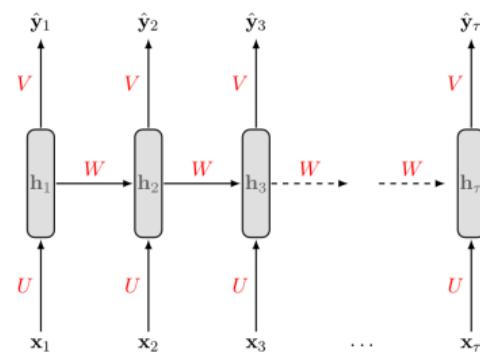
BPTT Step by Step (9/20)



BPTT Step by Step (10/20)



BPTT Step by Step (11/20)



BPTT Step by Step (12/20)

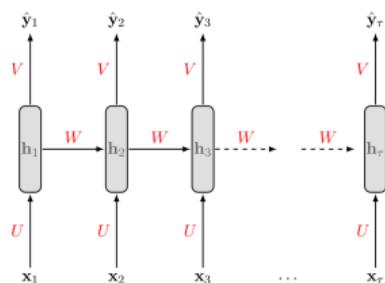
$$s^{(t)} = u^T x^{(t)} + w h^{(t-1)}$$

$$h^{(t)} = \tanh(s^{(t)})$$

$$z^{(t)} = v h^{(t)}$$

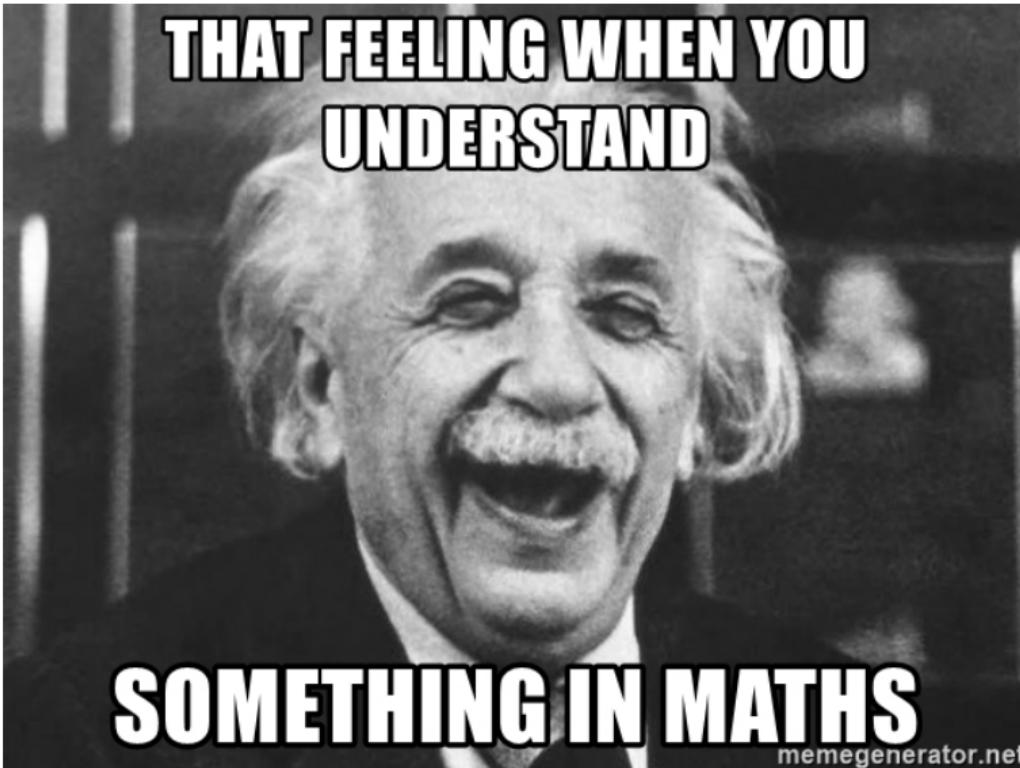
$$\hat{y}^{(t)} = \text{softmax}(z^{(t)})$$

$$J^{(t)} = \text{cross_entropy}(y^{(t)}, \hat{y}^{(t)}) = - \sum y^{(t)} \log \hat{y}^{(t)}$$





**THAT FEELING WHEN YOU
UNDERSTAND**



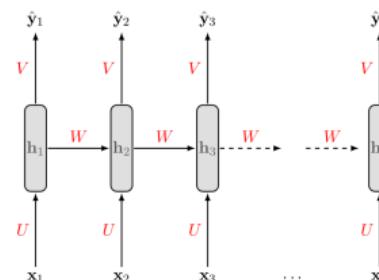
SOMETHING IN MATHS

memegenerator.net

BPTT Step by Step (13/20)

$$J^{(t)} = \text{cross_entropy}(y^{(t)}, \hat{y}^{(t)}) = - \sum y^{(t)} \log \hat{y}^{(t)}$$

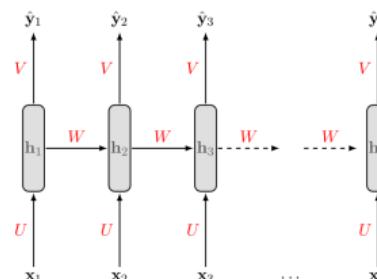
- We treat the **full sequence** as one training example.



BPTT Step by Step (13/20)

$$J^{(t)} = \text{cross_entropy}(y^{(t)}, \hat{y}^{(t)}) = - \sum y^{(t)} \log \hat{y}^{(t)}$$

- We treat the **full sequence** as one training example.
- The **total error E** is just the **sum of the errors at each time step**.
- E.g., $E = J^{(1)} + J^{(2)} + \dots + J^{(t)}$





BPTT Step by Step (14/20)

- ▶ $J^{(t)}$ is the total cost, so we can say that a 1-unit increase in v , w or u will impact each of $J^{(1)}$, $J^{(2)}$, until $J^{(t)}$ individually.



BPTT Step by Step (14/20)

- ▶ $J^{(t)}$ is the total cost, so we can say that a 1-unit increase in v , w or u will impact each of $J^{(1)}$, $J^{(2)}$, until $J^{(t)}$ individually.
- ▶ The gradient is equal to the sum of the respective gradients at each time step t .



BPTT Step by Step (14/20)

- ▶ $J^{(t)}$ is the total cost, so we can say that a 1-unit increase in v , w or u will impact each of $J^{(1)}$, $J^{(2)}$, until $J^{(t)}$ individually.
- ▶ The gradient is equal to the sum of the respective gradients at each time step t .
- ▶ For example if $t = 3$ we have: $E = J^{(1)} + J^{(2)} + J^{(3)}$



BPTT Step by Step (14/20)

- ▶ $J^{(t)}$ is the total cost, so we can say that a 1-unit increase in v , w or u will impact each of $J^{(1)}$, $J^{(2)}$, until $J^{(t)}$ individually.
- ▶ The gradient is equal to the sum of the respective gradients at each time step t .
- ▶ For example if $t = 3$ we have: $E = J^{(1)} + J^{(2)} + J^{(3)}$

$$\frac{\partial E}{\partial v} = \sum_t \frac{\partial J^{(t)}}{\partial v} = \frac{\partial J^{(3)}}{\partial v} + \frac{\partial J^{(2)}}{\partial v} + \frac{\partial J^{(1)}}{\partial v}$$

BPTT Step by Step (14/20)

- ▶ $J^{(t)}$ is the total cost, so we can say that a 1-unit increase in v , w or u will impact each of $J^{(1)}$, $J^{(2)}$, until $J^{(t)}$ individually.
- ▶ The gradient is equal to the sum of the respective gradients at each time step t .
- ▶ For example if $t = 3$ we have: $E = J^{(1)} + J^{(2)} + J^{(3)}$

$$\frac{\partial E}{\partial v} = \sum_t \frac{\partial J^{(t)}}{\partial v} = \frac{\partial J^{(3)}}{\partial v} + \frac{\partial J^{(2)}}{\partial v} + \frac{\partial J^{(1)}}{\partial v}$$

$$\frac{\partial E}{\partial w} = \sum_t \frac{\partial J^{(t)}}{\partial w} = \frac{\partial J^{(3)}}{\partial w} + \frac{\partial J^{(2)}}{\partial w} + \frac{\partial J^{(1)}}{\partial w}$$

BPTT Step by Step (14/20)

- ▶ $J^{(t)}$ is the total cost, so we can say that a 1-unit increase in v , w or u will impact each of $J^{(1)}$, $J^{(2)}$, until $J^{(t)}$ individually.
- ▶ The gradient is equal to the sum of the respective gradients at each time step t .
- ▶ For example if $t = 3$ we have: $E = J^{(1)} + J^{(2)} + J^{(3)}$

$$\frac{\partial E}{\partial v} = \sum_t \frac{\partial J^{(t)}}{\partial v} = \frac{\partial J^{(3)}}{\partial v} + \frac{\partial J^{(2)}}{\partial v} + \frac{\partial J^{(1)}}{\partial v}$$

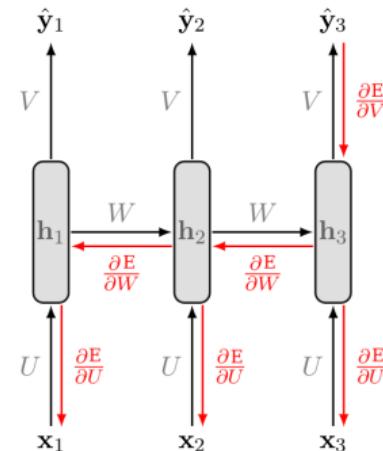
$$\frac{\partial E}{\partial w} = \sum_t \frac{\partial J^{(t)}}{\partial w} = \frac{\partial J^{(3)}}{\partial w} + \frac{\partial J^{(2)}}{\partial w} + \frac{\partial J^{(1)}}{\partial w}$$

$$\frac{\partial E}{\partial u} = \sum_t \frac{\partial J^{(t)}}{\partial u} = \frac{\partial J^{(3)}}{\partial u} + \frac{\partial J^{(2)}}{\partial u} + \frac{\partial J^{(1)}}{\partial u}$$

BPTT Step by Step (15/20)

- ▶ Let's start with $\frac{\partial E}{\partial v}$.
- ▶ A change in v will only impact $J^{(3)}$ at time $t = 3$, because it plays no role in computing the value of anything other than $z^{(3)}$.

$$\frac{\partial E}{\partial v} = \sum_t \frac{\partial J^{(t)}}{\partial v} = \frac{\partial J^{(3)}}{\partial v} + \frac{\partial J^{(2)}}{\partial v} + \frac{\partial J^{(1)}}{\partial v}$$

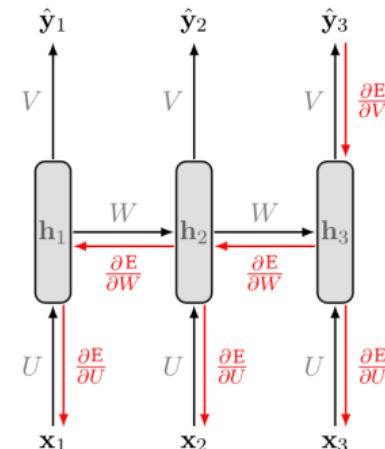


BPTT Step by Step (15/20)

- ▶ Let's start with $\frac{\partial E}{\partial v}$.
- ▶ A change in v will only impact $J^{(3)}$ at time $t = 3$, because it plays no role in computing the value of anything other than $z^{(3)}$.

$$\frac{\partial E}{\partial v} = \sum_t \frac{\partial J^{(t)}}{\partial v} = \frac{\partial J^{(3)}}{\partial v} + \frac{\partial J^{(2)}}{\partial v} + \frac{\partial J^{(1)}}{\partial v}$$

$$\frac{\partial J^{(3)}}{\partial v} = \frac{\partial J^{(3)}}{\partial \hat{y}^{(3)}} \frac{\partial \hat{y}^{(3)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial v}$$



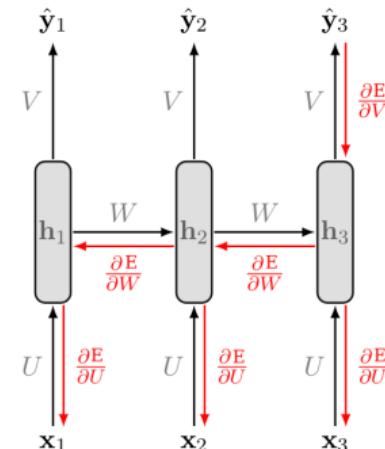
BPTT Step by Step (15/20)

- ▶ Let's start with $\frac{\partial E}{\partial v}$.
- ▶ A change in v will only impact $J^{(3)}$ at time $t = 3$, because it plays no role in computing the value of anything other than $z^{(3)}$.

$$\frac{\partial E}{\partial v} = \sum_t \frac{\partial J^{(t)}}{\partial v} = \frac{\partial J^{(3)}}{\partial v} + \frac{\partial J^{(2)}}{\partial v} + \frac{\partial J^{(1)}}{\partial v}$$

$$\frac{\partial J^{(3)}}{\partial v} = \frac{\partial J^{(3)}}{\partial \hat{y}^{(3)}} \frac{\partial \hat{y}^{(3)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial v}$$

$$\frac{\partial J^{(2)}}{\partial v} = \frac{\partial J^{(2)}}{\partial \hat{y}^{(2)}} \frac{\partial \hat{y}^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial v}$$



BPTT Step by Step (15/20)

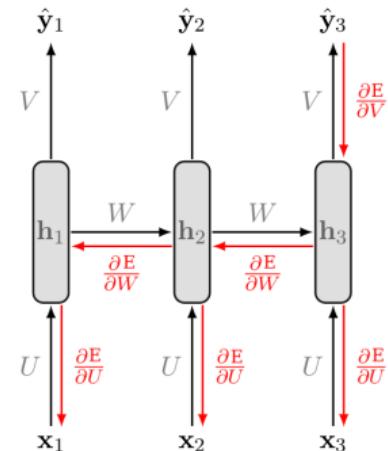
- ▶ Let's start with $\frac{\partial E}{\partial v}$.
- ▶ A change in v will only impact $J^{(3)}$ at time $t = 3$, because it plays no role in computing the value of anything other than $z^{(3)}$.

$$\frac{\partial E}{\partial v} = \sum_t \frac{\partial J^{(t)}}{\partial v} = \frac{\partial J^{(3)}}{\partial v} + \frac{\partial J^{(2)}}{\partial v} + \frac{\partial J^{(1)}}{\partial v}$$

$$\frac{\partial J^{(3)}}{\partial v} = \frac{\partial J^{(3)}}{\partial \hat{y}^{(3)}} \frac{\partial \hat{y}^{(3)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial v}$$

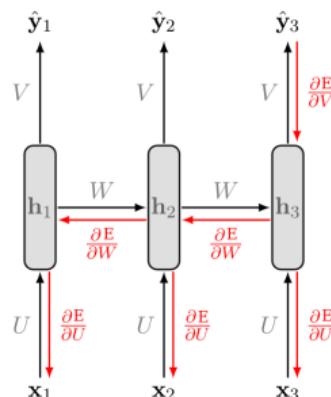
$$\frac{\partial J^{(2)}}{\partial v} = \frac{\partial J^{(2)}}{\partial \hat{y}^{(2)}} \frac{\partial \hat{y}^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial v}$$

$$\frac{\partial J^{(1)}}{\partial v} = \frac{\partial J^{(1)}}{\partial \hat{y}^{(1)}} \frac{\partial \hat{y}^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial v}$$



BPTT Step by Step (16/20)

- ▶ Let's compute the derivatives of $\frac{\partial J}{\partial w}$ and $\frac{\partial J}{\partial u}$, which are **computed the same**.
- ▶ A change in w at $t = 3$ will impact our cost J in 3 separate ways:
 1. When computing the value of $h^{(1)}$.
 2. When computing the value of $h^{(2)}$, which depends on $h^{(1)}$.
 3. When computing the value of $h^{(3)}$, which depends on $h^{(2)}$, which depends on $h^{(1)}$.

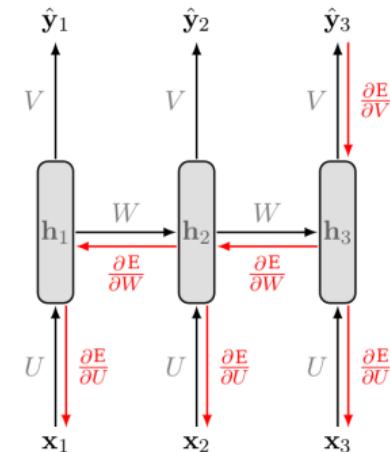


BPTT Step by Step (17/20)

- we compute our individual gradients as:

$$\sum_t \frac{\partial J^{(t)}}{\partial w} = \frac{\partial J^{(3)}}{\partial w} + \frac{\partial J^{(2)}}{\partial w} + \frac{\partial J^{(1)}}{\partial w}$$

$$\frac{\partial J^{(1)}}{\partial w} = \frac{\partial J^{(1)}}{\partial \hat{y}^{(1)}} \frac{\partial \hat{y}^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial s^{(1)}} \frac{\partial s^{(1)}}{\partial w}$$

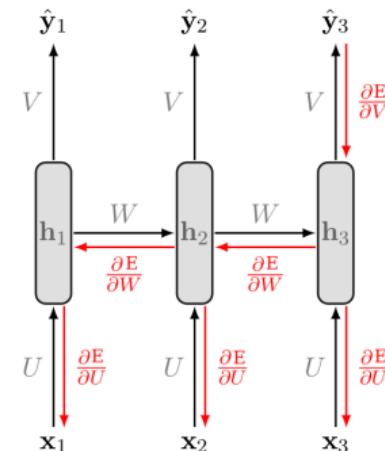


BPTT Step by Step (18/20)

- we compute our individual gradients as:

$$\sum_t \frac{\partial J^{(t)}}{\partial w} = \frac{\partial J^{(3)}}{\partial w} + \frac{\partial J^{(2)}}{\partial w} + \frac{\partial J^{(1)}}{\partial w}$$

$$\begin{aligned} \frac{\partial J^{(2)}}{\partial w} = & \frac{\partial J^{(2)}}{\partial \hat{y}^{(2)}} \frac{\partial \hat{y}^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial s^{(2)}} \frac{\partial s^{(2)}}{\partial w} + \\ & \frac{\partial J^{(2)}}{\partial \hat{y}^{(2)}} \frac{\partial \hat{y}^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial s^{(2)}} \frac{\partial s^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial s^{(1)}} \frac{\partial s^{(1)}}{\partial w} \end{aligned}$$

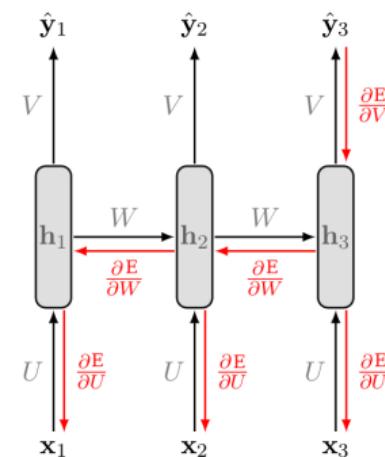


BPTT Step by Step (19/20)

- we compute our individual gradients as:

$$\sum_t \frac{\partial J^{(t)}}{\partial w} = \frac{\partial J^{(3)}}{\partial w} + \frac{\partial J^{(2)}}{\partial w} + \frac{\partial J^{(1)}}{\partial w}$$

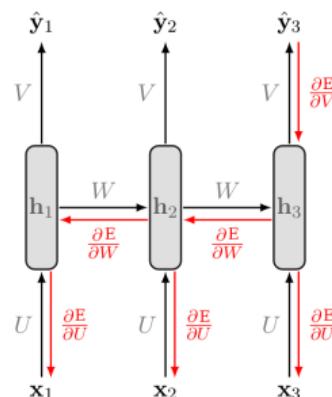
$$\begin{aligned} \frac{\partial J^{(3)}}{\partial w} &= \frac{\partial J^{(3)}}{\partial \hat{y}^{(3)}} \frac{\partial \hat{y}^{(3)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial s^{(3)}} \frac{\partial s^{(3)}}{\partial w} + \\ &\quad \frac{\partial J^{(3)}}{\partial \hat{y}^{(3)}} \frac{\partial \hat{y}^{(3)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial s^{(3)}} \frac{\partial s^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial s^{(2)}} \frac{\partial s^{(2)}}{\partial w} + \\ &\quad \frac{\partial J^{(3)}}{\partial \hat{y}^{(3)}} \frac{\partial \hat{y}^{(3)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial s^{(3)}} \frac{\partial s^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial s^{(2)}} \frac{\partial s^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial s^{(1)}} \frac{\partial s^{(1)}}{\partial w} \end{aligned}$$



BPTT Step by Step (20/20)

- More generally, a change in w will impact our cost $J^{(t)}$ on t separate occasions.

$$\frac{\partial J^{(t)}}{\partial w} = \sum_{k=1}^t \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial z^{(t)}} \frac{\partial z^{(t)}}{\partial h^{(t)}} \left(\prod_{j=k+1}^t \frac{\partial h^{(j)}}{\partial s^{(j)}} \frac{\partial s^{(j)}}{\partial h^{(j-1)}} \right) \frac{\partial h^{(k)}}{\partial s^{(k)}} \frac{\partial s^{(k)}}{\partial w}$$





LSTM



RNN Problems

- ▶ Sometimes we only need to look at **recent information** to perform the present task.
 - E.g., **predicting the next word** based on the previous ones.



RNN Problems

- ▶ Sometimes we only need to look at **recent information** to perform the present task.
 - E.g., **predicting the next word** based on the previous ones.
- ▶ In such cases, where the **gap between the relevant information and the place that it's needed is small**, RNNs can learn to use the past information.



RNN Problems

- ▶ Sometimes we only need to look at **recent information** to perform the present task.
 - E.g., **predicting the next word** based on the previous ones.
- ▶ In such cases, where the **gap between the relevant information and the place that it's needed is small**, RNNs can learn to use the past information.
- ▶ But, as that **gap grows**, RNNs become **unable to learn** to connect the information.
- ▶ RNNs may suffer from the **vanishing/exploding gradients problem**.



RNN Problems

- ▶ Sometimes we only need to look at **recent information** to perform the present task.
 - E.g., **predicting the next word** based on the previous ones.
- ▶ In such cases, where the **gap between the relevant information and the place that it's needed is small**, RNNs can learn to use the past information.
- ▶ But, as that **gap grows**, RNNs become **unable to learn** to connect the information.
- ▶ RNNs may suffer from the **vanishing/exploding gradients problem**.
- ▶ To solve these problems, **long short-term memory (LSTM)** have been introduced.



RNN Problems

- ▶ Sometimes we only need to look at **recent information** to perform the present task.
 - E.g., **predicting the next word** based on the previous ones.
- ▶ In such cases, where the **gap between the relevant information and the place that it's needed is small**, RNNs can learn to use the past information.
- ▶ But, as that **gap grows**, RNNs become **unable to learn** to connect the information.
- ▶ RNNs may suffer from the **vanishing/exploding gradients problem**.
- ▶ To solve these problems, **long short-term memory (LSTM)** have been introduced.
- ▶ In LSTM, the network can learn **what to store** and **what to throw away**.

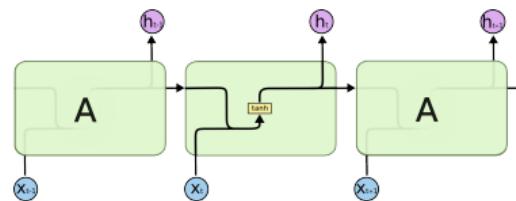


RNN Basic Cell vs. LSTM

- ▶ Without looking inside the box, the **LSTM** cell looks exactly like a **basic RNN cell**.

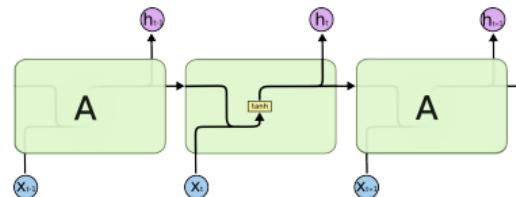
RNN Basic Cell vs. LSTM

- ▶ Without looking inside the box, the **LSTM** cell looks exactly like a **basic RNN** cell.
- ▶ A **basic RNN** contains a **single layer** in each cell.

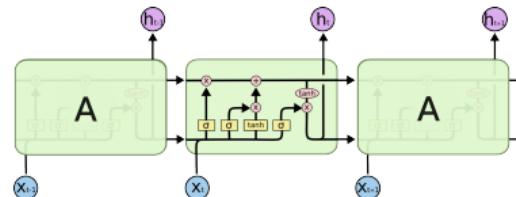


RNN Basic Cell vs. LSTM

- Without looking inside the box, the **LSTM** cell looks exactly like a **basic RNN** cell.
- A **basic RNN** contains a **single layer** in each cell.

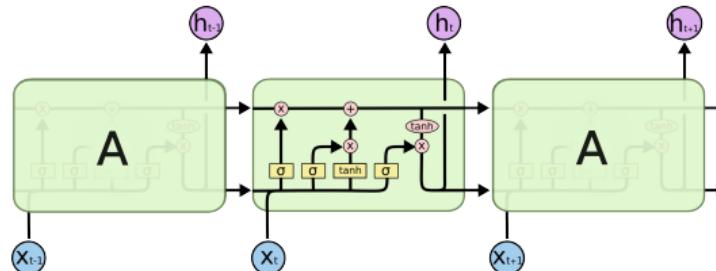


- An **LSTM** contains **four interacting layers** in each cell.



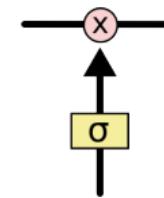
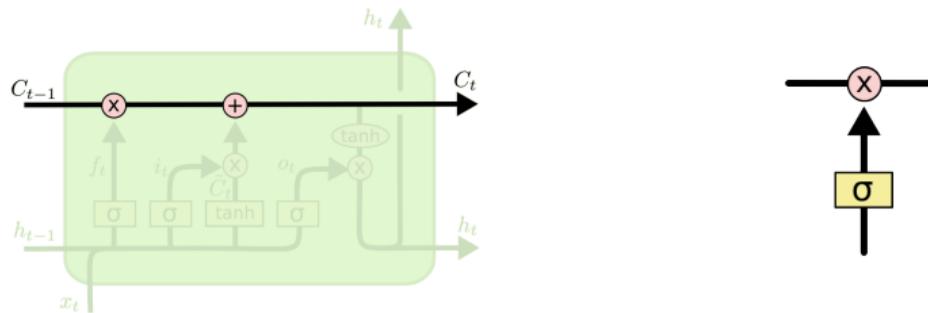
LSTM (1/2)

- ▶ In LSTM **state** is split in **two vectors**:
 1. $h^{(t)}$ (**h** stands for **hidden**): the **short-term** state
 2. $c^{(t)}$ (**c** stands for **cell**): the **long-term** state



LSTM (2/2)

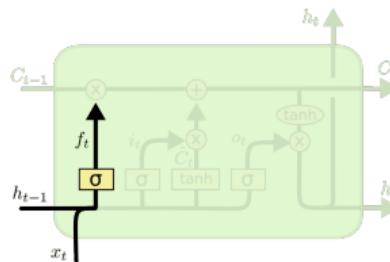
- ▶ The **cell state** (long-term state), the horizontal line on the top of the diagram.
- ▶ The LSTM can **remove/add information** to the **cell state**, regulated by **three gates**.
 - Forget gate, input gate and output gate



Step-by-Step LSTM Walk Through (1/4)

- ▶ Step one: decides what information we are going to throw away from the **cell state**.

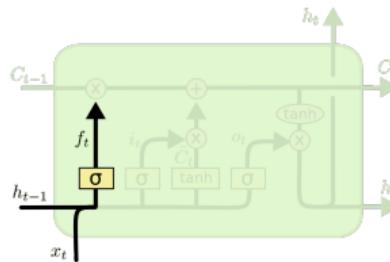
$$f^{(t)} = \sigma(u_f^T x^{(t)} + w_f h^{(t-1)})$$



Step-by-Step LSTM Walk Through (1/4)

- ▶ Step one: decides **what information** we are going to **throw away** from the **cell state**.
- ▶ This decision is made by a **sigmoid layer**, called the **forget gate** layer.

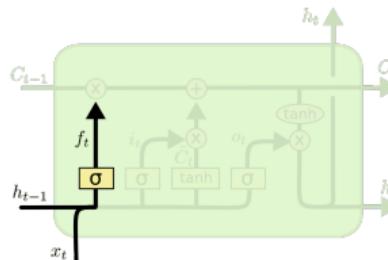
$$f^{(t)} = \sigma(u_f^T x^{(t)} + w_f h^{(t-1)})$$



Step-by-Step LSTM Walk Through (1/4)

- ▶ **Step one:** decides **what information** we are going to **throw away** from the **cell state**.
- ▶ This decision is made by a **sigmoid layer**, called the **forget gate** layer.
- ▶ It looks at $h^{(t-1)}$ and $x^{(t)}$, and outputs a number between 0 and 1 for each number in the cell state $c^{(t-1)}$.
 - 1 represents **completely keep this**, and 0 represents **completely get rid of this**.

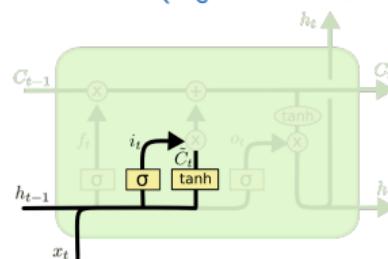
$$f^{(t)} = \sigma(u_f^T x^{(t)} + w_f h^{(t-1)})$$



Step-by-Step LSTM Walk Through (2/4)

- **Second step:** decides **what new information** we are going to **store** in the **cell state**. This has two parts:

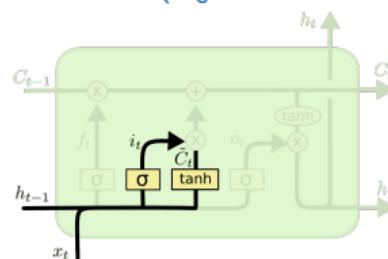
$$i^{(t)} = \sigma(\mathbf{u}_i^T \mathbf{x}^{(t)} + w_i h^{(t-1)})$$
$$\tilde{c}^{(t)} = \tanh(\mathbf{u}_{\tilde{c}}^T \mathbf{x}^{(t)} + w_{\tilde{c}} h^{(t-1)})$$



Step-by-Step LSTM Walk Through (2/4)

- ▶ **Second step:** decides **what new information** we are going to **store** in the **cell state**. This has two parts:
 - ▶ 1. A **sigmoid layer**, called the **input gate** layer, decides **which values** we will update.

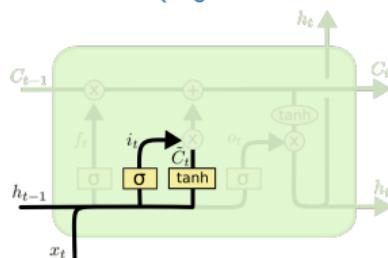
$$i^{(t)} = \sigma(\mathbf{u}_i^T \mathbf{x}^{(t)} + \mathbf{w}_i \mathbf{h}^{(t-1)})$$
$$\tilde{c}^{(t)} = \tanh(\mathbf{u}_{\tilde{c}}^T \mathbf{x}^{(t)} + \mathbf{w}_{\tilde{c}} \mathbf{h}^{(t-1)})$$



Step-by-Step LSTM Walk Through (2/4)

- ▶ **Second step:** decides **what new information** we are going to **store** in the **cell state**. This has two parts:
 - ▶ 1. A **sigmoid layer**, called the **input gate** layer, decides **which values** we will update.
 - ▶ 2. A **tanh layer** creates a vector of **new candidate values** that could be added to the state.

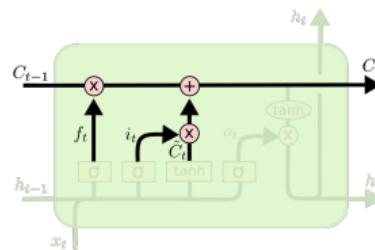
$$i^{(t)} = \sigma(\mathbf{u}_i^T \mathbf{x}^{(t)} + \mathbf{w}_i \mathbf{h}^{(t-1)})$$
$$\tilde{c}^{(t)} = \tanh(\mathbf{u}_{\tilde{c}}^T \mathbf{x}^{(t)} + \mathbf{w}_{\tilde{c}} \mathbf{h}^{(t-1)})$$



Step-by-Step LSTM Walk Through (3/4)

- Third step: updates the old cell state $c^{(t-1)}$, into the new cell state $c^{(t)}$.

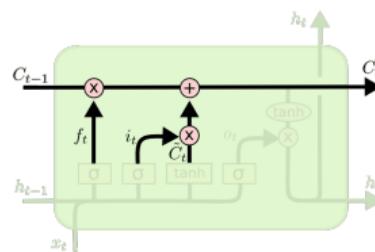
$$c^{(t)} = f^{(t)} \otimes c^{(t-1)} + i^{(t)} \otimes \tilde{c}^{(t)}$$



Step-by-Step LSTM Walk Through (3/4)

- ▶ Third step: updates the old cell state $c^{(t-1)}$, into the new cell state $c^{(t)}$.
- ▶ We multiply the old state by $f^{(t)}$, forgetting the things we decided to forget earlier.

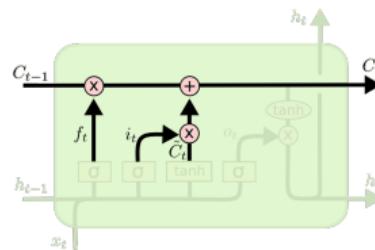
$$c^{(t)} = f^{(t)} \otimes c^{(t-1)} + i^{(t)} \otimes \tilde{c}^{(t)}$$



Step-by-Step LSTM Walk Through (3/4)

- ▶ **Third step:** updates the **old cell state** $c^{(t-1)}$, into the **new cell state** $c^{(t)}$.
- ▶ We multiply the **old state** by $f^{(t)}$, forgetting the things we decided to forget earlier.
- ▶ Then we add it $i^{(t)} \otimes \tilde{c}^{(t)}$.

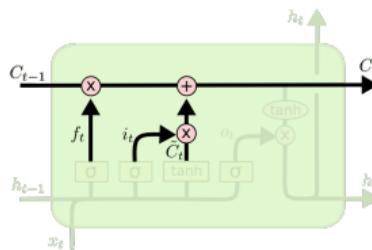
$$c^{(t)} = f^{(t)} \otimes c^{(t-1)} + i^{(t)} \otimes \tilde{c}^{(t)}$$



Step-by-Step LSTM Walk Through (3/4)

- ▶ Third step: updates the old cell state $c^{(t-1)}$, into the new cell state $c^{(t)}$.
- ▶ We multiply the old state by $f^{(t)}$, forgetting the things we decided to forget earlier.
- ▶ Then we add it $i^{(t)} \otimes \tilde{c}^{(t)}$.
- ▶ This is the new candidate values, scaled by how much we decided to update each state value.

$$c^{(t)} = f^{(t)} \otimes c^{(t-1)} + i^{(t)} \otimes \tilde{c}^{(t)}$$

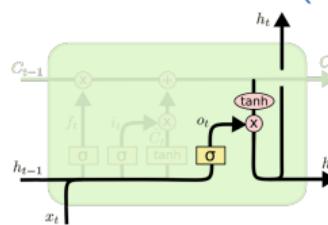


Step-by-Step LSTM Walk Through (4/4)

- ▶ **Fourth step:** decides about the **output**.

$$o^{(t)} = \sigma(u_o^T x^{(t)} + w_o h^{(t-1)})$$

$$h^{(t)} = o^{(t)} \otimes \tanh(c^{(t)})$$

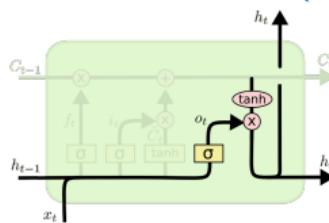


Step-by-Step LSTM Walk Through (4/4)

- ▶ **Fourth step:** decides about the **output**.
- ▶ First, runs a **sigmoid layer** that decides **what parts of the cell state** we are going to **output**.

$$o^{(t)} = \sigma(u_o^T x^{(t)} + w_o h^{(t-1)})$$

$$h^{(t)} = o^{(t)} \otimes \tanh(c^{(t)})$$

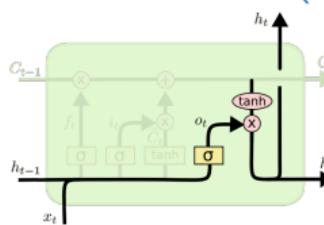


Step-by-Step LSTM Walk Through (4/4)

- ▶ **Fourth step:** decides about the **output**.
- ▶ First, runs a **sigmoid layer** that decides **what parts of the cell state** we are going to **output**.
- ▶ Then, puts the cell state through **tanh** and multiplies it by the output of the **sigmoid gate** (**output gate**), so that it **only outputs the parts it decided to**.

$$o^{(t)} = \sigma(u_o^T x^{(t)} + w_o h^{(t-1)})$$

$$h^{(t)} = o^{(t)} \otimes \tanh(c^{(t)})$$





LSTM in TensorFlow

► Use LSTM

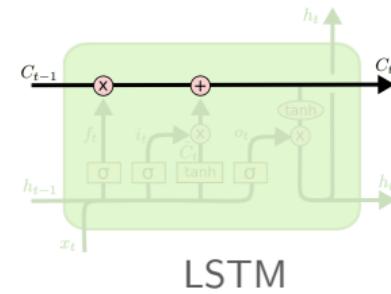
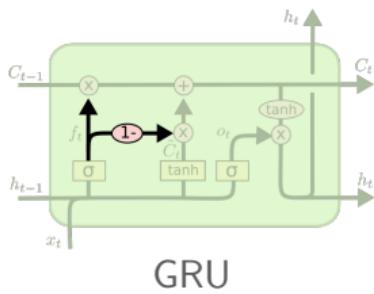
```
model = keras.models.Sequential([
    keras.layers.LSTM(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.LSTM(20),
    keras.layers.Dense(1)
])

model.compile(loss="mse", optimizer="adam", metrics=[last_time_step_mse])
history = model.fit(X_train, y_train, epochs=20)

model.evaluate(X_test, y_test, verbose=0)
```

Gated Recurrent Unit (GRU)

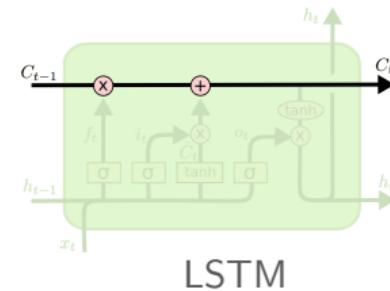
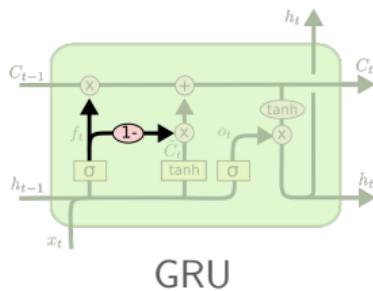
- The GRU cell is a simplified version of the LSTM cell.



Gated Recurrent Unit (GRU)

- ▶ The **GRU** cell is a **simplified version** of the LSTM cell.
- ▶ Instead of separately deciding **what to forget** and **what to add** to the new information to, it **makes those decisions together**.

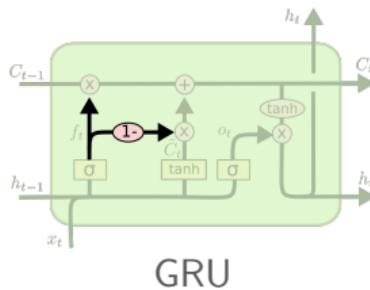
$$c^{(t)} = f^{(t)} \otimes c^{(t-1)} + (1 - f^{(t)}) \otimes \tilde{c}^{(t)}$$



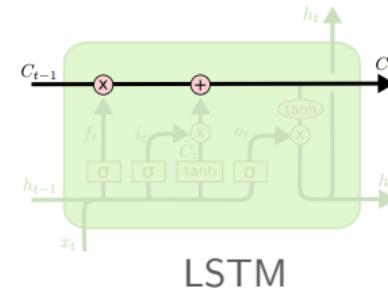
Gated Recurrent Unit (GRU)

- ▶ The **GRU** cell is a **simplified version** of the LSTM cell.
- ▶ Instead of separately deciding **what to forget** and **what to add** to the new information to, it **makes those decisions together**.
 - It only **forgets** when it is going to input something in its place.

$$c^{(t)} = f^{(t)} \otimes c^{(t-1)} + (1 - f^{(t)}) \otimes \tilde{c}^{(t)}$$



GRU

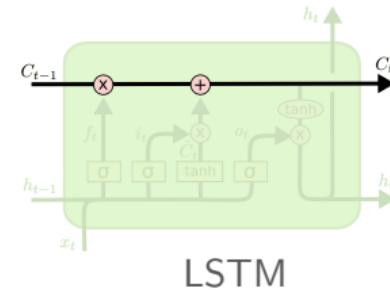
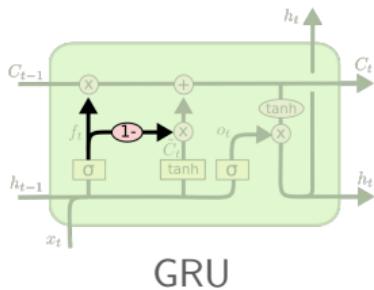


LSTM

Gated Recurrent Unit (GRU)

- ▶ The **GRU** cell is a **simplified version** of the LSTM cell.
- ▶ Instead of separately deciding **what to forget** and **what to add** to the new information to, it **makes those decisions together**.
 - It only **forgets** when it is going to input something in its place.
 - It only **inputs new values** to the state when it forgets something older.

$$c^{(t)} = f^{(t)} \otimes c^{(t-1)} + (1 - f^{(t)}) \otimes \tilde{c}^{(t)}$$





GRU in TensorFlow

► Use GRU

```
model = keras.models.Sequential([
    keras.layers.GRU(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.GRU(20),
    keras.layers.Dense(1)
])

model.compile(loss="mse", optimizer="adam", metrics=[last_time_step_mse])
history = model.fit(X_train, y_train, epochs=20)

model.evaluate(X_test, y_test, verbose=0)
```



Summary



Summary

- ▶ RNN
- ▶ Unfolding the network
- ▶ Three weights
- ▶ RNN design patterns
- ▶ Backpropagation through time
- ▶ LSTM and GRU



Reference

- ▶ Ian Goodfellow et al., Deep Learning (Ch. 10)
- ▶ Aurélien Géron, Hands-On Machine Learning (Ch. 15)
- ▶ Understanding LSTM Networks
<http://colah.github.io/posts/2015-08-Understanding-LSTMs>
- ▶ CS224d: Deep Learning for Natural Language Processing
<http://cs224d.stanford.edu>



Questions?



Transformers and Attention

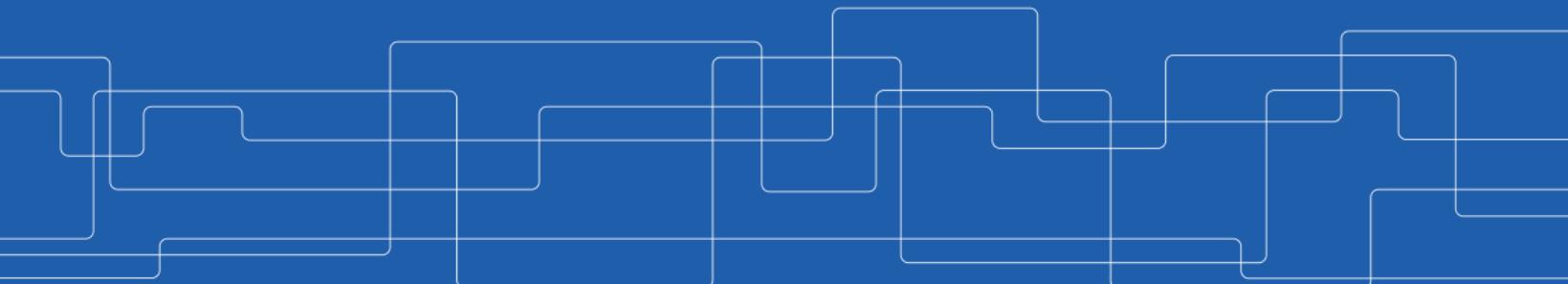
ID2223 Scalable Machine Learning and Deep Learning

Francisco J. Peña

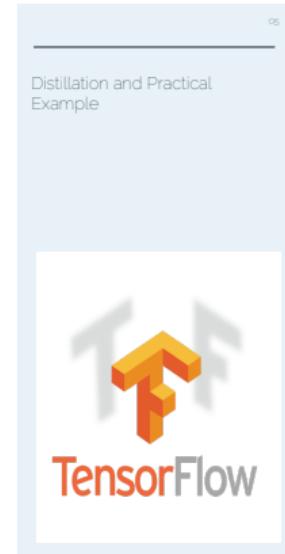
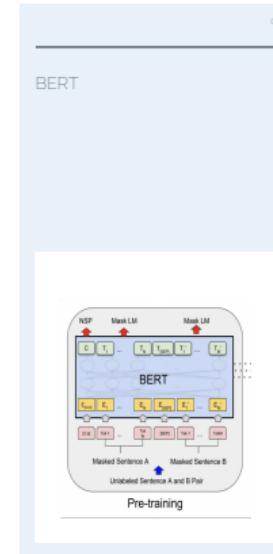
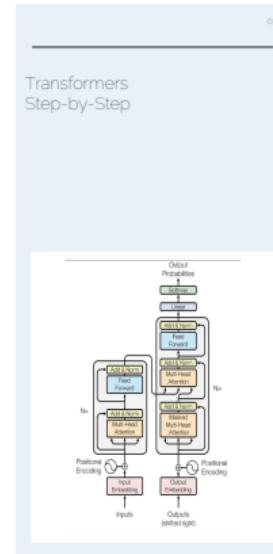
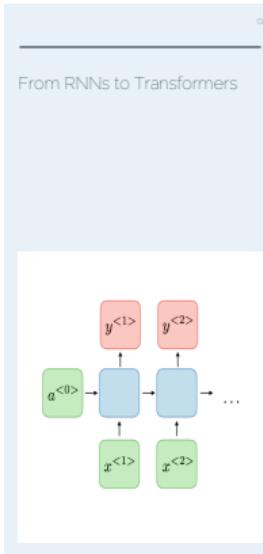
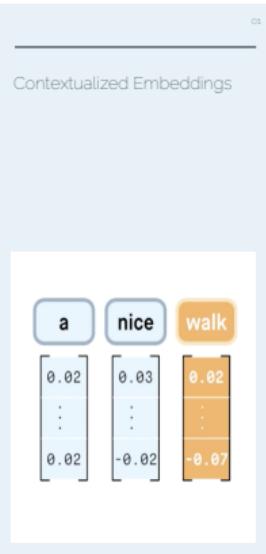
Postdoctoral Researcher, KTH

frape@kth.se

2021-12-02



Roadmap





Acknowledgements

Material based on:

- ▶ Christoffer Manning's [NLP Lectures at Stanford](#)
- ▶ [The Illustrated Transformer](#) by Jay Alammar
- ▶ [Slides](#) from Jacob
- ▶ [Self-attention Video](#) from Peltarion
- ▶ Slides from Karl Erliksson



Contextualized Embeddings

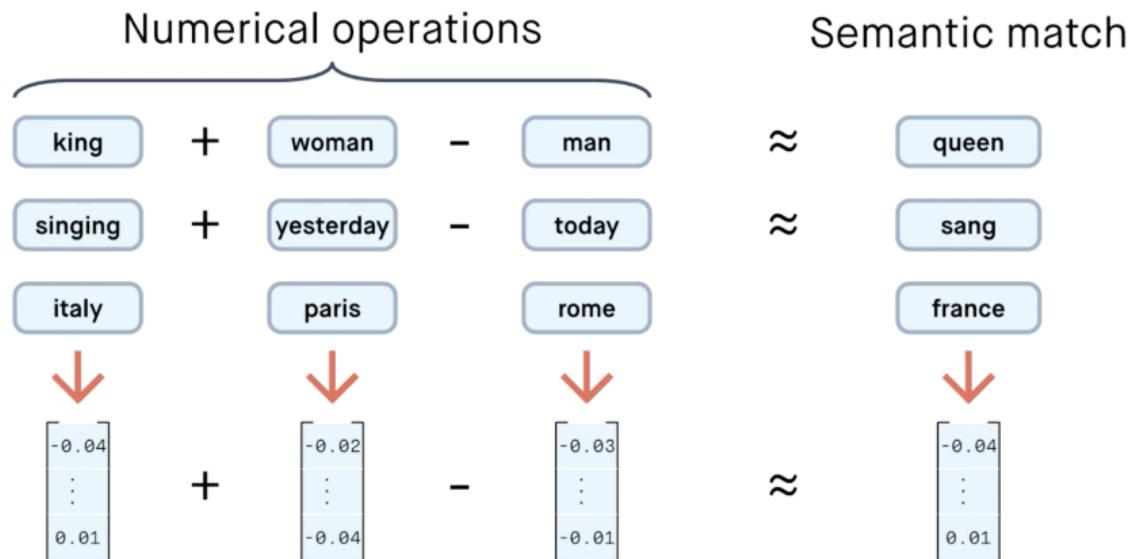
Background to Natural Language Processing (NLP)

- ▶ Word embeddings are the basis of NLP
- ▶ Popular embeddings like GloVe and Word2Vec are pre-trained on large text corpuses based on co-occurrence statistics
- ▶ “A word is characterized by the company it keeps” [Firth, 1957]

best	-	selling	music	artists
-0.11	0.01	-0.01	0.06	-0.02
0.01	0.07	-0.03	0.11	0.00
-0.17	-0.04	0.15	0.05	-0.05
:	:	:	:	:
0.13	-0.05	0.00	0.14	0.05
-0.13	-0.11	-0.07	-0.12	-0.12
-0.09	-0.25	0.05	-0.04	0.02

[Peltarion, 2020]

Word Embeddings



[Peltarion, 2020]

Word Embeddings

Problem: Word embeddings are **context-free**

a	nice	walk	by	the	river	bank
0.02	0.03	0.02	-0.00	-0.04	-0.01	-0.02
:	:	:	:	:	:	:
0.02	-0.02	-0.07	0.03	-0.03	-0.04	-0.03

walk	to	the	bank	and	get	cash
0.02	0.01	-0.04	-0.02	-0.02	-0.06	0.01
:	:	:	:	:	:	:
-0.07	0.02	-0.03	-0.03	0.02	0.04	-0.01

[Peltarion, 2020]

Word Embeddings

Problem: Word embeddings are **context-free**

a	nice	walk	by	the	river	bank
0.02	0.03	0.02	-0.00	-0.04	-0.01	-0.02
:	:	:	:	:	:	:
0.02	-0.02	-0.07	0.03	-0.03	-0.04	-0.03

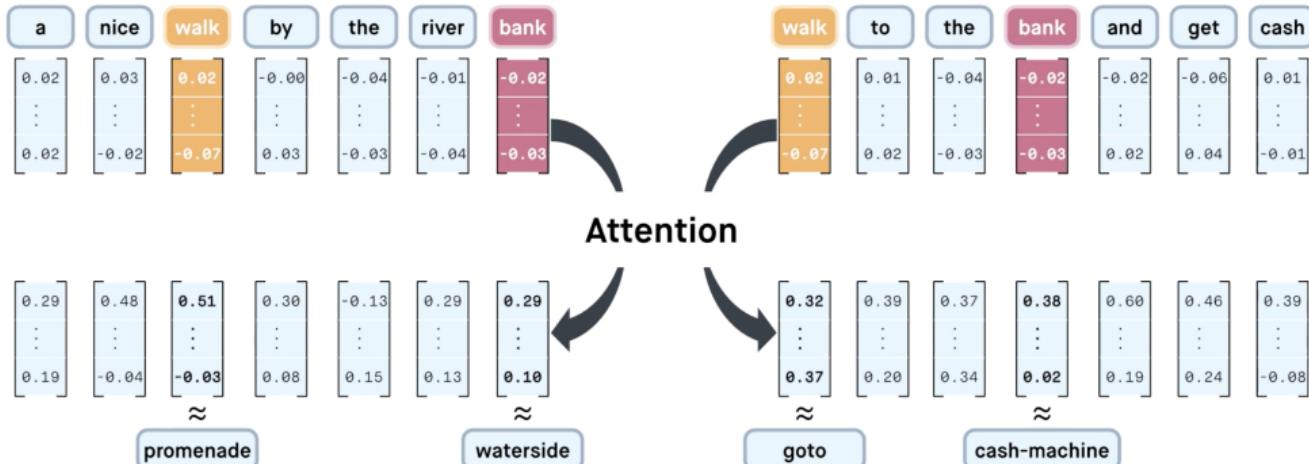
walk	to	the	bank	and	get	cash
0.02	0.01	-0.04	-0.02	-0.02	-0.06	0.01
:	:	:	:	:	:	:
-0.07	0.02	-0.03	-0.03	0.02	0.04	-0.01

[Peltarion, 2020]

Word Embeddings

Problem: Word embeddings are **context-free**

Solution: Create **contextualized** representation



[Peltarion, 2020]



From RNNs to Transformers

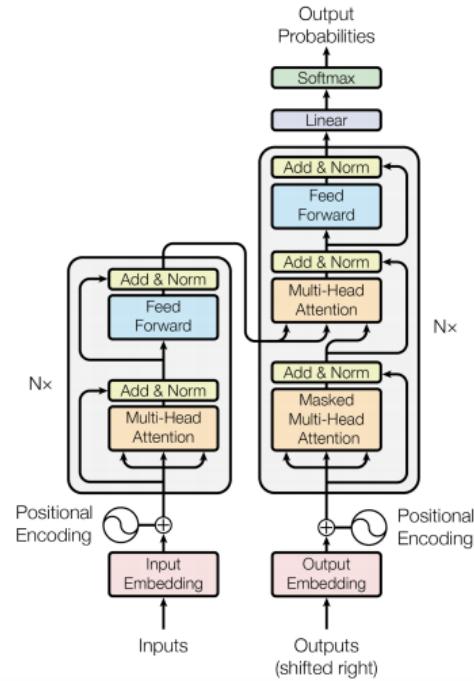


Problems with RNNs - Motivation for Transformers

- ▶ Sequential computations prevents parallelization
- ▶ Despite GRUs and LSTMs, RNNs still need attention mechanisms to deal with long range dependencies
- ▶ Attention gives us access to any state... Maybe we don't need the costly recursion?
- ▶ Then NLP can have deep models, solves our computer vision envy!

Attention is all you need! [Vaswani, 2017]

- ▶ Sequence-to-sequence model for Machine Translation
- ▶ Encoder-decoder architecture
- ▶ Multi-headed **self-attention**
 - Models context and no locality bias



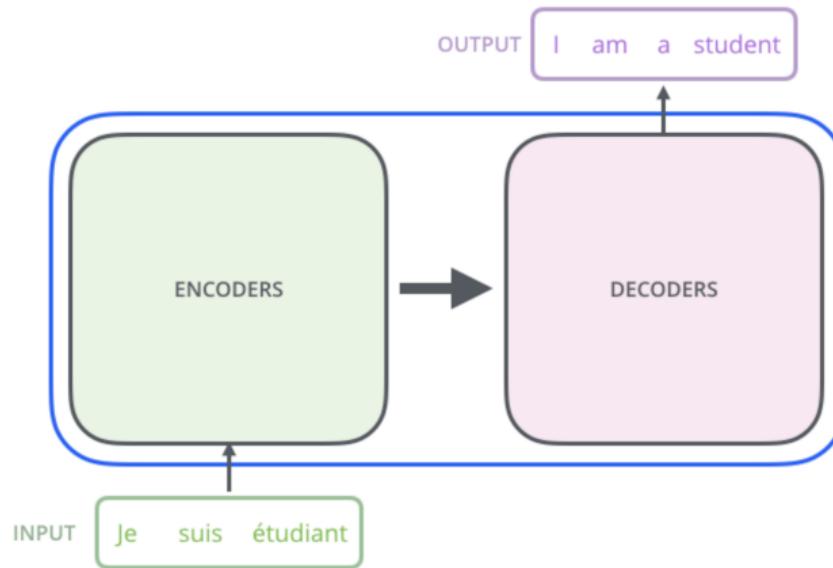
[Vaswani et al., 2017]



Transformers Step-by-Step



Understanding the Transformer: Step-by-Step

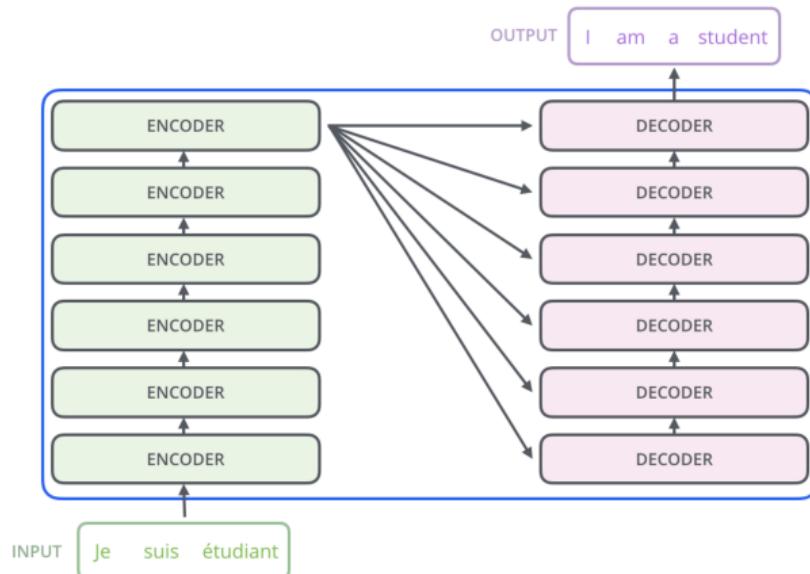


[Alammar, 2018]

Understanding the Transformer: Step-by-Step

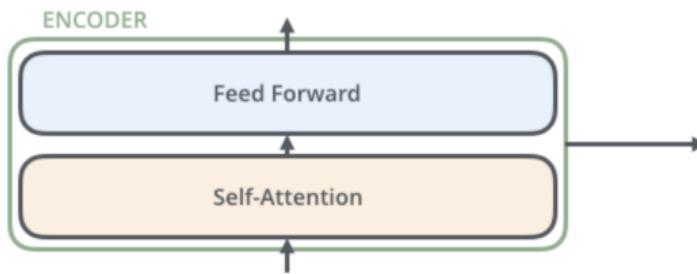
No recursion, instead
stacking encoder and
decoder blocks

- ▶ Originally: 6 layers
- ▶ BERT base: 12 layers
- ▶ BERT large: 24 layers
- ▶ GPT2-XL: 48 layers
- ▶ GPT3: 96 layers



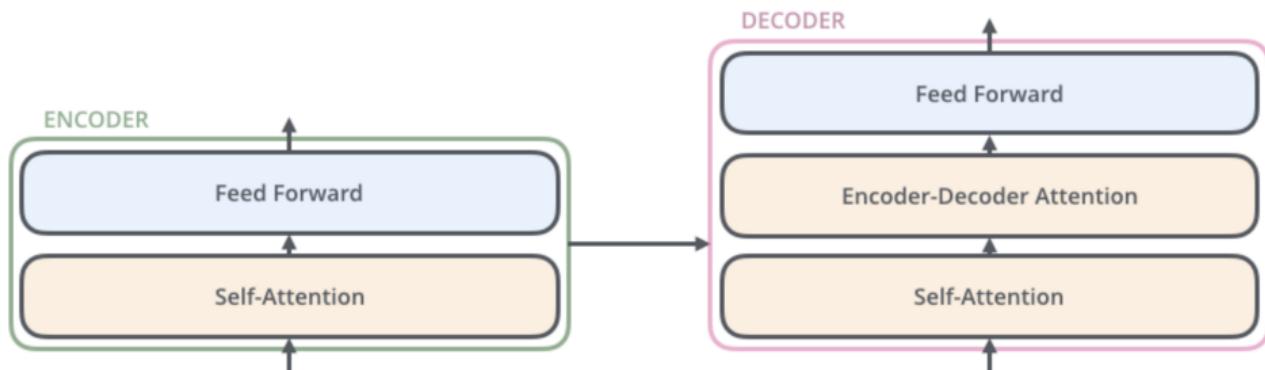
[Alammar, 2018]

The Encoder and Decoder Blocks



[Alammar, 2018]

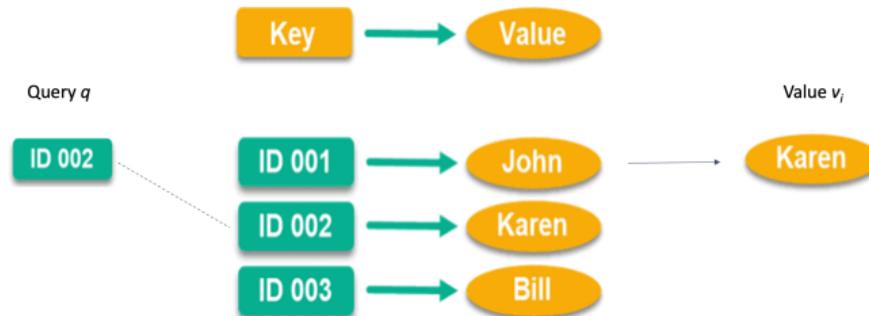
The Encoder Block



[Alammar, 2018]

Attention Preliminaries

Mimics the retrieval of a value v_i for a query q based on a key k_i in a database, but in a probabilistic fashion





Dot-Product Attention

- ▶ Queries, keys and values are vectors
- ▶ Output is a **weighted sum** of the values
- ▶ Weights are computed as the **scaled dot-product** (similarity) between the query and the keys

$$\text{Attention}(q, K, V) = \sum_i \text{Similarity}(q, k_i) \cdot v_i = \sum_i \frac{e^{q \cdot k_i / \sqrt{d_k}}}{\sum_j e^{q \cdot k_j / \sqrt{d_k}}} v_i$$

Output is a
row-vector

- ▶ Can stack multiple queries into a matrix Q

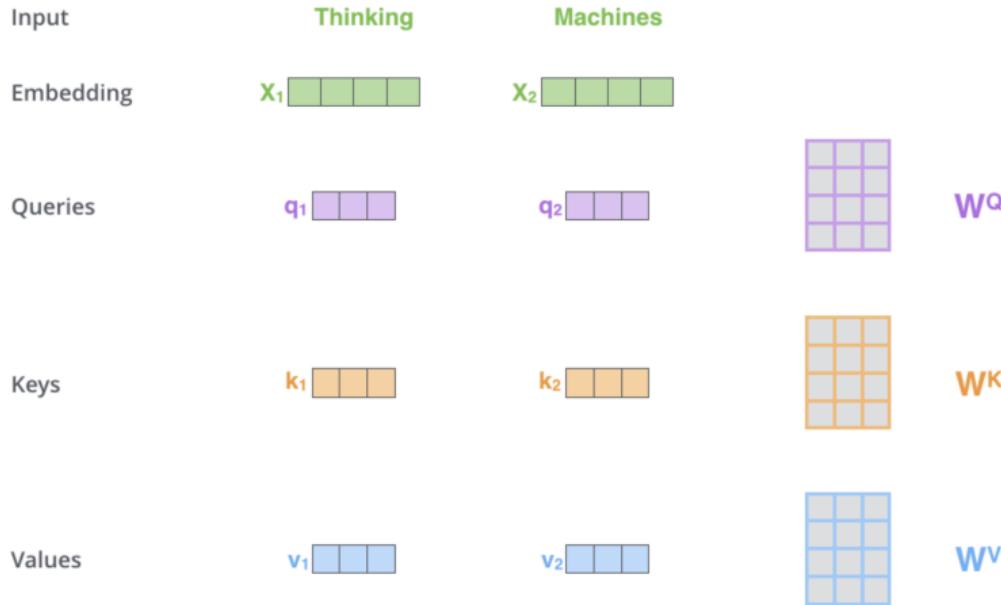
$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} \right) V$$

Output is again
a matrix

- ▶ Self-attention: Let the word embeddings be the queries, keys and values, i.e. **let the words select each other**

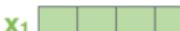
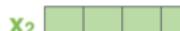
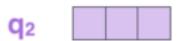


Self-Attention Mechanism



[Alammar, 2018]

Self-Attention Mechanism

Input	Thinking	Machines
Embedding	x_1 	x_2 
Queries	q_1 	q_2 
Keys	k_1 	k_2 
Values	v_1 	v_2 
Score	$q_1 \cdot k_1 = 112$	$q_1 \cdot k_2 = 96$
Divide by 8 ($\sqrt{d_k}$)	14	12
Softmax	0.88	0.12

[Alammar, 2018]

Self-Attention Mechanism in Matrix Notation

$$\mathbf{X} \times \mathbf{W}^Q = \mathbf{Q}$$

A diagram showing the multiplication of a green input matrix \mathbf{X} (3x3) by a purple weight matrix \mathbf{W}^Q (3x3) to produce a purple output matrix \mathbf{Q} (3x3).

$$\mathbf{X} \times \mathbf{W}^K = \mathbf{K}$$

A diagram showing the multiplication of a green input matrix \mathbf{X} (3x3) by an orange weight matrix \mathbf{W}^K (3x3) to produce an orange output matrix \mathbf{K} (3x3).

$$\mathbf{X} \times \mathbf{W}^V = \mathbf{V}$$

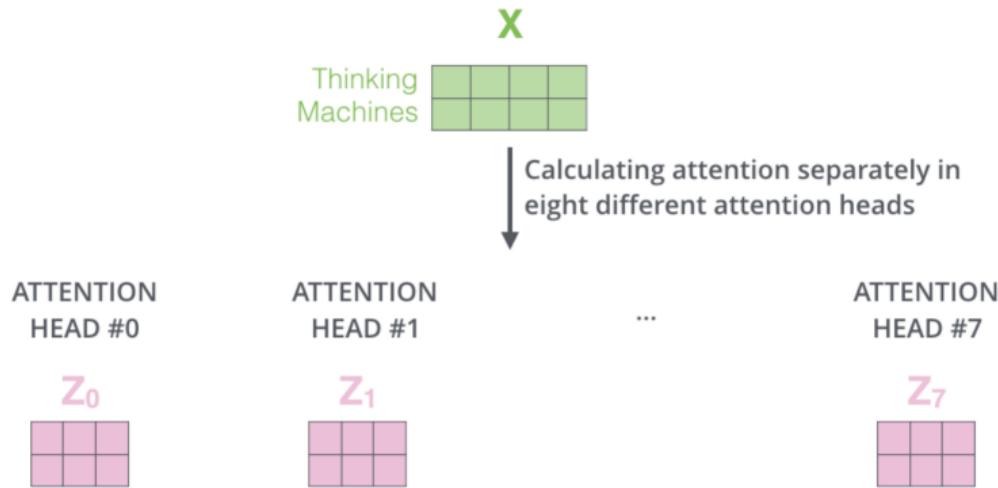
A diagram showing the multiplication of a green input matrix \mathbf{X} (3x3) by a light blue weight matrix \mathbf{W}^V (3x3) to produce a light blue output matrix \mathbf{V} (3x3).

$$\text{softmax}\left(\frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V} = \mathbf{Z}$$

The final computation of the self-attention mechanism. It shows the product of the query matrix \mathbf{Q} (3x3) and the transpose of the key matrix \mathbf{K}^T (3x3), divided by the square root of the dimension d_k , passed through a softmax function, and multiplied by the value matrix \mathbf{V} (3x3) to produce the output matrix \mathbf{Z} (3x3).

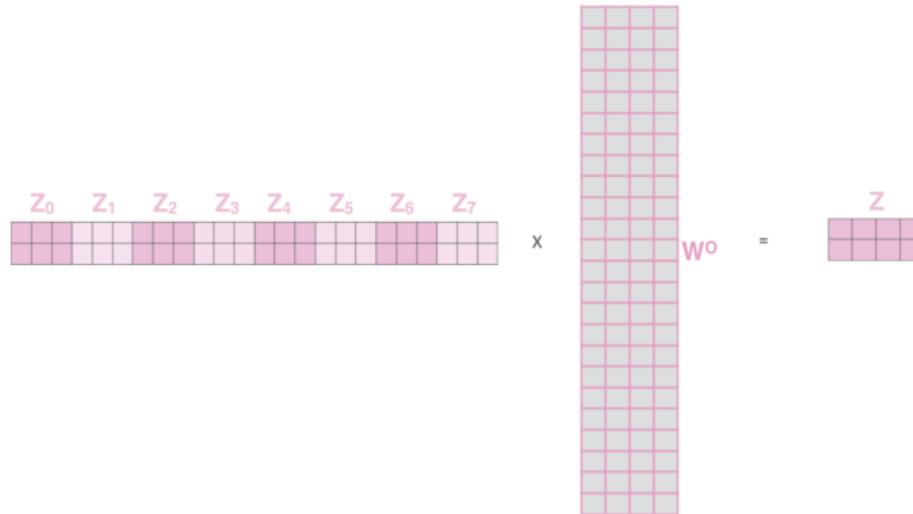
[Alammar, 2018]

Multi-Headed Self-Attention



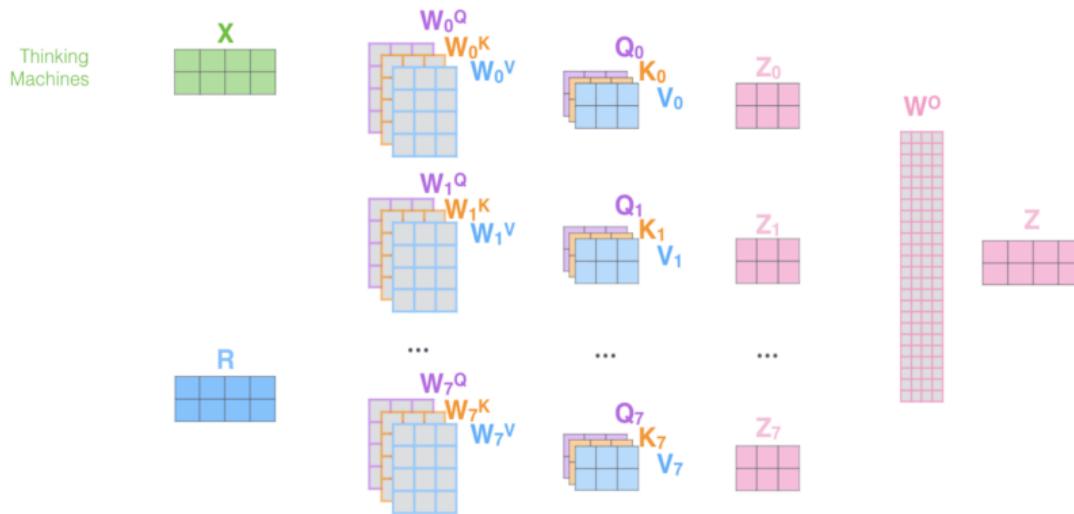
[Alammar, 2018]

Multi-Headed Self-Attention



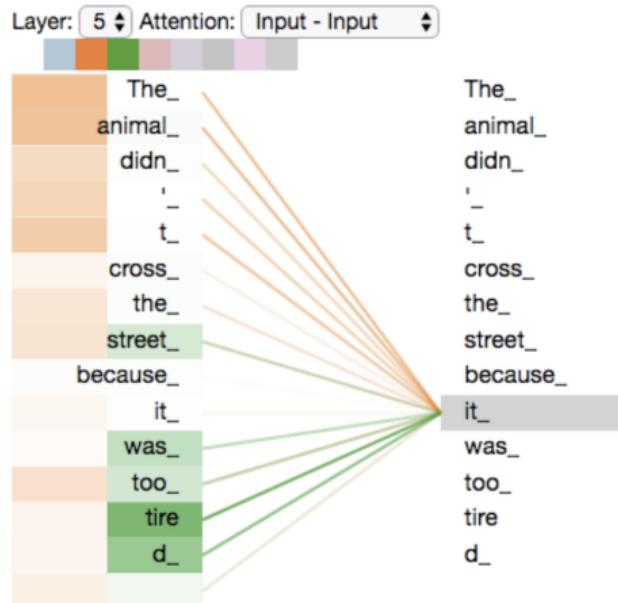
[Alammar, 2018]

Self-Attention: Putting It All Together



[Alammar, 2018]

Attention Visualized

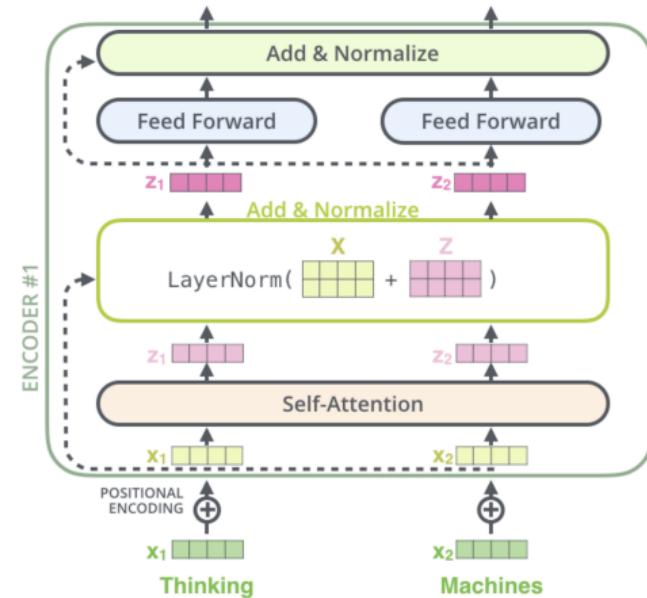


[Alammar, 2018]

The Full Encoder Block

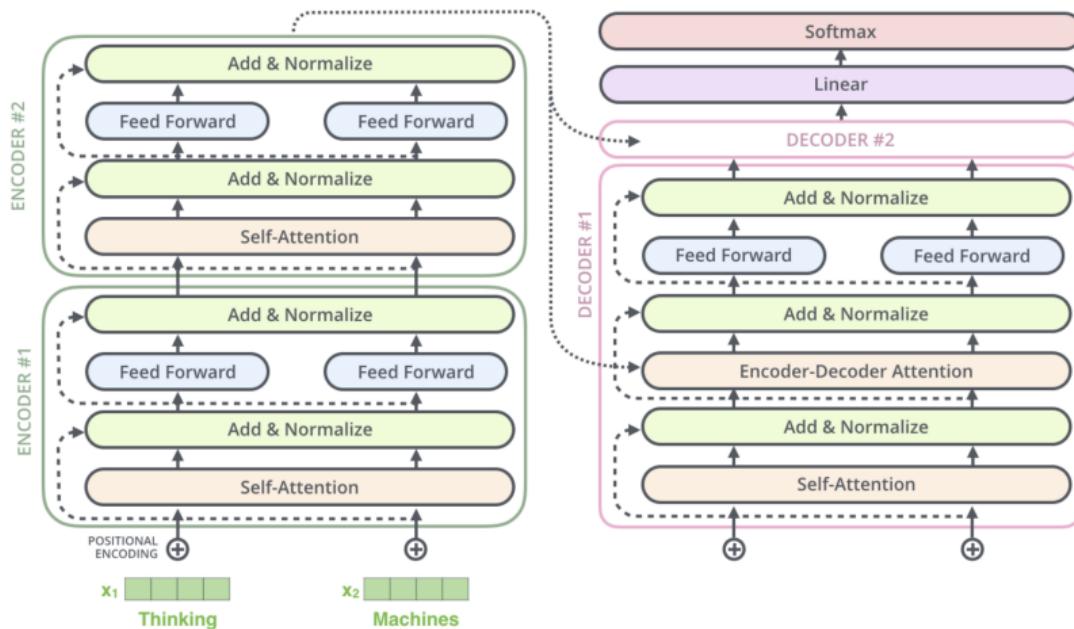
Encoder block consisting of:

- ▶ Multi-headed self-attention
- ▶ Feedforward NN (FC 2 layers)
- ▶ Skip connections
- ▶ Layer normalization - Similar to batch normalization but computed over features (words/tokens) for a single sample



[Alammar, 2018]

Encoder-Decoder Architecture - Small Example



[Alammar, 2018]



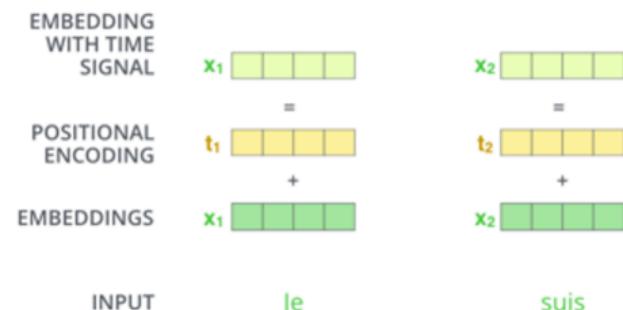
Positional Encodings

Encoder block consisting of:

- ▶ Attention mechanism has no locality bias - **no notion of word order**
 - ▶ Add **positional encodings** to input embeddings to let model learn relative positioning

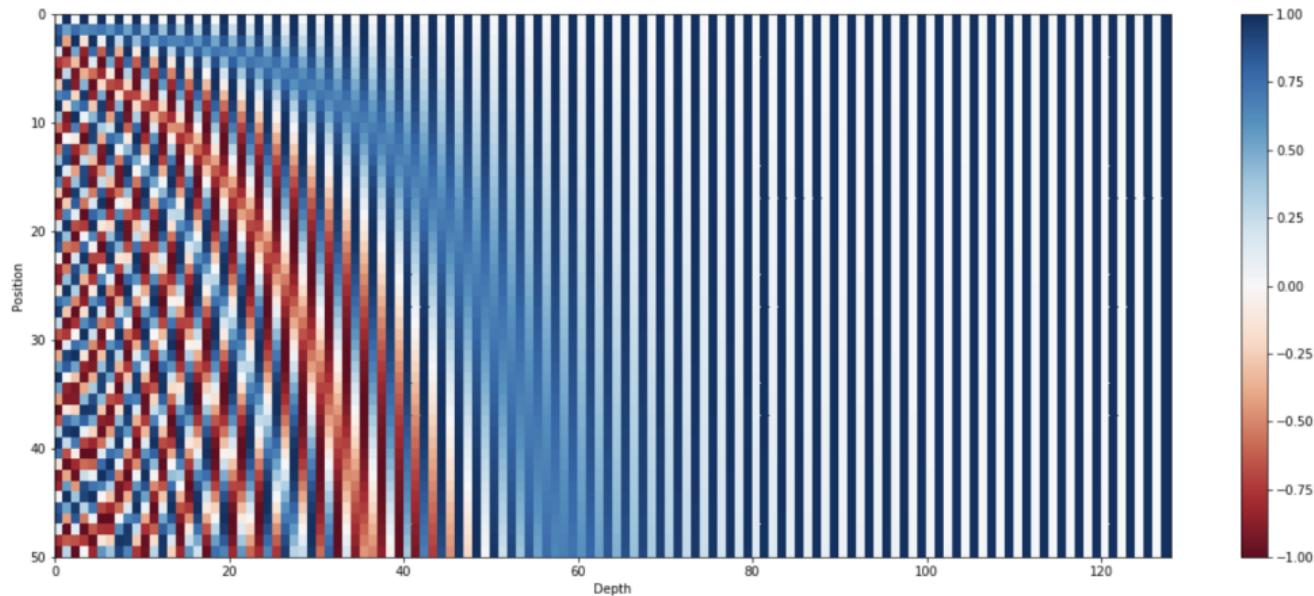
$$\text{PE}(\text{pos}, 2i) = \sin\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right)$$

$$\text{PE}(\text{pos}, 2i + 1) = \cos\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right)$$



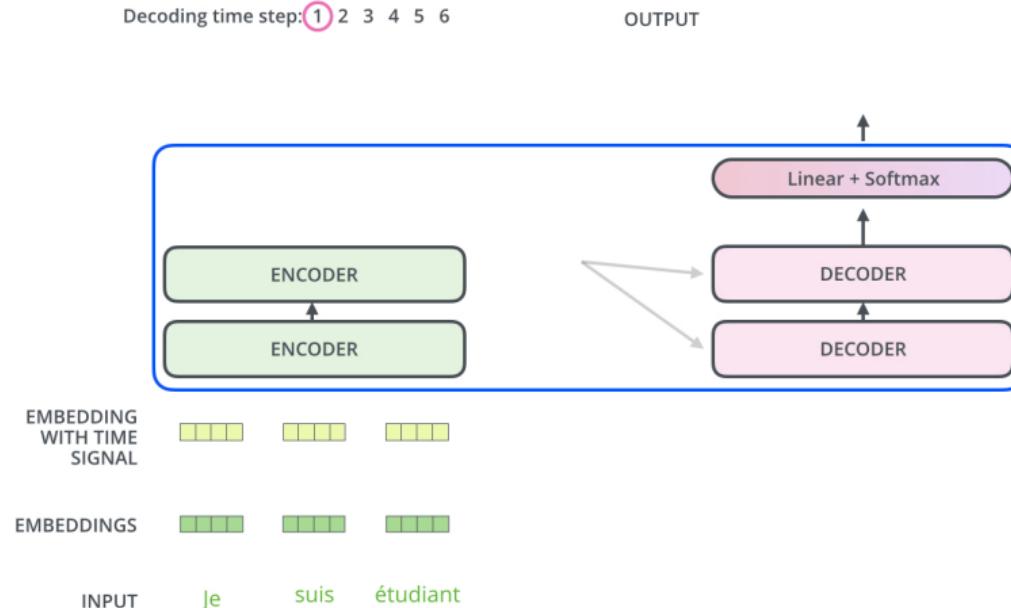
[Alammar, 2018]

Positional Encodings



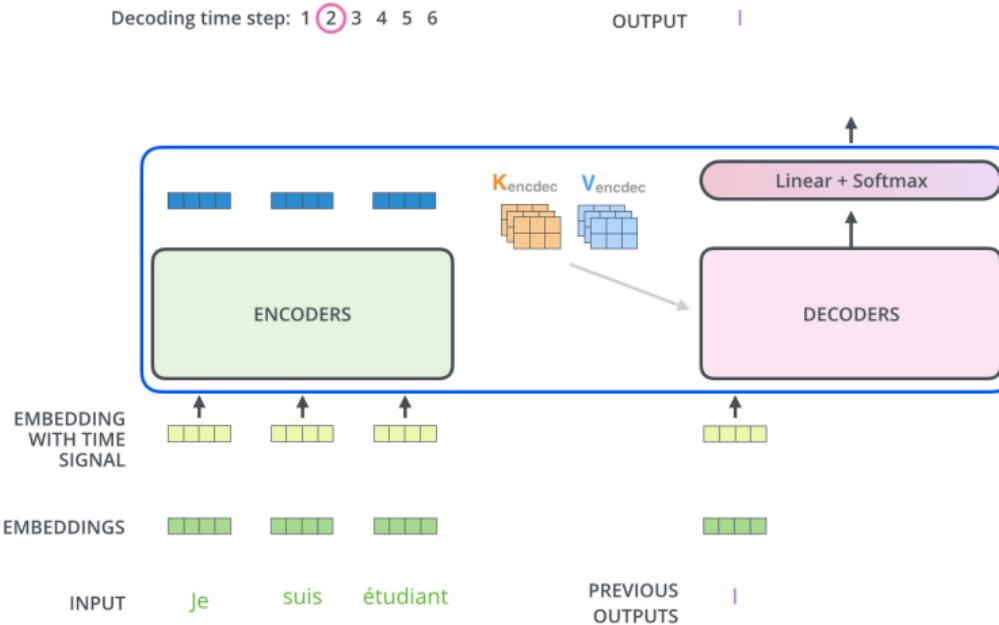
[Kazemnejad, 2019]

Let's start the encoding!



[Alammar, 2018]

Decoding procedure

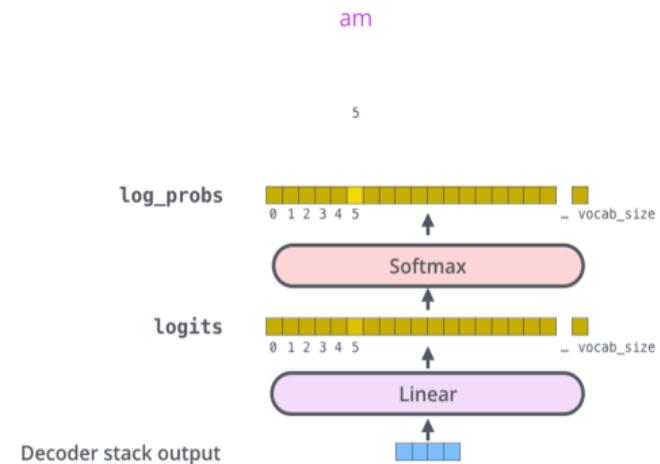


[Alammar, 2018]

Producing the output text

Encoder block consisting of:

- ▶ The output from the decoder is passed through a final fully connected **linear layer** with a **softmax** activation function
- ▶ Produces a probability distribution over the pre-defined vocabulary of output words (tokens)
- ▶ **Greedy decoding** picks the word with the highest probability at each time step



[Alammar, 2018]

Training Objective

Target Model Outputs

Output Vocabulary: a am I thanks student <eos>

position #1	0.0	0.0	1.0	0.0	0.0	0.0
-------------	-----	-----	------------	-----	-----	-----

position #2	0.0	1.0	0.0	0.0	0.0	0.0
-------------	-----	------------	-----	-----	-----	-----

position #3	1.0	0.0	0.0	0.0	0.0	0.0
-------------	------------	-----	-----	-----	-----	-----

position #4	0.0	0.0	0.0	0.0	1.0	0.0
-------------	-----	-----	-----	-----	------------	-----

position #5	0.0	0.0	0.0	0.0	0.0	1.0
-------------	-----	-----	-----	-----	-----	------------

a am I thanks student <eos>

Trained Model Outputs

Output Vocabulary: a am I thanks student <eos>

position #1	0.01	0.02	0.93	0.01	0.03	0.01
-------------	------	------	-------------	------	------	------

position #2	0.01	0.8	0.1	0.05	0.01	0.03
-------------	------	------------	-----	------	------	------

position #3	0.99	0.001	0.001	0.001	0.002	0.001
-------------	-------------	-------	-------	-------	-------	-------

position #4	0.001	0.002	0.001	0.02	0.94	0.01
-------------	-------	-------	-------	------	-------------	------

position #5	0.01	0.01	0.001	0.001	0.001	0.98
-------------	------	------	-------	-------	-------	-------------

a am I thanks student <eos>



[Alammar, 2018]

Complexity Comparison

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$

[Vaswani et al., 2017]



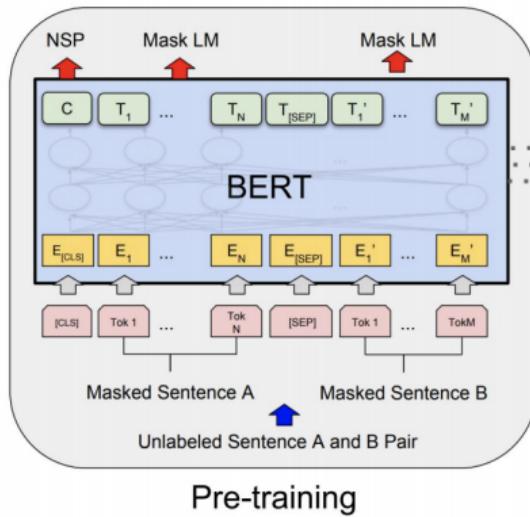
BERT

Bidirectional Encoder Representations from Transformers

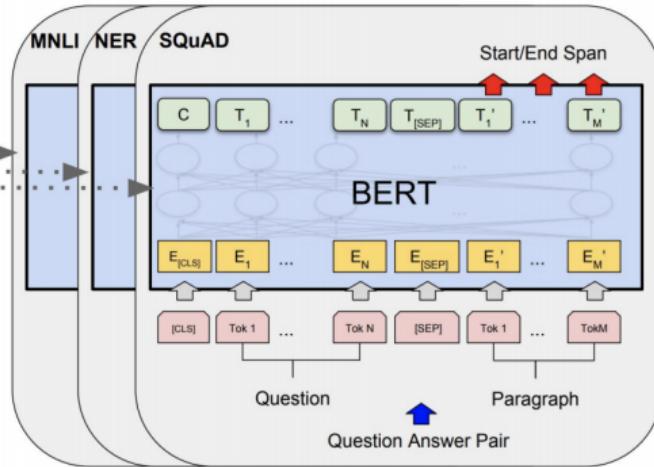
- ▶ Self-supervised pre-training of Transformers encoder for language understanding
- ▶ Fine-tuning for specific downstream task



BERT Training Procedure



Pre-training



Fine-Tuning

[Devlin et al., 2018]



BERT Training Objectives

Masked Language Modelling

the man went to the [MASK] to buy a [MASK] of milk

↑ ↑
store gallon

Next Sentence prediction

Sentence A = The man went to the store.

Sentence B = He bought a gallon of milk.

Label = IsNextSentence

Sentence A = The man went to the store.

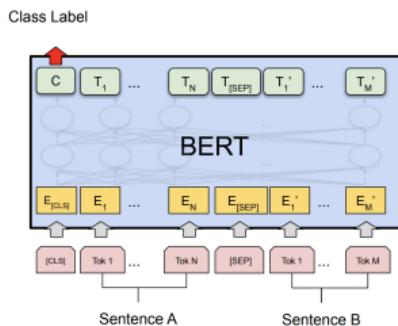
Sentence B = Penguins are flightless.

Label = NotNextSentence

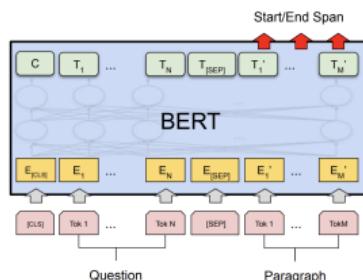
[Devlin et al., 2018]

BERT Fine-Tuning Examples

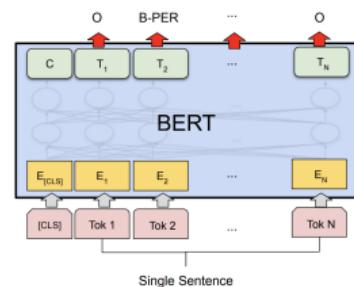
Sentence Classification



Question Answering



Named Entity Recognition



[Devlin et al., 2018]



Exploring the Limits of Transfer Learning (T5)

- ▶ Scaling up **models size** and amount of **training data** helps a lot
- ▶ Best model is 11B (!! parameters
- ▶ Exact **pre-training objective** (MLM, NSP, corruption) doesn't matter too much
- ▶ SuperGLUE benchmark:

Rank	Name	Model	URL	Score	BoolQ	CB	COPA	MultiRC	ReCoRD	RTE	WiC	WSC	AX-b	AX-g
1	SuperGLUE Human Baselines	SuperGLUE Human Baselines		89.8	89.0	95.8/98.9	100.0	81.8/51.9	91.7/91.3	93.6	80.0	100.0	76.6	99.3/99.7
+	2 T5 Team - Google	T5		89.3	91.2	93.9/96.8	94.8	88.1/63.3	94.1/93.4	92.5	76.9	93.8	65.6	92.7/91.9
+	3 Huawei Noah's Ark Lab	NEZHA-Plus		86.7	87.8	94.4/96.0	93.6	84.6/55.1	90.1/89.6	89.1	74.6	93.2	58.0	87.1/74.4
+	4 Alibaba PAI&ICBU	PAI Albert		86.1	88.1	92.4/96.4	91.8	84.6/54.7	89.0/88.3	88.8	74.1	93.2	75.6	98.3/99.2
+	5 Tencent Jarvis Lab	RoBERTa (ensemble)		85.9	88.2	92.5/95.6	90.8	84.4/53.4	91.5/91.0	87.9	74.1	91.8	57.6	89.3/75.6
6	Zhuiyi Technology	RoBERTa-mtl-adv		85.7	87.1	92.4/95.6	91.2	85.1/54.3	91.7/91.3	88.1	72.1	91.8	58.5	91.0/78.1
7	Facebook AI	RoBERTa		84.6	87.1	90.5/95.2	90.6	84.4/52.5	90.6/90.0	88.2	69.9	89.0	57.9	91.0/78.1

[Raffel et al., 2019]



Practical Examples



BERT in low-latency production settings

GOOGLE \ TECH \ ARTIFICIAL INTELLIGENCE

Google is improving 10 percent of searches by understanding language context

Say hello to BERT

By Dieter Bohn | @backlon | Oct 25, 2019, 3:01am EDT

Bing says it has been applying BERT since April

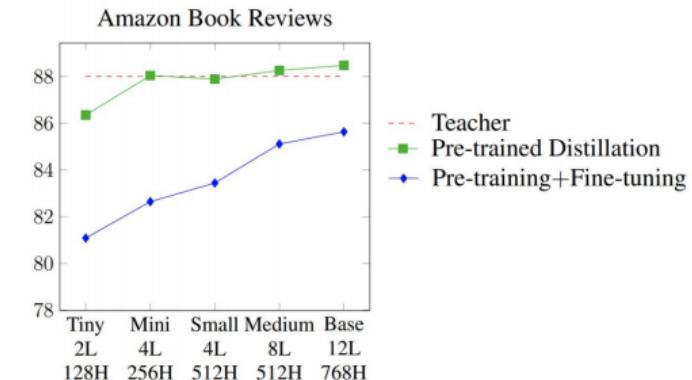
The natural language processing capabilities are now applied to all Bing queries globally.

[George Nguyen](#) on November 19, 2019 at 1:38 pm

[Devlin, 2020]

Distillation

- ▶ Modern pre-trained language models are **huge** and very **computationally expensive**
- ▶ How are these companies applying them to low-latency applications?
- ▶ Distillation!
 - Train SOTA **teacher model** (pre-training + fine-tuning)
 - Train smaller **student model** that **mimics** the teacher's output on a large dataset on unlabeled data
- ▶ Why does it work so well?



[Devlin, 2020] [Turc, 2020]



Transformers in TensorFlow using HuggingFace 😊

- ▶ The [HuggingFace Library](#) contains a majority of the recent pre-trained State-of-the-art NLP models, as well as over 4 000 community uploaded models
- ▶ Works with both [TensorFlow](#) and [PyTorch](#)



HUGGING FACE

[Back to home](#)

All Models and checkpoints

Also check out our list of [Community contributors](#) 🎉 and [Organizations](#) 🌐.

Search models...	Tags: All	Sort: Most downloads ▾
bert-base-uncased ★		
deepset/bert-large-uncased-whole-word-masking-squad2		
distilbert-base-uncased ★		
dccuchile/bert-base-spanish-wwm-cased ★		
microsoft/xprophetnet-large-wiki100-cased-xglue-ntg ★		
deepset/roberta-base-squad2 ★		
jplu/tf-xlm-roberta-base ★		
cl-tohoku/bert-base-japanese-whole-word-masking		
distilroberta-base ★		
bert-base-cased ★		
xlm-roberta-base ★		



Transformers in TensorFlow using HuggingFace 😊

```
from transformers import BertTokenizerFast, TFBertForSequenceClassification
from datasets import load_dataset
import tensorflow as tf

dataset = load_dataset("imdb").shuffle()
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

train_encodings = tokenizer(dataset['train']['text'], truncation=True, padding=True)
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings), dataset['train']['label']))
val_dataset = ... // Analogously

optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
model.compile(optimizer=optimizer, loss=model.compute_loss)
model.fit(train_dataset.batch(16), epochs=3, batch_size=16)

model.evaluate(val_dataset.batch(16), verbose=0)
```



Transformers in TensorFlow using HuggingFace 😊

```
from transformers import BertTokenizerFast, TFBertForSequenceClassification
from datasets import load_dataset
import tensorflow as tf

dataset = load_dataset("imdb").shuffle()
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

train_encodings = tokenizer(dataset['train']['text'], truncation=True, padding=True)
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings), dataset['train']['label']))
val_dataset = ... // Analogously

optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
model.compile(optimizer=optimizer, loss=model.compute_loss)
model.fit(train_dataset.batch(16), epochs=3, batch_size=16)

model.evaluate(val_dataset.batch(16), verbose=0)
```



Transformers in TensorFlow using HuggingFace 😊

```
from transformers import BertTokenizerFast, TFBertForSequenceClassification
from datasets import load_dataset
import tensorflow as tf

dataset = load_dataset("imdb").shuffle()
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

train_encodings = tokenizer(dataset['train']['text'], truncation=True, padding=True)
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings), dataset['train']['label']))
val_dataset = ... // Analogously

optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
model.compile(optimizer=optimizer, loss=model.compute_loss)
model.fit(train_dataset.batch(16), epochs=3, batch_size=16)

model.evaluate(val_dataset.batch(16), verbose=0)
```



Transformers in TensorFlow using HuggingFace 😊

```
from transformers import BertTokenizerFast, TFBertForSequenceClassification
from datasets import load_dataset
import tensorflow as tf

dataset = load_dataset("imdb").shuffle()
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

train_encodings = tokenizer(dataset['train']['text'], truncation=True, padding=True)
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings), dataset['train']['label']))
val_dataset = ... // Analogously

optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
model.compile(optimizer=optimizer, loss=model.compute_loss)
model.fit(train_dataset.batch(16), epochs=3, batch_size=16)

model.evaluate(val_dataset.batch(16), verbose=0)
```



Wrap Up

Summary

- ▶ Transformers have blown other architectures out of the water for NLP
- ▶ Get rid of recurrence and rely on **self-attention**
- ▶ NLP pre-training using **Masked Language Modelling**
- ▶ Most recent improvements using **larger models** and **more data**
- ▶ **Distillation** can make model serving and inference more tractable





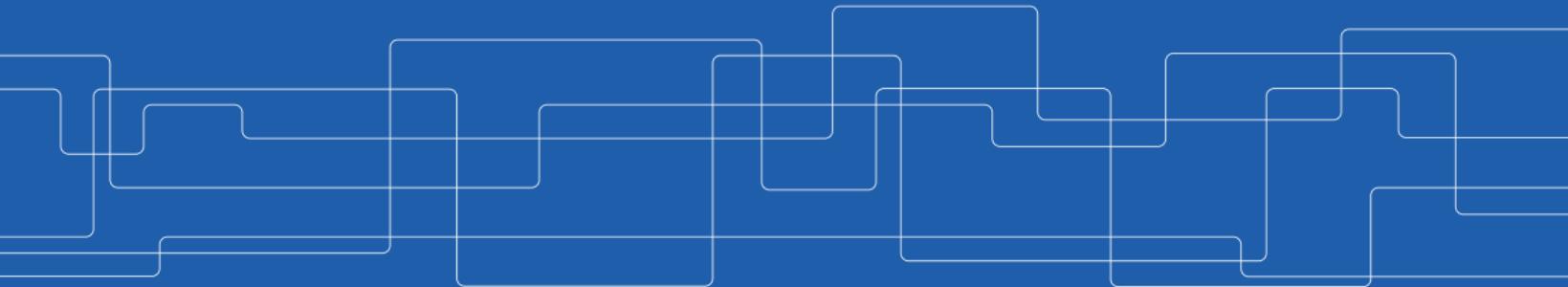
Thanks!
Questions?

Francisco J. Peña
Postdoctoral Researcher, KTH
frape@kth.se



Distributed Deep Learning

Amir H. Payberah
payberah@kth.se
2021-12-08



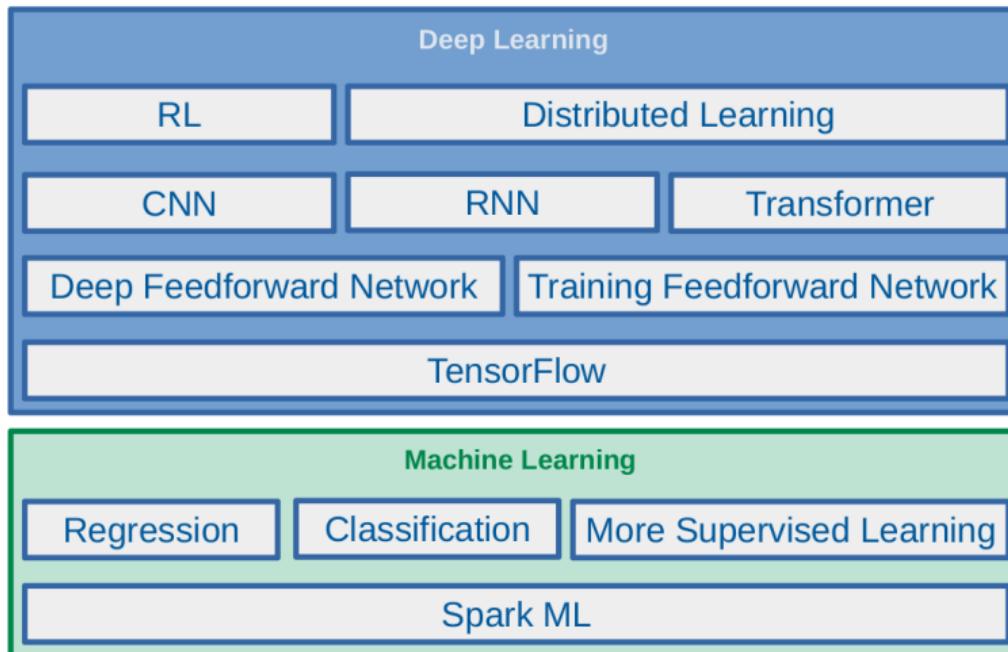


The Course Web Page

<https://id2223kth.github.io>
<https://tinyurl.com/6s5jy46a>

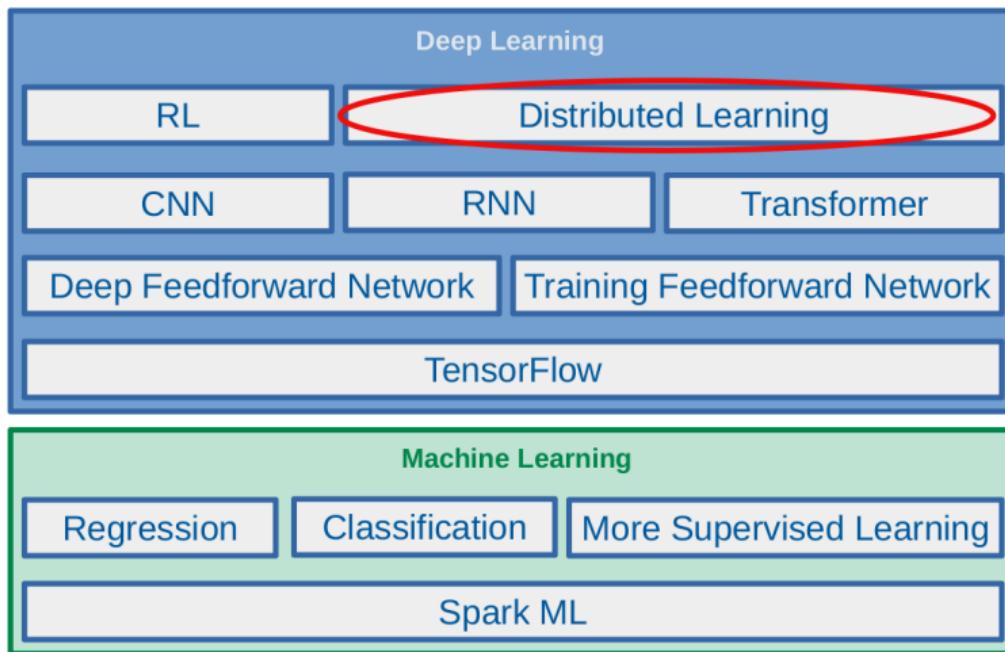


Where Are We?





Where Are We?

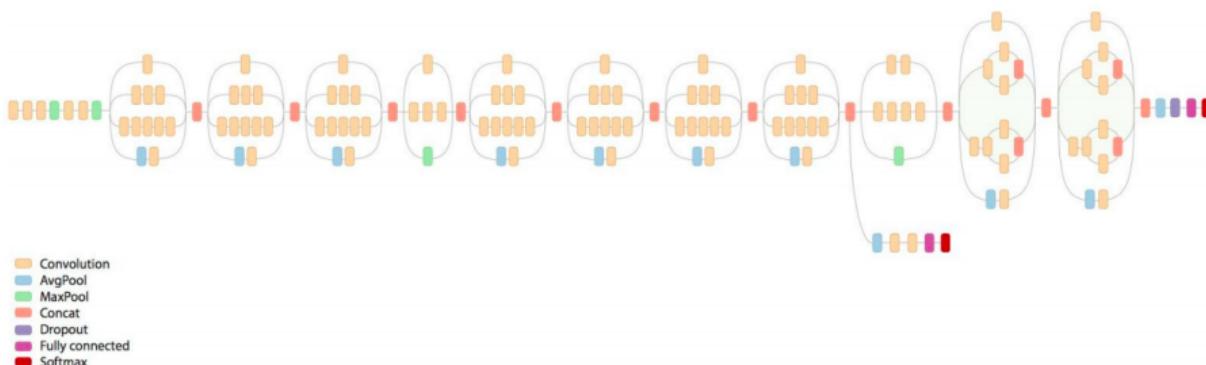




What is the problem?

Training Deep Neural Networks

- ▶ Computationally intensive
- ▶ Time consuming



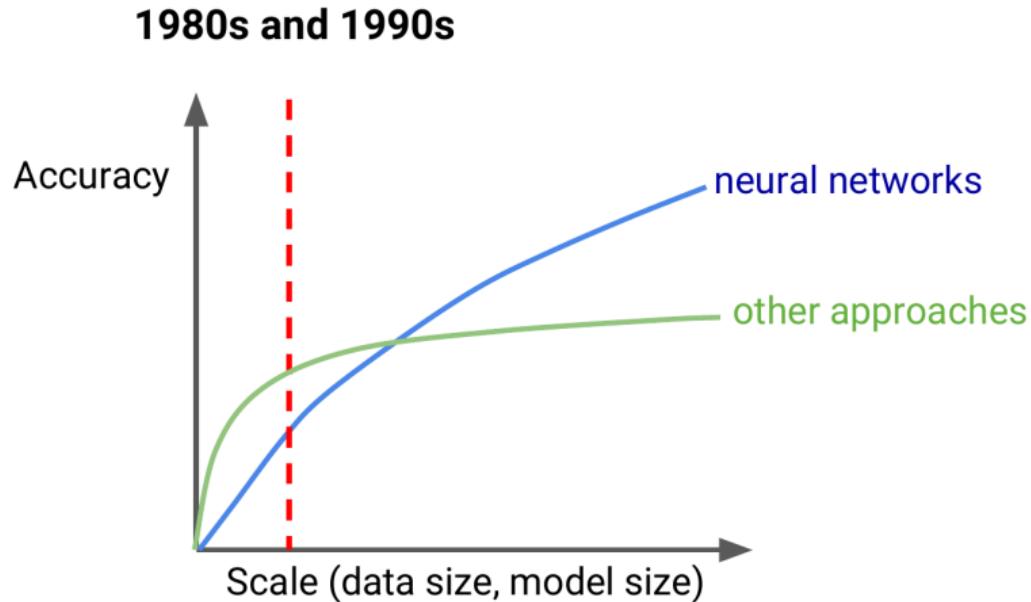
[<https://cloud.google.com/tpu/docs/images/inceptionv3onc--oview.png>]

Why?

- ▶ Massive amount of training dataset
- ▶ Large number of parameters

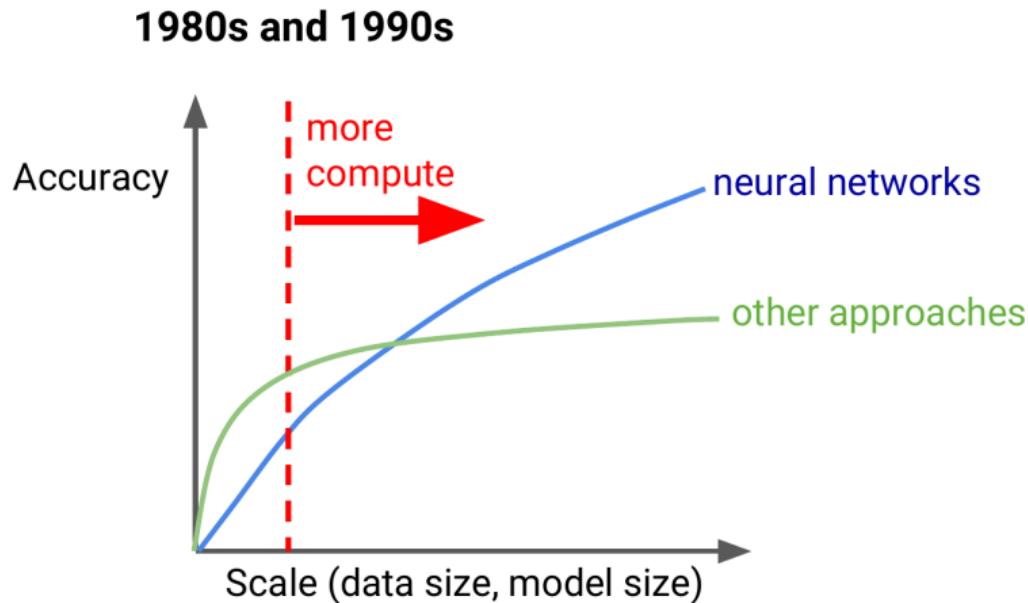


Accuracy vs. Data/Model Size



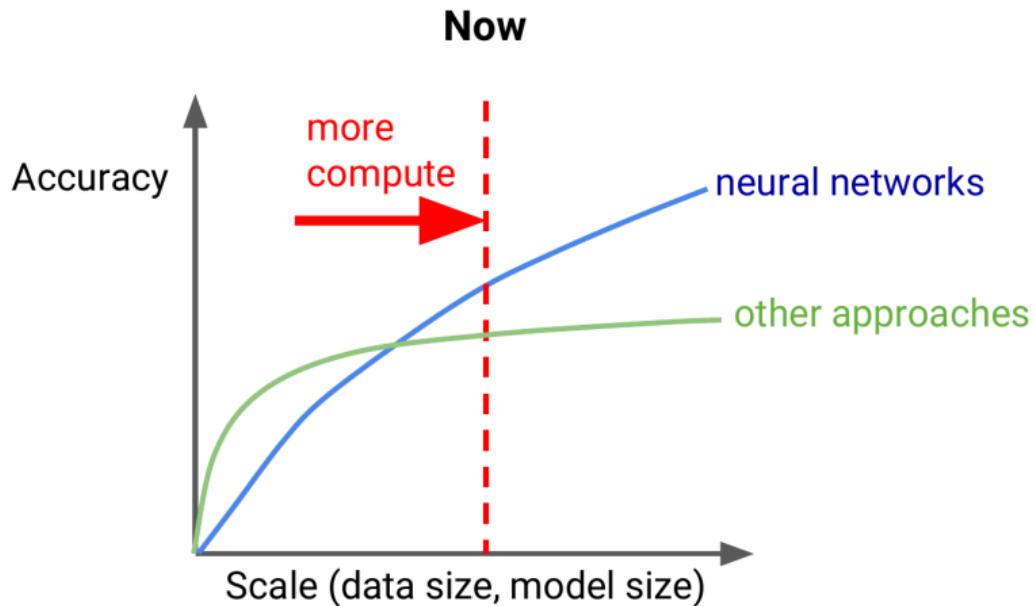
[Jeff Dean at AI Frontiers: Trends and Developments in Deep Learning Research]

Accuracy vs. Data/Model Size



[Jeff Dean at AI Frontiers: Trends and Developments in Deep Learning Research]

Accuracy vs. Data/Model Size



[Jeff Dean at AI Frontiers: Trends and Developments in Deep Learning Research]

Scalability



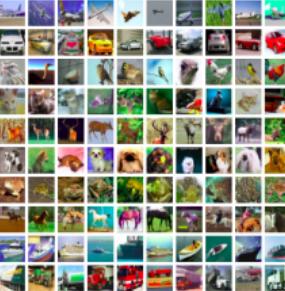


Fundamentals of Machine Learning



Training Dataset

- ▶ E.g., tabular data, image, text, etc.



Entities

Society and Culture Science and Mathematics Health Education and Reference Computers and Internet Sports
Business and Finance Entertainment and Music Family and Relationships Politics and Government

does anyone here play habbohotel and want 2 be friends? Answer: No on the first part and maybe on the second part. I got to think it over first.

Family and Relationships

Date	Cost	Actions	Offsite conversions	Impressions	Clicks
2017-04-04	29.44	461	4	5655	477
2017-04-03	74.08	1331	16	18170	1340
2017-04-02	76.09	1349	12	16877	1357
2017-04-01	76.79	1382	8	19757	1378
2017-03-31	77.28	1141	21	18598	1116
2017-03-30	68.62	1065	18	14847	1046
2017-03-29	64.9	1111	25	13994	1094
2017-03-28	65.12	1137	12	15952	1145
2017-03-27	66.98	1185	7	17970	1190
2017-03-26	64.94	1118	5	14410	1116
2017-03-25	66.3	1208	6	15123	1204
2017-03-24	67.38	1143		15296	1159
2017-03-23	65.59	1147	13	14972	1143
2017-03-22	68.19	1129	4	17959	1116
2017-03-21	64.78	1081		25810	1059

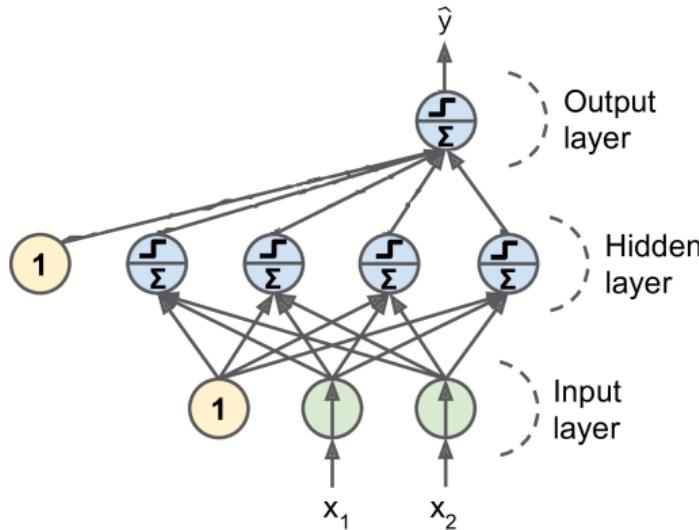


Model

- ▶ E.g., linear models, neural networks, etc.

Model

- ▶ E.g., linear models, neural networks, etc.
- ▶ $\hat{y} = f_w(x)$





Loss function

- ▶ How good \hat{y} is able to predict the expected outcome y .

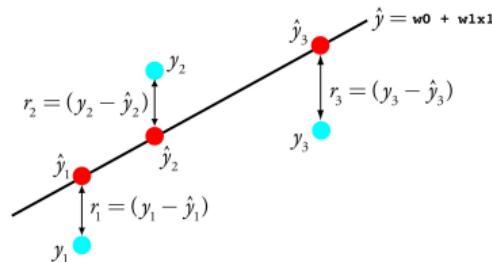


Loss function

- ▶ How good \hat{y} is able to predict the expected outcome y .
- ▶ $J(w) = \sum_{i=1}^m l(y_i, \hat{y}_i)$

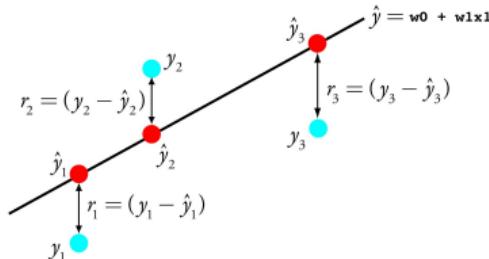
Loss function

- ▶ How good \hat{y} is able to predict the expected outcome y .
- ▶ $J(w) = \sum_{i=1}^m l(y_i, \hat{y}_i)$



Loss function

- ▶ How good \hat{y} is able to predict the expected outcome y .
- ▶ $J(w) = \sum_{i=1}^m l(y_i, \hat{y}_i)$



- ▶ E.g., $J(w) = \frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2$



Objective

- ▶ Minimize the loss function

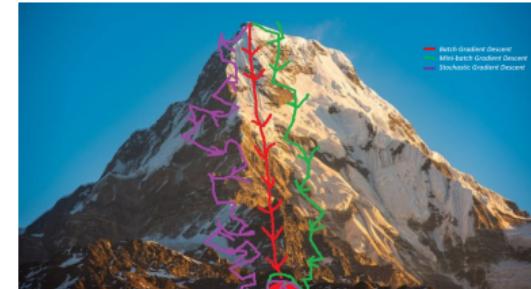


Objective

- ▶ Minimize the loss function
- ▶ $\arg \min_w J(w)$
- ▶ $J(w) = \sum_{i=1}^m l(y_i, \hat{y}_i)$

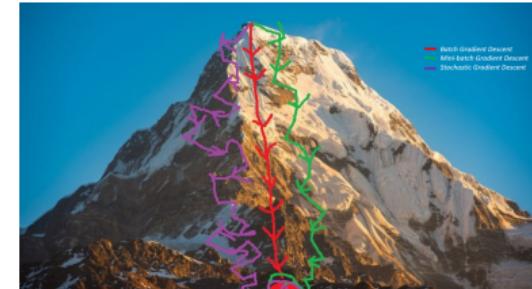
Training

► $J(w) = \sum_{i=1}^m l(y_i, \hat{y}_i)$



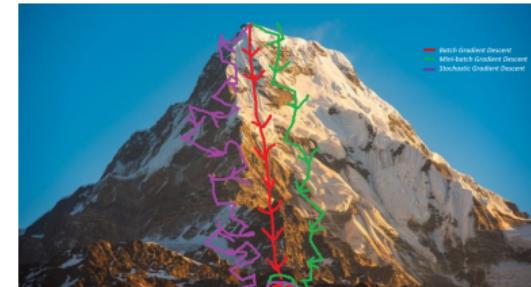
Training

- ▶ $J(w) = \sum_{i=1}^m l(y_i, \hat{y}_i)$
- ▶ Gradient descent, i.e., $w := w - \eta \nabla J(w)$



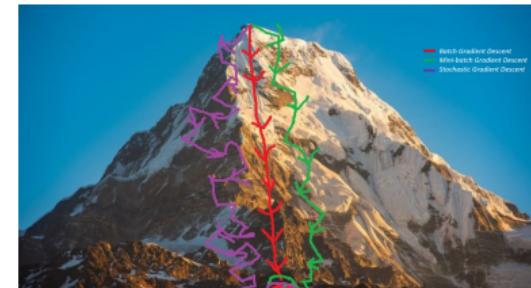
Training

- ▶ $J(w) = \sum_{i=1}^m l(y_i, \hat{y}_i)$
- ▶ Gradient descent, i.e., $w := w - \eta \nabla J(w)$
- ▶ Stochastic gradient descent, i.e., $w := w - \eta \tilde{g} J(w)$
 - \tilde{g} : gradient at a randomly chosen point.



Training

- ▶ $J(w) = \sum_{i=1}^m l(y_i, \hat{y}_i)$
- ▶ Gradient descent, i.e., $w := w - \eta \nabla J(w)$
- ▶ Stochastic gradient descent, i.e., $w := w - \eta \tilde{g} J(w)$
 - \tilde{g} : gradient at a **randomly** chosen point.
- ▶ Mini-batch gradient descent, i.e., $w := w - \eta \tilde{g}_B J(w)$
 - \tilde{g} : gradient with respect to a set of B **randomly** chosen points.





Let's Scale the Learning



Scalable Training

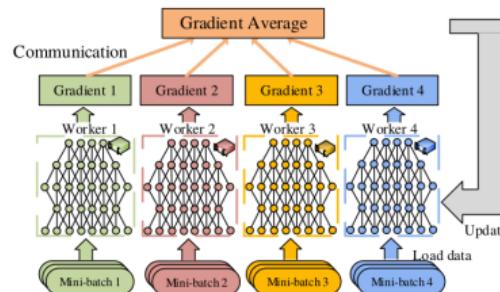
- ▶ Data parallelism
- ▶ Model parallelism



Data Parallelism

Data Parallelization (1/4)

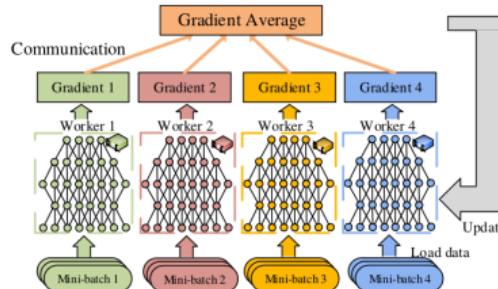
- ▶ Replicate a whole model on every device.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey , 2020]

Data Parallelization (1/4)

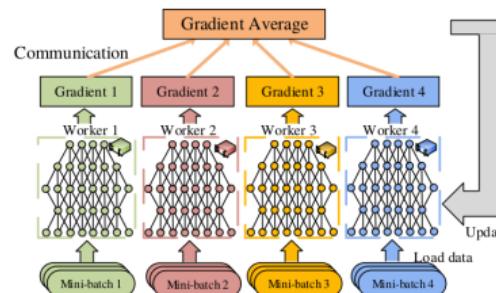
- ▶ Replicate a **whole model** on **every device**.
- ▶ Train **all replicas simultaneously**, using a **different mini-batch** for each.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (2/4)

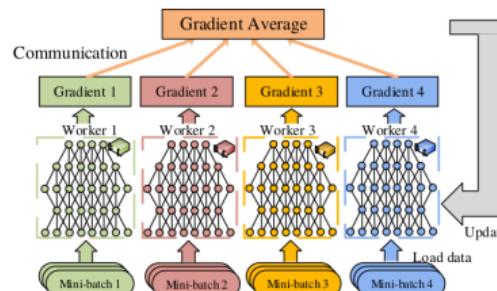
- ▶ k devices



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (2/4)

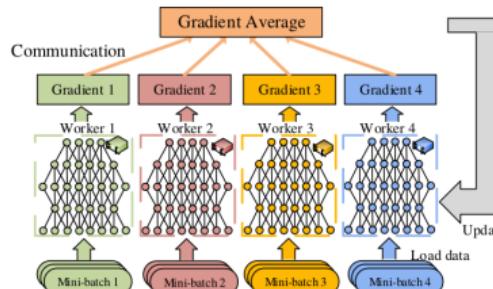
- ▶ k devices
- ▶ $J_j(w) = \sum_{i=1}^{b_j} l(y_i, \hat{y}_i), \forall j = 1, 2, \dots, k$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (2/4)

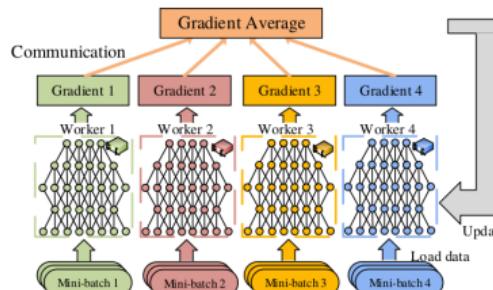
- ▶ k devices
- ▶ $J_j(w) = \sum_{i=1}^{b_j} l(y_i, \hat{y}_i), \forall j = 1, 2, \dots, k$
- ▶ $\tilde{g}_B J_j(w)$: gradient of $J_j(w)$ with respect to a set of B randomly chosen points at device j .



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (2/4)

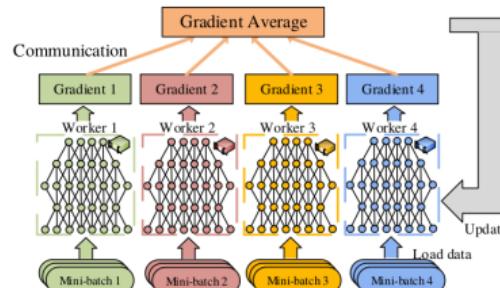
- ▶ k devices
- ▶ $J_j(w) = \sum_{i=1}^{b_j} l(y_i, \hat{y}_i), \forall j = 1, 2, \dots, k$
- ▶ $\tilde{g}_B J_j(w)$: gradient of $J_j(w)$ with respect to a set of B randomly chosen points at device j .
- ▶ Compute $\tilde{g}_B J_j(w)$ on each device j .



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (3/4)

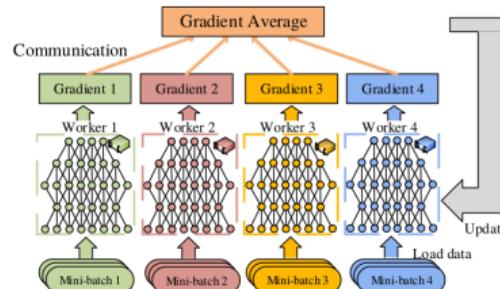
- ▶ Compute the **mean of the gradients**.
- ▶ $\tilde{g}_B J(w) = \frac{1}{k} \sum_{j=1}^k \tilde{g}_B J_j(w)$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (4/4)

- ▶ Update the model.
- ▶ $w := w - \eta \tilde{g}_B J(w)$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]



Data Parallelization Design Issues

- ▶ The **aggregation** algorithm
- ▶ Communication **synchronization** and frequency
- ▶ Communication **compression**



The Aggregation Algorithm



The Aggregation Algorithm

- ▶ How to aggregate gradients (compute the mean of the gradients)?



The Aggregation Algorithm

- ▶ How to aggregate gradients (compute the mean of the gradients)?
- ▶ Centralized - parameter server



The Aggregation Algorithm

- ▶ How to aggregate gradients (compute the mean of the gradients)?
- ▶ Centralized - parameter server
- ▶ Decentralized - all-reduce



The Aggregation Algorithm

- ▶ How to aggregate gradients (compute the mean of the gradients)?
- ▶ Centralized - parameter server
- ▶ Decentralized - all-reduce
- ▶ Decentralized - gossip

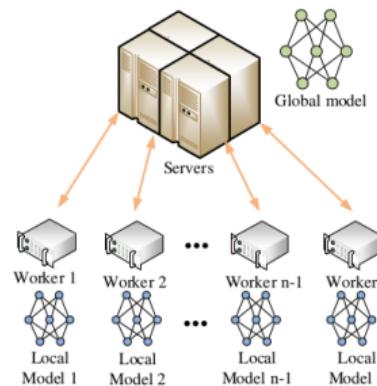


Aggregation - Centralized - Parameter Server

- ▶ Store the model parameters **outside of the workers**.

Aggregation - Centralized - Parameter Server

- ▶ Store the model parameters **outside of the workers**.
- ▶ **Workers** periodically report their **computed parameters** or **parameter updates** to a (set of) **parameter server(s) (PSs)**.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

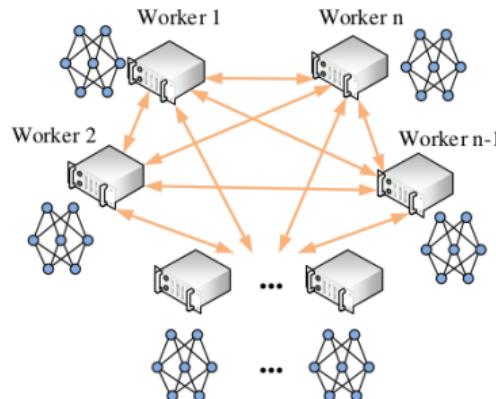


Aggregation - Distributed - All-Reduce

- ▶ Mirror all the model parameters across all workers (no PS).

Aggregation - Distributed - All-Reduce

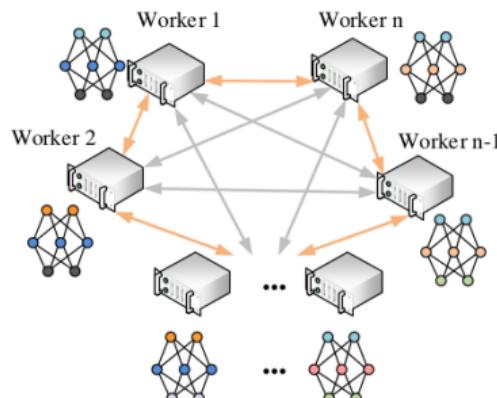
- ▶ Mirror all the model **parameters** across all workers (no PS).
- ▶ Workers **exchange** parameter updates **directly** via an **allreduce** operation.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Aggregation - Distributed - Gossip

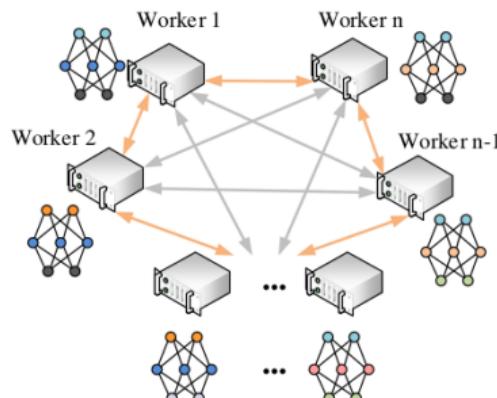
- ▶ No PS, and no global model.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Aggregation - Distributed - Gossip

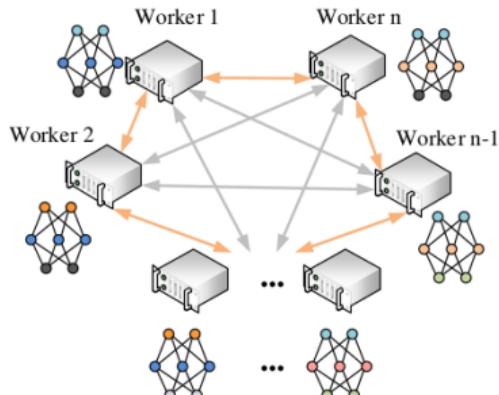
- ▶ No PS, and no global model.
- ▶ Every worker communicates updates with their neighbors.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Aggregation - Distributed - Gossip

- ▶ No PS, and no global model.
- ▶ Every worker communicates updates with their neighbors.
- ▶ The consistency of parameters across all workers only at the end of the algorithm.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]



Reduce and AllReduce (1/2)

- ▶ **Reduce**: reducing a **set of numbers** into a **smaller set of numbers** via a function.

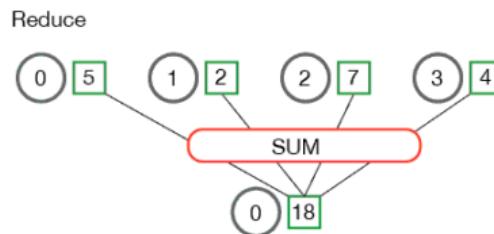


Reduce and AllReduce (1/2)

- ▶ **Reduce**: reducing a **set of numbers** into a **smaller set of numbers** via a function.
- ▶ E.g., `sum([1, 2, 3, 4, 5]) = 15`

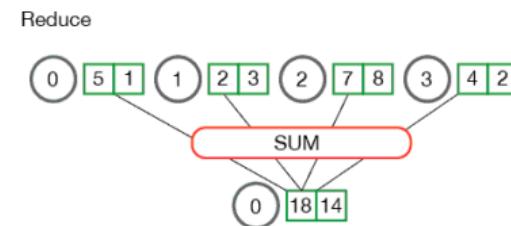
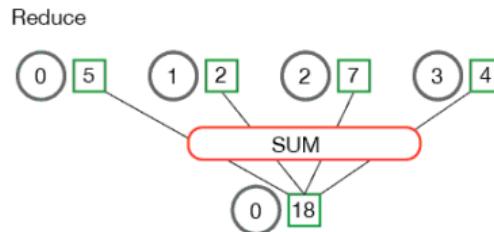
Reduce and AllReduce (1/2)

- ▶ **Reduce**: reducing a **set of numbers** into a **smaller set of numbers** via a function.
- ▶ E.g., `sum([1, 2, 3, 4, 5]) = 15`
- ▶ Reduce takes an **array of input** elements on each process and returns an **array of output** elements to the **root process**.



Reduce and AllReduce (1/2)

- ▶ **Reduce**: reducing a **set of numbers** into a **smaller set of numbers** via a function.
- ▶ E.g., `sum([1, 2, 3, 4, 5]) = 15`
- ▶ Reduce takes an **array of input** elements on each process and returns an **array of output** elements to the root process.



[<https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce>]

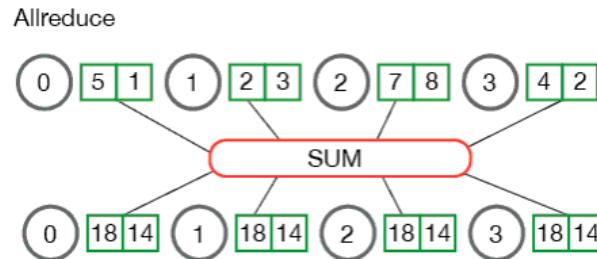


Reduce and AllReduce (2/2)

- ▶ AllReduce stores reduced results across all processes rather than the root process.

Reduce and AllReduce (2/2)

- ▶ AllReduce stores reduced results across all processes rather than the root process.



[<https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce>]

AllReduce Example

Initial state

Worker A
17 11 1 9

Worker B
5 13 23 14

Worker C
3 6 10 8

Worker D
12 7 2 12



After AllReduce operation

Worker A
37 37 36 43

Worker B
37 37 36 43

Worker C
37 37 36 43

Worker D
37 37 36 43

[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

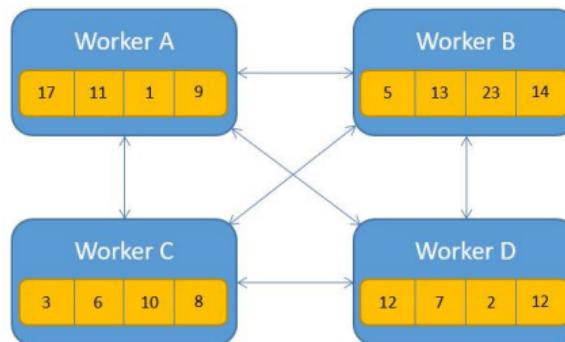


AllReduce Implementation

- ▶ All-to-all allreduce
- ▶ Master-worker allreduce
- ▶ Tree allreduce
- ▶ Round-robin allreduce
- ▶ Butterfly allreduce
- ▶ Ring allreduce

AllReduce Implementation - All-to-All AllReduce

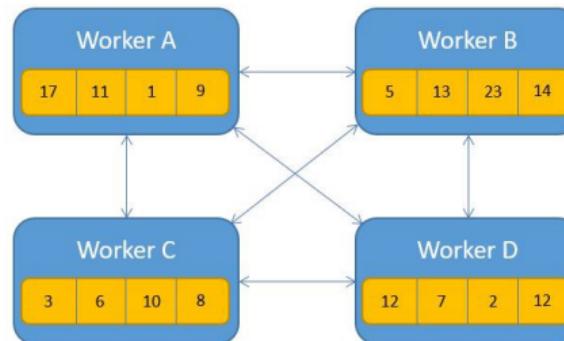
- ▶ Send the array of data to each other.
- ▶ Apply the reduction operation on each process.



[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

AllReduce Implementation - All-to-All AllReduce

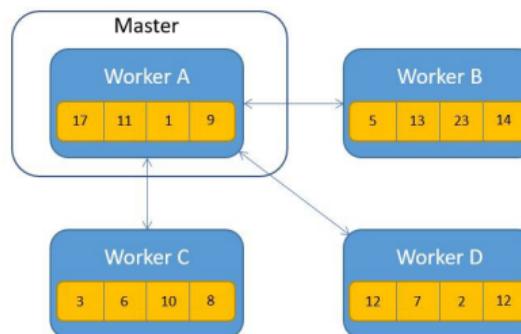
- ▶ Send the array of data to each other.
- ▶ Apply the reduction operation on each process.
- ▶ Too many unnecessary messages.



[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

AllReduce Implementation - Master-Worker AllReduce

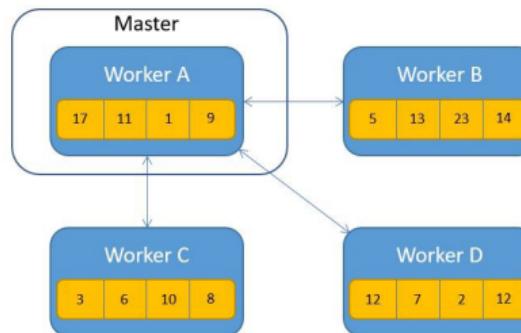
- ▶ Selecting one process as a **master**, gather all arrays into the master.
- ▶ Perform **reduction operations** locally in the **master**.
- ▶ Distribute the result to the **other processes**.



[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

AllReduce Implementation - Master-Worker AllReduce

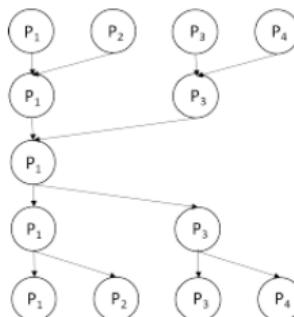
- ▶ Selecting one process as a **master**, gather all arrays into the master.
- ▶ Perform **reduction operations** locally in the **master**.
- ▶ **Distribute the result** to the **other processes**.
- ▶ The master becomes a **bottleneck** (**not scalable**).



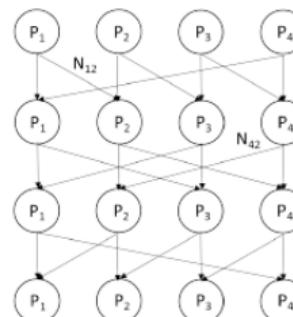
[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

AllReduce Implementation - Other implementations

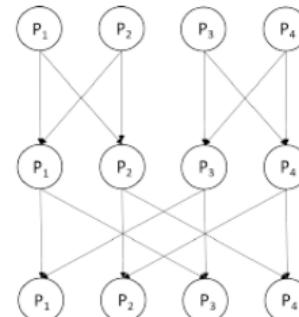
- ▶ Some try to minimize bandwidth.
- ▶ Some try to minimize latency.



(a) Tree AllReduce



(b) Round-robin AllReduce



(c) Butterfly AllReduce

[Zhao H. et al., arXiv:1312.3020, 2013]

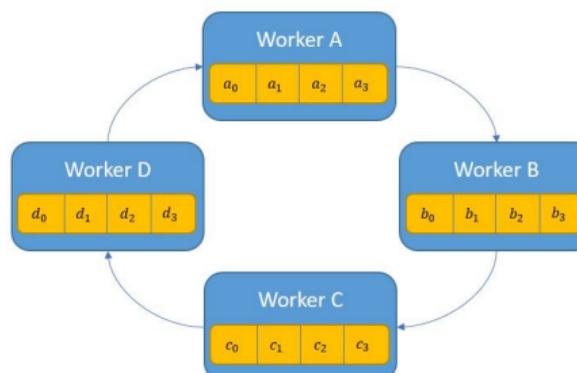


AllReduce Implementation - Ring-AllReduce (1/6)

- ▶ The **Ring-Allreduce** has two phases:
 1. First, the **share-reduce** phase
 2. Then, the **share-only** phase

AllReduce Implementation - Ring-AllReduce (2/6)

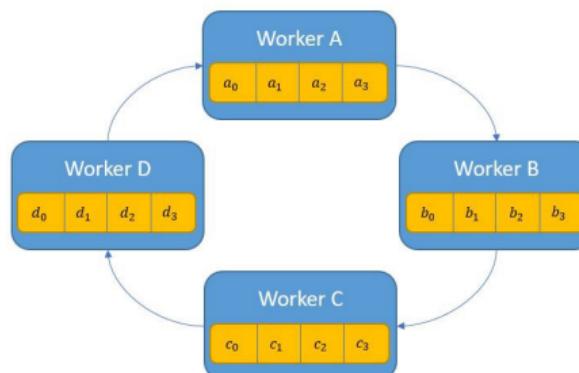
- In the **share-reduce** phase, each process p sends data to the process $(p+1) \% m$
 - m is the number of processes, and $\%$ is the modulo operator.



[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

AllReduce Implementation - Ring-AllReduce (2/6)

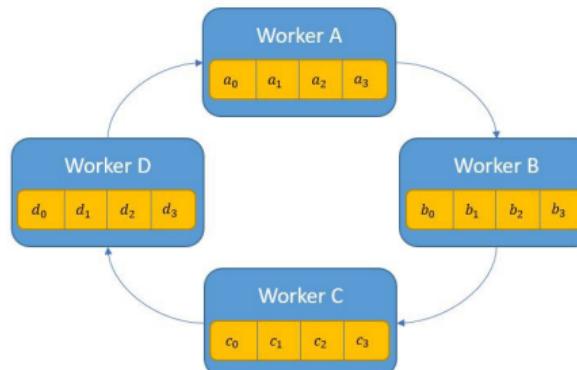
- ▶ In the **share-reduce** phase, each process p sends data to the process $(p+1) \% m$
 - m is the number of processes, and $\%$ is the modulo operator.
- ▶ The **array of data** on each process is divided to m chunks ($m=4$ here).



[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

AllReduce Implementation - Ring-AllReduce (2/6)

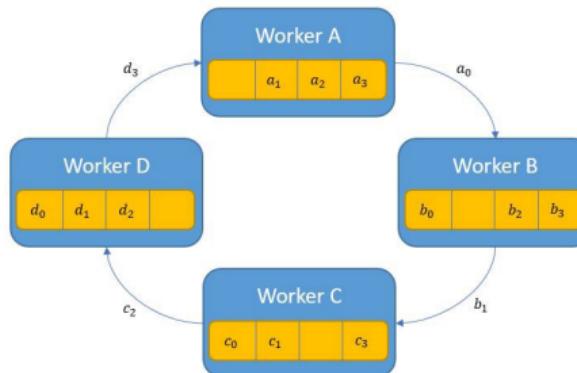
- In the **share-reduce** phase, each process p sends data to the process $(p+1) \% m$
 - m is the number of processes, and $\%$ is the modulo operator.
- The **array of data** on each process is divided to m chunks ($m=4$ here).
- Each one of these **chunks** will be **indexed** by i going forward.



[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

AllReduce Implementation - Ring-AllReduce (3/6)

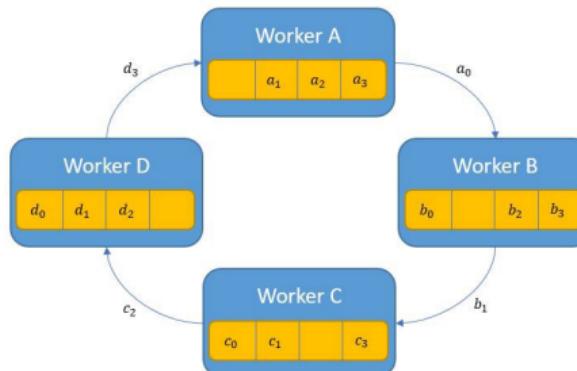
- In the first share-reduce step, process A sends a_0 to process B.



[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

AllReduce Implementation - Ring-AllReduce (3/6)

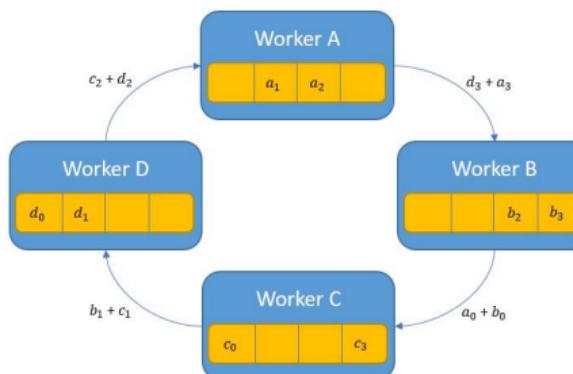
- ▶ In the **first share-reduce step**, process **A** sends **a₀** to process **B**.
- ▶ Process **B** sends **b₁** to process **C**, etc.



[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

AllReduce Implementation - Ring-AllReduce (4/6)

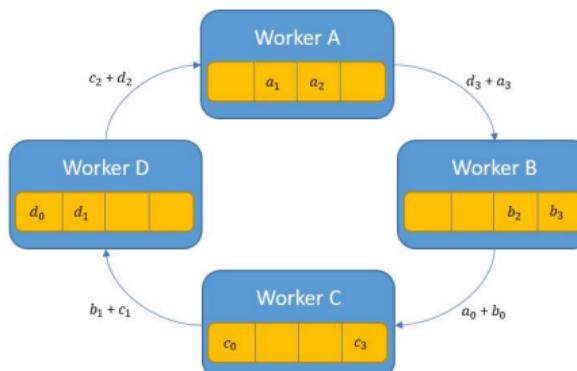
- When each process receives the data from the previous process, it applies the reduce operator (e.g., sum)



[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

AllReduce Implementation - Ring-AllReduce (4/6)

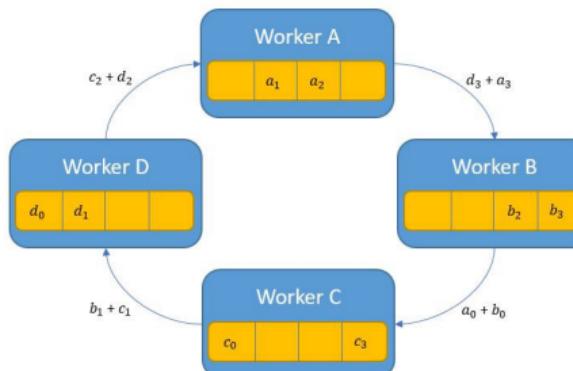
- When each process receives the data from the previous process, it applies the reduce operator (e.g., sum)
 - The reduce operator should be associative and commutative.



[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

AllReduce Implementation - Ring-AllReduce (4/6)

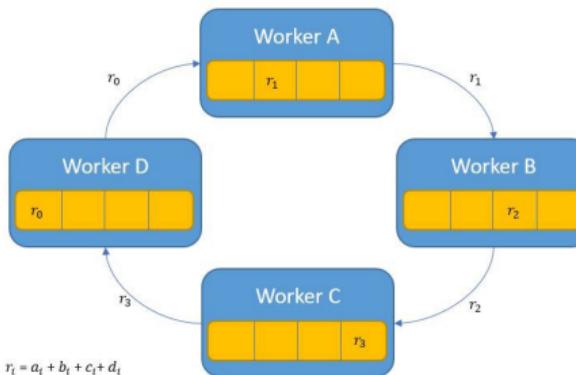
- When each process receives the data from the previous process, it applies the reduce operator (e.g., sum)
 - The reduce operator should be associative and commutative.
- It then proceeds to send it to the next process in the ring.



[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

AllReduce Implementation - Ring-AllReduce (5/6)

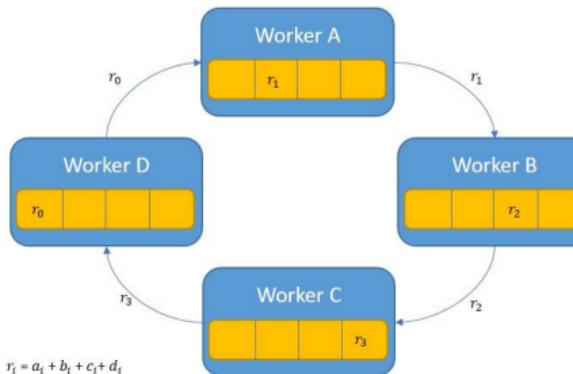
- The share-reduce phase finishes when each process holds the complete reduction of chunk i.



[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

AllReduce Implementation - Ring-AllReduce (5/6)

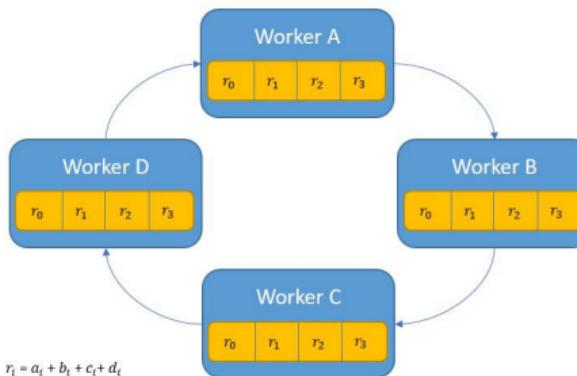
- ▶ The share-reduce phase finishes when each process holds the complete reduction of chunk i.
- ▶ At this point each process holds a part of the end result.



[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

AllReduce Implementation - Ring-AllReduce (6/6)

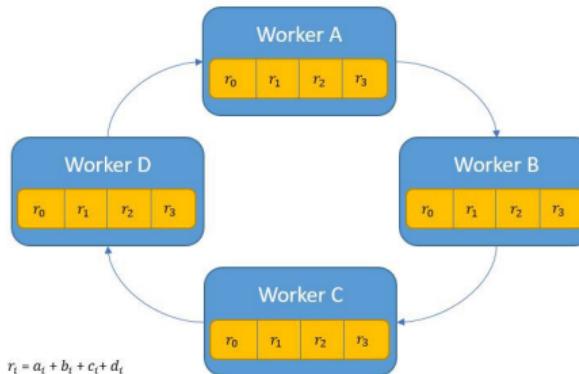
- The **share-only** step is the same process of sharing the data in a ring-like fashion without applying the reduce operation.



[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]

AllReduce Implementation - Ring-AllReduce (6/6)

- ▶ The **share-only** step is the same process of sharing the data in a ring-like fashion **without applying the reduce operation**.
- ▶ This **consolidates** the **result of each chunk** in **every process**.



[<https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da>]



Master-Worker AllReduce vs. Ring-AllReduce

- ▶ N : number of elements, m : number of processes



Master-Worker AllReduce vs. Ring-AllReduce

- ▶ N : number of elements, m : number of processes
- ▶ Master-Worker AllReduce



Master-Worker AllReduce vs. Ring-AllReduce

- ▶ N : number of elements, m : number of processes
- ▶ Master-Worker AllReduce
 - First each process sends N elements to the master: $N \times (m - 1)$ messages.



Master-Worker AllReduce vs. Ring-AllReduce

- ▶ N : number of elements, m : number of processes
- ▶ Master-Worker AllReduce
 - First each **process** sends N elements to the **master**: $N \times (m - 1)$ messages.
 - Then the **master** sends the results back to the **process**: another $N \times (m - 1)$ messages.



Master-Worker AllReduce vs. Ring-AllReduce

- ▶ N : number of elements, m : number of processes
- ▶ Master-Worker AllReduce
 - First each process sends N elements to the master: $N \times (m - 1)$ messages.
 - Then the master sends the results back to the process: another $N \times (m - 1)$ messages.
 - Total network traffic is $2(N \times (m - 1))$, which is proportional to m .



Master-Worker AllReduce vs. Ring-AllReduce

- ▶ N : number of elements, m : number of processes
- ▶ Master-Worker AllReduce
 - First each **process** sends N elements to the **master**: $N \times (m - 1)$ messages.
 - Then the **master** sends the results back to the **process**: another $N \times (m - 1)$ messages.
 - Total network traffic is $2(N \times (m - 1))$, which is **proportional** to m .
- ▶ Ring-AllReduce



Master-Worker AllReduce vs. Ring-AllReduce

- ▶ N : number of elements, m : number of processes
- ▶ Master-Worker AllReduce
 - First each **process** sends N elements to the **master**: $N \times (m - 1)$ messages.
 - Then the **master** sends the results back to the **process**: another $N \times (m - 1)$ messages.
 - Total network traffic is $2(N \times (m - 1))$, which is **proportional** to m .
- ▶ Ring-AllReduce
 - In the **share-reduce** step each **process** sends $\frac{N}{m}$ elements, and it does it $m - 1$ times:
 $\frac{N}{m} \times (m - 1)$ messages.

Master-Worker AllReduce vs. Ring-AllReduce

- ▶ N : number of elements, m : number of processes
- ▶ Master-Worker AllReduce
 - First each **process** sends N elements to the **master**: $N \times (m - 1)$ messages.
 - Then the **master** sends the results back to the **process**: another $N \times (m - 1)$ messages.
 - Total network traffic is $2(N \times (m - 1))$, which is **proportional** to m .
- ▶ Ring-AllReduce
 - In the **share-reduce** step each **process** sends $\frac{N}{m}$ elements, and it does it $m - 1$ times:
 $\frac{N}{m} \times (m - 1)$ messages.
 - On the **share-only** step, each **process** sends the result for the chunk it calculated: another
 $\frac{N}{m} \times (m - 1)$ messages.

Master-Worker AllReduce vs. Ring-AllReduce

- ▶ N : number of elements, m : number of processes
- ▶ Master-Worker AllReduce
 - First each **process** sends N elements to the **master**: $N \times (m - 1)$ messages.
 - Then the **master** sends the results back to the **process**: another $N \times (m - 1)$ messages.
 - Total network traffic is $2(N \times (m - 1))$, which is **proportional** to m .
- ▶ Ring-AllReduce
 - In the **share-reduce** step each **process** sends $\frac{N}{m}$ elements, and it does it $m - 1$ times:
 $\frac{N}{m} \times (m - 1)$ messages.
 - On the **share-only** step, each **process** sends the result for the chunk it calculated: another
 $\frac{N}{m} \times (m - 1)$ messages.
 - Total network traffic is $2(\frac{N}{m} \times (m - 1))$.



Communication Synchronization and Frequency



Synchronization

- ▶ When to synchronize the parameters among the parallel workers?



Communication Synchronization (1/2)

- ▶ Synchronizing the model replicas in **data-parallel** training requires **communication**
 - between **workers**, in **allreduce**
 - between **workers and parameter servers**, in the **centralized architecture**



Communication Synchronization (1/2)

- ▶ Synchronizing the model replicas in data-parallel training requires communication
 - between workers, in allreduce
 - between workers and parameter servers, in the centralized architecture
- ▶ The communication synchronization decides how frequently all local models are synchronized with others.



Communication Synchronization (2/2)

- ▶ It will influence:
 - The communication **traffic**
 - The **performance**
 - The **convergence** of model training



Communication Synchronization (2/2)

- ▶ It will influence:
 - The communication **traffic**
 - The **performance**
 - The **convergence** of model training

- ▶ There is a **trade-off** between the communication **traffic** and the **convergence**.



Reducing Synchronization Overhead

- ▶ Two directions for improvement:



Reducing Synchronization Overhead

- ▶ Two directions for improvement:
 1. To **relax** the **synchronization** among all workers.



Reducing Synchronization Overhead

- ▶ Two directions for improvement:
 1. To **relax** the **synchronization** among all workers.
 2. The **frequency of communication** can be **reduced** by more computation in one iteration.

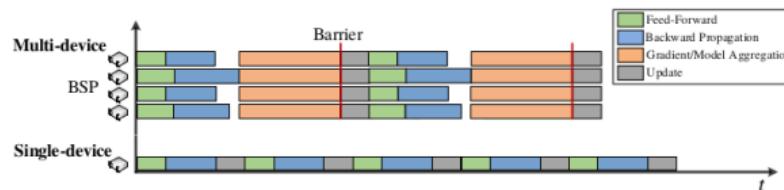


Communication Synchronization Models

- ▶ Synchronous
- ▶ Stale-synchronous
- ▶ Asynchronous
- ▶ Local SGD

Communication Synchronization - Synchronous

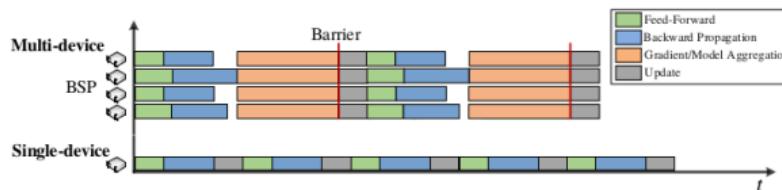
- ▶ After each **iteration**, the workers **synchronize** their parameter updates.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Synchronous

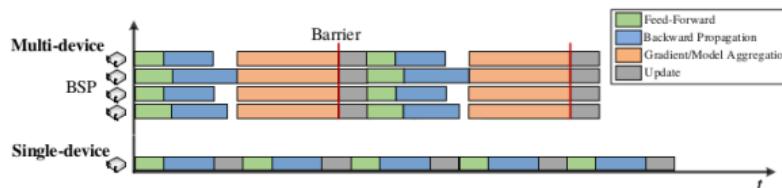
- ▶ After each **iteration**, the workers **synchronize** their parameter updates.
- ▶ Every worker must **wait** for **all workers** to **finish** the transmission of all parameters in the current iteration, before the **next training**.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Synchronous

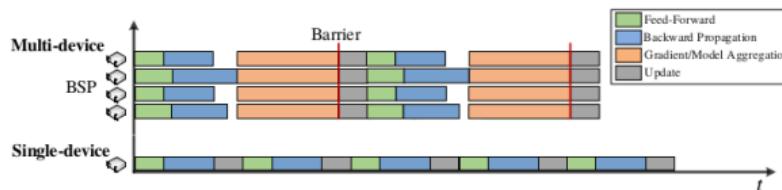
- ▶ After each **iteration**, the workers **synchronize** their parameter updates.
- ▶ Every worker must **wait** for **all workers** to **finish** the transmission of all parameters in the current iteration, before the **next training**.
- ▶ **Stragglers** can influence the overall system **throughput**.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Synchronous

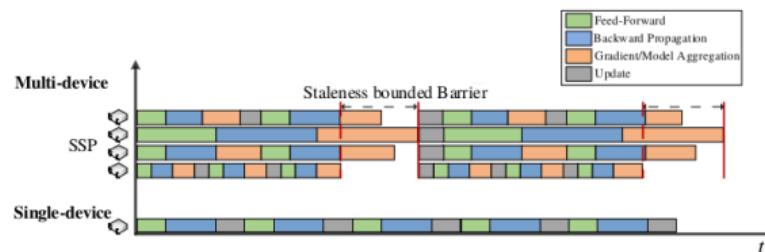
- ▶ After each **iteration**, the workers **synchronize** their parameter updates.
- ▶ Every worker must **wait** for **all workers** to **finish** the transmission of all parameters in the current iteration, before the **next training**.
- ▶ **Stragglers** can influence the overall system **throughput**.
- ▶ High **communication** cost that **limits** the system **scalability**.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (1/2)

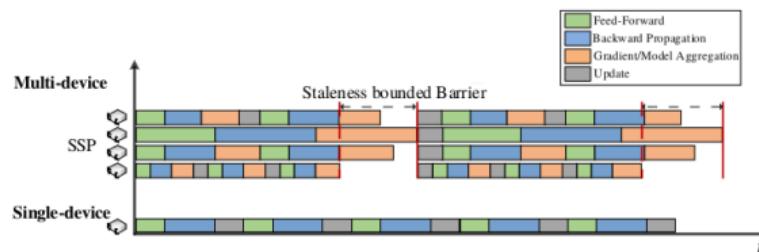
- ▶ Alleviate the straggler problem without losing synchronization.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (1/2)

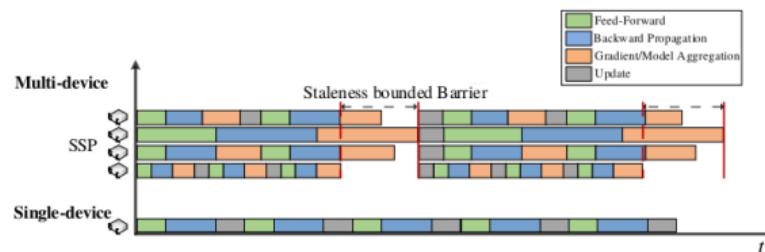
- ▶ Alleviate the straggler problem without losing synchronization.
- ▶ The faster workers to do **more updates** than the **slower workers** to **reduce the waiting time** of the faster workers.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (1/2)

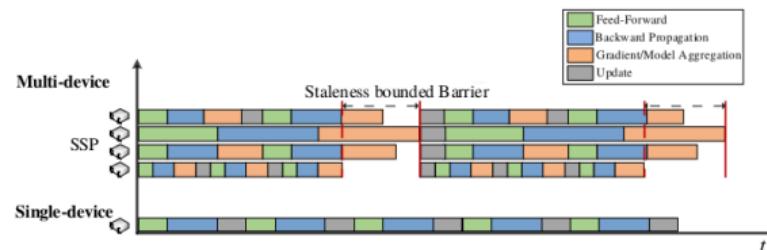
- ▶ Alleviate the straggler problem without losing synchronization.
- ▶ The faster workers to do **more updates** than the **slower workers** to reduce the waiting time of the faster workers.
- ▶ **Staleness bounded barrier** to limit the **iteration gap** between the fastest worker and the slowest worker.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (2/2)

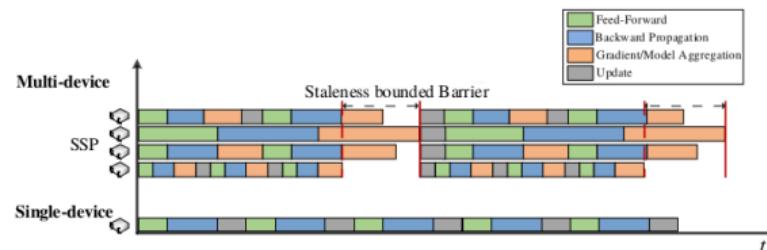
- ▶ For a maximum staleness bound s , the update formula of worker i at iteration $t+1$:
- ▶ $w_{i,t+1} := w_0 - \eta(\sum_{k=1}^t \sum_{j=1}^n G_{j,k} + \sum_{k=t-s}^t G_{i,k} + \sum_{(j,k) \in S_{i,t+1}} G_{j,k})$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (2/2)

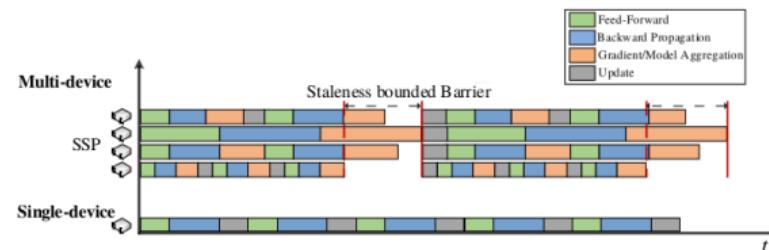
- ▶ For a maximum staleness bound s , the update formula of worker i at iteration $t+1$:
- ▶ $w_{i,t+1} := w_0 - \eta(\sum_{k=1}^t \sum_{j=1}^n G_{j,k} + \sum_{k=t-s}^t G_{i,k} + \sum_{(j,k) \in S_{i,t+1}} G_{j,k})$
- ▶ The update has three parts:



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (2/2)

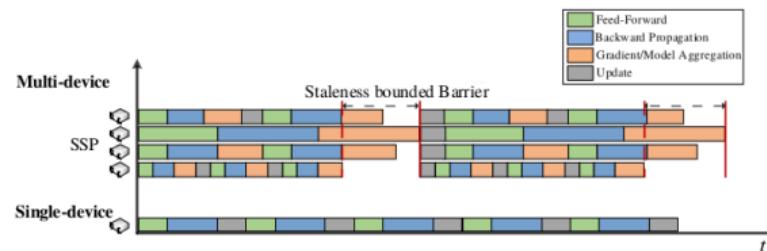
- ▶ For a maximum staleness bound s , the update formula of worker i at iteration $t+1$:
- ▶ $w_{i,t+1} := w_0 - \eta(\sum_{k=1}^t \sum_{j=1}^n G_{j,k} + \sum_{k=t-s}^t G_{i,k} + \sum_{(j,k) \in S_{i,t+1}} G_{j,k})$
- ▶ The update has three parts:
 1. Guaranteed pre-window updates from clock 1 to t over all workers.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (2/2)

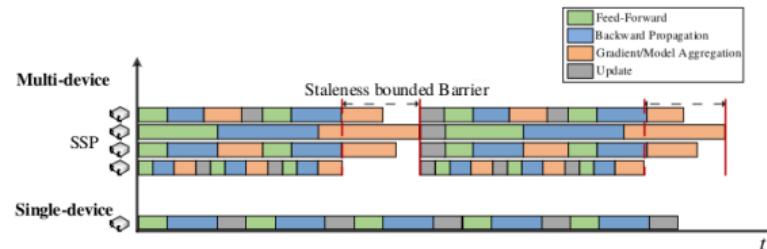
- ▶ For a maximum staleness bound s , the update formula of worker i at iteration $t+1$:
- ▶ $w_{i,t+1} := w_0 - \eta(\sum_{k=1}^t \sum_{j=1}^n G_{j,k} + \sum_{k=t-s}^t G_{i,k} + \sum_{(j,k) \in S_{i,t+1}} G_{j,k})$
- ▶ The update has three parts:
 1. Guaranteed pre-window updates from clock 1 to t over all workers.
 2. Guaranteed read-my-writes in-window updates made by the querying worker i .



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (2/2)

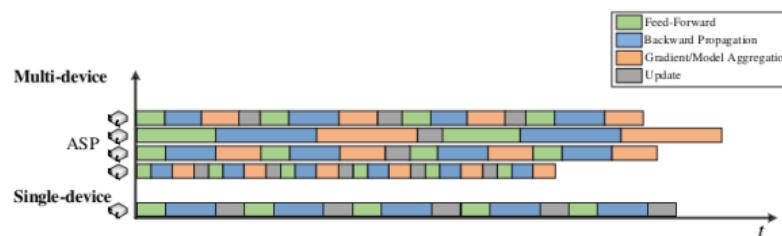
- ▶ For a maximum staleness bound s , the update formula of worker i at iteration $t+1$:
- ▶ $w_{i,t+1} := w_0 - \eta(\sum_{k=1}^t \sum_{j=1}^n G_{j,k} + \sum_{k=t-s}^t G_{i,k} + \sum_{(j,k) \in S_{i,t+1}} G_{j,k})$
- ▶ The update has three parts:
 1. Guaranteed pre-window updates from clock 1 to t over all workers.
 2. Guaranteed read-my-writes in-window updates made by the querying worker i .
 3. Best-effort in-window updates. $S_{i,t+1}$ is some subset of the updates from other workers during period $[t - s]$.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Asynchronous (1/2)

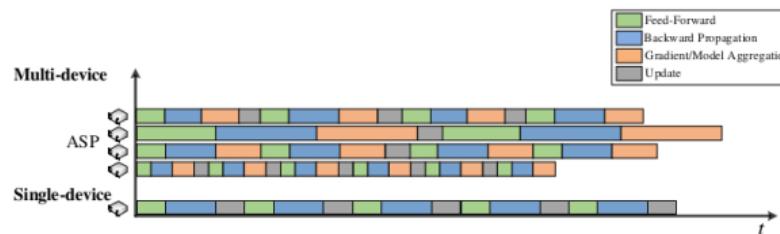
- ▶ It completely eliminates the synchronization.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Asynchronous (1/2)

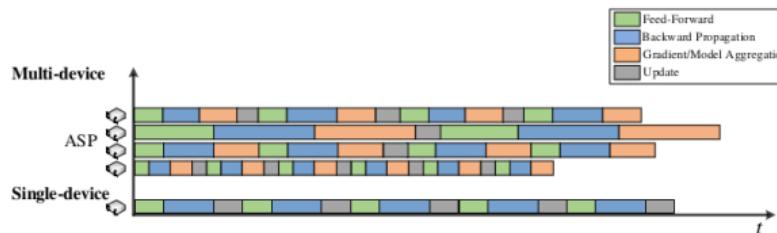
- ▶ It completely **eliminates** the synchronization.
- ▶ Each work **transmits its gradients** to the PS after it calculates the gradients.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Asynchronous (1/2)

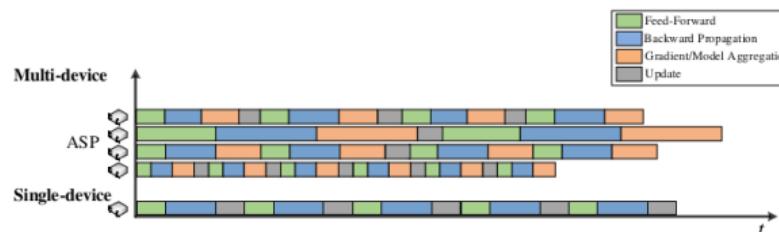
- ▶ It completely **eliminates** the synchronization.
- ▶ Each work **transmits its gradients** to the PS after it calculates the gradients.
- ▶ The PS updates the global model **without waiting** for the other workers.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Asynchronous (2/2)

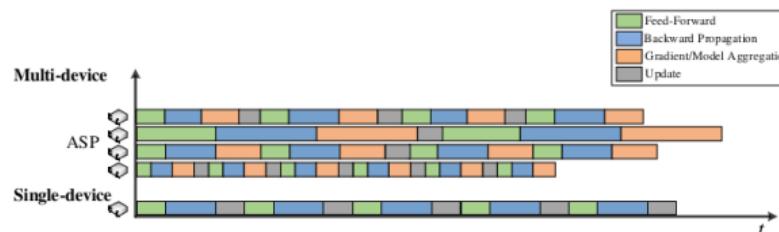
$$\triangleright w_{t+1} := w_t - \eta \sum_{i=1}^n G_{i,t-\tau_{k,i}}$$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Asynchronous (2/2)

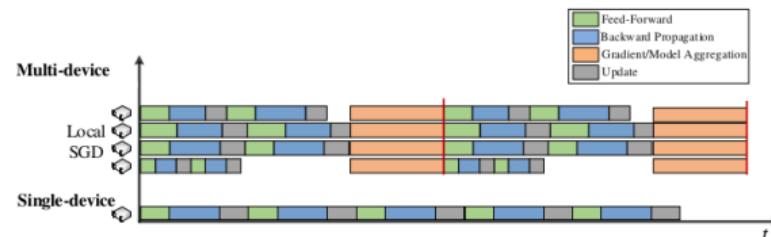
- ▶ $w_{t+1} := w_t - \eta \sum_{i=1}^n G_{i,t-\tau_{k,i}}$
- ▶ $\tau_{k,i}$ is the time delay between the moment when worker i calculates the gradient at the current iteration.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Local SGD

- ▶ All workers **run several iterations**, and then **averages all local models** into the newest global model.

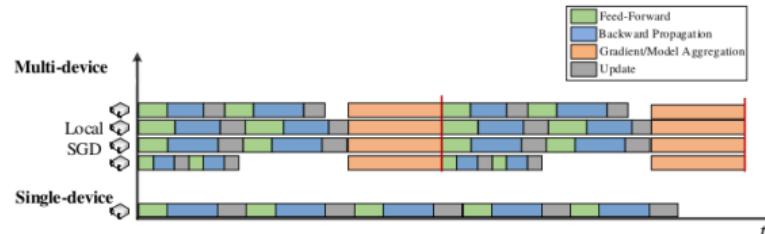


[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Local SGD

- ▶ All workers **run several iterations**, and then **averages all local models** into the newest global model.
- ▶ If \mathcal{I}_T represents the synchronization timestamps, then:

$$w_{i,t+1} = \begin{cases} w_{i,t} - \eta g_{i,t} & \text{if } t+1 \notin \mathcal{I}_T \\ w_{i,t} - \eta \frac{1}{n} \sum_{i=1}^n g_{i,t} & \text{if } t+1 \in \mathcal{I}_T \end{cases}$$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]



Communication Compression



Communication Compression

- ▶ Reduce the communication traffic with **little impact** on the model convergence.



Communication Compression

- ▶ Reduce the communication traffic with **little impact** on the model convergence.
- ▶ Compress the exchanged gradients or models **before transmitting** across the network.



Communication Compression

- ▶ Reduce the communication traffic with **little impact** on the model convergence.
- ▶ Compress the exchanged gradients or models **before transmitting** across the network.
- ▶ Quantization

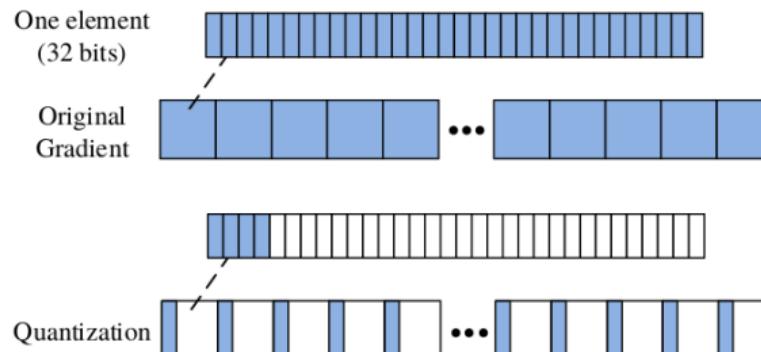


Communication Compression

- ▶ Reduce the communication traffic with **little impact** on the model convergence.
- ▶ Compress the exchanged gradients or models **before transmitting** across the network.
- ▶ Quantization
- ▶ Sparsification

Communication Compression - Quantization

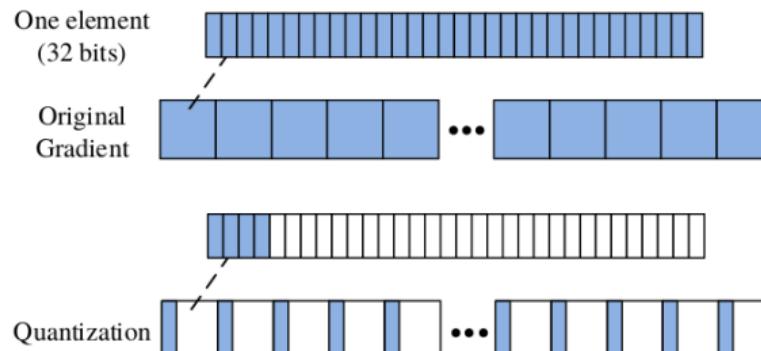
- ▶ Using lower bits to represent the data.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Compression - Quantization

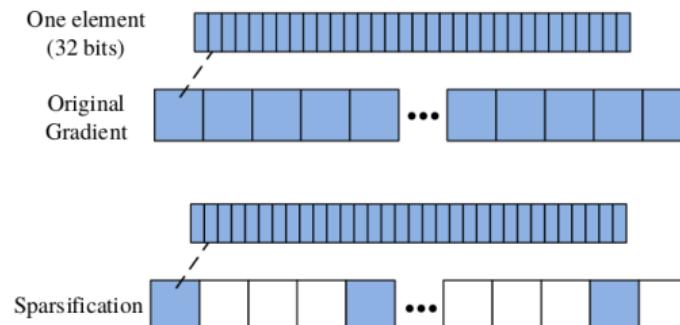
- ▶ Using lower bits to represent the data.
- ▶ The gradients are of low precision.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Compression - Sparsification

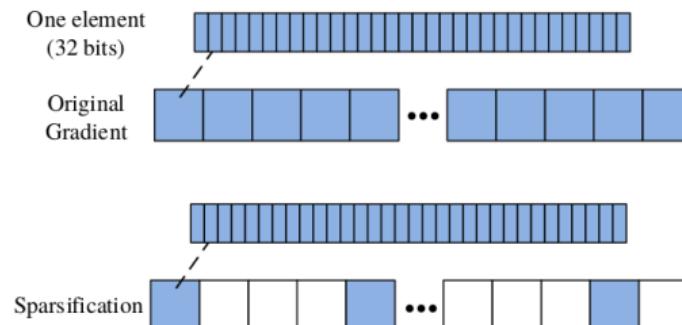
- ▶ Reducing the **number of elements** that are transmitted at each iteration.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Compression - Sparsification

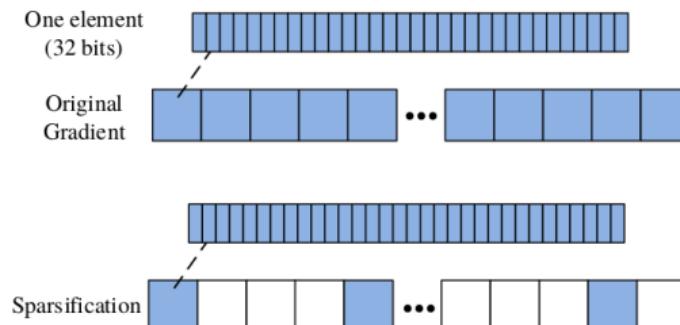
- ▶ Reducing the **number of elements** that are transmitted at each iteration.
- ▶ Only **significant gradients** are required to **update the model parameter** to guarantee the convergence of the training.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Compression - Sparsification

- ▶ Reducing the **number of elements** that are transmitted at each iteration.
- ▶ Only **significant gradients** are required to **update the model parameter** to guarantee the convergence of the training.
- ▶ E.g., the **zero-valued** elements are no need to transmit.



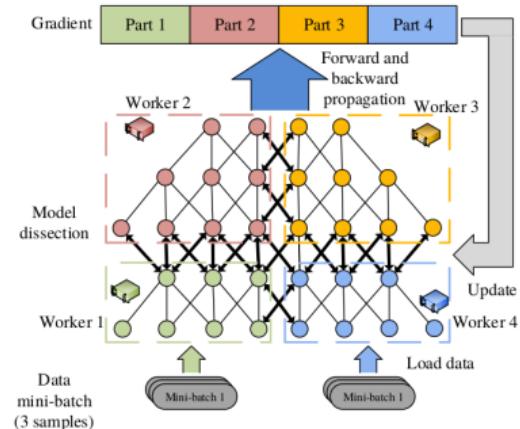
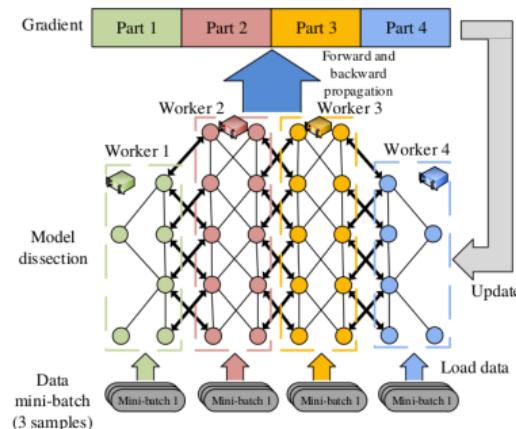
[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]



Model Parallelism

Model Parallelization

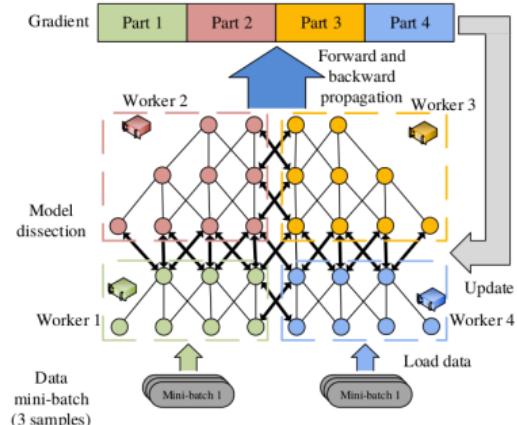
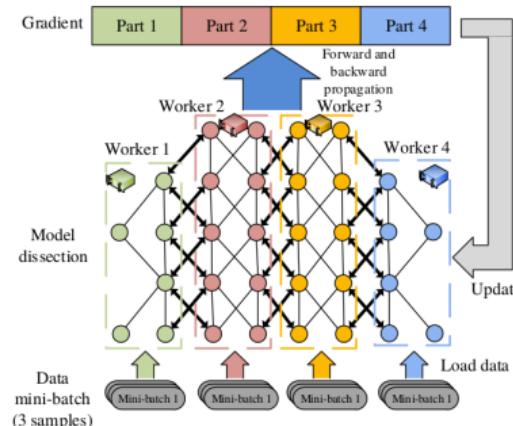
- The model is split across multiple devices.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Model Parallelization

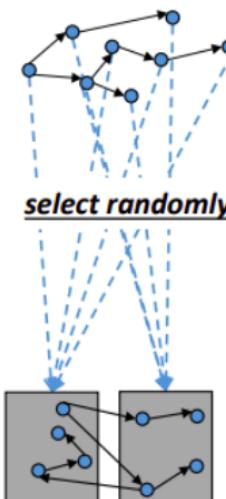
- The model is split across multiple devices.
- Depends on the architecture of the NN.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Model Parallelization - Hash Partitioning

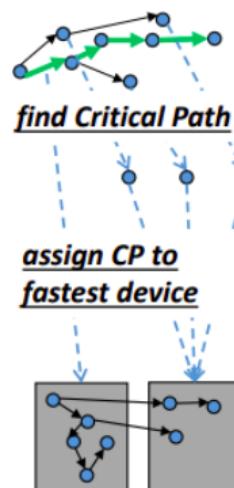
- ▶ Randomly assign vertices to devices proportionally to the capacity of the devices by using a **hash function**.



[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]

Model Parallelization - Critical Path

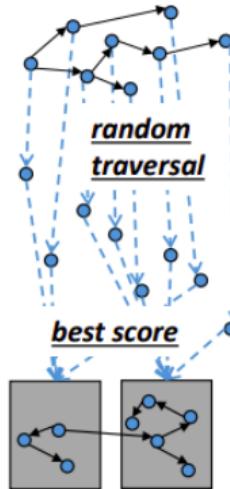
- ▶ Assigning the complete **critical path** to the fastest device.
- ▶ **Critical path**: the path with the **longest computation time** from source to sink vertex.



[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]

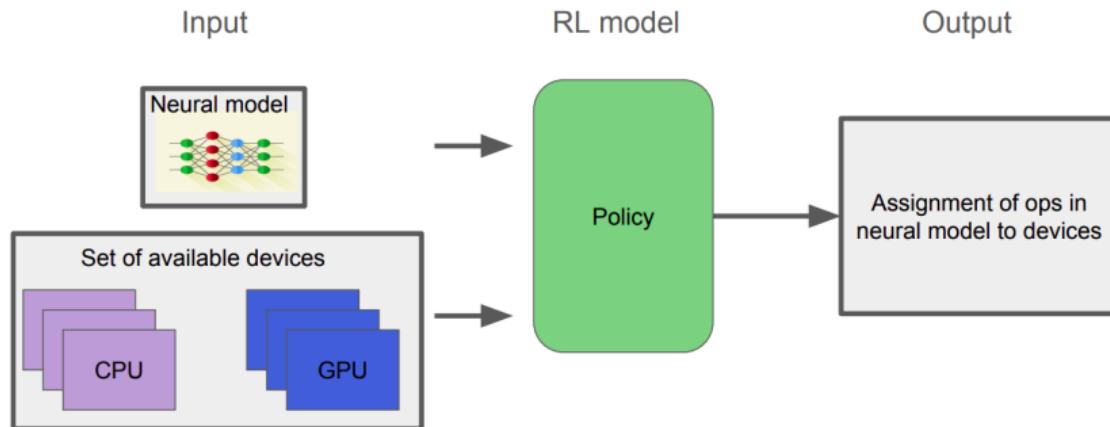
Model Parallelization - Multi-Objective Heuristics

- ▶ Different **objectives**, e.g., memory, importance, traffic, and execution time



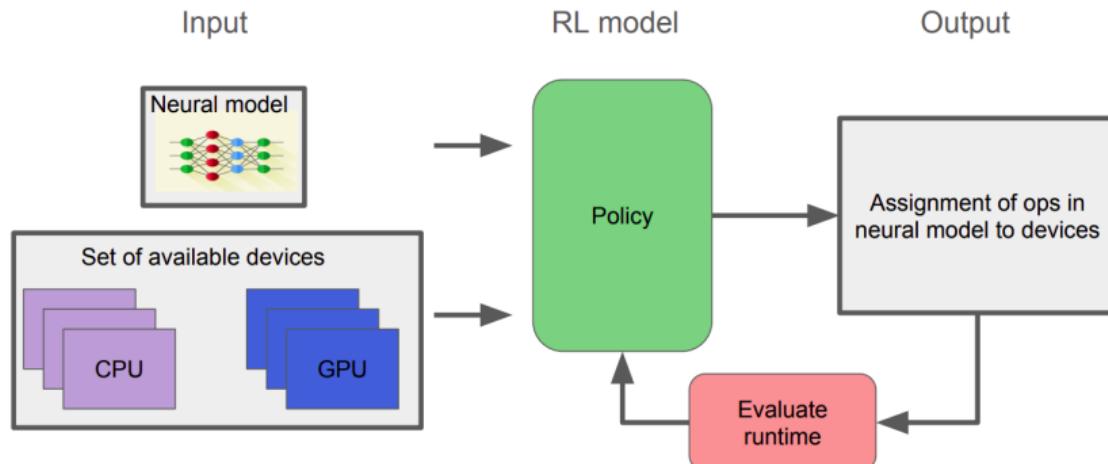
[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]

Model Parallelization - Reinforcement Learning (1/5)



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]

Model Parallelization - Reinforcement Learning (2/5)



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]



Model Parallelization - Reinforcement Learning (3/5)

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)}[R(\mathcal{P})|\mathcal{G}]$
- ▶ Objective: $\arg \min_w J(w)$



Model Parallelization - Reinforcement Learning (3/5)

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)}[R(\mathcal{P})|\mathcal{G}]$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ \mathcal{G} : input neural graph



Model Parallelization - Reinforcement Learning (3/5)

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)}[R(\mathcal{P})|\mathcal{G}]$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ \mathcal{G} : input neural graph
- ▶ R : runtime

Model Parallelization - Reinforcement Learning (3/5)

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)}[R(\mathcal{P})|\mathcal{G}]$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ \mathcal{G} : input neural graph
- ▶ R : runtime
- ▶ $J(w)$: expected runtime

Model Parallelization - Reinforcement Learning (3/5)

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)}[R(\mathcal{P})|\mathcal{G}]$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ \mathcal{G} : input neural graph
- ▶ R : runtime
- ▶ $J(w)$: expected runtime
- ▶ w : trainable parameters of policy



Model Parallelization - Reinforcement Learning (3/5)

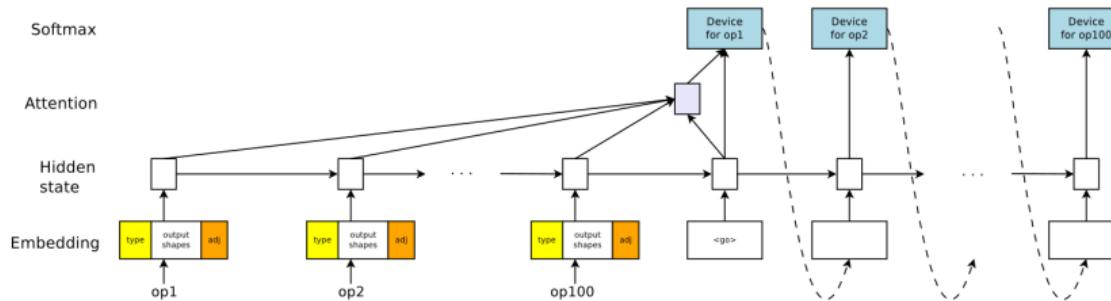
- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)}[R(\mathcal{P})|\mathcal{G}]$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ \mathcal{G} : input neural graph
- ▶ R : runtime
- ▶ $J(w)$: expected runtime
- ▶ w : trainable parameters of policy
- ▶ $\pi(\mathcal{P}|\mathcal{G}, w)$: policy

Model Parallelization - Reinforcement Learning (3/5)

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)}[R(\mathcal{P})|\mathcal{G}]$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ \mathcal{G} : input neural graph
- ▶ R : runtime
- ▶ $J(w)$: expected runtime
- ▶ w : trainable parameters of policy
- ▶ $\pi(\mathcal{P}|\mathcal{G}, w)$: policy
- ▶ \mathcal{P} : output placements $\in \{1, 2, \dots, num_ops\}^{num_devices}$

Model Parallelization - Reinforcement Learning (4/5)

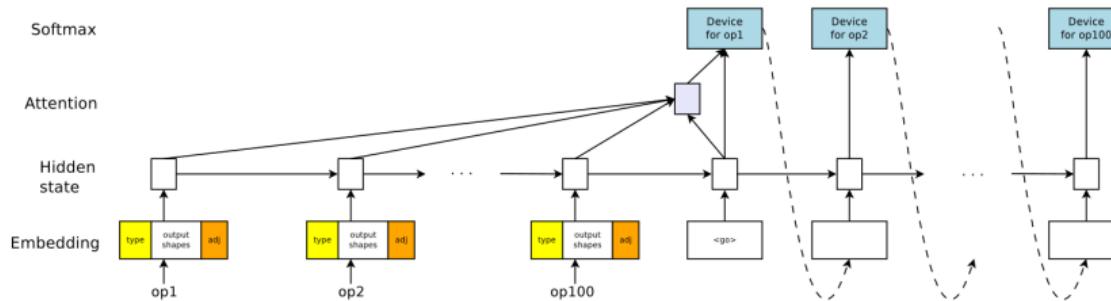
- ▶ RL reward function based on execution runtime.



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]

Model Parallelization - Reinforcement Learning (4/5)

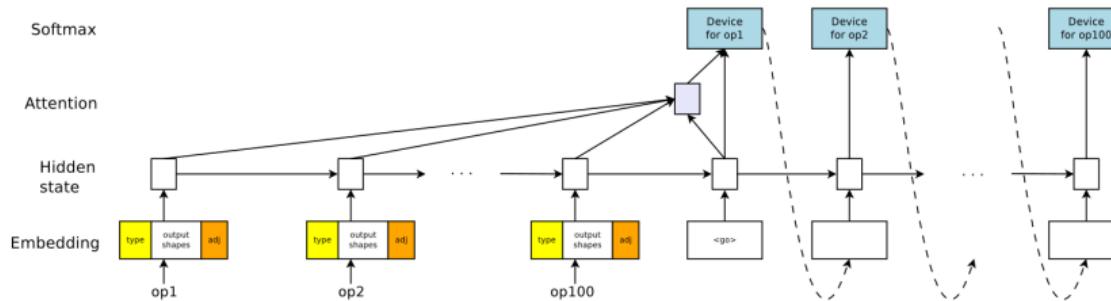
- ▶ RL reward function based on execution runtime.
- ▶ The RL policy is defined as a seq-to-seq model.



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]

Model Parallelization - Reinforcement Learning (4/5)

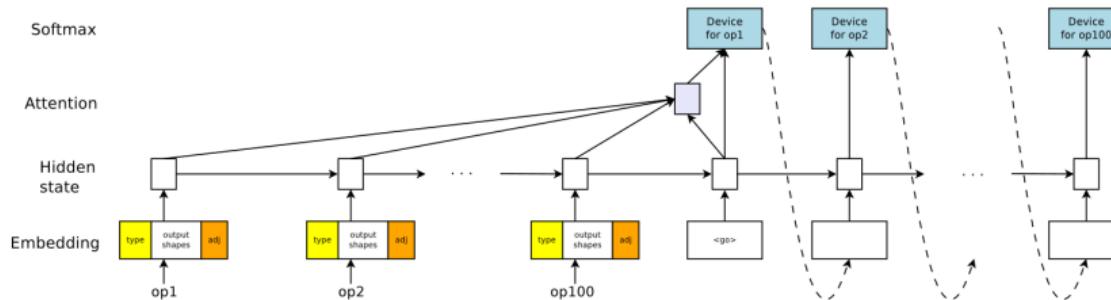
- ▶ RL reward function based on execution runtime.
- ▶ The RL policy is defined as a seq-to-seq model.
- ▶ RNN Encoder receives graph embedding for each operation.



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]

Model Parallelization - Reinforcement Learning (4/5)

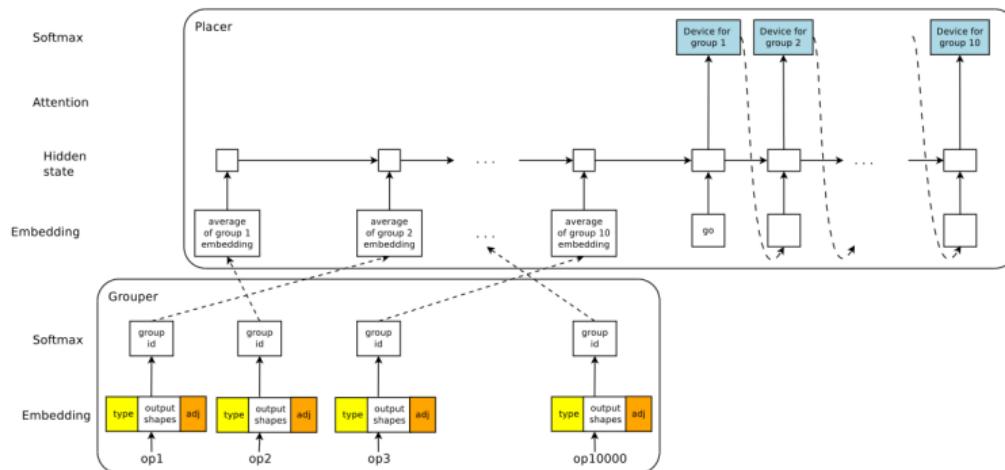
- ▶ RL reward function based on execution runtime.
- ▶ The RL policy is defined as a seq-to-seq model.
- ▶ RNN Encoder receives graph embedding for each operation.
- ▶ RNN Decoder predicts a device placement for each operation.



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]

Model Parallelization - Reinforcement Learning (5/5)

- ▶ Grouping operations.
- ▶ Prediction is for **group placement**, not for a single operation.



[Mirhoseini et al., A Hierarchical Model for Device Placement, 2018]



Summary



Summary

- ▶ Scalability matters
- ▶ Parallelization
- ▶ Data Parallelization
 - Parameter server vs. AllReduce
 - Synchronized vs. asynchronous
- ▶ Model Parallelization
 - Random, critical path, multi-objective, RL



Thanks!

HOPSWORKS

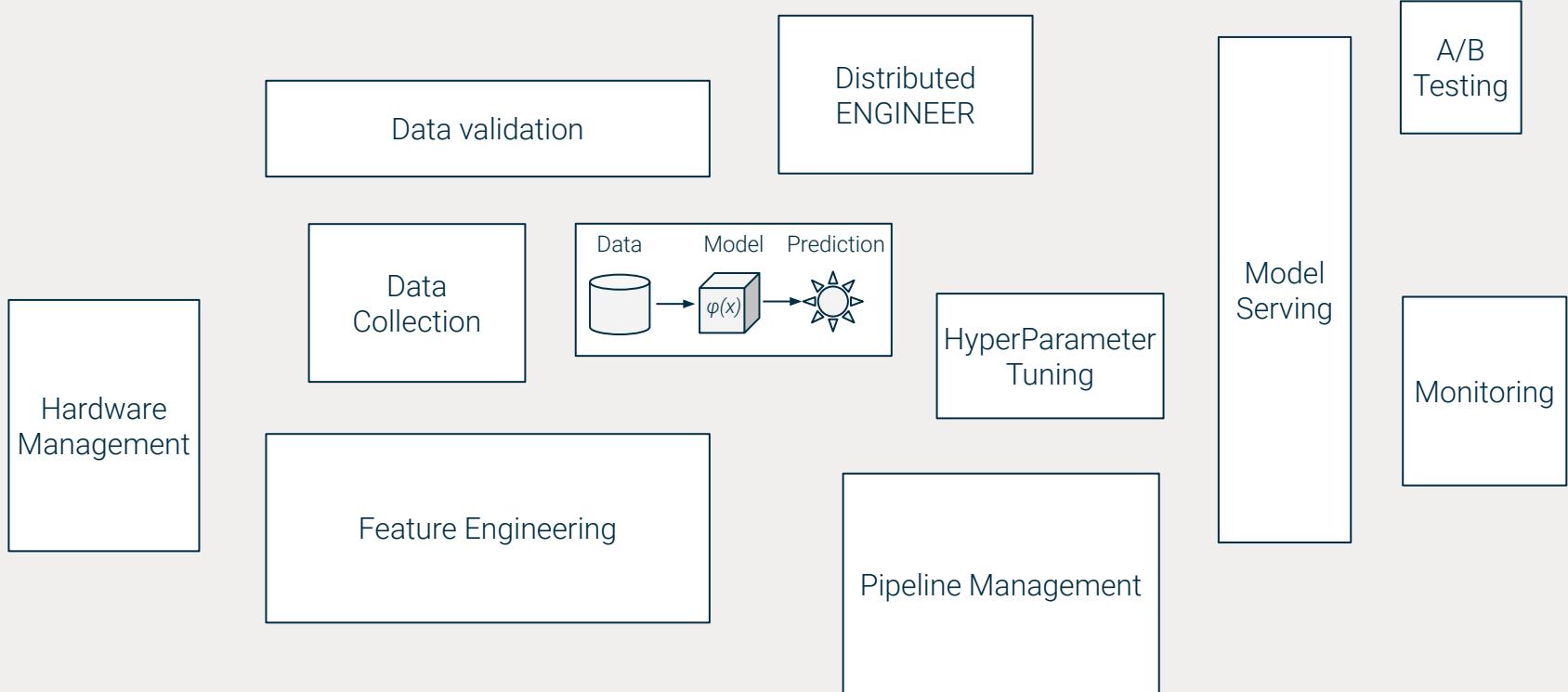
KTH Lecture for ID2223

Dr. Jim Dowling^{1,2}

Slides together with Alexandru A. Ormenisan^{1,2}, Mahmoud Ismail^{1,2}

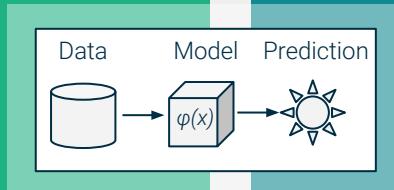


Growing Consensus on how to manage complexity of AI



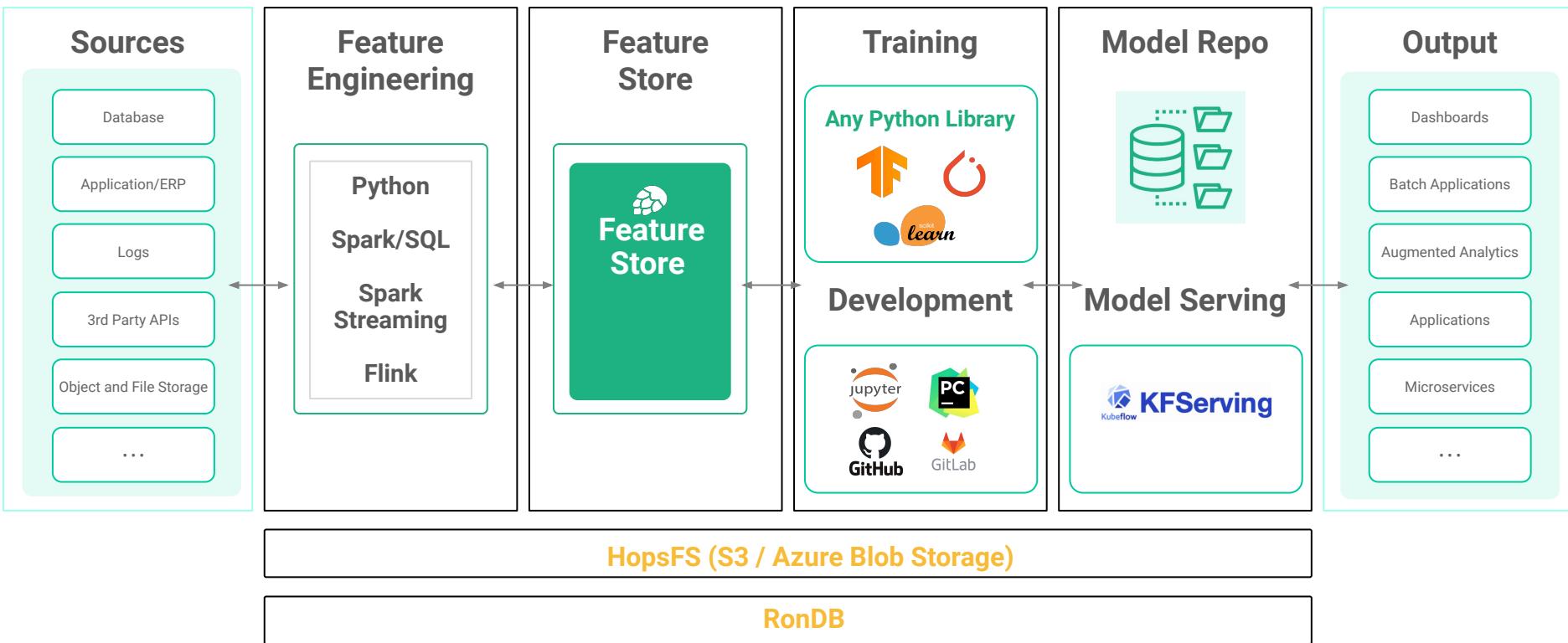


FEATURE STORE



ML PLATFORM TRAIN and SERVE

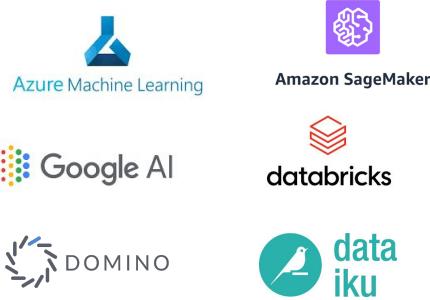
Hopsworks - Design and Operate AI Applications at Scale



Hopsworks is an Open, Modular Feature Store



Data Science



Teams use the tools of their choice,
integrated with the
Hopsworks Feature Store

Data Engineering



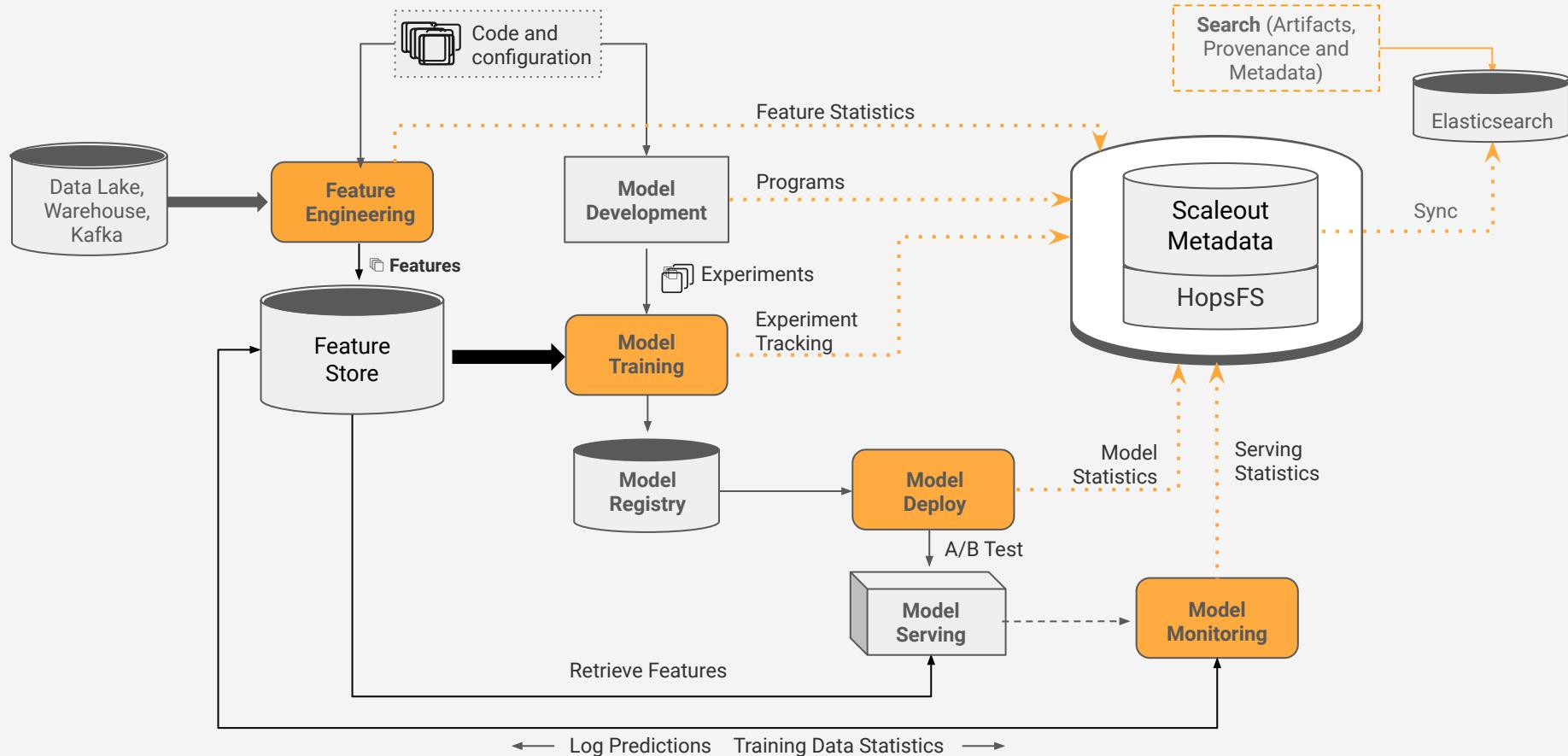
Model Serving



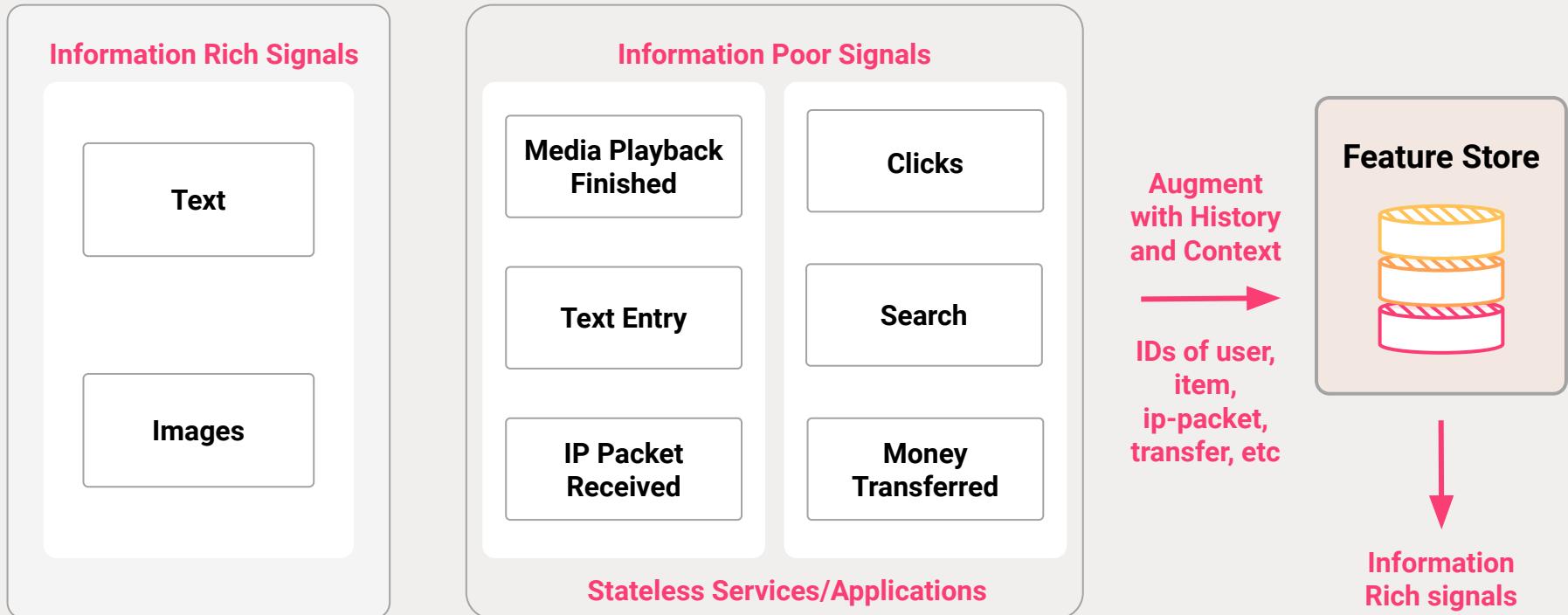
Compliance & Regulatory



Hopsworks End-to-End Machine Learning (ML) Pipelines



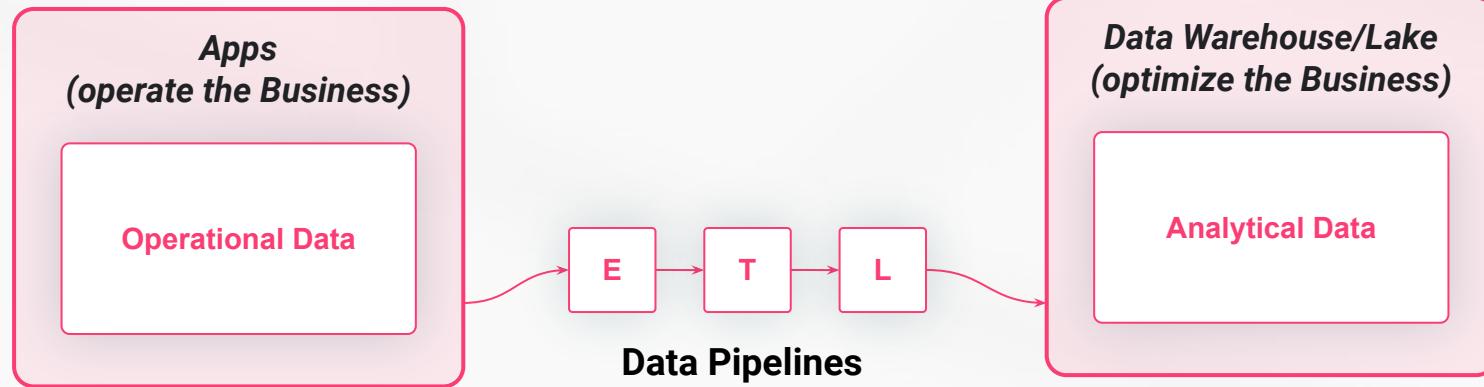
The Feature Store - From Information Poor Signals to Information Rich Signals



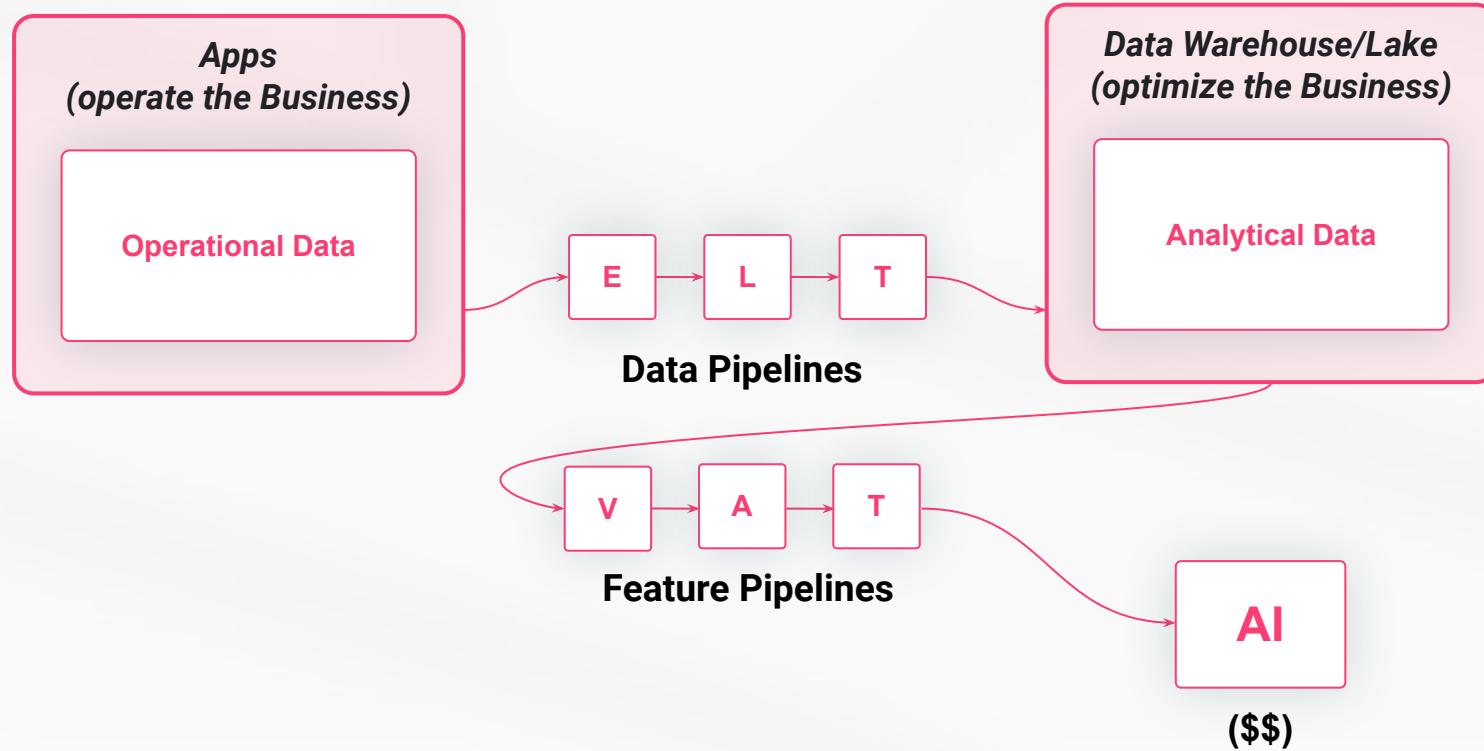
Commercial Feature Stores for Machine Learning



How the feature store fits in your data infrastructure

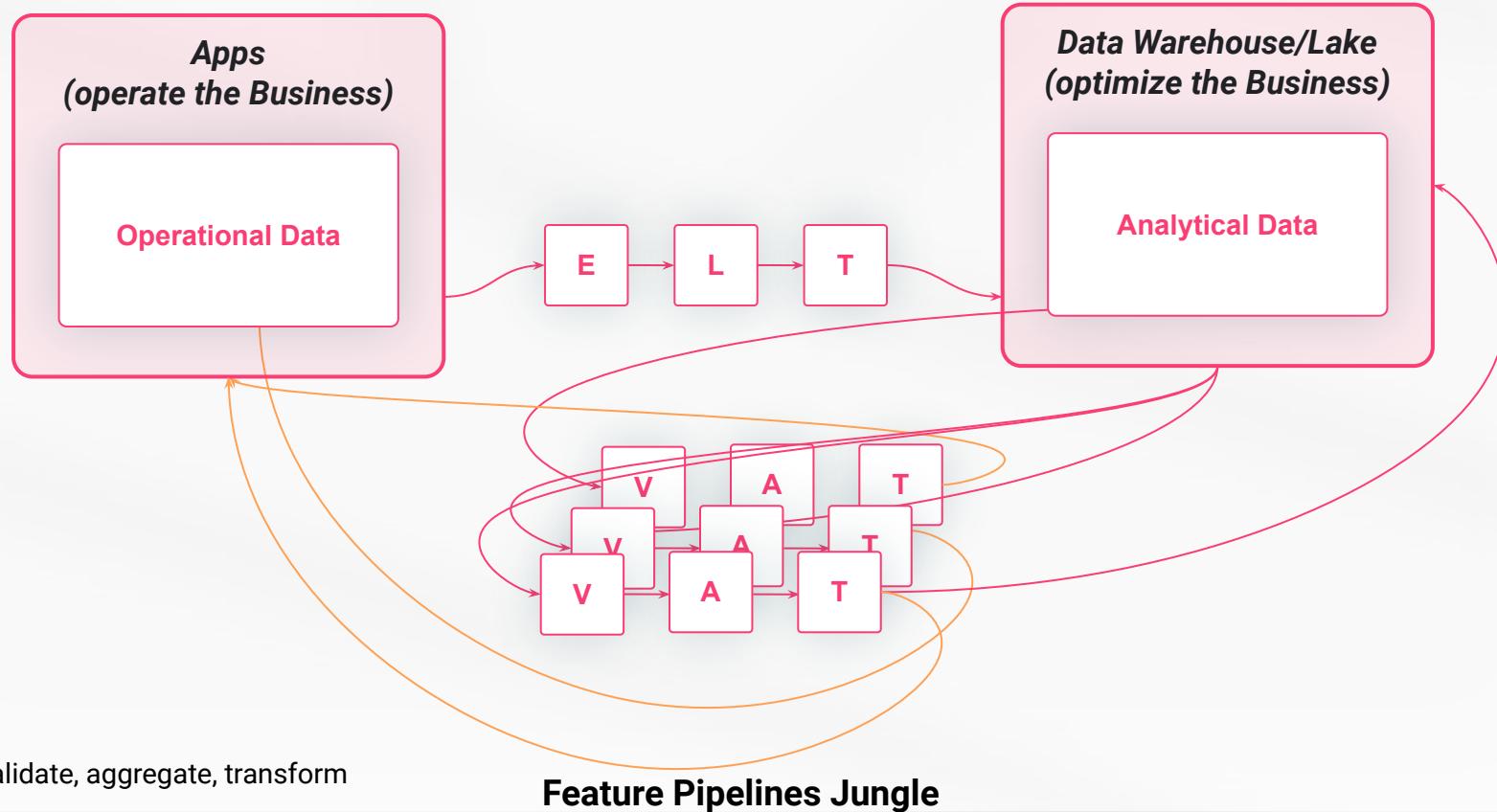


How the feature store fits in your data infrastructure

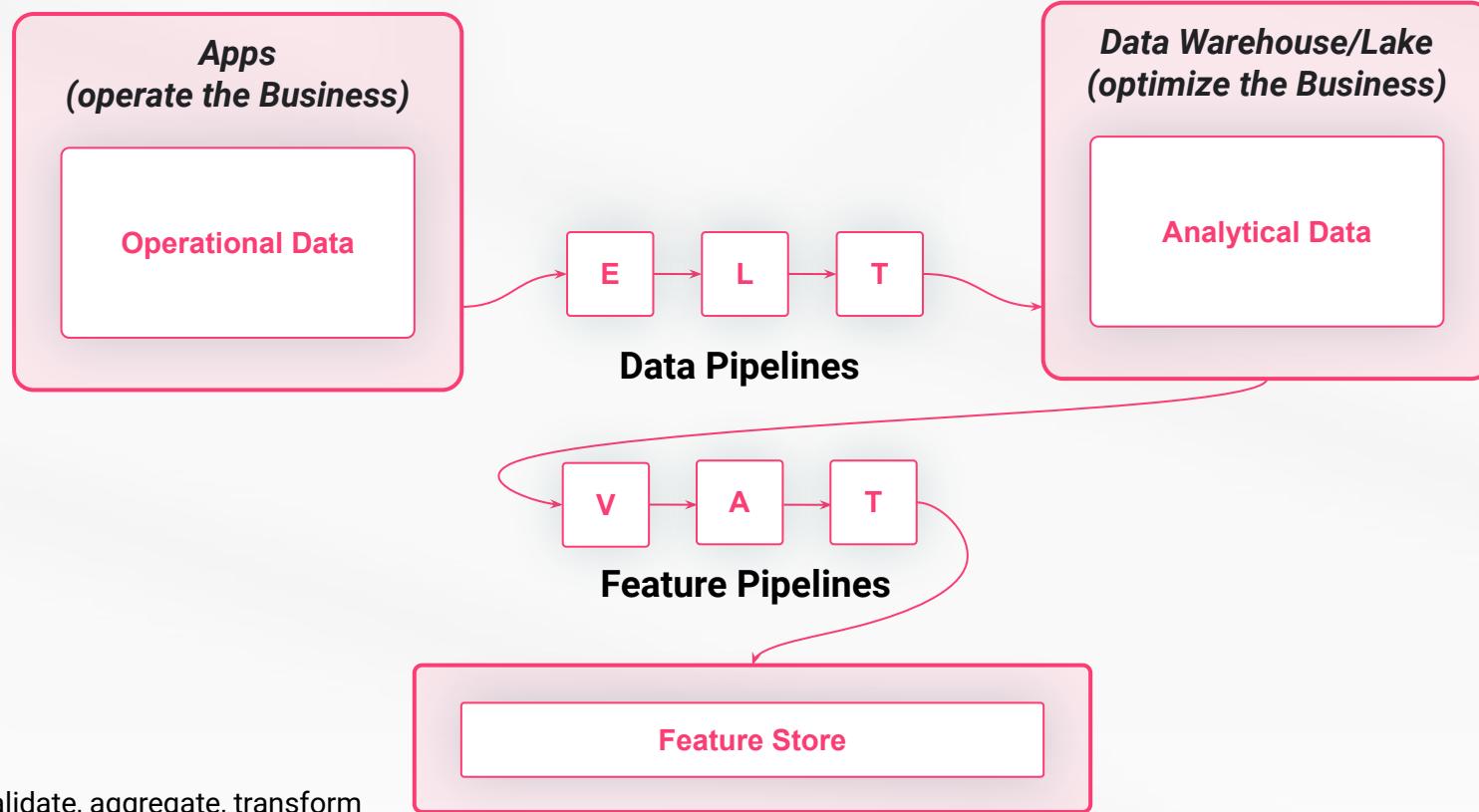


VAT = validate, aggregate, transform

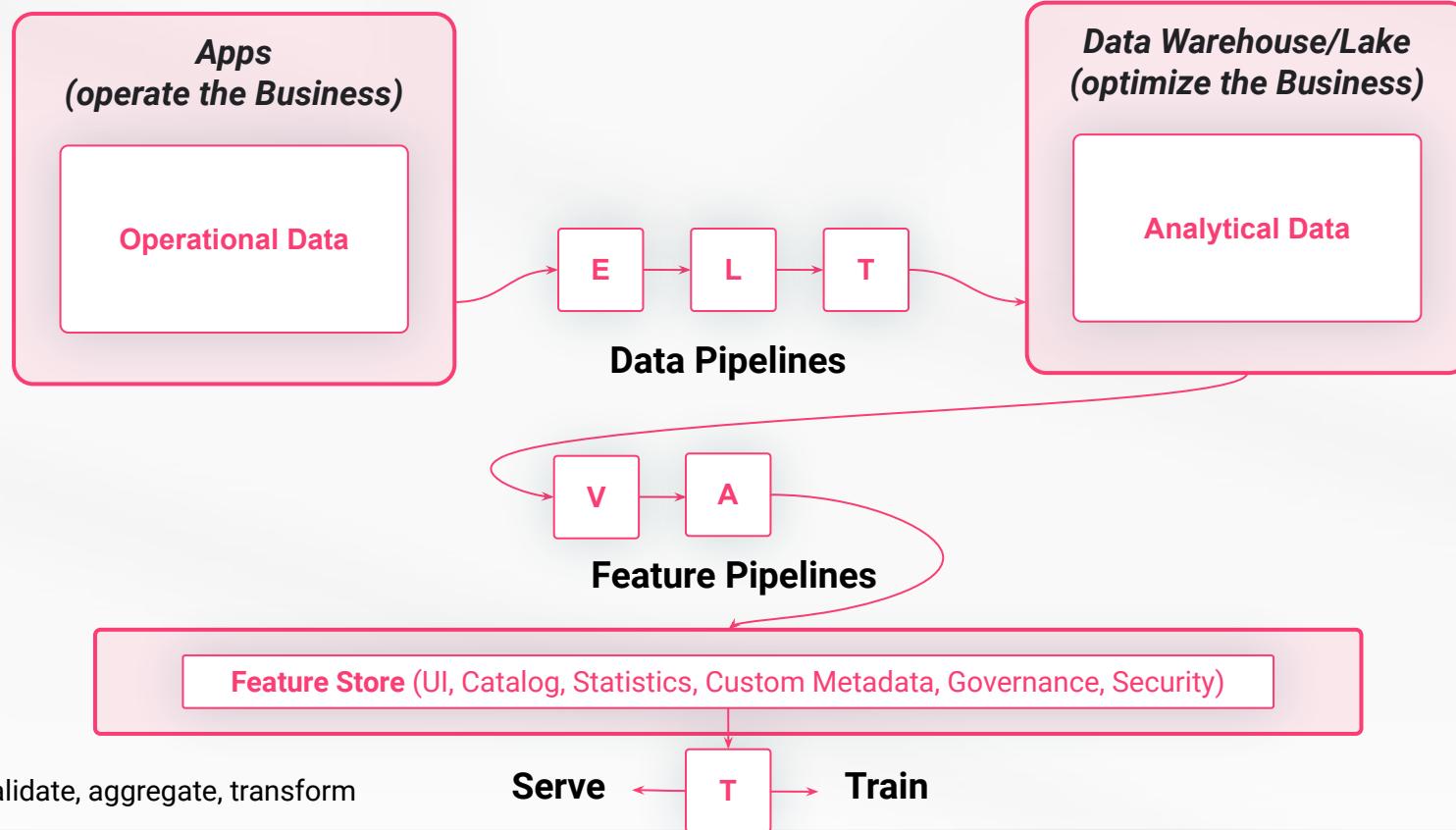
How the feature store fits in your data infrastructure



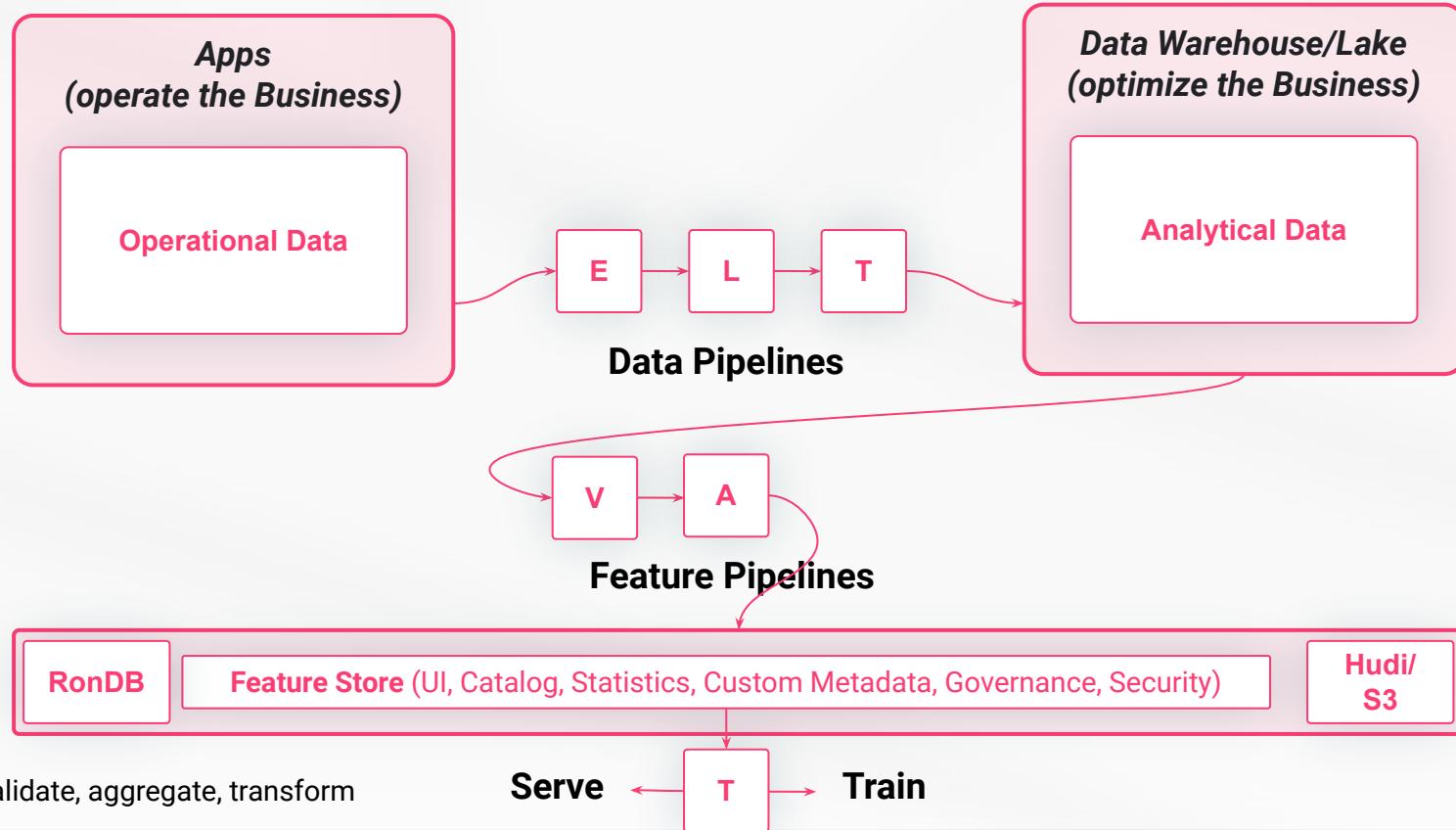
How the feature store fits in your data infrastructure



How the feature store fits in your data infrastructure



How the feature store fits in your data infrastructure



Enables Collaboration between folks who speak different languages



Data Scientist
Python



Data Engineer
SQL, Spark, Flink, Python

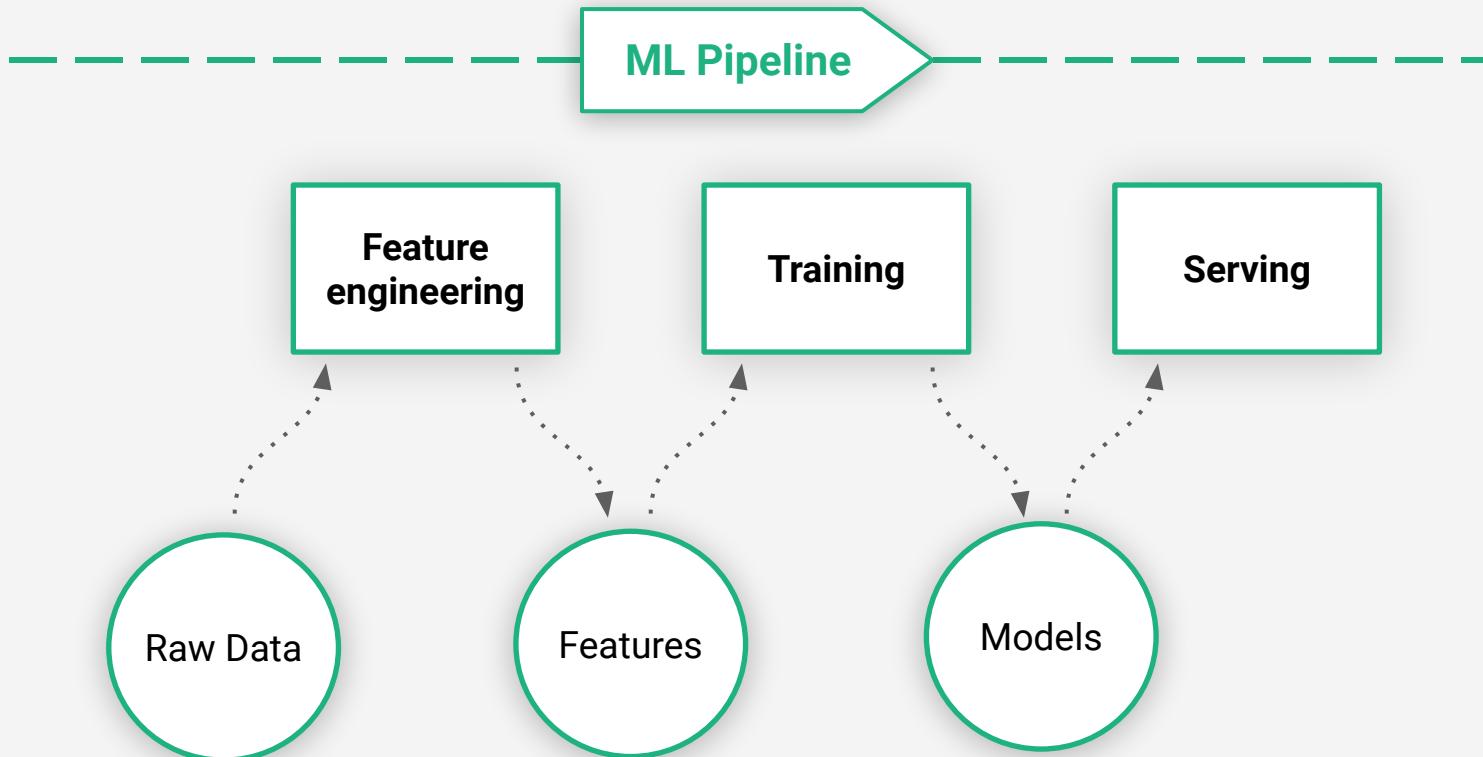


ML Engineer
Kubernetes, Serverless

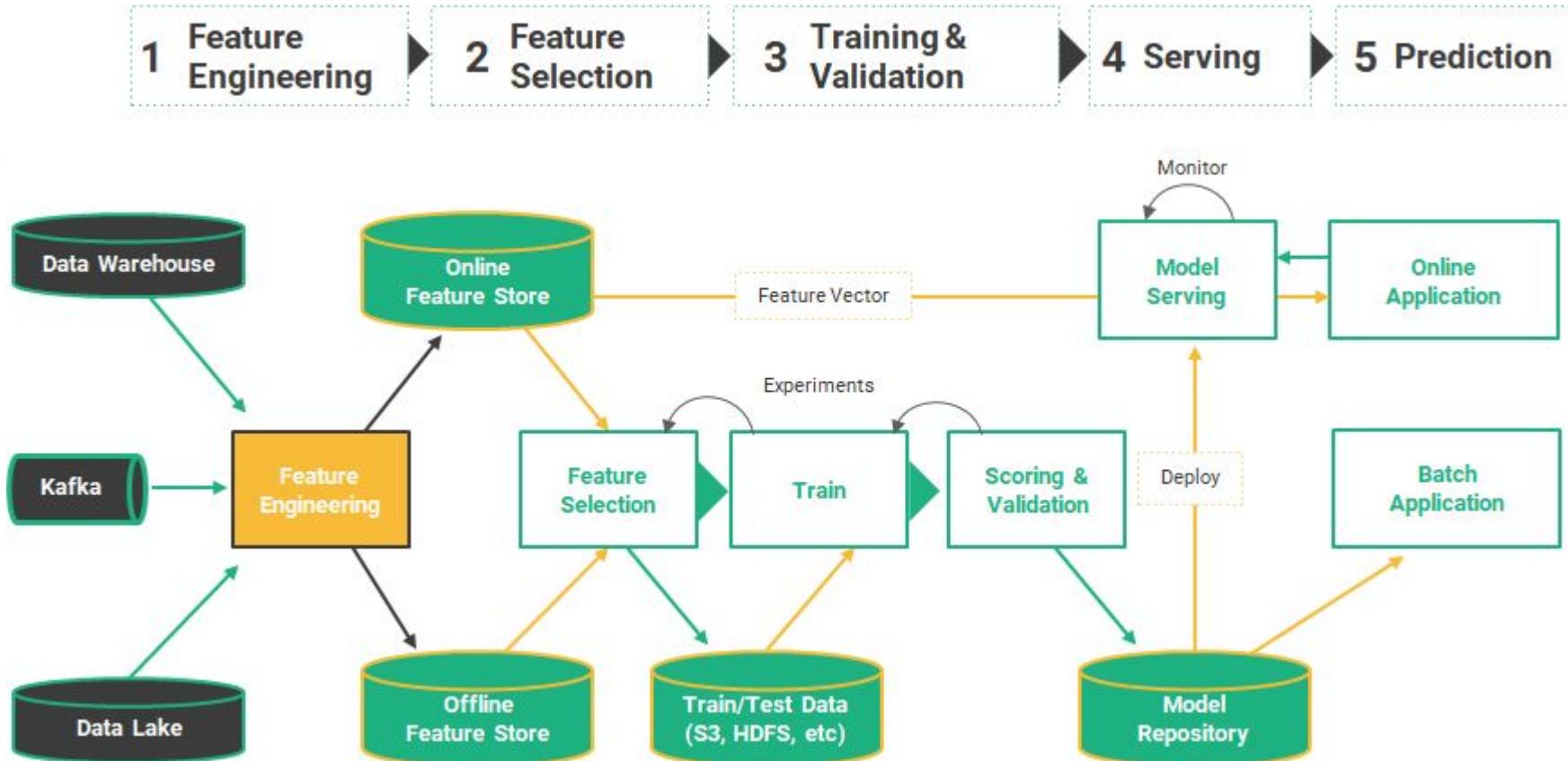
Road To AI Value



What are ML Pipelines?



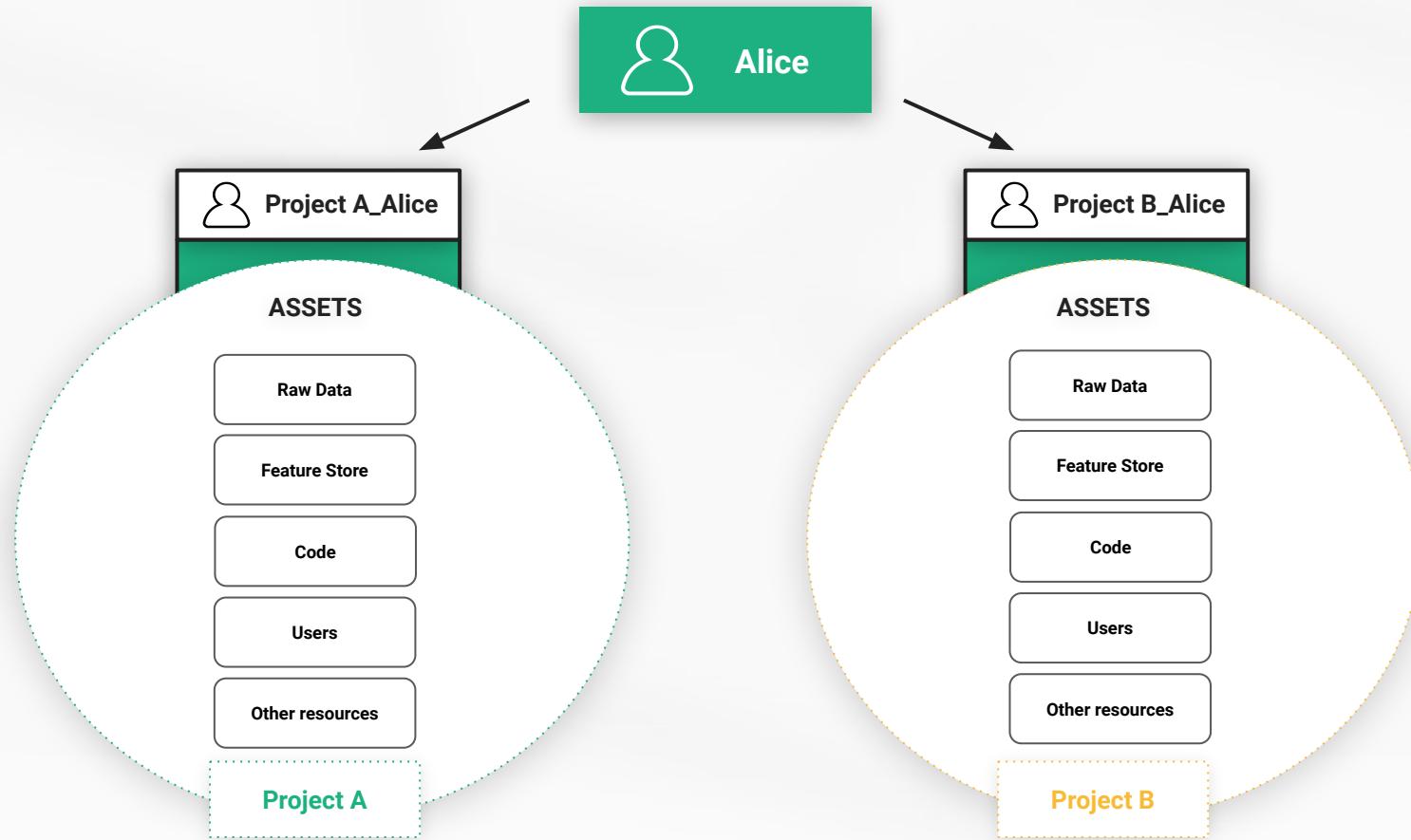
End-to-End Machine Learning (ML) Pipelines



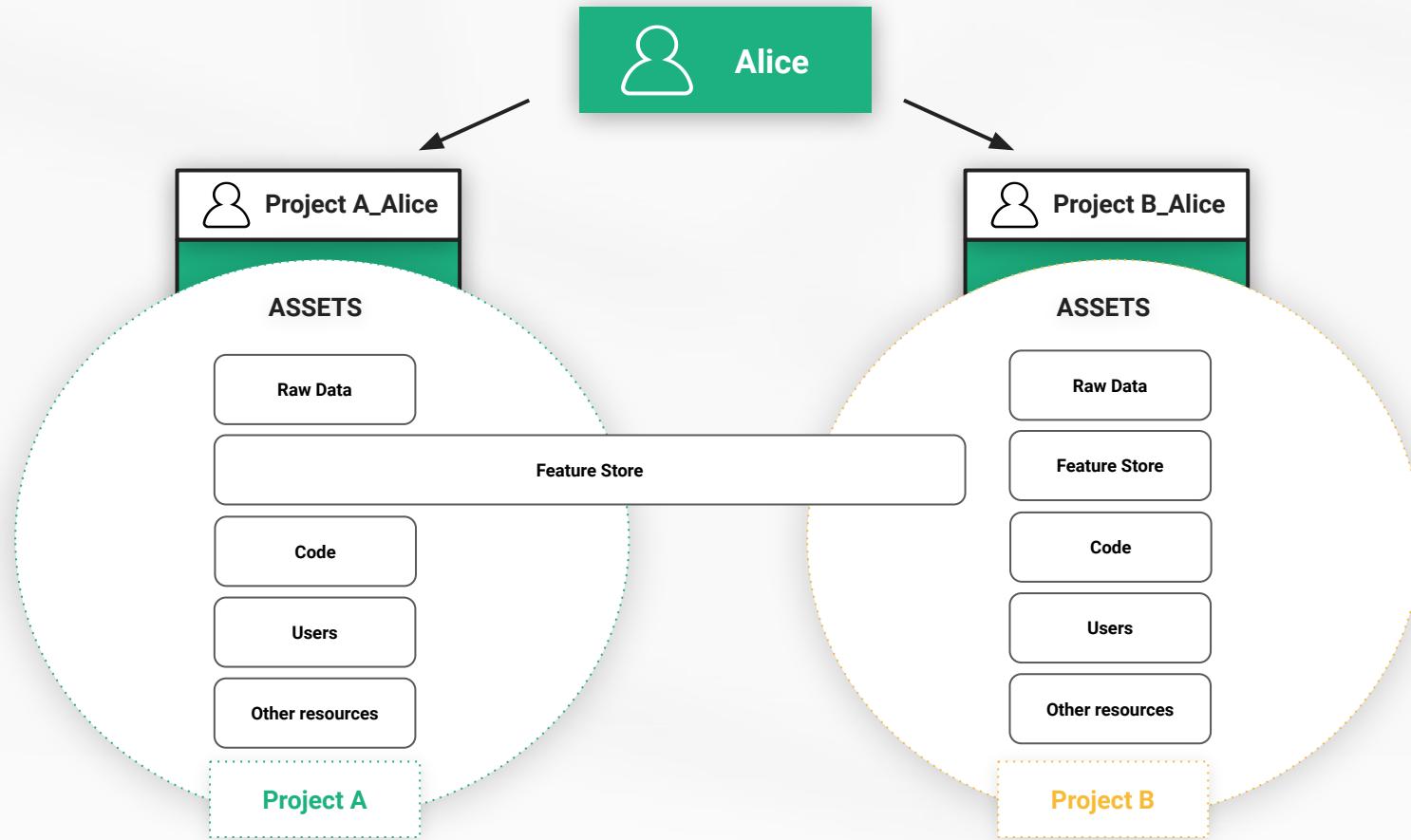


How it Started

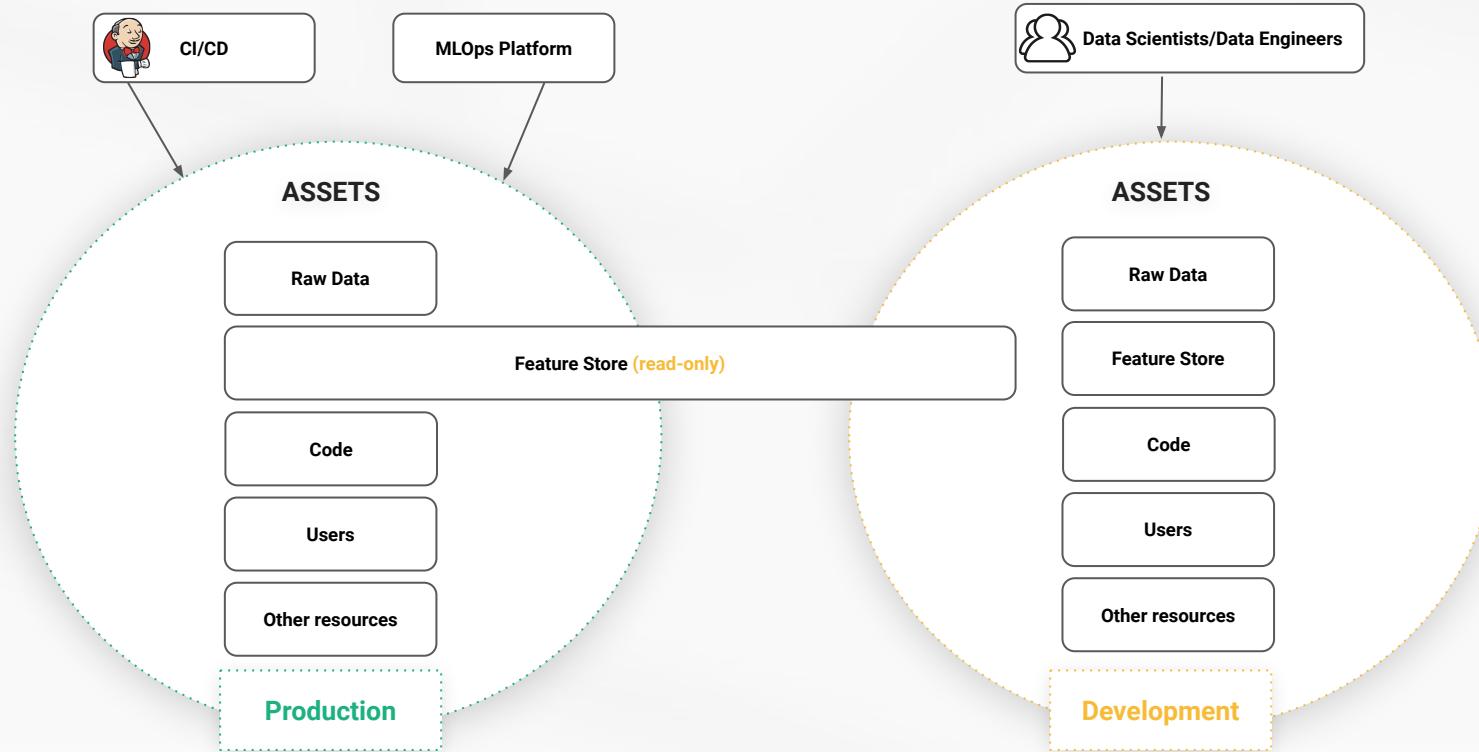
Project Based Multi Tenancy



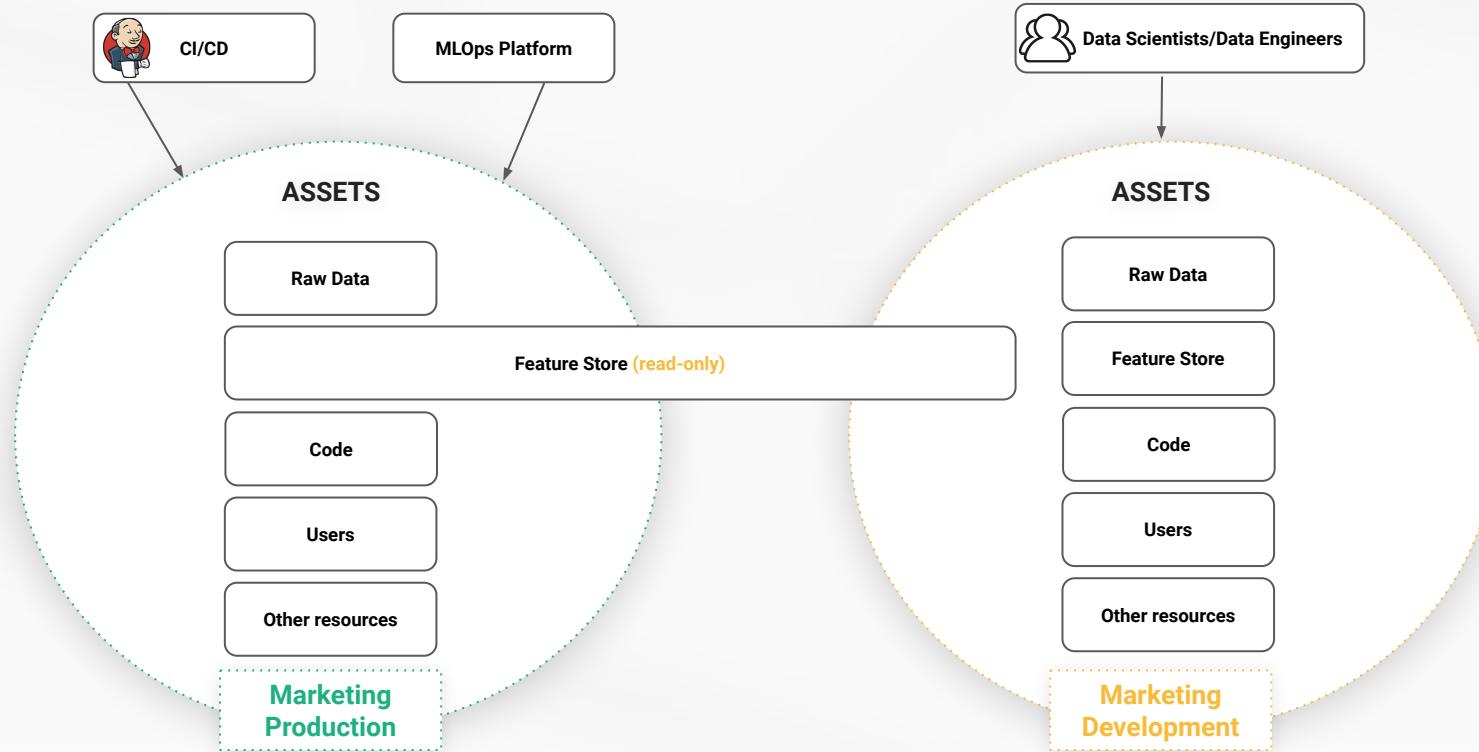
Project Based Multi Tenancy - Shared Data



Project Based Multi Tenancy - Production/Development



Project Based Multi Tenancy - Mix Structure

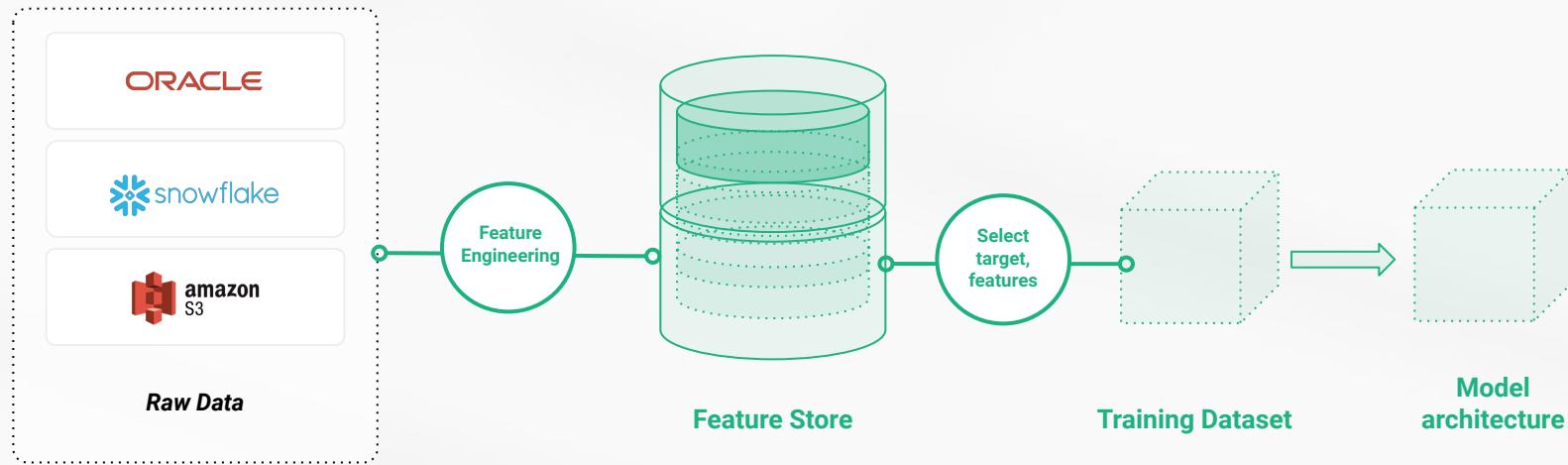




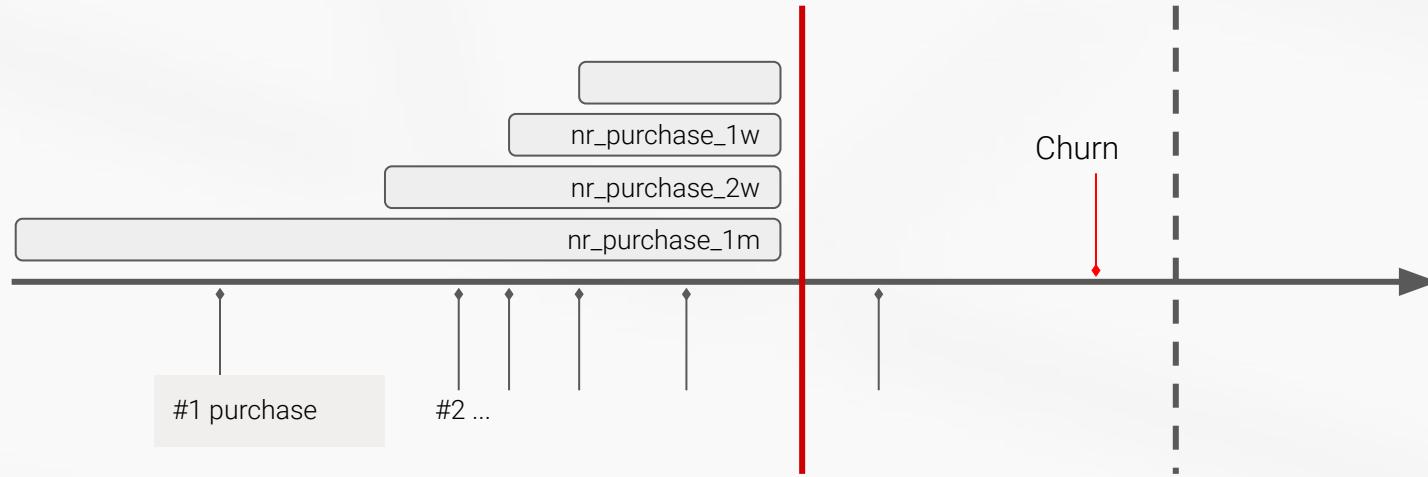
HOPSWORKS

Feature Engineering

From Data to Features to Training Data to Models



Feature engineering - Aggregation example



Transaction

transaction_time

2021/02/10

id

23

Feature Aggregation

2021/01/24

42

2021/01/20

47

Now

Features

Entity

1

1w

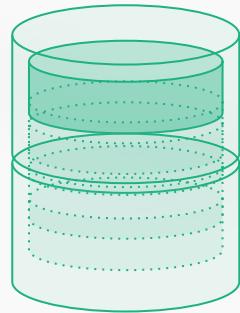
1

1m

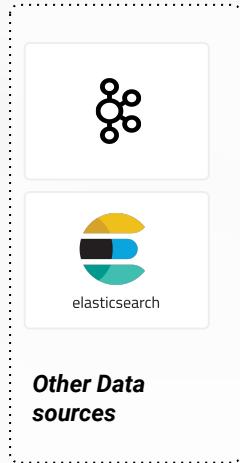
2



Feature engineering

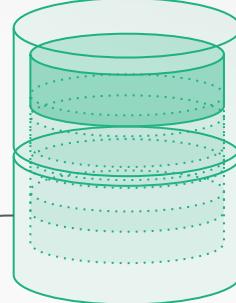
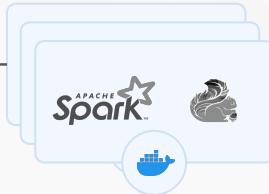


Existing feature groups

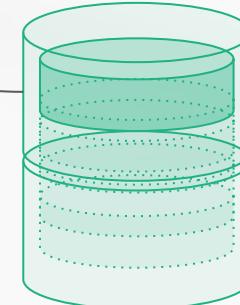


Other Data sources

```
Dataframe = (Python/PySpark/Spark/Flink based feature engineering)  
  
fg = fs.create_feature_group("churn",  
    version=1,  
    description="Customer information about activity of contract",  
    online_enabled=True,  
    primary_key=["customer_id", "contract_id"],  
    event_time="ts")  
  
fg.save(dataframe)
```

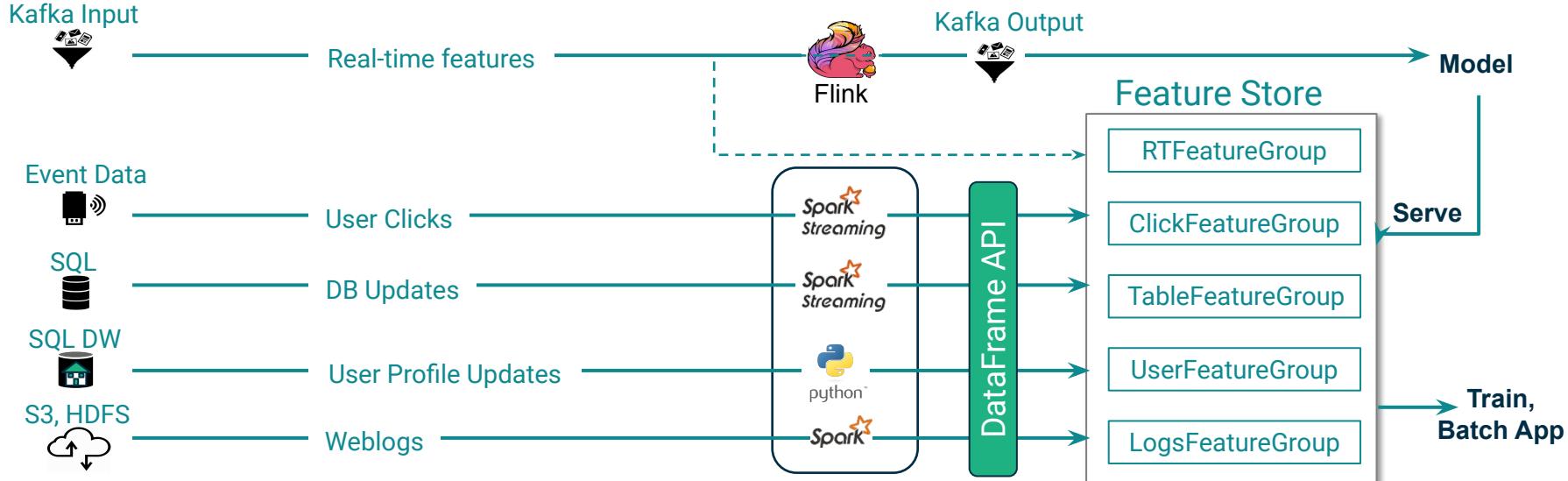


Online feature store



Offline feature store

Reusable Features are stored in FeatureGroups



Feature Pipelines update the Feature Store (2 Databases!) with data from backend Platforms



Low
Latency
Features

Real-Time Data



User-Entered Features (<2
secs)

Event Data



Click features every 10
secs



CDC data every 30
secs

SQL DW



User profile updates every
hour

S3, HDFS



Featurized weblogs data every
day

Feature Store

Online
Feature
Store

Offline
Feature
Store

Online
App

<10ms

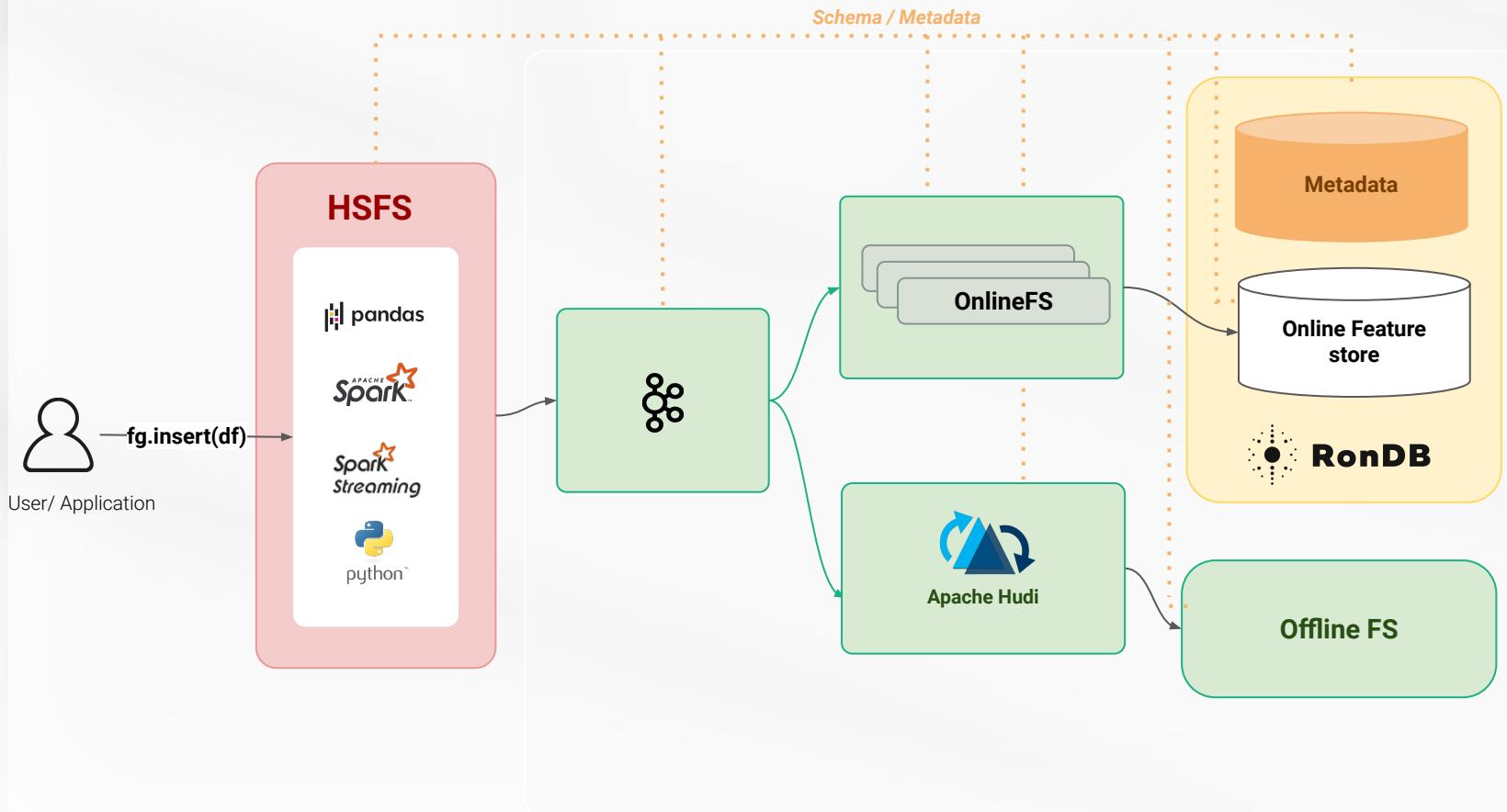
Train,
Batch App

TBs/PBs

High
Latency
Features

No existing database is both scalable (PBs) and low latency (<10ms). Hence, online + offline Feature Stores.

Streaming Applications can write Fresh Features





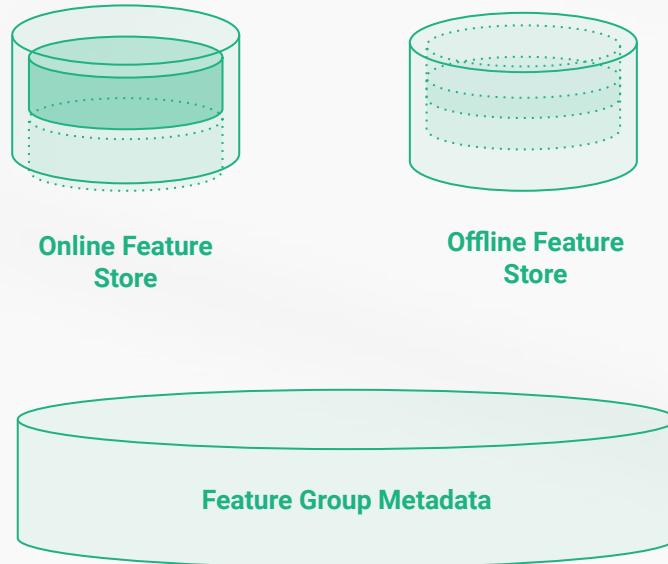
HOPSWORKS

Feature Groups

Cached feature groups



- Feature groups stored on Hopsworks
- Can be available both offline and online.
- Documentation:
https://docs.hopsworks.ai/feature-store-api/latest/generated/feature_group/
- Example:
https://examples.hopsworks.ai/feature_restore/hsfs/basics/feature_engineering/





Schema versioning

- Each feature group has a version number
- Version numbers allow users to identify breaking changes to the schema (feature dropped, change in the way a feature is being computed)
- Appending feature to a feature group is not considered a breaking change

Data versioning

- Calling `insert()/save()` on a feature group generates a new data commit.
- Data commits allow users to track how the data changed during the lifetime of a feature group.
- Users can navigate the commit history using the [Activity UI](#)
- Using the `as_of` method, users can retrieve features from a feature group, at a specific point in time.

Metadata - Activity



List actions performed on a feature group:

- Feature group creation
- Data ingestion
- Statistics computation
- Data validation

The screenshot shows the Hopsworks Metadata interface for a feature group named "cc_fraud". The left sidebar has a tree view with "card_transactions" expanded. The main area displays a log of events for "card_transactions" on "2021-10-12". The log shows seven "Data ingestion" events, each with a timestamp, commit ID, and statistics (rows updated/deleted). The last event is highlighted in green.

Time	Action	Commit	Statistics
2021-10-12 18:49:33	Data ingestion	commit 2021-10-12 18:49:33	0 new rows, 43k updated rows, 0 deleted rows
2021-10-12 18:48:32	Data ingestion	commit 2021-10-12 18:48:32	0 new rows, 6 updated rows, 0 deleted rows
2021-10-12 18:44:26	Data ingestion	commit 2021-10-12 18:44:26	0 new rows, 43k updated rows, 0 deleted rows
2021-10-12 18:41:15	Data ingestion	commit 2021-10-12 18:41:15	0 new rows, 1 updated rows, 0 deleted rows
2021-10-12 16:01:12	Data ingestion	commit 2021-10-12 16:01:12	0 new rows, 43k updated rows, 0 deleted rows
2021-10-12 15:57:58	Data ingestion	commit 2021-10-12 15:57:58	0 new rows, 7 updated rows, 0 deleted rows
2021-10-12 15:27:21	Data ingestion	commit 2021-10-12 15:27:21	57 new rows, 43k updated rows, 0 deleted rows
2021-10-12 15:21:36	Data ingestion	commit 2021-10-12 15:21:36	87k new rows, 0 updated rows, 0 deleted rows

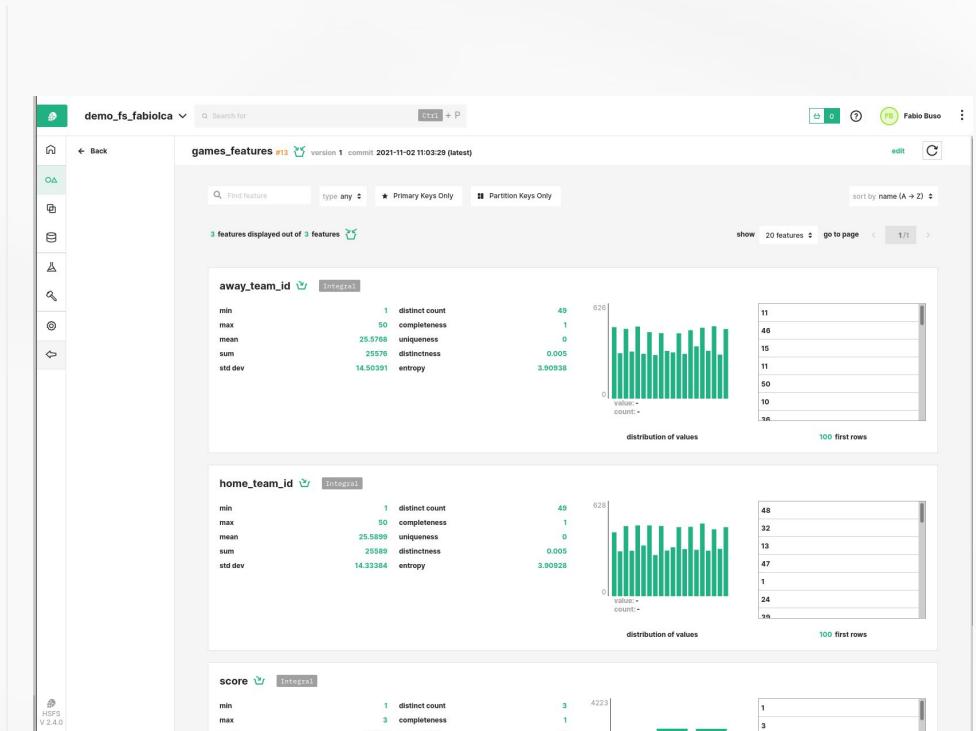
Metadata - Statistics



Statistics are computed at feature group level for each [data commit](#)

Hopworks computes automatically:
descriptive statistics, histograms and
correlations between features

Statistics can be explored from the UI.





Tags allow users to specify arbitrary metadata and make it searchable throughout the feature store.

Tags require a schema that can be defined and enforced at platform level.

Tags can be manipulated through the UI or through the [APIs](#)

Tags		edit
pii		edit properties
pii	true	
privacy-officer	Fabio Buso	

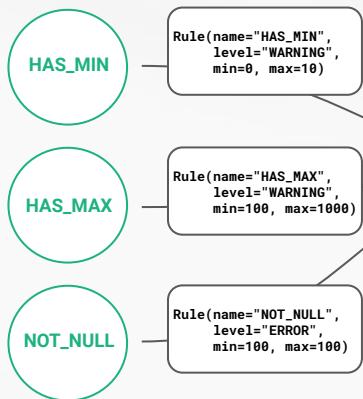


HOPSWORKS

Data Validation



Rules



Expectation



Feature Groups



These are the rules supported by the platform:
https://docs.hopworks.ai/feature-store-api/latest/generated/feature_validation/#rule-definitions

An expectation is a set of rules with specific values and rule severity

Expectations can be applied to specific features on a feature group, or to the entire set of features of a feature group

https://docs.hopworks.ai/feature-store-api/latest/generated/feature_validation/#rule-definitions

Validation types



Validation Type	Success	Warning	Failure
Strict	Insertion	Reject	Reject
Warning	Insertion	Insertion	Reject
All	Insertion	Insertion	Insertion
None	No data validation performed		

An expectation is a set of rules with specific values and rule severity

Expectations can be applied to specific features on a feature group, or to the entire set of features of a feature group

https://docs.hopsworks.ai/feature-store-api/latest/generated/feature_validation/#rule-definitions

```
Dataframe = (Python/PySpark/Spark/Flink based feature engineering)
fg = fs.create_feature_group("churn",
                             version=1,
                             description="Customer/contract information about activity of
contract",
                             validation_type="STRICT",
                             primary_key=["customer_id", "contract_id"])

fg.save(dataframe)
```

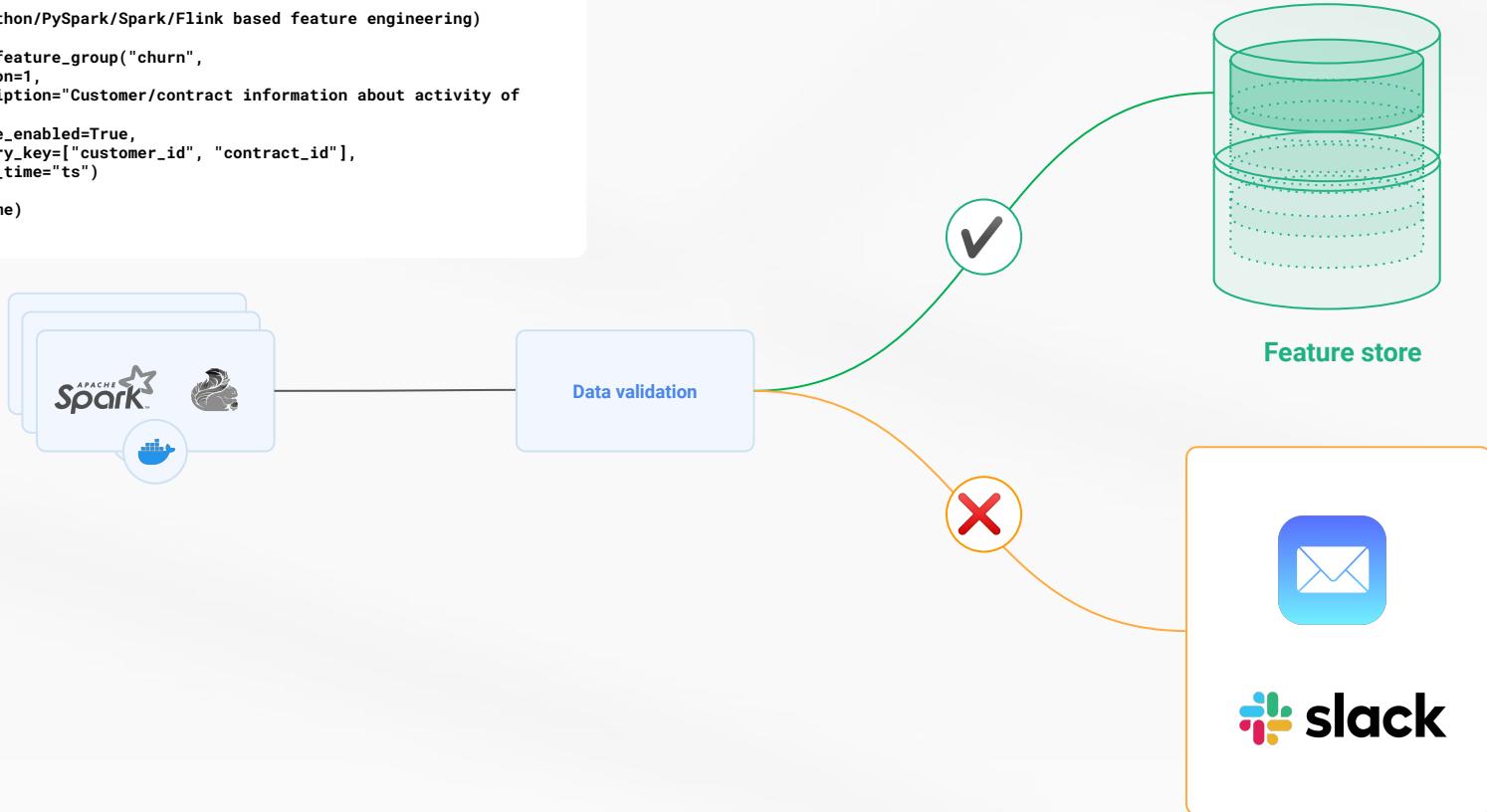
Data Validation



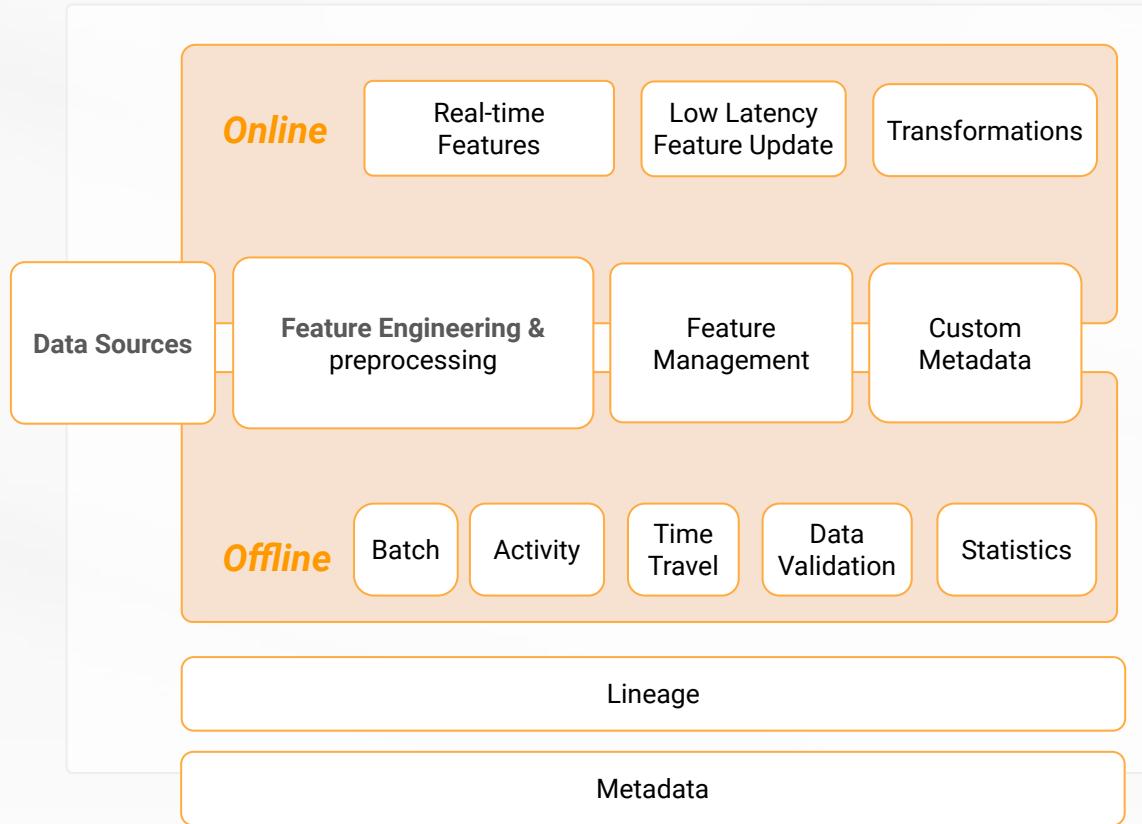
```
Dataframe = (Python/PySpark/Spark/Flink based feature engineering)

fg = fs.create_feature_group("churn",
    version=1,
    description="Customer/contract information about activity of
contract",
    online_enabled=True,
    primary_key=["customer_id", "contract_id"],
    event_time="ts")

fg.save(dataframe)
```

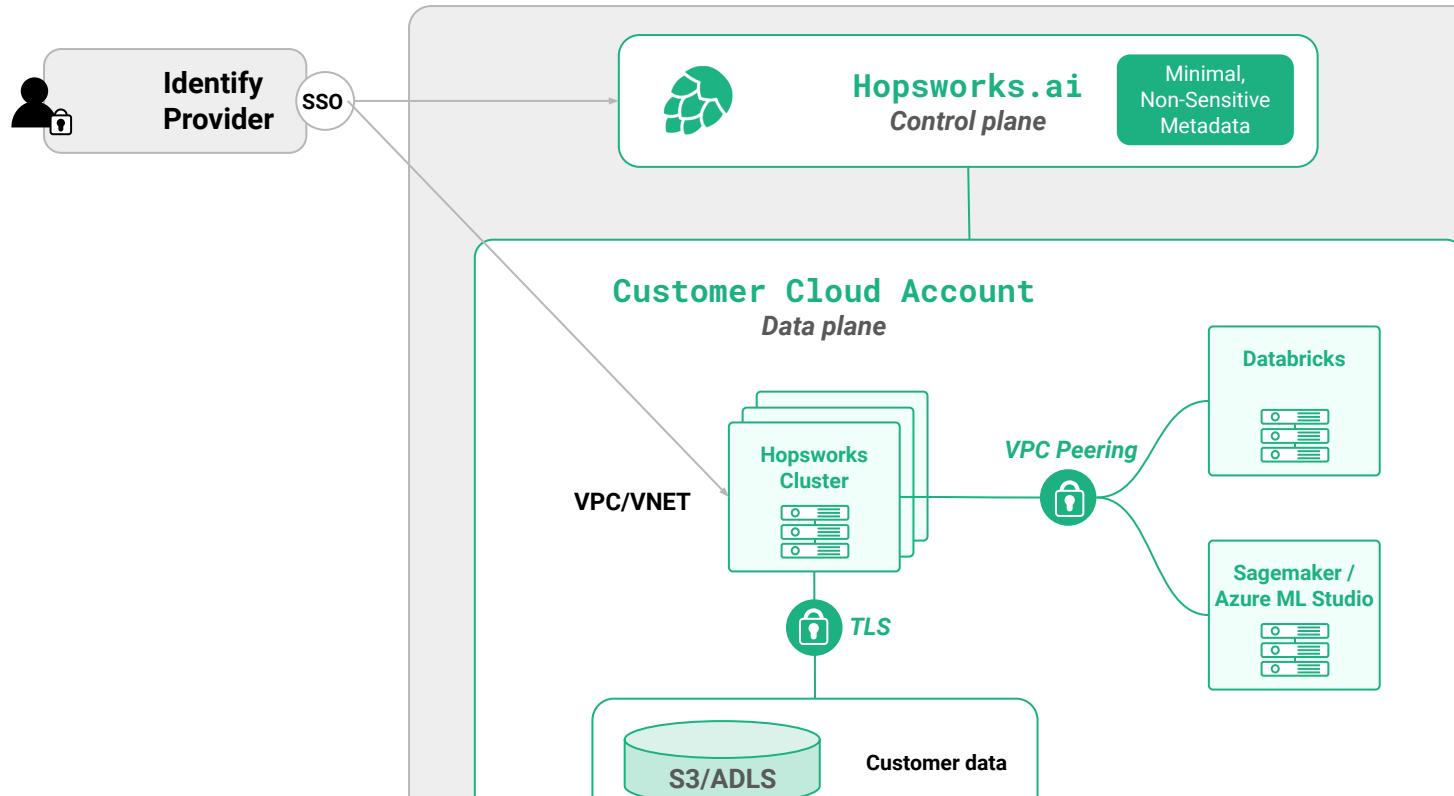


Hopsworks Feature Store

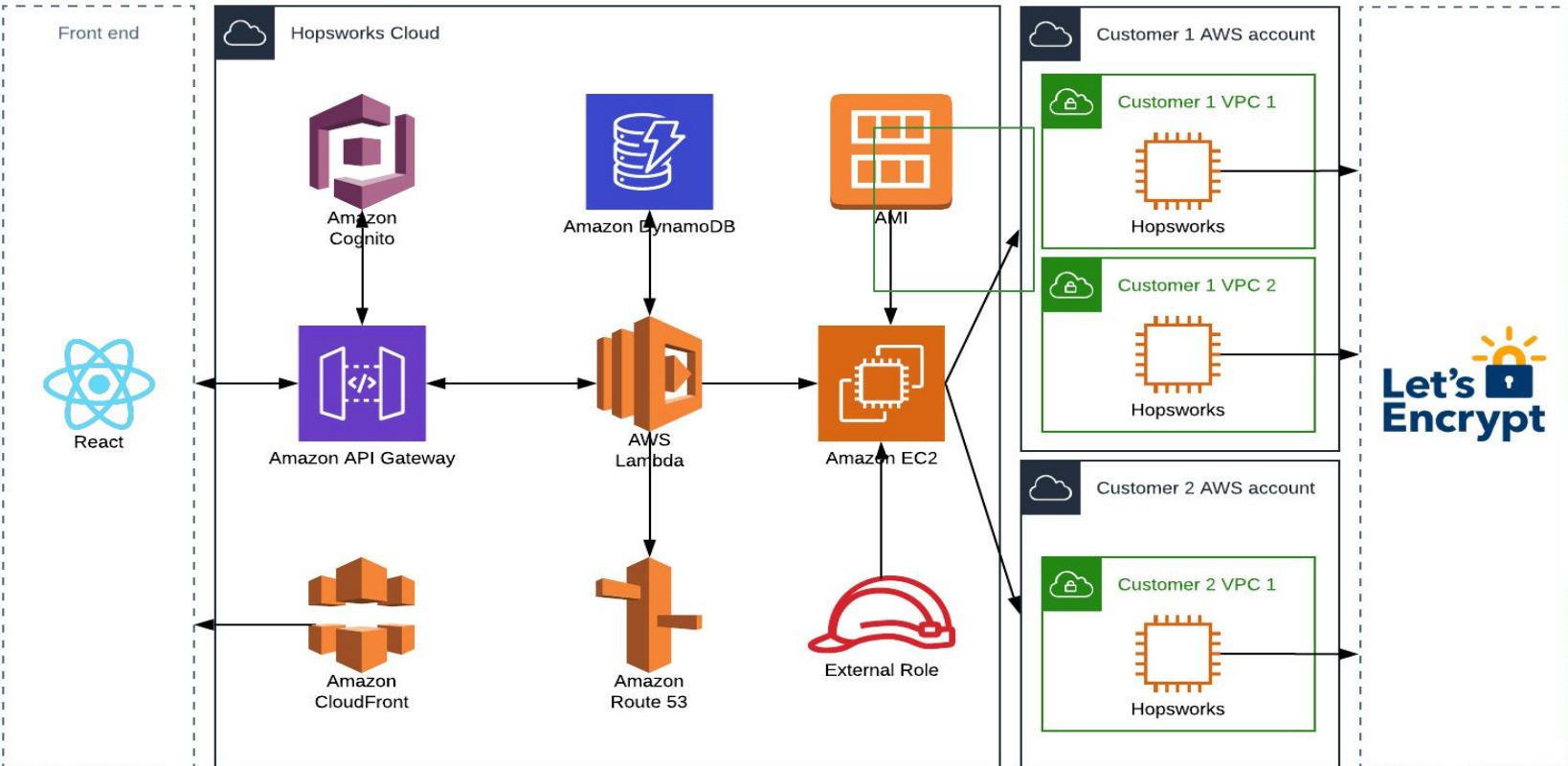


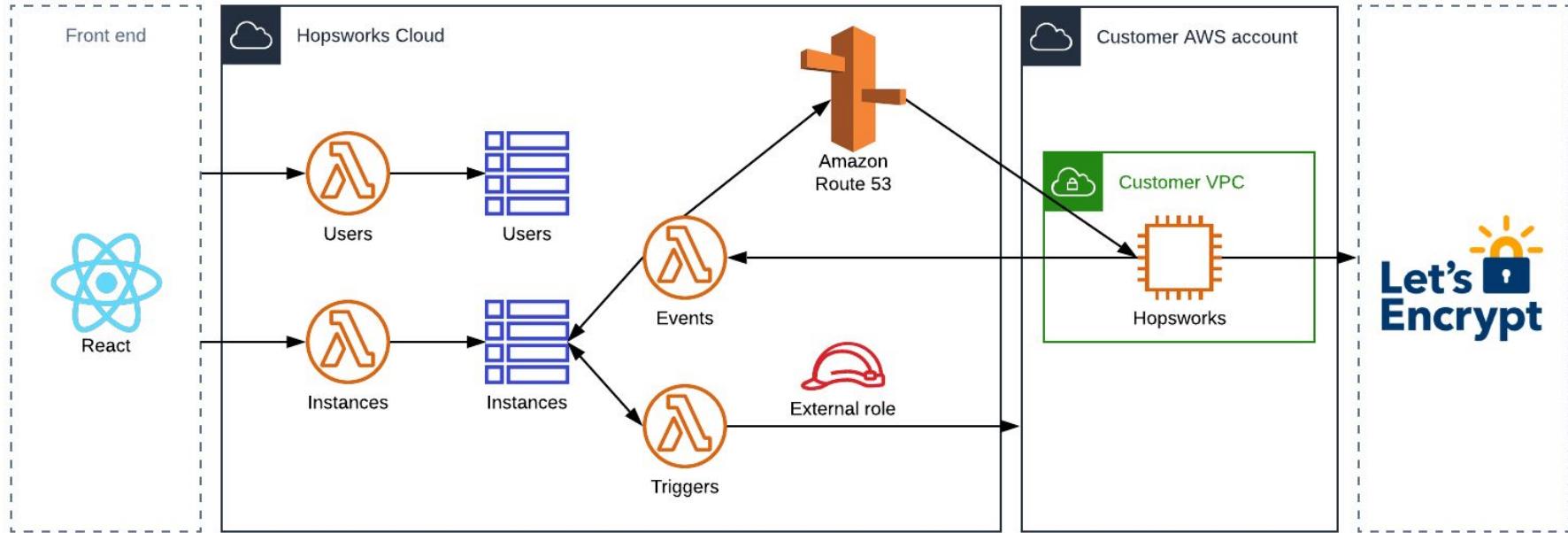


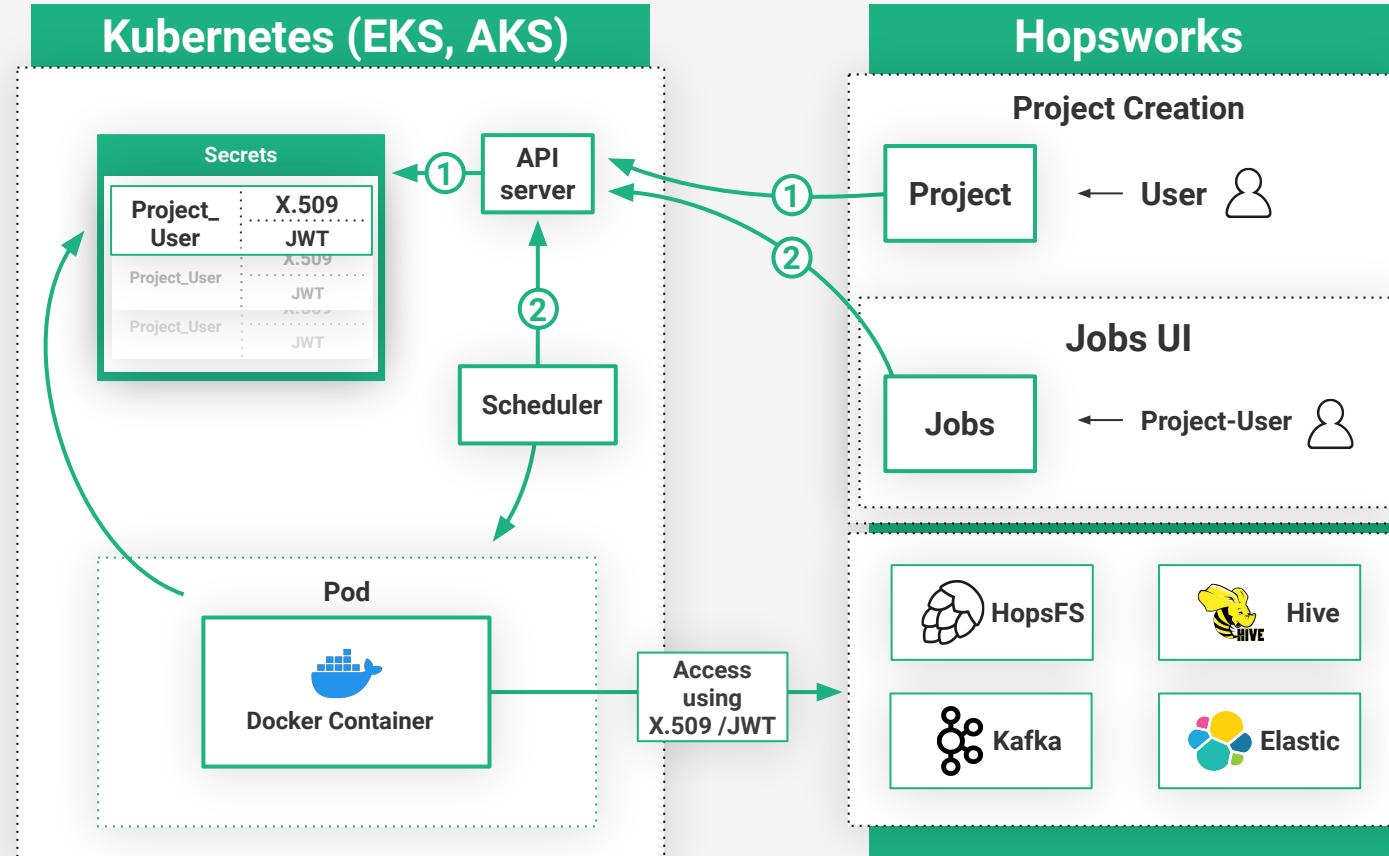
Hopsworks.ai
Deployment / Security



Serverless Platform on AWS - Amplify, Cognito, CloudFront, Lambdas, Route 53, DynamoDB



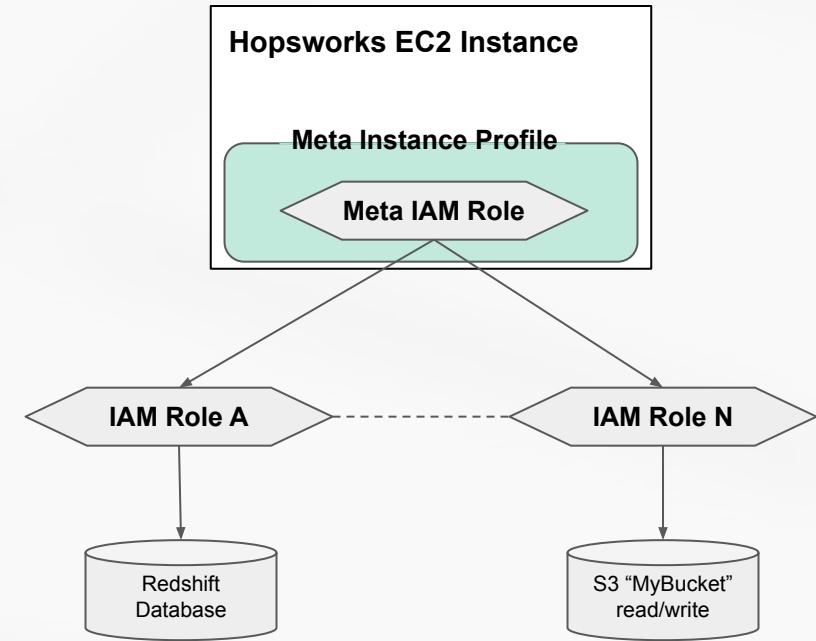
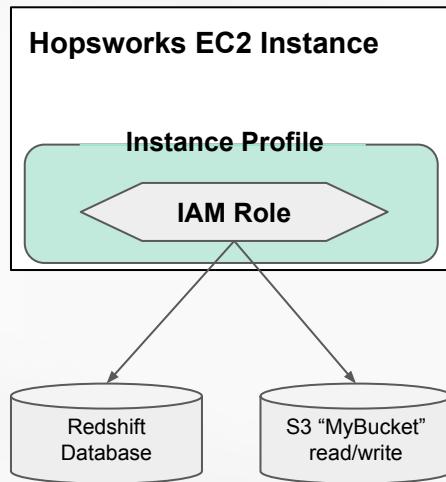




IAM Instance Profile

OR

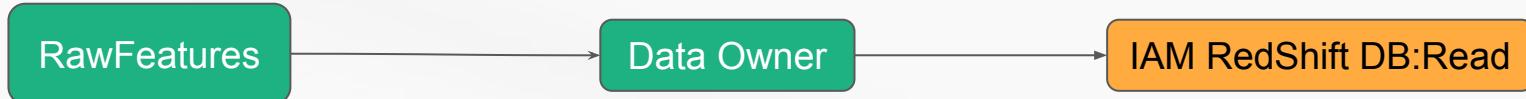
IAM Role Chaining



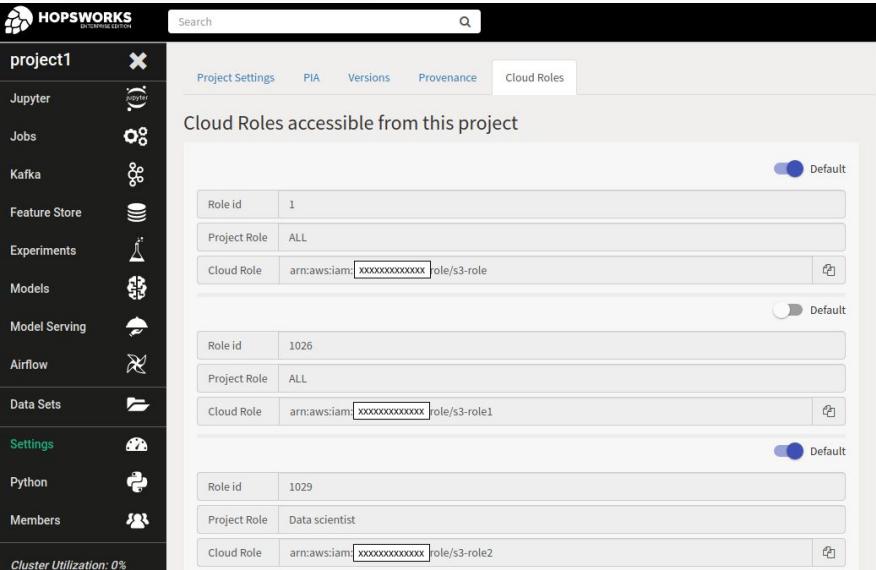
Configuring AWS IAM Role Chaining



Example: Only allow admins of Project 'RawFeatures' to read from Redshift



Assuming AWS IAM Roles



The screenshot shows the Hopsworks Enterprise Edition interface. On the left, a sidebar for 'project1' lists various services: Jupyter, Jobs, Kafka, Feature Store, Experiments, Models, Model Serving, Airflow, Data Sets, Settings, Python, and Members. Below the sidebar, it says 'Cluster Utilization: 0%'. At the top right, there's a search bar and a 'Cloud Roles' button. Under 'Cloud Roles accessible from this project', there are three entries:

- Role id: 1**
Project Role: ALL
Cloud Role: arn:aws:iam:xxxxxxxxxxxx:role/s3-role
- Role id: 1026**
Project Role: ALL
Cloud Role: arn:aws:iam:xxxxxxxxxxxx:role/s3-role1
- Role id: 1029**
Project Role: Data scientist
Cloud Role: arn:aws:iam:xxxxxxxxxxxx:role/s3-role2

Each role entry has a 'Default' toggle switch.

```
from hops.credentials_provider import get_role, assume_role
credentials = assume_role(role_arn=get_role(1))
spark.read.csv("s3a://resource/test.csv").show()
```

```
import io.hops.util.CredentialsProvider
val creds = CredentialsProvider.assumeRole(CredentialsProvider.getRole(1))
spark.read.csv("s3a://resource/test.csv").show()
```





Metadata is **data** that describes other **data**.



Metastore (Database)



Provenance queries

- SQL or Free-Text or Graph?
- Update Throughput?
- Latency of queries?
- Size of Metadata?

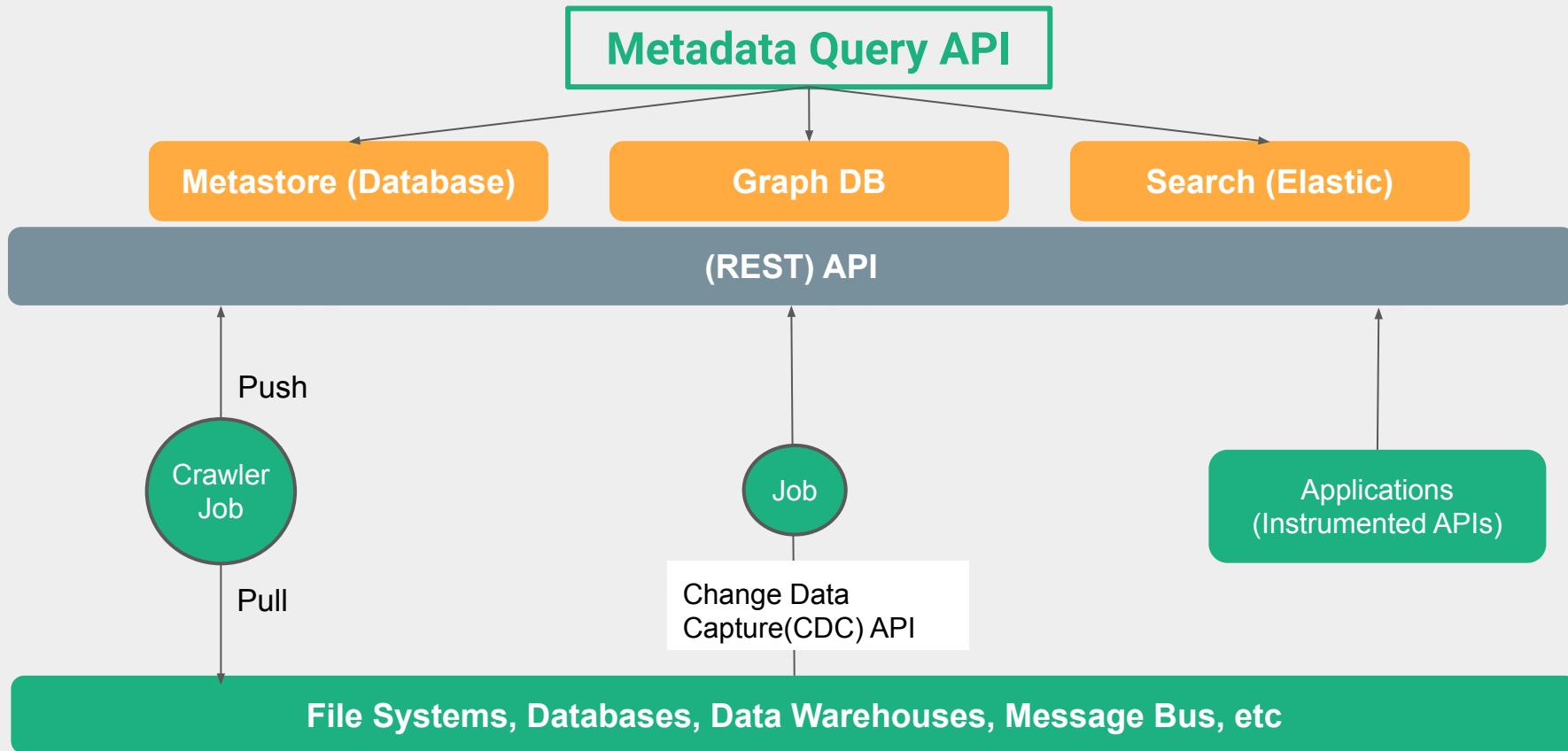
File System (S3, HopsFS, etc)

Metadata Cataloging Systems - a whole industry



<https://www.dataplatformschool.com/blog/w0y8q0-the-data-governance-zoo>

3 Mechanisms for Metadata Collection. Polyglot Metadata Storage for Efficient Querying.





Metastore



?

Consistency issues
Synchronization

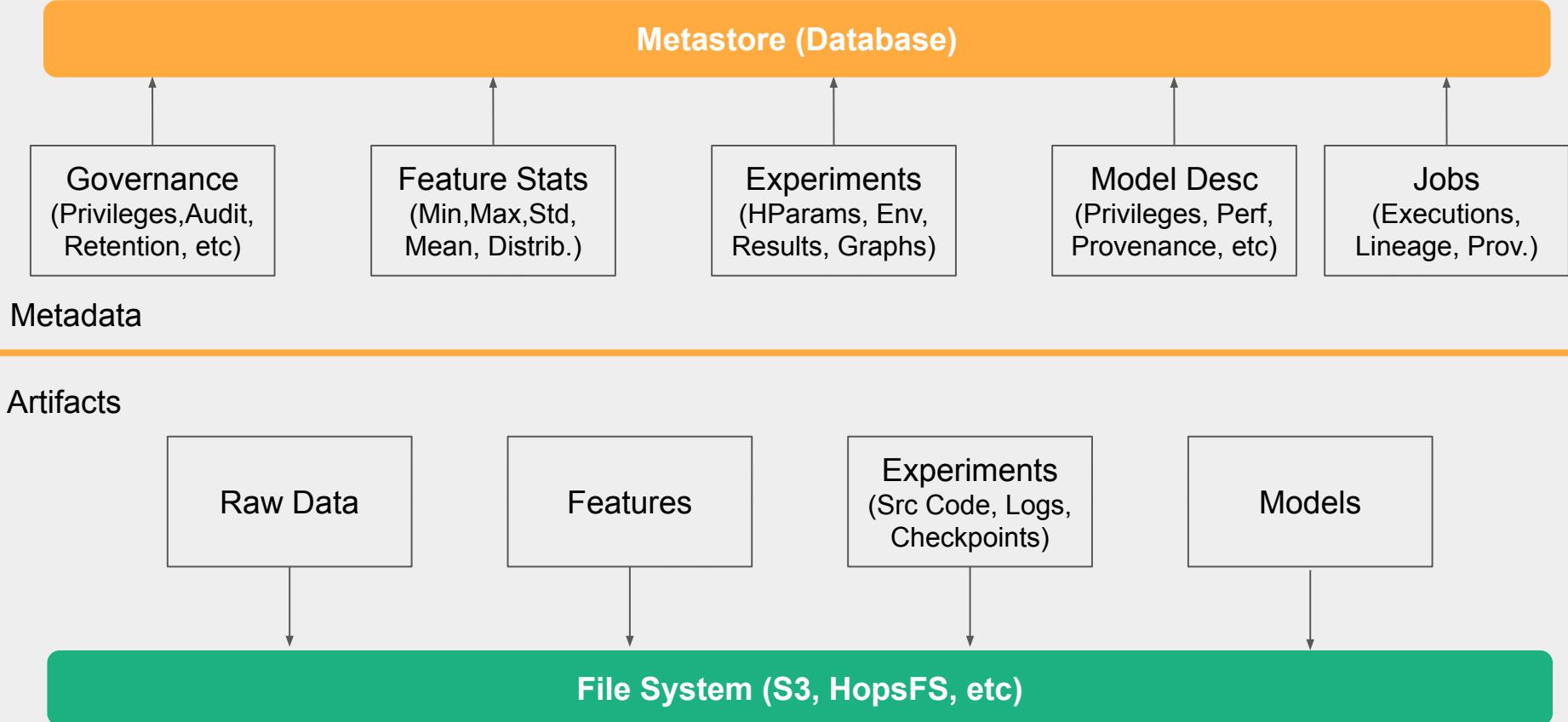
File System (S3, HopsFS, etc)



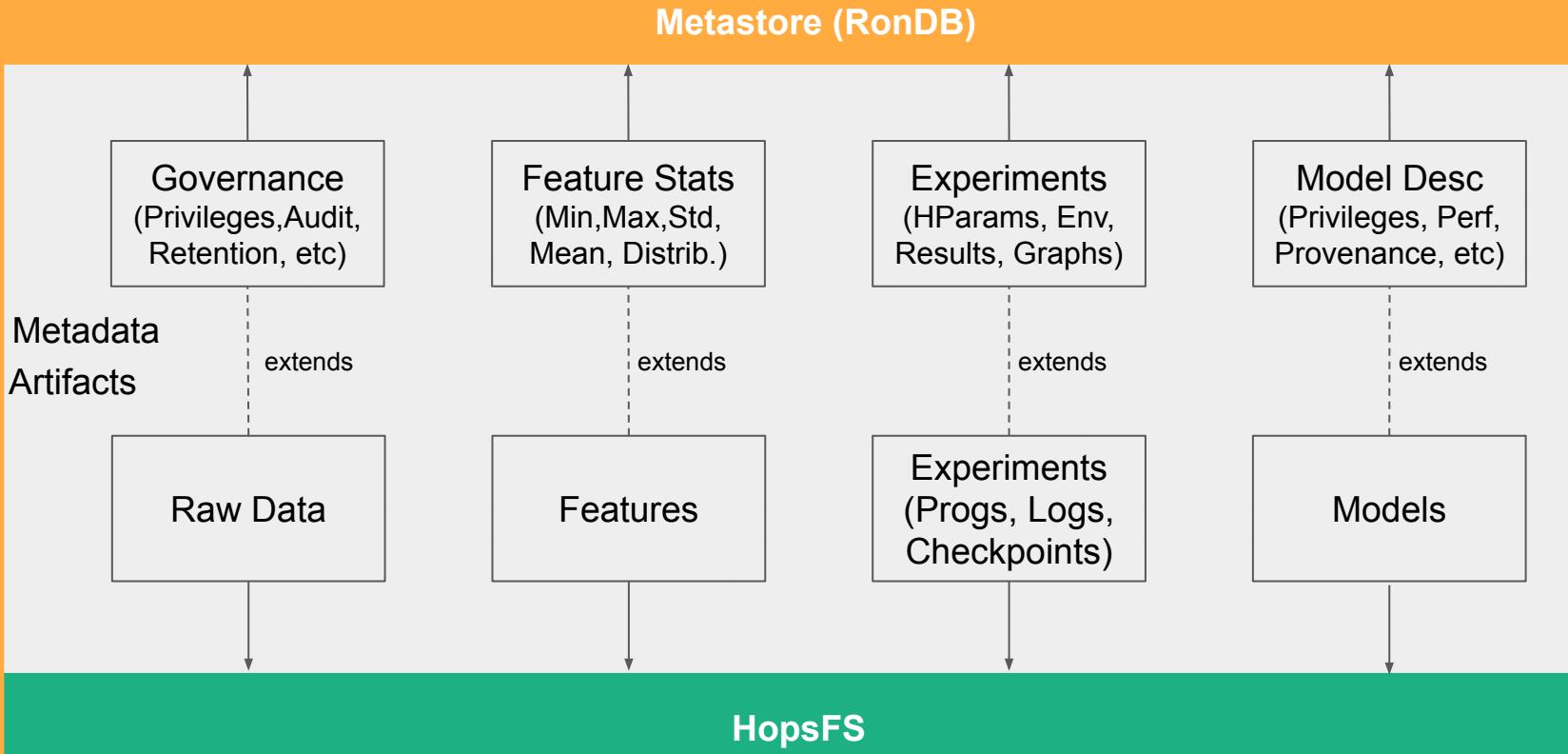
Metadata is data that describes other data.

Unspoken Assumption:
Why are Data and Metadata always separate stores?

Artifacts and Metadata in End-to-End ML Pipelines



Mechanism 4: Artifacts and Metadata in the same system - a Unified Metadata Layer (Hopsworks)





Implicit

Bottom-up tracking of provenance.

Requires redesigning the platform.

Conventions link files to artifacts.

Metadata is strongly consistent with storage platform.

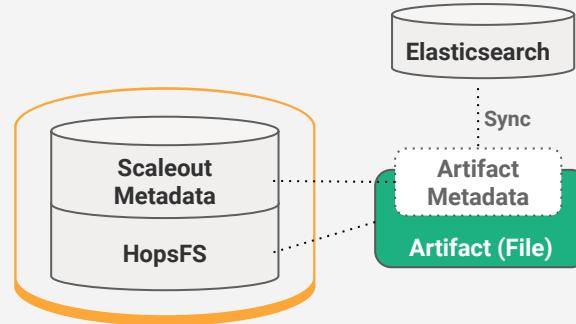


Explicit

Top-down tracking of provenance.

Push/Pull, CDC, or instrumented application or library code.

Standalone Metadata Store.



ePipe: Near Real-Time Polyglot Persistence of HopsFS Metadata

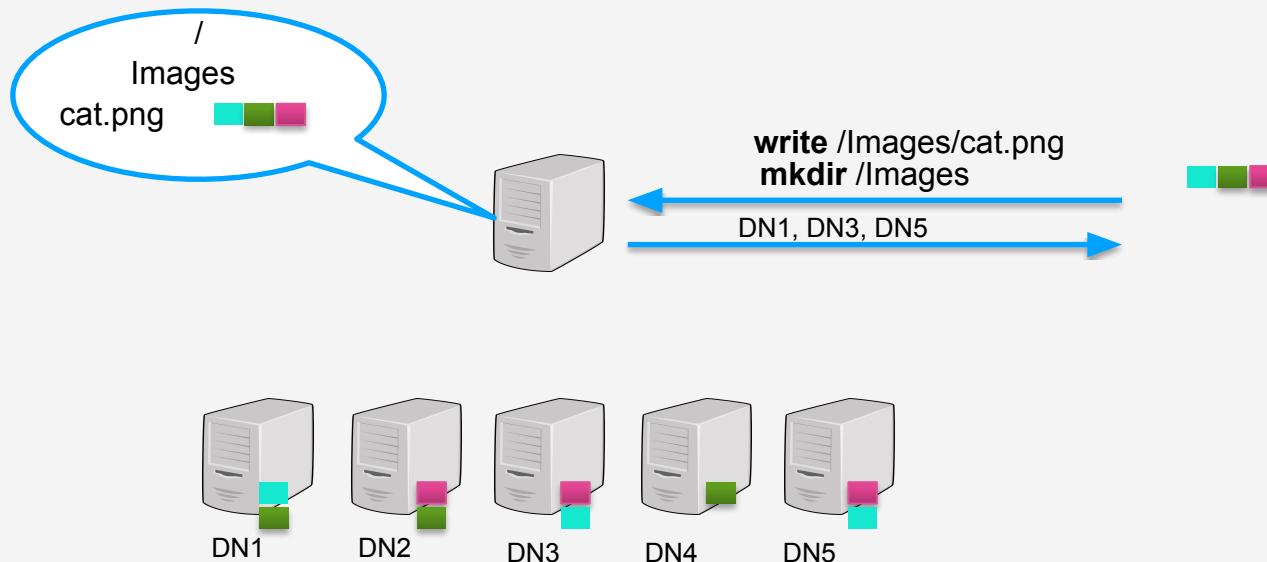
Mahmoud Ismail¹, Mikael Ronström², Seif Haridi¹, Jim Dowling¹

¹ KTH - Royal Institute of Technology ² Oracle

What is HopsFS?



- Highly scalable next-generation distribution of **HDFS**



What is HopsFS?

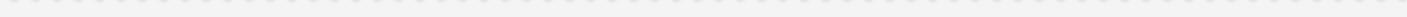
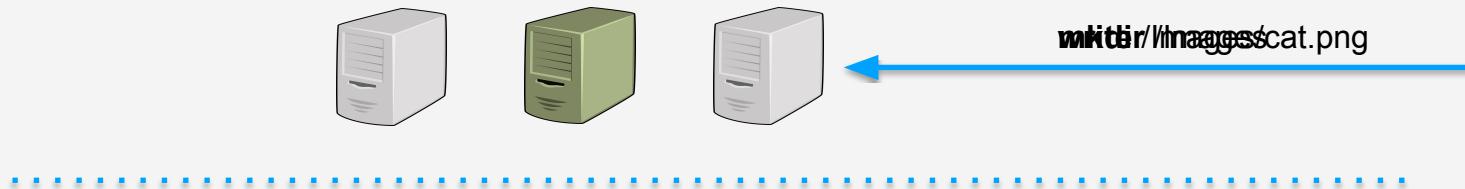


Metadata storage

RonDB



inodeID	name	parentID
1	/	0
2	Images	1
3	cat.png	2



**Block storage
(Datenodes)**



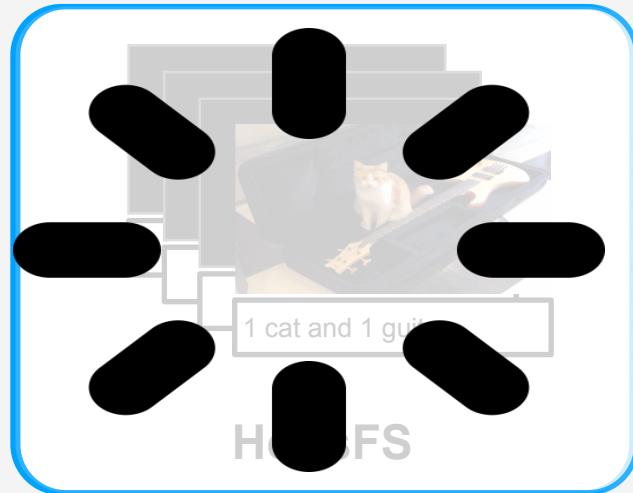


- Drop-in replacement distribution of HDFS
- **16X - 37X** the throughput of HDFS
- **37** larger clusters than HDFS
- **10** times lower latency



ePipe: Near Real-Time Polyglot Persistence of HopsFS

Metadata



Get all images with 1 cat and 1 guitar



Full-text search is not supported by RonDB

Polyglot Persistence - Replicating Metadata to External Systems for Efficient Querying



ePipe: Near Real-Time ~~Polyglot Persistence~~ of HopsFS

Metadata

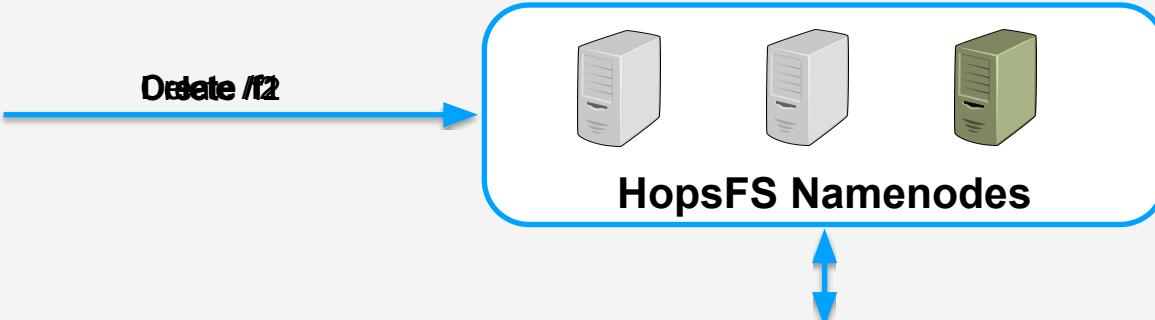


- ePipe is a databus that provides replicated metadata as a service for HopsFS
- ePipe internally
 - creates a consistent and correctly ordered change stream for HopsFS metadata
 - and eventually delivers the change stream with low latency (sub second) (**Near Real-time**) to consumers



- Extend HopsFS with a logging table to log file system changes
- Leverage the RonDB event API to stream changes on the logging table to ePipe
- ePipe enriches the file system events with appropriate data and publish the enriched events to the consumers

Inodes table and logging table updated in the same Transaction to ensure Consistency/Integrity



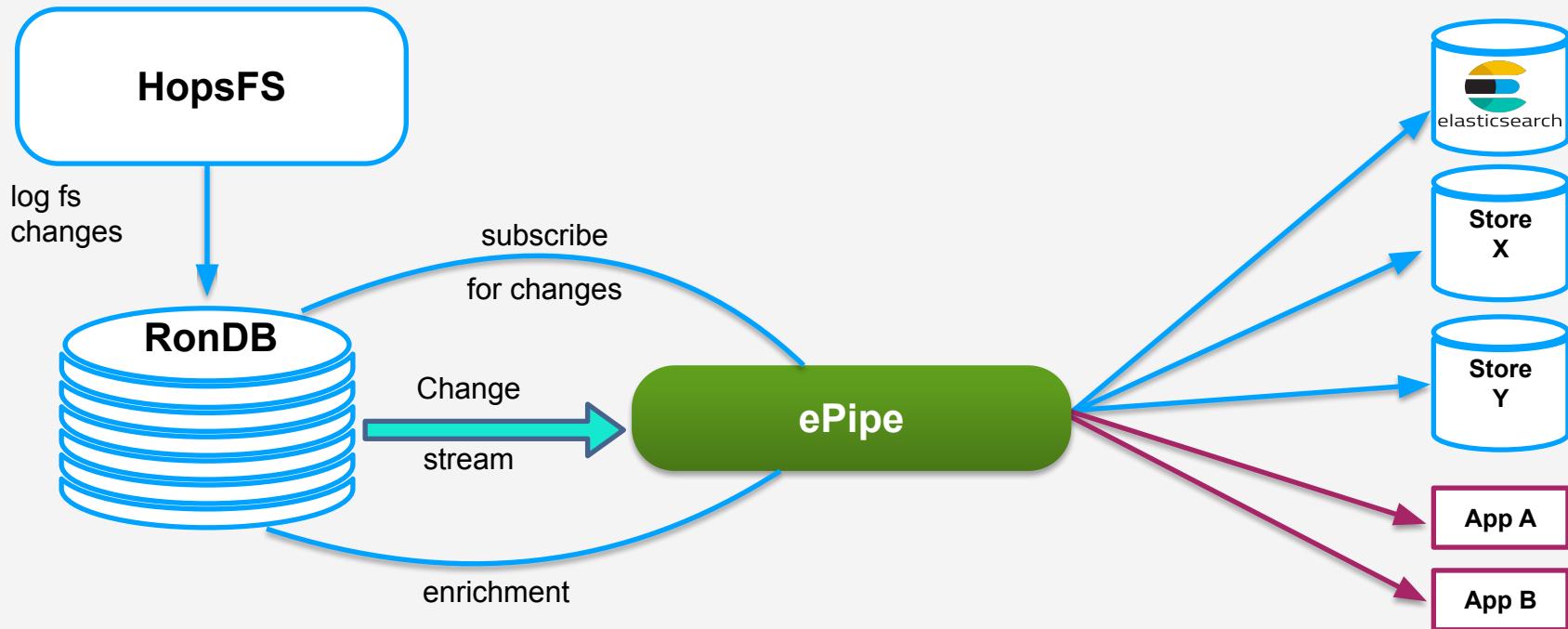
RonDB

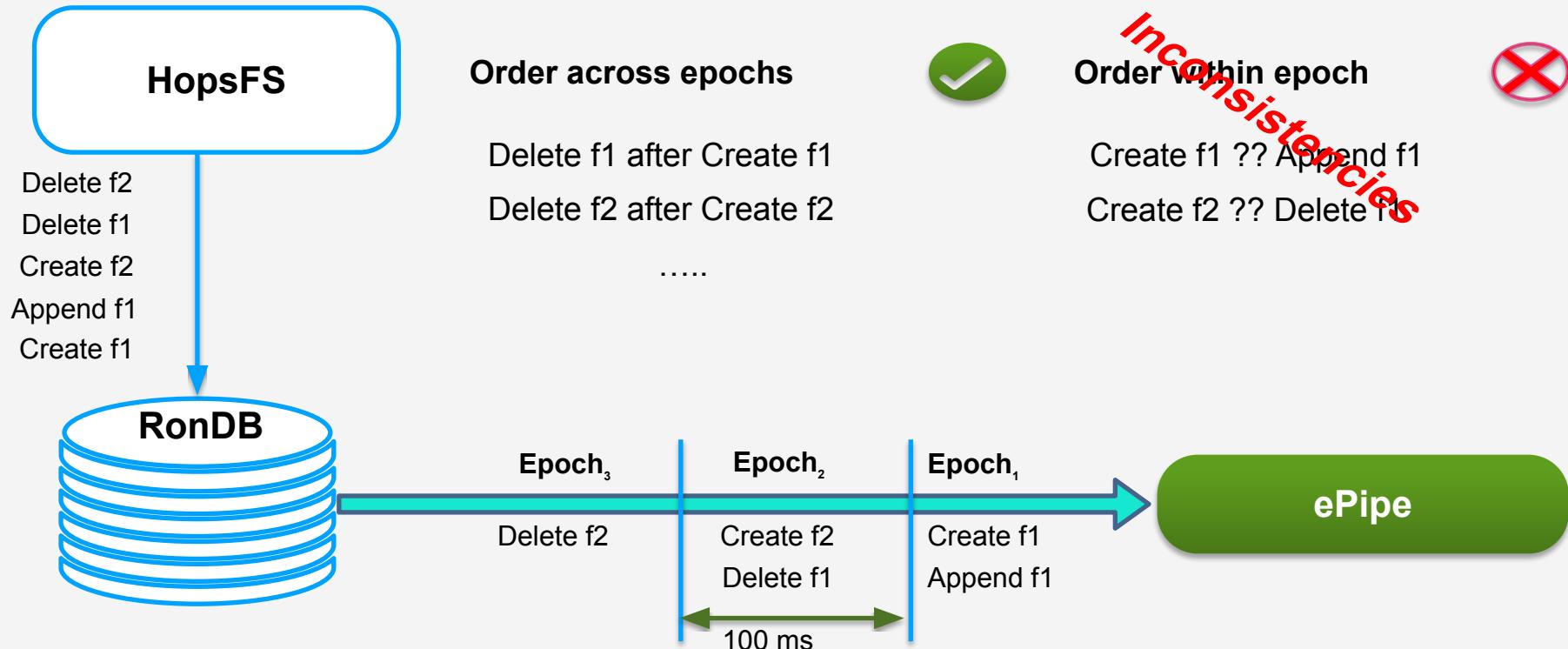
Inodes table

inodeID	name	parentID	
1	/	0	
2	f1	1	
3	f2	1	

logging table

name	operation
f1	CREATE
f2	CREATE
f2	DELETE
f1	DELETE







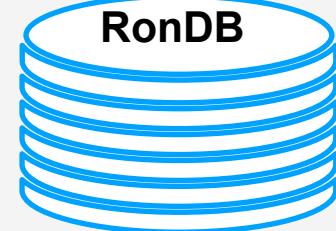
- **Property 1:** The epochs are totally ordered.
- **Property 2:** The changes within the same transaction happen in the same epoch.
- **Property 3:** The changes on files are ordered only if they are in different epochs, that is, no ordering is guaranteed within the same epoch.



HopsFS

Delete f2 ,**2**
 Delete f1 ,**3**
 Create f2 ,**1**
 Append f1 ,**2**
 Create f1 ,**1**

We introduced a version number per inode
 which we will increment whenever
 a change occurs to an inode.



Append f1 after Create f1



Create f2 ?? Delete f1



Epoch₃ Epoch₂ Epoch₁

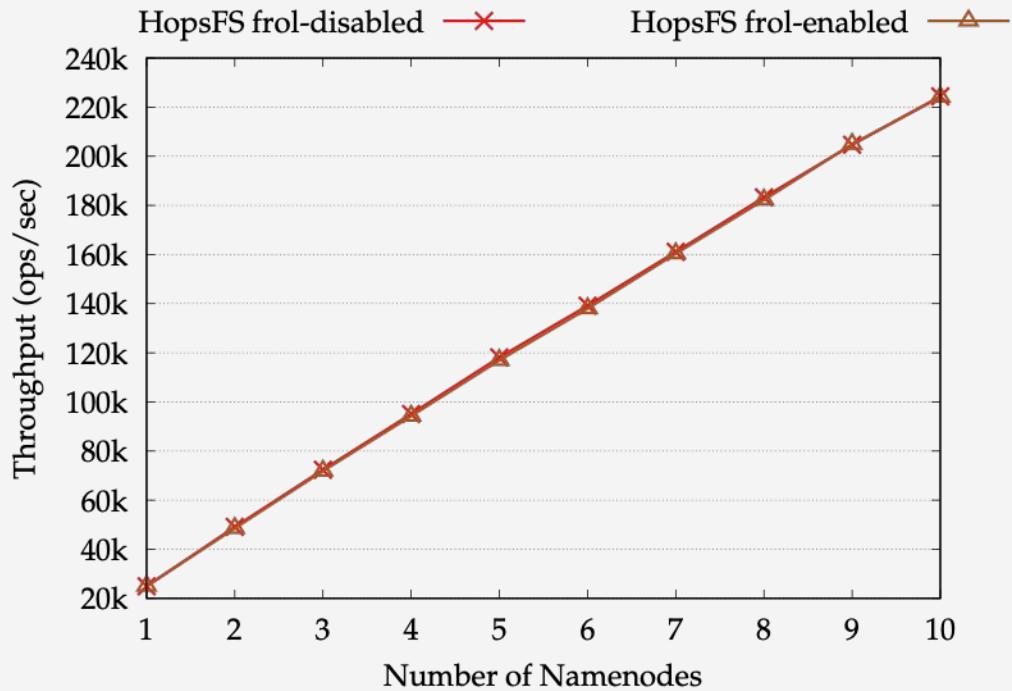
Delete f2 ,**2** Create f2 ,**1** Create f1 ,**1**
 Delete f1 ,**3** Append f1 ,**2**

ePipe

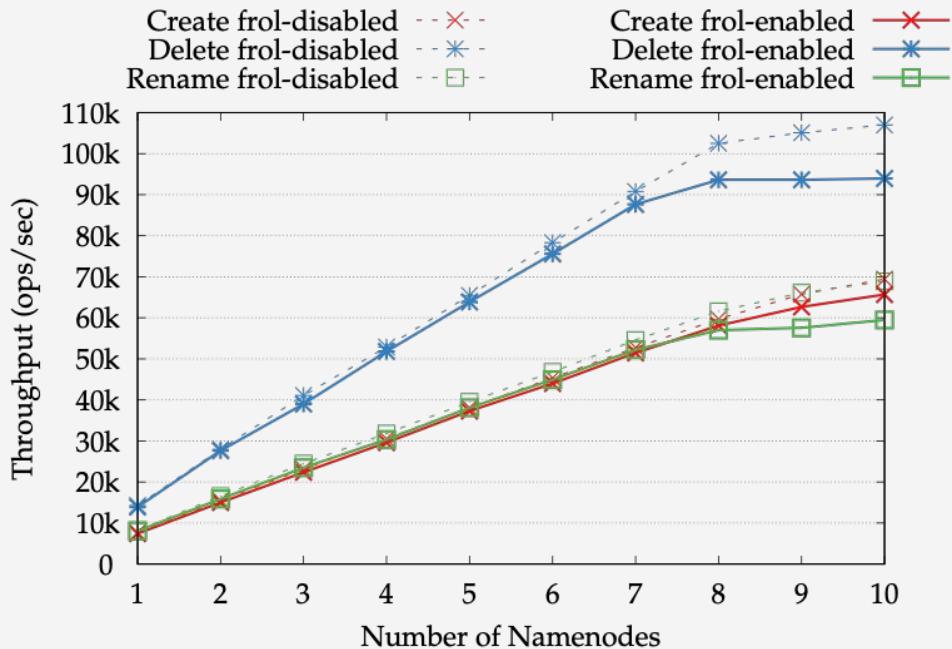


- **Property 1 & 2 & 3**
- **Property 4 & 5:** The version number ensures the serializability of the changes on the same file/directory within epochs.
- **Property 6:** The order of changes for different files/directories within the same epoch doesn't matter.

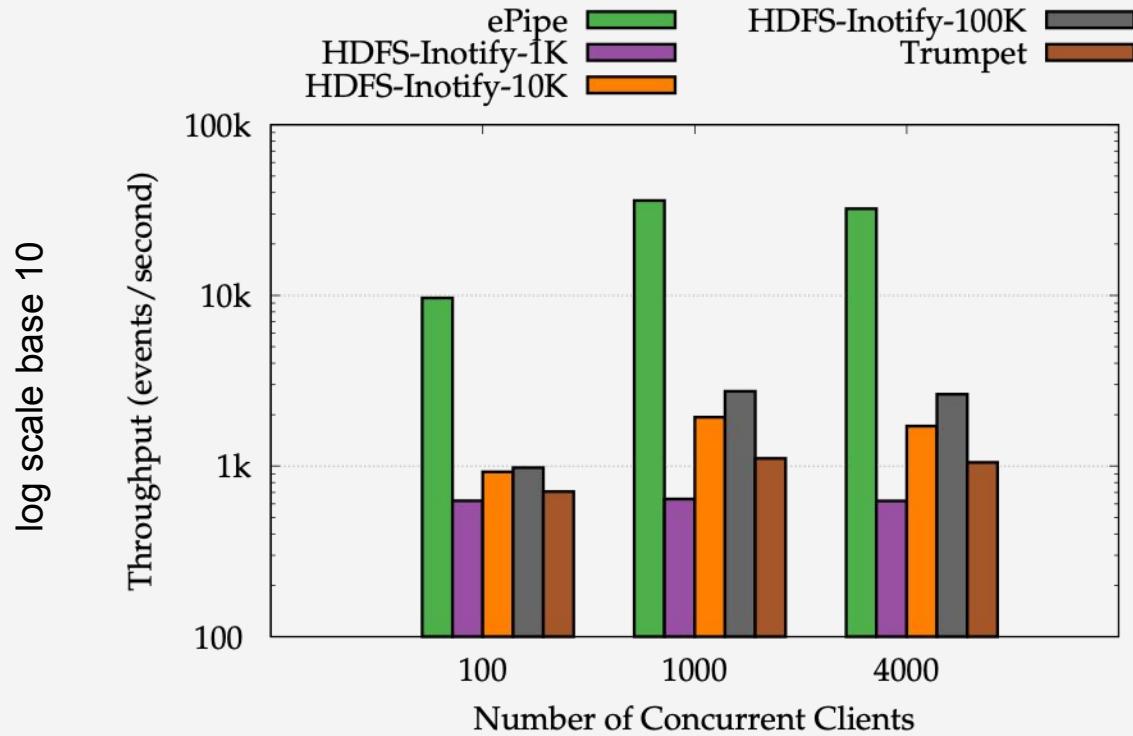
Logging overhead on HopsFS



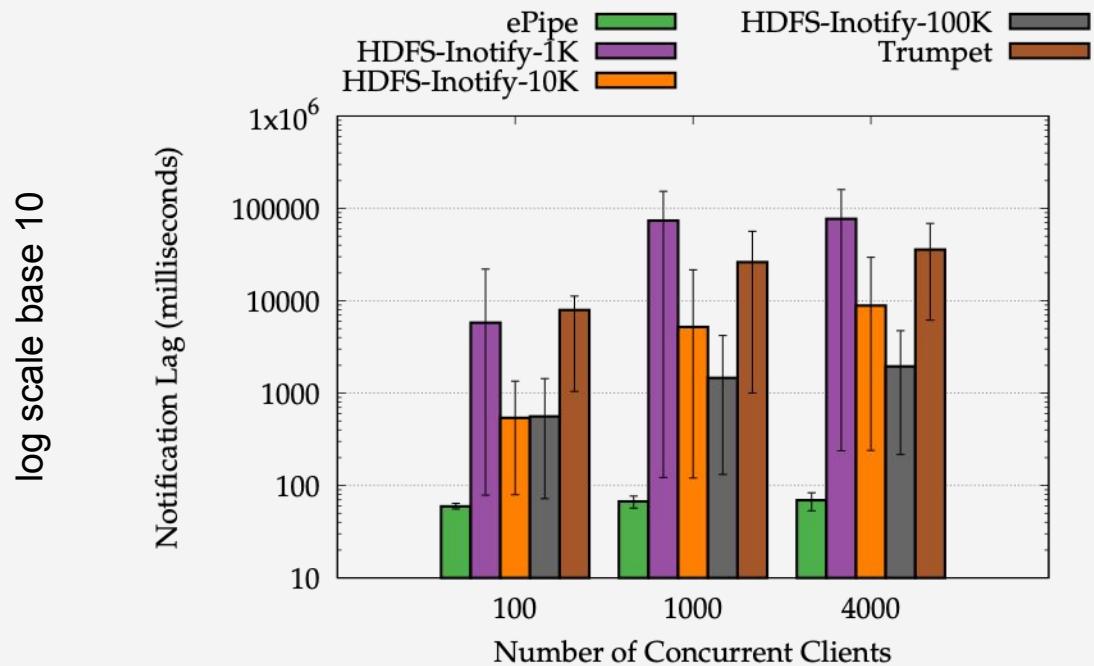
Logging overhead on HopsFS



Notifications Throughput



Latency: average Lag Time



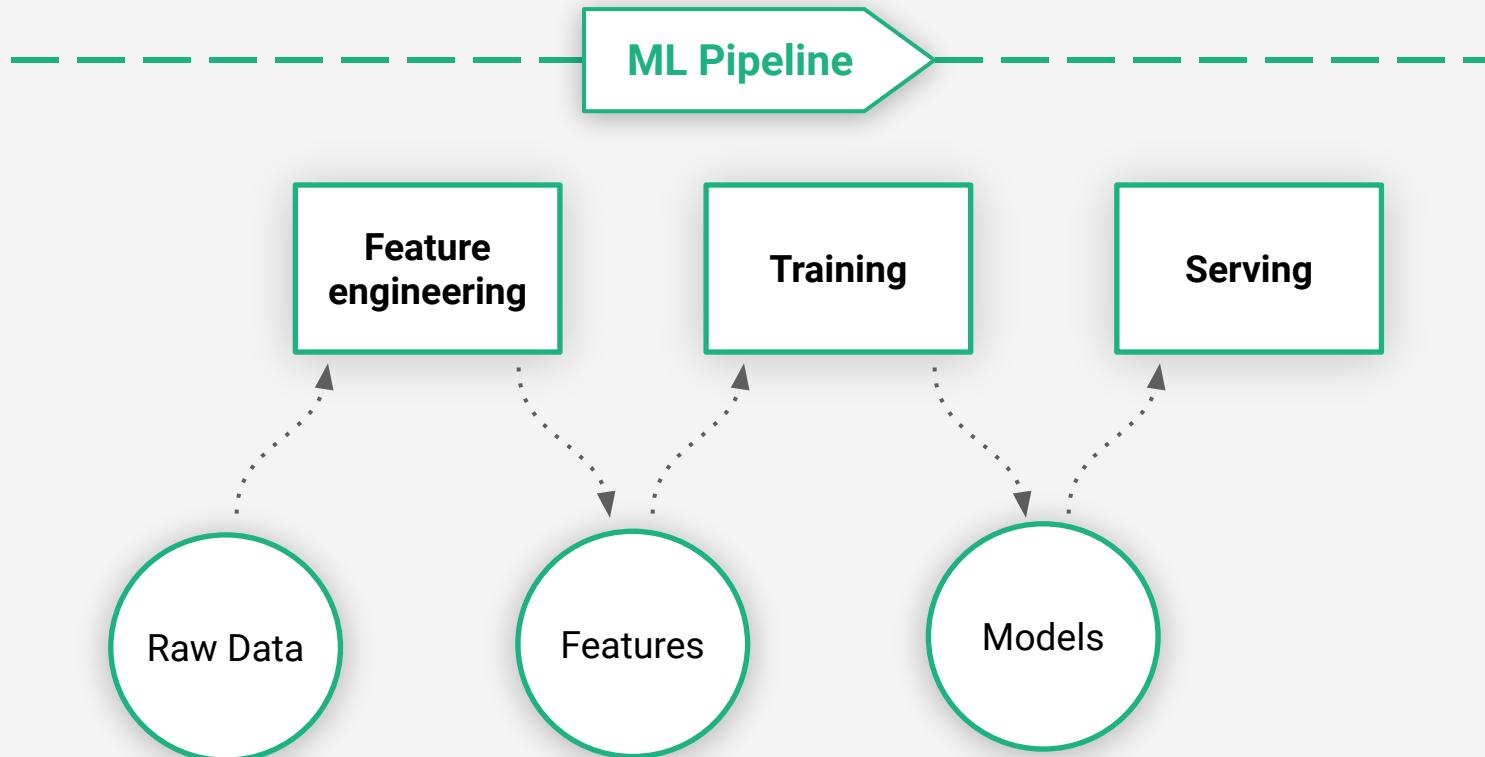


- Supports failure recovery thanks to the persistent logging table
 - The log entries are deleted only once the associated events are successfully replicated to the downstream consumers.
 - At least once delivery semantics.
- Pluggable architecture
 - For example, filter events based on file name or any other attribute.
- Not Limited to HopsFS
 - Can be extended to watch for other logging tables for different purposes.



- A databus that provides replicated metadata as a service for HopsFS
- Low overhead on HopsFS
- Low replication lag (sub-second)
- High throughput
- Pluggable architecture

What is provenance - ML Pipeline





- Provenance improves understanding of complex ML Pipelines.
- Provenance should not change the core ML pipeline code.
- Provenance facilitates Debugging, Analyzing, Automating and Cleaning of ML Pipelines.
- Provenance and Time Travel facilitate reproducibility of experiments.
- In Hopsworks, we introduced a new mechanism for provenance based on embedded metadata in a scale-out consistent metadata layer.

MLFlow Metadata - Explicit API calls



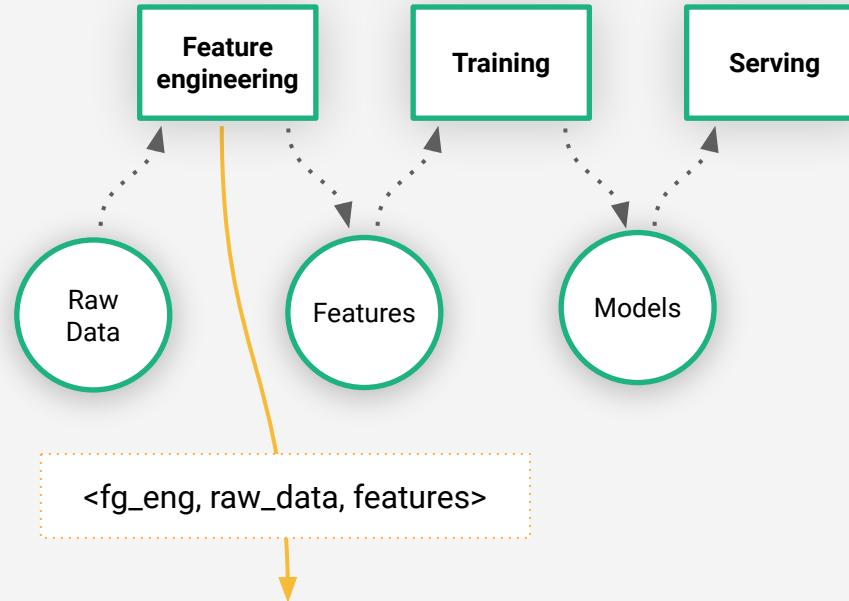
```
def train(data_path, max_depth, min_child_weight, estimators, model_name):
    X_train, X_test, y_train, y_test = build_data(..)
    mlflow.set_tracking_uri("jdbc:mysql://username:password@host:3306/database")
    mlflow.set_experiment("My Experiment")
    with mlflow.start_run() as run:
        ...
        mlflow.log_param("max_depth", max_depth)
        mlflow.log_param("min_child_weight", min_child_weight)
        mlflow.log_param("estimators", estimators)
        with open("test.txt", "w") as f:
            f.write("hello world!")
        mlflow.log_artifacts("/full/path/to/test.txt")
        ...
        model.fit(X_train, y_train) # auto-logging
        ...
        mlflow.tensorflow.log_model(model, "tensorflow-model",
            registered_model_name=model_name)
```



```
def train(data_path, max_depth, min_child_weight, estimators):
    X_train, X_test, y_train, y_test = build_data(..)
    ...
    print("hello world") # monkeypatched - prints in notebook
    ...
    model.fit(X_train, y_train) # auto-logging
    ...
    #Saves model to "hopsfs://Projects/myProj/models/.."
    hops.export_model(model, "tensorflow", ..., model_name)
    ...
    # maggy makes an API call to track this dict
    return {'accuracy': accuracy, 'loss': loss, 'diagram': 'diagram.png'}
```

```
from maggy import experiment
experiment.lagom(train, name="My Experiment", ...)
```

What is provenance - Metadata

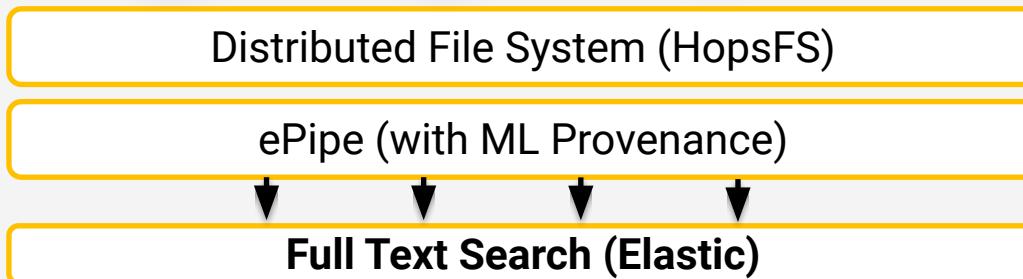
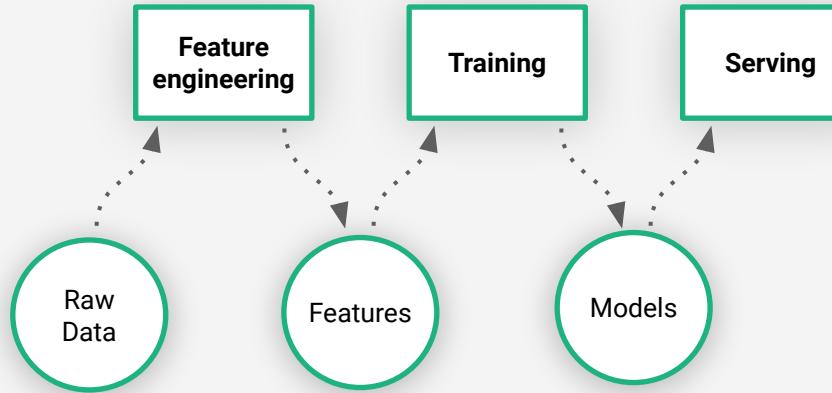


Pipeline code

In []:

```
add(fg_eng, raw_data, features)
...
add(training, features, model)
```

Let the platform manage the metadata!





ML Artifacts



Features, Feature Metadata,
Train/Test Datasets
Models, Model Metadata



Possibly thousands of files

Distributed File System



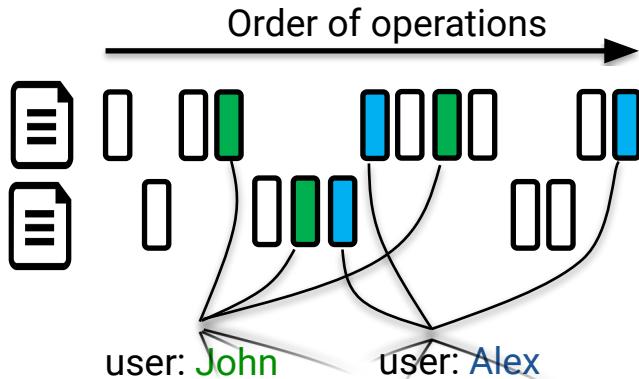
Generate thousands of operations

Change Data Capture (CDC)

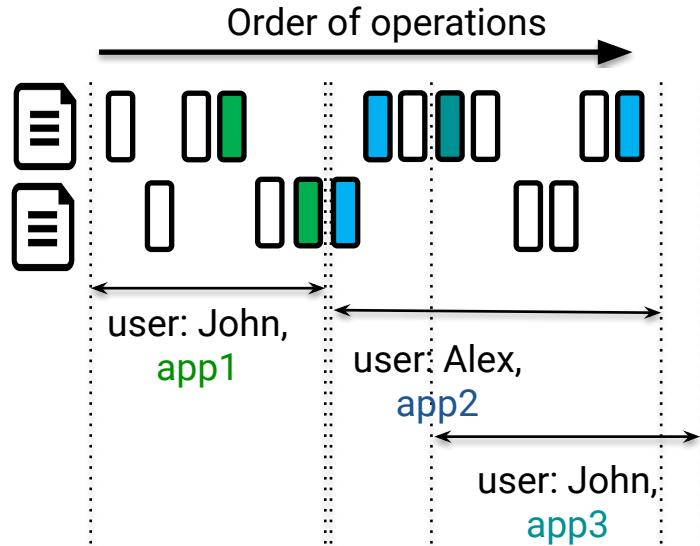
Capture only relevant operations



More context for file system operations?



Are any of these operations related?

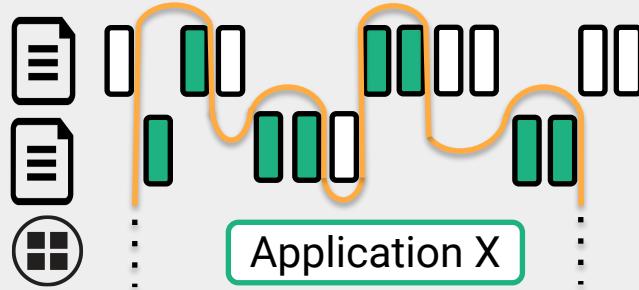


Certificates (with AppId) enabled FS Operation



Distributed File System

Read/Write/Create/Delete/XAttr/Metadata



Additional Context

`<file, op, user_id, app_id, job_id, pipeline_id>`

Resource Manager - Yarn (Application Context)

Link input/output files via Apps

Job Manager - Hopsworks (Job Context)

Different Executions of the same Job

Workflow Manager - Airflow (Pipeline Context)

Jobs as Stages of the same Pipeline



Hopsworks Conventions

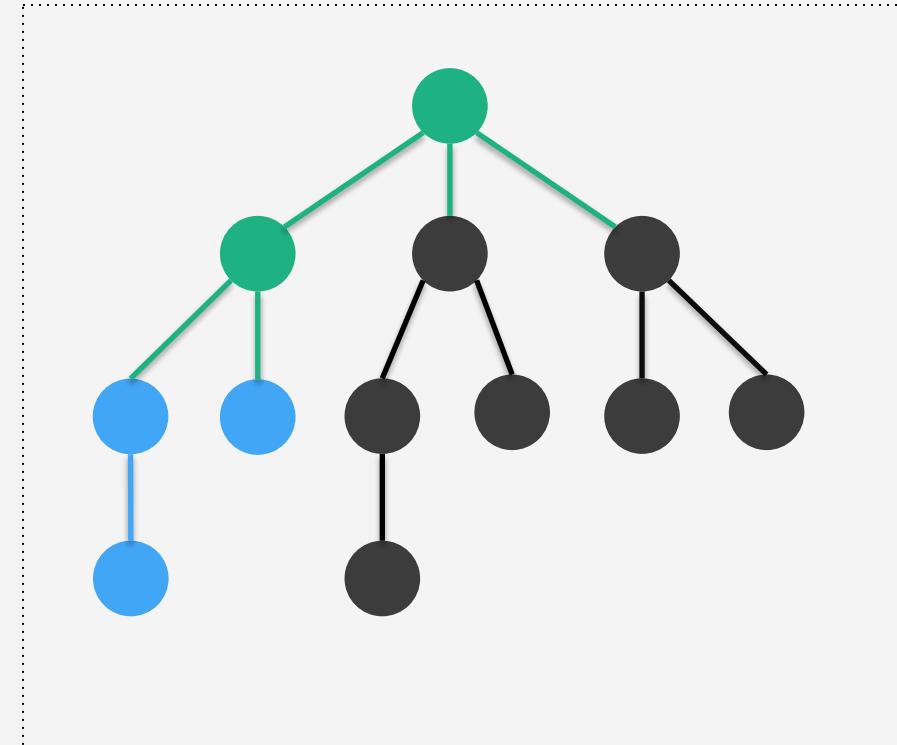
```
└── /featurestore  
    ├── /training_datasets  
    ├── /models  
    ├── /logs  
    └── /notebooks
```



- **Path based filtering**

Example

Project
└─ /featurestore
 └─ /training_datasets
 └─ /models





Path based filtering

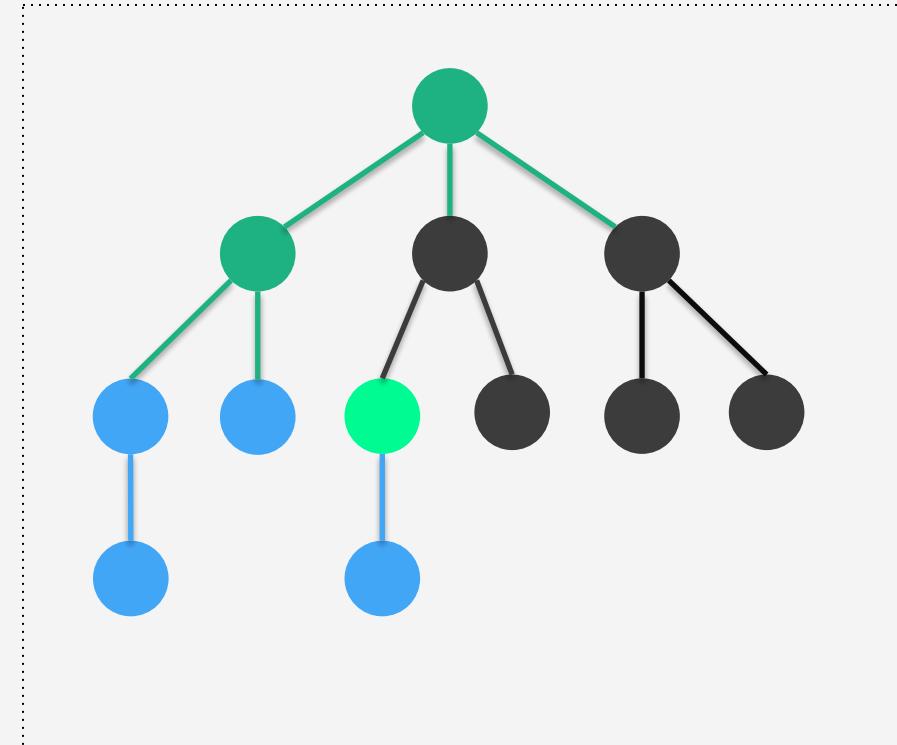
- **Tag based filtering**

Example:

Custom metadata based on HDFS XAttr.

Tag: <tutorial>, <debug>

Tags can enable logging of all operations,
if path based filtering is not easy to set





Path based filtering

Tag based filtering

- **Coalesce FS Operations**

Example:

Read file₁

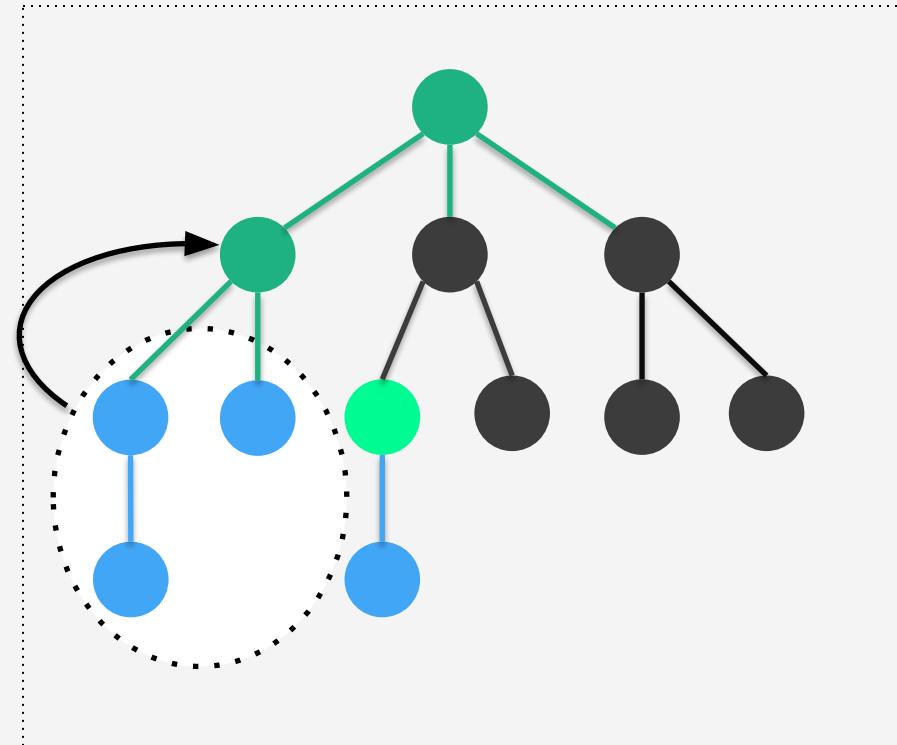
Read file₂

...

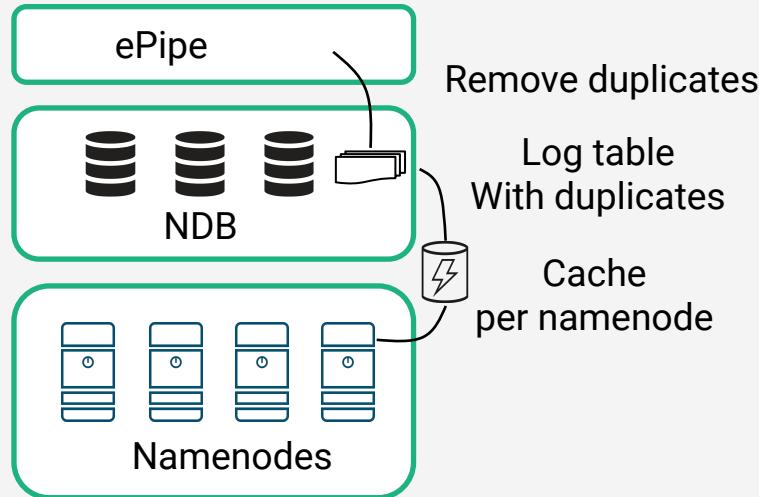
Read file_n



Access,
Training Dataset

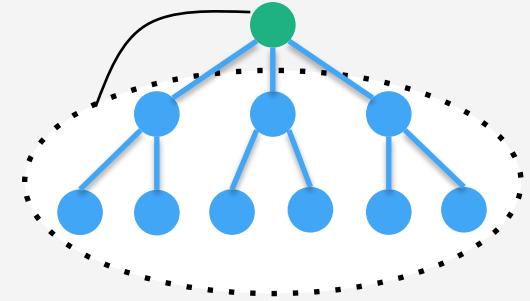


Optimization - FS Operation Coalesce



In []:

```
hops.load_training_dataset(  
    "/Projects/LC/Training_Datasets/ImageNet")  
...  
hops.save_model("/Projects/LC/Models/ResNet")
```



Parent Create	Artifact Create
Parent Delete	Artifact Delete
Children Read	Artifact Access
Children Create/Delete/ Append/Truncate	Artifact Mutation



Path based filtering

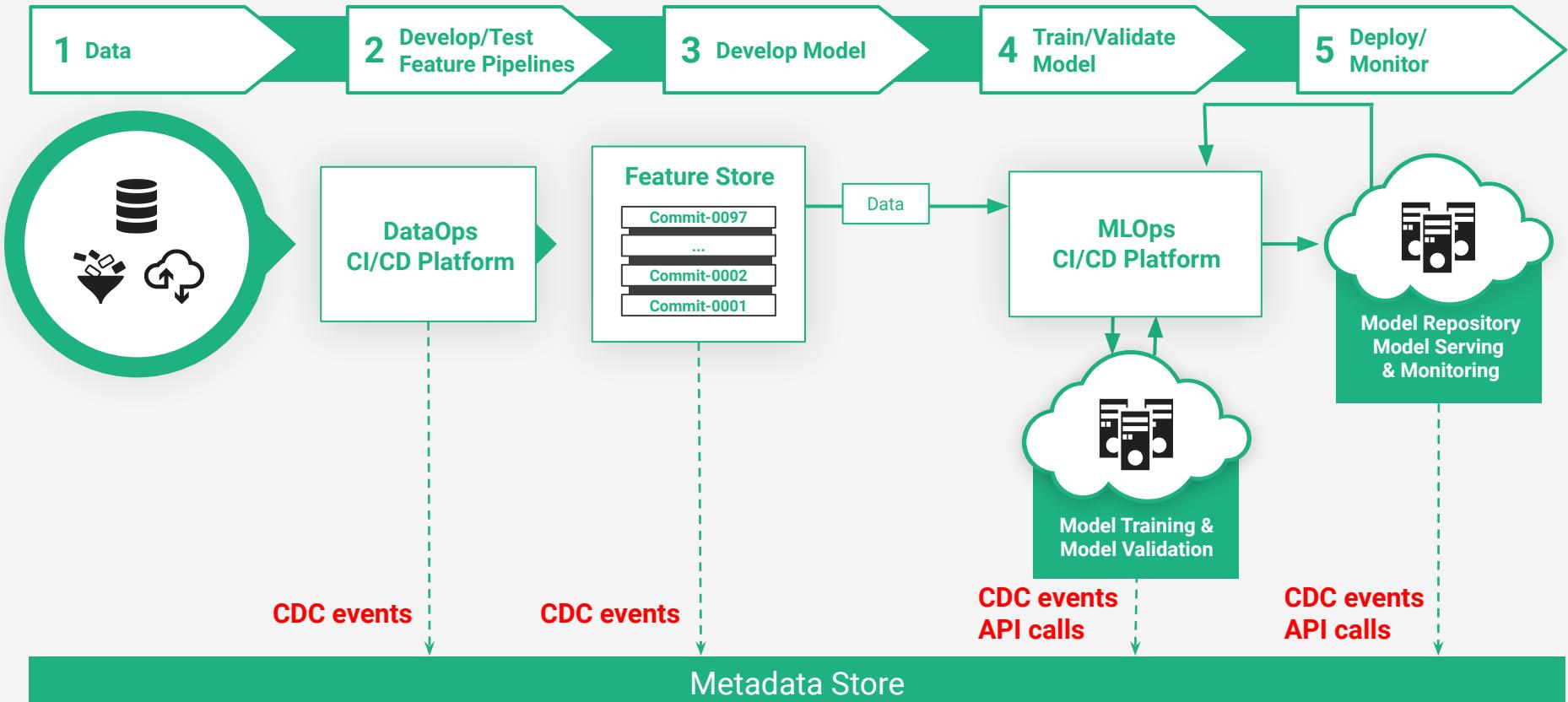
Tag based filtering

Coalesce FS Operations

- **Filtered Operations**

Filesystem Op	Metadata Stored
Create/Delete	Artifact existence
XAttr	Add metadata to artifact
Read	Artifact used by ..
Children Files Create/Delete	Artifact mutation
Append/Truncate	Artifact mutation
Permissions/ACL	Artifact metadata mutation

Hopsworks ML Pipelines





Bias Detected



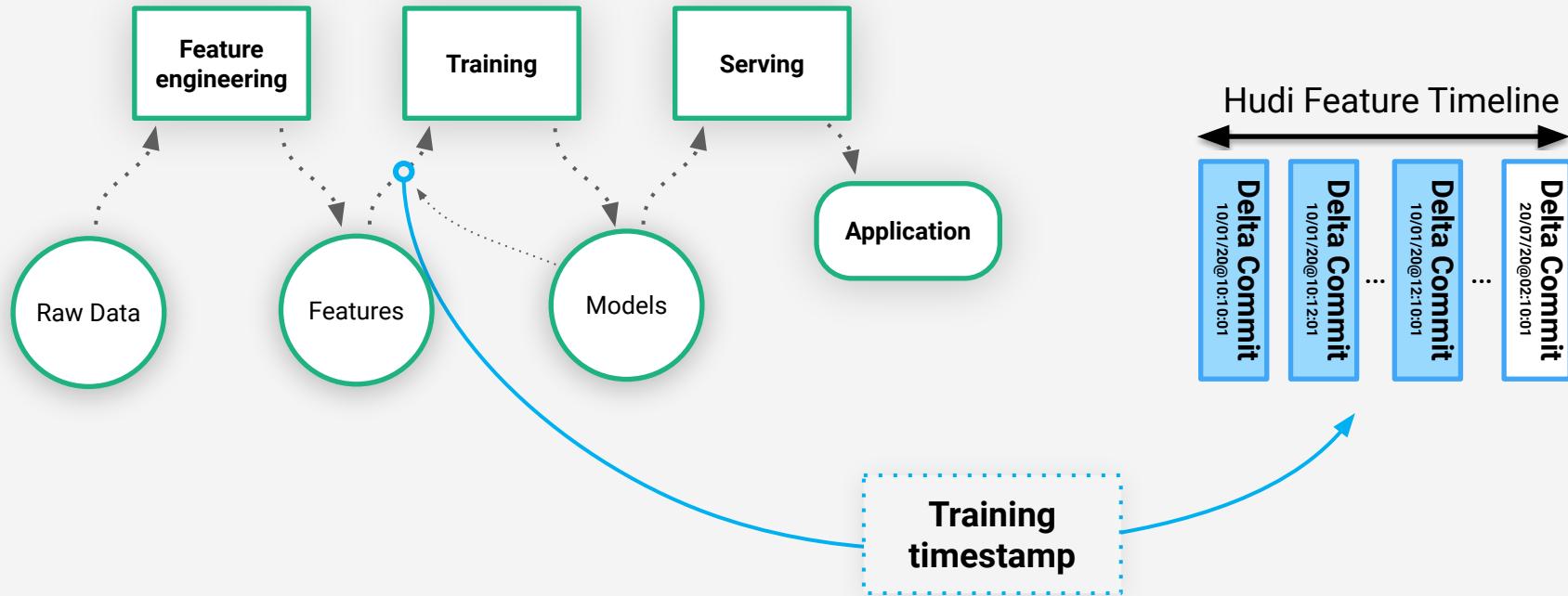
What do I do





Claim of Model Bias!

Can we determine the exact features used?





HOPSWORKS.ai

BY LOGICAL CLOCKS



@hopsworks



<http://github.com/logicalclocks/hopsworks>

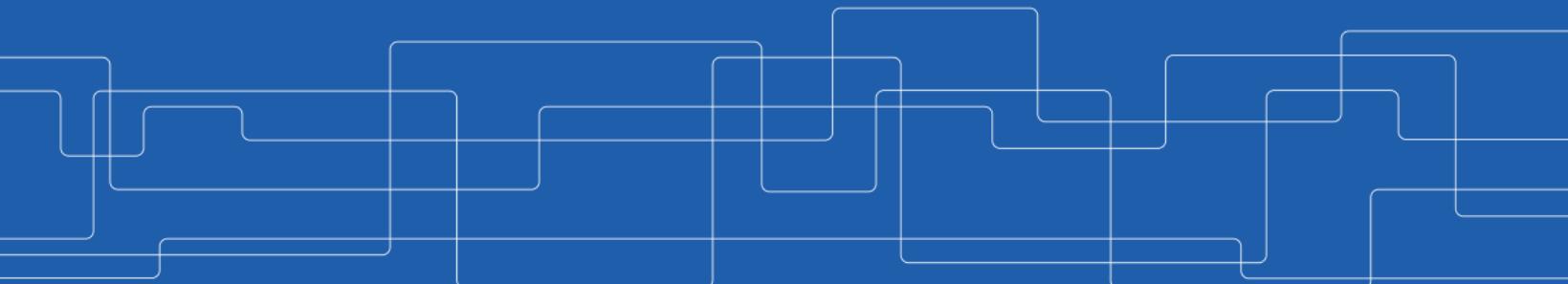


- [Ormenisan et al, Time-travel and Provenance for ML Pipelines, Usenix OpML 2020](#)
- [Niazi et al, HopsFS, Usenix Fast 2017](#)
- [Ismail et al, ePipe, CCGrid 2019](#)
- [Small Files in HopsFS, ACM Middleware 2018](#)
- [Ismail et al, HopsFS-S3, ACM Middleware 2020](#)
- [Meister et al, Oblivious Training Functions, 2020](#)
- [Hopsworks](#)



Distributed Roubust Learning

Amir H. Payberah
payberah@kth.se
2021-12-15



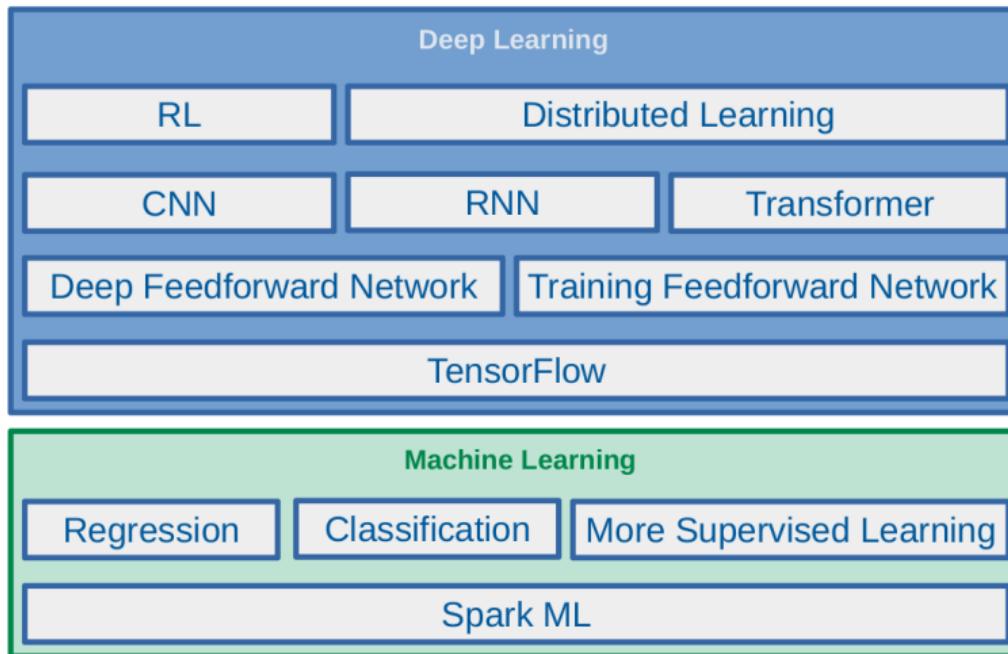


The Course Web Page

<https://id2223kth.github.io>
<https://tinyurl.com/6s5jy46a>

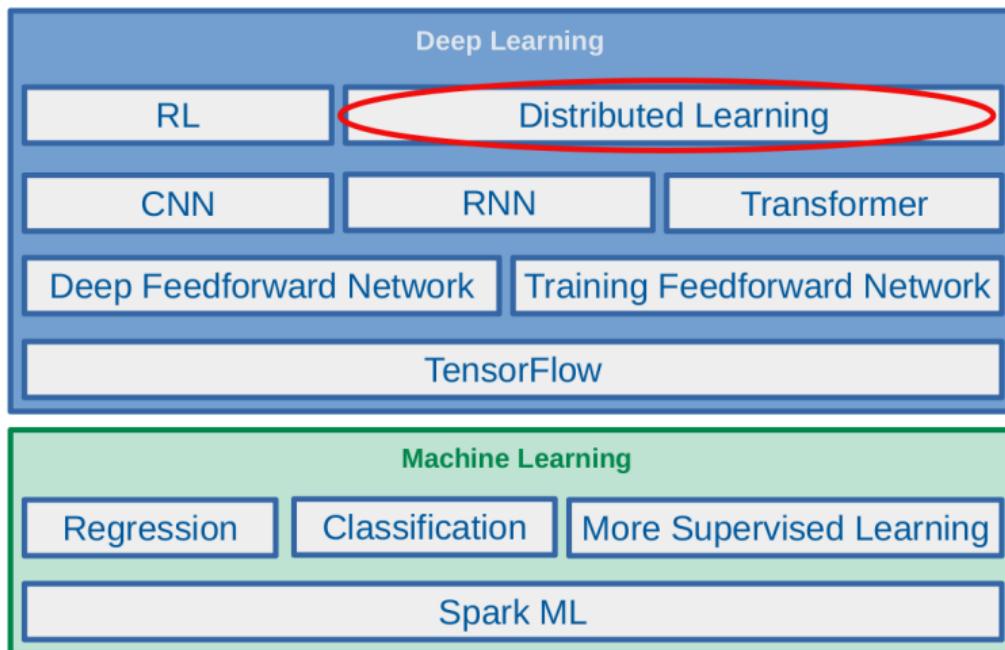


Where Are We?





Where Are We?





Adversarial Goals



Adversarial Goals

- ▶ Confidentiality and privacy
 - Confidentiality of the model or the data.



Adversarial Goals

- ▶ Confidentiality and privacy
 - Confidentiality of the model or the data.
- ▶ Integrity
 - Integrity of the predictions



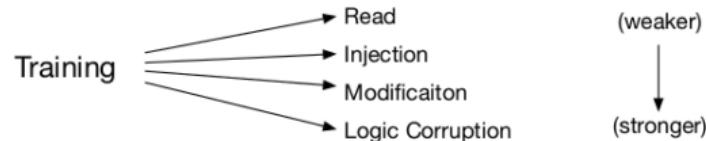
Adversarial Goals

- ▶ Confidentiality and privacy
 - Confidentiality of the model or the data.
- ▶ Integrity
 - Integrity of the predictions
- ▶ Availability
 - Availability of the system deploying machine learning



Adversarial Capabilities for Integrity Attacks

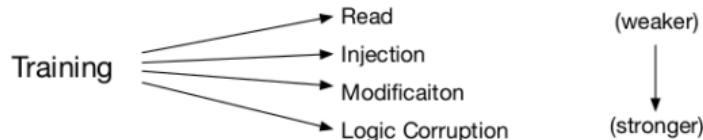
► Training phase



[Papernot et al., SoK: Security and Privacy in Machine Learning, 2018]

Adversarial Capabilities for Integrity Attacks

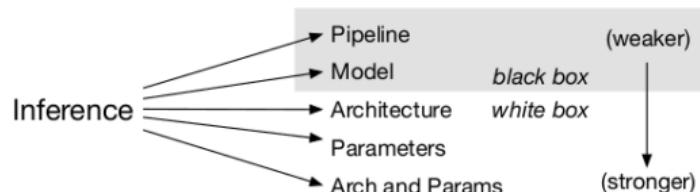
► Training phase



[Papernot et al., SoK: Security and Privacy in Machine Learning, 2018]

► Inference phase

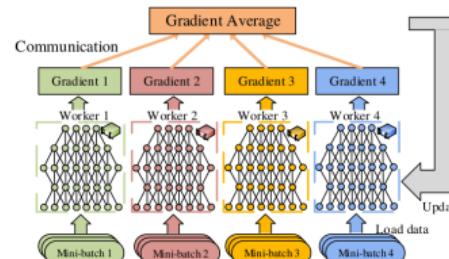
- White box
- Black box



[Papernot et al., SoK: Security and Privacy in Machine Learning, 2018]

Our Focus and Goal

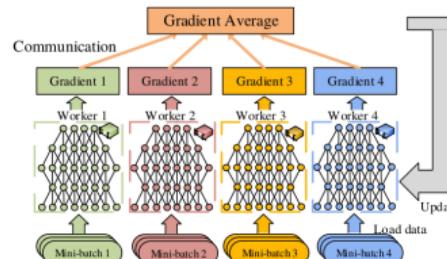
- ▶ Data parallelization



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Our Focus and Goal

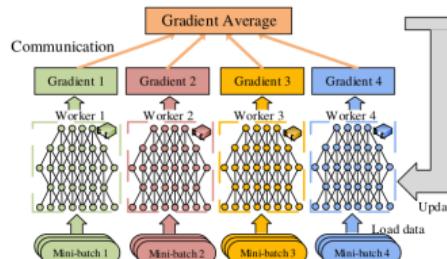
- ▶ Data parallelization
- ▶ Each worker is prone to adversarial attack.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Our Focus and Goal

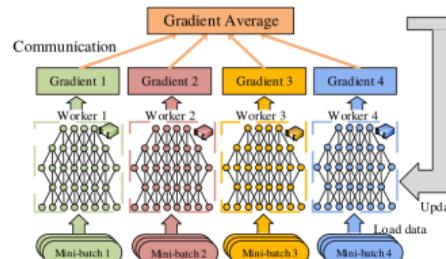
- ▶ Data parallelization
- ▶ Each worker is prone to adversarial attack.
- ▶ Adversarial attacks: some unknown subset of computing devices are compromised and behave adversarially (e.g., sending out malicious messages)



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Our Focus and Goal

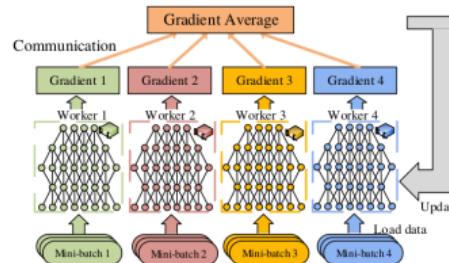
- ▶ Data parallelization
- ▶ Each worker is prone to adversarial attack.
- ▶ Adversarial attacks: some unknown subset of computing devices are compromised and behave adversarially (e.g., sending out malicious messages)
- ▶ Our goal: integrity of the model in the training phase



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (1/3)

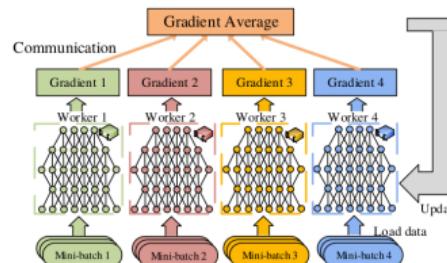
- One parameter server, and n workers.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (1/3)

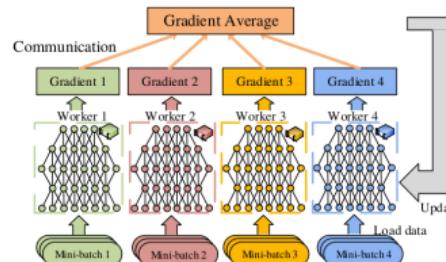
- ▶ One **parameter server**, and **n workers**.
- ▶ Computation is divided into **synchronous rounds**.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (1/3)

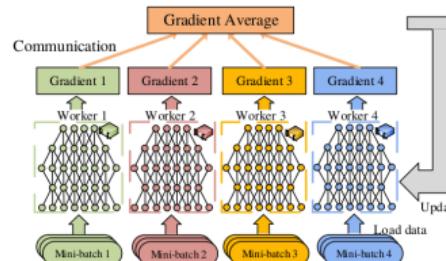
- ▶ One **parameter server**, and n **workers**.
- ▶ Computation is divided into **synchronous rounds**.
- ▶ During round t , the **parameter server** broadcasts its parameter vector $w \in \mathbb{R}^d$ to all the **workers**.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (2/3)

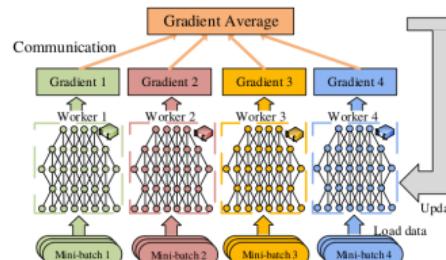
- ▶ At each round t , each **correct worker i** computes $G_i(w_t, \beta)$.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (2/3)

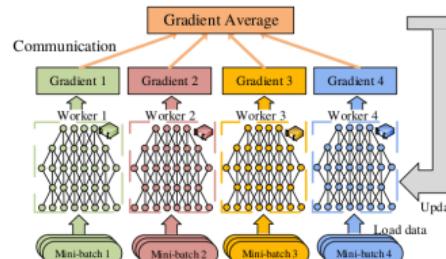
- ▶ At each round t , each correct worker i computes $G_i(w_t, \beta)$.
- ▶ $G_i(w_t, \beta)$: the local estimate of the gradient of the loss function $\nabla J(w_t)$.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (2/3)

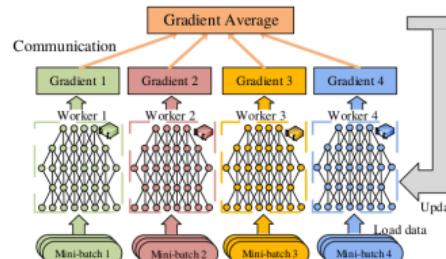
- ▶ At each round t , each correct worker i computes $G_i(w_t, \beta)$.
- ▶ $G_i(w_t, \beta)$: the local estimate of the gradient of the loss function $\nabla J(w_t)$.
- ▶ β : a mini-batch of i.i.d. samples drawn from the dataset.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (2/3)

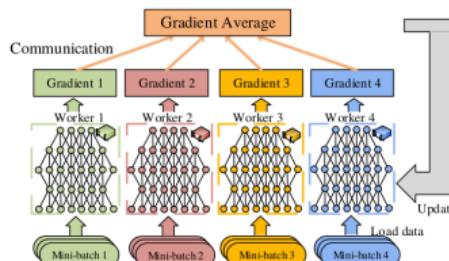
- ▶ At each round t , each correct worker i computes $G_i(w_t, \beta)$.
- ▶ $G_i(w_t, \beta)$: the local estimate of the gradient of the loss function $\nabla J(w_t)$.
- ▶ β : a mini-batch of i.i.d. samples drawn from the dataset.
- ▶ $G_i(w_t, \beta) = \frac{1}{|\beta|} \sum_{x \in \beta} \nabla l_i(w_t, x)$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (3/3)

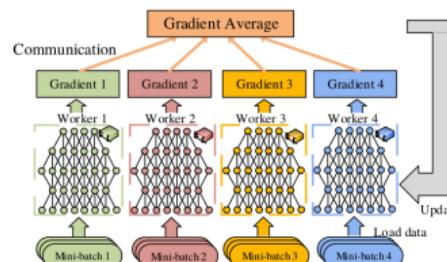
- ▶ The parameter server computes $F(G_1, G_2, \dots, G_n)$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (3/3)

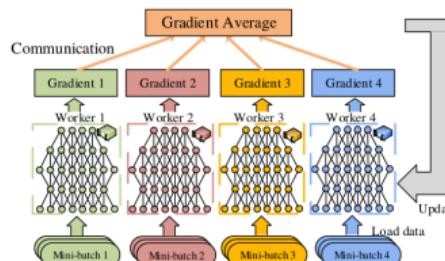
- ▶ The parameter server computes $F(G_1, G_2, \dots, G_n)$
- ▶ Gradient Aggregation Rule (GAR): $F(G_1, G_2, \dots, G_n) = \frac{1}{n} \sum_{i=1}^n G_i$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (3/3)

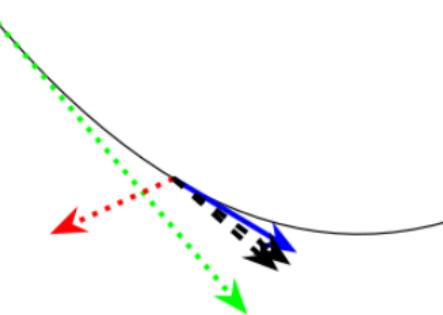
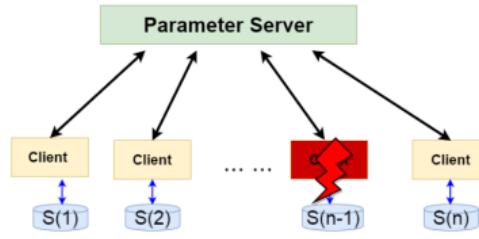
- ▶ The parameter server computes $F(G_1, G_2, \dots, G_n)$
- ▶ Gradient Aggregation Rule (GAR): $F(G_1, G_2, \dots, G_n) = \frac{1}{n} \sum_{i=1}^n G_i$
- ▶ The parameter server updates the parameter vector $w \leftarrow w - \gamma F(G_1, G_2, \dots, G_n)$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed SGD with Byzantine Workers

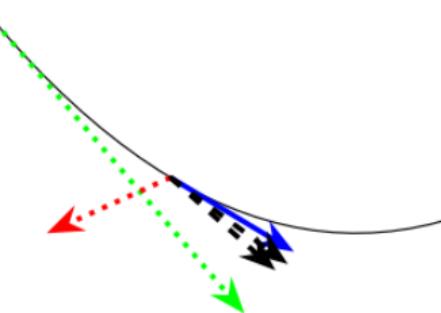
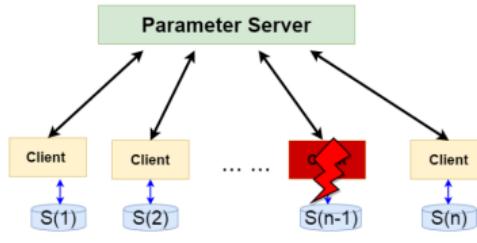
- ▶ Among the n workers, f of them are possibly **Byzantine** (behaving arbitrarily).



[El-Mhamdi et al., Fast and Secure Distributed Learning in High Dimension, 2019]

Distributed SGD with Byzantine Workers

- ▶ Among the n workers, f of them are possibly **Byzantine** (behaving arbitrarily).
- ▶ A **Byzantine** worker b proposes a vector G_b that can deviate arbitrarily from the vector it is supposed.



[El-Mhamdi et al., Fast and Secure Distributed Learning in High Dimension, 2019]



Averaging GAR and Byzantine Workers

- ▶ Averaging GAR: $F(G_1, G_2, \dots, G_n) = \frac{1}{n} \sum_{i=1}^n G_i$
- ▶ $w \leftarrow w - \gamma F(G_1, G_2, \dots, G_n)$



Averaging GAR and Byzantine Workers

- ▶ Averaging GAR: $F(G_1, G_2, \dots, G_n) = \frac{1}{n} \sum_{i=1}^n G_i$
- ▶ $w \leftarrow w - \gamma F(G_1, G_2, \dots, G_n)$
- ▶ Even a single Byzantine worker can prevent convergence.



Averaging GAR and Byzantine Workers

- ▶ Averaging GAR: $F(G_1, G_2, \dots, G_n) = \frac{1}{n} \sum_{i=1}^n G_i$
- ▶ $w \leftarrow w - \gamma F(G_1, G_2, \dots, G_n)$
- ▶ Even a **single** Byzantine worker can **prevent convergence**.
- ▶ **Proof:** if the Byzantine worker proposes $G_n = nU - \sum_{i=1}^{n-1} G_i$, then $F = U$.



(α, f) -Byzantine-Resilience (1/2)

- ▶ Assume n workers, where f of them are Byzantine workers.



(α, f) -Byzantine-Resilience (1/2)

- ▶ Assume n workers, where f of them are Byzantine workers.
- ▶ $\alpha \in [0, \pi/2]$ and $f \in \{0, \dots, n\}$.



(α, f) -Byzantine-Resilience (1/2)

- ▶ Assume n workers, where f of them are Byzantine workers.
- ▶ $\alpha \in [0, \pi/2]$ and $f \in \{0, \dots, n\}$.
- ▶ $(G_1, \dots, G_{n-f}) \in (\mathbb{R}^d)^{n-f}$ are i.i.d. random vectors



(α, f) -Byzantine-Resilience (1/2)

- ▶ Assume n workers, where f of them are **Byzantine** workers.
- ▶ $\alpha \in [0, \pi/2]$ and $f \in \{0, \dots, n\}$.
- ▶ $(G_1, \dots, G_{n-f}) \in (\mathbb{R}^d)^{n-f}$ are i.i.d. random vectors
 - $G_i \sim g$
 - $\mathbb{E}[g] = J$, where $J = \nabla J(w)$



(α, f) -Byzantine-Resilience (1/2)

- ▶ Assume n workers, where f of them are **Byzantine** workers.
- ▶ $\alpha \in [0, \pi/2]$ and $f \in \{0, \dots, n\}$.
- ▶ $(G_1, \dots, G_{n-f}) \in (\mathbb{R}^d)^{n-f}$ are **i.i.d.** random vectors
 - $G_i \sim g$
 - $\mathbb{E}[g] = J$, where $J = \nabla J(w)$
- ▶ $(B_1, \dots, B_f) \in (\mathbb{R}^d)^f$ are random vectors, possibly **dependent** between them and the vectors (G_1, \dots, G_{n-f})

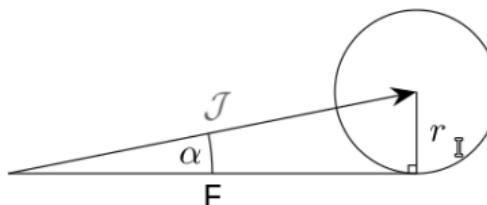


(α, f) -Byzantine-Resilience (2/2)

- ▶ A GAR F is said to be **(α, f) -Byzantine-resilient** if, for any $1 \leq j_1 < \dots < j_f \leq n$, the vector $F(G_1, \dots, B_1, \dots, B_{j_f}, \dots, G_n)$ satisfies:

(α, f) -Byzantine-Resilience (2/2)

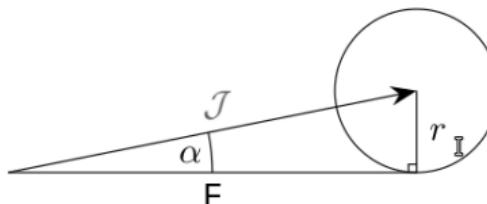
- ▶ A GAR \mathbf{F} is said to be **(α, f) -Byzantine-resilient** if, for any $1 \leq j_1 < \dots < j_f \leq n$, the vector $\mathbf{F}(G_1, \dots, B_1, \dots, B_{j_f}, \dots, G_n)$ satisfies:
 1. Vector \mathbf{F} that is **not too far** from the **real gradient** \mathcal{J} , i.e., $\|\mathbb{E}[\mathbf{F}] - \mathcal{J}\| \leq r$.



[Blanchard et al., Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent, 2017]

(α, f) -Byzantine-Resilience (2/2)

- ▶ A GAR \mathbf{F} is said to be **(α, f) -Byzantine-resilient** if, for any $1 \leq j_1 < \dots < j_f \leq n$, the vector $\mathbf{F}(G_1, \dots, B_1, \dots, B_{j_f}, \dots, G_n)$ satisfies:
 1. Vector \mathbf{F} that is **not too far** from the **real gradient** \mathcal{J} , i.e., $\|\mathbb{E}[\mathbf{F}] - \mathcal{J}\| \leq r$.
 2. Moments of \mathbf{F} should be **controlled** by the moments of the (correct) gradient estimator \mathbf{g} , where $\mathbb{E}[\mathbf{g}] = \mathcal{J}$.



[Blanchard et al., Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent, 2017]



Byzantine-Resilience GAR

- ▶ Median
- ▶ Krum
- ▶ Multi-Krum
- ▶ Brute



Median

- ▶ $n \geq 2f + 1$

Median

- ▶ $n \geq 2f + 1$
- ▶ $\text{median}(x_1, \dots, x_n) = \arg \min_{x \in \mathbb{R}} \sum_{i=1}^n |x_i - x|$



Median

- ▶ $n \geq 2f + 1$
- ▶ $\text{median}(x_1, \dots, x_n) = \arg \min_{x \in \mathbb{R}} \sum_{i=1}^n |x_i - x|$
- ▶ d : the gradient vectors dimension.
- ▶ Geometric median
 $F = \text{GeoMed}(G_1, \dots, G_n) = \arg \min_{G \in \mathbb{R}^d} \sum_{i=1}^n \|G_i - G\|$

Median

- ▶ $n \geq 2f + 1$
- ▶ $\text{median}(x_1, \dots, x_n) = \arg \min_{x \in \mathbb{R}} \sum_{i=1}^n |x_i - x|$
- ▶ d : the gradient vectors dimension.

- ▶ Geometric median

$$F = \text{GeoMed}(G_1, \dots, G_n) = \arg \min_{G \in \mathbb{R}^d} \sum_{i=1}^n \|G_i - G\|$$

- ▶ Marginal median

$$F = \text{MarMed}(G_1, \dots, G_n) = \begin{pmatrix} \text{median}(G_1[1], \dots, G_n[1]) \\ \vdots \\ \text{median}(G_1[d], \dots, G_n[d]) \end{pmatrix} \quad (1)$$



Krum

- $n \geq 2f + 3$



- ▶ $n \geq 2f + 3$
- ▶ Idea: to preclude the vectors that are too far away.

- ▶ $n \geq 2f + 3$
- ▶ Idea: to **preclude** the vectors that are **too far away**.
- ▶ $s(i) = \sum_{i \rightarrow j} \|G_i - G_j\|^2$, the score of the worker **i**.

- ▶ $n \geq 2f + 3$
- ▶ Idea: to **preclude** the vectors that are **too far away**.
- ▶ $s(i) = \sum_{i \rightarrow j} \|G_i - G_j\|^2$, the score of the worker **i**.
- ▶ $i \rightarrow j$ denotes that G_j belongs to the $n - f - 2$ closest vectors to G_i .

- ▶ $n \geq 2f + 3$
- ▶ Idea: to preclude the vectors that are too far away.
- ▶ $s(i) = \sum_{i \rightarrow j} \|G_i - G_j\|^2$, the score of the worker i .
- ▶ $i \rightarrow j$ denotes that G_j belongs to the $n - f - 2$ closest vectors to G_i .
- ▶ $F(G_1, \dots, G_n) = G_{i_*}$

- ▶ $n \geq 2f + 3$
- ▶ Idea: to preclude the vectors that are too far away.
- ▶ $s(i) = \sum_{i \rightarrow j} \|G_i - G_j\|^2$, the score of the worker i .
- ▶ $i \rightarrow j$ denotes that G_j belongs to the $n - f - 2$ closest vectors to G_i .
- ▶ $F(G_1, \dots, G_n) = G_{i_*}$
- ▶ G_{i_*} refers to the worker minimizing the score, $s(i_*) \leq s(i)$ for all i .



Multi-Krum

- ▶ Multi-Krum computes the **score** for each vector proposed (as in Krum).



Multi-Krum

- ▶ Multi-Krum computes the **score** for each vector proposed (as in Krum).
- ▶ It selects **m** vectors G_{1*}, \dots, G_{m*} , which score the **best** ($1 \leq m \leq n - f - 2$).

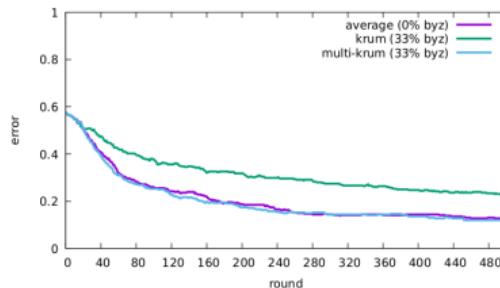


Multi-Krum

- ▶ Multi-Krum computes the **score** for each vector proposed (as in Krum).
- ▶ It selects m vectors G_{1*}, \dots, G_{m*} , which score the **best** ($1 \leq m \leq n - f - 2$).
- ▶ It outputs their average $\frac{1}{m} \sum_i G_{i*}$.

Multi-Krum

- ▶ Multi-Krum computes the **score** for each vector proposed (as in Krum).
- ▶ It selects m vectors G_{1*}, \dots, G_{m*} , which score the **best** ($1 \leq m \leq n - f - 2$).
- ▶ It outputs their average $\frac{1}{m} \sum_i G_{i*}$.
- ▶ The cases $m = 1$ and $m = n$ correspond to **Krum** and **averaging**, respectively.



[Blanchard et al., Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent, 2017]



Brute

- $n \geq 2f + 1$



Brute

- ▶ $n \geq 2f + 1$
- ▶ $\mathcal{Q} = \{G_1, G_2, \dots, G_n\}$

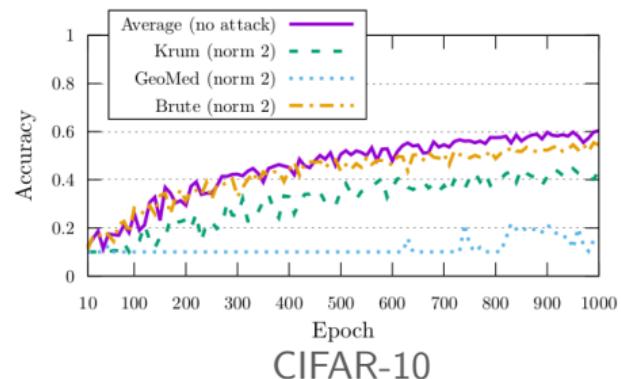
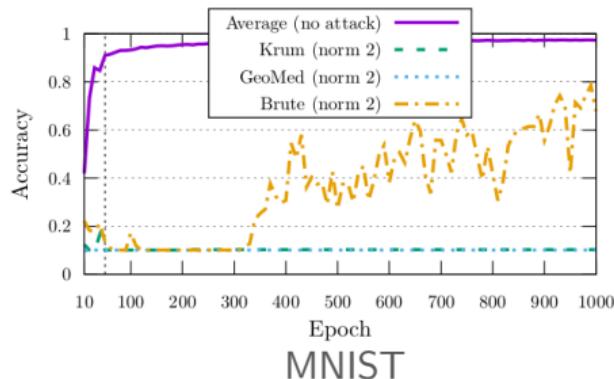


Brute

- ▶ $n \geq 2f + 1$
- ▶ $\mathcal{Q} = \{G_1, G_2, \dots, G_n\}$
- ▶ $\mathcal{R} = \{\mathcal{X} | \mathcal{X} \subset \mathcal{Q}, |\mathcal{X}| = n - f\}$
 - The set of all the **subsets** of $n - f$

- ▶ $n \geq 2f + 1$
- ▶ $\mathcal{Q} = \{G_1, G_2, \dots, G_n\}$
- ▶ $\mathcal{R} = \{\mathcal{X} | \mathcal{X} \subset \mathcal{Q}, |\mathcal{X}| = n - f\}$
 - The set of all the **subsets** of $n - f$
- ▶ $\mathcal{S} = \arg \min_{\mathcal{X} \in \mathcal{R}} (\max_{(G_i, G_j) \in \mathcal{X}^2} (||G_i - G_j||))$
 - Selects the **$n - f$ most clumped gradients** among the submitted ones.

- ▶ $n \geq 2f + 1$
- ▶ $\mathcal{Q} = \{G_1, G_2, \dots, G_n\}$
- ▶ $\mathcal{R} = \{\mathcal{X} | \mathcal{X} \subset \mathcal{Q}, |\mathcal{X}| = n - f\}$
 - The set of all the **subsets** of $n - f$
- ▶ $\mathcal{S} = \arg \min_{\mathcal{X} \in \mathcal{R}} (\max_{(G_i, G_j) \in \mathcal{X}^2} (||G_i - G_j||))$
 - Selects the **$n - f$ most clumped gradients** among the submitted ones.
- ▶ $F(G_1, \dots, G_n) = \frac{1}{n-f} \sum_{G \in \mathcal{S}} G$



[El Mhamdi et al., The Hidden Vulnerability of Distributed Learning in Byzantium, 2018]



Weak Byzantine Resilience

- ▶ Limitation of previous aggregation methods.
- ▶ If gradient dimension $d \gg 1$, then the distance function between two vectors $\|\mathbf{x} - \mathbf{y}\|_p$, cannot distinguish these two cases:



Weak Byzantine Resilience

- ▶ Limitation of previous aggregation methods.
- ▶ If gradient dimension $d \gg 1$, then the distance function between two vectors $\|\mathbf{X} - \mathbf{Y}\|_p$, cannot distinguish these two cases:
 - ▶ 1. Does \mathbf{X} and \mathbf{Y} disagree a bit on each coordinate?



Weak Byzantine Resilience

- ▶ Limitation of previous aggregation methods.
- ▶ If gradient dimension $d \gg 1$, then the distance function between two vectors $\|\mathbf{X} - \mathbf{Y}\|_p$, cannot distinguish these two cases:
 - ▶ 1. Does \mathbf{X} and \mathbf{Y} disagree a bit on each coordinate?
 - ▶ 2. Does \mathbf{X} and \mathbf{Y} disagree a lot on only one?



Strong Byzantine Resilience

- ▶ Ensuring convergence (as in weak Byzantine resilience functions).



Strong Byzantine Resilience

- ▶ Ensuring convergence (as in weak Byzantine resilience functions).
- ▶ Ensures that each coordinate is agreed on by a majority of vectors that were selected by a Byzantine resilient aggregation rule A.



Strong Byzantine Resilience

- ▶ Ensuring convergence (as in weak Byzantine resilience functions).
- ▶ Ensures that each coordinate is agreed on by a majority of vectors that were selected by a Byzantine resilient aggregation rule A.
- ▶ A can be Brute, Krum, Median, etc.



Strong Byzantine Resilience

- ▶ Ensuring convergence (as in weak Byzantine resilience functions).
- ▶ Ensures that each coordinate is agreed on by a majority of vectors that were selected by a Byzantine resilient aggregation rule A.
- ▶ A can be Brute, Krum, Median, etc.
- ▶ Bulyan is a strong Byzantine-resilience algorithm.



The Hidden Vulnerability of Distributed Learning in Byzantium



Bulyan - Step One (1/2)

- ▶ $n \geq 4f + 3$
- ▶ A two step process.



Bulyan - Step One (1/2)

- ▶ $n \geq 4f + 3$
- ▶ A two step process.
- ▶ The first one is to recursively use A to select $\theta = n - 2f$ gradients:



Bulyan - Step One (1/2)

- ▶ $n \geq 4f + 3$
- ▶ A two step process.
- ▶ The first one is to recursively use \mathbf{A} to select $\theta = n - 2f$ gradients:
 1. With \mathbf{A} , choose, among the proposed vectors, the closest one to \mathbf{A} 's output (for Krum this would be the exact output of \mathbf{A}).



Bulyan - Step One (1/2)

- ▶ $n \geq 4f + 3$
- ▶ A two step process.
- ▶ The first one is to recursively use \mathbf{A} to select $\theta = n - 2f$ gradients:
 1. With \mathbf{A} , choose, among the proposed vectors, the closest one to \mathbf{A} 's output (for Krum this would be the exact output of \mathbf{A}).
 2. Remove the chosen gradient from the received set and add it to the selection set \mathbf{S} .



Bulyan - Step One (1/2)

- ▶ $n \geq 4f + 3$
- ▶ A two step process.
- ▶ The first one is to recursively use A to select $\theta = n - 2f$ gradients:
 1. With A , choose, among the proposed vectors, the closest one to A 's output (for Krum this would be the exact output of A).
 2. Remove the chosen gradient from the received set and add it to the selection set S .
 3. Loop back to step 1 if $|S| < \theta$.



Bulyan - Step One (2/2)

- ▶ $\theta = n - 2f \geq 2f + 3$, thus $S = (s_1, \dots, s_\theta)$ contains a majority of non-Byzantine gradients.



Bulyan - Step One (2/2)

- ▶ $\theta = n - 2f \geq 2f + 3$, thus $S = (s_1, \dots, s_\theta)$ contains a majority of non-Byzantine gradients.
- ▶ For each $i \in [1..d]$, the median of the θ coordinates i of the selected gradients is always bounded by coordinates from non-Byzantine submissions.



Bulyan - Step Two

- ▶ The second step is to generate the resulting gradient $\mathbf{F} = (F[1], \dots, F[d])$.



Bulyan - Step Two

- ▶ The second step is to generate the resulting gradient $\mathbf{F} = (F[1], \dots, F[d])$.
- ▶ $\forall i \in [1..d], F[i] = \frac{1}{\beta} \sum_{x \in M[i]} X[i]$



Bulyan - Step Two

- ▶ The second step is to generate the resulting gradient $\mathbf{F} = (F[1], \dots, F[d])$.
- ▶ $\forall i \in [1..d], F[i] = \frac{1}{\beta} \sum_{x \in M[i]} X[i]$
- ▶ $\beta = \theta - 2f \geq 3$



Bulyan - Step Two

- ▶ The second step is to generate the resulting gradient $\mathbf{F} = (F[1], \dots, F[d])$.
- ▶ $\forall i \in [1..d], F[i] = \frac{1}{\beta} \sum_{x \in M[i]} X[i]$
- ▶ $\beta = \theta - 2f \geq 3$
- ▶ $M[i] = \arg \min_{R \subset S, |R|=\beta} (\sum_{x \in R} |X[i] - \text{median}[i]|)$



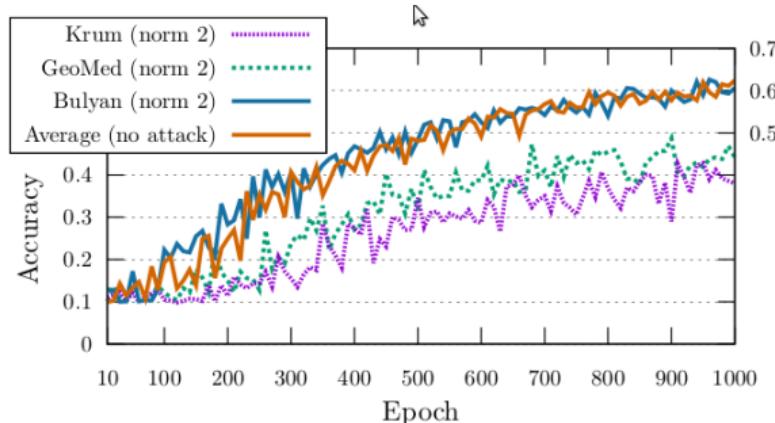
Bulyan - Step Two

- ▶ The second step is to generate the resulting gradient $\mathbf{F} = (F[1], \dots, F[d])$.
- ▶ $\forall i \in [1..d], F[i] = \frac{1}{\beta} \sum_{x \in M[i]} X[i]$
- ▶ $\beta = \theta - 2f \geq 3$
- ▶ $M[i] = \arg \min_{R \subset S, |R|=\beta} (\sum_{x \in R} |X[i] - \text{median}[i]|)$
- ▶ $\text{median}[i] = \arg \min_{m=Y[i], Y \in S} (\sum_{z \in S} |Z[i] - m|)$



Bulyan - Step Two

- ▶ The second step is to generate the resulting gradient $\mathbf{F} = (F[1], \dots, F[d])$.
- ▶ $\forall i \in [1..d], F[i] = \frac{1}{\beta} \sum_{x \in M[i]} X[i]$
- ▶ $\beta = \theta - 2f \geq 3$
- ▶ $M[i] = \arg \min_{R \subset S, |R|=\beta} (\sum_{x \in R} |X[i] - \text{median}[i]|)$
- ▶ $\text{median}[i] = \arg \min_{m=Y[i], Y \in S} (\sum_{z \in S} |Z[i] - m|)$
- ▶ Each i th coordinate of \mathbf{F} is equal to the average of the β closest i th coordinates to the median i th coordinate of the θ selected gradients.



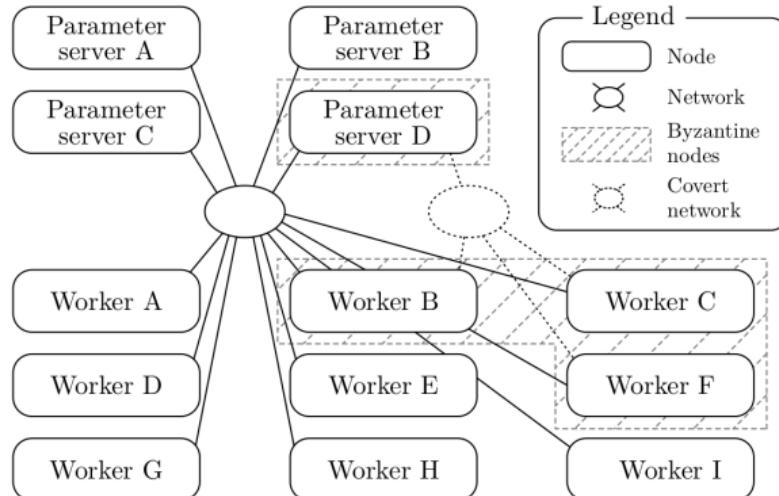
[El Mhamdi et al., The Hidden Vulnerability of Distributed Learning in Byzantium, 2018]



What if parameter servers are Byzantine?



SGD: Decentralized Byzantine Resilience



[El Mhamdi et al., SGD: Decentralized Byzantine Resilience, 2019]



- ▶ Byzantine tolerant learning algorithm that is



- ▶ Byzantine tolerant learning algorithm that is
 1. Resilience to Byzantine workers.



- ▶ Byzantine tolerant learning algorithm that is
 1. Resilience to Byzantine workers.
 2. Resilience to Byzantine parameter servers.



- ▶ Byzantine tolerant learning algorithm that is
 1. Resilience to Byzantine workers.
 2. Resilience to Byzantine parameter servers.
- ▶ GuanYu tolerates up to $\frac{1}{3}$ Byzantine servers and $\frac{1}{3}$ Byzantine workers.



- ▶ Byzantine tolerant learning algorithm that is
 1. Resilience to Byzantine workers.
 2. Resilience to Byzantine parameter servers.
- ▶ GuanYu tolerates up to $\frac{1}{3}$ Byzantine servers and $\frac{1}{3}$ Byzantine workers.
- ▶ GuanYu uses a GAR for aggregating workers' gradients and Median for aggregating models received from servers.



Assumptions and Notations (1/2)

- ▶ Asynchronous network: the **lack of any bound** on communication delays.



Assumptions and Notations (1/2)

- ▶ Asynchronous network: the **lack of any bound** on communication delays.
- ▶ Synchronous training: **bulk-synchronous** training.



Assumptions and Notations (1/2)

- ▶ Asynchronous network: the **lack of any bound** on communication delays.
- ▶ Synchronous training: **bulk-synchronous** training.
 - The parameter server does **not need to wait** for all the workers' gradients to make progress, and vice versa.



Assumptions and Notations (1/2)

- ▶ Asynchronous network: the **lack of any bound** on communication delays.
- ▶ Synchronous training: **bulk-synchronous** training.
 - The parameter server does **not need to wait** for all the workers' gradients to make progress, and vice versa.
 - The **quorums** indicate the **number of messages to wait** before aggregating them.



Assumptions and Notations (2/2)

- ▶ $n_{ps} \geq 3f_{ps} + 3$ the total number of parameter servers, among which f_{ps} are Byzantine.



Assumptions and Notations (2/2)

- ▶ $n_{ps} \geq 3f_{ps} + 3$ the total number of parameter servers, among which f_{ps} are Byzantine.
- ▶ $n_{wr} \geq 3f_{wr} + 3$ the total number of workers, among which f_{wr} are Byzantine.



Assumptions and Notations (2/2)

- ▶ $n_{ps} \geq 3f_{ps} + 3$ the total number of parameter servers, among which f_{ps} are Byzantine.
- ▶ $n_{wr} \geq 3f_{wr} + 3$ the total number of workers, among which f_{wr} are Byzantine.
- ▶ M the coordinate-wise median (used in both workers and servers).



Assumptions and Notations (2/2)

- ▶ $n_{ps} \geq 3f_{ps} + 3$ the total number of parameter servers, among which f_{ps} are Byzantine.
- ▶ $n_{wr} \geq 3f_{wr} + 3$ the total number of workers, among which f_{wr} are Byzantine.
- ▶ M the coordinate-wise median (used in both workers and servers).
- ▶ F the GAR function (used in the servers)



Assumptions and Notations (2/2)

- ▶ $n_{ps} \geq 3f_{ps} + 3$ the total number of parameter servers, among which f_{ps} are Byzantine.
- ▶ $n_{wr} \geq 3f_{wr} + 3$ the total number of workers, among which f_{wr} are Byzantine.
- ▶ M the coordinate-wise median (used in both workers and servers).
- ▶ F the GAR function (used in the servers)
- ▶ $2f_{ps} + 3 \leq q_{ps} \leq n_{ps} - f_{ps}$ the quorum used for M .



Assumptions and Notations (2/2)

- ▶ $n_{ps} \geq 3f_{ps} + 3$ the total number of parameter servers, among which f_{ps} are Byzantine.
- ▶ $n_{wr} \geq 3f_{wr} + 3$ the total number of workers, among which f_{wr} are Byzantine.
- ▶ M the coordinate-wise median (used in both workers and servers).
- ▶ F the GAR function (used in the servers)
- ▶ $2f_{ps} + 3 \leq q_{ps} \leq n_{ps} - f_{ps}$ the quorum used for M .
- ▶ $2f_{wr} + 3 \leq q_{wr} \leq n_{wr} - f_{wr}$ the quorum used for F .



Assumptions and Notations (2/2)

- ▶ $n_{ps} \geq 3f_{ps} + 3$ the total number of parameter servers, among which f_{ps} are Byzantine.
- ▶ $n_{wr} \geq 3f_{wr} + 3$ the total number of workers, among which f_{wr} are Byzantine.
- ▶ M the coordinate-wise median (used in both workers and servers).
- ▶ F the GAR function (used in the servers)
- ▶ $2f_{ps} + 3 \leq q_{ps} \leq n_{ps} - f_{ps}$ the quorum used for M .
- ▶ $2f_{wr} + 3 \leq q_{wr} \leq n_{wr} - f_{wr}$ the quorum used for F .
- ▶ d the dimension of the parameter space \mathbb{R}^d .



GuanYu Algorithm - Step 1

- ▶ At each step t , each non-Byzantine server i broadcasts its current parameter vector w_i^t to every worker.



GuanYu Algorithm - Step 1

- ▶ At each step t , each non-Byzantine server i broadcasts its current parameter vector w_i^t to every worker.
- ▶ Each non-Byzantine worker j aggregates with M the q_{ps} first received w^t .



GuanYu Algorithm - Step 1

- ▶ At each step t , each **non-Byzantine server i** broadcasts its current **parameter vector w_i^t** to **every worker**.
- ▶ Each **non-Byzantine worker j** aggregates with M the q_{ps} first received w^t .
- ▶ And computes an estimate G_j^t of the gradient at the aggregated parameters.



GuanYu Algorithm - Step 2

- ▶ Each non-Byzantine worker j broadcasts its computed gradient estimation G_j^t to every parameter server.



GuanYu Algorithm - Step 2

- ▶ Each non-Byzantine worker j broadcasts its computed gradient estimation G_j^t to every parameter server.
- ▶ Each non-Byzantine parameter server i aggregates with F the q_{wr} first received G^t .



GuanYu Algorithm - Step 2

- ▶ Each non-Byzantine worker j broadcasts its computed gradient estimation G_j^t to every parameter server.
- ▶ Each non-Byzantine parameter server i aggregates with F the q_{wr} first received G^t .
- ▶ And performs a local parameter update with the aggregated gradient, resulting in \bar{w}_i^t .



GuanYu Algorithm - Step 3

- ▶ Each **non-Byzantine parameter server i** broadcasts \bar{w}_i^{t+1} to every other **parameter servers**.



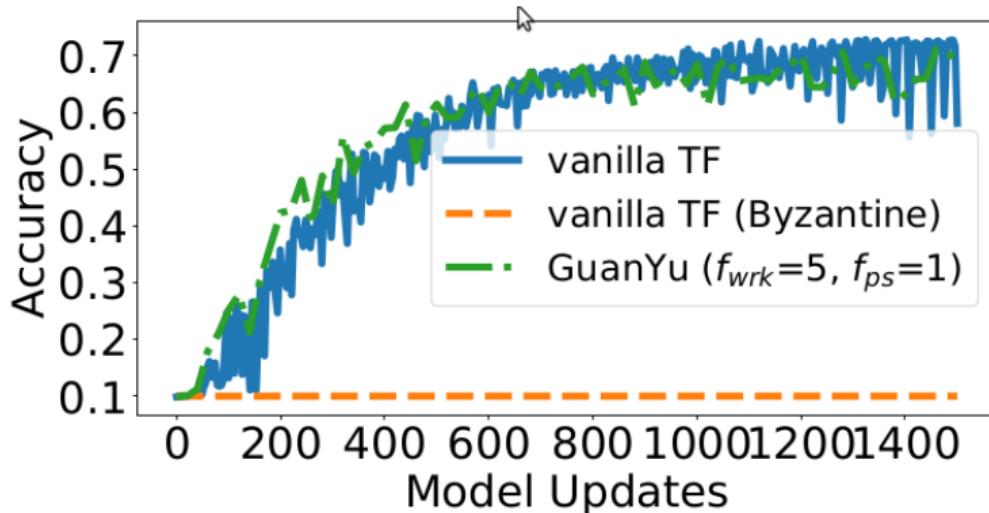
GuanYu Algorithm - Step 3

- ▶ Each non-Byzantine parameter server i broadcasts \bar{w}_i^{t+1} to every other parameter servers.
- ▶ They aggregate with M the q_{ps} first received \bar{w}_k^{t+1} .



GuanYu Algorithm - Step 3

- ▶ Each non-Byzantine parameter server i broadcasts \bar{w}_i^{t+1} to every other parameter servers.
- ▶ They aggregate with M the q_{ps} first received \bar{w}_k^{t+1} .
- ▶ This aggregated parameter vector is \bar{w}_i^{t+1} .



[El Mhamdi et al., SGD: Decentralized Byzantine Resilience, 2019]



Summary



Summary

- ▶ Integrity in data-parallel learning
- ▶ Weak Byzantine resilience
- ▶ Strong Byzantine resilience
- ▶ Byzantine parameter servers



Reference

- ▶ Xie et al., Generalized Byzantine-tolerant SGD, 2018
- ▶ Blanchard et al., Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent, 2017
- ▶ El Mhamdi et al., The Hidden Vulnerability of Distributed Learning in Byzantium, 2018
- ▶ Damaskinos et al., AGGREGATOR: Byzantine Machine Learning via Robust Gradient Aggregation, 2019
- ▶ El Mhamdi et al., SGD: Decentralized Byzantine Resilience, 2019
- ▶ El Mhamdi et al., Fast Machine Learning with Byzantine Workers and Servers, 2019



Questions?