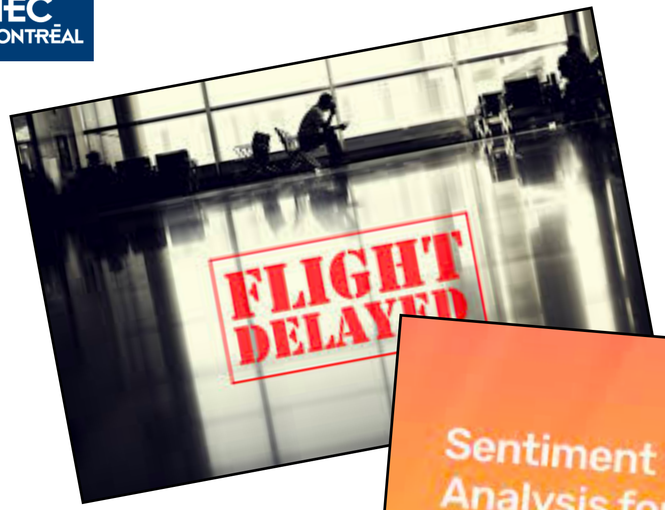
Machine Learning for Large-Scale Data Analysis and Decision Making (MATH80629A) Winter 2022

Week #9 - **Summary**

Projects


FLIGHT
DELAYED


Sentiment
Analysis for
Yelp Reviews
Neutral
Positive
Negative


surprised: 0.68
disgusted: 0.88
sad: 0.92


DIABETES


THE GREY
THE REVENANT


CREDIT CARD
1234 5678 90

Announcement

- **Homework 1:** due **March 9, 2021**
  late submission policy (10 points penalty) **due March 16, 2021**

- **Homework 2:** due **March 14, 2021**
  Upload answer to question 1 to Gradescope (not ZoneCours)

- Next week: no video capsules, and no quiz

Today

- **Sixth Quiz** on Gradescope!
- Summary of Parallel computational paradigms for large-scale data processing
- Brief summary of what you have seen so far
- Q&A

Quiz 5

Login to your Gradescope account

Data & Computation

Data & Computation

- **We generate massive quantities of data**

Data & Computation

- **We generate massive quantities of data**

  1. **Google 4K searches/s, Twitter: 6K tweets/s, Amazon: 100s sold products/s** (source: internetlifestats.com)

Data & Computation

- **We generate massive quantities of data**

  1. **Google 4K searches/s, Twitter: 6K tweets/s, Amazon: 100s sold products/s** (source: internetlifestats.com)

  2. **Banks, insurance companies, etc.**

Data & Computation

- **We generate massive quantities of data**

  1. **Google 4K searches/s, Twitter: 6K tweets/s, Amazon: 100s sold products/s** (source: internetlifestats.com)

  2. **Banks, insurance companies, etc.**

  3. **Modestly-sized websites**

Data & Computation

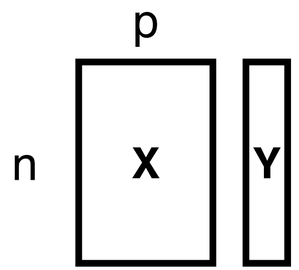
- **We generate massive quantities of data**

  1. **Google 4K searches/s, Twitter: 6K tweets/s, Amazon: 100s sold products/s** (source: internetlifestats.com)

  2. **Banks, insurance companies, etc.**

  3. **Modestly-sized websites**

- **Both large n and large p**



Data & Computation

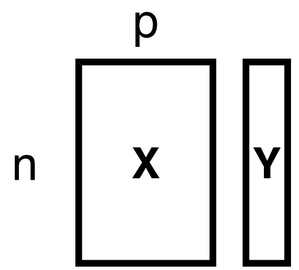
- **We generate massive quantities of data**

  1. **Google 4K searches/s, Twitter: 6K tweets/s, Amazon: 100s sold products/s** (source: internetlifestats.com)

  2. **Banks, insurance companies, etc.**

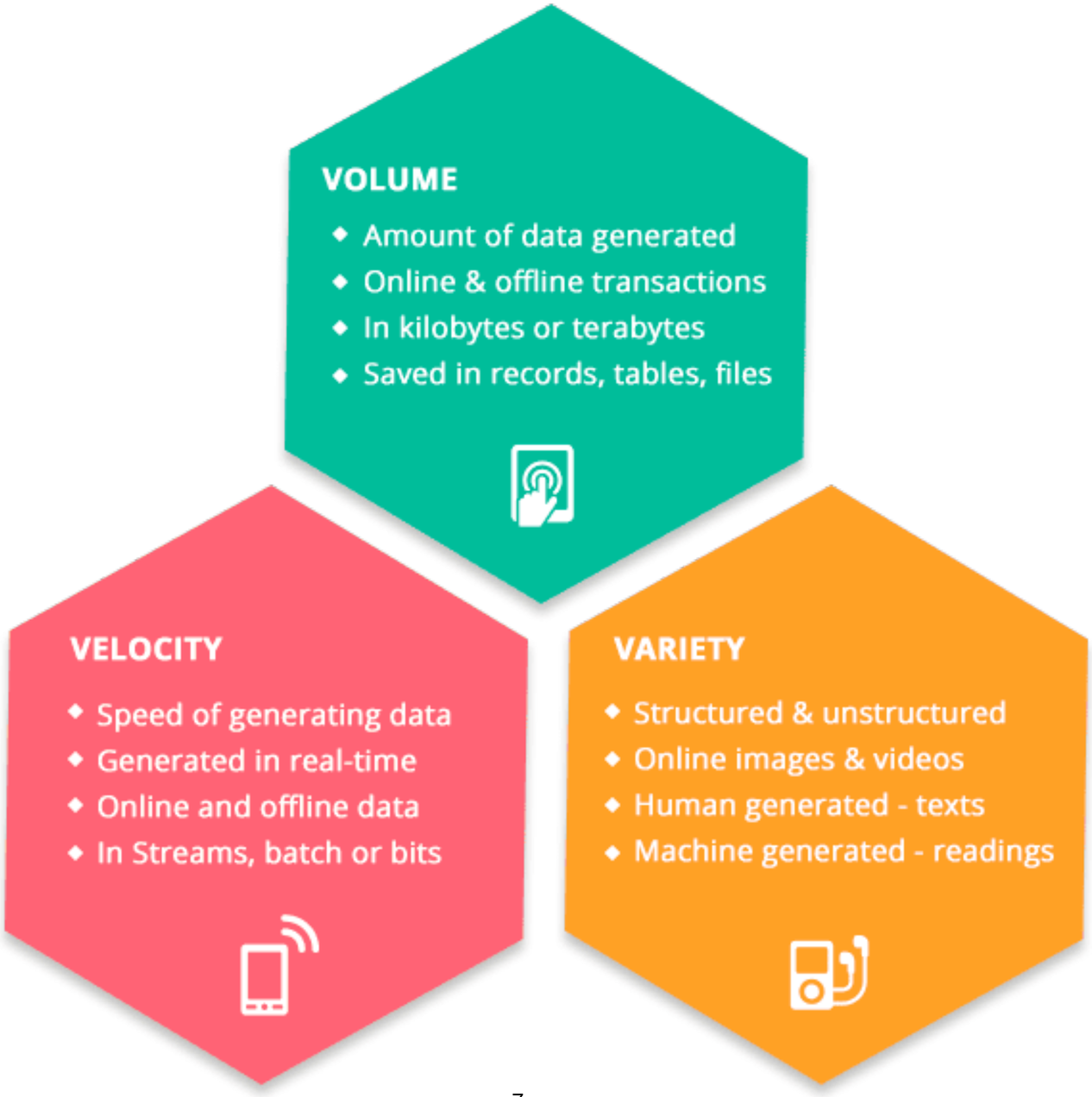
  3. **Modestly-sized websites**

- **Both large n and large p**

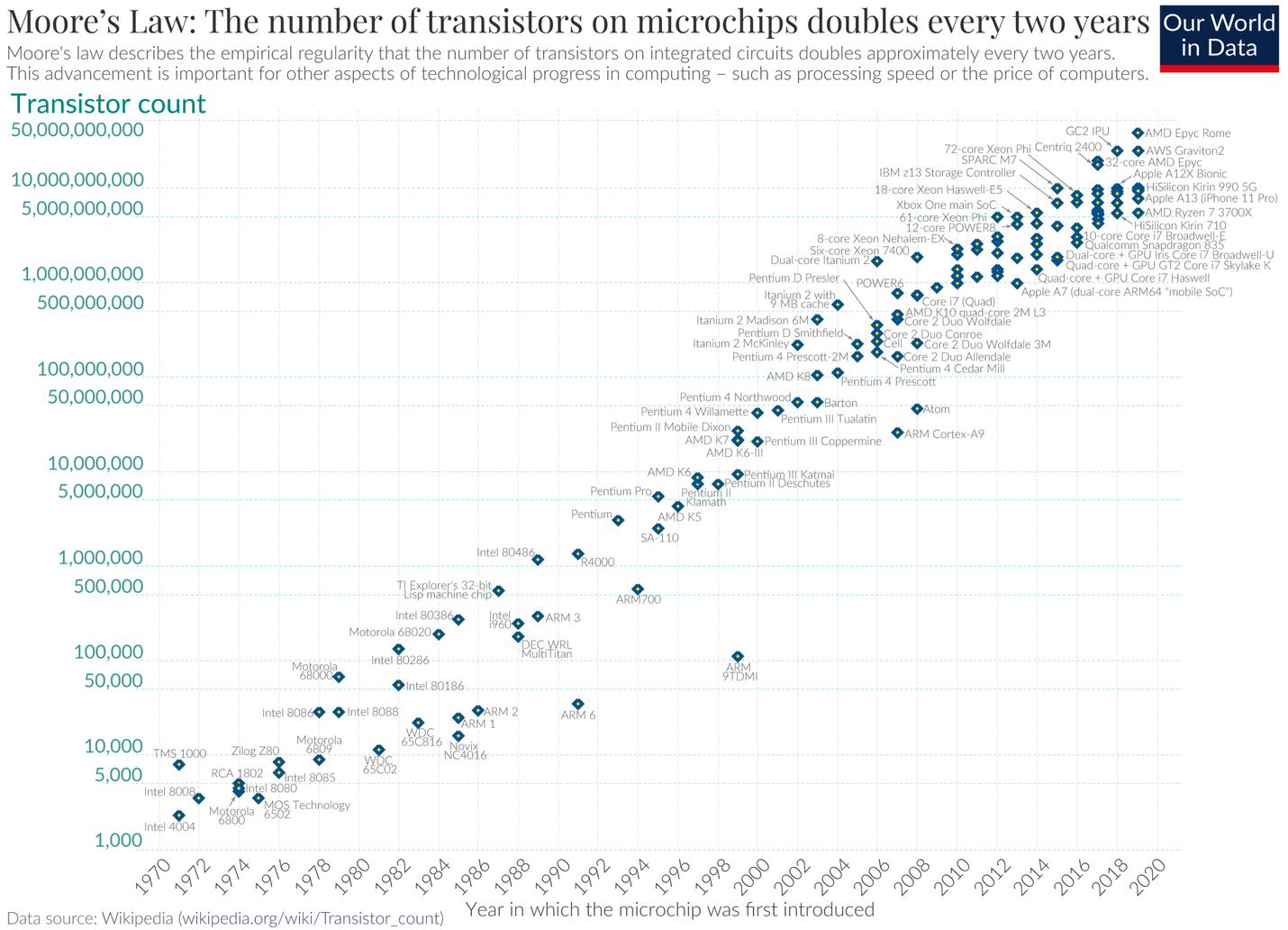


- **In general computation will scale up with the data**

THE 3Vs OF BIG DATA



Moore’s law


Moore’s Law: The number of transistors on microchips doubles every two years
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years.
This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.
Our World in Data
Transistor count
50,000,000,000
10,000,000,000
5,000,000,000
1,000,000,000
500,000,000
100,000,000
50,000,000
10,000,000
5,000,000
1,000,000
500,000
100,000
50,000
10,000
5,000
1,000
GC2 IPU
AMD Epyc Rome
72-core Xeon Phi
Centriq 2400
AWS Graviton2
SPARC M7
32-core AMD Epyc
IBM z13 Storage Controller
Apple A12X Bionic
18-core Xeon Haswell-E5
HiSilicon Kirin 990 5G
Apple A13 (iPhone 11 Pro)
Xbox One main SoC
61-core Xeon Phi
AMD Ryzen 7 3700X
12-core POWER8
HiSilicon Kirin 710
8-core Xeon Nehalem-EX
10-core Core i7 Broadwell-E
Six-core Xeon 7400
Qualcomm Snapdragon 835
Dual-core Itanium 2
Dual-core + GPU Iris Core i7 Broadwell-U
Quad-core + GPU GT2 Core i7 Skylake K
Pentium D Presler
POWER6
Quad-core + GPU Core i7 Haswell
Itanium 2 with 9 MB cache
Apple A7 (dual-core ARM64 "mobile SoC")
Core i7 (Quad)
Itanium 2 Madison 6M
AMD K10 quad-core 2M L3
Core 2 Duo Wolfdale
Pentium D Smithfield
Core 2 Duo Conroe
Itanium 2 McKinley
Cell
Core 2 Duo Wolfdale 3M
Pentium 4 Prescott-2M
Core 2 Duo Allendale
Pentium 4 Cedar Mill
AMD K8
Pentium 4 Prescott
Pentium 4 Northwood
Barton
Pentium 4 Willamette
Atom
Pentium III Tualatin
Pentium II Mobile Dixon
ARM Cortex-A9
AMD K7
Pentium III Coppermine
AMD K6-III
AMD K6
Pentium III Katmai
Pentium II Deschutes
Pentium Pro
Pentium II Klamath
Pentium
AMD K5
SA-110
Intel 80486
R4000
TI Explorer's 32-bit Lisp machine chip
ARM700
Intel 80386
Intel i960
ARM 3
Motorola 68020
DEC WRL MultiTitan
Intel 80286
ARM 9TDMI
Motorola 68000
Intel 80186
Intel 8086
Intel 8088
ARM 2
ARM 6
ARM 1
WDC 65C816
Motorola 6809
Novix NC4016
TMS 1000
Zilog Z80
WDC 65C02
RCA 1802
Intel 8085
Intel 8008
Intel 8080
MOS Technology 6502
Motorola 6800
Intel 4004
1970
1972
1974
1976
1978
1980
1982
1984
1986
1988
1990
1992
1994
1996
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
Year in which the microchip was first introduced
Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
OurWorldinData.org – Research and data to make progress against the world's largest problems.
Licensed under CC-BY by the authors Hannah Ritchie and Max Roser.

Modern Computation paradigms

Modern Computation paradigms

- Floating point operations per second (Flop)
- Smart phone ~ 0.6 TFlops
- 1 Tera: 1,000 Giga

Modern Computation paradigms

- Floating point operations per second (Flop)
- Smart phone ~ 0.6 TFlops
- 1 Tera: 1,000 Giga

1. **“Single” computers**

- **Large Computers**
  - **513, 855 TFlops**

https://www.top500.org/lists/top500/list/2020/06/

Photo from Riken

Modern Computation paradigms

- Floating point operations per second (Flop)
- Smart phone ~ 0.6 TFlops
- 1 Tera: 1,000 Giga

1. “Single” computers

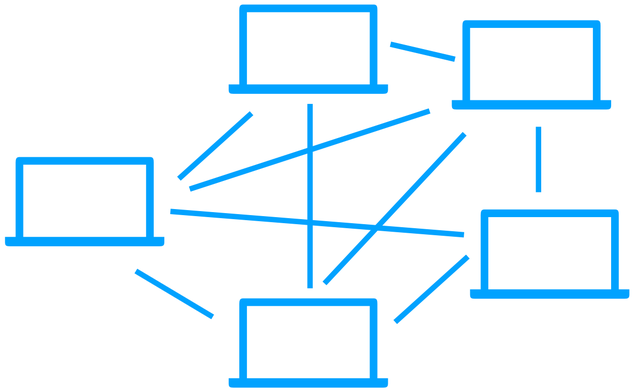
- **Large Computers**
  - **513, 855 TFlops**

https://www.top500.org/lists/top500/list/2020/06/

Photo from Riken

2. Distributed computation

- **~200, 000 TFlops (Folding@home)**



Modern Computation paradigms

- Floating point operations per second (Flop)
- Smart phone ~ 0.6 TFlops
- 1 Tera: 1,000 Giga

1. “Single” computers

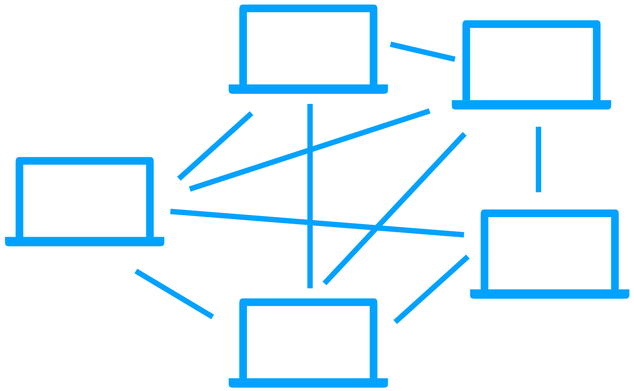
- **Large Computers**
  - **513, 855 TFlops**

https://www.top500.org/lists/top500/list/2020/06/

Photo from Riken

2. Distributed computation

- **~200, 000 TFlops (Folding@home)**



3. Specialized hardware

- **Focusses on subset of operations**
  - **Graphical Processing Unit (GPU), Field Programmable Gated Array (FPGA)**
  - **~10 TFlops**

Modern Computation paradigms

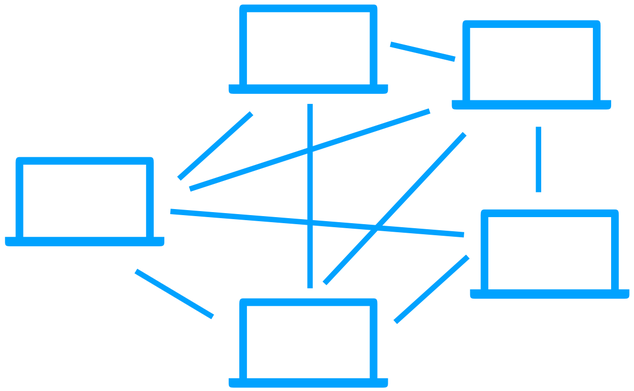
- Floating point operations per second (Flop)
- Smart phone ~ 0.6 TFlops
- 1 Tera: 1,000 Giga

1. **“Single” computers**
   - **Large Computers**
     - **513, 855 TFlops**

https://www.top500.org/lists/top500/list/2020/06/

Photo from Riken

2. **Distributed computation**
   - **~200, 000 TFlops (Folding@home)**



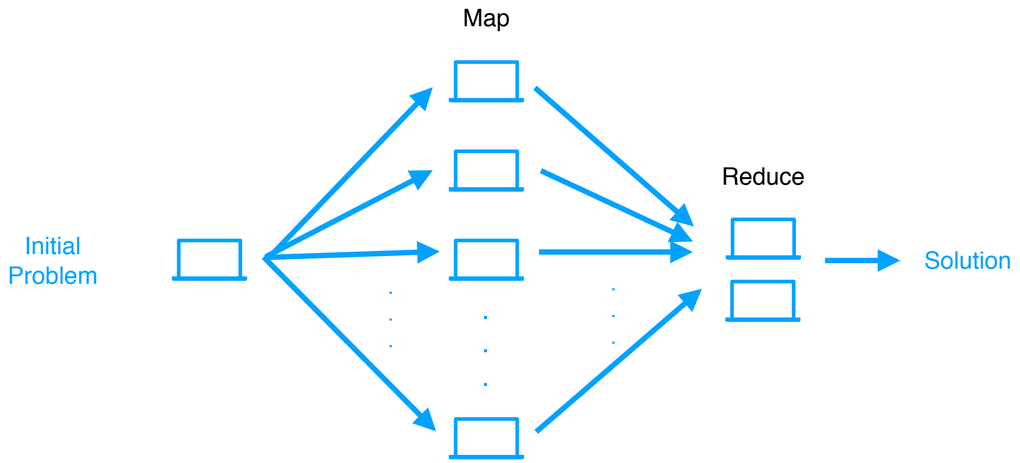
3. **Specialized hardware**
   - **Focusses on subset of operations**
     - **Graphical Processing Unit (GPU), Field Programmable Gated Array (FPGA)**
   - **~10 TFlops**

MapReduce
(Apache) Hadoop

- **Two types of tasks:**

  A. **Map:** Solve a subproblem (filtering operation)

  B. **Reduce:** Combine the results of map workers (summary operation)



MapReduce is quite versatile

- **A few examples of “map-reduceable” problems:**
  - **Intuition: Your problem needs to be decomposable into map functions and reduce functions**
  - Sorting, filtering, distinct values, basic statistics
  - Finding common friends, sql-like queries, sentiment analysis

MapReduce for machine learning

1. **Training linear regression**

   - Reminder: there is a closed-form solution

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$$

MapReduce for machine learning

1. **Training linear regression**

   - Reminder: there is a closed-form solution

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$$

$$\mathbf{w} = (\sum_{ij} \mathbf{X}_i^\top \mathbf{X}_j)^{-1} (\sum_{i} \mathbf{X}_i^\top \mathbf{Y}_i)$$

MapReduce for machine learning

1. **Training linear regression**

- Reminder: there is a closed-form solution

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$$

$$\mathbf{w} = (\sum_{ij} \mathbf{X}_i^\top \mathbf{X}_j)^{-1} (\sum_{i} \mathbf{X}_i^\top \mathbf{Y}_i)$$

- Each term in the sums can be computed independently

MapReduce for machine learning

1. **Training linear regression**

- Reminder: there is a closed-form solution

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$$

$$\mathbf{w} = (\sum_{ij} \mathbf{x}_i^\top \mathbf{x}_j)^{-1} (\sum_{i} \mathbf{x}_i^\top \mathbf{y}_i)$$

- Each term in the sums can be computed independently

A. **Map**

$\mathbf{x}_0^\top \mathbf{x}_1$

MapReduce for machine learning

1. **Training linear regression**

   - Reminder: there is a closed-form solution

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$$

$$\mathbf{w} = (\sum_{ij} \mathbf{X}_i^\top \mathbf{X}_j)^{-1} (\sum_{i} \mathbf{X}_i^\top \mathbf{Y}_i)$$

   - Each term in the sums can be computed independently

A. **Map**

$$\mathbf{X}_0^\top \mathbf{X}_1$$

2. **Other models we studied have a closed form solution** (e.g., Naive Bayes and LDA)

MapReduce for machine learning

1. **Training linear regression**

   - Reminder: there is a closed-form solution

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$$

$$\mathbf{w} = (\sum_{ij} \mathbf{X}_i^\top \mathbf{X}_j)^{-1} (\sum_{i} \mathbf{X}_i^\top \mathbf{Y}_i)$$

   - Each term in the sums can be computed independently

A. **Map**

$$\mathbf{X}_0^\top \mathbf{X}_1$$

2. **Other models we studied have a closed form solution** (e.g., Naive Bayes and LDA)

3. **Hyper-parameter search**

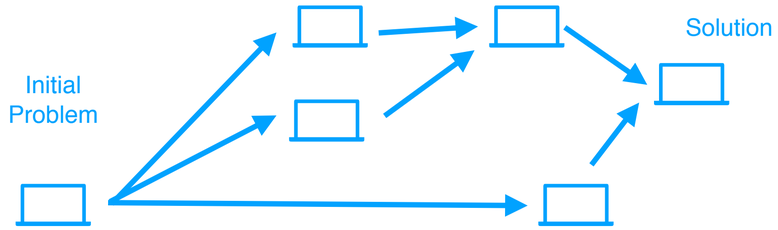
   - A neural network with 2 hidden layers and 5 hidden units per layer and another with 3 hidden layers and 10 hidden units

Shortcomings of MapReduce

- **Many models are fitted with iterative algorithms**
  - **Gradient descent:**
    1. Find the gradient for the current set parameters
    2. Update the parameters with the gradient
- **Not ideal for MapReduce**
  - Would require several iterations of MapReduce
  - Each time the data is read/written from/to the disk

(Apache) Spark

- **Advantages over MapReduce**



1. Less restrictive computations graph
   (DAG instead of Map then Reduce)

   - Doesn't have to write to disk in-between operations

2. Richer set of transformations

   - map, filter, cartesian, union, intersection, distinct, etc.

3. In-memory processing

Parallel gradient descent

- **Logistic Regression**

$$y = \frac{1}{1 + \exp(-w_0 - w_1 x_1 - w_2 x_2 - \ldots - w_p x_p)}$$

Parallel gradient descent

- **Logistic Regression**

$$y = \frac{1}{1 + \exp(-w_0 - w_1 x_1 - w_2 x_2 - \ldots - w_p x_p)}$$

- **No closed-form solution, can use gradients**

$$\frac{\partial \, \text{Loss}(Y, X, w)}{\partial w_i}$$

Parallel gradient descent

- **Logistic Regression**

$$y = \frac{1}{1 + \exp(-w_0 - w_1 x_1 - w_2 x_2 - \ldots - w_p x_p)}$$

- **No closed-form solution, can use gradients**

$$\frac{\partial \, \text{Loss}(Y, X, w)}{\partial w_i}$$

- **Loss functions are often decomposable**

$$\frac{\partial \, \sum_j \text{Loss}(Y_j, X_j, w)}{\partial w_i}$$

Parallel gradient descent

- **Logistic Regression**

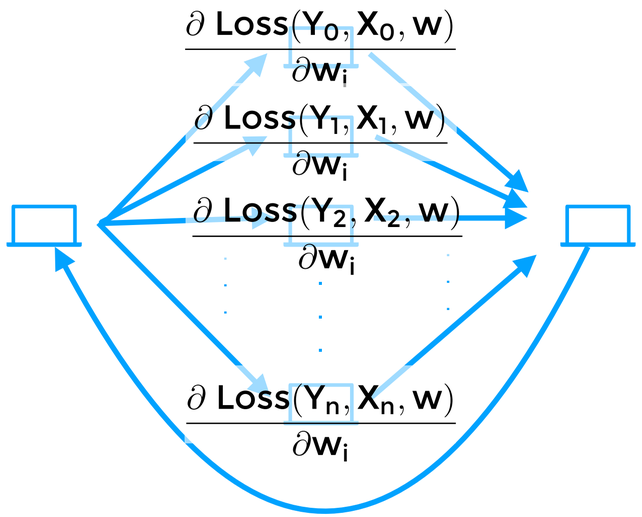
$$y = \frac{1}{1 + \exp(-w_0 - w_1 x_1 - w_2 x_2 - \ldots - w_p x_p)}$$

- **No closed-form solution, can use gradients**

$$\frac{\partial \text{ Loss}(Y, X, w)}{\partial w_i}$$

- **Loss functions are often decomposable**

$$\frac{\partial \sum_j \text{Loss}(Y_j, X_j, w)}{\partial w_i}$$



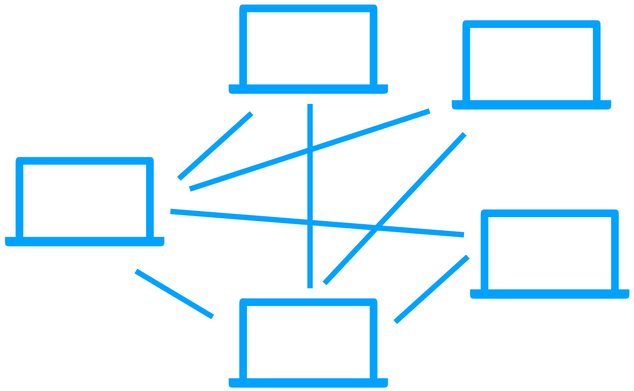
Modern Computation paradigms

1. “Single” computers
   - Large Computers
     - 513, 855 TFlops

https://www.top500.org/lists/top500/list/2020/06/

Photo from Riken

2. Distributed computation
   - ~200, 000 TFlops (Folding@home)



3. Specialized hardware
   - Focusses on subset of operations
     - Graphical Processing Unit (GPU), Field Programmable Gated Array (FPGA)
     - ~10 TFlops

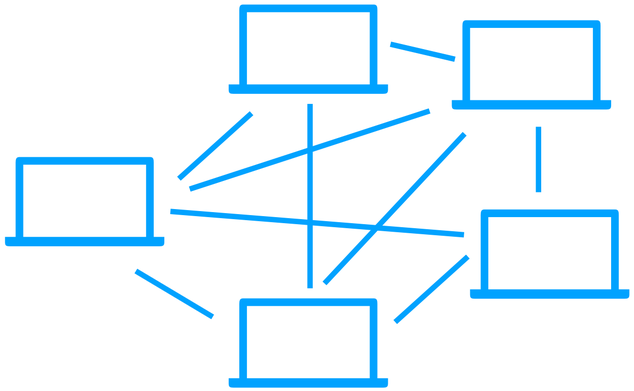
Modern Computation paradigms

1. “Single” computers
   - Large Computers
     - 513, 855 TFlops

https://www.top500.org/lists/top500/list/2020/06/

Photo from Riken

2. Distributed computation
   - ~200, 000 TFlops (Folding@home)



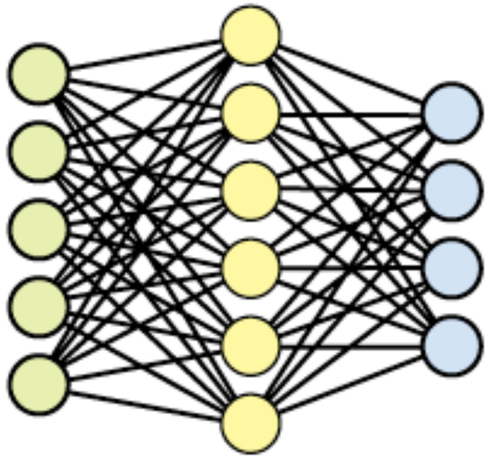
3. Specialized hardware
   - Focusses on subset of operations
     - Graphical Processing Unit (GPU), Field Programmable Gated Array (FPGA)
     - ~10 TFlops

Equipment

http://www.personal.psu.edu/users/d/l/dlm99/cpu.html

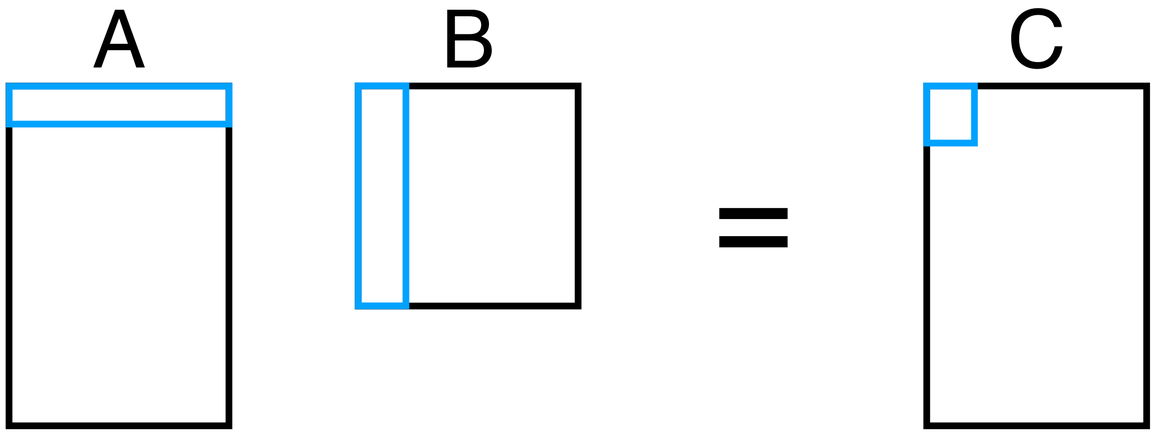
- Central Processing Unit (CPU)
- Computer brain
- Executes all instructions coming from programs
- Arithmetic calculations, reads / writes, etc

Neural Networks

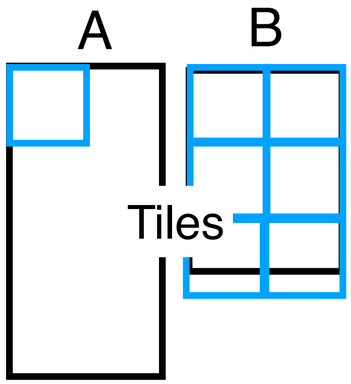


- **Linear algebra:**
  - Multiplication *vector X matrix*
    - Calculate the activation of a neuron
    - Calculate the activation of a neuron for several
  - Matrix X matrix multiplication
    - Calculate the activation of several neurons for several data
- The dimensions of these objects depend on the size of the data (mini batch) and the size of the networks

Parallelize a Neural Networks

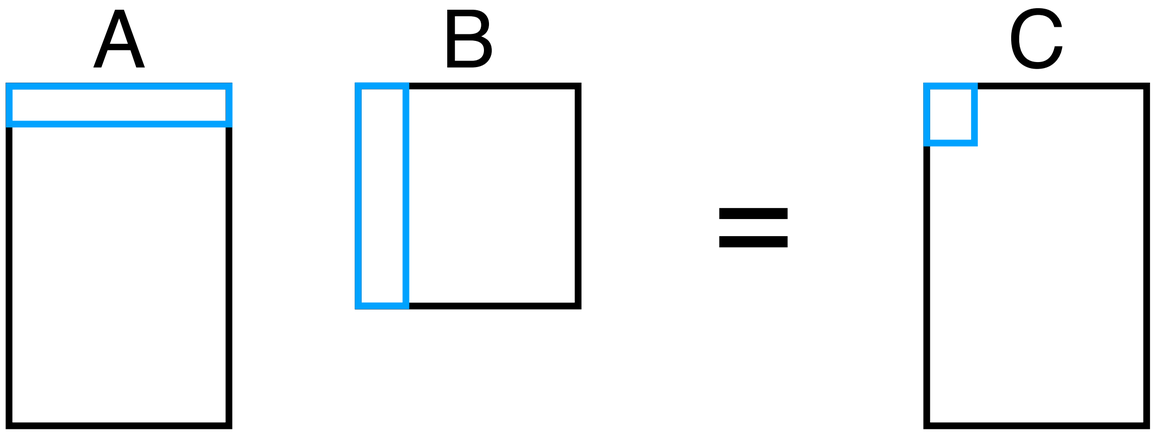


$$C = AB$$



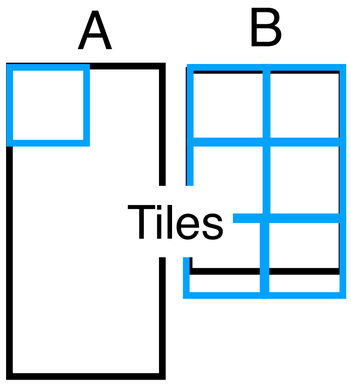
- Values of **C** can be calculated independently of each other
- Matrix multiplications can be parallelized
- By parallelizing, one can obtain very significant computational gains
- Depends on the size of **C**
- In practice we cut into tiles
- Note: this is not the case for all transactions on matrices. In particular, calculate the inverse of a matrix

Parallelize a Neural Networks



$$C = AB$$

$$C_{ij} = \sum_{k} A_{ik} B_{kj}$$



- Values of **C** can be calculated independently of each other
- Matrix multiplications can be parallelized
- By parallelizing, one can obtain very significant computational gains
- Depends on the size of **C**
- In practice we cut into tiles
- Note: this is not the case for all transactions on matrices. In particular, calculate the inverse of a matrix

Specialized equipment

NVidia

- Graphical Processing Unit (GPU)
- Originally used for games
- Specialized for linear algebra operations
- Quick access to memory
- Several GPUs per computer

https://cryptomining-blog.com/tag/multi-gpu-mining-rig/

CPU or GPU?

- **Memory Bandwidth**: Bandwidth is one of the main reasons why GPUs are faster for computing than CPUs.
  CPU takes up the jobs sequentially and it has a fewer number of cores than its counterpart,GPU.

- **Parallelism:** For large chunks of memory, GPUs provide the best memory bandwidth while having almost no drawback due to latency via thread parallelism.

- **Data:** The larger the computations, the more is the advantage of GPU over CPU.

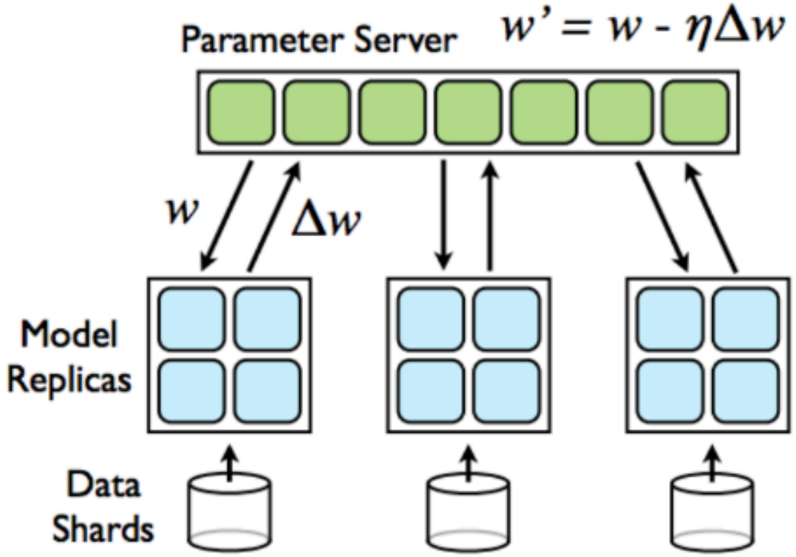
- **Optimization**: Optimizing the tasks are far easier in CPU than in GPU

- **Cost:** The power cost of GPU is higher than CPU.

 https://medium.com/@shachishah.ce/do-we-really-need-gpu-for-deep-learning-47042c02efe2

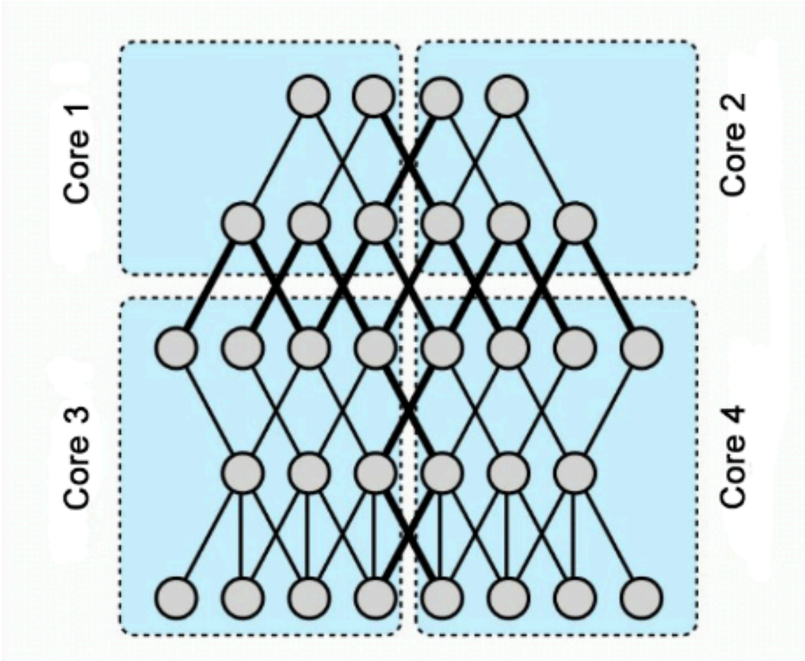
Parallel SGD

Data Parallelism



Every core/GPU trains the full model given partitioned data.

Model Parallelism

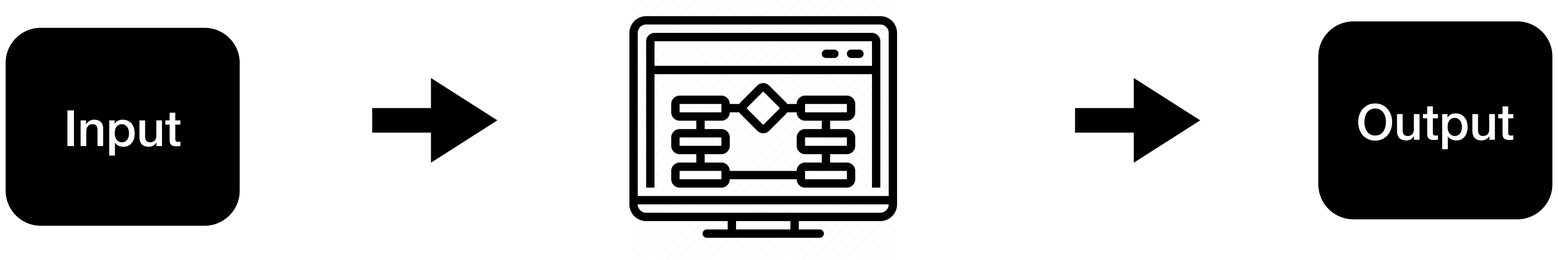


Every core/GPU train a partitioned model given full data.

Even more Specialized equipment

- Tensorflow Processing Unit (TPU)
- Developed by Google for Neural Networks (ASIC)
- For matrix multiplication operations
- Training and testing
- Lower operating precision than GPUs
- Several TPUs per “computer”

Large Scale Computation



- Model Simplification
- Optimization approximation
- Computation parallelism

Wang, Meng, et al. "A survey on large-scale machine learning." *IEEE Transactions on Knowledge and Data Engineering* (2020).

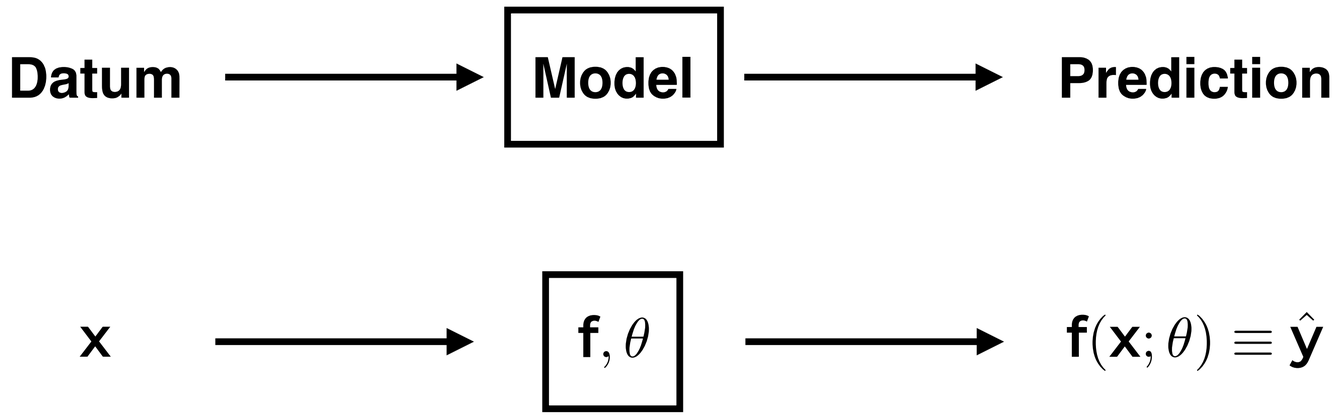
Example

https://colab.research.google.com/github/lcharlin/80-629/blob/master/week10-ParallelComputations/CPU_GPU_TPU.ipynb

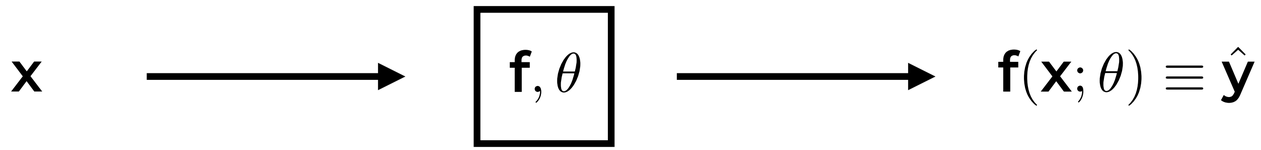
- It takes few changes to our code to use a GPU
- Beware of CPU-GPU transfers

- **Brief summary of what we have seen so far**
- **Explain concepts within a single framework**
- **Focus on a few more advanced concepts**

Supervised Machine Learning

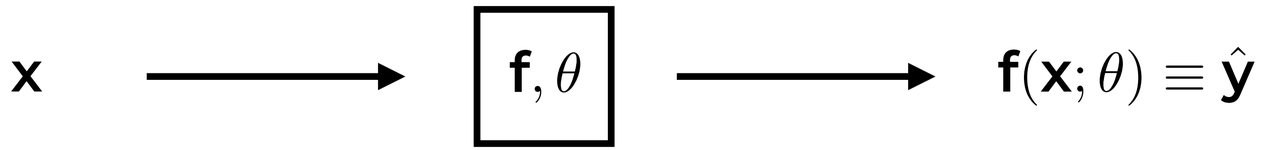


Loss function

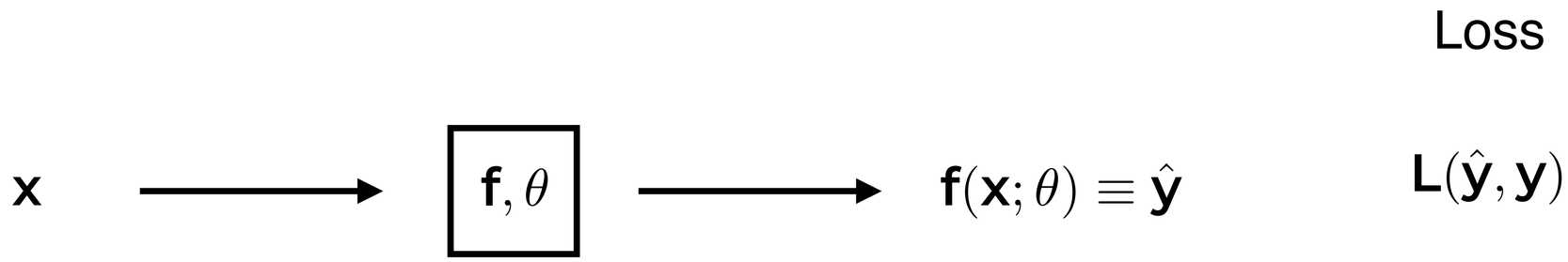


Loss function

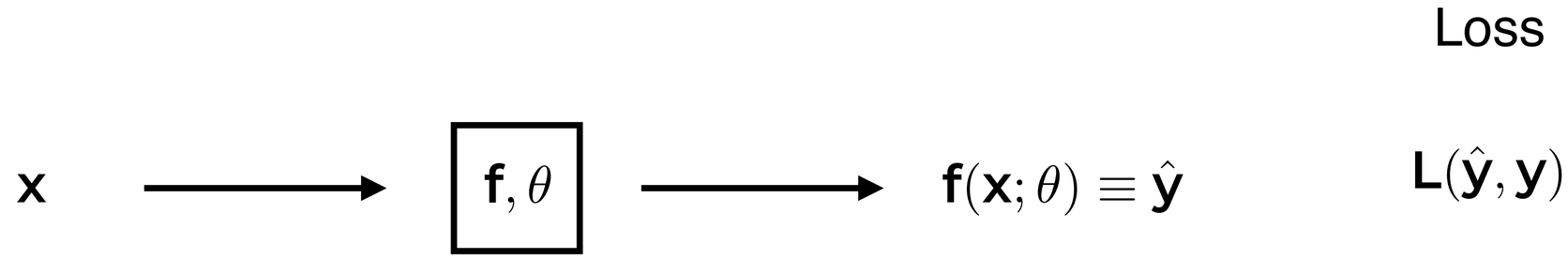
Loss



Loss function



Loss function

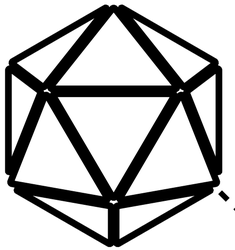


Different losses for different types of y's

$y \in \mathcal{R}$	Regression	$(\hat{y} - y)^2$
y categorical e.g., $\{cat, dog, bird\}$	Classification	accuracy
$y \in \{0, 1\}$	Binary Classification	AUC

Learning Process

Distribution over (x,y): P(x,y)



Loss

$x_{train} \rightarrow \boxed{f, \theta} \rightarrow \hat{y}_{train}$

$L(\hat{y}_{train}, y_{train})$

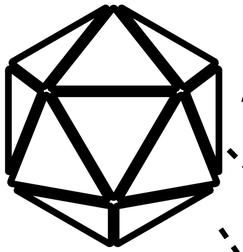
$x_{test} \rightarrow \boxed{f, \hat{\theta}} \rightarrow \hat{y}_{test}$

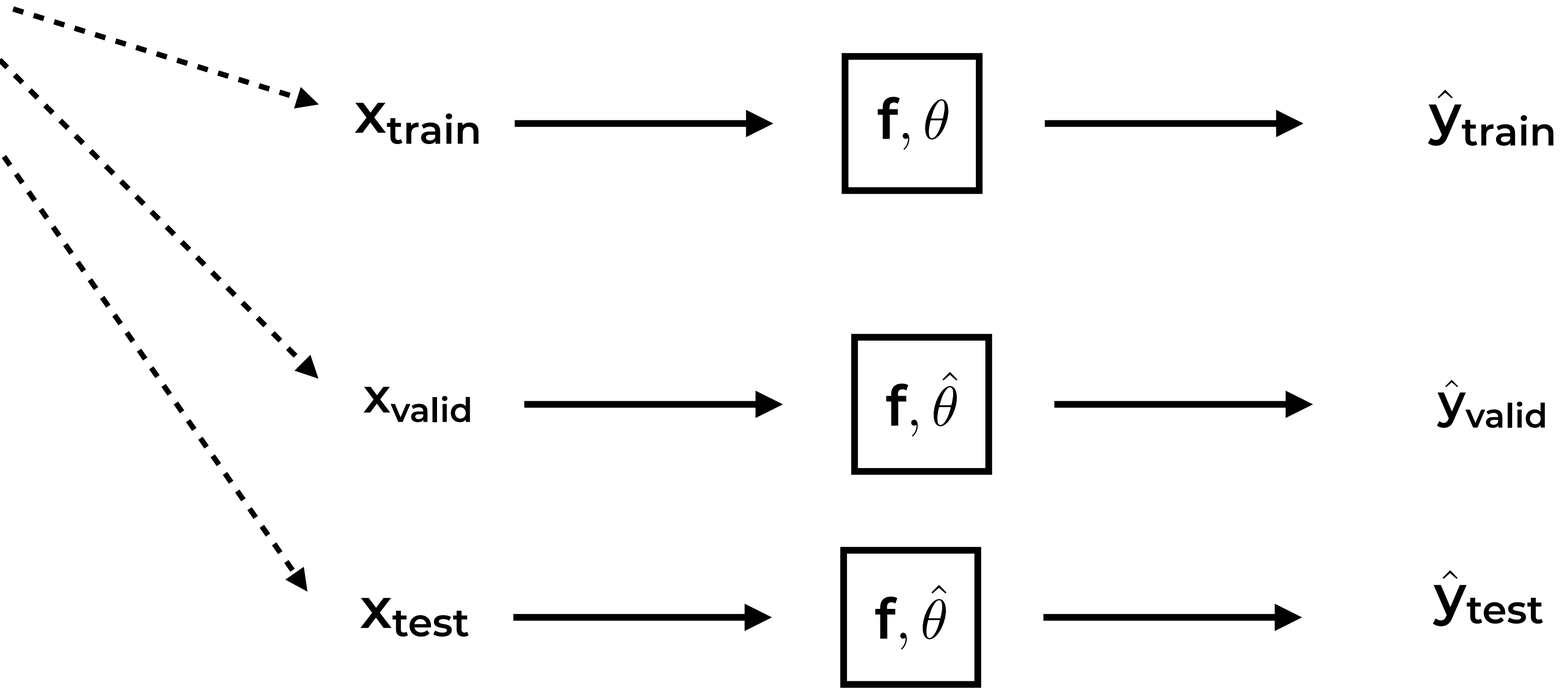
$L(\hat{y}_{test}, y_{test})$

Learning Process

In practice

Distribution over (x,y): P(x,y)





Loss

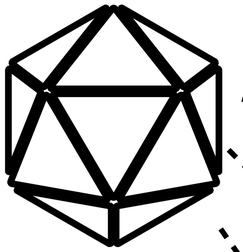
$L(\hat{y}_{train}, y_{train})$

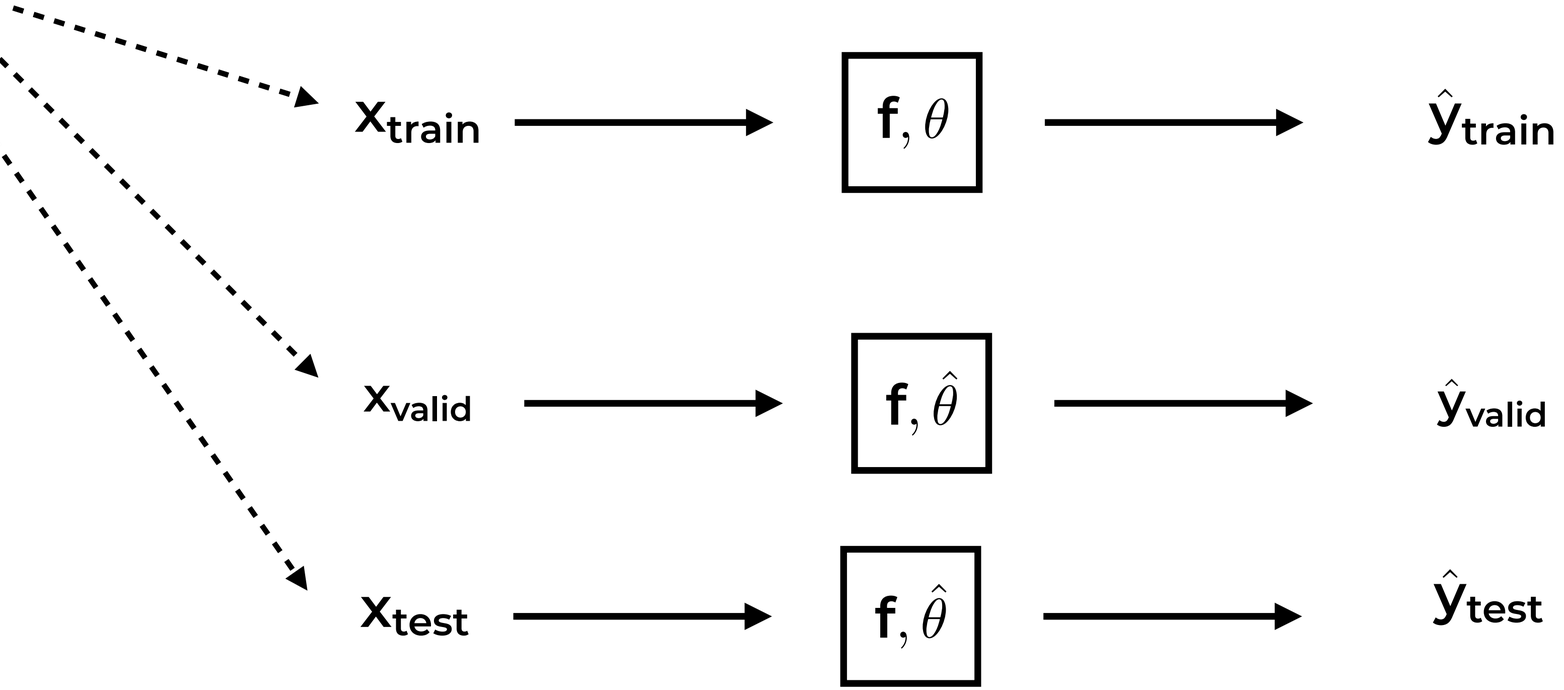
$L(\hat{y}_{test}, y_{test})$

Learning Process

In practice

Distribution over (x,y): P(x,y)





Loss

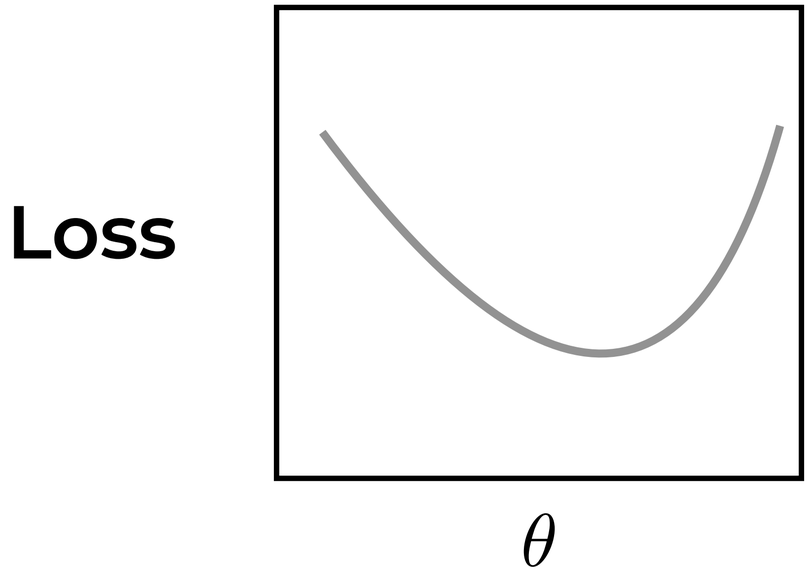
$L(\hat{y}_{train}, y_{train})$

Useful:
- **to select hyper-parameters**
- **To pick the best model**

$L(\hat{y}_{test}, y_{test})$

Learning

- **Learn: Change the parameters to obtain better predictions**



- **In other words: change the parameters to minimize the loss**
  - **Take the derivative of the loss wrt the parameter:** $\frac{\mathrm{d}\ \mathrm{Loss}}{\mathrm{d}\theta}$

Different models

- f: linear regression, θ has a closed-form solution
- f: neural network, θ does not have a closed-forum solution. Gradient descent is used

- Given a training set: $\{(x_{train}, y_{train})\}$

- Initialize $\hat{\theta}_1$ randomly

for t $= 1, 2, \ldots$ (epochs) **do**
  for i $= 1, 2, \ldots$ (datum) **do**

    - Obtain the predictions $\{f(x_{train}; \hat{\theta}_t)\}$ (Forward propagation)

    - Compute the Loss: $Loss_{ti} := L(f(x_i; \hat{\theta}_t), y_i)$

    - Find the derivative of the loss: $\frac{d\ Loss_{ti}}{d\ \hat{\theta}_t}$
    - Update parameters: $\hat{\theta}_{t+1} = \hat{\theta}_t - \alpha \frac{d\ Loss_{ti}}{d\ \hat{\theta}_t}$
    - If $||\hat{\theta}_{t+1} - \hat{\theta}_t||_2^2 < \epsilon$ then stop
  end for
end for

- Given a training set: $\{(x_{train}, y_{train})\}$

- Initialize $\hat{\theta}_1$ randomly

Stochastic Gradient Descent

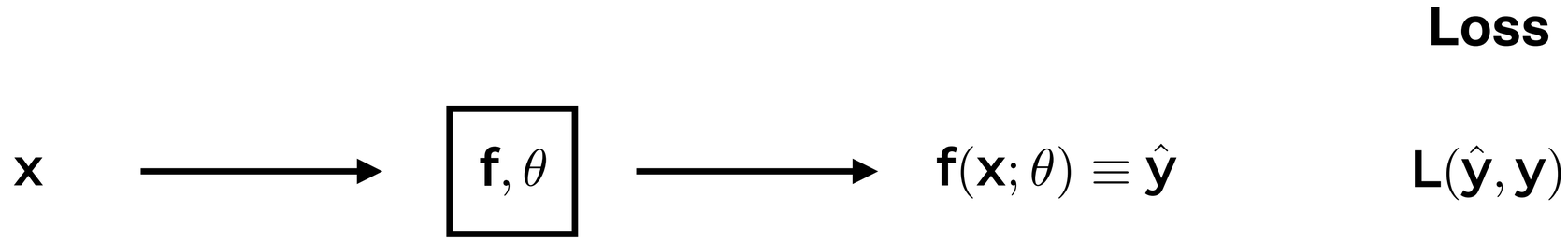
for t $= 1, 2, \ldots$ (epochs) **do**
  for i $= 1, 2, \ldots$ (datum) **do**

    - Obtain the predictions $\{f(x_{train}; \hat{\theta}_t)\}$ (Forward propagation)

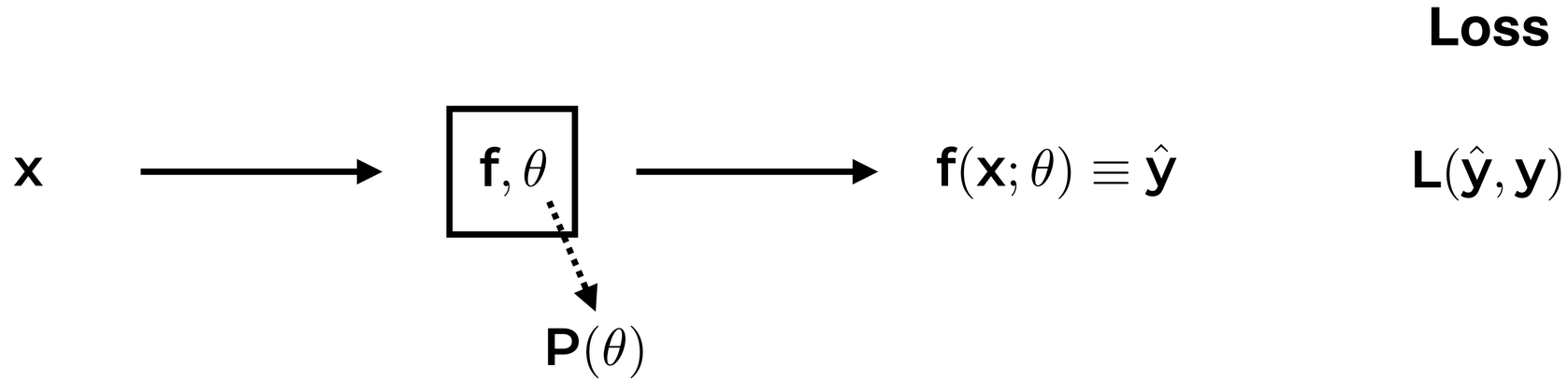
    - Compute the Loss: $Loss_{ti} := L(f(x_i; \hat{\theta}_t), y_i)$

    - Find the derivative of the loss: $\frac{d\ Loss_{ti}}{d\ \hat{\theta}_t}$
    - Update parameters: $\hat{\theta}_{t+1} = \hat{\theta}_t - \alpha \frac{d\ Loss_{ti}}{d\ \hat{\theta}_t}$
    - If $||\hat{\theta}_{t+1} - \hat{\theta}_t||_2^2 < \epsilon$ then stop
  end for
end for

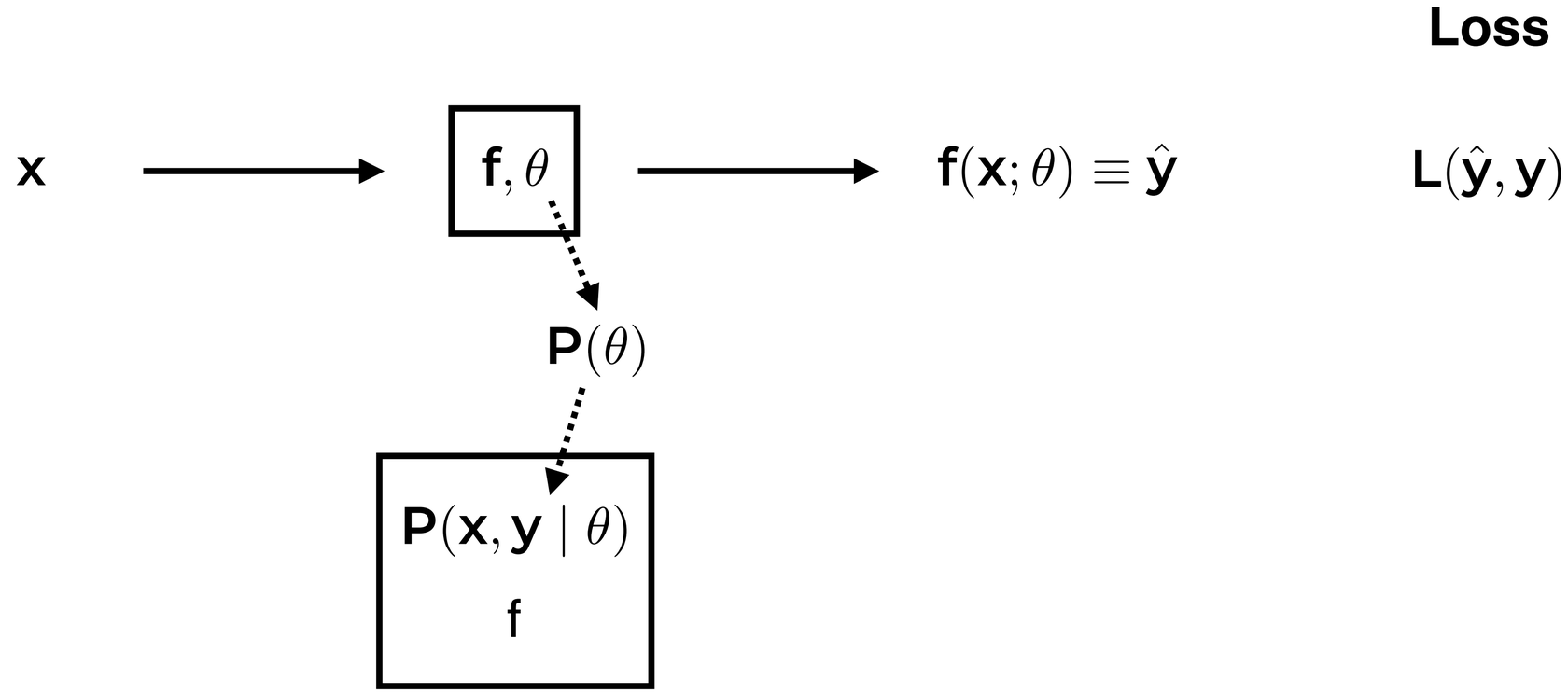
Probabilistic Models



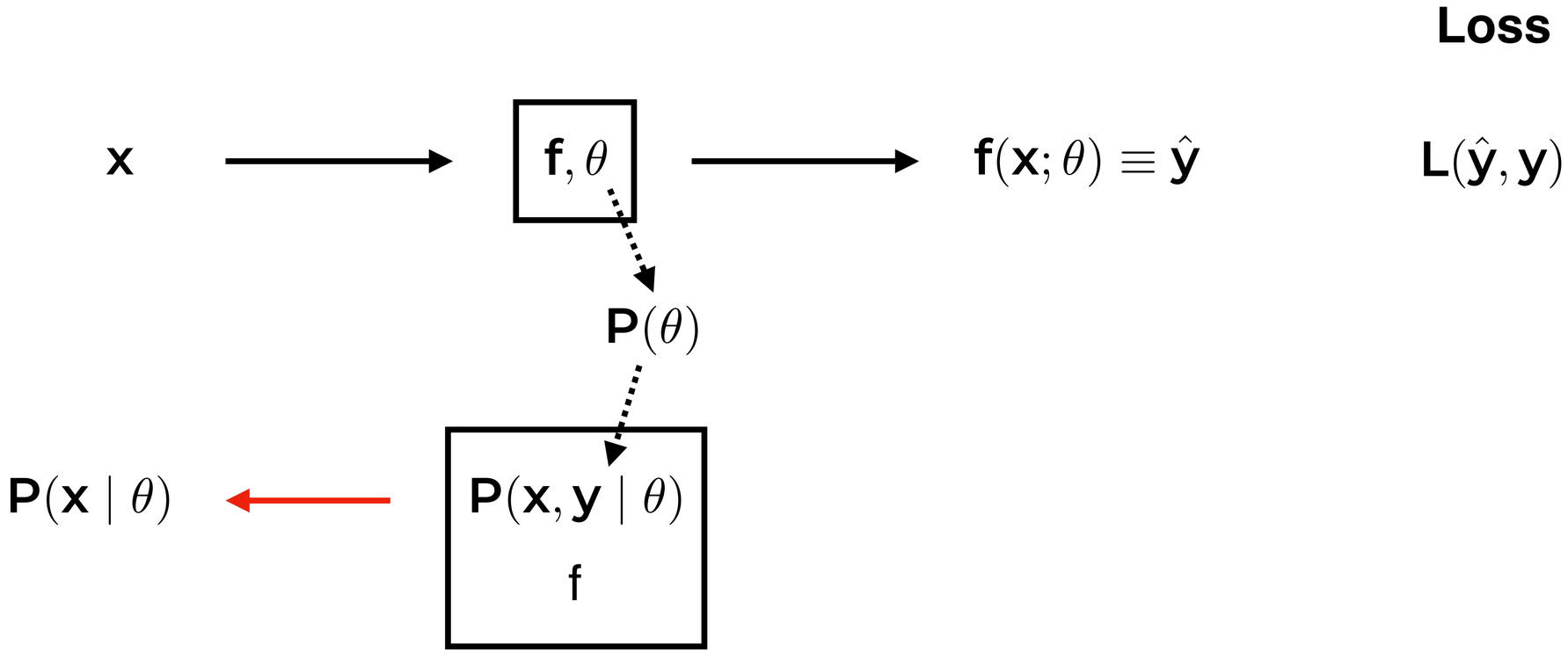
Probabilistic Models



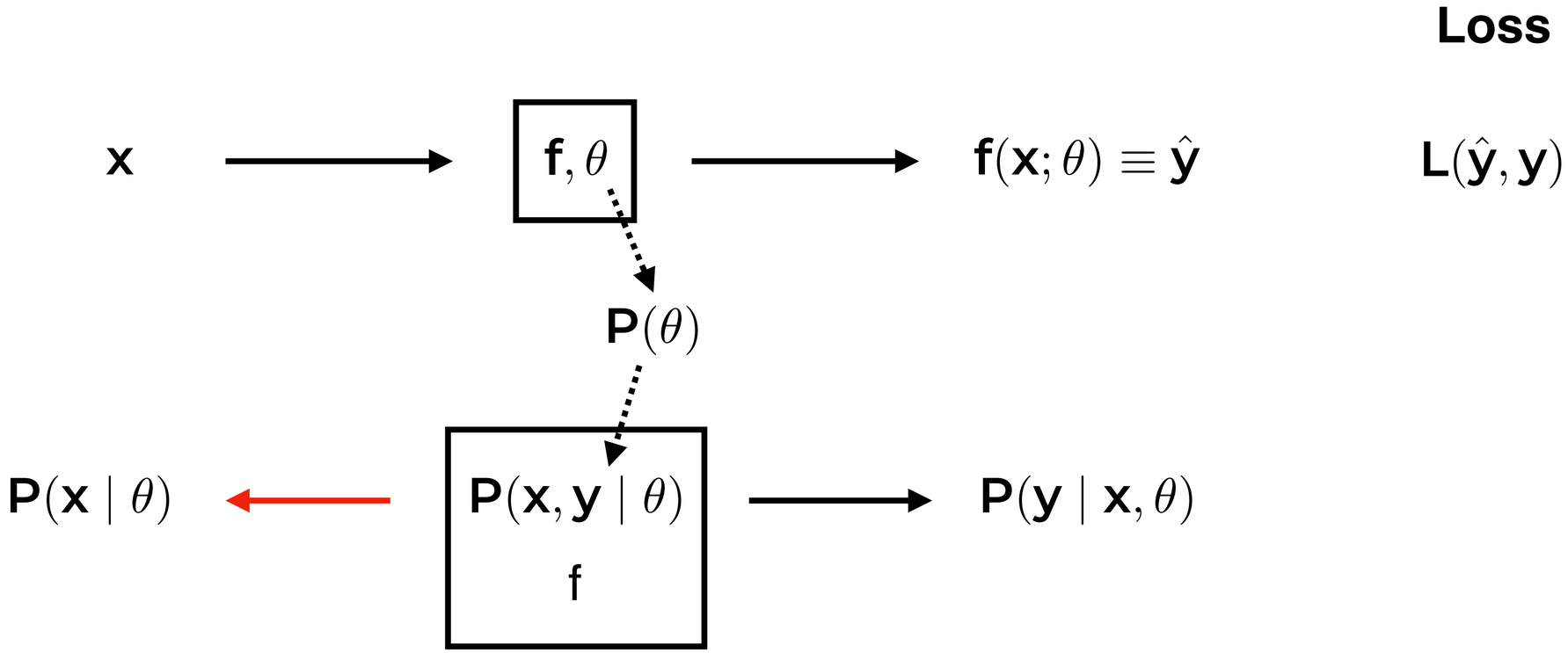
Probabilistic Models



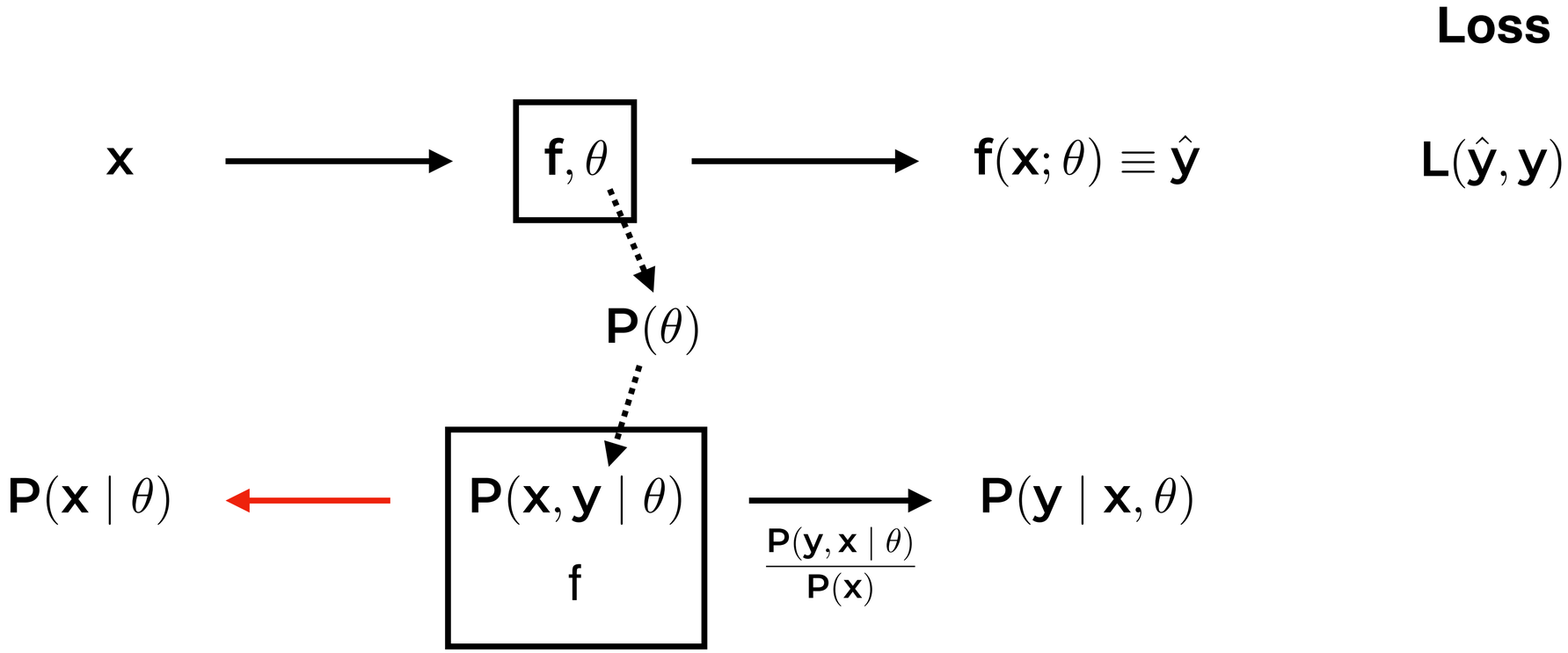
Probabilistic Models



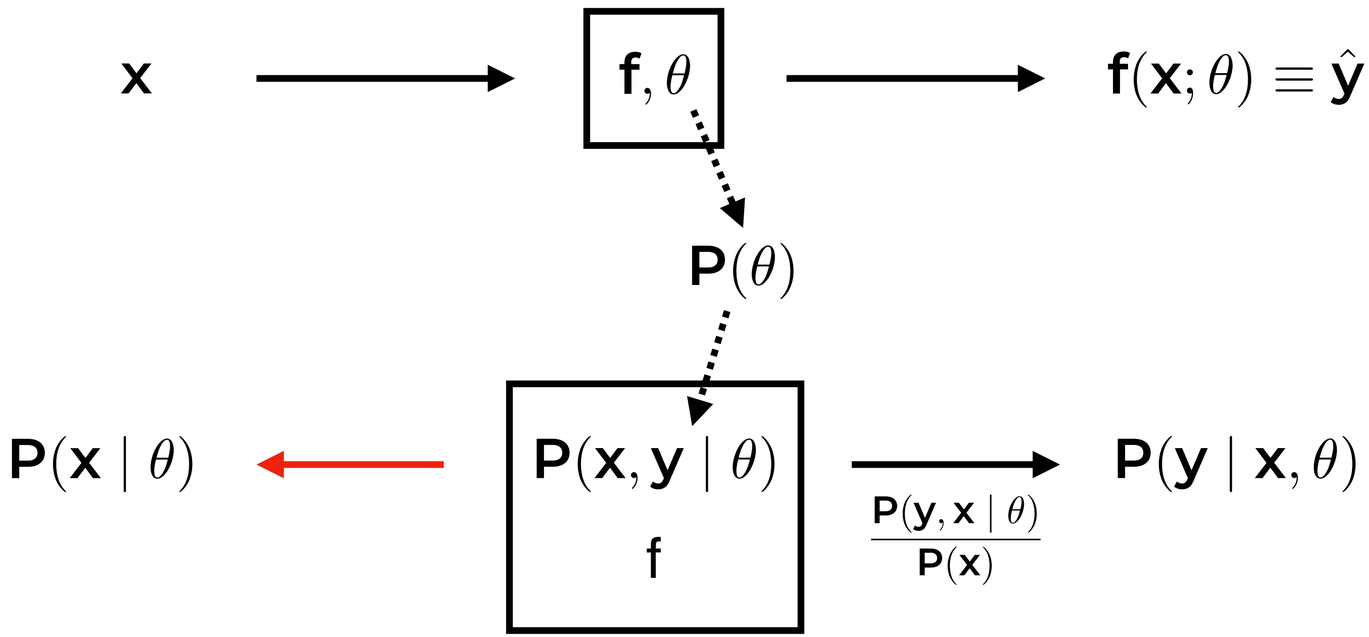
Probabilistic Models



Probabilistic Models



Probabilistic Models



Loss

$L(\hat{y}, y)$

Likelihood

$P(\text{data} \mid \text{params})$

$P(x, y \mid \theta)$

Example

Data: 952 1064 965 1037 871 1029 1138 (unsupervised problem)

Example

Data: 952 1064 965 1037 871 1029 1138 (unsupervised problem)

Model: $\mathbf{P}(\mathbf{x} \mid \theta) := \mathcal{N}(\boldsymbol{\mu}, \mathbf{1})$

Example

Data: 952 1064 965 1037 871 1029 1138 (unsupervised problem)

Model: $\mathbf{P}(\mathbf{x} \mid \theta) := \mathcal{N}(\boldsymbol{\mu}, \mathbf{1})$

Likelihood for a single datum:

Example

Data: 952 1064 965 1037 871 1029 1138 (unsupervised problem)

Model: $\mathbf{P}(\mathbf{x} \mid \theta) := \mathcal{N}(\boldsymbol{\mu}, \mathbf{1})$

Likelihood for a single datum:

$$\mathbf{Likelihood}(\mathbf{x} \mid \boldsymbol{\mu}, \mathbf{1}) = \frac{\mathbf{1}}{\sqrt{\mathbf{2\pi}}} \exp -\frac{(\mathbf{x} - \boldsymbol{\mu})^2}{\mathbf{2}}$$

Example

Data: 952 1064 965 1037 871 1029 1138 (unsupervised problem)

Model: $P(x \mid \theta) := \mathcal{N}(\mu, 1)$

Likelihood for a single datum:

$$\text{Likelihood}(x \mid \mu, 1) = \frac{1}{\sqrt{2\pi}} \exp -\frac{(x-\mu)^2}{2}$$

Log-Likelihood

$$= \log \frac{1}{\sqrt{2\pi}} \exp -\frac{(x-\mu)^2}{2}$$

$$= \log 1 - \frac{1}{2}\log 2\pi - \frac{(x-\mu)^2}{2}$$

Example

Data: 952 1064 965 1037 871 1029 1138 (unsupervised problem)

Model: $\mathbf{P}(\mathbf{x} \mid \theta) := \mathcal{N}(\boldsymbol{\mu}, \mathbf{1})$

Likelihood for a single datum:

$$\textbf{Likelihood}(\mathbf{x} \mid \boldsymbol{\mu}, \mathbf{1}) = \frac{\mathbf{1}}{\sqrt{\mathbf{2\pi}}} \exp -\frac{(\mathbf{x} - \boldsymbol{\mu})^2}{\mathbf{2}}$$

Log-Likelihood

$$= \log \frac{\mathbf{1}}{\sqrt{\mathbf{2\pi}}} \exp -\frac{(\mathbf{x} - \boldsymbol{\mu})^2}{\mathbf{2}}$$

$$= \mathbf{log1} - \frac{\mathbf{1}}{\mathbf{2}}\mathbf{log2\pi} - \frac{(\mathbf{x} - \boldsymbol{\mu})^2}{\mathbf{2}}$$

What value of $\boldsymbol{\mu}$ maximizes it?

Example

Data: 952 1064 965 1037 871 1029 1138 (unsupervised problem)

Model: $P(x \mid \theta) := \mathcal{N}(\mu, 1)$

Likelihood for a single datum:

$$\text{Likelihood}(x \mid \mu, 1) = \frac{1}{\sqrt{2\pi}} \exp -\frac{(x-\mu)^2}{2}$$

Log-Likelihood

$$= \log \frac{1}{\sqrt{2\pi}} \exp -\frac{(x-\mu)^2}{2}$$

$$= \log 1 - \frac{1}{2}\log 2\pi - \frac{(x-\mu)^2}{2}$$

What value of μ maximizes it?

$$\frac{d\ \text{Log-Likelihood}}{d\ \mu}$$

$$= \frac{d\ \frac{(x-\mu)^2}{2}}{d\ \mu}$$

$$= (x - \mu)$$

set to 0

$$\mu = x$$

Example

Data: 952 1064 965 1037 871 1029 1138 (unsupervised problem)

Model: $P(x \mid \theta) := \mathcal{N}(\mu, 1)$

Likelihood for a single datum:

$$\text{Likelihood}(x \mid \mu, 1) = \frac{1}{\sqrt{2\pi}} \exp -\frac{(x-\mu)^2}{2}$$

Log-Likelihood

$$= \log \frac{1}{\sqrt{2\pi}} \exp -\frac{(x-\mu)^2}{2}$$

$$= \log 1 - \frac{1}{2}\log 2\pi - \frac{(x-\mu)^2}{2}$$

What value of μ maximizes it?

$$\frac{d\ \text{Log-Likelihood}}{d\ \mu}$$

$$= \frac{d\ \frac{(x-\mu)^2}{2}}{d\ \mu}$$

$$= (x - \mu)$$

set to 0

$\mu = x = 952$

Data: (x,y)
Naive Bayes

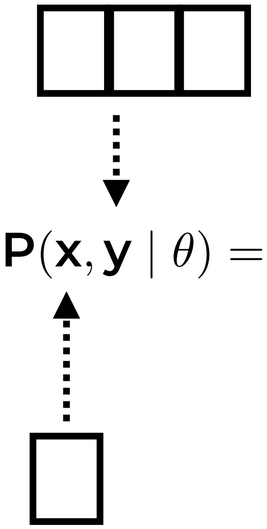
Data: (x,y)
Naive Bayes

$$\text{Model} : \mathbf{P}(\mathbf{x}, \mathbf{y} \mid \theta) = \mathbf{P}(\mathbf{x} \mid \mathbf{y}, \theta)\mathbf{P}(\mathbf{y} \mid \theta)$$

Data: (x,y)
Naive Bayes

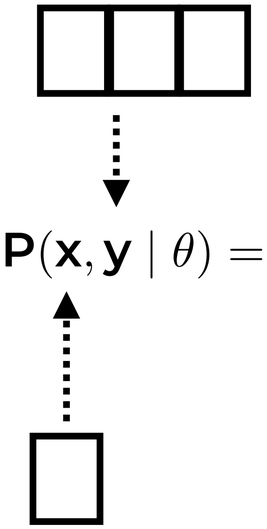
$$\text{Model} : \mathbf{P}(\mathbf{x}, \mathbf{y} \mid \theta) = \mathbf{P}(\mathbf{x} \mid \mathbf{y}, \theta)\mathbf{P}(\mathbf{y} \mid \theta)$$

Data: (x,y)
Naive Bayes



$$\text{Model} : \mathbf{P}(\mathbf{x}, \mathbf{y} \mid \theta) = \mathbf{P}(\mathbf{x} \mid \mathbf{y}, \theta)\mathbf{P}(\mathbf{y} \mid \theta)$$

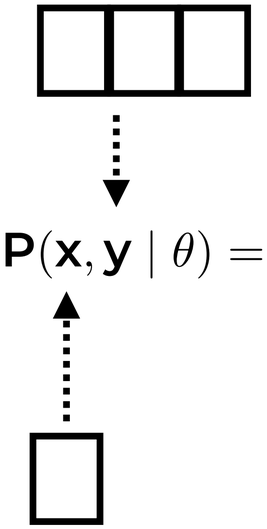
Data: (x,y)
Naive Bayes



$$\text{Model} : \mathbf{P}(\mathbf{x}, \mathbf{y} \mid \theta) = \mathbf{P}(\mathbf{x} \mid \mathbf{y}, \theta)\mathbf{P}(\mathbf{y} \mid \theta)$$

Data: x
Gaussian Mixture Models

Data: (x,y)
Naive Bayes

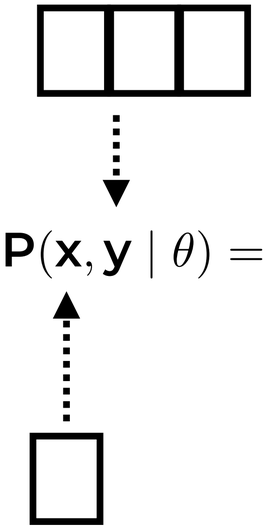


$$\text{Model}: \mathbf{P}(\mathbf{x}, \mathbf{y} \mid \theta) = \mathbf{P}(\mathbf{x} \mid \mathbf{y}, \theta)\mathbf{P}(\mathbf{y} \mid \theta)$$

Data: x
Gaussian Mixture Models

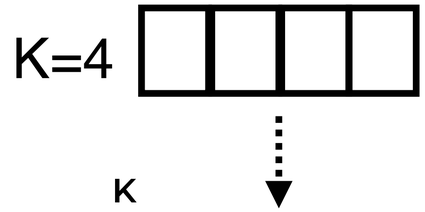
$$\text{Model}: \mathbf{P}(\mathbf{x} \mid \theta) = \sum_{\mathbf{k}=1}^{\mathbf{K}} \mathbf{P}(\theta_{\mathbf{x}} = \mathbf{k}) \underbrace{\mathbf{P}(\mathbf{x} \mid \theta_{\mathbf{k}})}_{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{\mathbf{k}}, \Sigma_{\mathbf{k}})} \quad \text{(K components)}$$

Data: (x,y)
Naive Bayes



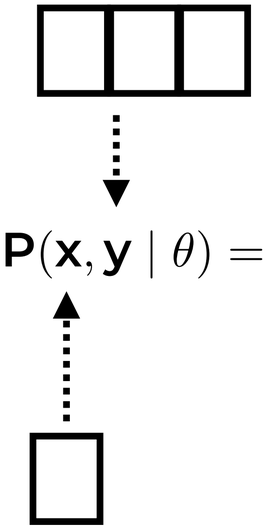
$$\text{Model}: \mathbf{P}(\mathbf{x}, \mathbf{y} \mid \theta) = \mathbf{P}(\mathbf{x} \mid \mathbf{y}, \theta)\mathbf{P}(\mathbf{y} \mid \theta)$$

Data: x
Gaussian Mixture Models



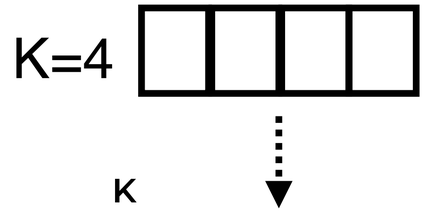
$$\text{Model}: \mathbf{P}(\mathbf{x} \mid \theta) = \sum_{\mathbf{k}=1}^{\mathbf{K}} \mathbf{P}(\theta_{\mathbf{x}} = \mathbf{k}) \underbrace{\mathbf{P}(\mathbf{x} \mid \theta_{\mathbf{k}})}_{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{\mathbf{k}}, \Sigma_{\mathbf{k}})} \quad \text{(K components)}$$

Data: (x,y)
Naive Bayes



$$\text{Model}: \mathbf{P}(\mathbf{x}, \mathbf{y} \mid \theta) = \mathbf{P}(\mathbf{x} \mid \mathbf{y}, \theta)\mathbf{P}(\mathbf{y} \mid \theta)$$

Data: x
Gaussian Mixture Models



$$\text{Model}: \mathbf{P}(\mathbf{x} \mid \theta) = \sum_{\mathbf{k}=1}^{\mathbf{K}} \mathbf{P}(\theta_{\mathbf{x}} = \mathbf{k}) \underbrace{\mathbf{P}(\mathbf{x} \mid \theta_{\mathbf{k}})}_{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{\mathbf{k}}, \Sigma_{\mathbf{k}})} \quad \textbf{(K components)}$$

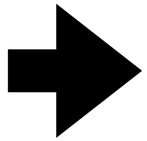
$$\textbf{Max. likelihood (MLE)} \;:\; \hat{\theta}_{\mathbf{MLE}} = \arg\max_{\theta} \mathbf{P}(\mathbf{x} \mid \theta)$$

$$\textbf{Max. a posteriori (MAP)} \;:\; \hat{\theta}_{\mathbf{MAP}} = \arg\max_{\theta} \mathbf{P}(\theta \mid \mathbf{x}, \gamma) = \arg\max_{\theta} \frac{\mathbf{P}(\mathbf{x} \mid \theta)\mathbf{P}(\theta \mid \gamma)}{\mathbf{P}(\mathbf{x})}$$

MLE VS. MAP

$$\theta_{MLE} = \arg\max_{\theta} P(X|\theta)$$

$$= \arg\max_{\theta} \prod_{i} P(x_i|\theta)$$



$$\theta_{MLE} = \arg\max_{\theta} \log P(X|\theta)$$

$$= \arg\max_{\theta} \log \prod_{i} P(x_i|\theta)$$

$$= \arg\max_{\theta} \sum_{i} \log P(x_i|\theta)$$

https://agustinus.kristia.de/techblog/2017/01/01/mle-vs-map/

MLE VS. MAP

$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)}$$

$$\propto P(X|\theta)P(\theta)$$

Bayes's rule

$$\theta_{MAP} = \arg\max_{\theta} P(X|\theta)P(\theta)$$

$$= \arg\max_{\theta} \log P(X|\theta) + \log P(\theta)$$

$$= \arg\max_{\theta} \log \prod_{i} P(x_i|\theta) + \log P(\theta)$$

$$= \arg\max_{\theta} \sum_{i} \log P(x_i|\theta) + \log P(\theta)$$

https://agustinus.kristia.de/techblog/2017/01/01/mle-vs-map/

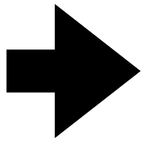
MLE VS. MAP

$$\theta_{MAP} = \arg\max_{\theta} P(X|\theta)P(\theta)$$

$$= \arg\max_{\theta} \log P(X|\theta) + \log P(\theta)$$

$$= \arg\max_{\theta} \log \prod_{i} P(x_i|\theta) + \log P(\theta)$$

$$= \arg\max_{\theta} \sum_{i} \log P(x_i|\theta) + \log P(\theta)$$



$$\theta_{MAP} = \arg\max_{\theta} \sum_{i} \log P(x_i|\theta) + \log P(\theta)$$

$$= \arg\max_{\theta} \sum_{i} \log P(x_i|\theta) + const$$

$$= \arg\max_{\theta} \sum_{i} \log P(x_i|\theta)$$

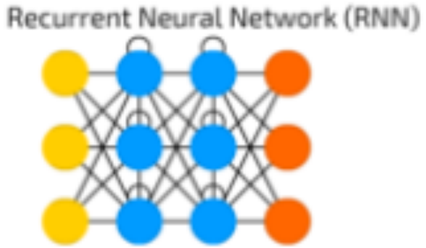
$$= \theta_{MLE}$$

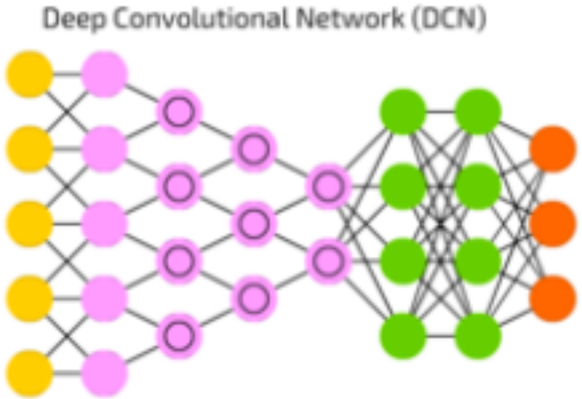
MLE is a special case of MAP, where the prior is uniform!

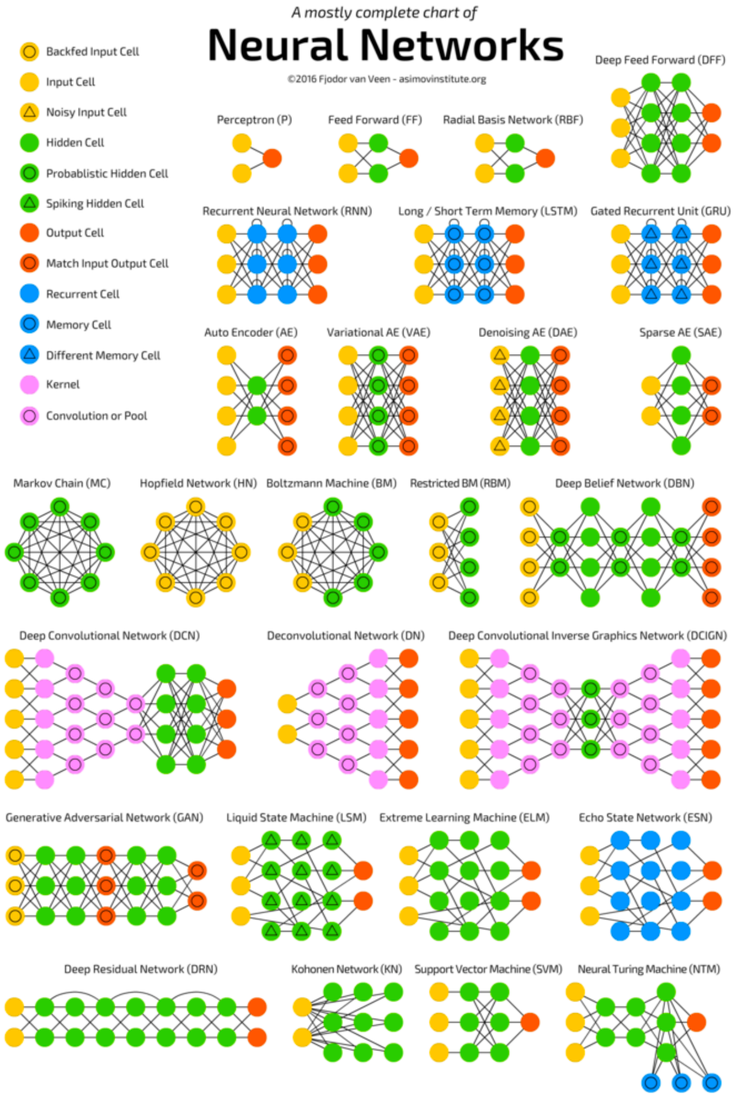
MLPs / RNNs / CNNs

- **MLPs:** layers are fully-connected to the next layer
- **RNNs:** inputs at each layer
  - **Typical application:** time-series modelling
- **CNNs:** replace matrix multiplications by convolutions (sparse connections, weight sharing) + pooling
  - **Typical application:** object recognition in images

Various Architectures







Leijnen, Stefan, and Fjodor van Veen. "The neural network zoo." *Multidisciplinary Digital Publishing Institute Proceedings*. Vol. 47. No. 1. 2020.