

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

Fall 2021

Week #11 - Summary

Announcement

- **Homework 1:** late submission policy (5 points penalty) **due November 7, 2021**
- **Homework 2:** due **November 14, 2021**
Upload answer to question 1 to Gradescope (not Zone Cours)
- Next week: no video capsules, and no quiz
- Final exam: **in-person**, at 9-12 on **December 16, 2021**

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 TIME

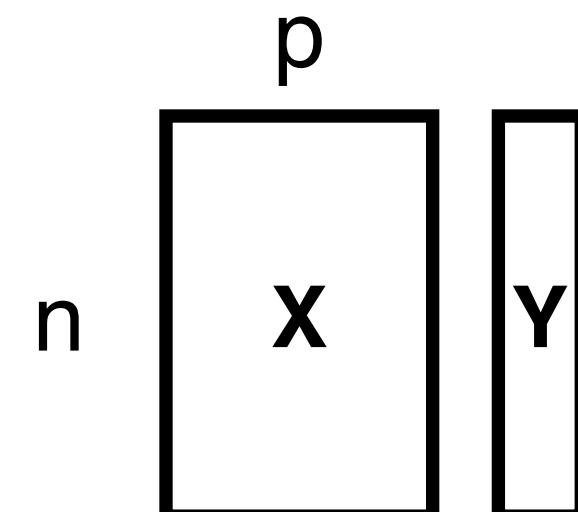
A colorful, three-dimensional sign with a glowing effect. The letters are outlined in black and filled with a gradient of colors: teal, yellow, red, orange, and green. Small yellow lights are visible along the edges of the letters.

Quiz 5

Login to your Gradescope account

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



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

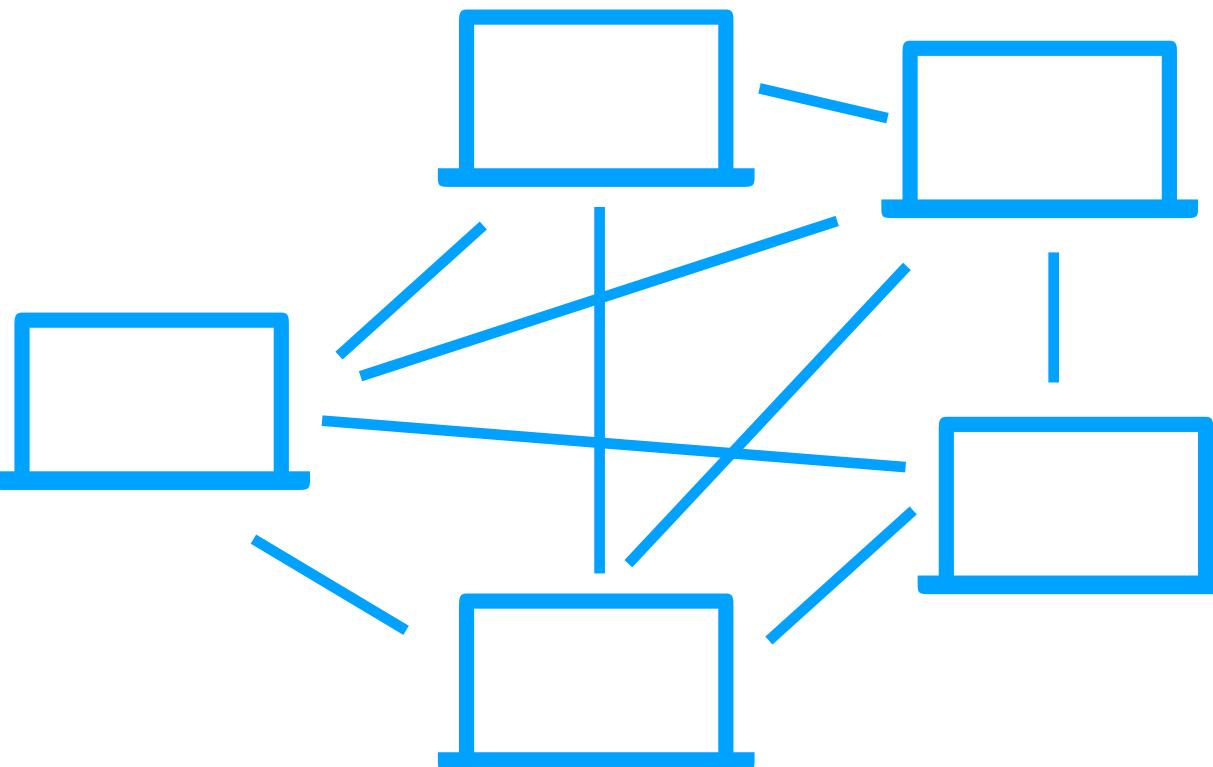
<https://www.top500.org/lists/top500/list/2020/06/>



Photo from Riken

2. Distributed computation

- ~200, 000 TFlops (Folding@home)



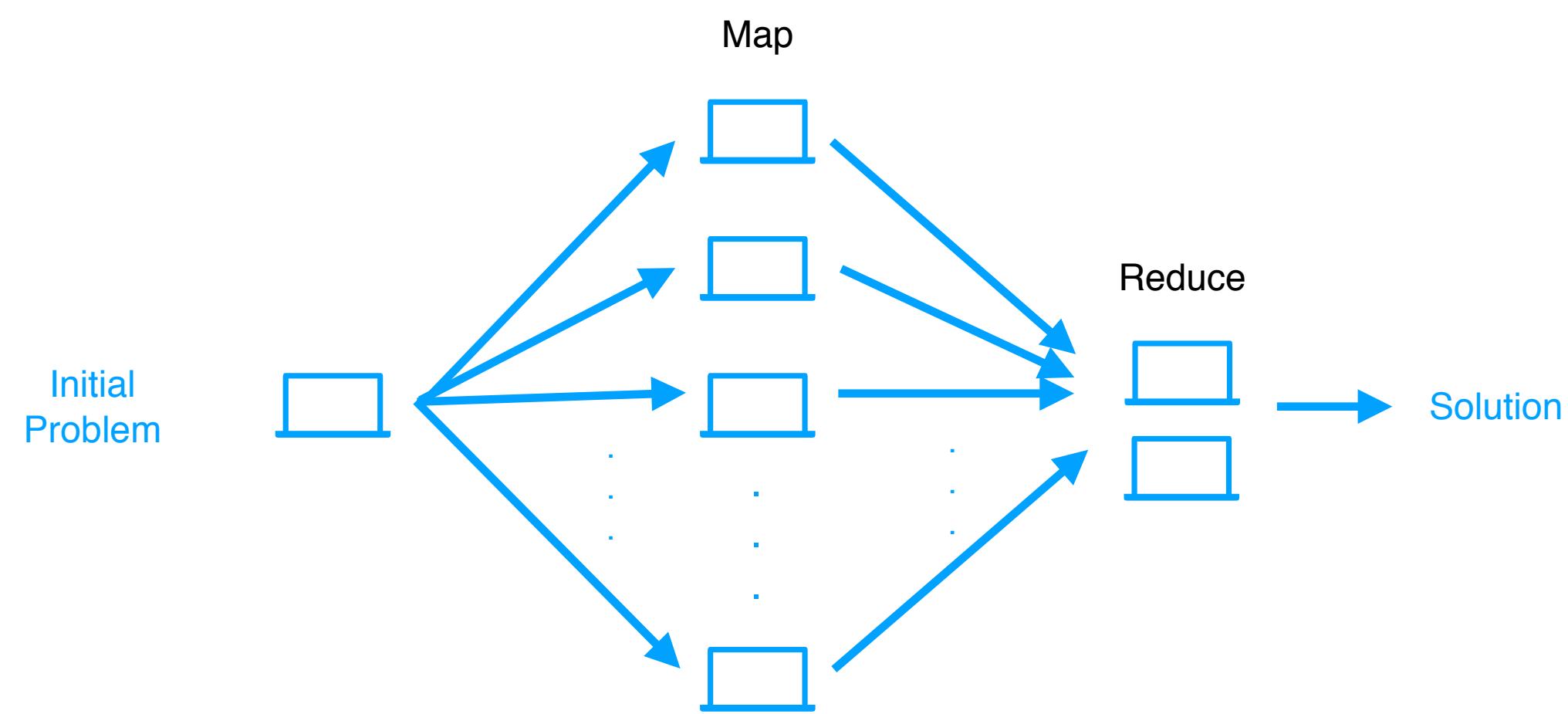
3. Specialized hardware

- Focuses on subset of operations
- Graphical Processing Unit (GPU), Field Programmable Gate Array (FPGA)
- ~10 TFlops

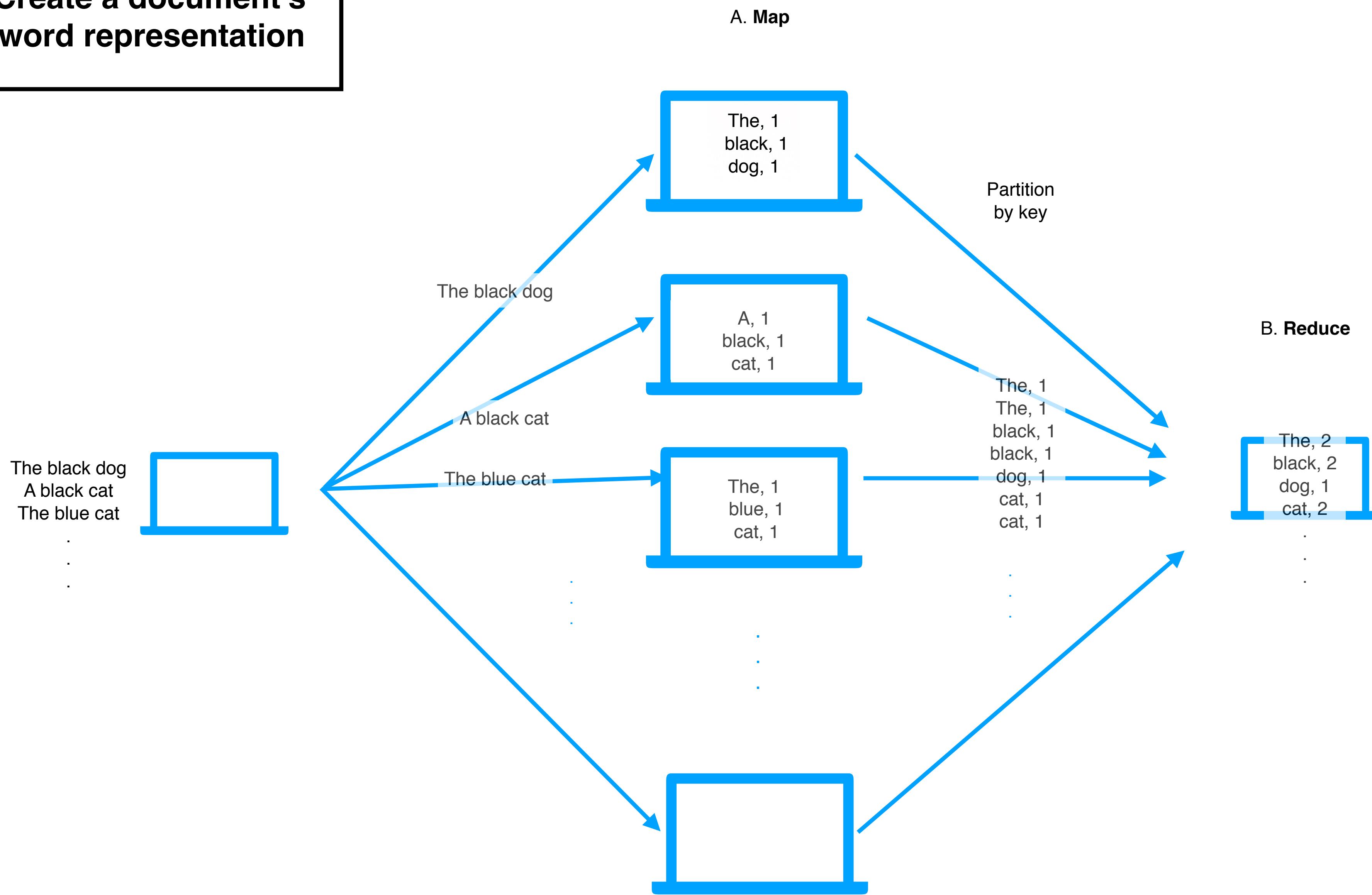


MapReduce (Apache) Hadoop

- **Two types of tasks:**
 - A. Map:** Solve a subproblem (filtering operation)
 - B. Reduce:** Combine the results of map workers (summary operation)



TASK: Create a document's bag-of-word representation



MapReduce is quite versatile

- A few examples of “map-reducible” 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

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

- Each term in the sums can be computed independently

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

A. Map

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

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

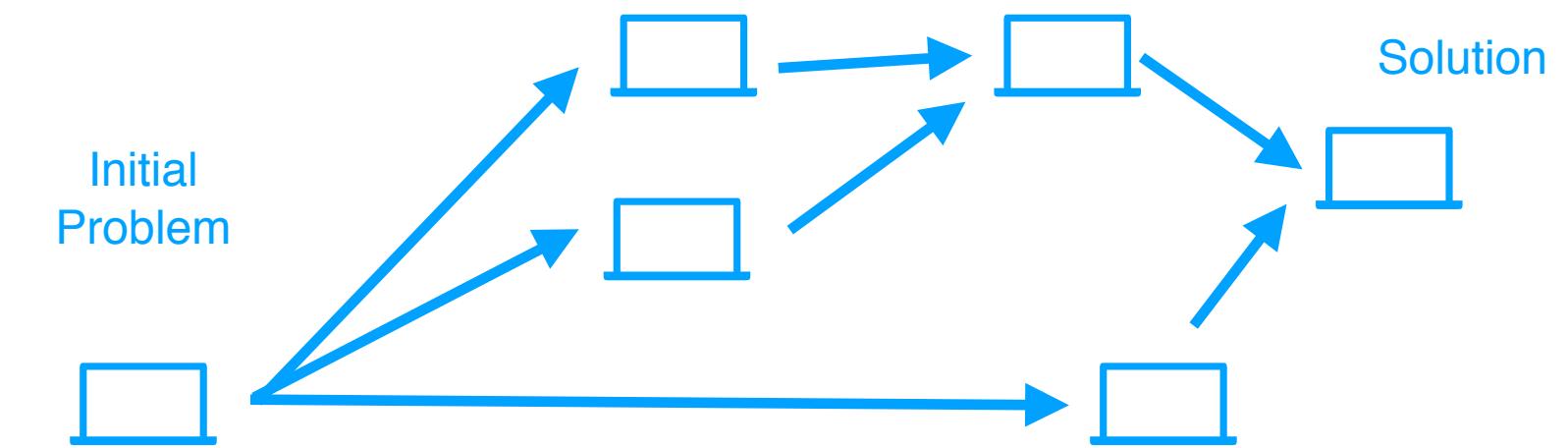
Shortcomings of MapReduce

- 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

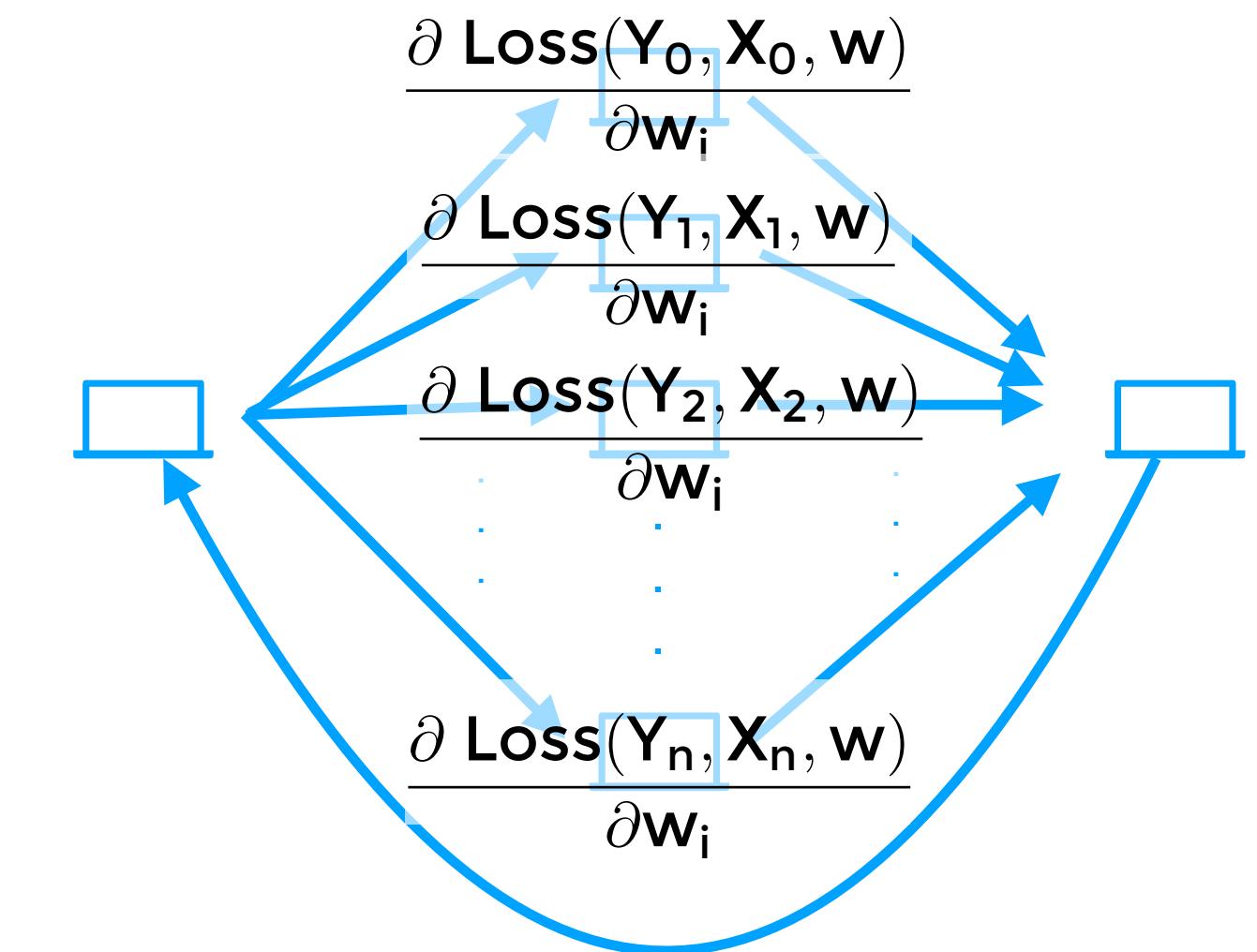
$$y = \frac{1}{1 + \exp(-w_0 - w_1x_1 - w_2x_2 - \dots - w_px_p)}$$

- 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

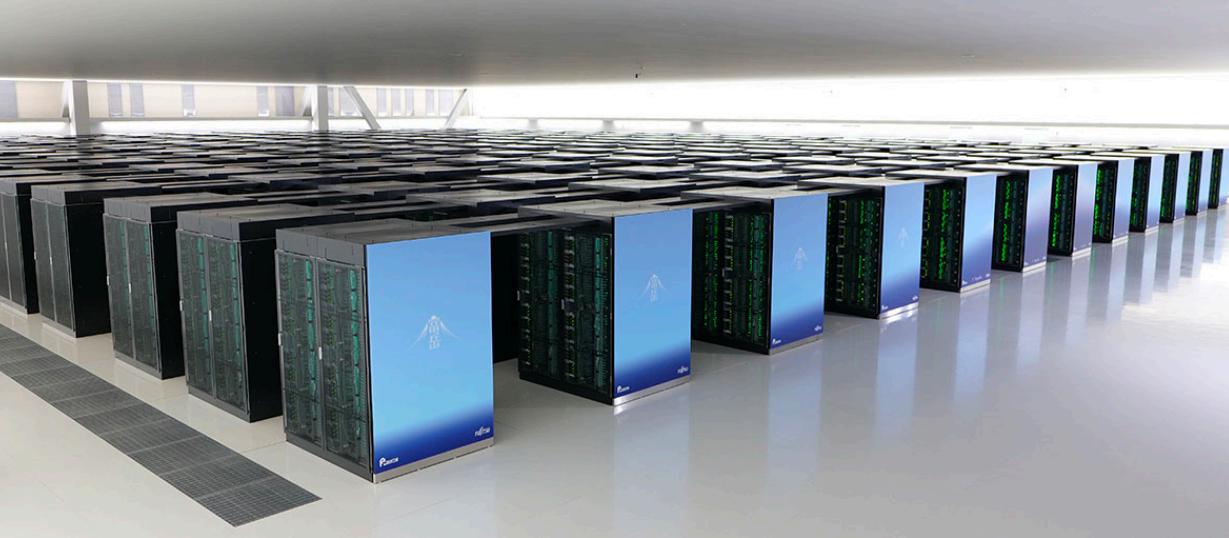
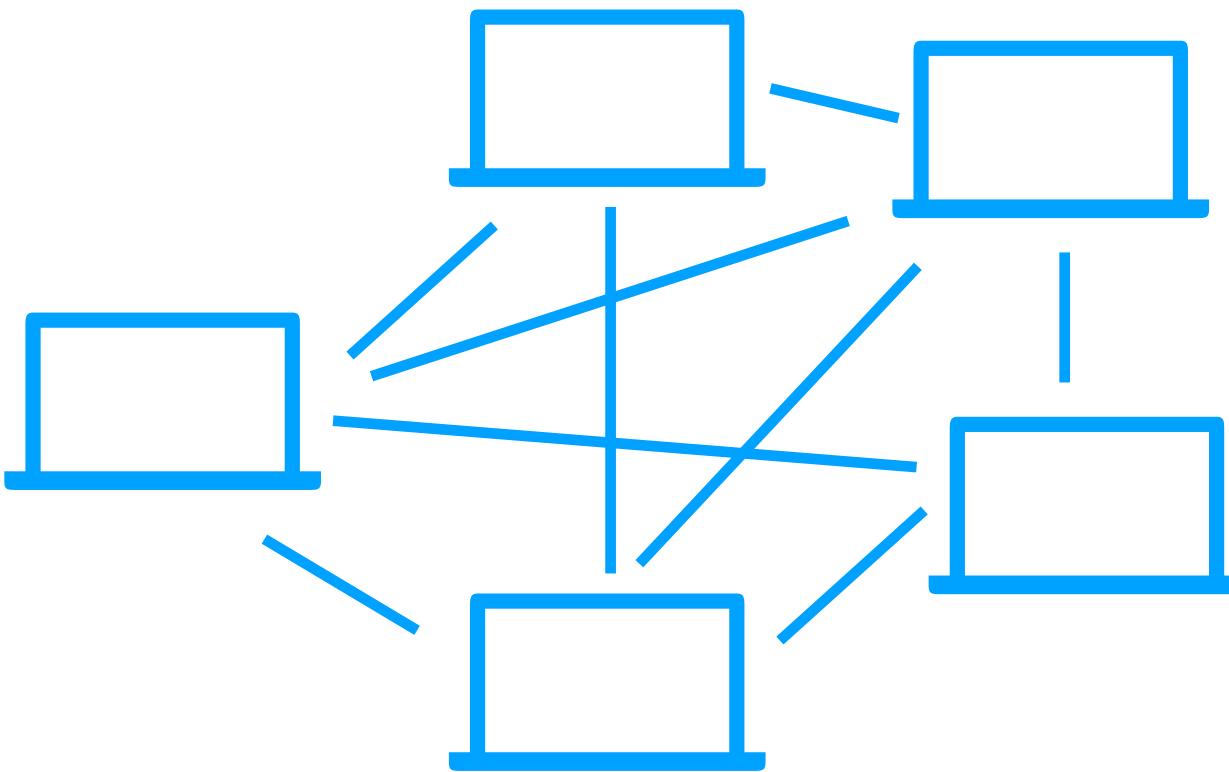


Photo from Riken

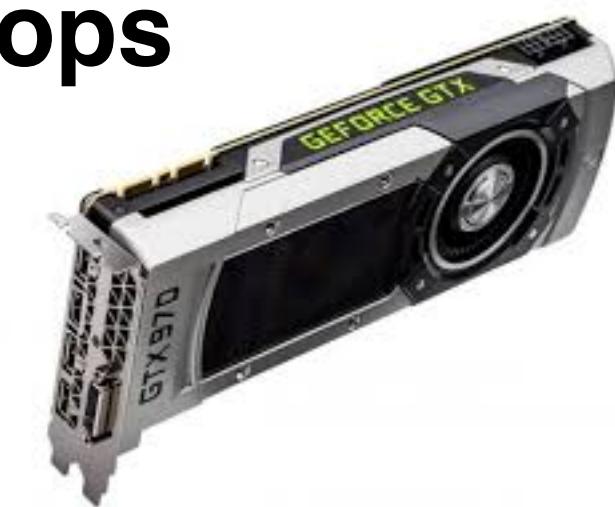
2. Distributed computation

- ~200, 000 TFlops
(Folding@home)



3. Specialized hardware

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



Equipment

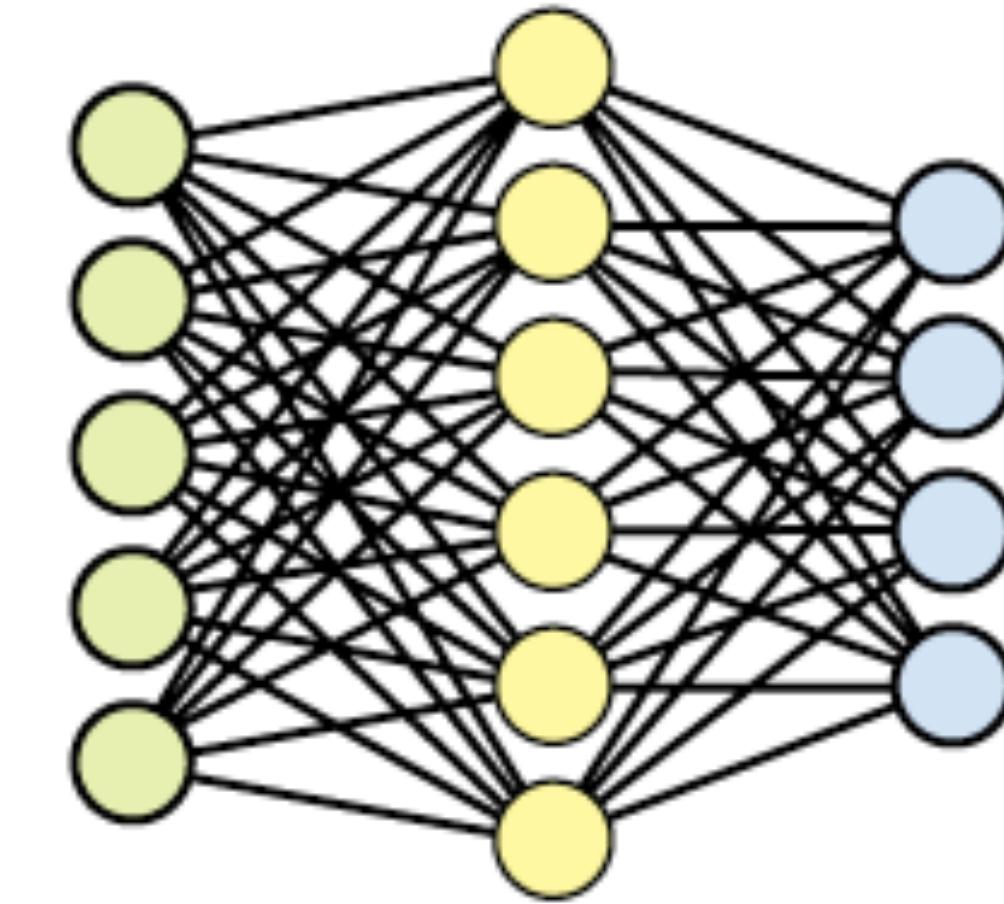


<http://www.personal.psu.edu/users/dl/dlm99/cpu.html>

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

Neural Networks

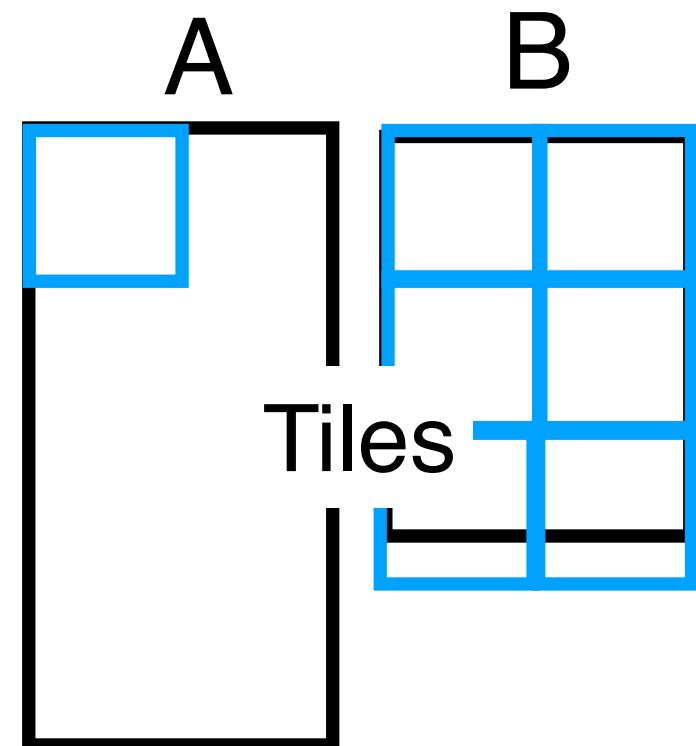
- Linear algebra:
- Multiplication *vector X matrix*
 - Calculate the activation of a neuron
 - Calculate the activation of a neuron for several data
- Matrix X matrix multiplication
 - Calculate the activation of several neurons for several data
- The dimensions of these objects depend on the size of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



Parallelize a Neural Networks

$$\begin{matrix} A \\ \boxed{} \end{matrix} \quad \begin{matrix} B \\ \boxed{} \end{matrix} \quad = \quad \begin{matrix} C \\ \boxed{} \end{matrix}$$

$$C = AB$$



- 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

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/>

Even more Specialized equipment

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

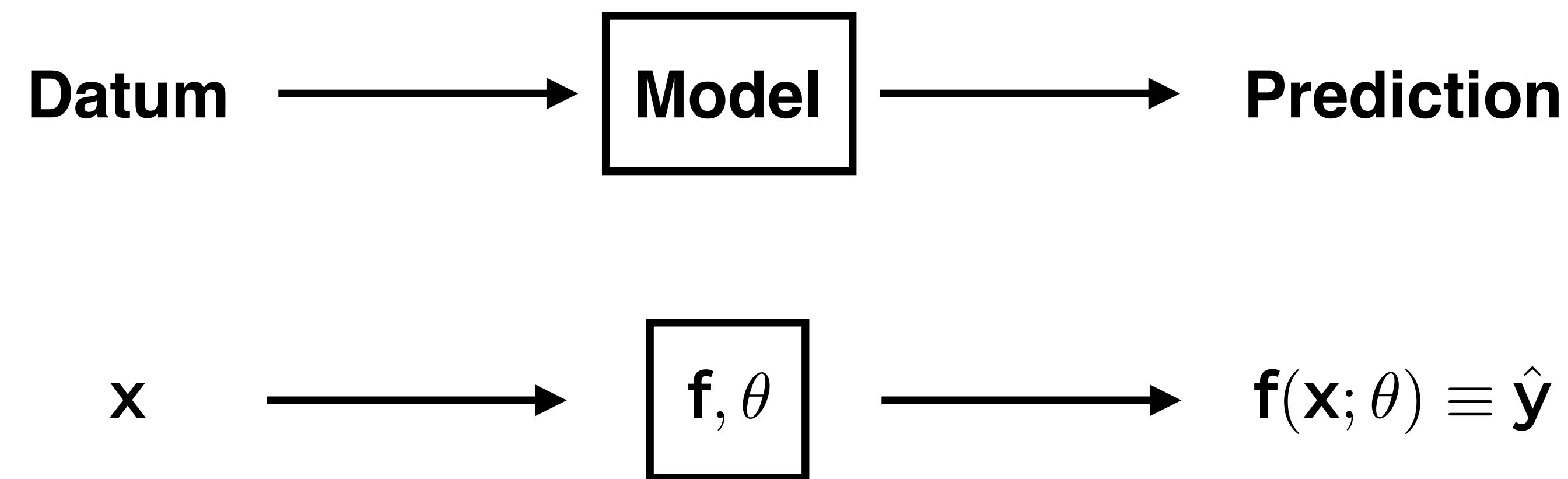
Example

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

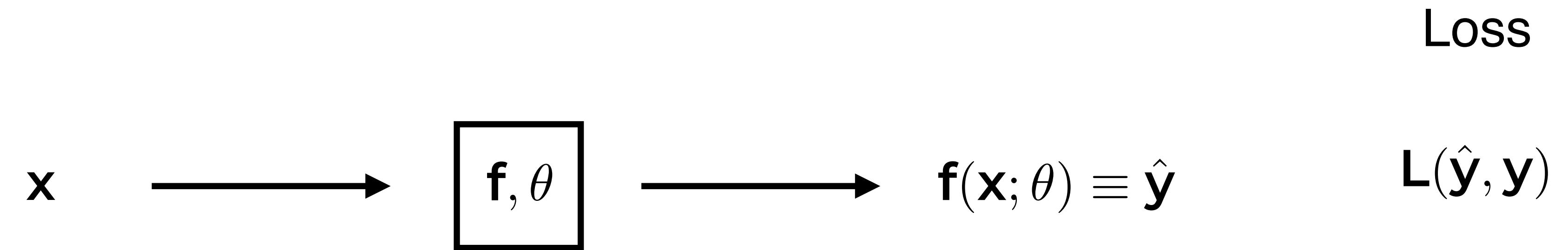
- It takes few changes to our code to use a GPU
- Beware of CPU-GPU transfers

- **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



Different losses for different types of y 's

$y \in \mathcal{R}$

y categorical e.g., {cat, dog, bird}

$y \in \{0, 1\}$

Regression

Classification

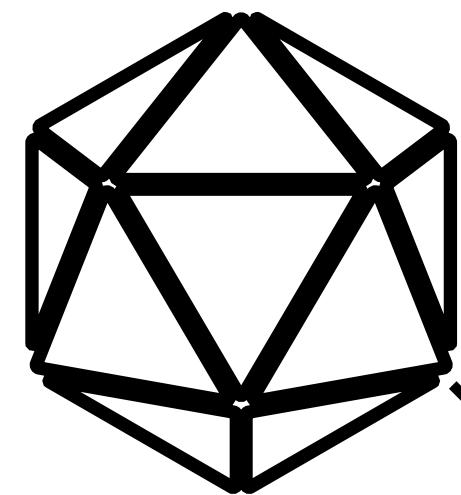
Binary Classification

$(\hat{y} - y)^2$

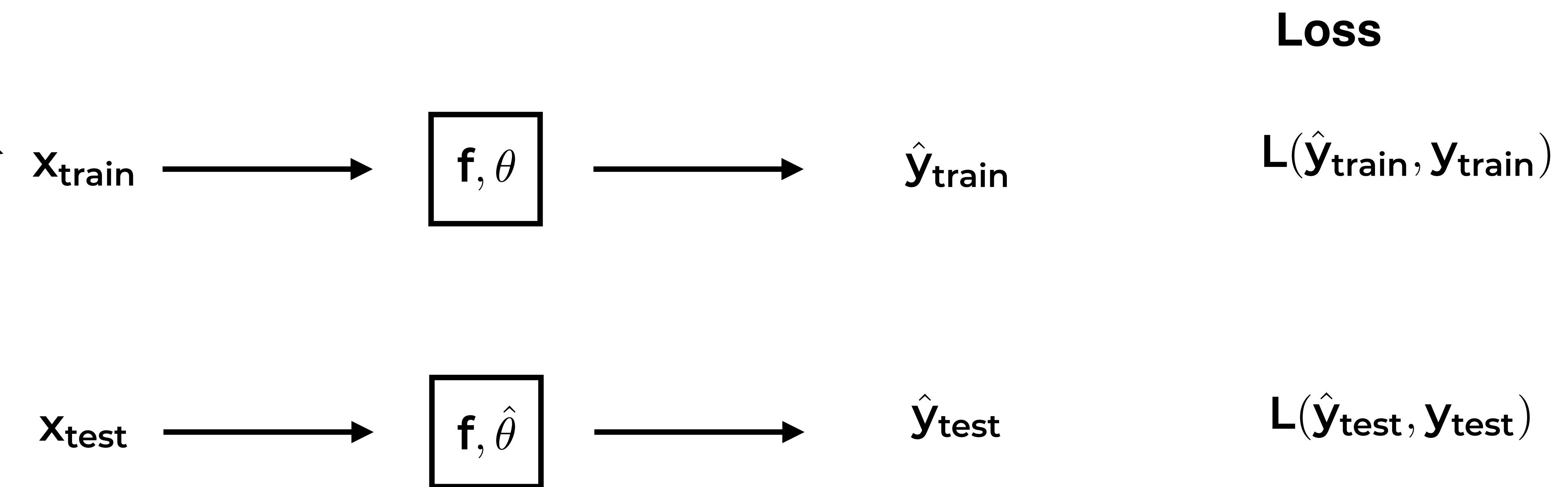
accuracy

AUC

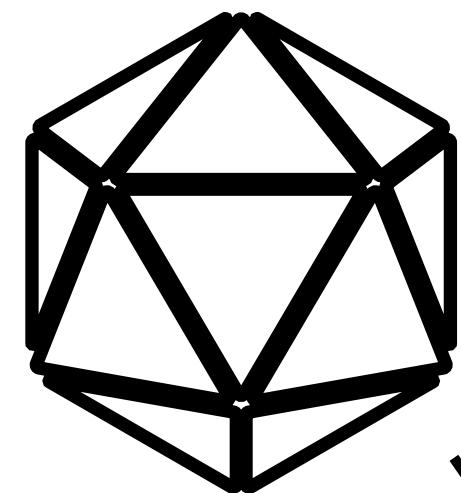
Distribution
over (x,y) :
 $P(x,y)$



Learning Process

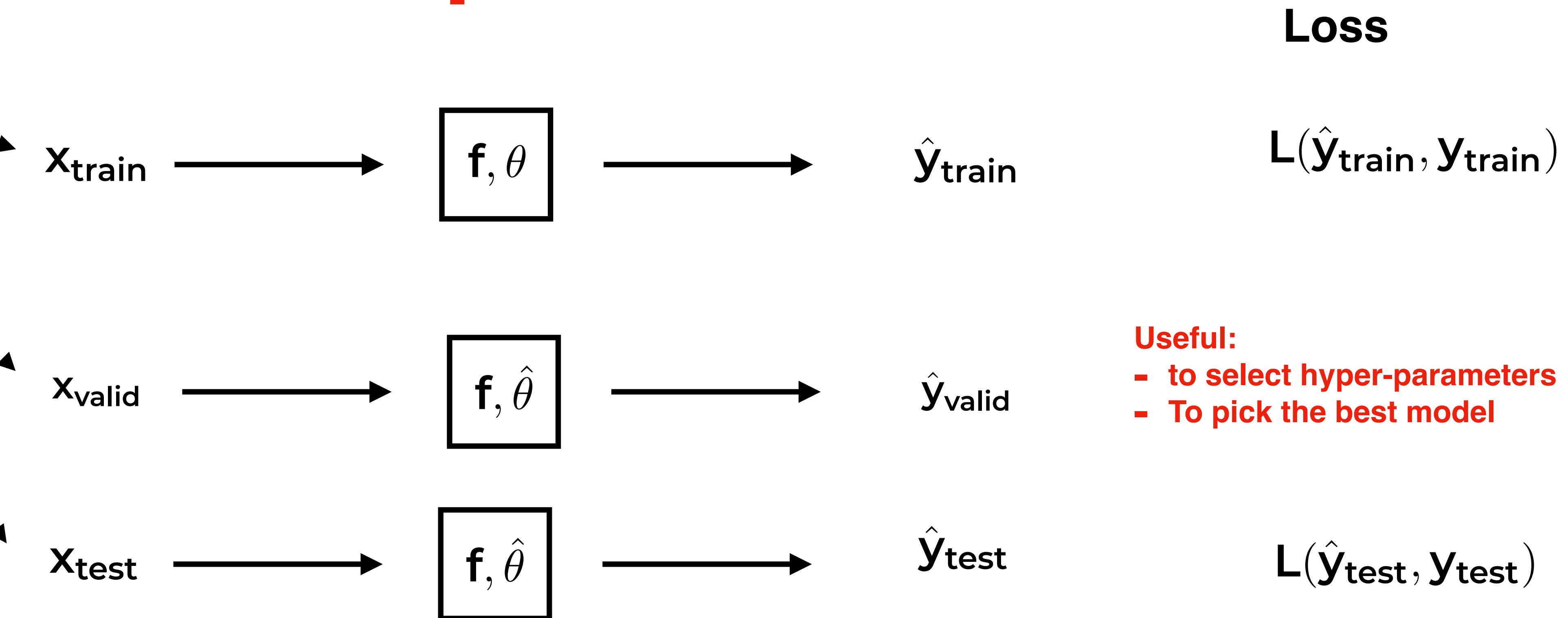


Distribution
over (x,y) :
 $P(x,y)$



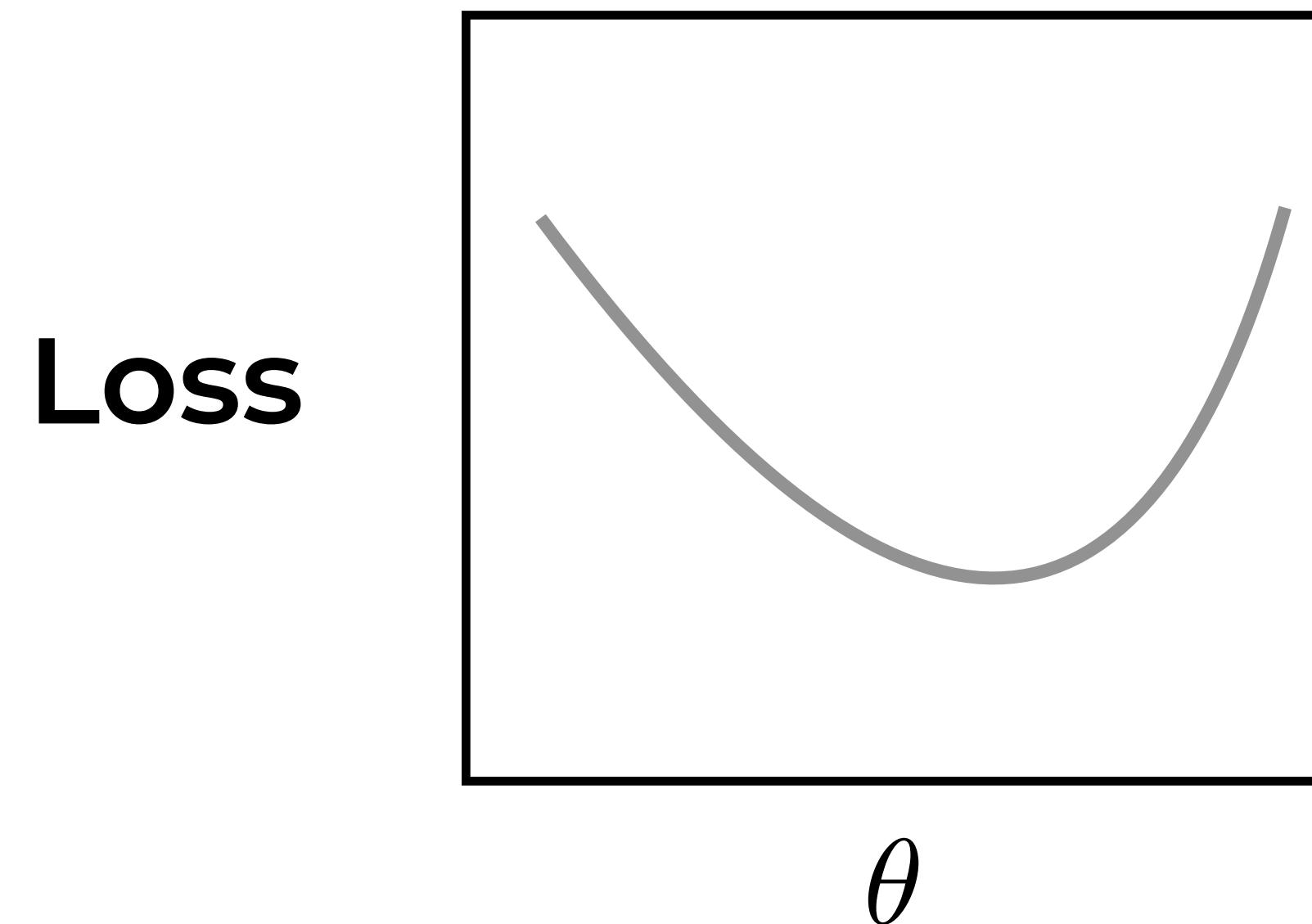
Learning Process

In practice



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:

$$\frac{d \text{ Loss}}{d\theta}$$

Different models

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

- Given a training set: $\{(\mathbf{x}_{\text{train}}, \mathbf{y}_{\text{train}})\}$

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

```
for t = 1, 2, ... (epochs) do
    for i = 1, 2, ... (datum) do
```

Stochastic Gradient Descent

- Obtain the predictions $\{f(\mathbf{x}_{\text{train}}; \hat{\theta}_t)\}$ (Forward propagation)

- Compute the Loss: $\text{Loss}_{ti} := L(f(\mathbf{x}_i; \hat{\theta}_t), \mathbf{y}_i)$

- Find the derivative of the loss: $\frac{d \text{Loss}_{ti}}{d \hat{\theta}_t}$

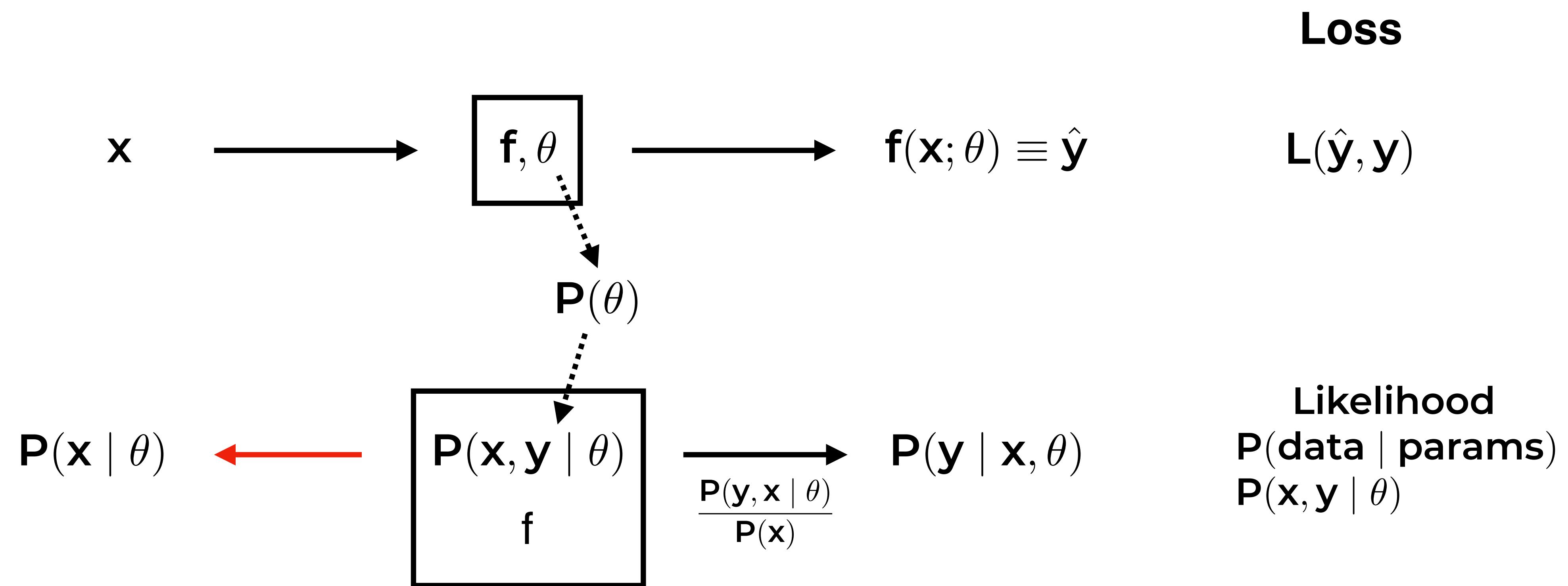
- Update parameters: $\hat{\theta}_{t+1} = \hat{\theta}_t - \alpha \frac{d \text{Loss}_{ti}}{d \hat{\theta}_t}$

- If $\|\hat{\theta}_{t+1} - \hat{\theta}_t\|_2^2 < \epsilon$ then stop

end for

end for

Probabilistic Models



Example

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

Likelihood for a single datum:

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

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?

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

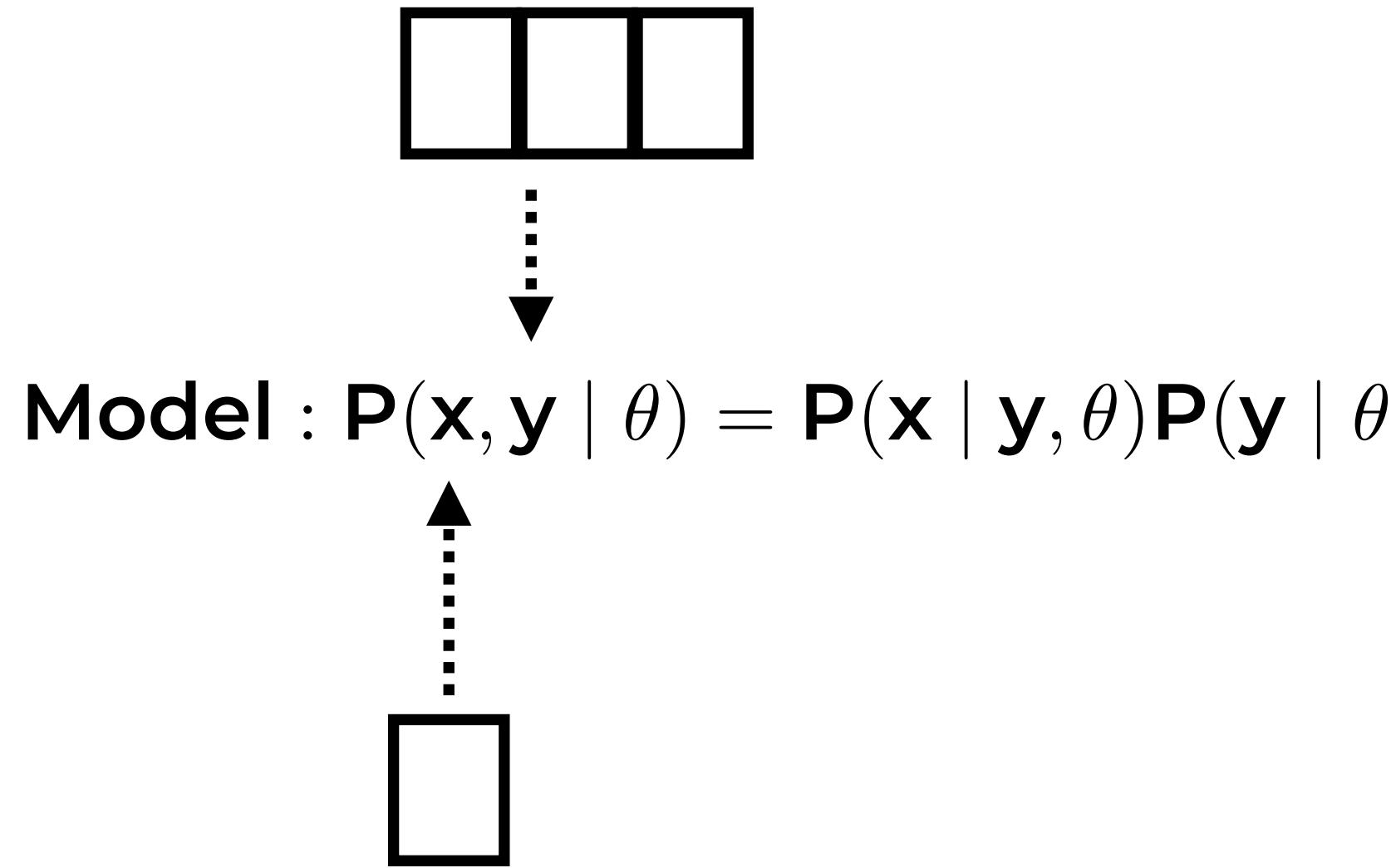
$$= \frac{d \frac{(x-\mu)^2}{2}}{d \mu}$$

$$= (x - \mu)$$

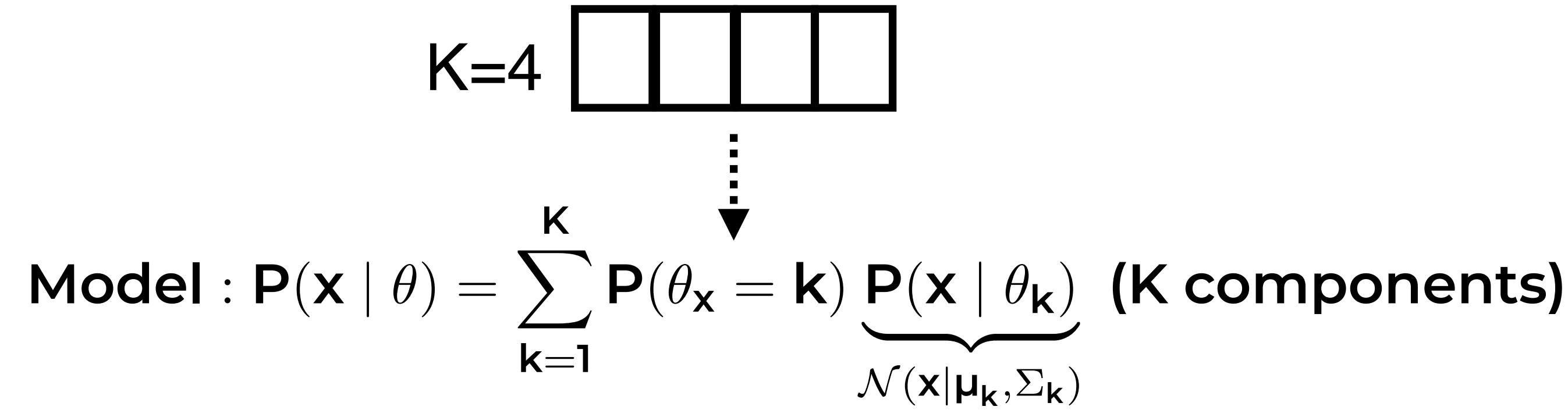
set to 0

$$\mu = x = 952$$

Data: (x,y) Naive Bayes



Data: x Gaussian Mixture Models



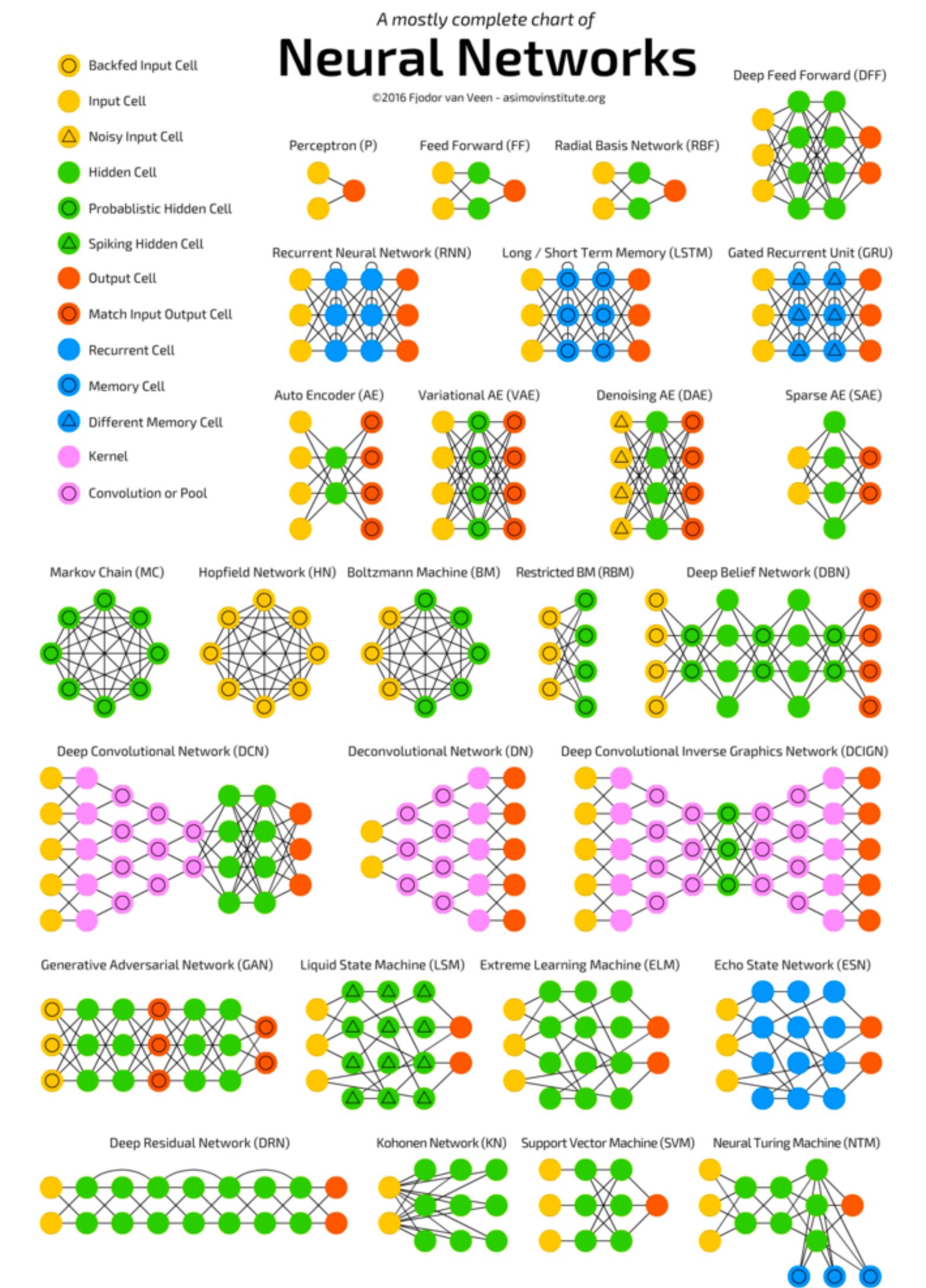
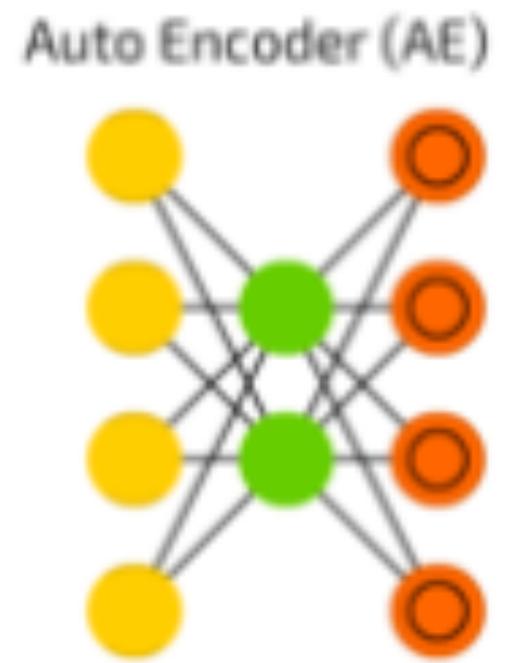
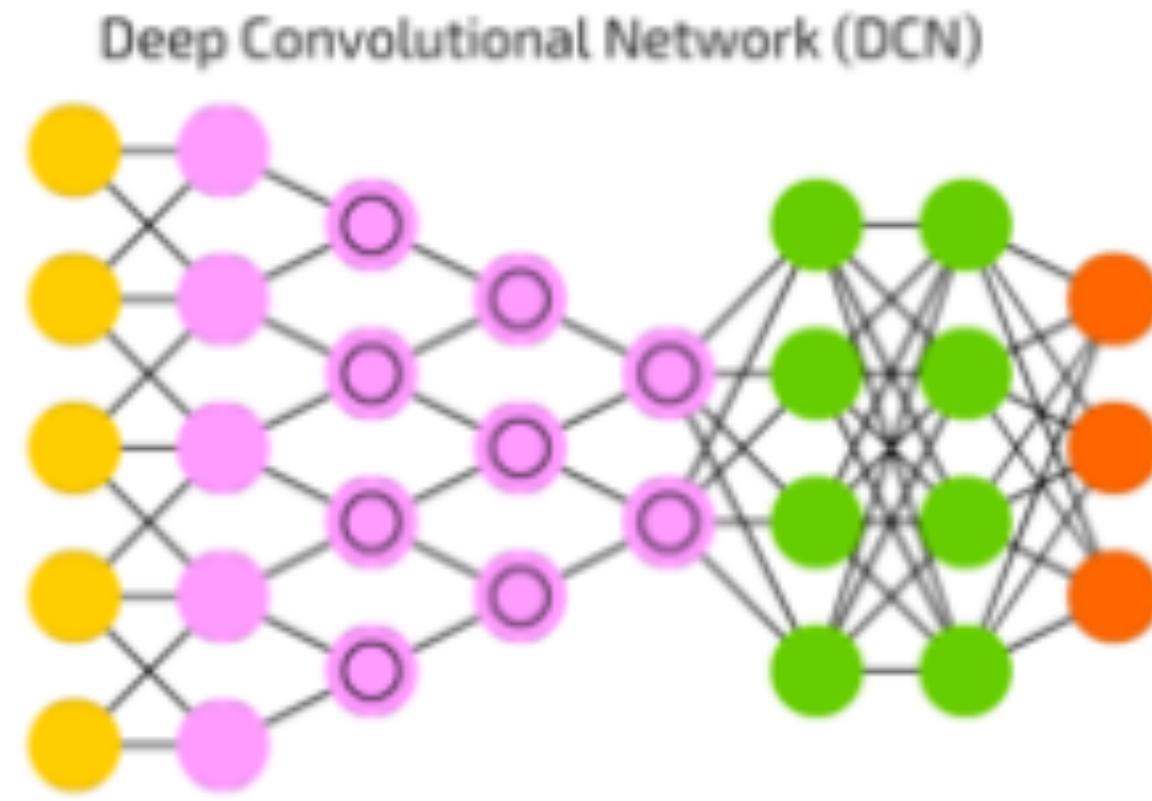
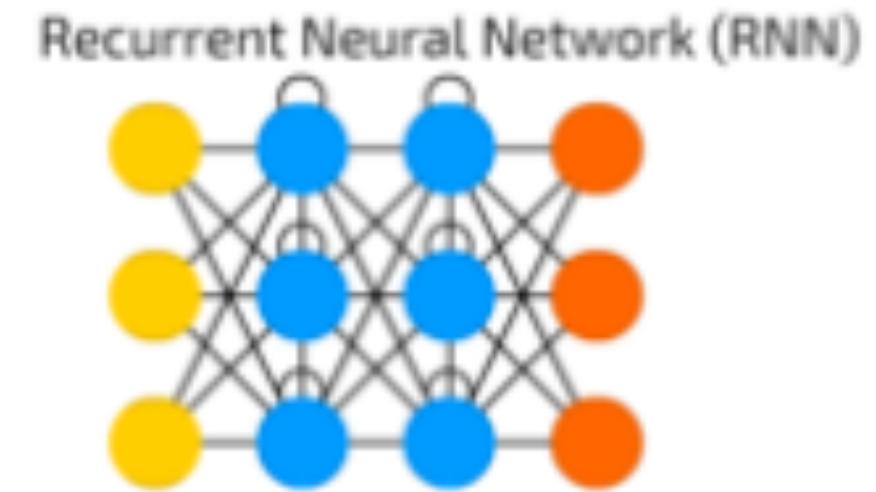
Max. likelihood (MLE) : $\hat{\theta}_{MLE} = \arg \max_{\theta} P(x | \theta)$

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

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.