Deep Learning: From Theory to Practice

Javier Montoya, Dr. Sc. ETH
javier.montoya@hslu.ch

(Responsible AI) Entrepreneurship & Innovation, Rotkreuz, 13th March 2023

Organisational Matters

Schedule:

18:30 - 19:20	Lecture
19:30 - 20:50	Lab Exercise

Material:

- Slides for the theoretical and practical part.
- Lab assignment (Jupyter Notebook, Google Colab).
- You are not allowed to distribute or use the material without consent.

#1 Introduction

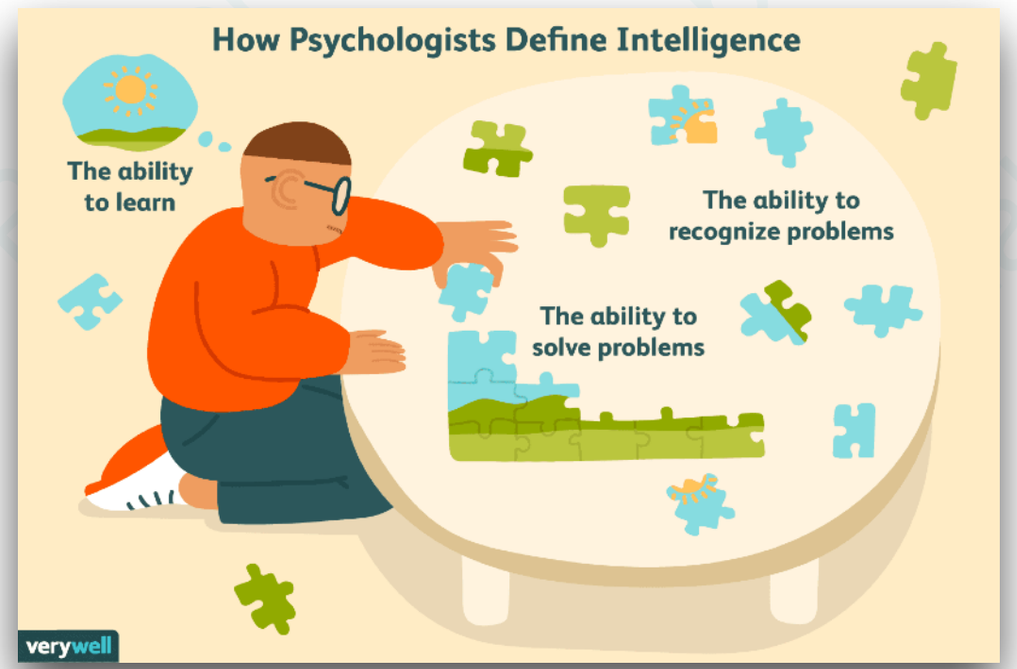
What is Artificial Intelligence?

Intelligence → *intelligere (latin)* → to comprehend or perceive**.** [Wikipedia]

Intelligence: general mental capacity that involves:

- Capability to reason and to plan
- Solve problems and think abstractly
- Comprehend complex ideas
- Learn from experience and apply knowledge to manipulate one's environment

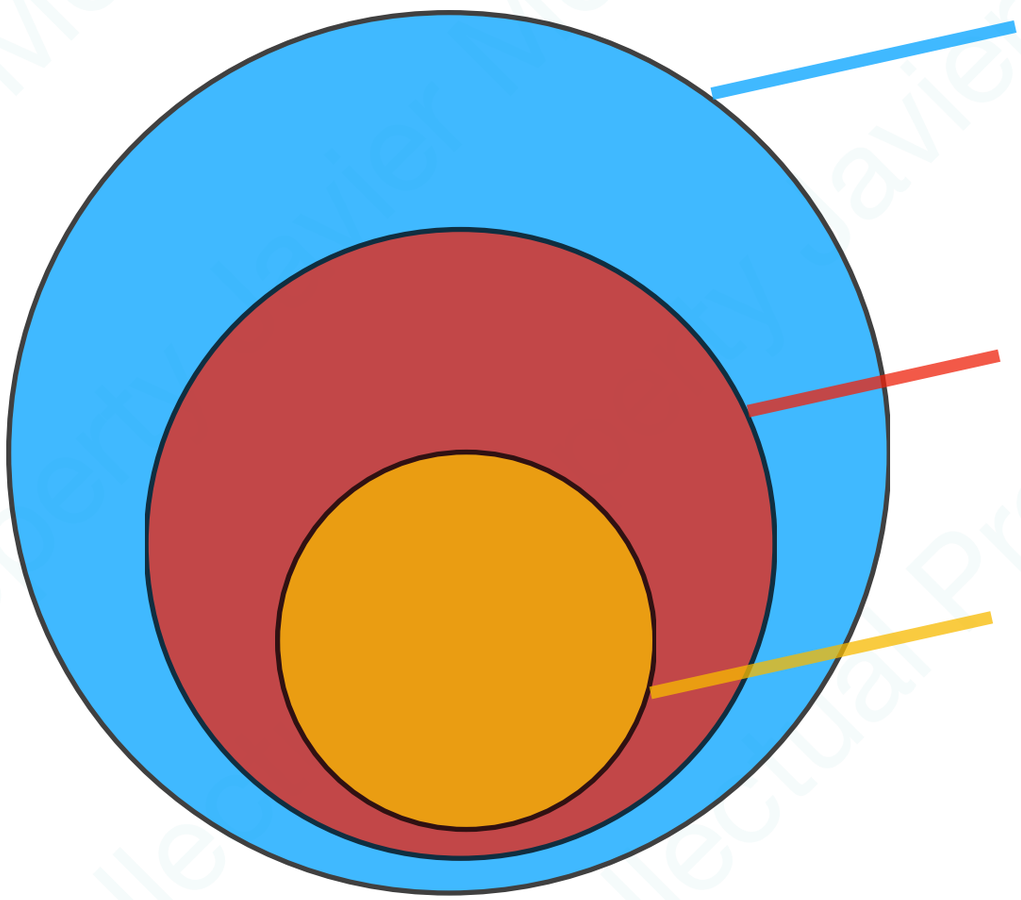
[Gottfredson, Mainstream science on intelligence]



- **Learn from experience:** The acquisition, retention, and use of knowledge is an important component of intelligence.
- **Recognise problems:** To put knowledge to use, people must be able to identify possible problems in the environment that need to be addressed.
- **Solve problems:** People must then be able to take what they have learned to come up with a useful solution to a problem they have noticed in the world around them.

[Verywellmind]

What is Artificial Intelligence?



Artificial Intelligence

- Techniques that enables machines to imitate intelligent human behaviour.
- Goal: solve tasks humans are good at, e.g sensing, reasoning, acting, adapting.

Machine Learning

- Subset of algorithms based on statistical models that enables machine: (i) to learn, (ii) improve from experience, and (iii) make accurate predictions.

Deep Learning

- Methods inspired by the information processing patterns in the **brain**.

What is Artificial Intelligence?

*“**Machine Learning** is the field of study that gives computers the ability to learn without being explicitly programmed”.*

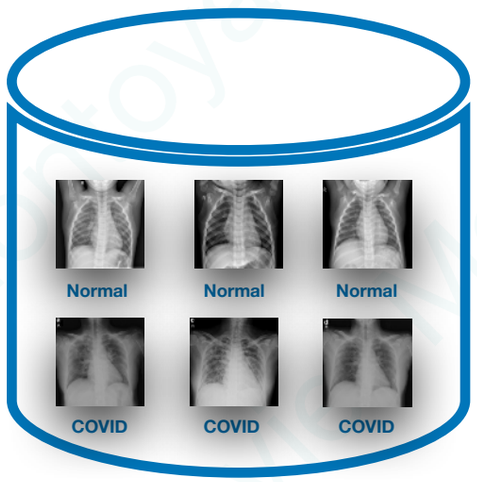
Arthur L. Samuel, AI Pioneer, 1959

#2 Machine Learning: Basic Concepts

The Neural Network Pipeline

TRAINING PHASE

Training Dataset



$$\mathbb{D} = \{x_i \in \mathbb{R}^d, y_i \in \mathbb{R}\}_{i=1}^{n}$$

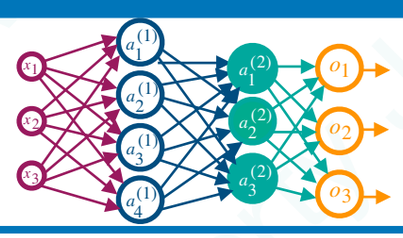
3 Compute Error via Loss Function $\hat{y} \neq y$

1 Train

Learned Model $f(\,)$

2 Predict

$\hat{y}$



$\hat{y}$ Predictions

y Labels

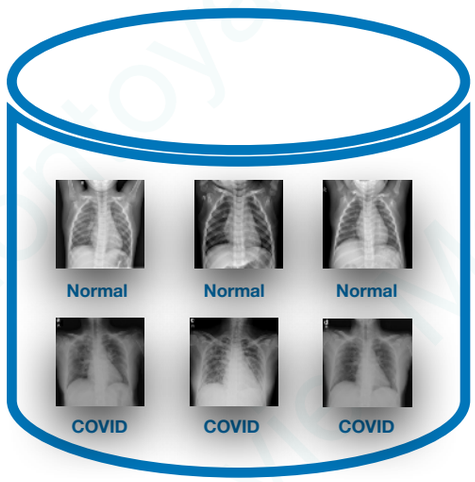
	VII	I
	I	XII

Confusion Matrix

The Neural Network Pipeline

TRAINING PHASE

Training Dataset



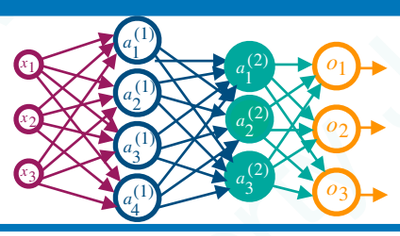
$$\mathbb{D} = \{x_i \in \mathbb{R}^d, y_i \in \mathbb{R}\}_{i=1}^{n}$$

1 Train

Learned Model $f(\,)$

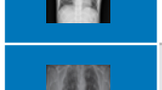
2 Predict $\hat{y}$

3 Compute Error via Loss Function $\hat{y} \neq y$

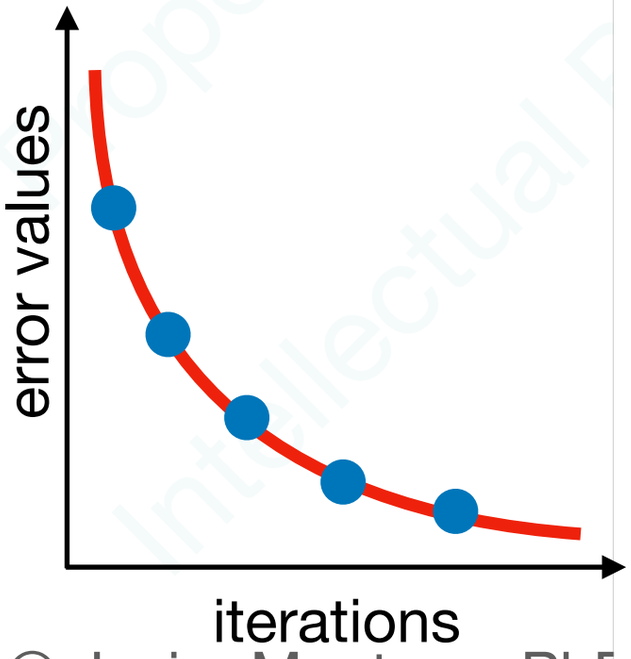


$\hat{y}$ Predictions

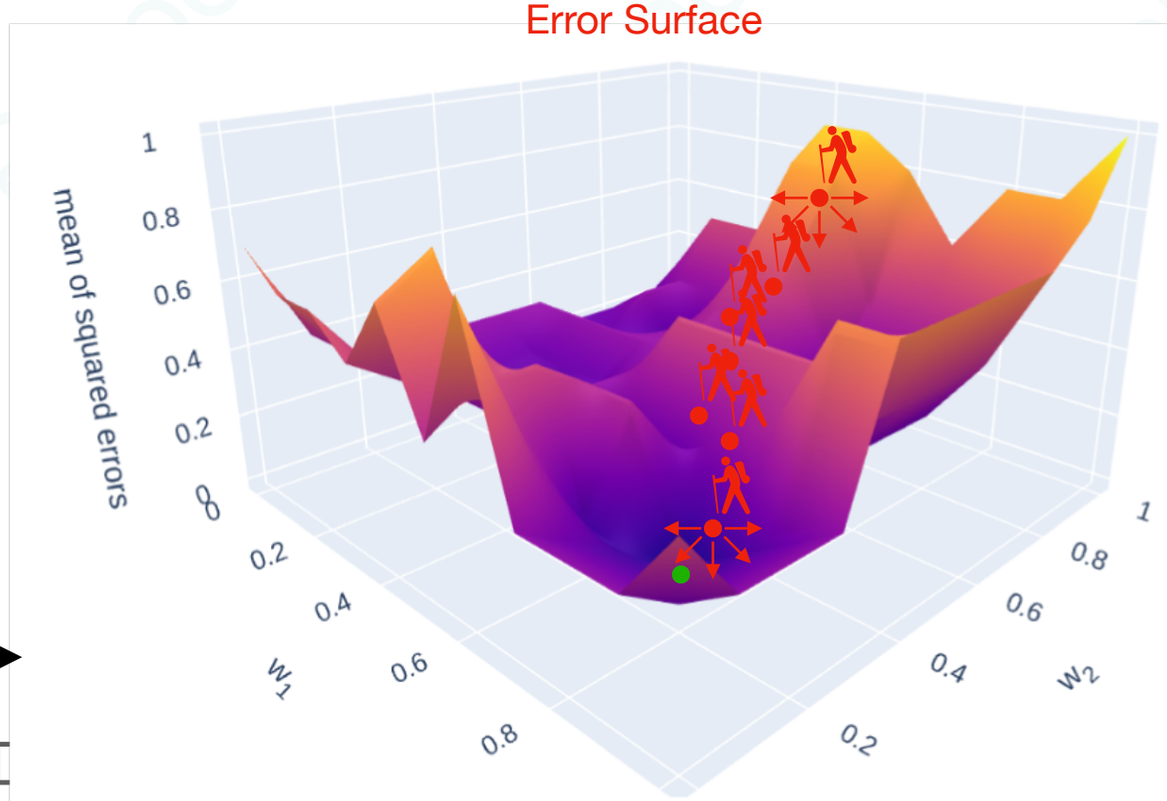
y Labels

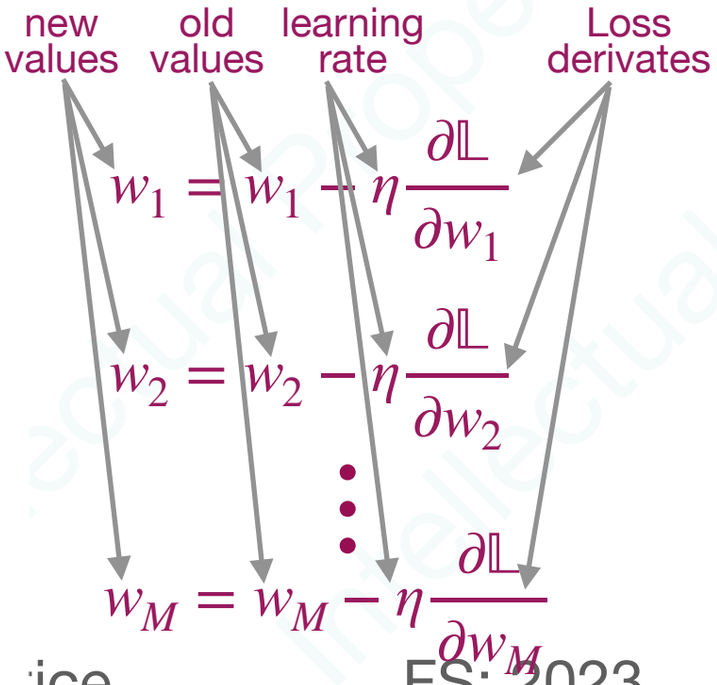
		
	VII	I
	I	XII

Confusion Matrix



Error Surface

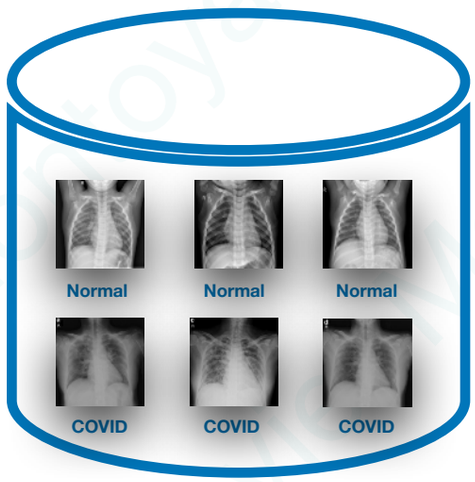




The Neural Network Pipeline

TRAINING PHASE

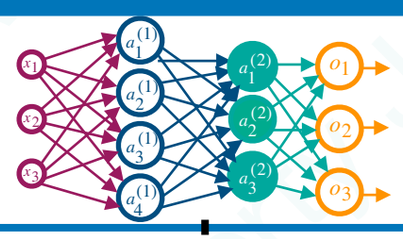
Training Dataset



$$\mathbb{D} = \{x_i \in \mathbb{R}^d, y_i \in \mathbb{R}\}_{i=1}^{n}$$

1 Train

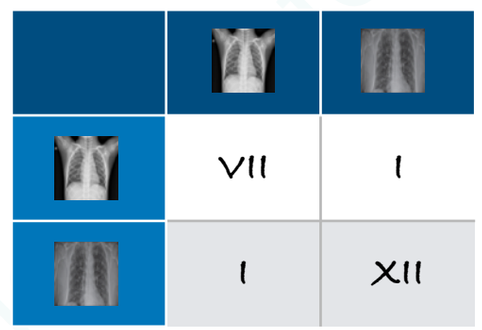
Learned Model $f()$



2 Predict → $\hat{y}$

3 Compute Error via Loss Function $\hat{y} \neq y$

$\hat{y}$ Predictions

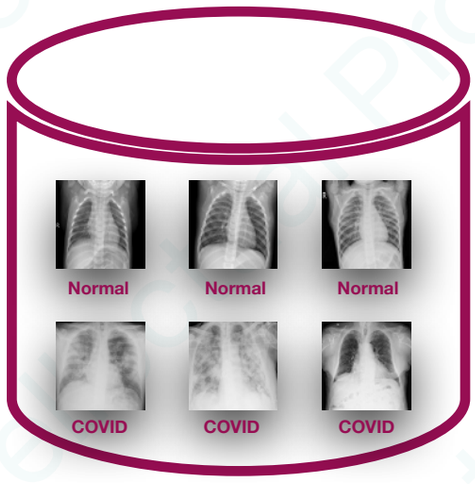


Confusion Matrix

4

TESTING PHASE

Unseen Dataset



$$\mathbb{D} = \{x_i \in \mathbb{R}^d, y_i \in \mathbb{R}\}_{i=1}^{n}$$

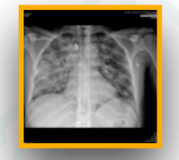
5 Evaluate

Learned Model $f()$

6 Output → $\hat{y}$

$$\hat{y} = f(x)$$

Prediction



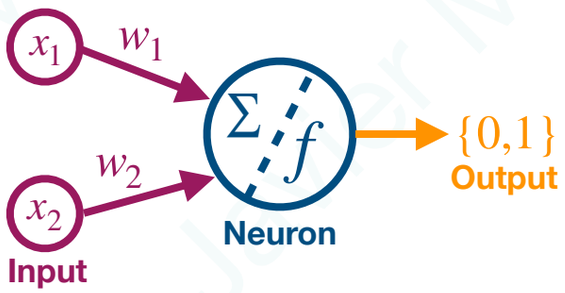
COVID

#3 Milestones of AI

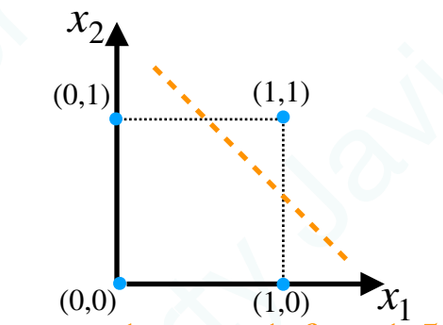
A Brief History of A.I

1943: McCulloch and Pitts (Artificial Neuron)

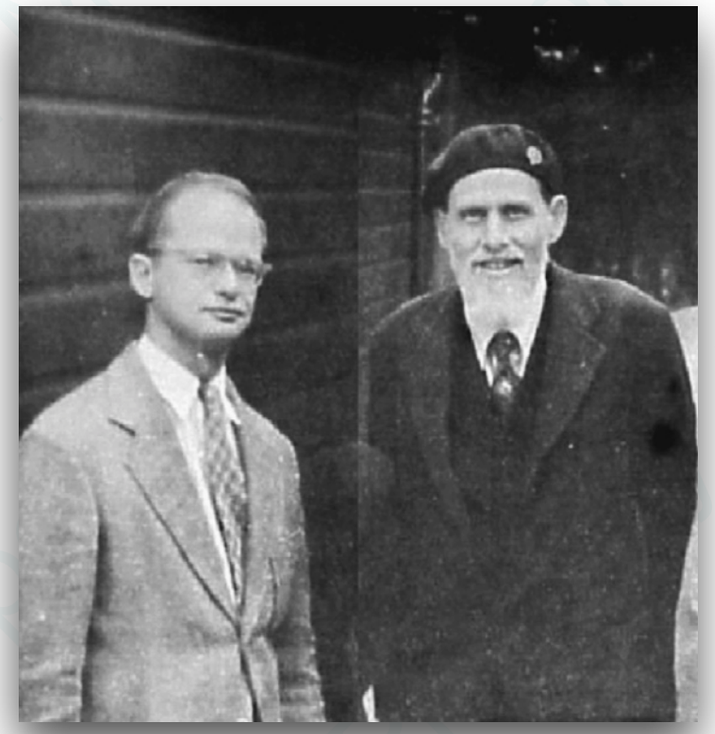
- Early model of artificial neuron:



0	0	0
0	1	0
1	0	0
1	1	1



$w_0 = 1, w_1 = 1, \theta = 1.5$



- Linear neural activation:

$$f_w(x) = \begin{cases} 1, & \text{if } w^T x \geq \theta \\ 0, & \text{if } w^T x < \theta \end{cases}$$

- Mainly used for AND/OR logic gates.
- No learning procedure.

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY*

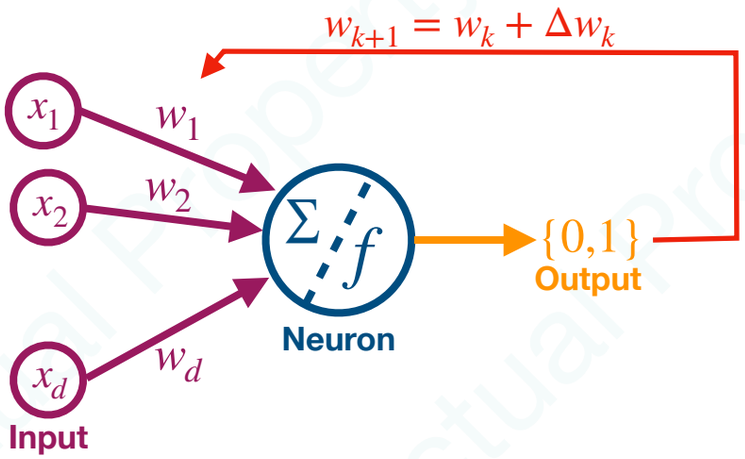
WARREN S. MCCULLOCH AND WALTER PITTS
University of Illinois, College of Medicine,
Department of Psychiatry at the Illinois Neuropsychiatric Institute,
University of Chicago, Chicago, U.S.A.

A Brief History of A.I

1958-1962: Rosenblatt (Perceptron)

- First model to train an artificial neuron.
- Optimisation of perceptron:

$$w_{k+1} = w_k + \Delta w_k$$



Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

F. ROSENBLATT

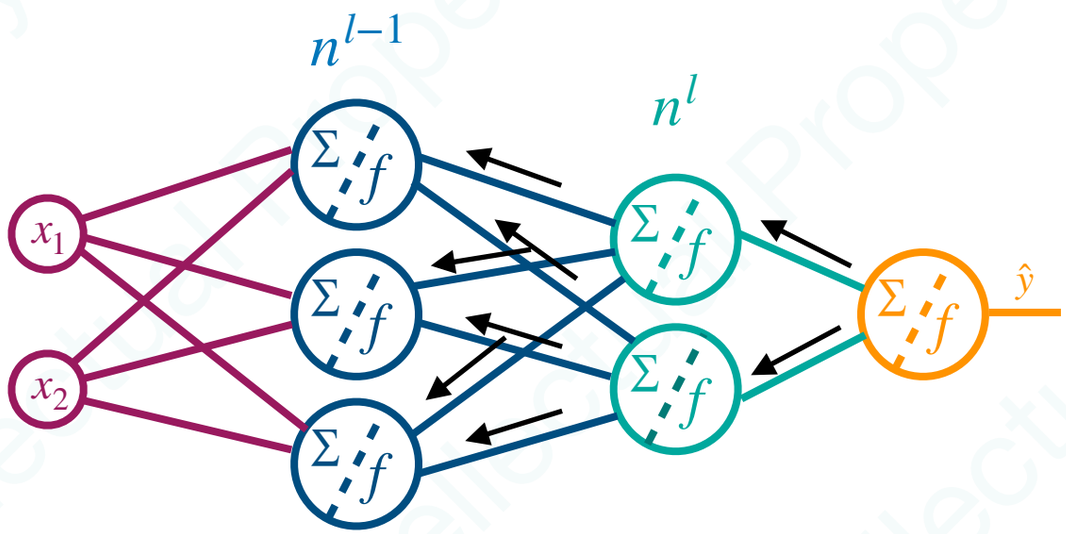
Cornell Aeronautical Laboratory

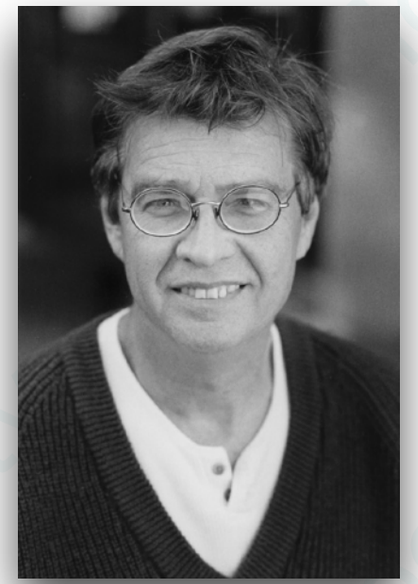
A Brief History of A.I

1986: Backpropagation Algorithm

- Main training procedure for actual deep networks based on gradients.

- Enables the automatic learning of model parameters.





Learning representations by back-propagating errors

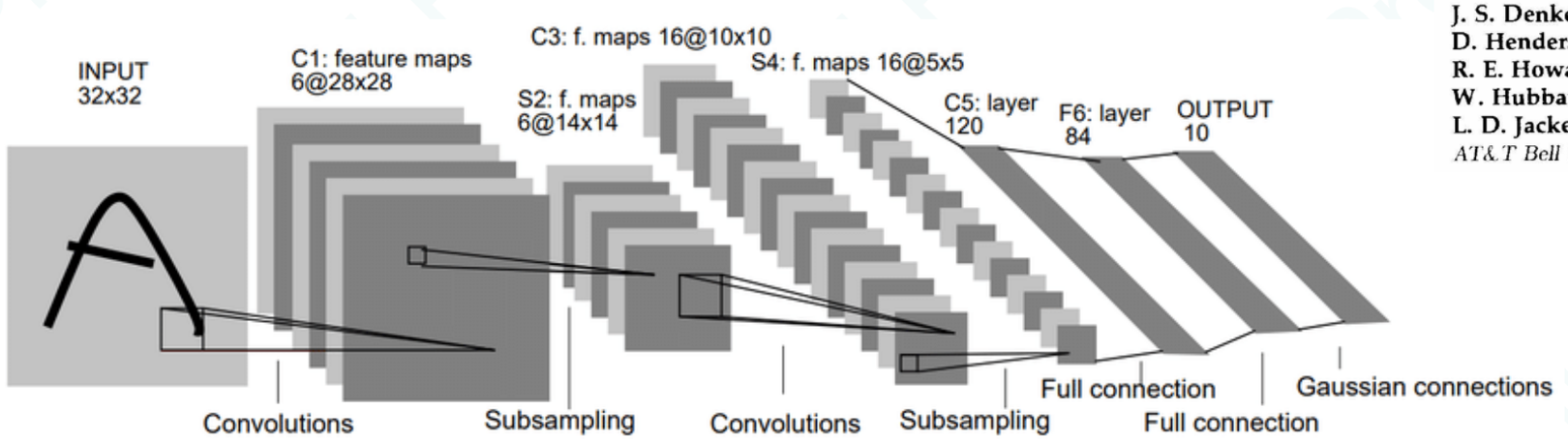
David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams

Nature **323**, 533–536 (1986) | Cite this article

A Brief History of A.I

1998: Convolutional Neural Networks

- Neural architecture intended for visual analysis tasks.
- Local features: feature sharing and pooling.
- Promising results on digit recognition.



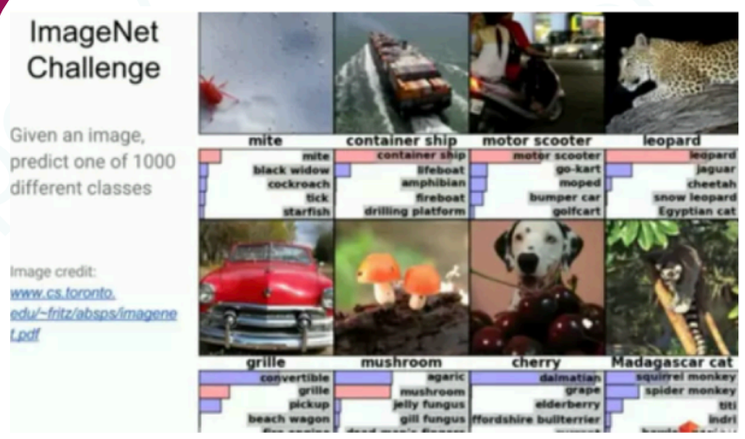
Backpropagation Applied to Handwritten Zip Code Recognition

Y. LeCun
B. Boser
J. S. Denker
D. Henderson
R. E. Howard
W. Hubbard
L. D. Jackel
AT&T Bell Laboratories Holmdel, NJ 07733 USA

A Brief History of A.I

2010: Benchmark datasets

- **Imagenet:**
  - Largest image recognition challenge.



IMAGENET (2010):

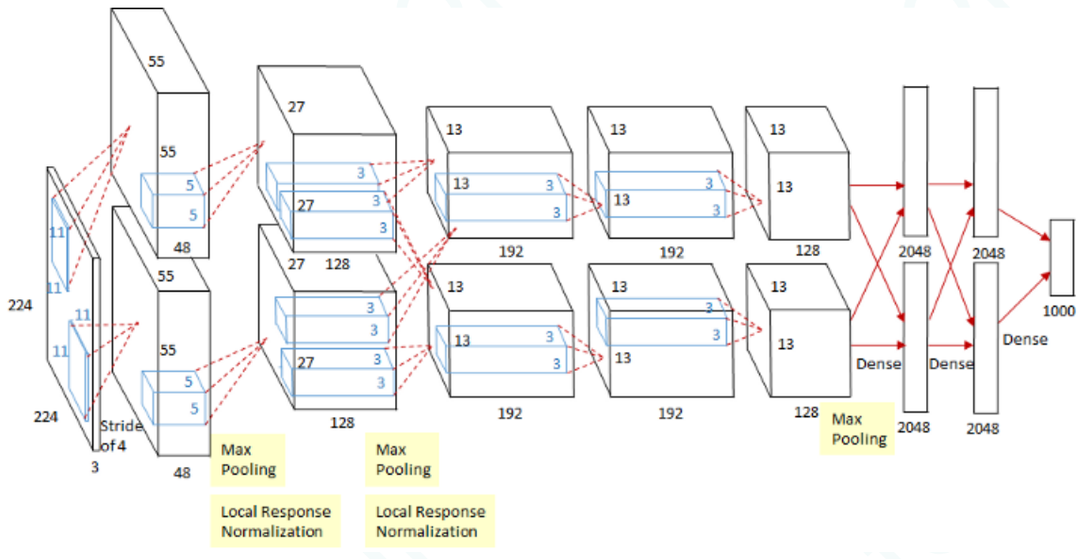
- ~1.5 Million of Images, 1000 classes.
- Image Dimensions: full resolution.
- Largest Dimension: 4288x2848x3.
- Smallest Dimension: 75x56x3.
- http://www.image-net.org

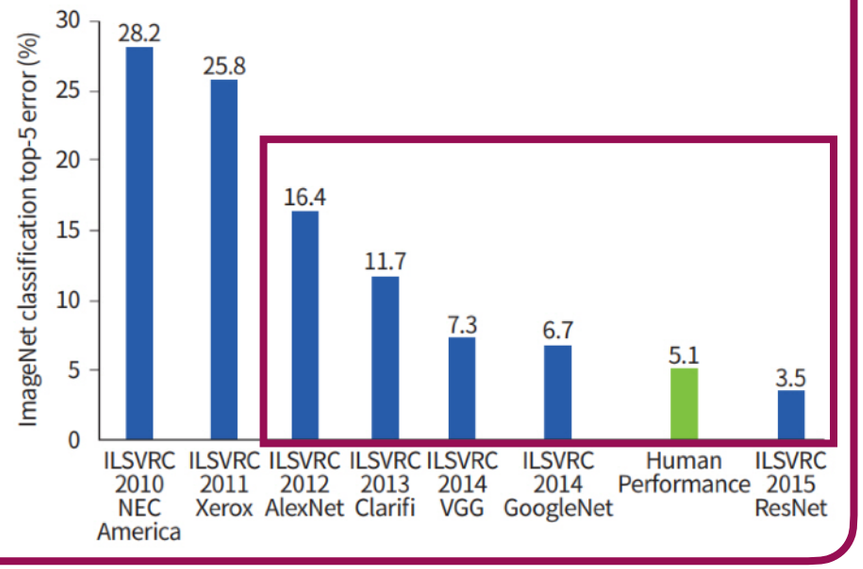
A Brief History of A.I

2012: Deep Learning Models

- **AlexNet:**
  - First deep learning model to win ImageNet competition.
  - Significant improvements over SOTA.

ImageNet Classification with Deep Convolutional Neural Networks





A Brief History of A.I

Why now?

1. Large Data

- Easier acquisition.
- Bigger numbers.

2. Hardware

- New optimised HW.
- Larger storage.

Graphic Processing Unit (GPU)

3. Software

- New Models
- New Toolboxes

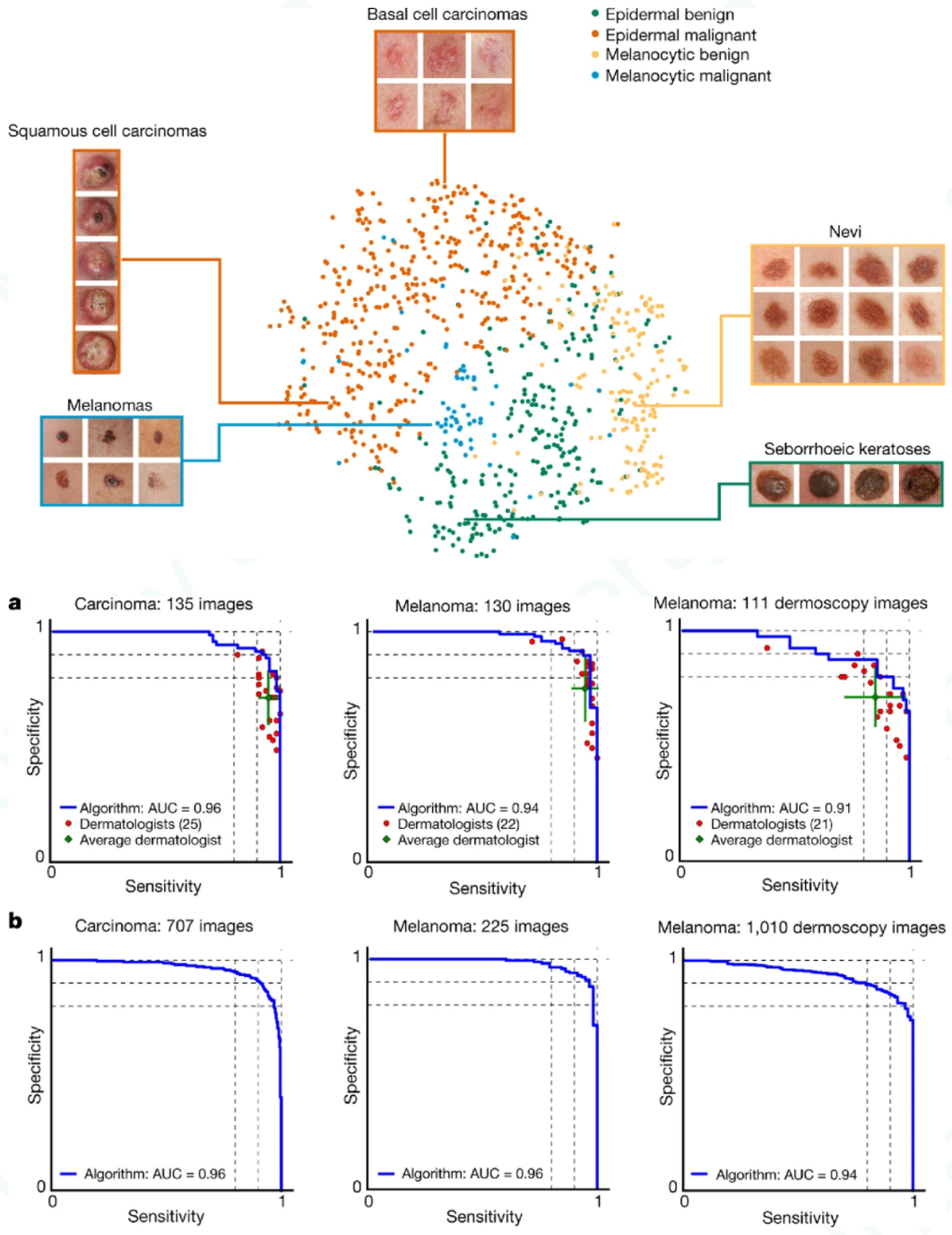
PyTorch

TensorFlow

#4 AI as Business Disruptor

AI as Business Disruptor

Digital Health



Overview:

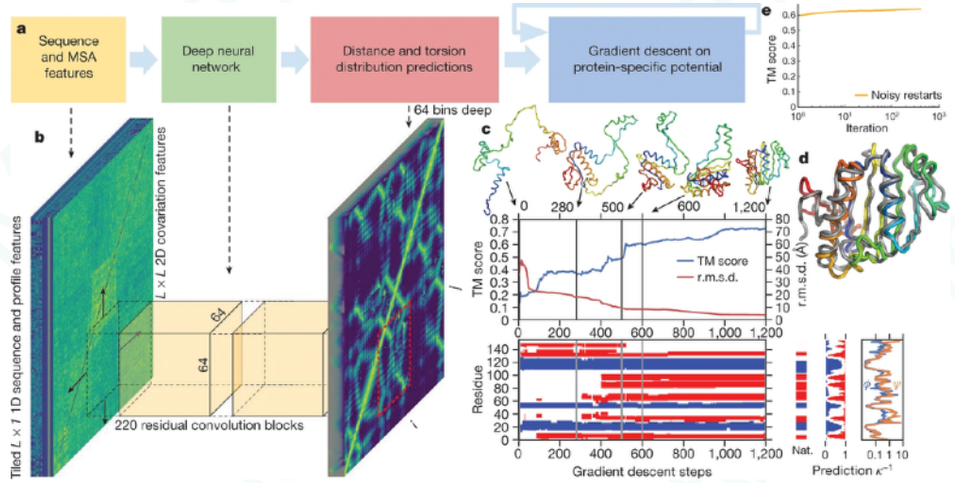
- Automated analysis of skin cancer based on clinical images and disease labels.
- Experiments were validated against 21 board-certified dermatologists.
- Cellphone based analysis provides access to diagnostic care (6.3billion)

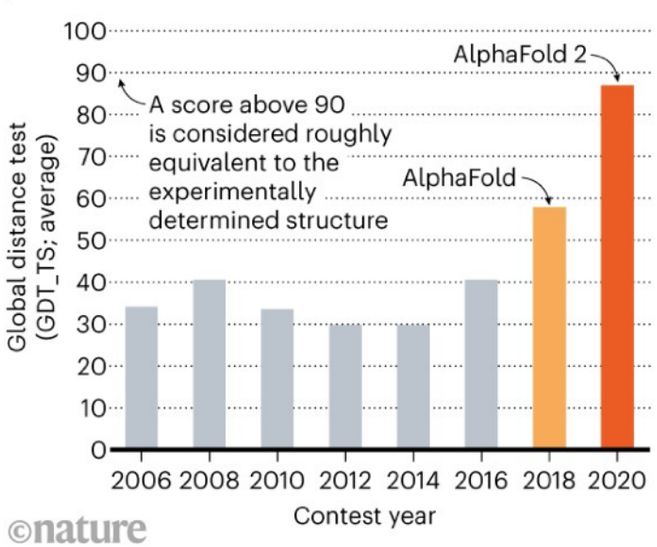
Dermatologist-level classification of skin cancer with deep neural networks

Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau & Sebastian Thrun

AI as Business Disruptor

Biology





Overview:

- Prediction of protein structure to predict the 3D shape of a protein.
- AlphaFold achieves a new SOTA.
- It provides new insights into the function and malfunction of proteins.

Improved protein structure prediction using potentials from deep learning

Andrew W. Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander W. R. Nelson, Alex Bridgland, Hugo Penedones, Stig Petersen, Karen Simonyan, Steve Crossan, Pushmeet Kohli, David T. Jones, David Silver, Koray Kavukcuoglu & Demis Hassabis

Nature **577**, 706–710 (2020) | Cite this article

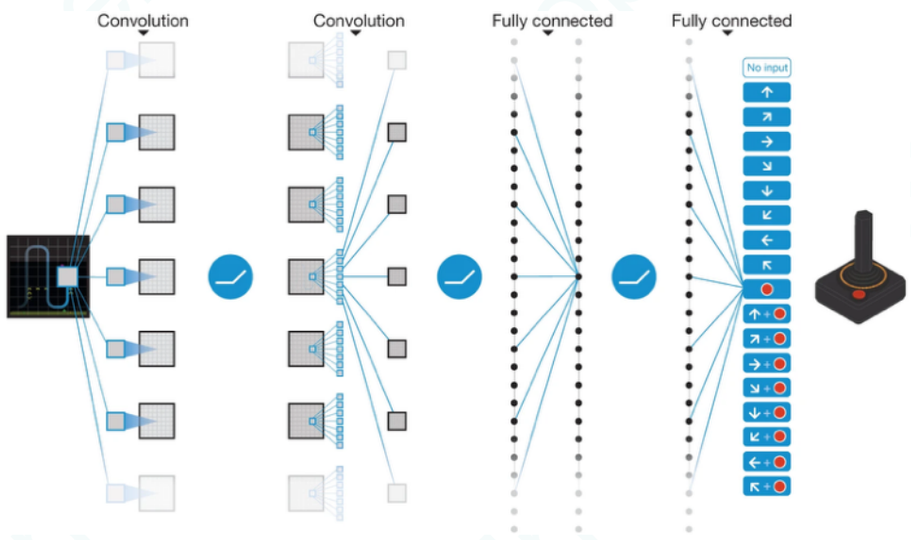
Highly accurate protein structure prediction with AlphaFold

John Jumper, Richard Evans, [...] Demis Hassabis

Nature **596**, 583–589 (2021) | Cite this article

AI as Business Disruptor

Gaming



Overview:

- Deep Q-Network: Novel AI agent that learns policies from high-dimensional sensory inputs.
- The Agent achieves professional human scores.
- It relies only on pixel information and game scores as inputs.

Human-level control through deep reinforcement learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis

Nature **518**, 529–533 (2015) | Cite this article

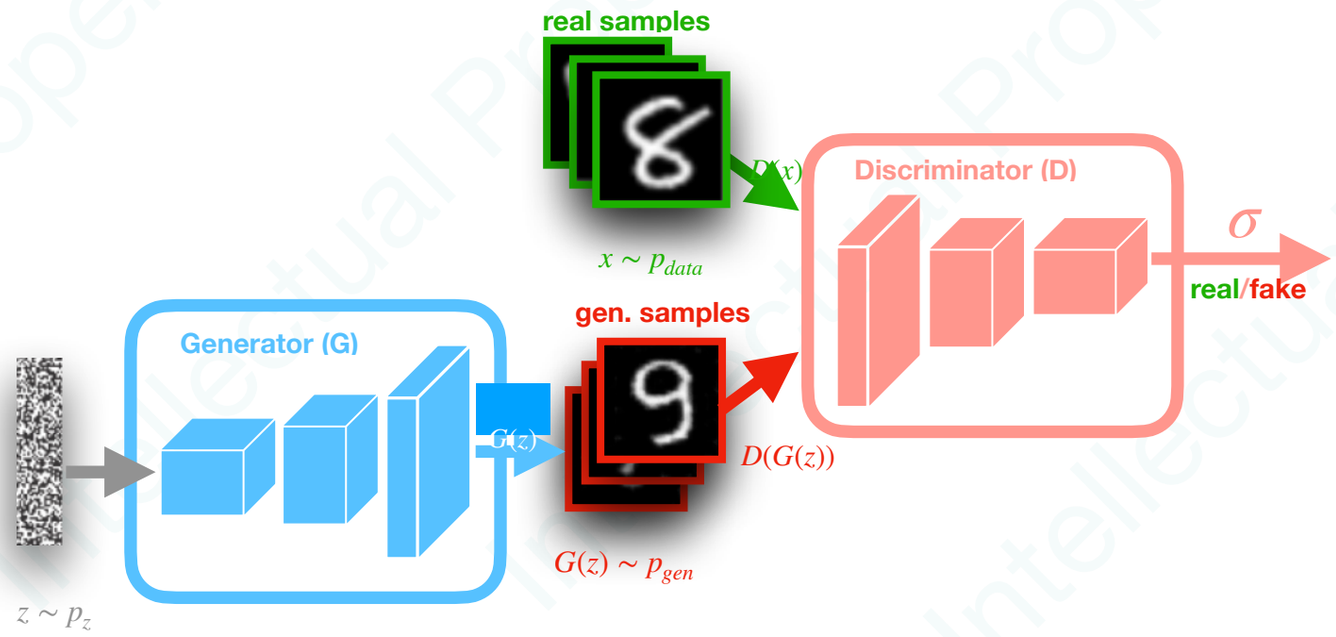
Self-taught system

AI as Business Disruptor

Data Synthesis

Overview:

- Enables the generation of real-looking images based on existing data.
- Applications can include: gaming, film industry, clothing industry, architecture, photography, etc.



Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie*, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair†, Aaron Courville, Yoshua Bengio‡
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

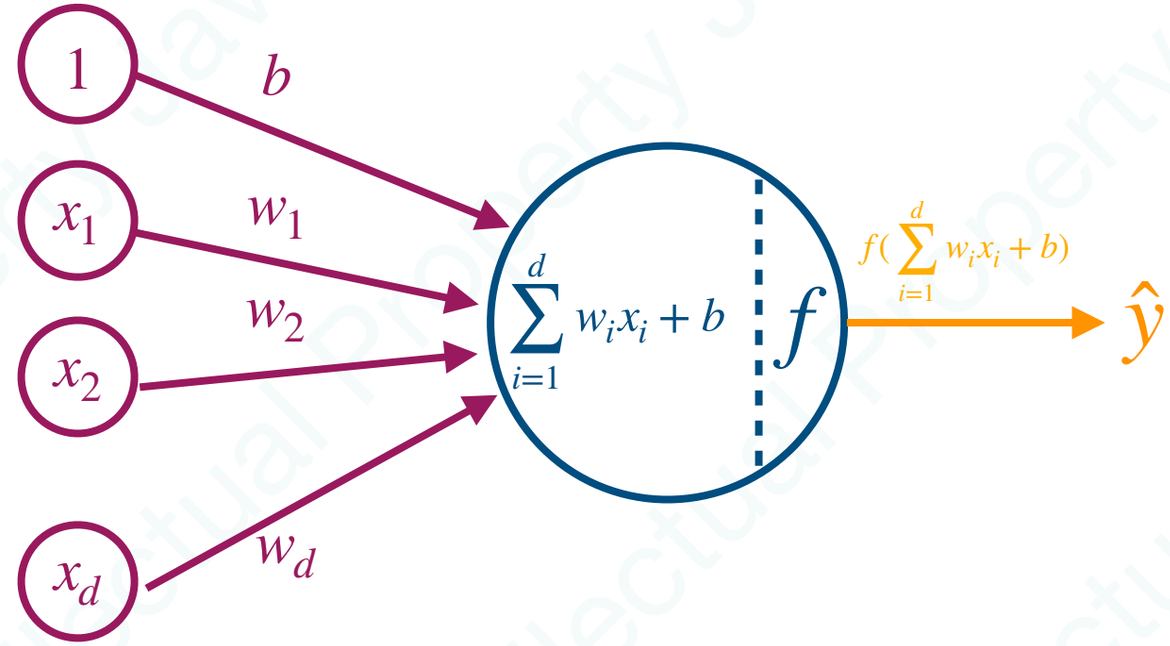
#5 Artificial Neural Networks

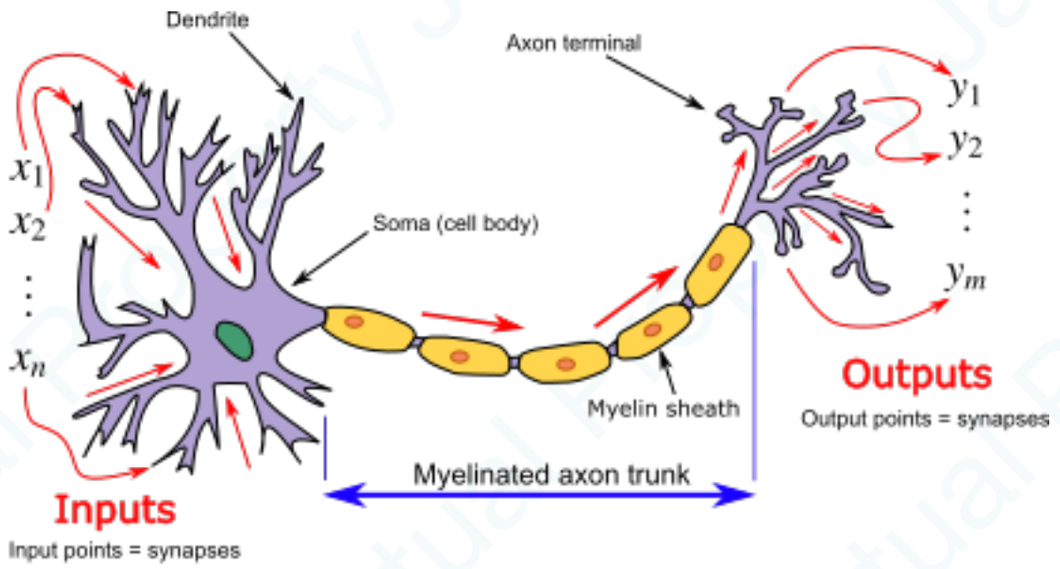
Artificial Neural Networks

The Neuron Metaphor

Biological Neurons:
- They accept information from multiple inputs.
- They activate and transmit information to other neurons.

Artificial Neurons:
- They multiple inputs by weights (edges).
- Apply some function to define their activation.



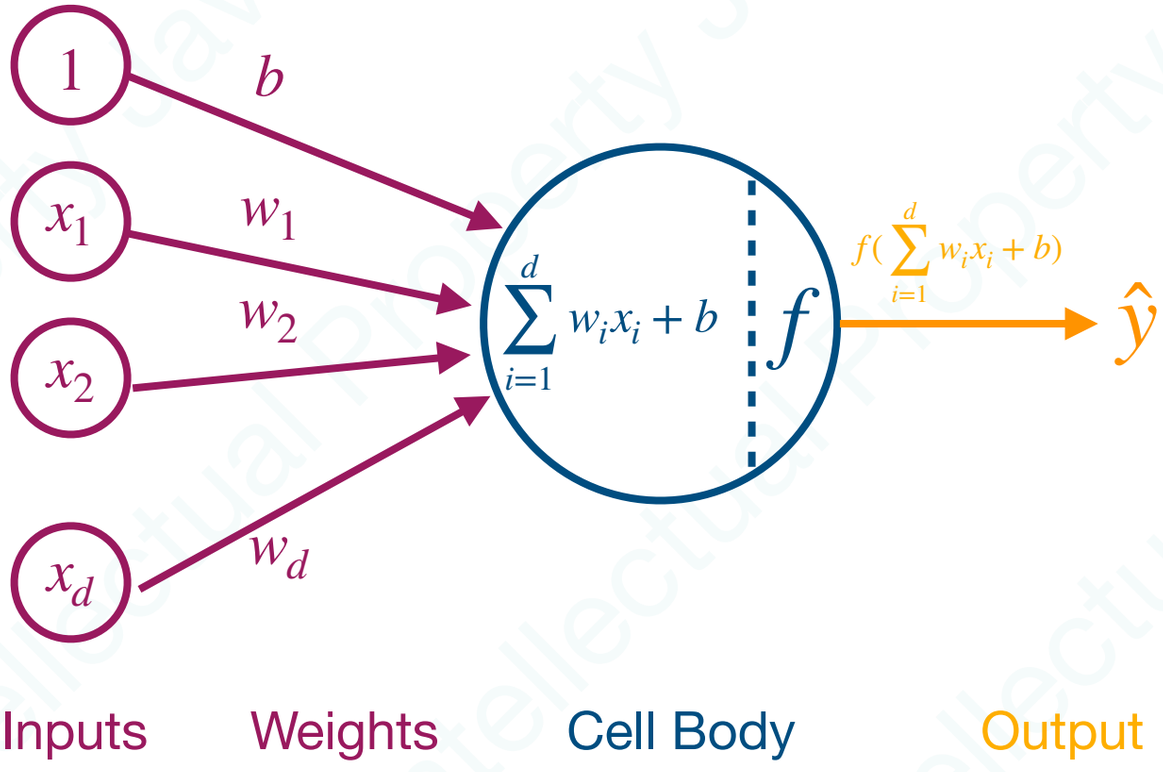


Artificial Neural Networks

The Perceptron Model

One of the earliest models based on the fundamental properties of the nervous system at that time. It comprises:

- A linear function that aggregates the input signals.
- A threshold function that determines whether the neuron fires or no.
- A learning mechanism to adjust the connection weights (error-correction).



Activation Function

Weighted Inputs

Output

$$\hat{y} = f(\sum_{i=1}^{d} w_i x_i + b)$$

$$= f(w^T x + b)$$

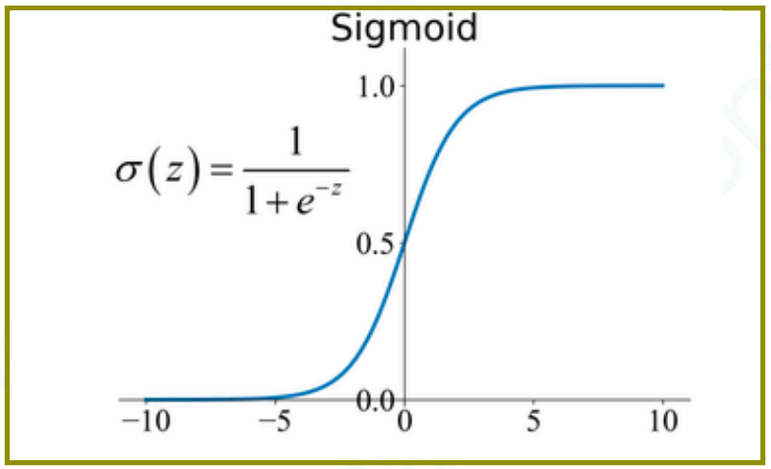
with:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}; \; W = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}; \; b \in \mathbb{R}$$

Artificial Neural Networks

Activation Functions: Types

Sigmoid Function



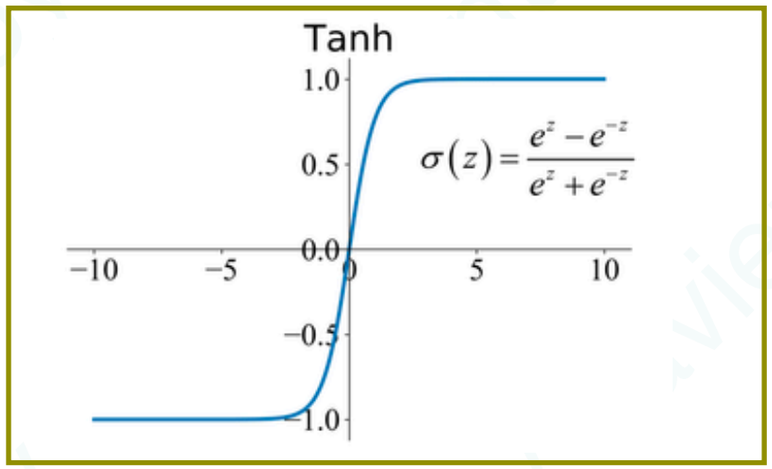
$$\textbf{sigma}(z) = \frac{1}{1+e^{-z}}$$

Overview:

- Normalises its input to the range [0,1].
- It is useful for predicting probabilities.

```
>>> m = nn.Sigmoid()
>>> input = torch.randn(2)
>>> output = m(input)
```

Hyperbolic Tangent



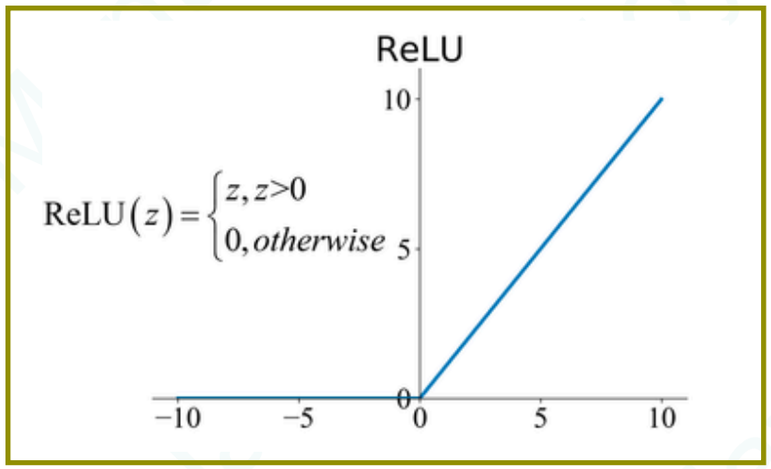
$$\textbf{tanh}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Overview:

- Normalises its input to the range [-1,1].
- It is mainly used for binary classification tasks

```
>>> m = nn.Tanh()
>>> input = torch.randn(2)
>>> output = m(input)
```

Rectified Linear Unit



$$\textbf{ReLU}(z) = \textbf{max}(0,z)$$

Overview:

- Normalises its input to the range $[0,\inf[$
- All the negative values become zero.

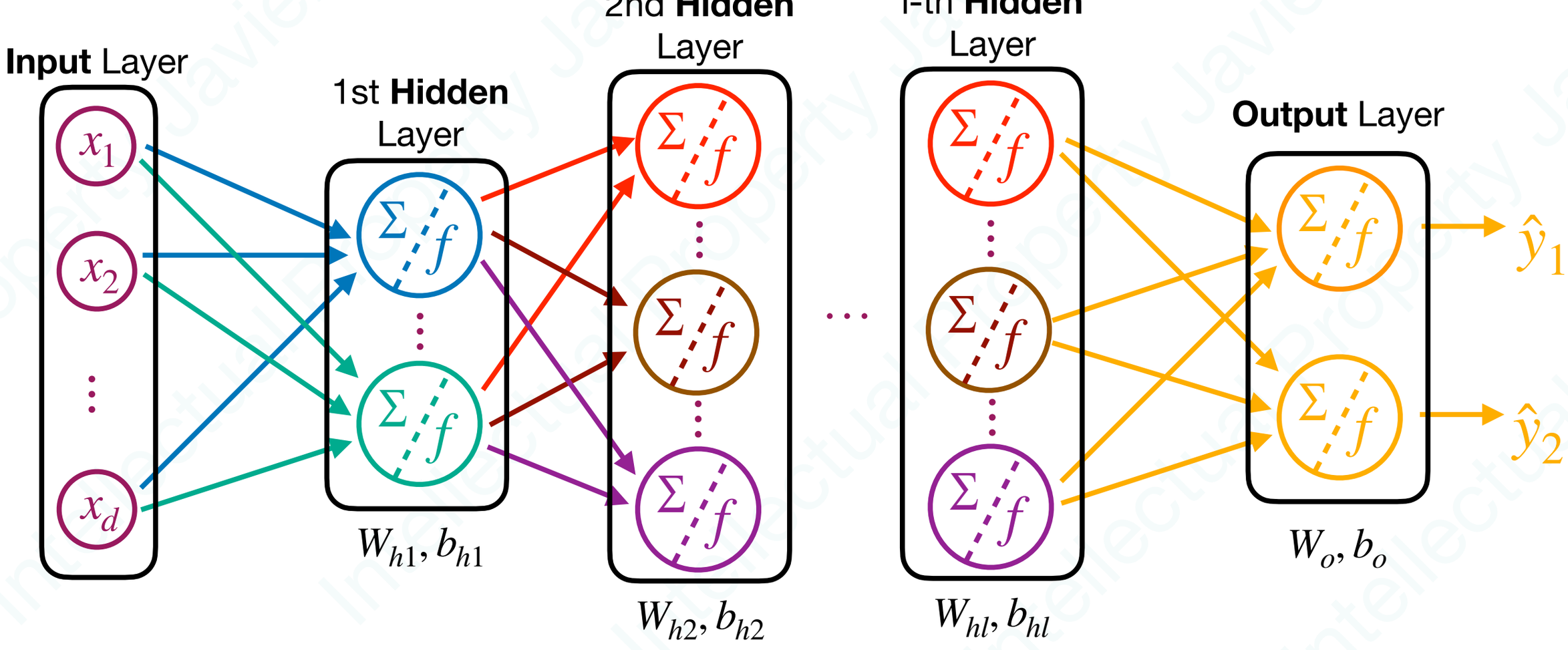
```
>>> m = nn.ReLU()
>>> input = torch.randn(2)
>>> output = m(input)
```

Artificial Neural Networks

Multi-layer Perceptron

A **MLP** is a supervised learning approach that learns a function $f(\ ) : \mathbb{R}^d \rightarrow \mathbb{R}^o$ by training on a dataset $\mathbb{D}$.

- A MLP learns a non-linear function approximation for classification or regression.
- It consists of a set of layers: {input, hidden, output} containing a set of neurons.
- Different hyper parameters need to be tuned.

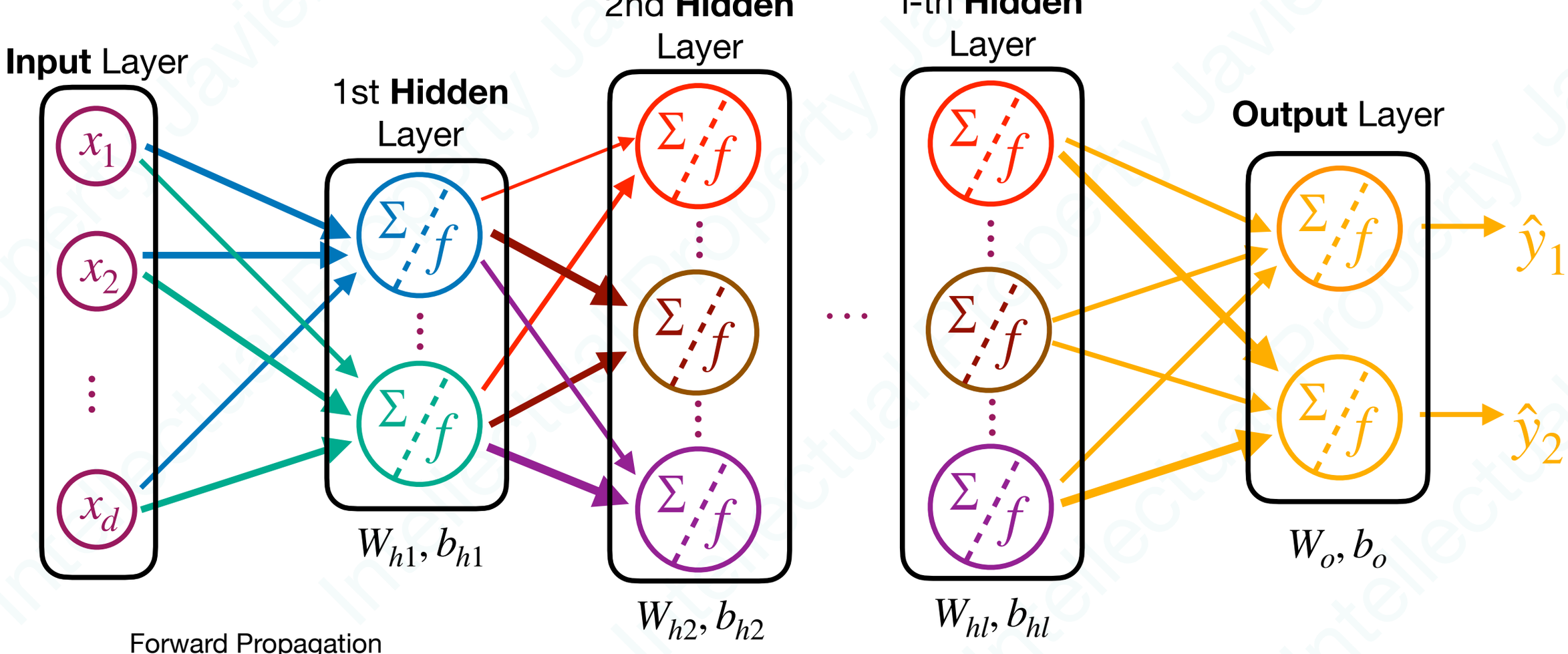


Artificial Neural Networks

Multi-layer Perceptron

A **MLP** is a supervised learning approach that learns a function $f(\ ) : \mathbb{R}^d \rightarrow \mathbb{R}^o$ by training on a dataset $\mathbb{D}$.

- A MLP learns a non-linear function approximation for classification or regression.
- It consists of a set of layers: {input, hidden, output} containing a set of neurons.
- Different hyper parameters need to be tuned.

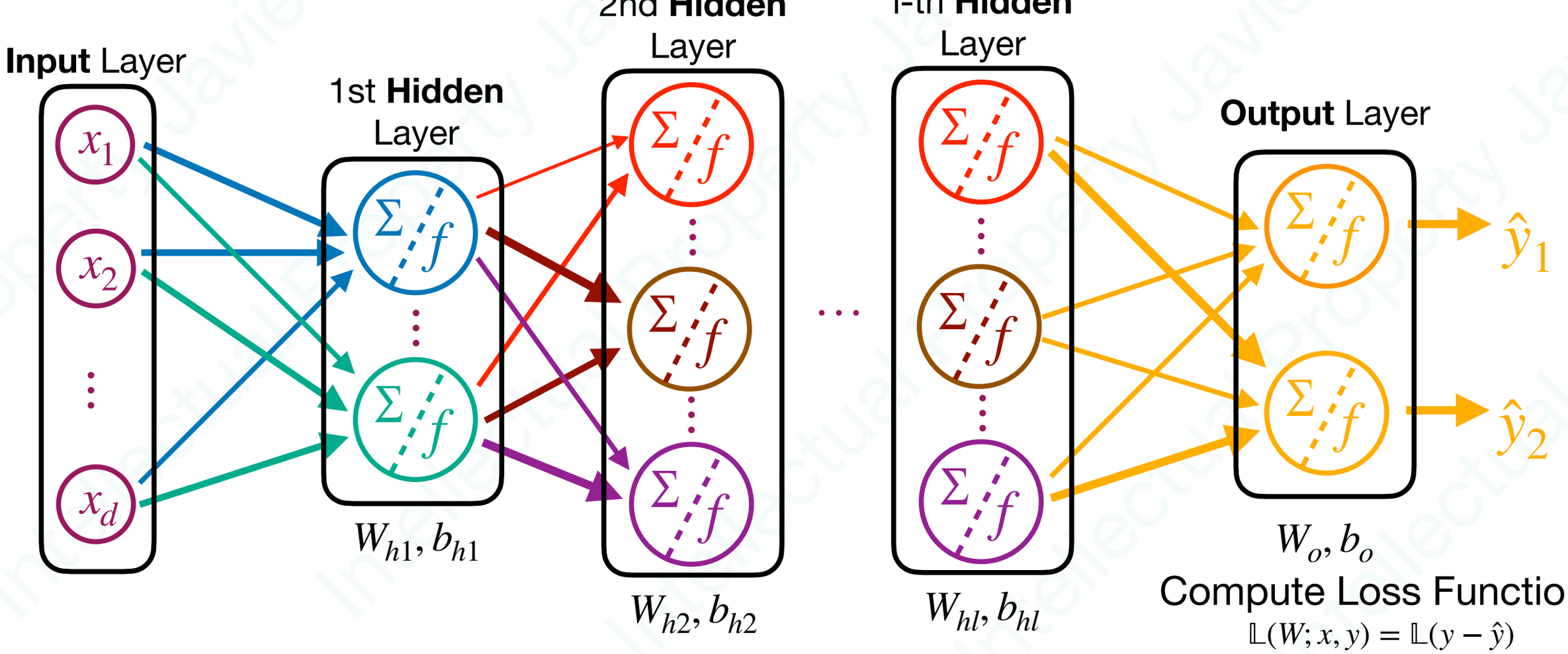


Artificial Neural Networks

Multi-layer Perceptron

A **MLP** is a supervised learning approach that learns a function $f(\ ) : \mathbb{R}^d \rightarrow \mathbb{R}^o$ by training on a dataset $\mathbb{D}$.

- A MLP learns a non-linear function approximation for classification or regression.
- It consists of a set of layers: {input, hidden, output} containing a set of neurons.
- Different hyper parameters need to be tuned.

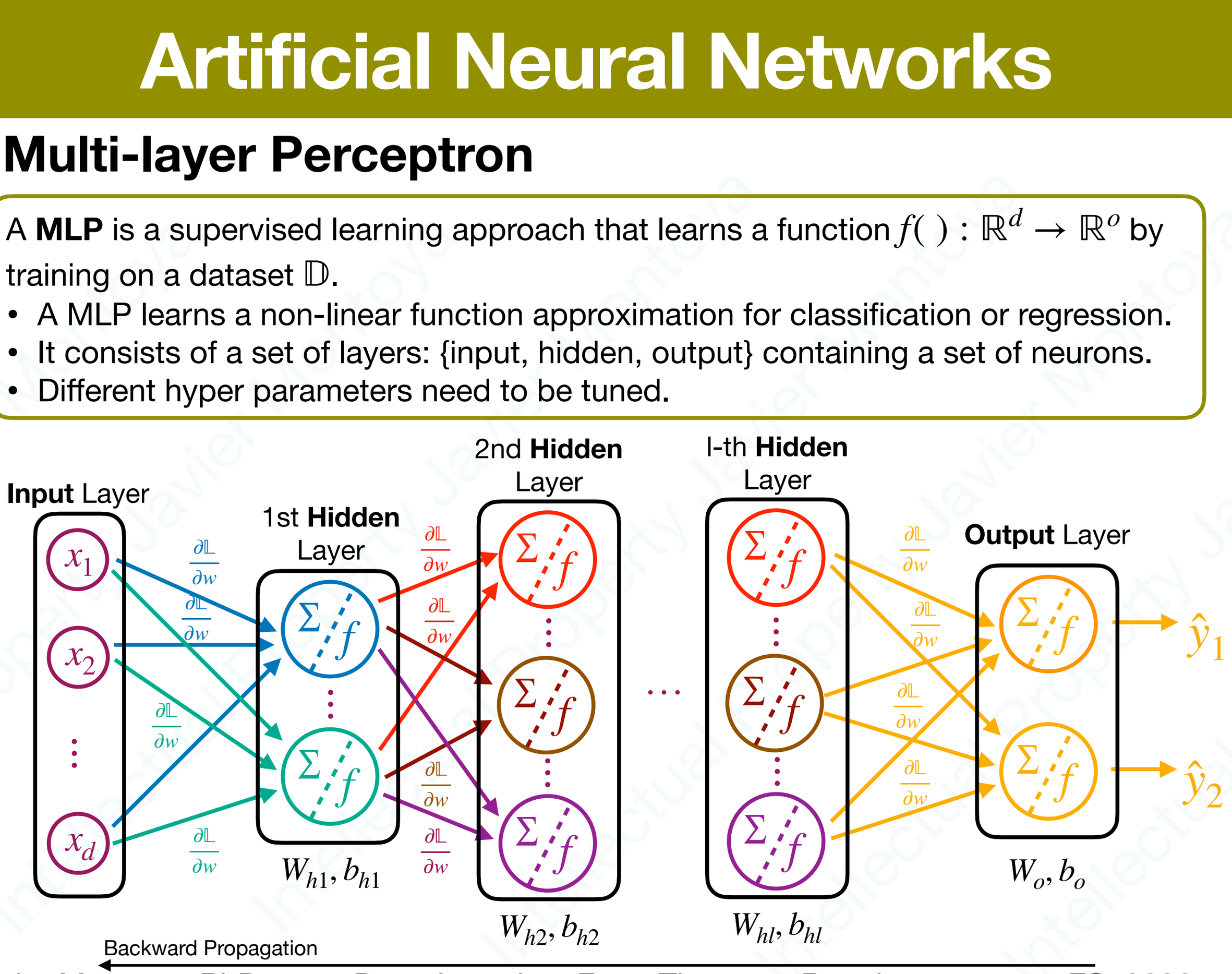


Artificial Neural Networks

Multi-layer Perceptron

A **MLP** is a supervised learning approach that learns a function $f(\ ) : \mathbb{R}^d \rightarrow \mathbb{R}^o$ by training on a dataset $\mathbb{D}$.

- A MLP learns a non-linear function approximation for classification or regression.
- It consists of a set of layers: {input, hidden, output} containing a set of neurons.
- Different hyper parameters need to be tuned.

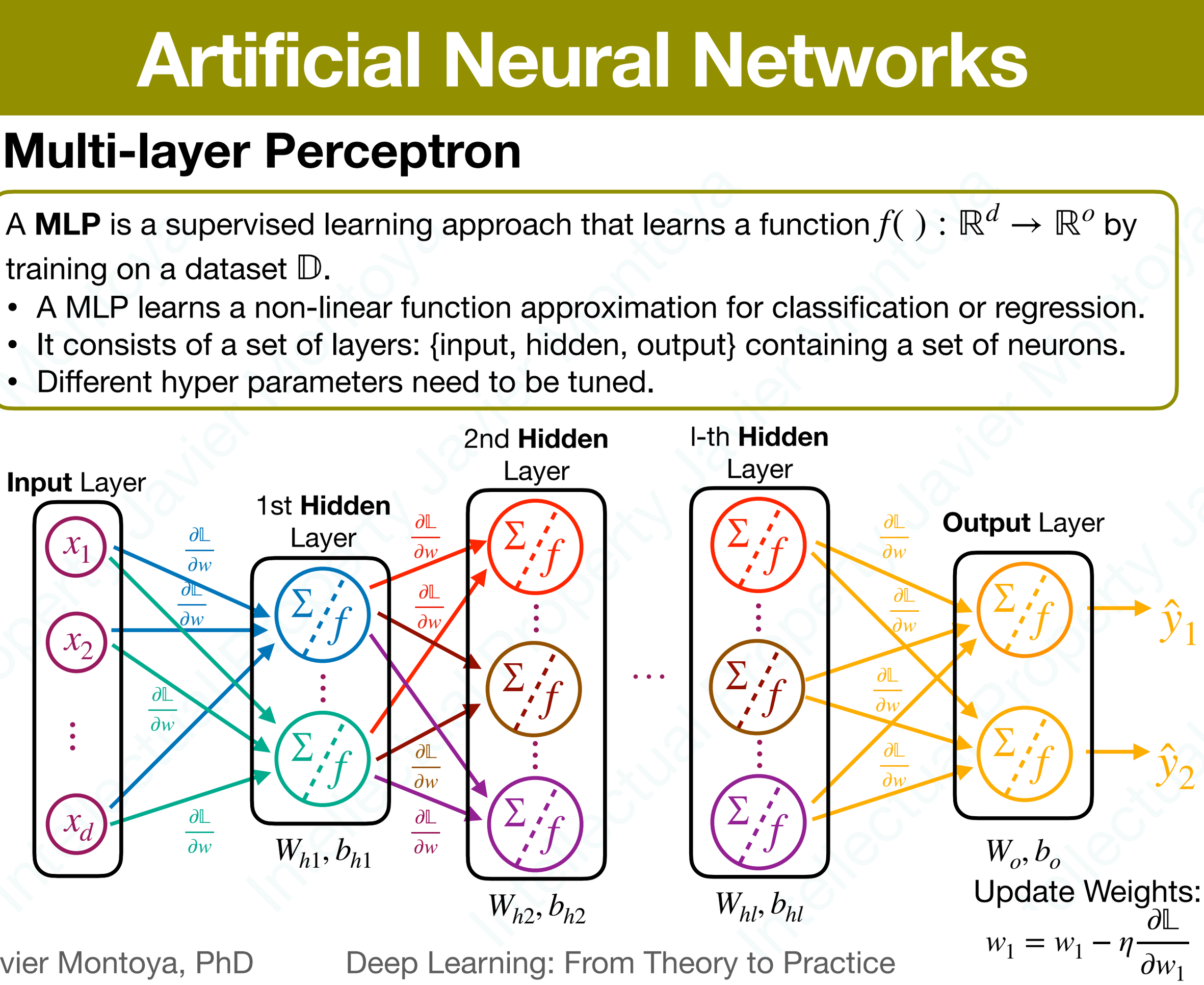


Artificial Neural Networks

Multi-layer Perceptron

A **MLP** is a supervised learning approach that learns a function $f(\ ) : \mathbb{R}^d \rightarrow \mathbb{R}^o$ by training on a dataset $\mathbb{D}$.

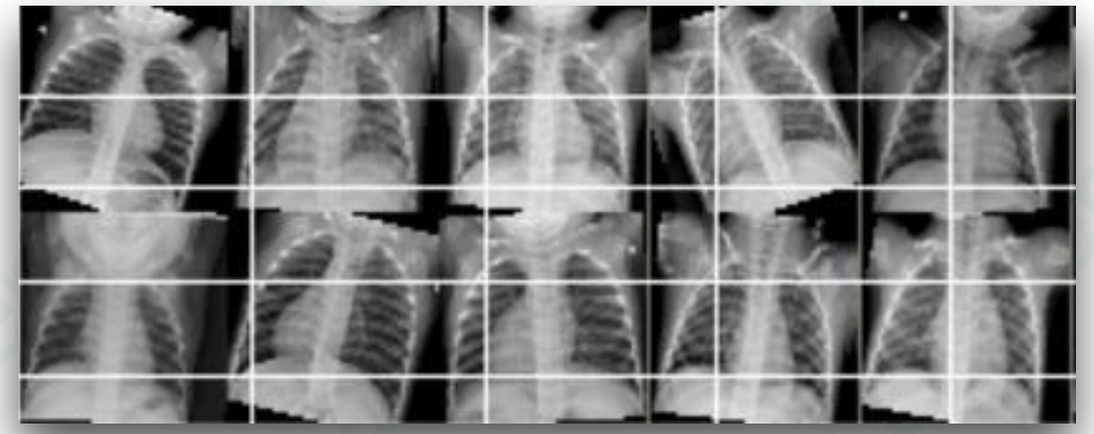
- A MLP learns a non-linear function approximation for classification or regression.
- It consists of a set of layers: {input, hidden, output} containing a set of neurons.
- Different hyper parameters need to be tuned.



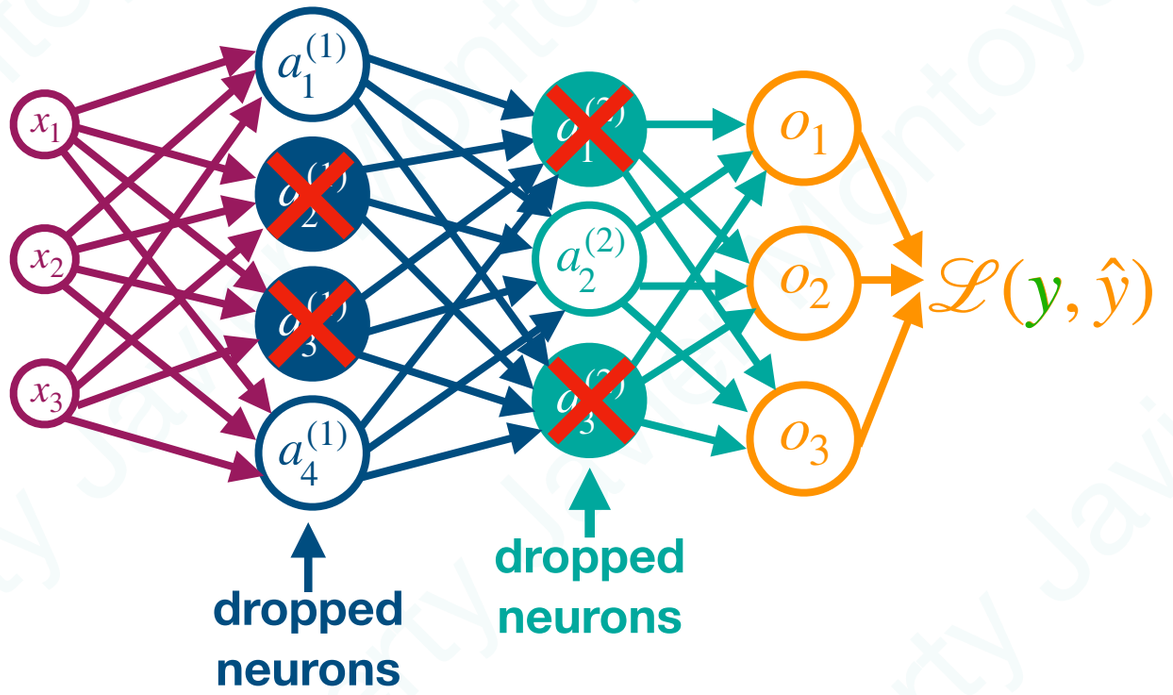
Artificial Neural Networks

Improving model generalisation

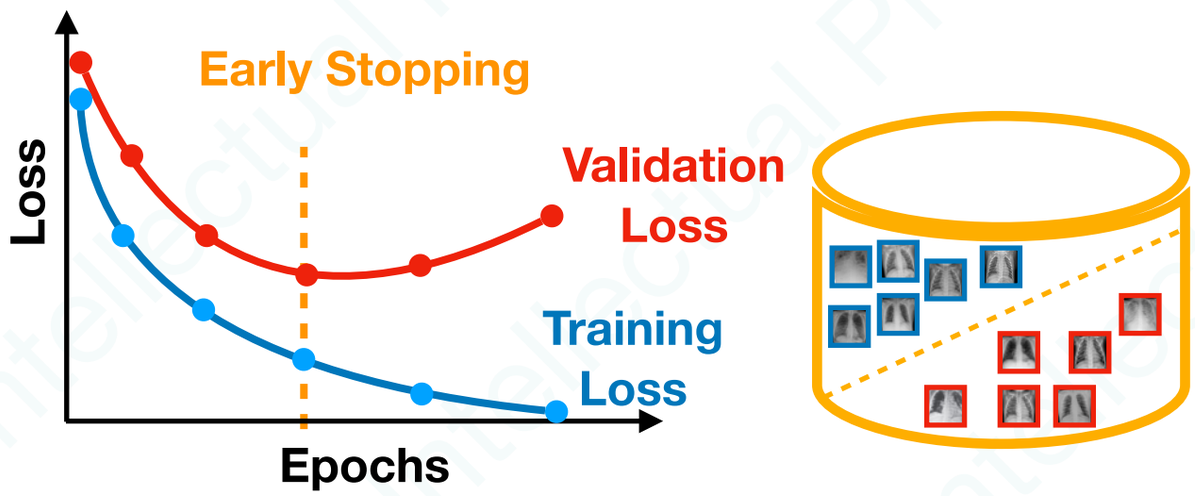
Data Augmentation:



Dropout:



Early Stopping:



Artificial Neural Networks: PyTorch

Class inheritance: nn.Module
General building block for NNs

Class Constructor:
Initialise model parameters

```python
class MultilayerPerceptron(torch.nn.Module):

    def __init__(self, num_features, num_classes, drop_proba,
                 num_hidden_1, num_hidden_2):
        super(MultilayerPerceptron, self).__init__()

        self.my_network = torch.nn.Sequential(
            torch.nn.Linear(num_features, num_hidden_1),
            torch.nn.ReLU(),
            torch.nn.Dropout(drop_proba),
            torch.nn.Linear(num_hidden_1, num_hidden_2),
            torch.nn.ReLU(),
            torch.nn.Dropout(drop_proba),
            torch.nn.Linear(num_hidden_2, num_classes)
        )

    def forward(self, x):
        logits = self.my_network(x)
        probas = F.softmax(logits, dim=1)
        return logits, probas
```

Architecture Definition

Forward Propagation:
The forward() method must be overwritten for any subclass of nn.Module

Artificial Neural Networks: PyTorch

Class inheritance: nn.Module
General building block for NNs

Class Constructor:
Initialise model parameters

```python
class MultilayerPerceptron(torch.nn.Module):

    def __init__(self, num_features, num_classes, drop_p
                 num_hidden_1, num_hidden_2):
        super(MultilayerPerceptron, self).__init__()

        self.my_network = torch.nn.Sequential(
            torch.nn.Linear(num_features, num_hidden_1),
            torch.nn.ReLU(),
            torch.nn.Dropout(drop_proba),
            torch.nn.Linear(num_hidden_1, num_hidden_2),
            torch.nn.ReLU(),
            torch.nn.Dropout(drop_proba),
            torch.nn.Linear(num_hidden_2, num_classes)
        )

    def forward(self, x):
        logits = self.my_network(x)
        probas = F.softmax(logits, dim=1)
        return logits, probas
```

torch.nn.Module

def __init__( ):

def forward( ):

Architecture Definition

Forward Propagation:
The forward() method must be overwritten for any subclass of nn.Module

Artificial Neural Networks: PyTorch


```python
class MultilayerPerceptron(torch.nn.Module):

    def __init__(self, num_features, num_classes, drop_proba,
                 num_hidden_1, num_hidden_2):
        super(MultilayerPerceptron, self).__init__()

        self.my_network = torch.nn.Sequential(
            torch.nn.Linear(num_features, num_hidden_1),
            torch.nn.ReLU(),
            torch.nn.Dropout(drop_proba),
            torch.nn.Linear(num_hidden_1, num_hidden_2),
            torch.nn.ReLU(),
            torch.nn.Dropout(drop_proba),
            torch.nn.Linear(num_hidden_2, num_classes)
        )

    def forward(self, x):
        logits = self.my_network(x)
        probas = F.softmax(logits, dim=1)
        return logits, probas
```

Architecture Definition

Artificial Neural Networks: PyTorch

Training Procedure

```
for epoch in range(EPOCHS):
    print('EPOCH {}:'.format(epoch_number + 1))

    # Make sure gradient tracking is on, and do a pass over the data
    model.train(True)
    avg_loss = train_one_epoch(epoch_number, writer)

    # We don't need gradients on to do reporting
    model.train(False)

    running_vloss = 0.0
    for i, vdata in enumerate(validation_loader):
        vinputs, vlabels = vdata
        voutputs = model(vinputs)
        vloss = loss_fn(voutputs, vlabels)
        running_vloss += vloss

    avg_vloss = running_vloss / (i + 1)
    print('LOSS train {} valid {}'.format(avg_loss, avg_vloss))

    # Log the running loss averaged per batch
    # for both training and validation
    writer.add_scalars('Training vs. Validation Loss',
                    { 'Training' : avg_loss, 'Validation' : avg_vloss },
                    epoch_number + 1)
    writer.flush()

    # Track best performance, and save the model's state
    if avg_vloss < best_vloss:
        best_vloss = avg_vloss
        model_path = 'model_{}_{}'.format(timestamp, epoch_number)
        torch.save(model.state_dict(), model_path)

    epoch_number += 1
```

Training over different epochs

Set the model on training status

Set the model on evaluation phase

Set the model on evaluation status

Evaluate the model on the validation set

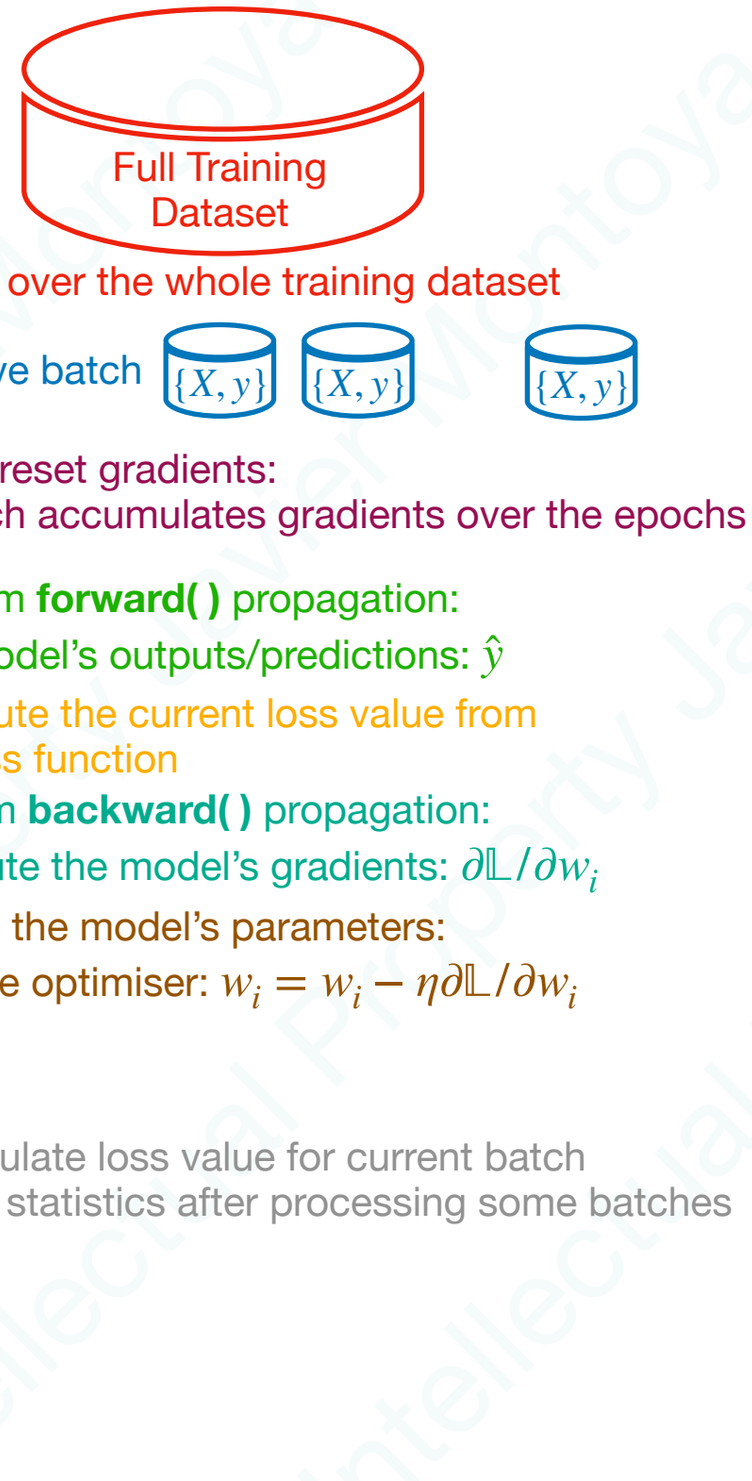
Record and show training and validation statistics after 1x epoch

Track and save best model on the validation set

Artificial Neural Networks: PyTorch

Training Procedure

```python
def train_one_epoch(epoch_index, tb_writer):
    running_loss = 0.
    last_loss = 0.

    # Here, we use enumerate(training_loader) instead of
    # iter(training_loader) so that we can track the batch
    # index and do some intra-epoch reporting
    for i, data in enumerate(training_loader):
        # Every data instance is an input + label pair
        inputs, labels = data

        # Zero your gradients for every batch!
        optimizer.zero_grad()

        # Make predictions for this batch
        outputs = model(inputs)

        # Compute the loss and its gradients
        loss = loss_fn(outputs, labels)
        loss.backward()

        # Adjust learning weights
        optimizer.step()

        # Gather data and report
        running_loss += loss.item()
        if i % 1000 == 999:
            last_loss = running_loss / 1000 # loss per batch
            print('  batch {} loss: {}'.format(i + 1, last_loss))
            tb_x = epoch_index * len(training_loader) + i + 1
            tb_writer.add_scalar('Loss/train', last_loss, tb_x)
            running_loss = 0.

    return last_loss
```



Deep Learning: From Theory to Practice

Javier Montoya, Dr. Sc. ETH
javier.montoya@hslu.ch

(Responsible AI) Entrepreneurship & Innovation, Rotkreuz, 13th March 2023