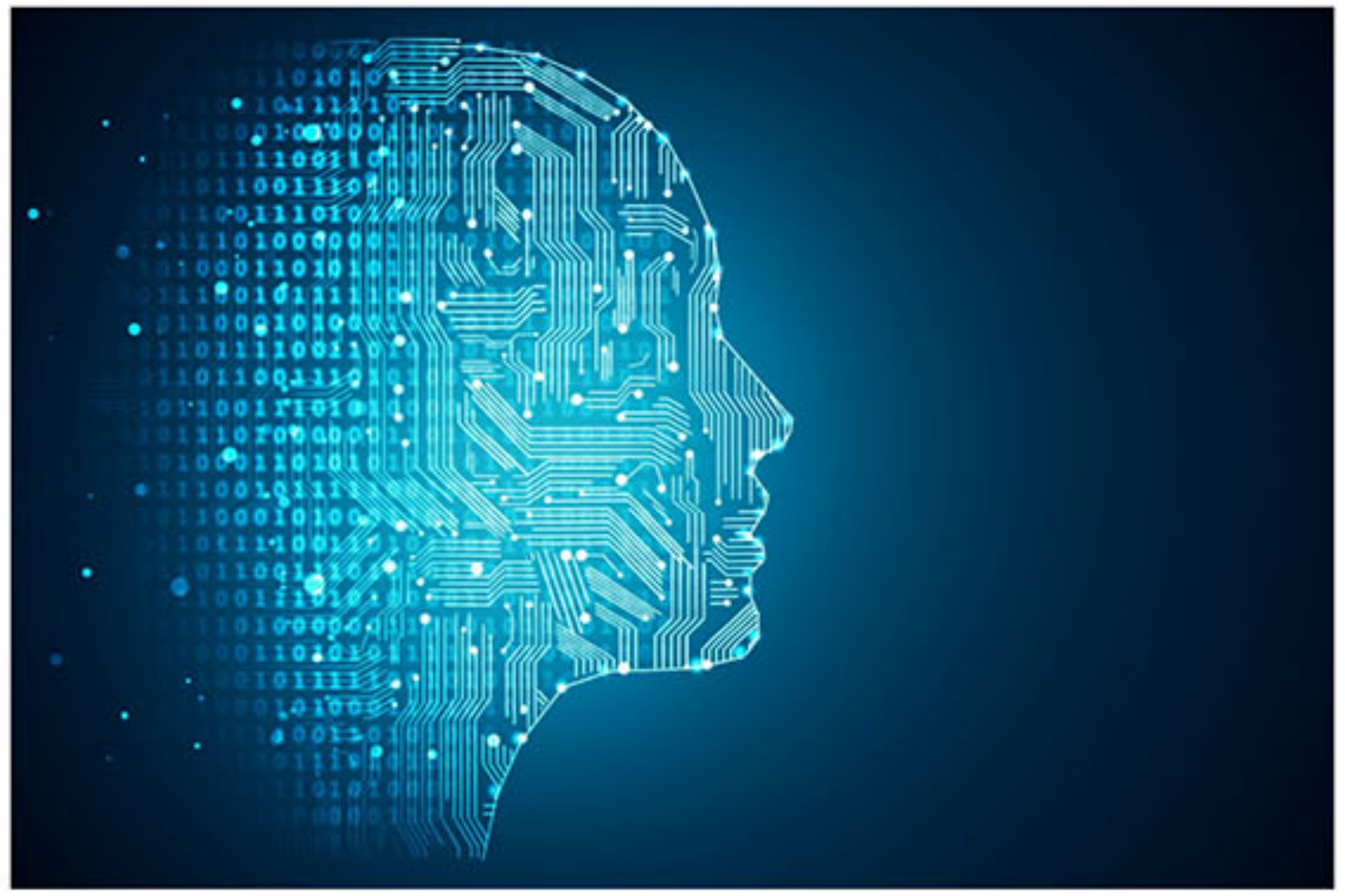
PRINCETON UNIVERSITY

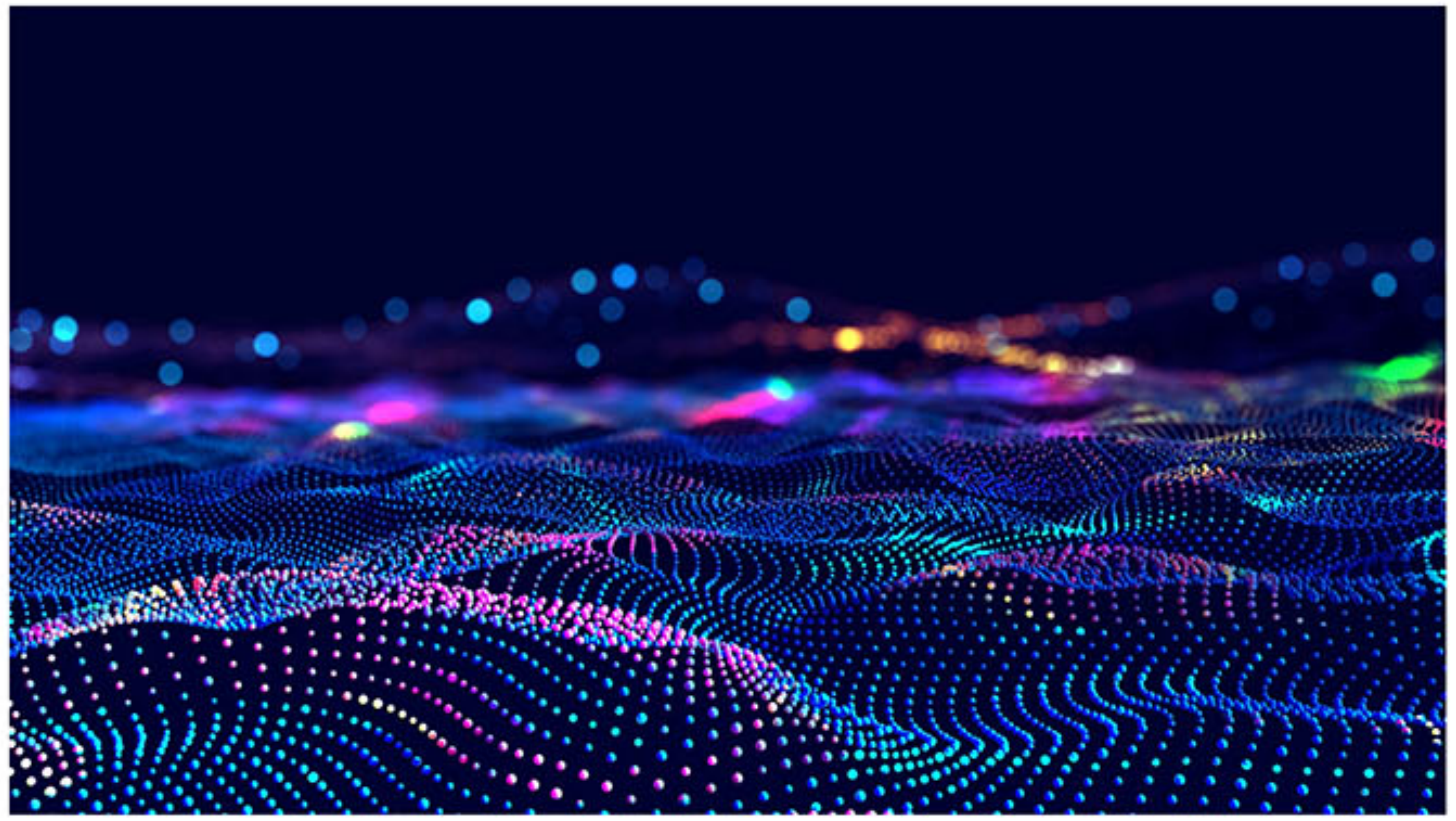
CENTER FOR STATISTICS AND MACHINE LEARNING

A Hands-On Introduction to Machine Learning



Wintersession 2025
January 15–17,21

Julian Gold
Gage DeZoort



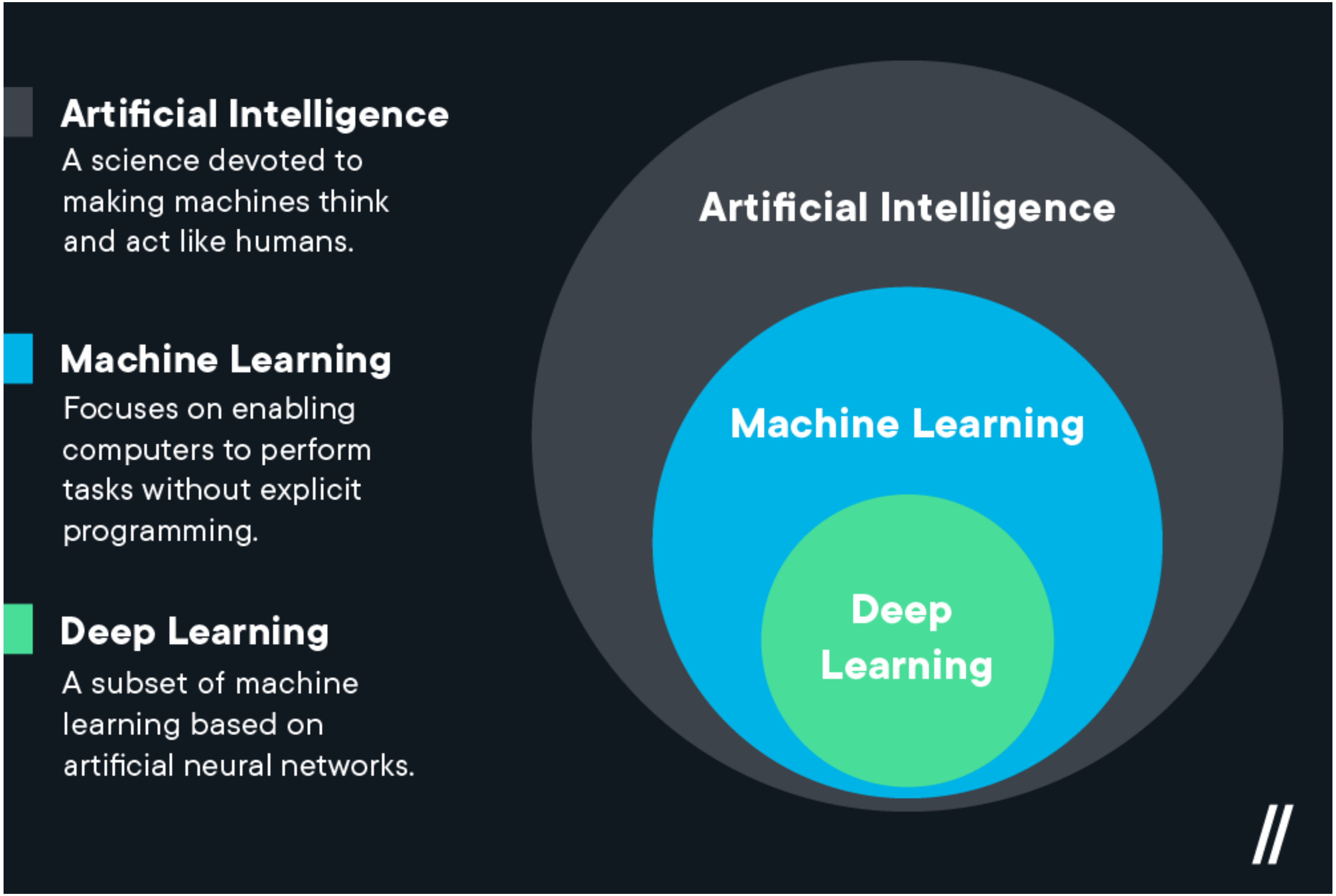
With materials from:

Brian Arnold, Gage DeZoort, Julian Gold, Jonathan Halverson, Christina Peters, Savannah Thias, Amy Winecoff

DataX
Accelerating Scientific Discovery at Princeton

Mini-Course Outline

Date	Topic	Instructor
Wed. 1/15	Machine Learning Overview and Simple Models	Julian Gold
Thu. 1/16	Model Evaluation and Improving Performance	Julian Gold
Fri. 1/17	Introduction to Neural Networks	Gage DeZoort
Tue. 1/21	Survey of Neural Network Architectures	Gage DeZoort
Wed.+Thu. 1/22-1/23	Getting Started with LLMs with PLI	Simon Park, Abhishek Panigrahi
Wed. 1/22	Graph Neural Networks for Your Research	Gage DeZoort
Wed. 1/22	Machine Learning for the Physical Sciences	C. Jespersen, R. Pastrana, Q. Gallagher, H. Johnson

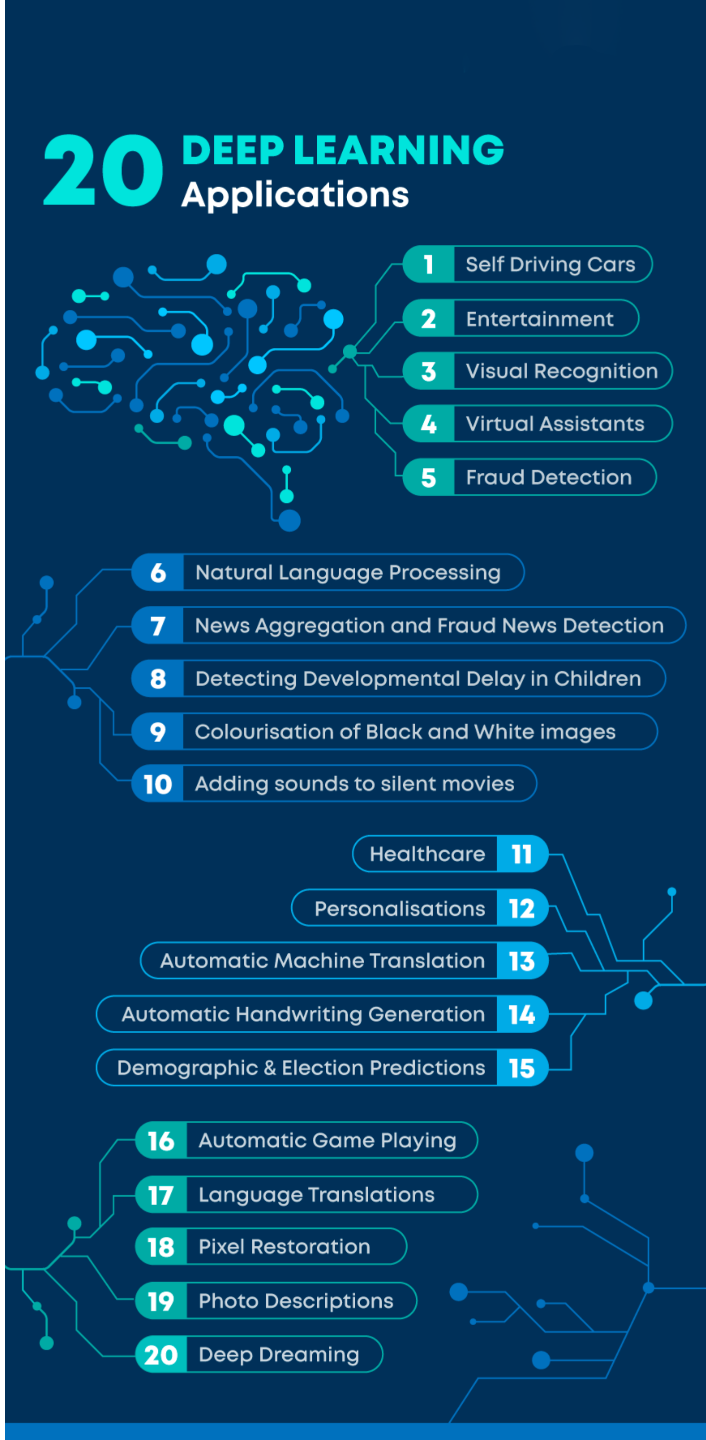


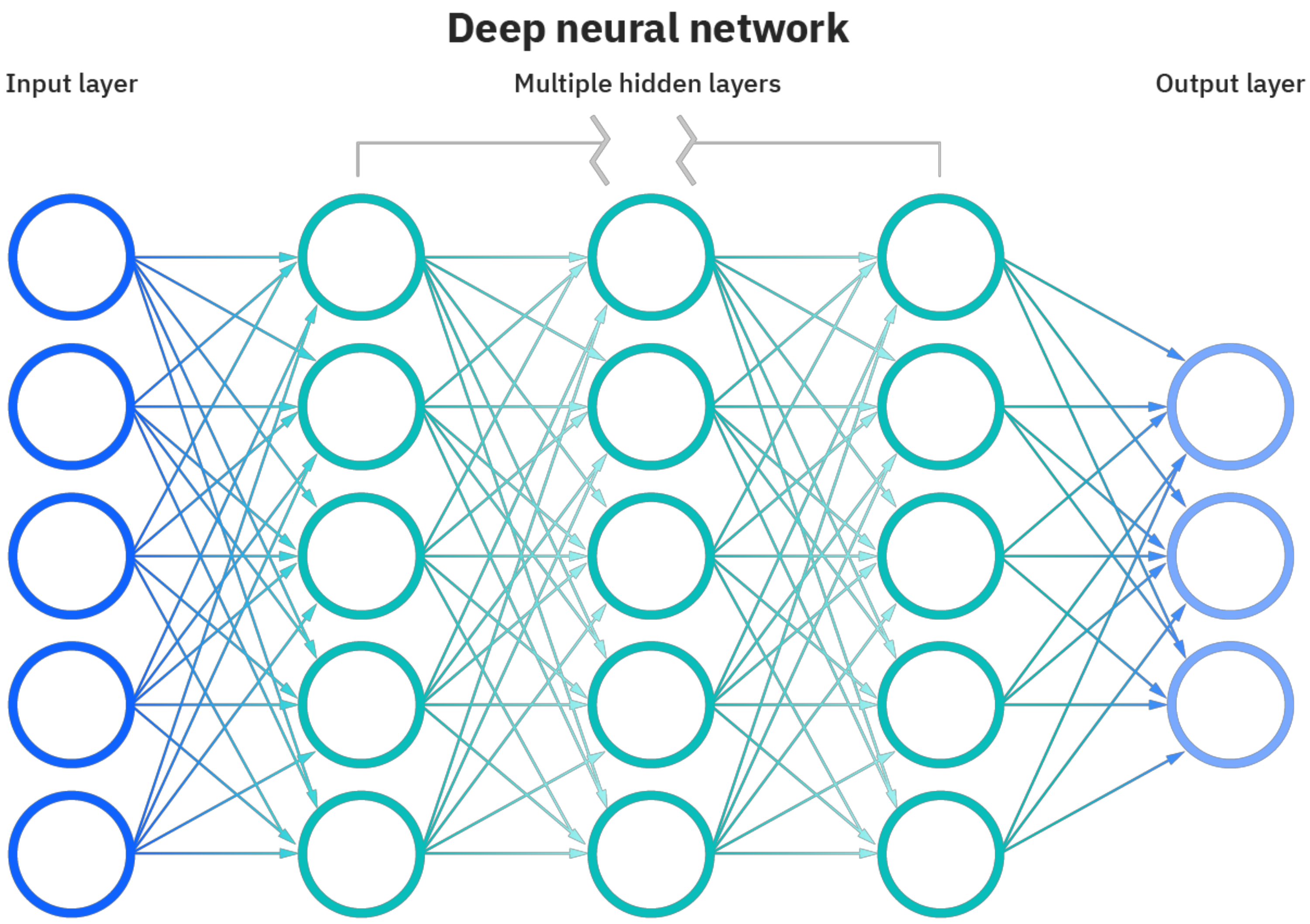
[Deep Learning vs. Machine Learning — What's the Difference? | Flatiron School]

WHY USE DL?

- Data with complex (highly non-linear) relationships
- Big data – many examples to leverage
- High-dimensional data
- Data with complicated structure (images, video, language, social networks etc.)

[20 Deep Learning Applications in 2022 Across Industries | Great Learning (mygreatlearning.com)]

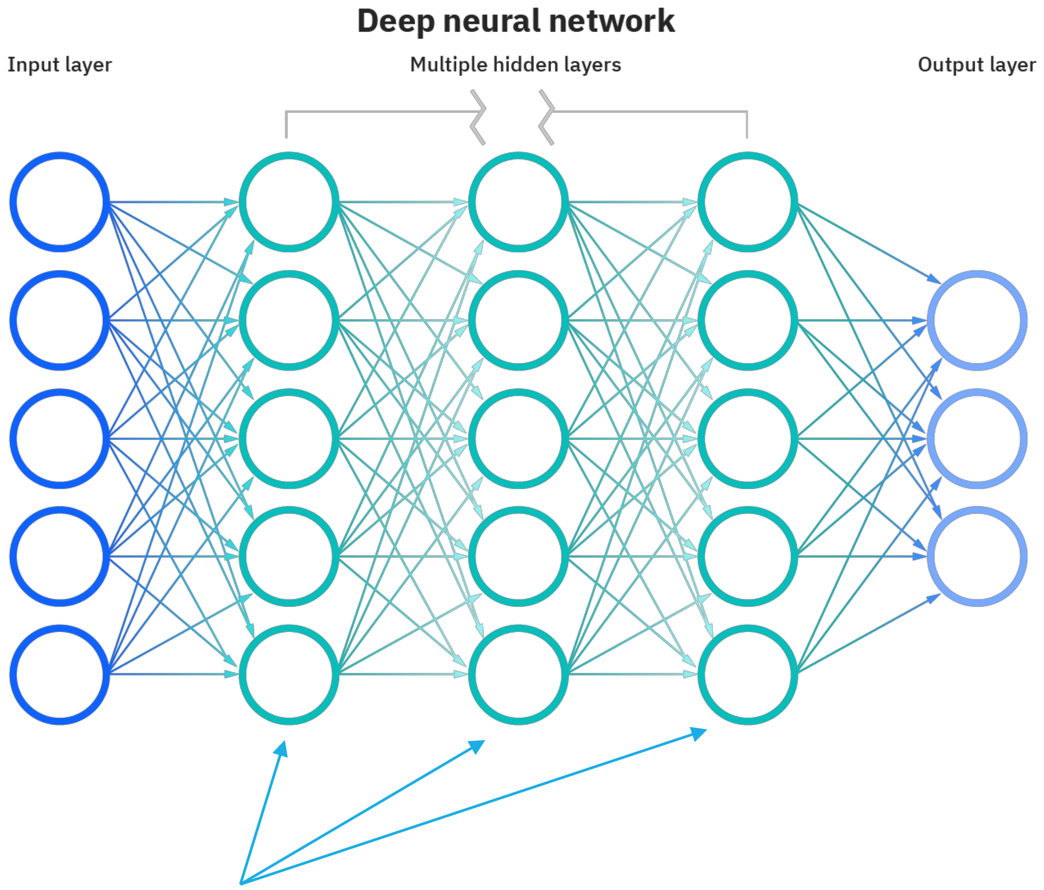




[What are Neural Networks? - United Kingdom | IBM]

DEEP NEURAL NETWORKS

- A class of ML algorithms based on *artificial neural networks* (ANNs)
  - Neural network → networks of neurons responding to stimuli
  - “Deep” → multiple layers of neurons interacting in sequence



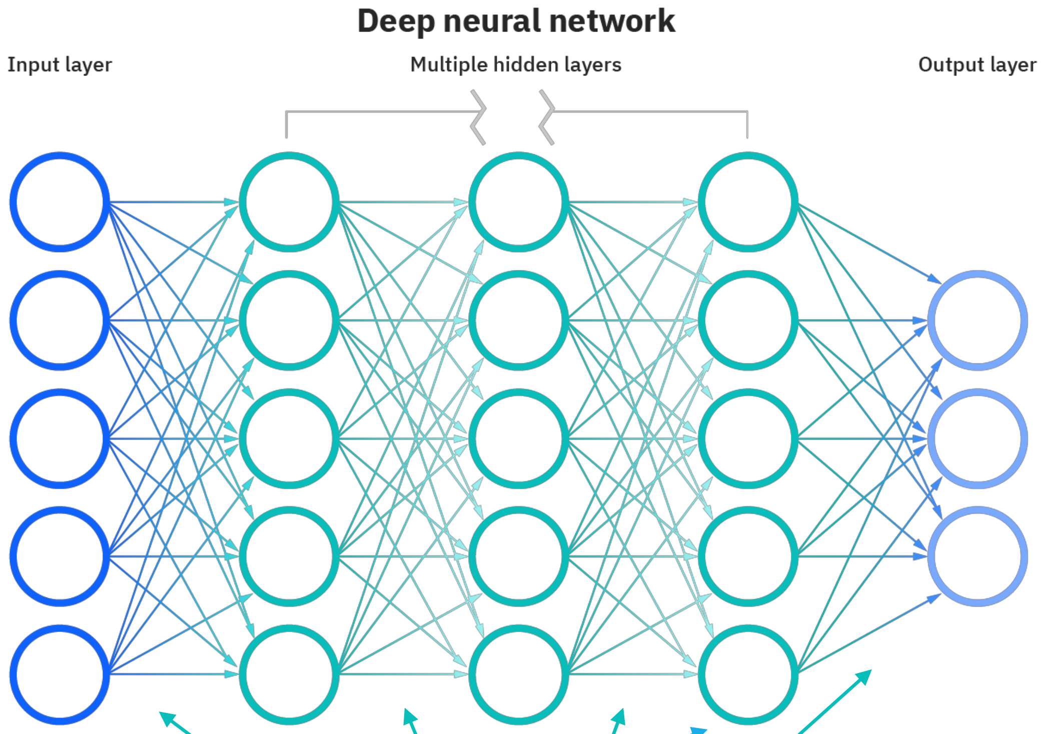
Hidden layers of *neurons*, *hidden units*, or simply *nodes*

DEEP NEURAL NETWORKS

- A class of ML algorithms based on *artificial neural networks* (ANNs)
  - Neural network → networks of neurons responding to stimuli
  - "Deep" → multiple layers of neurons interacting in sequence
- Practically speaking, DNNs are non-linear models designed to leverage complicated relationships in data

$$f(x \mid parameters) = output$$

Adjustable, e.g. fit to data in supervised learning



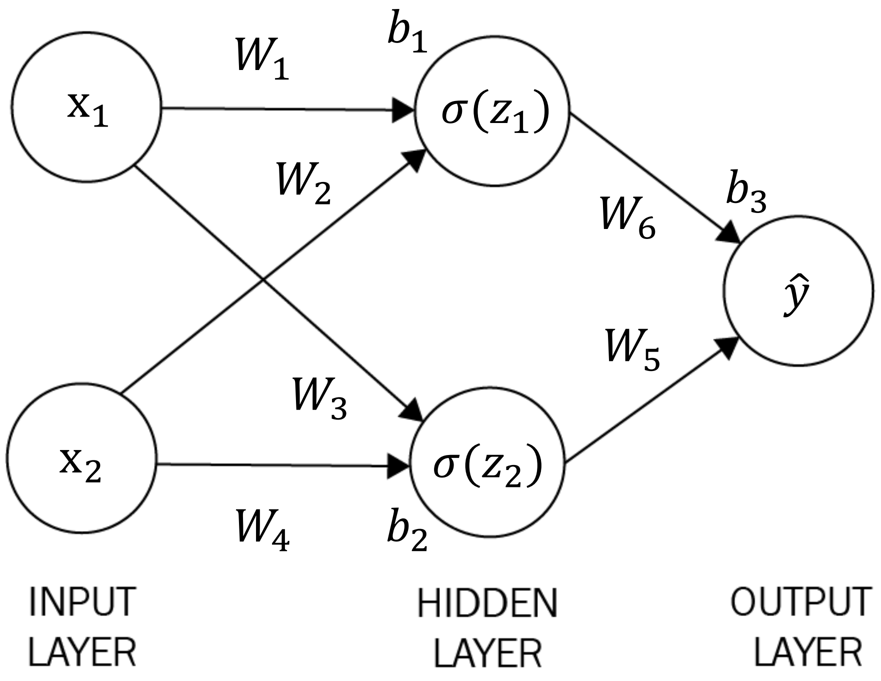
Hidden layers of *neurons*, *hidden units*, or simply *nodes*

Learned (trainable!) *parameters* encode neural communications

SIMPLE NN

- **Example:** Classification problem with
  $\mathrm{x_i} \in \mathbb{R}^2 \qquad y_i \in \{0,1\}$
- Let's draw a test data point and pass it through a simple ANN:

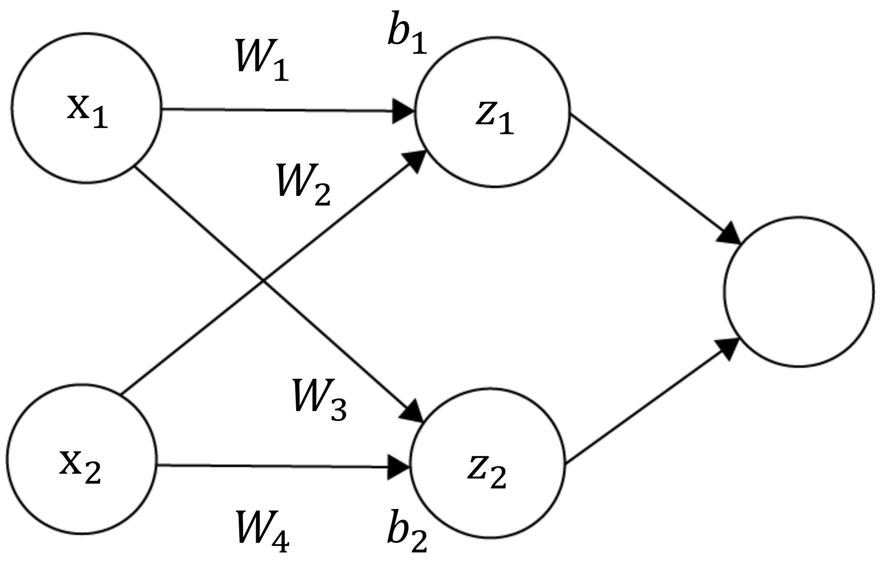
$$x = (x_1, x_2) \qquad y = 1$$



- Single hidden layer with 2 neurons
- **Key Ingredients:**
  - *Trainable weights* $W_1, W_2, W_3, W_4, W_5, W_6$ and *biases* b_1, b_2, b_3
  - Non-linear *activation functions* (called *non-linearities*) $\sigma(z)$

[The simplest artificial neural network - Practical Convolutional Neural Networks [Book] (oreilly.com)](https://www.oreilly.com)

SIMPLE NN



1. Compute *preactivations* at each neuron

$$z_1 = w_1 x_1 + w_2 x_2 + b_1$$
$$z_2 = w_3 x_1 + w_4 x_2 + b_2$$

Or, in matrix notation:

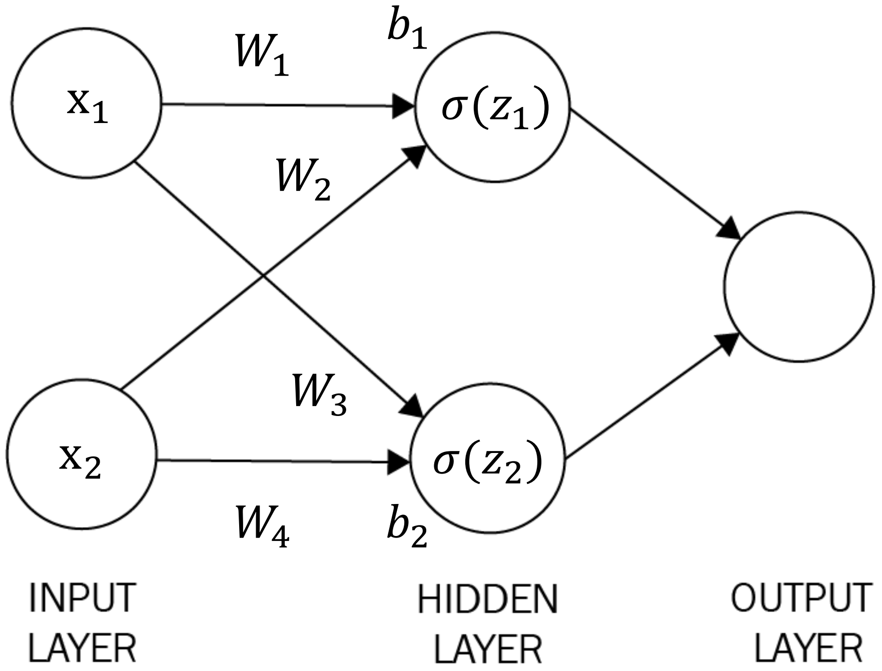
$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

the *weight matrix* mixes up the inputs and feeds them to the each neuron

the *bias* vector adds constants to the mixed inputs

Note: this is a linear operation!!

SIMPLE NN



1. Compute *preactivations* at each neuron

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

2. Calculate how much the neuron *activates* given the strength of the *pre-activation*

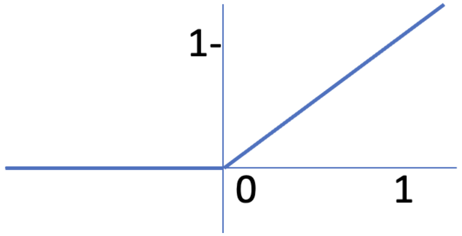
$\sigma(z) \rightarrow$ activation function

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \rightarrow \begin{pmatrix} \sigma(z_1) \\ \sigma(z_2) \end{pmatrix}$$

NON-LINEAR ACTIVATION FUNCTIONS

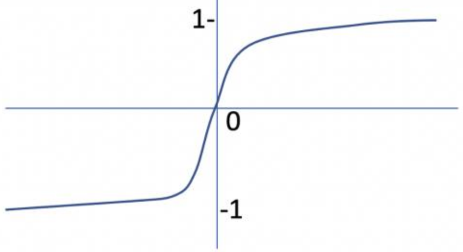
- Non-linear activation functions allow us to learn complex relationships by "intervening" between linear operations
- In practice, they allow neurons to switch "on" and "off" to varying degrees

Rectified Linear Unit (ReLU)

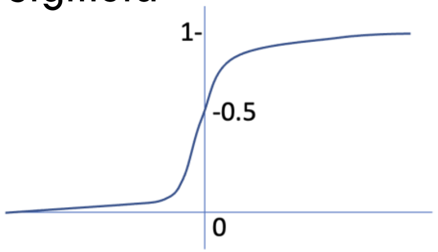


← Most popular choice; simple to compute (fast training), no "saturating" regions with tiny gradients

Hyperbolic Tangent

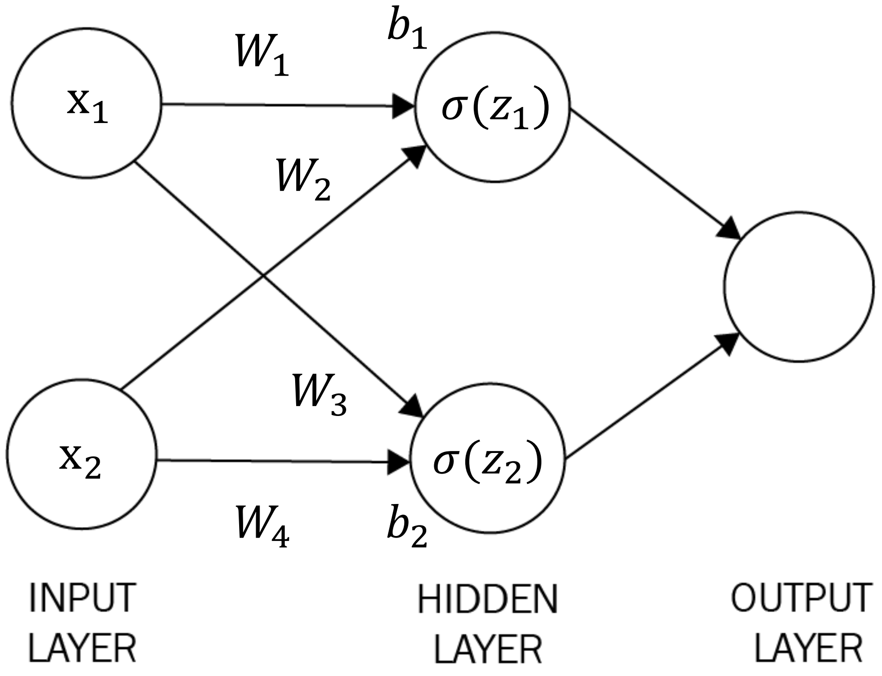


Sigmoid



The Importance and Reasoning behind Activation Functions | by Zack Brodtman | Towards Data Science

SIMPLE NN



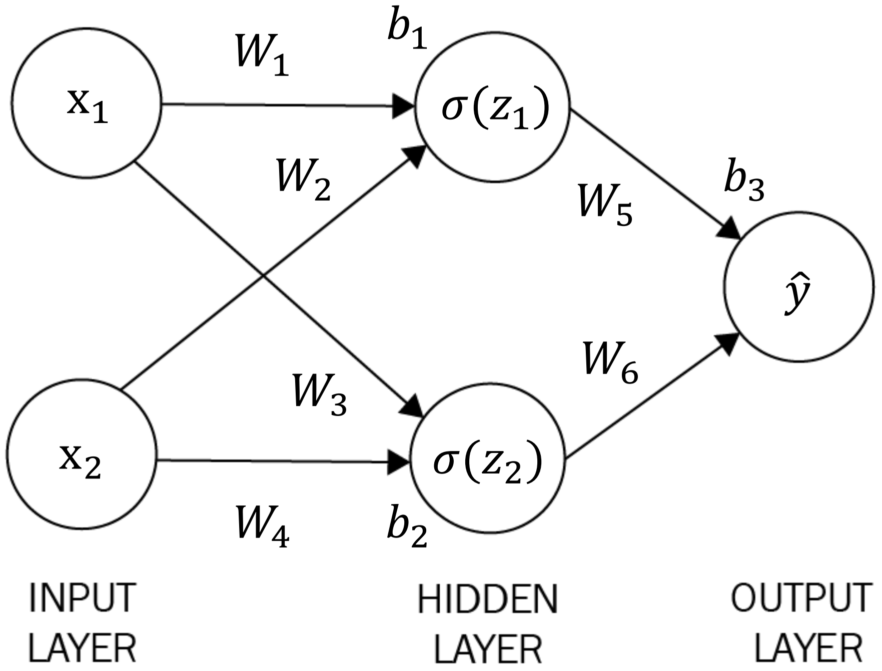
1. Compute *preactivations* at each neuron

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

2. Calculate how much the neuron *activates* given the strength of the *pre-activation*

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \to \begin{pmatrix} \sigma(z_1) \\ \sigma(z_2) \end{pmatrix}$$

HOW IT WORKS



1. Compute *preactivations* at each neuron

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

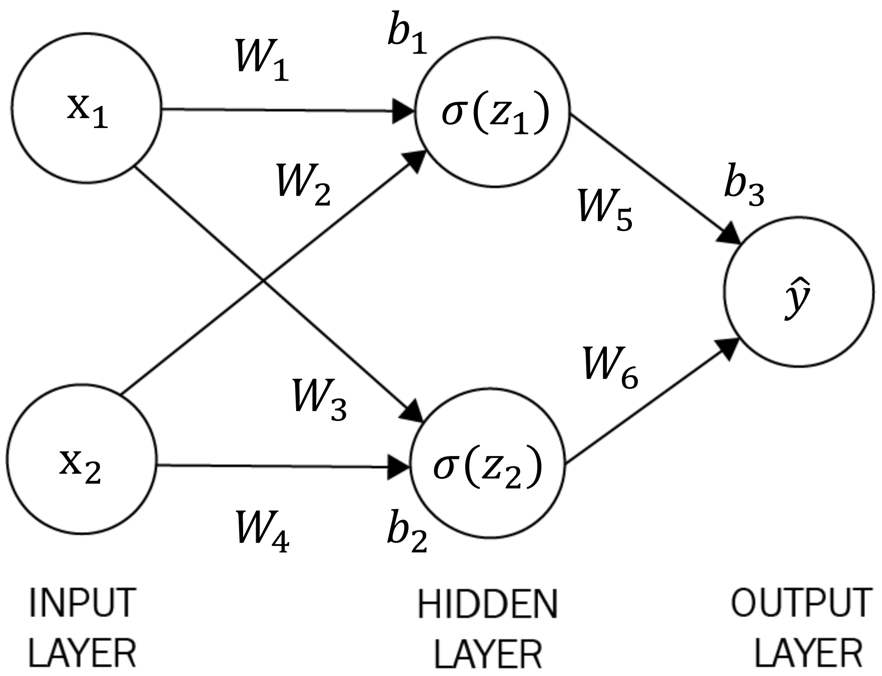
2. Calculate how much the neuron *activates* given the strength of the *pre-activation*

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \to \begin{pmatrix} \sigma(z_1) \\ \sigma(z_2) \end{pmatrix}$$

3. Calculate model outputs

$$\hat{y} = \begin{pmatrix} w_5 & w_6 \end{pmatrix} \begin{pmatrix} \sigma(z_1) \\ \sigma(z_2) \end{pmatrix} + b_3$$

HOW IT WORKS



Forward Pass → Predictions

1. Compute *preactivations* at each neuron

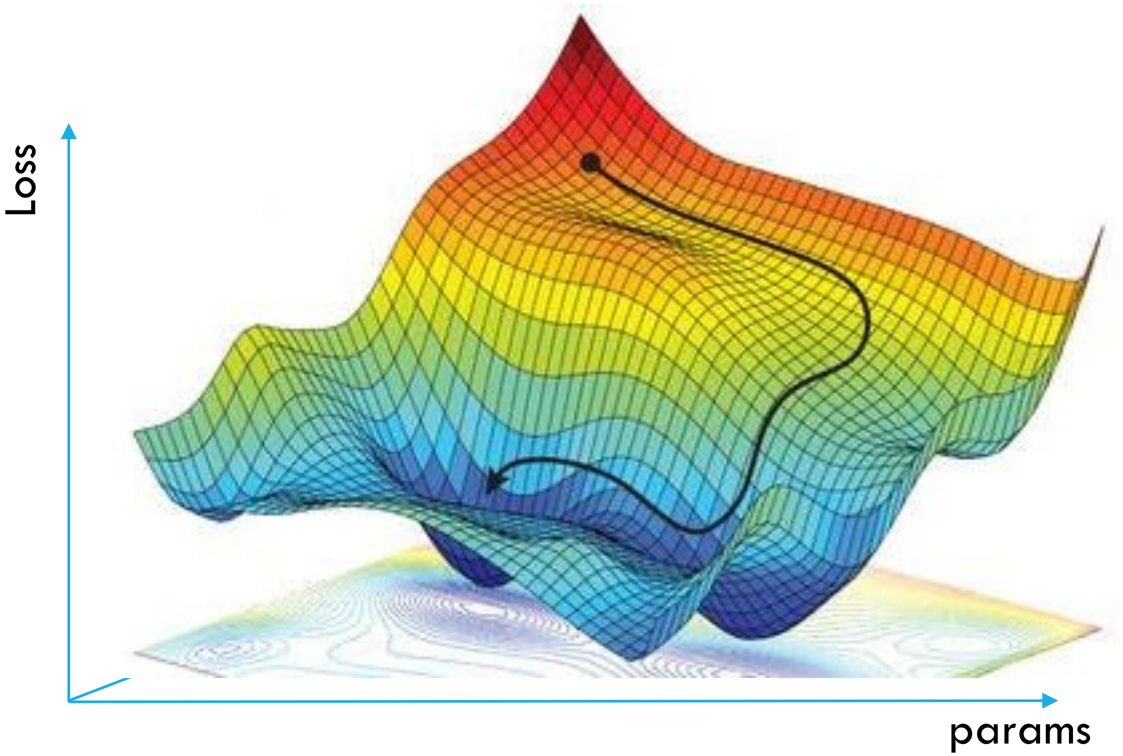
2. Calculate how much the neuron *activates* given the strength of the *pre-activation*
(Repeat 1 and 2 to some fixed depth)

3. Calculate model outputs

When we define an NN, we fix an *architecture* by specifying:

- Number of hidden layers (here 1)
- Dimension of the hidden layers (here 2)
- Activation functions
- Random initial values for weights and biases

TRAINING A NN



The loss function may be very complicated in practice!

Training a NN → find "optimal" weights, biases

- Start by defining a **loss function** $L(y, \hat{y})$
- Compute the gradient of $L(y, \hat{y})$ with respect to each weight and bias
- Update the weights and biases via **gradient descent:**

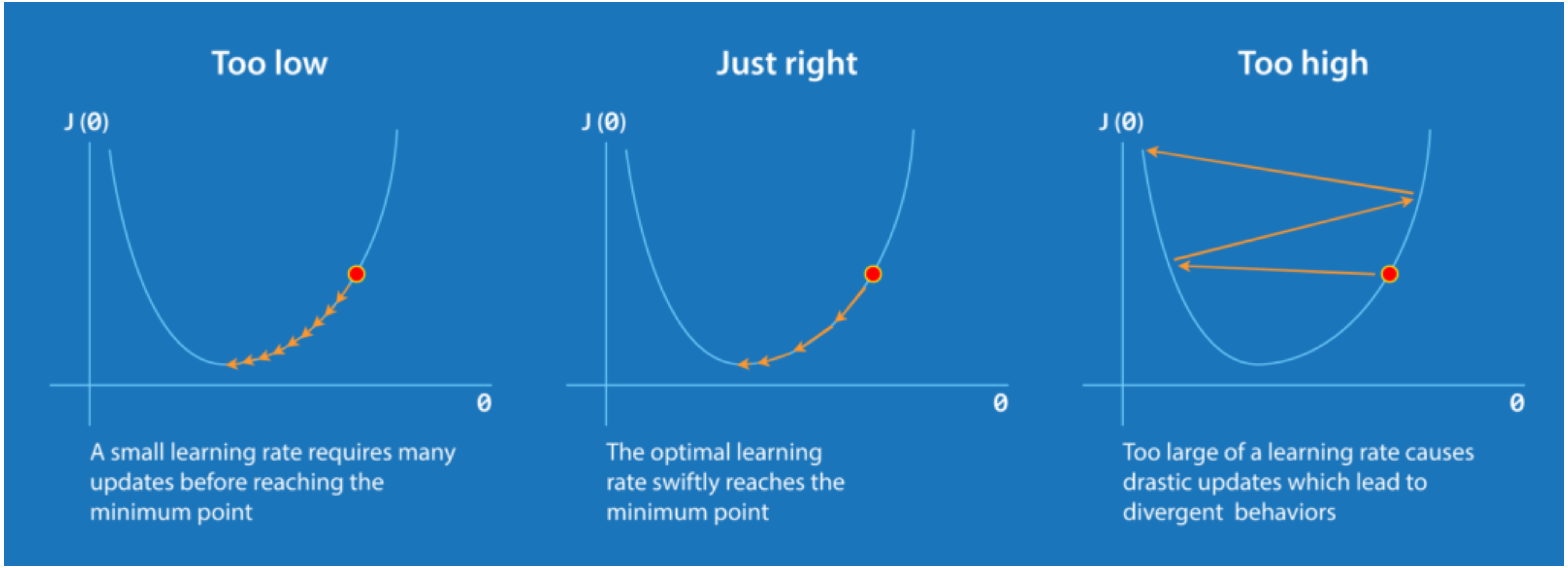
$$W_1^{(k+1)} = W_1^{(k)} - \gamma \frac{\partial L}{\partial W_1}\bigg|_{W_1^{(k)}}$$

$$b_1^{(k+1)} = b_1^{(k)} - \gamma \frac{\partial L}{\partial b_1}\bigg|_{b_1^{(k)}}$$

etc.

γ → learning rate

LEARNING RATES



[What is Gradient Descent? Gradient Descent in Machine Learning (mygreatlearning.com)]

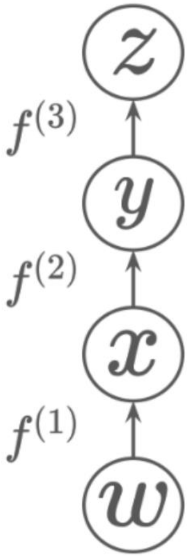
TRAINING A NN

What we want: $\frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}$

Generically, “gradients”

Backward Pass (Backpropagation)

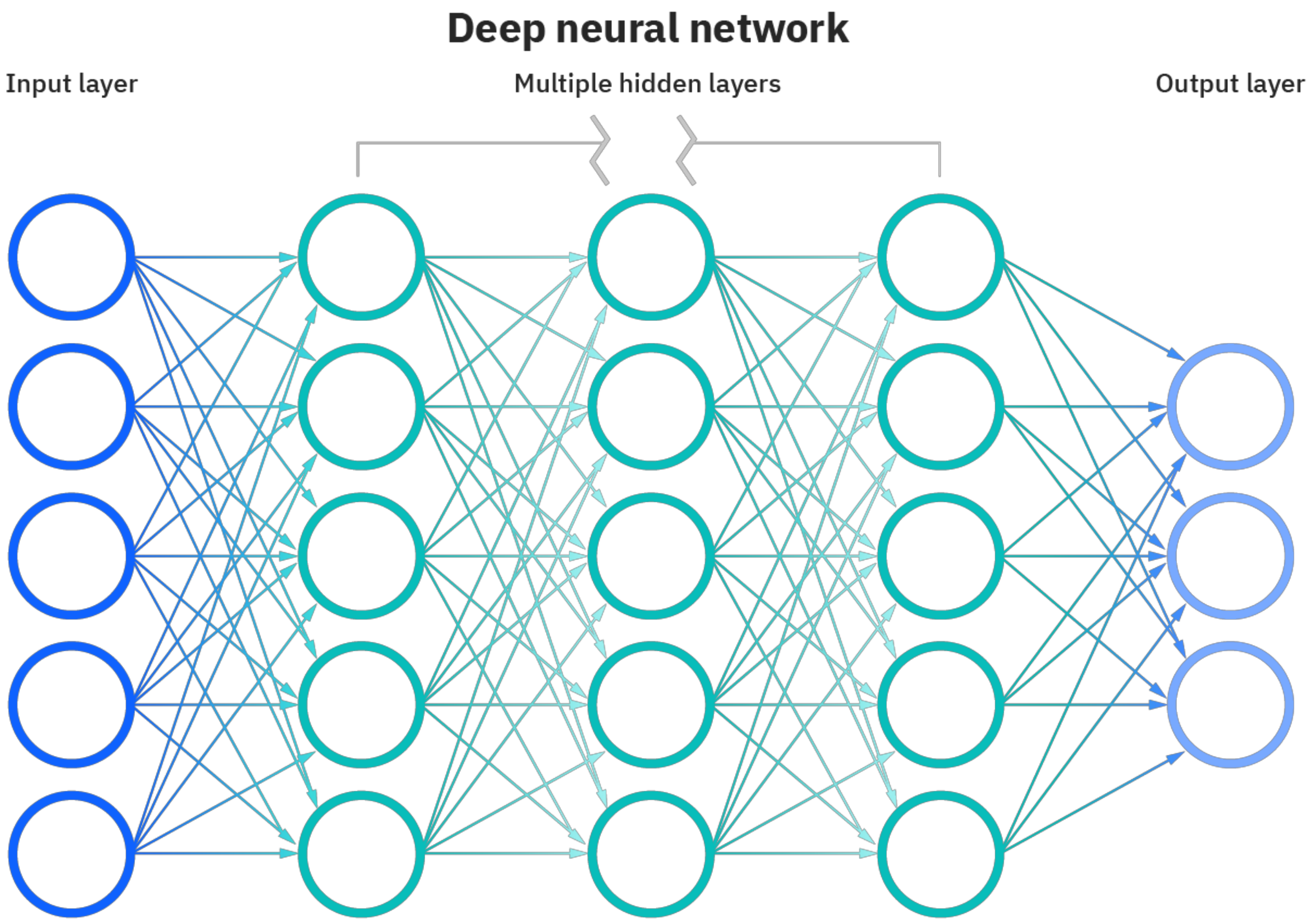
Apply the chain rule to calculate derivatives of the loss with respect to the weights/biases



$$\frac{\partial z}{\partial w}$$
$$= \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}$$
$$= f^{(3)\prime}(y) f^{(2)\prime}(x) f^{(1)\prime}(w)$$

PSEUDOCODE

```
for each training epoch:
  for each (input, truth) in train_data:
      prediction = NN(train_data)
      loss = loss_function(prediction, truth)
      gradients = compute_gradients(loss)
      new_params = grad_descent(gradients, NN.parameters, lr)
      model.update(new_params)
```



[What are Neural Networks? - United Kingdom | IBM]

SHOULD YOU USE DL FOR YOUR PROJECT?

Why DL?

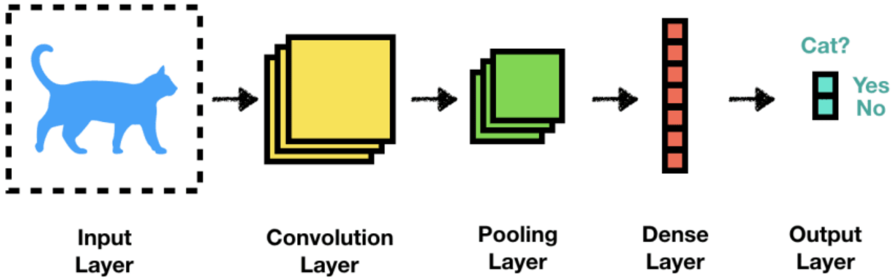
- Produce predictions in multiple feed-forward stages → powerful feature extraction:
  - Less need for data pre-processing
  - Ability to leverage large amounts of data ("big data")
- Handle more complicated data representations like images, sentences, and graphs
- Can be designed to perform complicated (multi-stage) tasks end-to-end

Why not DL?

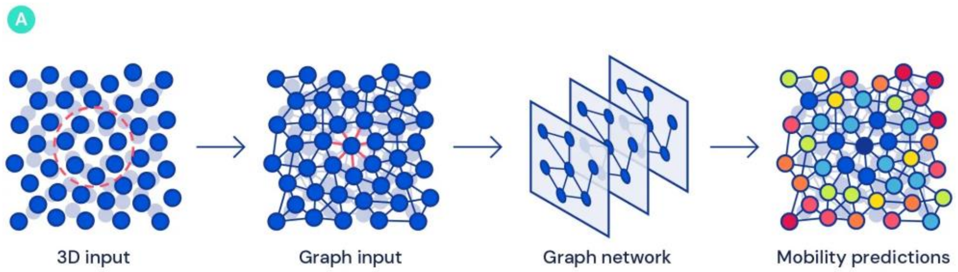
- Developing and training large DL models is computationally expensive (slow, resource intensive) and often requires specialized computing hardware
- It usually takes a lot of data to train DL models

MORE ARCHITECTURES

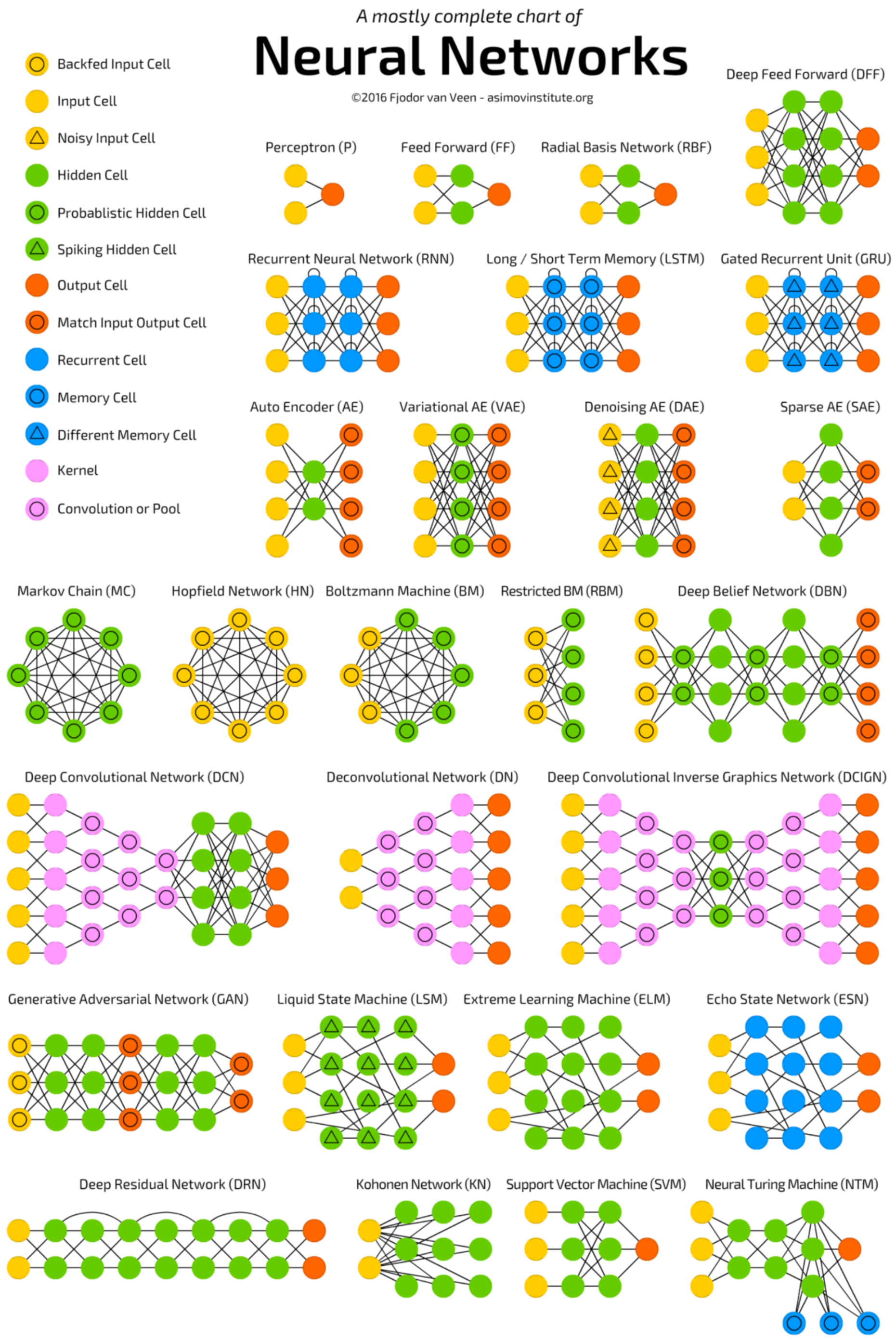
- NNs are the building blocks for more complicated architectures, e.g.
  - **Convolutional Neural Networks** are frequently applied to images or other grid data
  - **Recurrent Neural Networks** are applied to sequences like sentences
  - **Graph Neural Networks** operate on networks of objects (nodes) connected by their relationships (edges)
  - **Generative Adversarial Networks** are used to generate new data (e.g. photographs) similar to a reference set



Convolutional Neural Network: A Step By Step Guide | by Shashikant | Towards Data Science



Towards understanding glasses with graph neural networks (deepmind.com)



A bit outdated, but fun to see the creativity…

[The mostly complete chart of Neural Networks, explained | by Andrew Tch | Towards Data Science]

Time for some NN practice!

Please navigate to Day 3!

PrincetonUniversity/intro_machine_learning (github.com)