Title: Machine Learning Lecture - 230302

Speakers: Lauren Hayward

Collection: Machine Learning for Many-Body Physics (2022/2023)

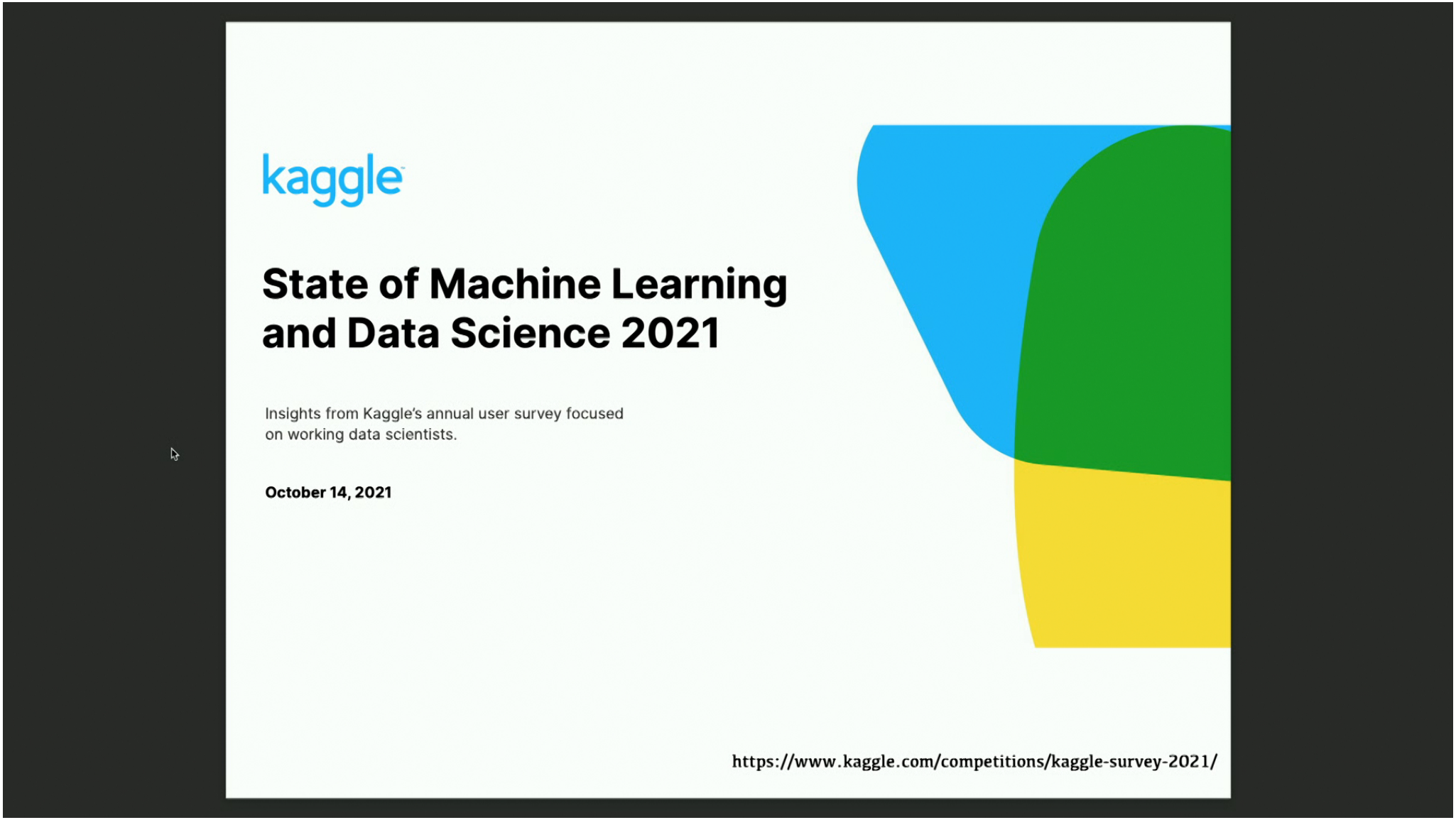
Date: March 02, 2023 - 9:00 AM

URL: https://pirsa.org/23030029

Machine Learning for Many-Body Physics

Lecture 3

PERIMETER
SCHOLARS
INTERNATIONAL


kaggle
State of Machine Learning and Data Science 2021
Insights from Kaggle's annual user survey focused on working data scientists.
October 14, 2021
https://www.kaggle.com/competitions/kaggle-survey-2021/

This is our 5th year conducting an in-depth user survey & publicly sharing the results.

Over 25,000 data scientists and ML engineers submitted responses on their backgrounds and day to day experience – everything from educational details to salaries to preferred technologies and techniques.

https://www.kaggle.com/competitions/kaggle-survey-2021/

- **Respondents were asked about most commonly used algorithms**

- **Options included:**
  - Linear or logistic regression
  - Decision trees or random forests
  - Gradient boosting machines
  - Convolutional neural networks
  - Bayesian approaches
  - Dense neural networks
  - Recurrent neural networks
  - Transformer networks
  - Generative adversarial networks
  - Evolutionary approaches

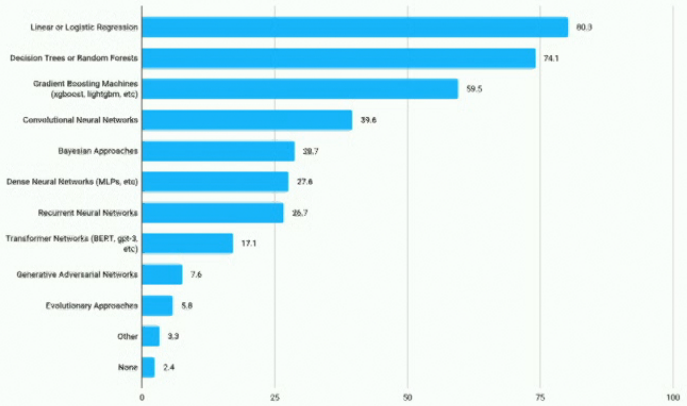
https://www.kaggle.com/competitions/kaggle-survey-2021/

Methods & Algorithms

Like last year, the most commonly used algorithms were linear and logistic regression, followed closely by decision trees and random forests.

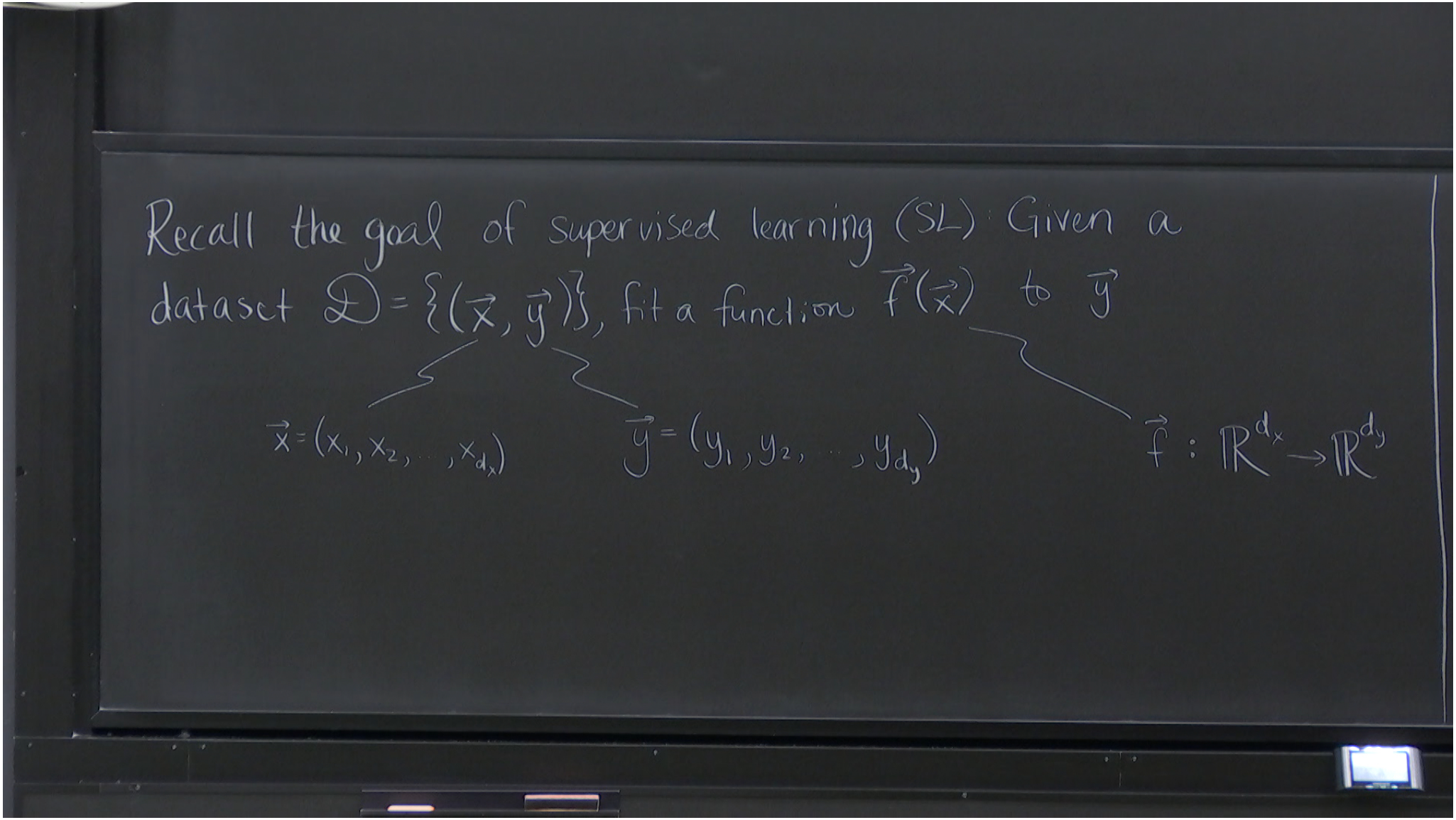
Of more complex methods, gradient boosting machines and convolutional neural networks were the most popular approaches.

Methods and Algorithms Usage



Kaggle | State of ML & Data Science 2021

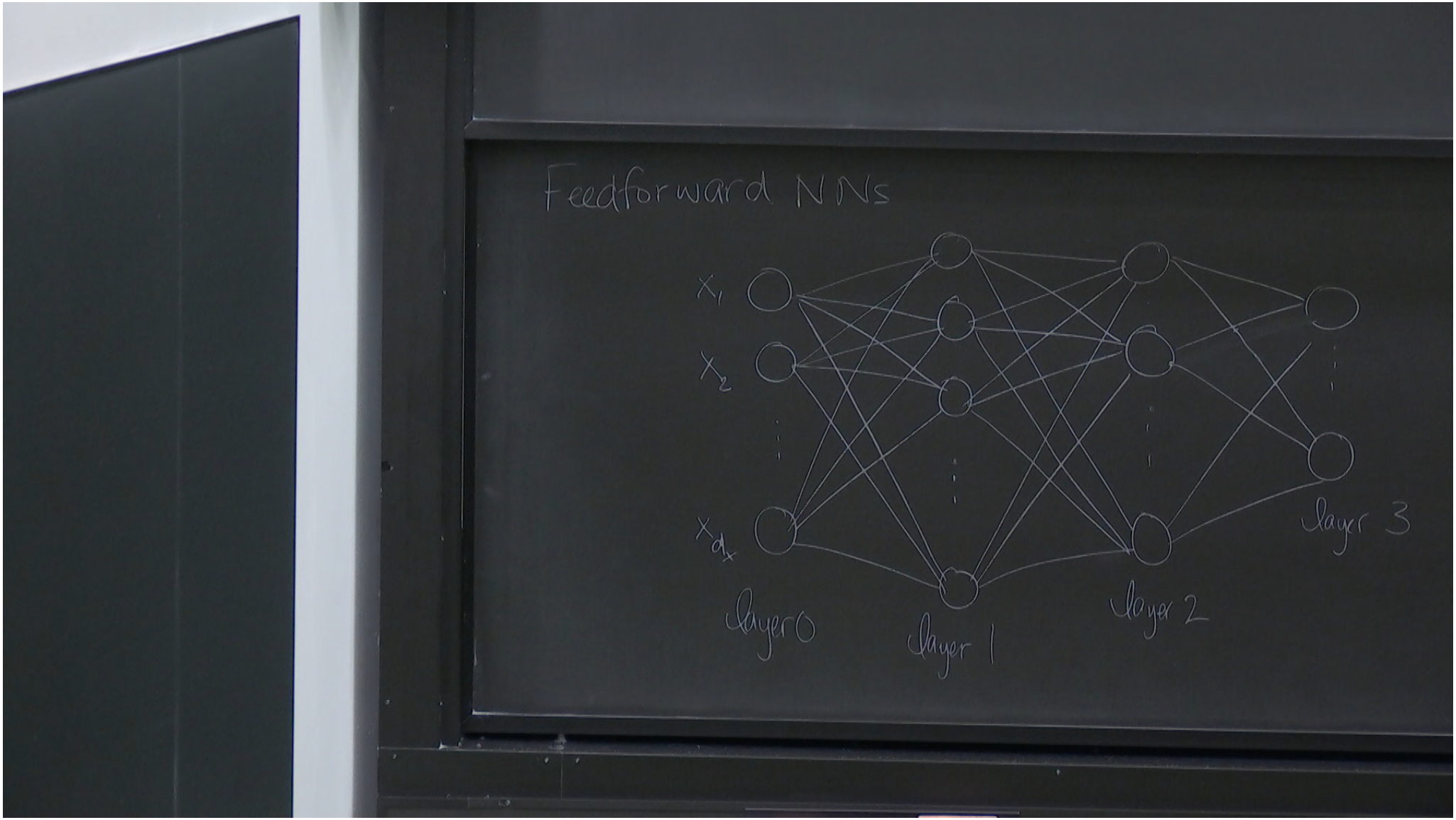
https://www.kaggle.com/competitions/kaggle-survey-2021/


Recall the goal of supervised learning (SL): Given a dataset $\mathcal{D} = \{(\vec{x}, \vec{y})\}$, fit a function $\vec{f}(\vec{x})$ to $\vec{y}$
$\vec{x} = (x_1, x_2, \cdots, x_{d_x})$
$\vec{y} = (y_1, y_2, \cdots, y_{d_y})$
$\vec{f} : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$

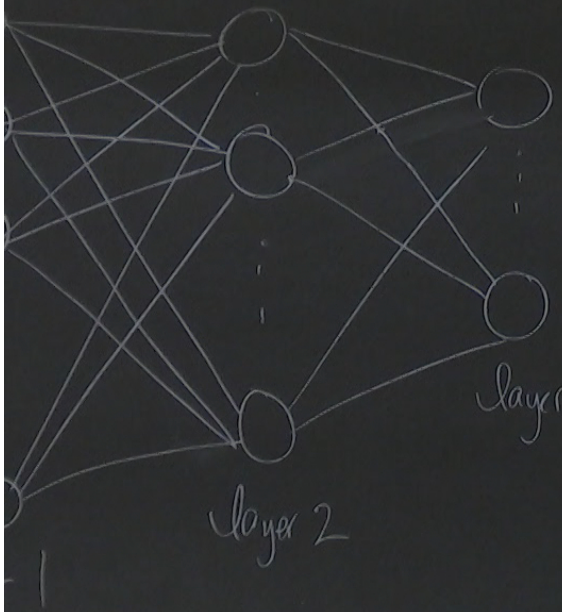
$: \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$

Outline for today: intro to SL with feedforward neural networks (NNs)

- Network architecture and propagation of information
  - ↳ weights
  - ↳ biases
  - ↳ activation functions
- Expressivity of NNs
- Basics of training ("learning")
  - ↳ cost functions
  - ↳ learning algorithms


Feedforward NNs
x_1
x_2
x_d
layer 0
layer 1
layer 2
layer 3

- Basics of training ("learn
  - ↳ cost functions
  - ↳ learning algorithms



Each ○ is a "neuron"

Notation:

- $n_l \equiv$ # of neurons in layer l
- $L \equiv$ largest value of l (eg. $L=3$ here)
- $a_j^{(l)} \equiv$ output from the j^{th} neuron in layer l

Then:

$n_0 = d_x$

$n_L = d_y$

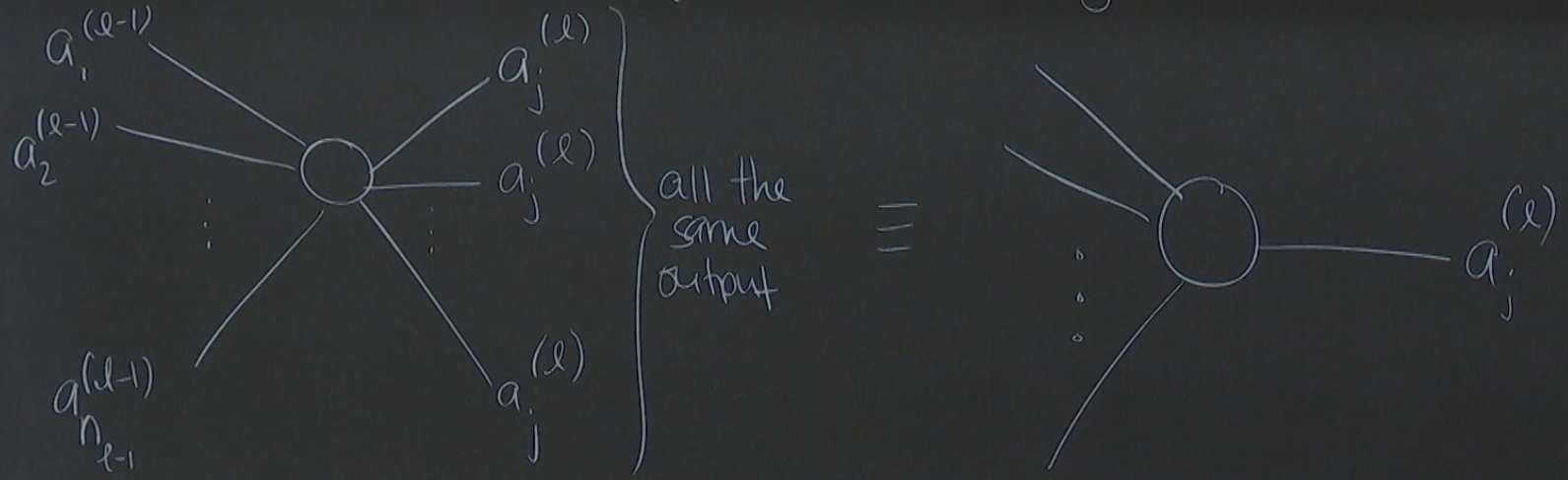
$a_i^{(0)} = x_i \quad (1 \le i \le d_x)$

ayer l

layer 0 layer 1 layer 2

$a_j^{(\ell)} = \sigma$

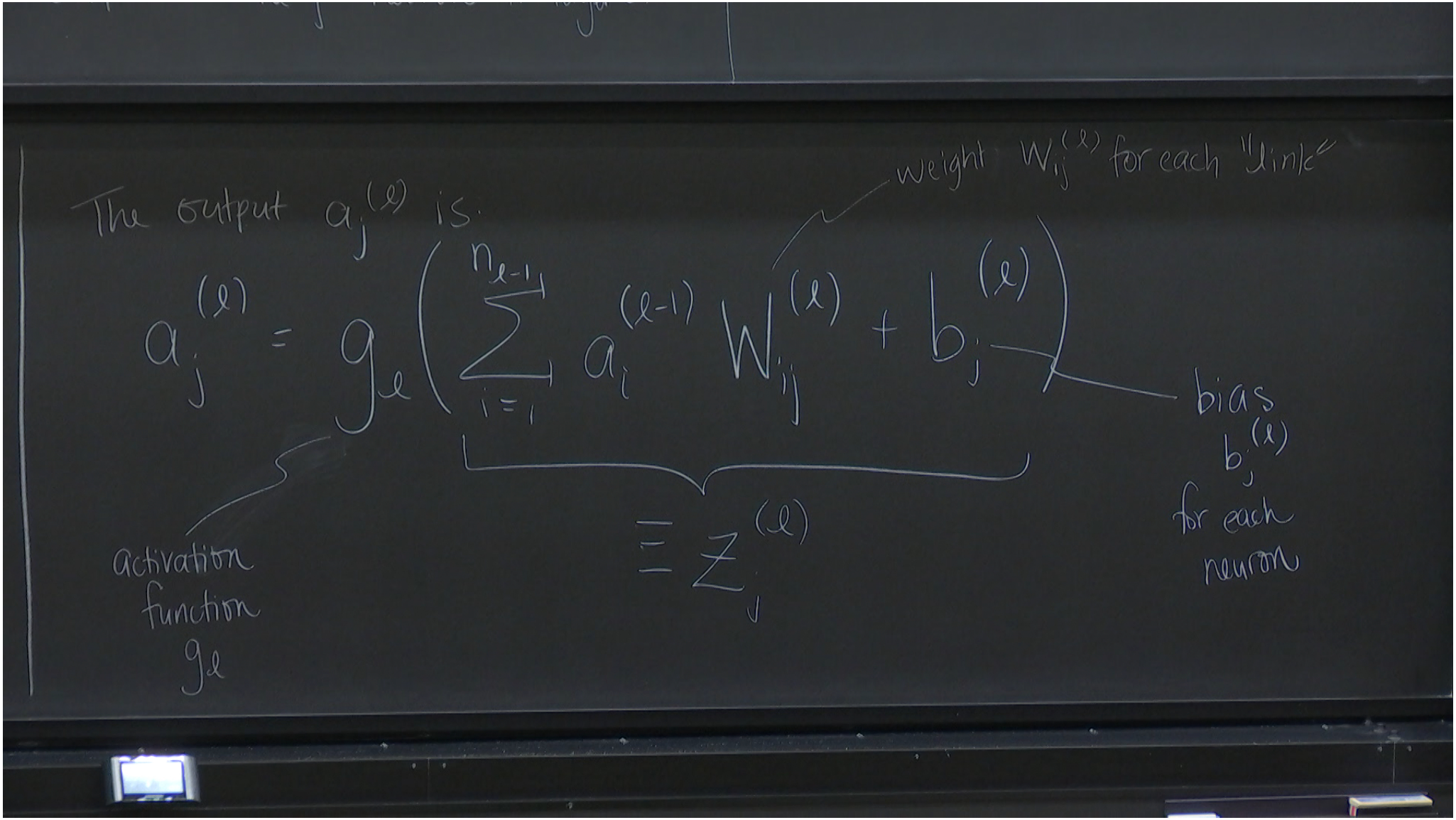
Let's zoom in on the j^{th} neuron in layer $\ell > 0$:



$\equiv$ output from the j^{th} neuron in layer ℓ

The output $a_j^{(\ell)}$ is:

$$a_j^{(\ell)} = g_\ell\left(\underbrace{\sum_{i=1}^{n_{\ell-1}} a_i^{(\ell-1)} W_{ij}^{(\ell)} + b_j^{(\ell)}}_{\equiv}\right)$$


The output $a_j^{(\ell)}$ is:
$a_j^{(\ell)} = g_\ell\left(\sum_{i=1}^{n_{\ell-1}} a_i^{(\ell-1)} W_{ij}^{(\ell)} + b_j^{(\ell)}\right)$
weight $W_{ij}^{(\ell)}$ for each "link"
bias $b_j^{(\ell)}$ for each neuron
activation function g_ℓ
$= z_j^{(\ell)}$

The weights and biases are changed as the network "learns"

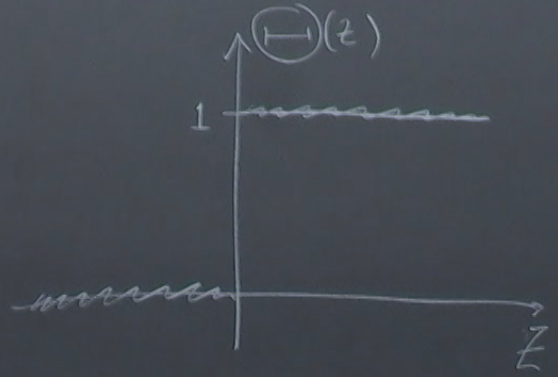
We can write the output from layer ℓ as:

$$\vec{a}^{(\ell)} \equiv \left(a_1^{(\ell)}, a_2^{(\ell)}, \ldots, a_{n_\ell}^{(\ell)}\right)$$

Then the output from the last layer is $\vec{a}^{(L)} = \vec{f}(\vec{x})$

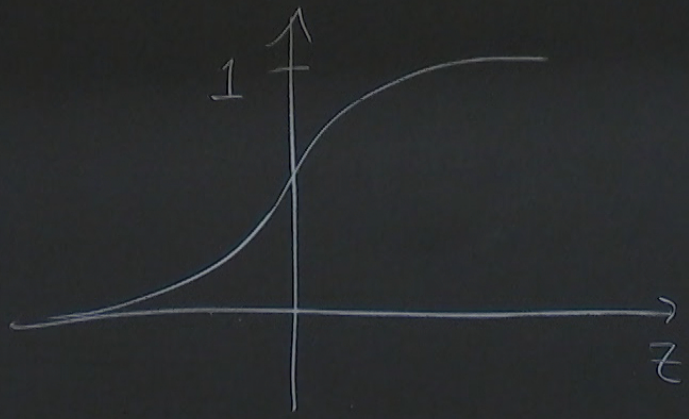
Activation functions: various choices are possible

① Perceptron $\Theta(z)$ (step function

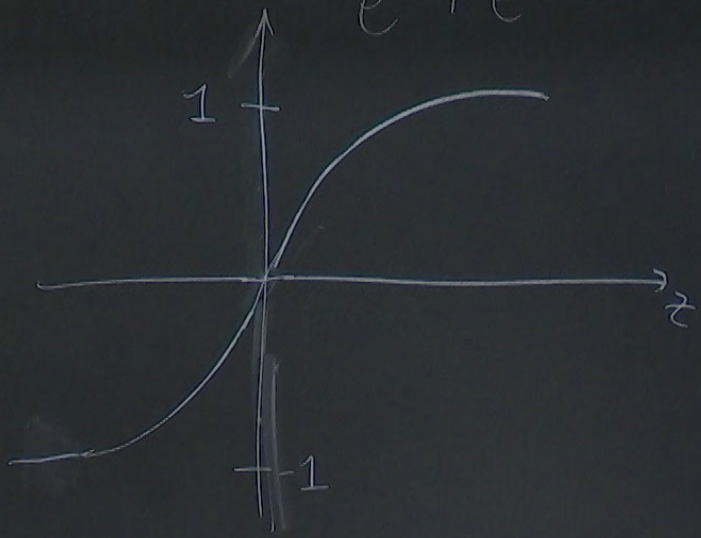


learning is difficult because most algorithms calculate the gradient of the neuron's output

② Sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$

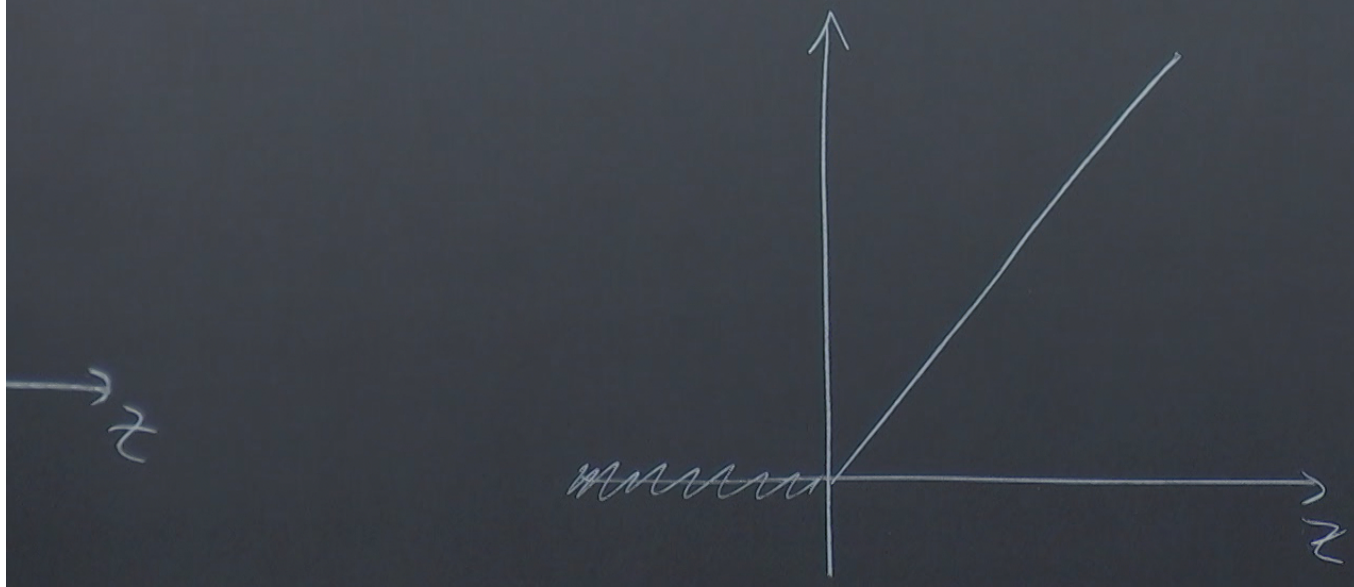


③ $\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$



④ Rectified linear unit

$$ReLU(z) = \max(0, z)$$

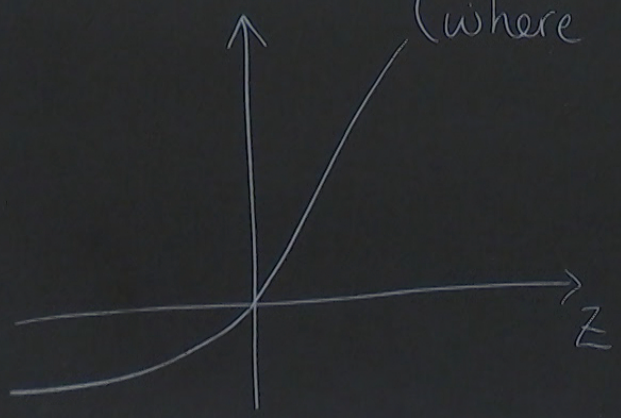


-1

(5) Exponential linear unit

$$ELU(z) = \begin{cases} \alpha(e^z - 1) & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

(where $\alpha > 0$)



For classification problems with $d_y > 1$, it is common to apply the softmax activation function to the last layer:

$$\left[\text{Softmax}(z_1, z_2, \ldots, z_{n_\ell})\right]_i = \frac{e^{z_i}}{\sum_{j=1}^{n_\ell} e^{z_j}} \quad (1 \le i \le n_\ell)$$

Can interpret outputs as probabilities

$n_0 = d_0$

Expressivity of NNs (See Neilsen, Chap. 4)

Universal approximation theorem: given a continuous $f(x)$, there exists a NN with one hidden layer that can approximate $f(x)$ with arbitrary accuracy

Caveats

$\vec{X} = (x_1, x_2, \ldots x_{d_x})$

- doesn't guarantee physical interpretability
- the NN required might be huge
- knowing that such a NN exists does not guarantee we can find it
- a more expressible function can be more prone to overfitting

Caveats

$\vec{X} = (x_1, x_2, \ldots x_{d_x})$

- doesn't guarantee physical interpretability
- the NN required might be huge
- knowing that such a NN exists does not guarantee we can find it
- a more expressible function can be more prone to overfitting

Now let's get our NN to learn!

Cost function

The goal of SL with NNs is to find weights and biases such that for input $\vec{x}$, the NN output $\vec{a}^{(L)}(\vec{x})$ is "close" to the label $\vec{y}(\vec{x})$

We use a cost function to measure how well the NN approximates the labels.

Different options:

(1) Mean-squared error (MSE)

$$C_{MSE} = \frac{1}{2|\mathcal{D}|} \sum_{\vec{x} \in \mathcal{D}} \sum_{i=1}^{n_L} \underbrace{\left[a_i^{(L)}(\vec{x}) - y_i(\vec{x}) \right]^2}_{\| \vec{a}^{(L)}(\vec{x}) - \vec{y}(\vec{x}) \|^2}$$

(2) Cro

(2) Cross entropy (CE) (for $0 \le y_i(\vec{x}) \le 1$, $0 \le a_i^{(L)}(\vec{x}) \le 1$)

$$C_{CE} = \frac{-1}{|\mathcal{D}|} \sum_{\vec{x} \in \mathcal{D}} \sum_{i=1}^{n_L} \left[y_i(\vec{x}) \log a_i^{(L)}(\vec{x}) + \left(1 - y_i(\vec{x})\right) \log\left(1 - a_i^{L}(\vec{x})\right) \right]$$

Next time we will introduce algorithms that aim to minimize C