MACHINE LEARNING IN PHYSICS
FOUNDATIONS 2

Harrison B. Prosper
PHY 6938 Fall 2024

Recap: AI, ML, and DL

Artificial Intelligence
Algorithms that cause machines to exhibit human- or *super-human*-level intelligence.

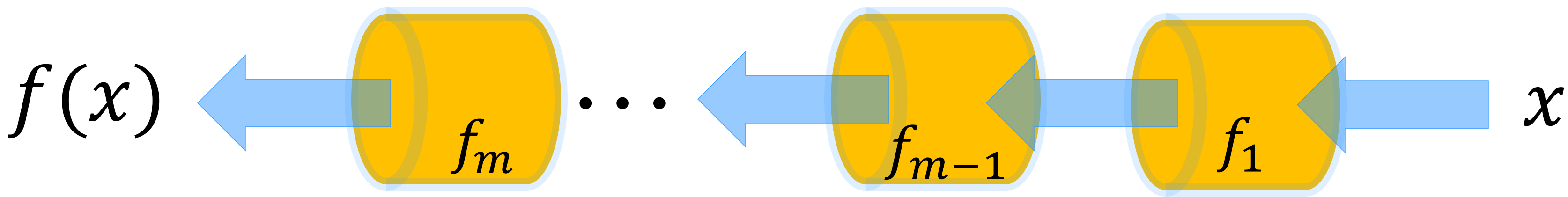
Machine Learning
Algorithms for modeling data

Deep Learning
ML using (large) neural networks

Recap: Deep Learning

Uses functions formed by *composition:*

$$
\begin{aligned}
f(x) &= f_m \circ f_{m-1} \circ \cdots f_1 \\
&= f_m\left(f_{m-1}\left(\ldots f_1(x)\right) \ldots\right)
\end{aligned}
$$



Example:

$$f(x) = \mathrm{softmax}\left(\mathrm{dropout}\left(\mathrm{linear}\left(\mathrm{flatten}\left(\boldsymbol{g}\left(\boldsymbol{c}\left(\boldsymbol{h}\left(\boldsymbol{c}(x)\right)\right)\right)\right)\right)\right)\right)$$

Recap: The Perceptron

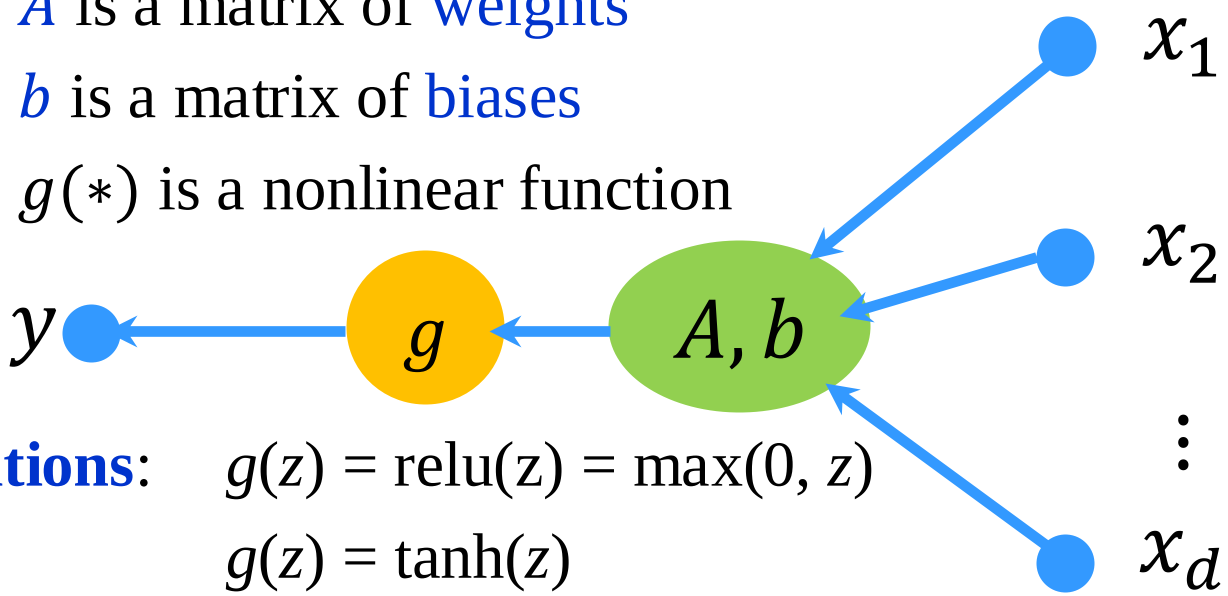
$$y = g(xA^T + b)$$

x is matrix (one or more rows) of input data

A is a matrix of weights

b is a matrix of biases

$g(*)$ is a nonlinear function



Activations: $g(z)$ = relu(z) = max(0, z)

$g(z)$ = tanh(z)

$g(z)$ = sigmoid(z) = 1 / (1 + exp(-z))

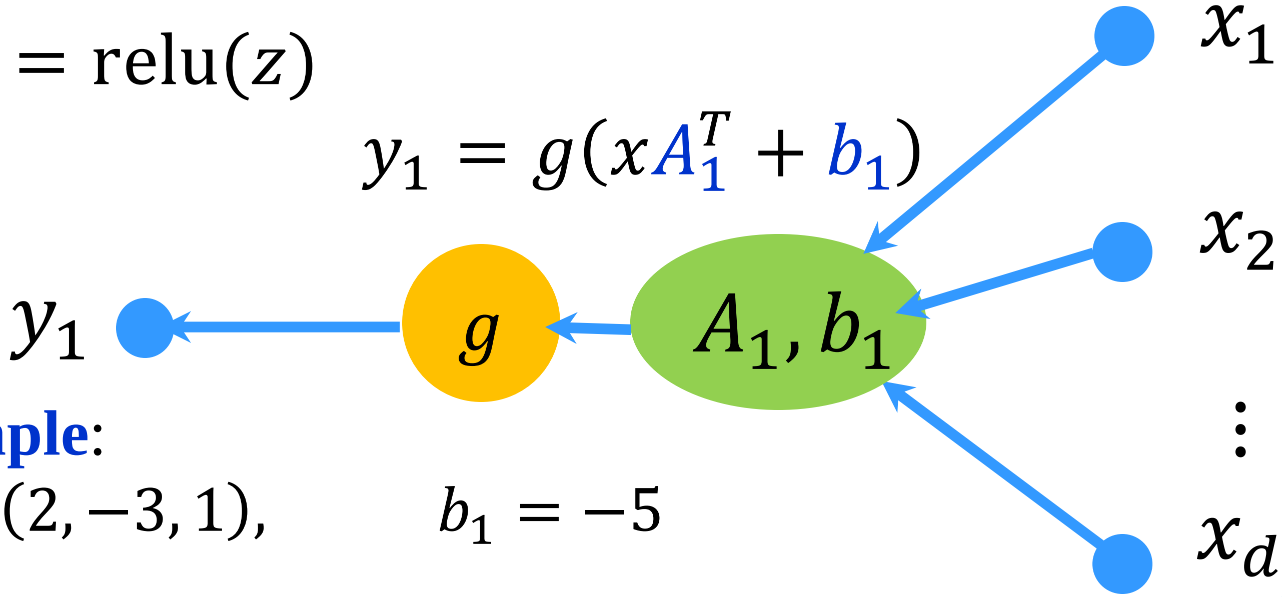
https://news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon

MULTI-NODE PERCEPTRON

Multi-Node Perceptron

$g(z) = \mathrm{relu}(z)$

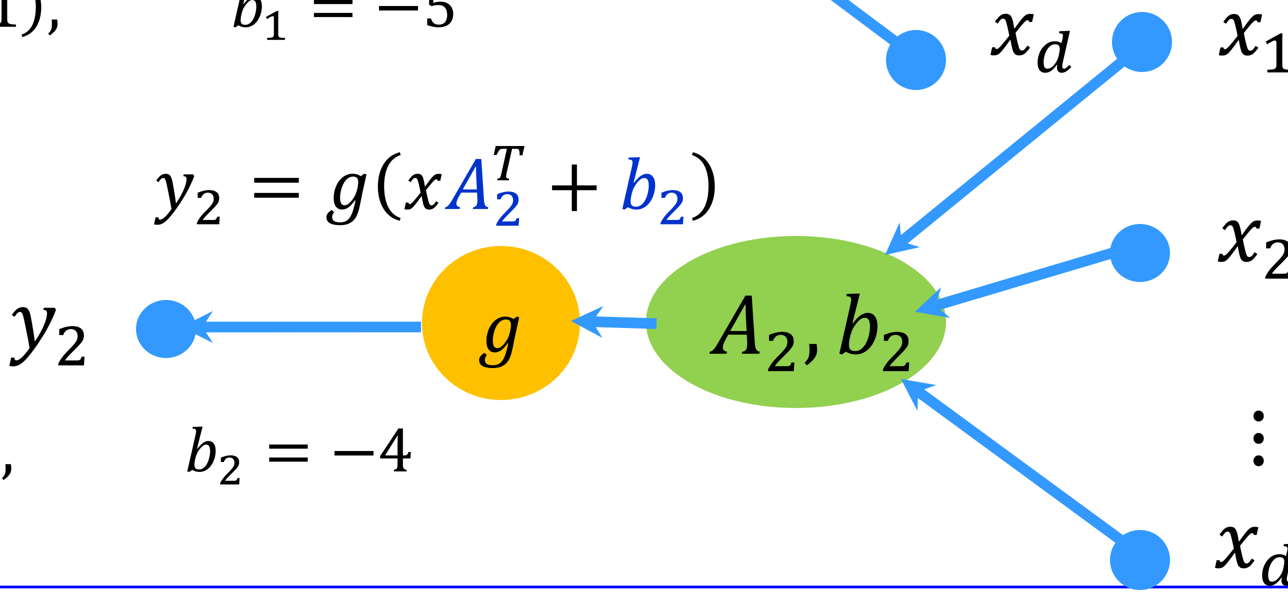
$y_1 = g(xA_1^T + b_1)$



Example:

$A_1 = (2, -3, 1)$, $\quad b_1 = -5$

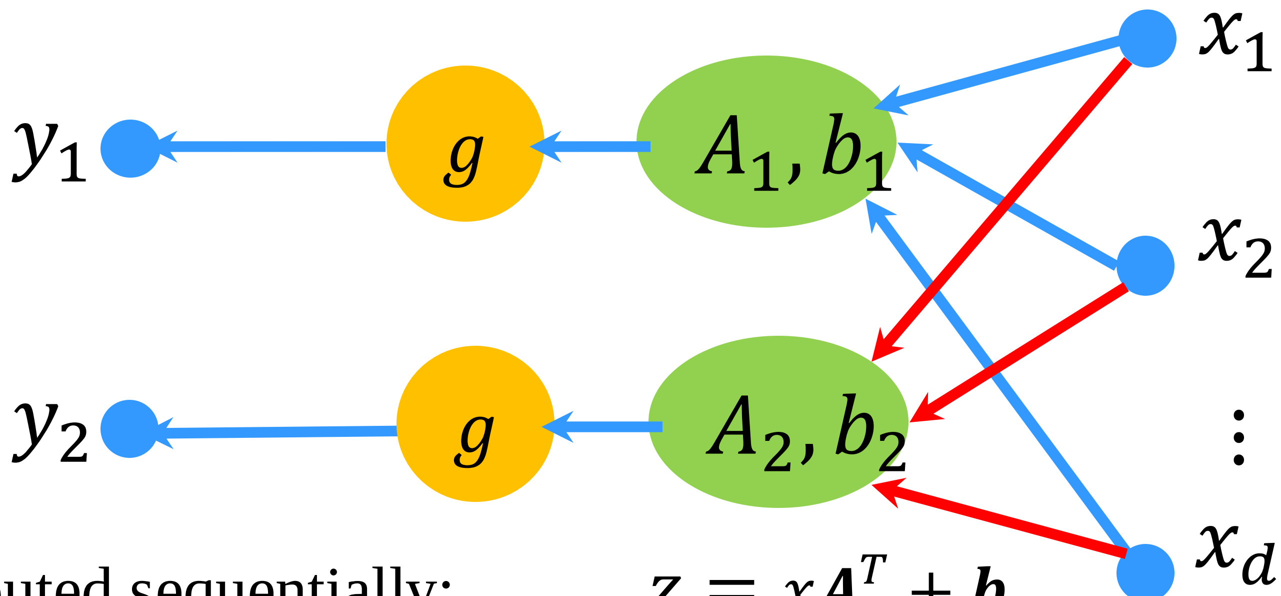
$y_2 = g(xA_2^T + b_2)$



$A_2 = (1, 2, 3)$, $\quad b_2 = -4$

Multi-Node Perceptron

A multi-node perceptron is often drawn as follows,



computed sequentially: $z = x\boldsymbol{A}^T + \boldsymbol{b}$

$y = g(z)$ and, usually, the

activation function $g(z)$ is applied *elementwise*, that is, to every element of its matrix input. (Demo)

Multi-Node Perceptron

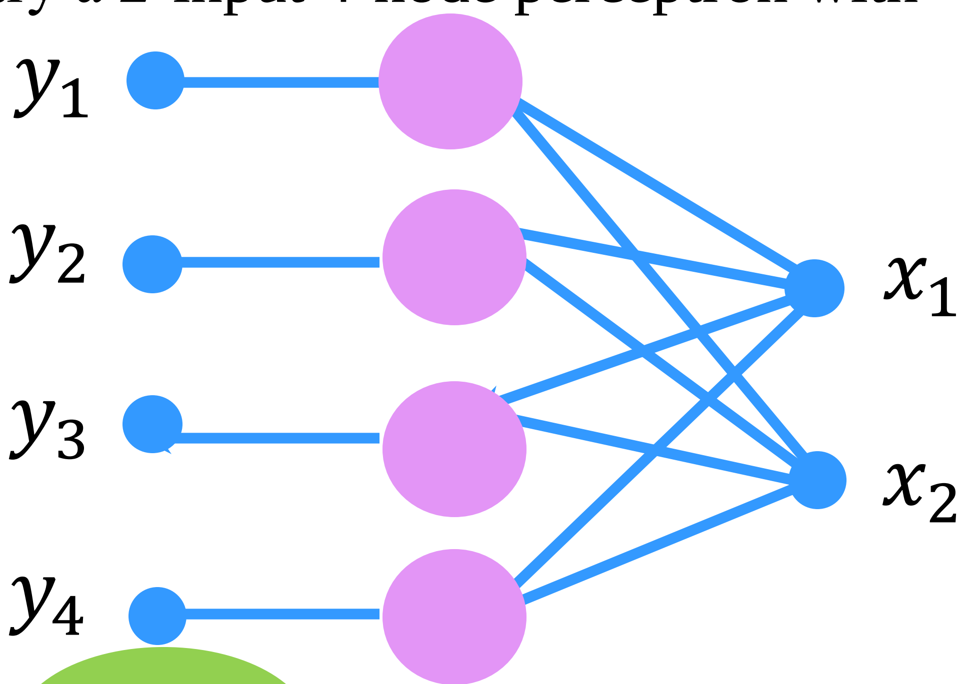
Consider the task of modeling the data triplets $D = \{(x_1, x_2, y)_{i=1}^{N}\}$ with a function of the form $y = f(x_1, x_2, \omega)$. Let's try a 2-input 4-node perceptron with $g(z) = z\sigma(z)$,

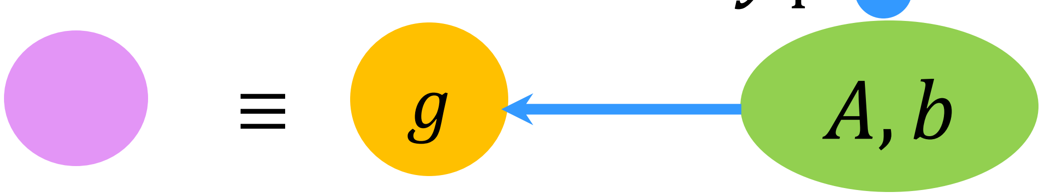
where,

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

$z\sigma(z)$ is called a silu.

ω are free parameters.

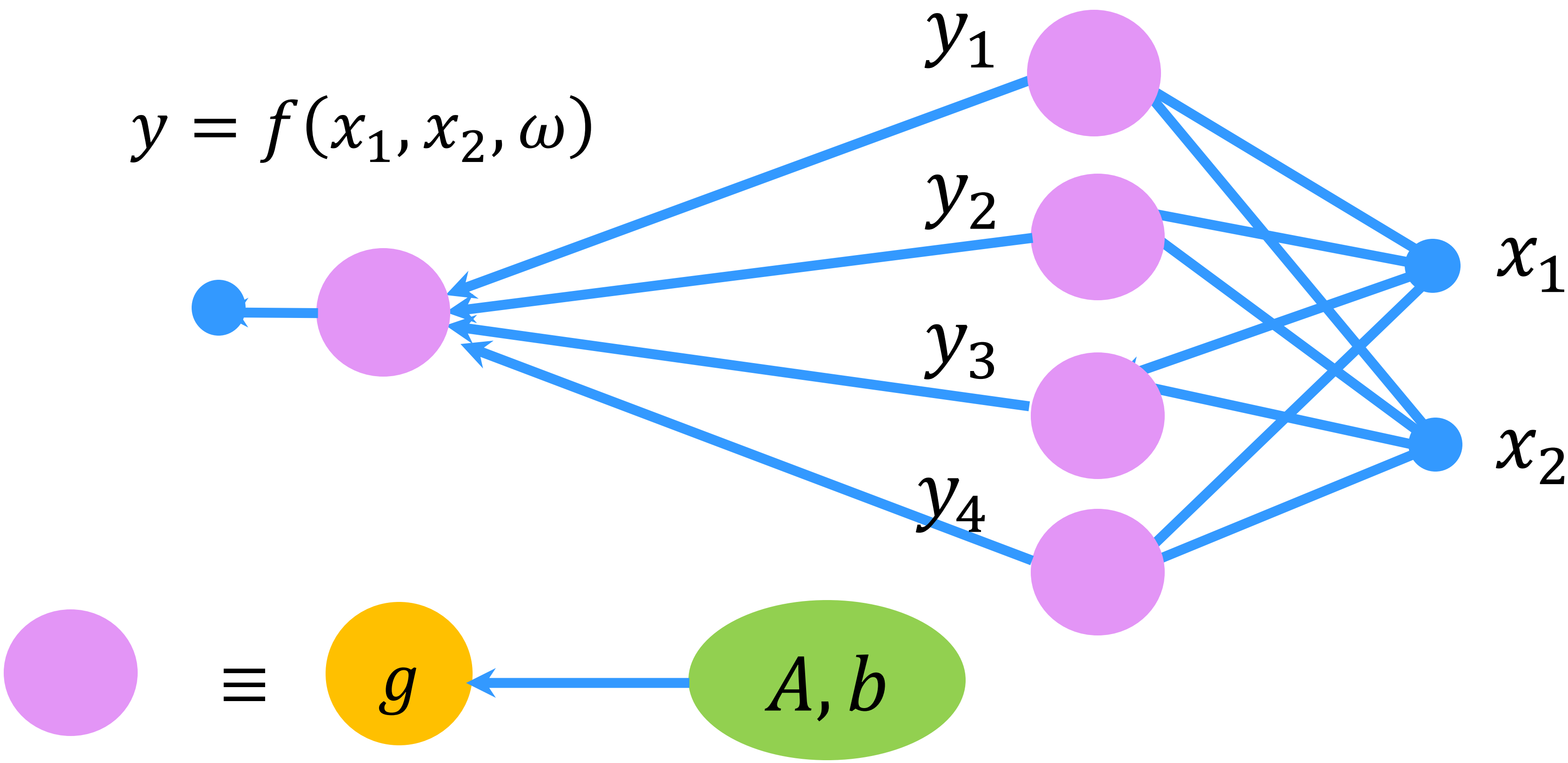




Multi-Node Perceptron

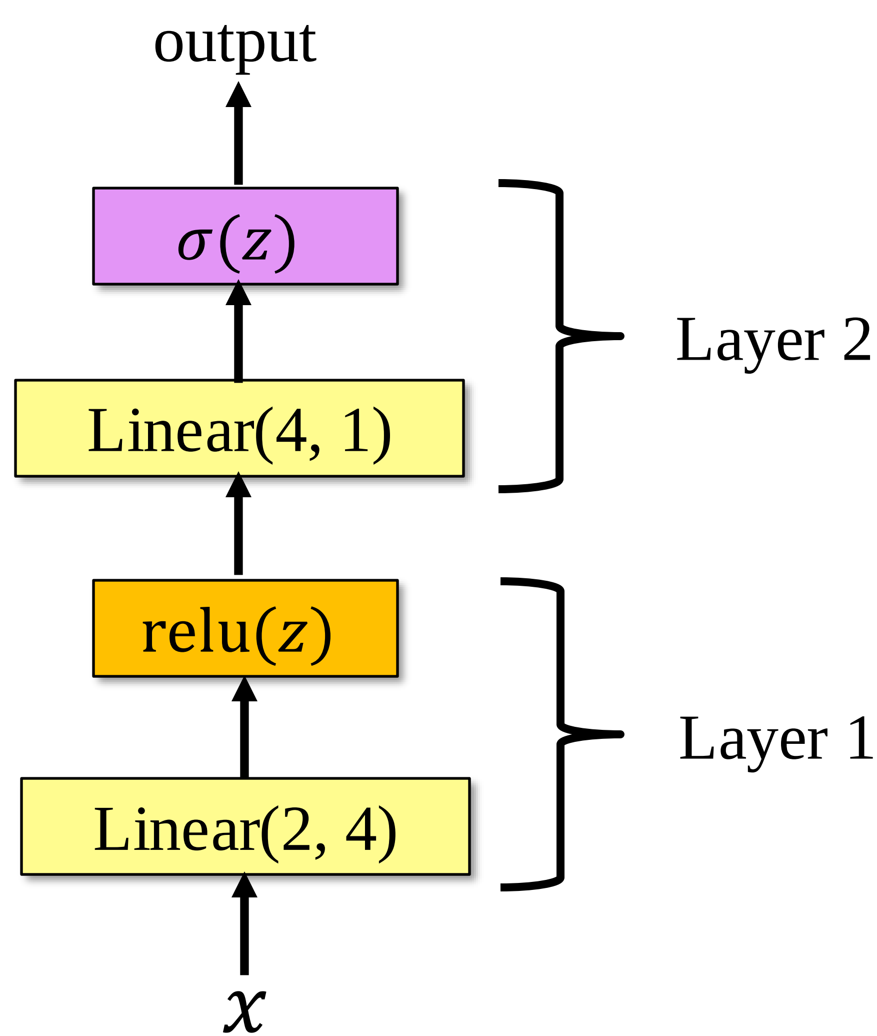
Then, let's feed the output of the first perceptron into a 4-input 1-node perceptron:

$$y = f(x_1, x_2, \omega)$$



Multi-Node Perceptron

Given the complexity of ML models (i.e., ML functions), it is now common to use a graphical representation of these models that uses higher-level components.



$$f(z) = zA^T + b$$

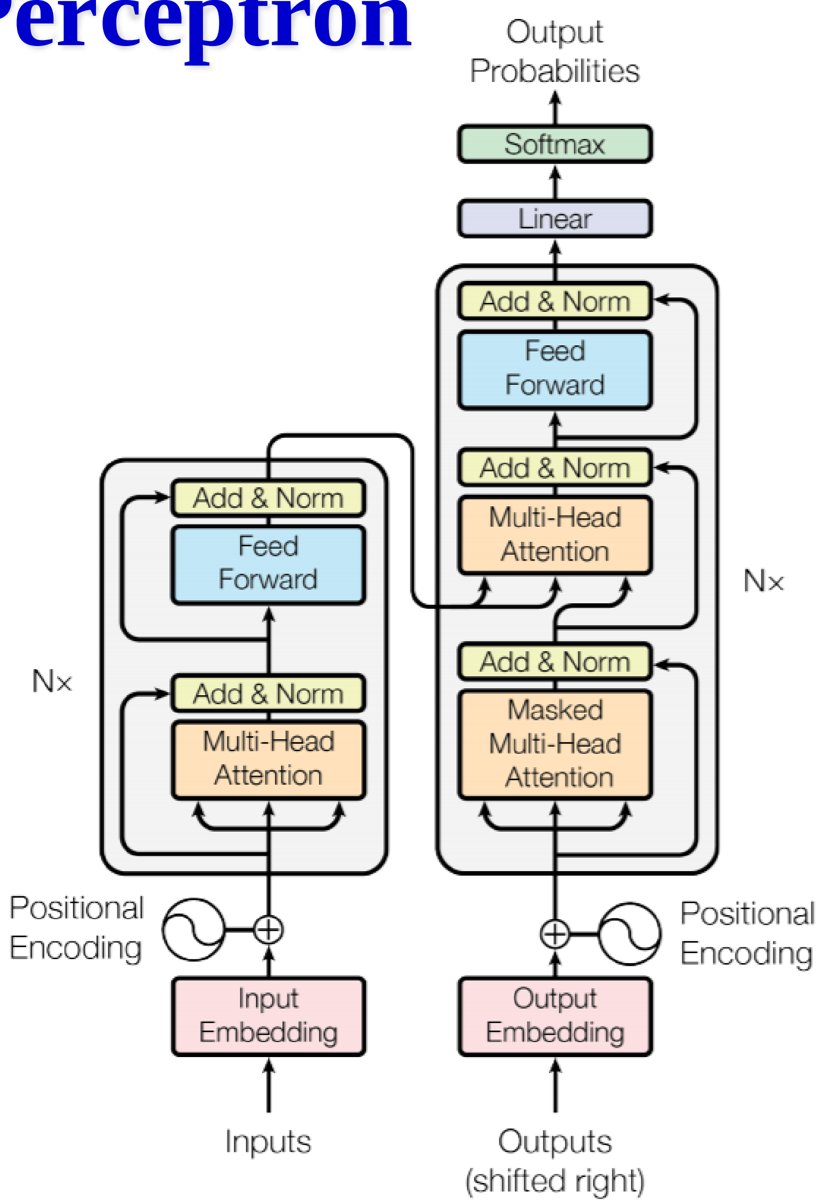
$$\text{relu}(z) = \max(0, z)$$

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

Multi-Node Perceptron

Here, for example, is a graphical representation of **transformer**, the model that powers ChatGPT.

There are 2 x 96 of these "layers"!

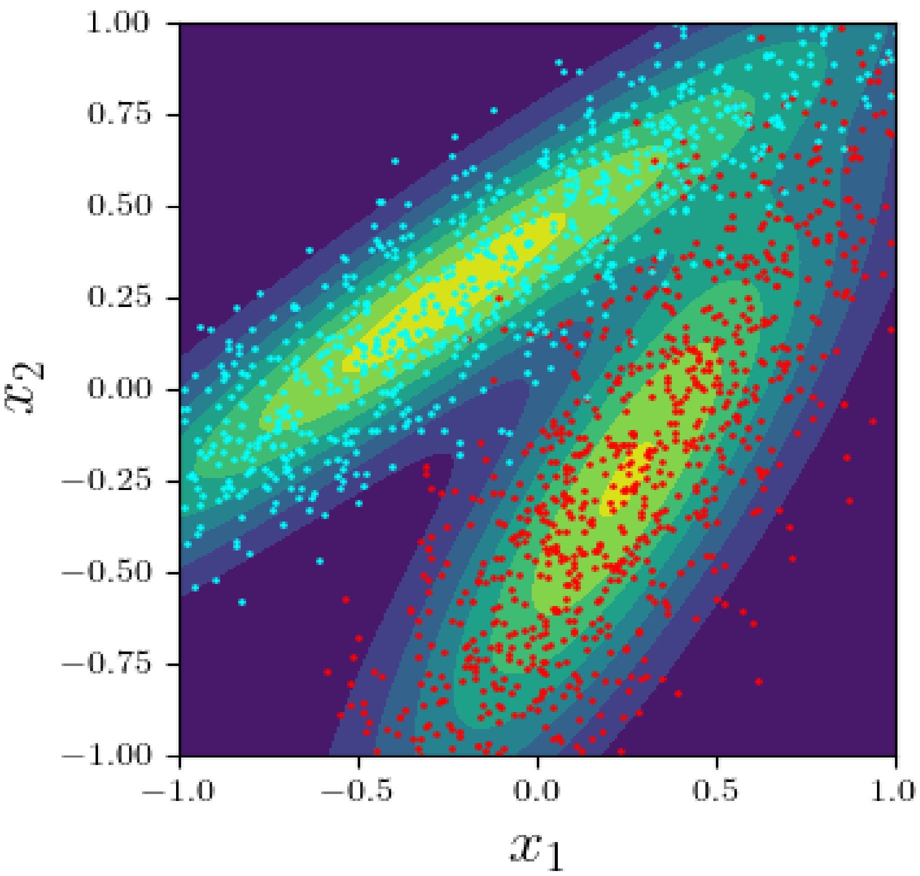


LOSS FUNCTIONS

2D Dataset

- In this part of the course, we'll use a simple synthetic dataset comprising two classes of objects characterized by real numbers (x_1, x_2).

- The data are generated from two bivariate normal distributions.



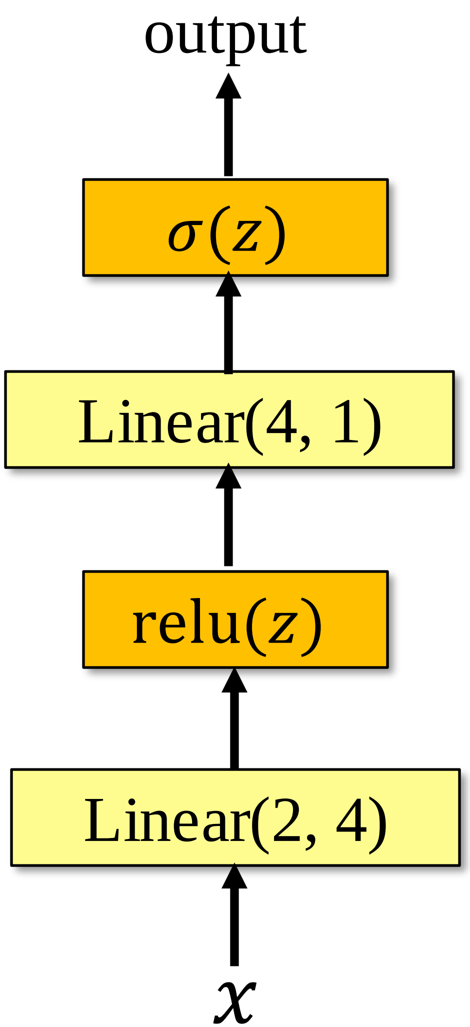
Loss Functions

Suppose our goal is to approximate an unknown function

$$y = f(x_1, x_2)$$

given a dataset $D = \{(x_1, x_2, y)_{i=1}^{N}\}$,

perhaps using the *model* on the right.



Loss Functions

A well-known approach is to approximate $f(x_1, x_2)$ with a parameterized function $f(x_1, x_2, \omega)$ and find the *best-fit values* ω^* by minimizing

$$R(\omega) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f_i)^2$$

with respect to the parameters ω, where $f_i = f(x_{1i}, x_{2i}, \omega)$.

This is often referred to as a **least-squared fit**.

Loss Functions

In machine learning, one minimizes

$$R(\omega) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f_i)$$

where the function, $L(y_i, f_i)$, called the **loss function**, measures the discrepancy between the desired **target** y and the model f.

The quantity $R(\omega)$ is called the **empirical risk** and the task of minimizing it is called **empirical risk minimization**.

This is just a generalization of least-squared fitting.

Risk Functional

In mathematics one can often gain insight by taking the limit of an expression. Let's take the limit of

$$R(\omega) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f_i)$$

as $N \to \infty$, that is, as the amount of data grows without limit.

In that limit, the empirical risk becomes the **risk functional**,

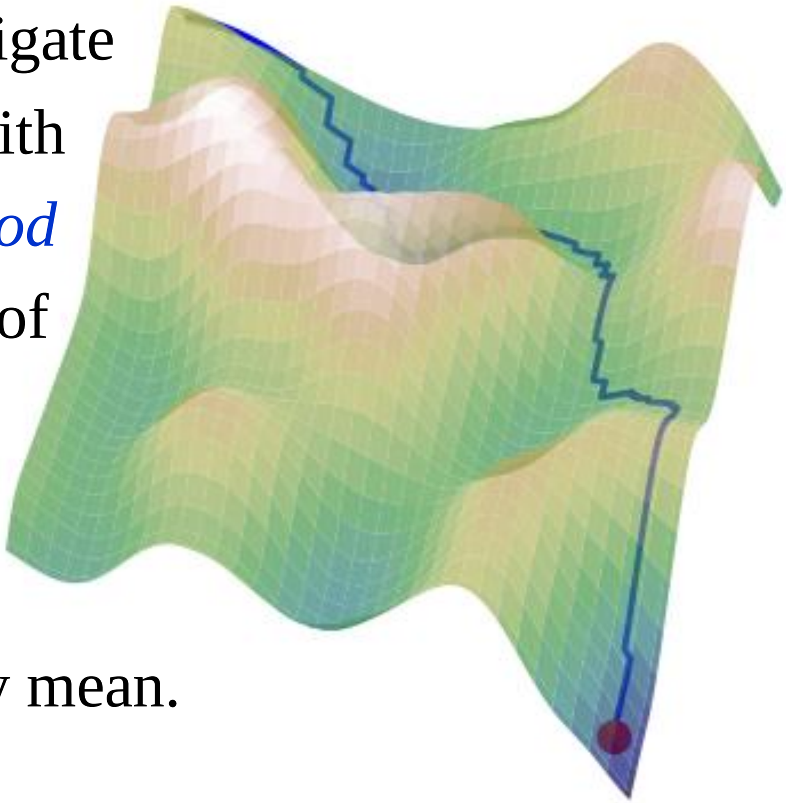
$$R[f] = \int dx \int dy \, L(y, f) \, p(x, y)$$

where $p(x, y) dx dy$ is the probability distribution of the data.

Risk Functional: Landscape

The empirical risk defines a highly corrugated very high-dimensional "landscape" in the space of parameters.

The goal of an **optimizer** is to navigate the landscape ($R(\omega)$) associated with a *finite* amount of data to find a *good approximation* of the lowest point of the landscape ($R[f]$) associated with an *infinite* amount of data. When ML researchers say that a model *generalizes* this is what they mean.



Summary

- We described the multi-node perceptron and how machine learning models are represented graphically.
- We illustrated the calculation of the output of a perceptron.
- We discussed the generalization of least-squared fitting to empirical risk minimization.