Deep Learning and Applications

DSA 5204 • Lecture 2
Dr Low Yi Rui (Aaron)
Department of Mathematics

National University of Singapore

Homework 1

Homework 1 has been uploaded on Canvas

Submission through Canvas submission folder

Due: 4th Feb 2023

Take note of the late submission policies

Project Instructions

Please read Canvas for project instructions

TLDR version

- Form groups of 5-7 by 31 January
- Three main components
  - **Proposal (Homework 2)**
  - **Presentation with Q&A (live, during last 2 classes)**
  - **Report**
- Any questions, please post on Canvas forums

Last Time

- **T**ask, **E**xperience, **P**erformance
  Machine Learning = Improve P on T with E
- Linear regression models as a simplest example
$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$$
- Capture nonlinearity using linear basis model
$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})$$
- Today, we will look at a further extension of these ideas to neural networks

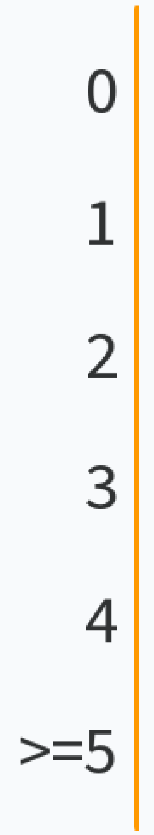
Motivation for Representation Learning and Deep Learning

Let’s play a game – Round 1

1. Choose 3 different numbers out of $\{0,1,\dots,9\}$
2. Write them down on a piece of paper. Don’t change them!
3. Now, for the game:
    1. I will give you a “magic” number
    2. You add 0, 1, 2 or all 3 of your chosen numbers, to get as close to this number as possible. Record the absolute difference
    3. Example: You chose $\{3,5,9\}$. I give 15. The closest you can get is $5+9=14$. Difference is 1. This is your score
    4. Example: You chose $\{3,5,9\}$. I give 1. The closest you can get is 0. Difference is 1. This is your score

When poll is active, respond at **pollev.com/aaronyrlow**

Round 1.1: The magic number is 12. What is your score?



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

When poll is active, respond at **pollev.com/aaronyrlow**

Round 1.2: The magic number is 2. What is your score?

When poll is active, respond at **pollev.com/aaronyrlow**

Round 1.3: The magic number is 20. What is your score?

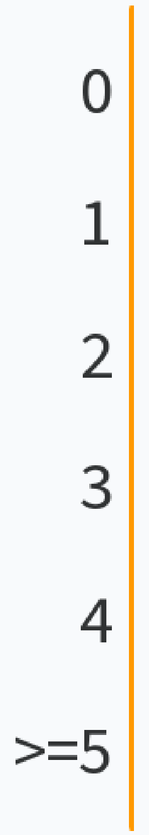
Round 2

For the next round, implement the following changes

- instead of picking 3 numbers out of $\{0,1,\dots,9\}$, pick 6 instead
- Now, you can pick either 0, 1, 2 or 3 of the 6 numbers you picked to get close to the answer
- Example: you picked $\{0,1,2,3,4,8\}$
  - **The given number is 5, you can pick $2, 3$, and your score is 0**
  - **The given number is 18, you can pick $3, 4, 8$, and your score is 3 (note that you cannot pick more than 3)**

When poll is active, respond at **pollev.com/aaronyrlow**

Round 2.1: The magic number is 4. What is your score?



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

When poll is active, respond at **pollev.com/aaronyrlow**

Round 2.2: The magic number is 10. What is your score?

Round 2.3: The magic number is 19. What is your score?

Recap

Round 1:

$$\text{Score} = \left| f - \sum_{i=1}^{3} w_i \phi_i \right| \qquad w_i \in \{0,1\}$$

Given number Your numbers

Round 2:

$$\text{Score} = \left| f - \sum_{i=1}^{3} w_i \phi_i(f) \right| \qquad w_i \in \{0,1\}$$

$$\boldsymbol{\phi}(f) = \{3 \text{ choices based on } f\}$$

Why did we do better in round 2?

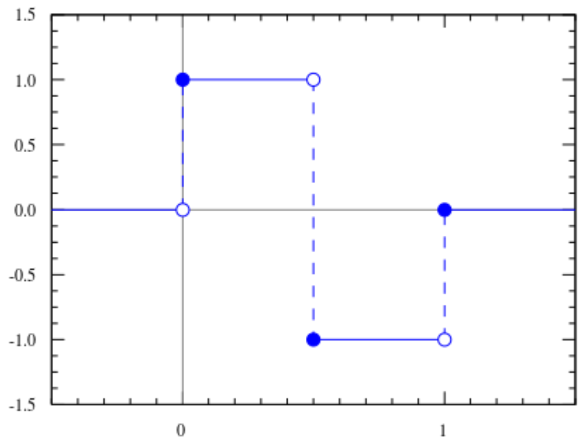
Back to linear basis models

Recall that linear basis models can be chosen with **universality**

Examples (1D):

$$\boldsymbol{\phi}(x) = (1, x, x^2, x^3, \dots)$$
$$\boldsymbol{\phi}(x) = (1, \cos(x), \sin(x), \cos(2x), \sin(2x), \dots)$$
$$\boldsymbol{\phi}(x) = \left(2^{\frac{n}{2}}\psi(2^n x - k)\right)_{n,\,k=1,2,\dots}$$

Haar Wavelet: $\psi(x) =$



Why don't we just use linear basis models with a large m?

Approximation Efficiency

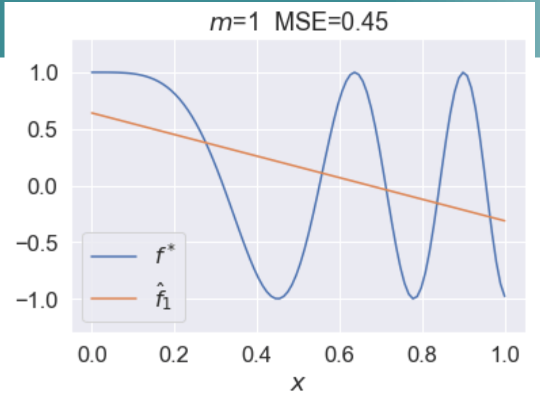
Goal: approximate some function f^*

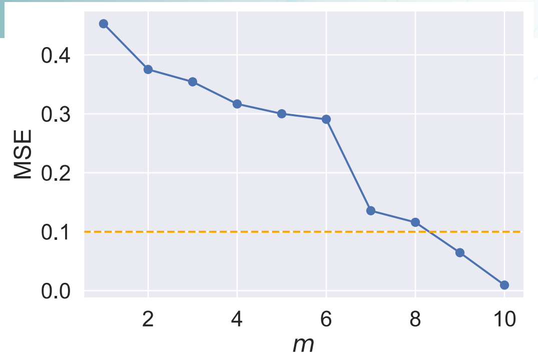
Consider two basis sets:

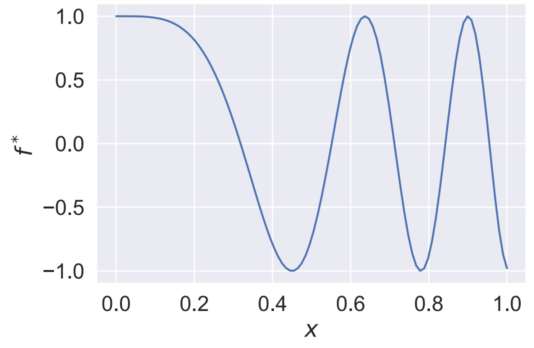
$$\boldsymbol{\phi}_m(x) = (1, x, x^2, x^3, \dots, x^m)$$
$$\boldsymbol{\phi}_m(x) = (1, \cos(x), \cos(2x), \dots, \cos(mx))$$

Let us consider fitting some function f^* with varying basis and varying m

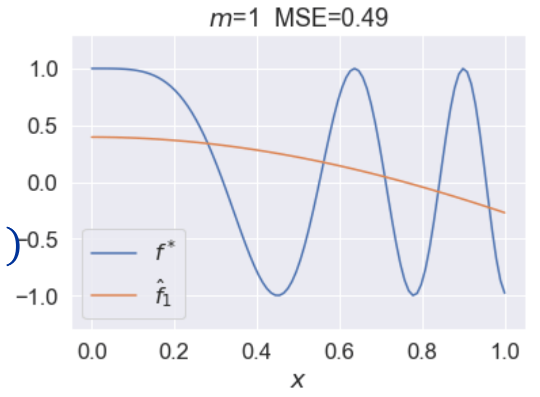
$$\boldsymbol{\phi}_m(x) = (1, x, x^2, x^3, \dots, x^m)$$

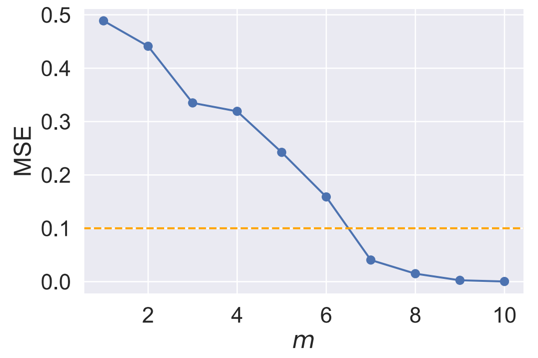






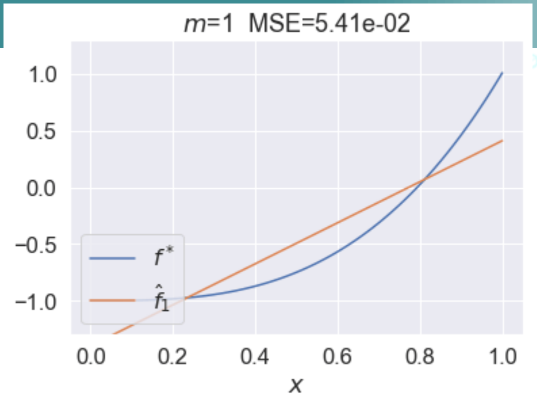
f^*

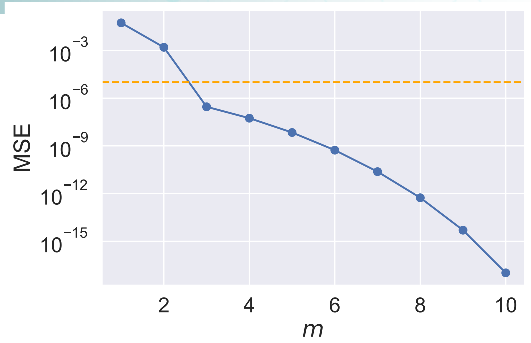


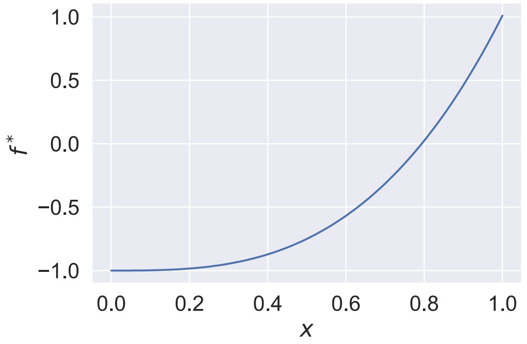


$$\boldsymbol{\phi}_m(x) = (1, \cos(x), \cos(2x), \dots, \cos(mx))$$

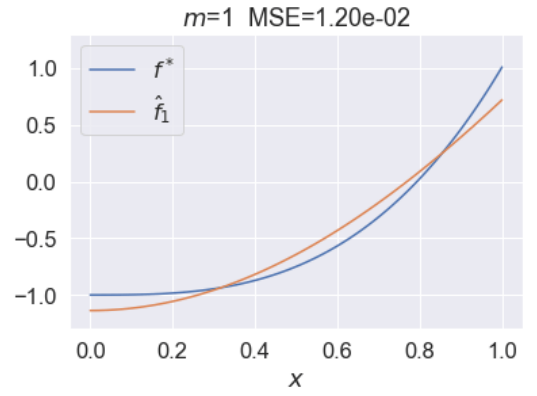
$$\phi_m(x) = (1, x, x^2, x^3, \dots, x^m)$$

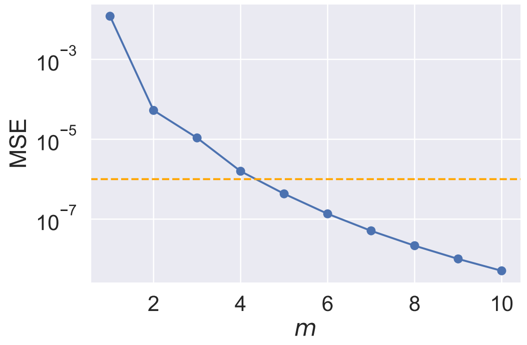






f^*



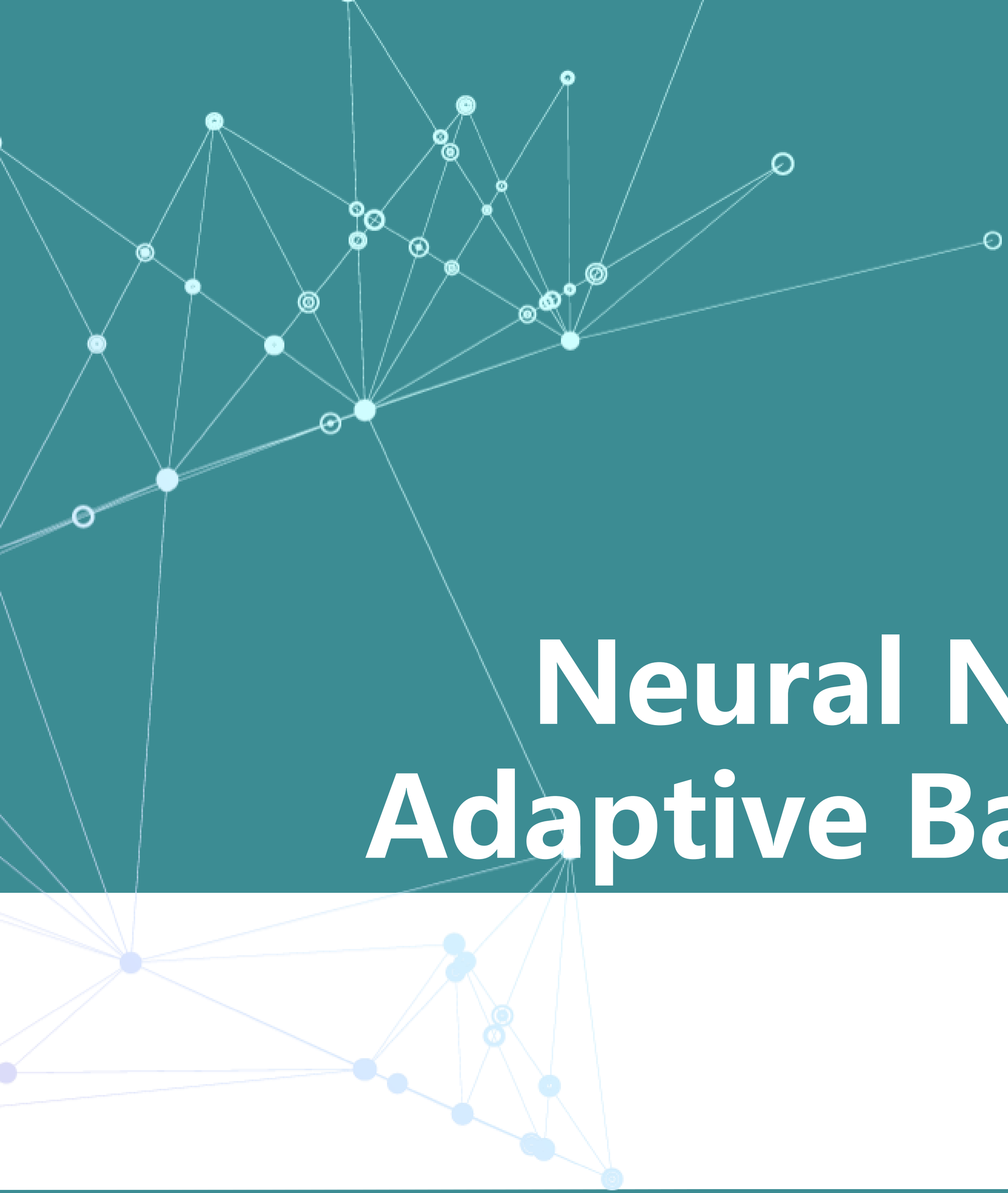


$$\phi_m(x) = (1, \cos(x), \cos(2x), \dots, \cos(mx))$$

Adaptive Approach

Previous experiments show

- How well we can learn with a given computational budget ($\boldsymbol{m}$) depends **both** on the target (f^*) and the choice of basis ($\boldsymbol{\phi}$)
- No $\boldsymbol{\phi}$ works universally well for **all** f^*
- Adapt the choice of $\boldsymbol{\phi}$ to data!



Neural Networks as Adaptive Basis Models

Adaptive Basis Models

Usual linear basis models have the form

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) \qquad \boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{w} \in \mathbb{R}^m$$

$\Leftrightarrow$ weights / coefficients

Adaptive models

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}; \boldsymbol{\theta}) \qquad \boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{w} \in \mathbb{R}^m, \boldsymbol{\theta} \in \mathbb{R}^p$$

$\downarrow$ feature map is adapted to data.

The parameters $\boldsymbol{\theta}$ are learned from data to find the best possible basis.

Example (Nonlinear Approximation)

Consider

$$f^* = 1 + x^{10}$$

Polynomial basis

$$\boldsymbol{\phi}_m(x) = (1, x, x^2, \dots, x^{m-1})$$

Adaptive polynomial basis

$$\boldsymbol{\phi}_m(x; \boldsymbol{\theta}) = \left(x^{\theta_0}, x^{\theta_1}, \dots, x^{\theta_{m-1}}\right)$$

$$\boldsymbol{\theta} \in \mathbb{N}^{\boldsymbol{m}}$$

DeVore, R. A. Nonlinear Approximation. ***Acta numerica*** *1998, 7, 51–150.*

Learned Basis as Feature Maps

The basis $\{\boldsymbol{\phi}(\boldsymbol{x};\boldsymbol{\theta}) : \boldsymbol{\theta} \in \mathbb{R}^p\}$ is adapted to data by adjusting $\boldsymbol{\theta}$.

For this reason, we also call

$$\phi_i(\cdot;\boldsymbol{\theta}) : \mathbb{R}^d \to \mathbb{R}$$

feature maps, as they extract information from dataset.

Equivalently, they give good **representation** of the data

Example (Learning XOR)

The **exclusive or (XOR)** function on two binary variables

$$f^*(x_1, x_2) = \begin{cases} 1 & \text{exactly one of } x_i \text{ is } 1 \\ 0 & \text{otherwise} \end{cases}$$

Let us build a simple linear model to represent this function

Data Matrix:

$$X = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \qquad y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Empirical Risk:

$$R(\boldsymbol{\theta}) = \frac{1}{4}\sum_{i=1}^{4}\left(f^*(\boldsymbol{x}_i) - f(\boldsymbol{x}_i;\boldsymbol{\theta})\right)^2$$

Linear Model:

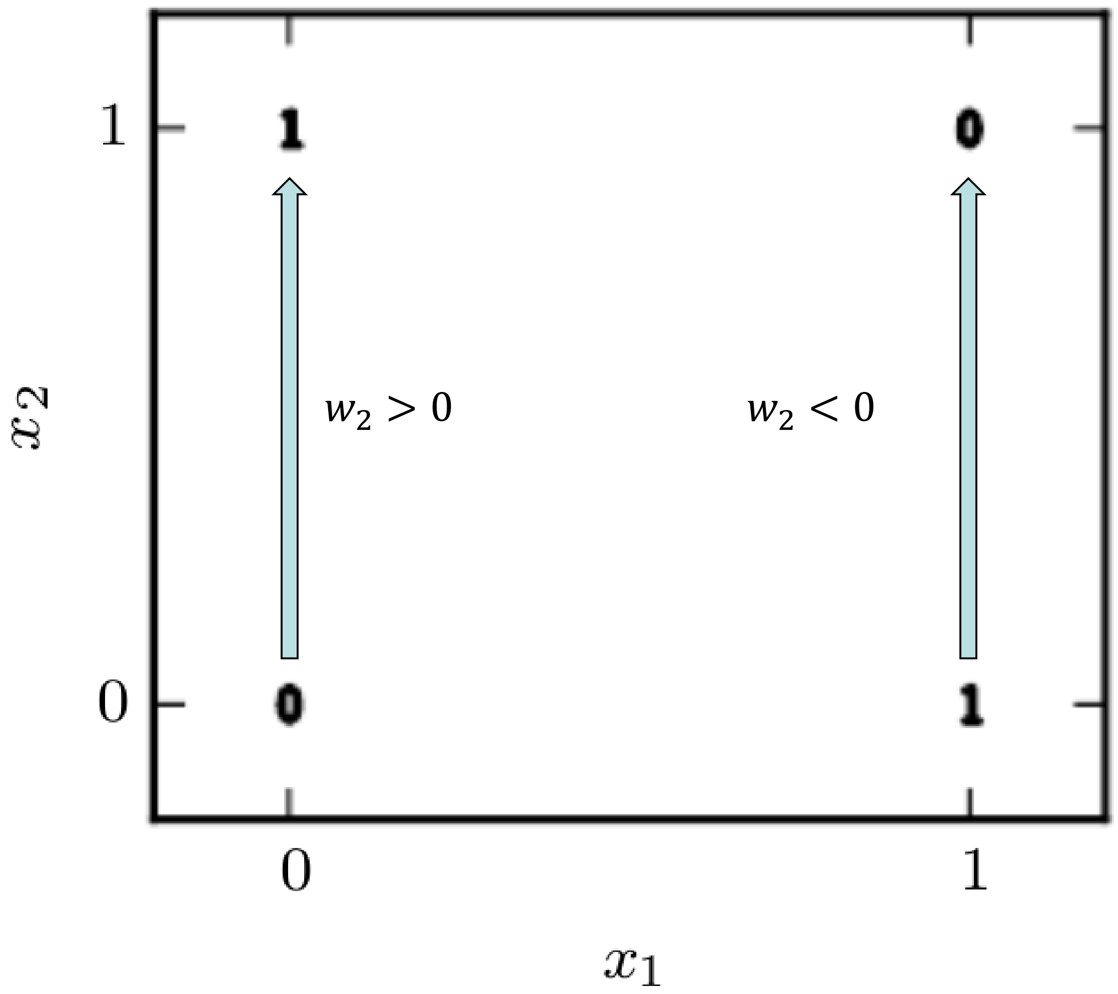
$$f(\boldsymbol{x};\boldsymbol{\theta}) = f(\boldsymbol{x}:\boldsymbol{w},b) = \boldsymbol{w}^T\boldsymbol{x} + b$$

Solving the regression problem gives

$$\boldsymbol{w} = (0,0) \qquad b = \frac{1}{2}$$

Hence, $f(x;w,b) \equiv \frac{1}{2}$

Why is linear model not enough?



Neural Network for the XOR Function

We are going to write $f = f^{(2)} \circ f^{(1)}$:

$$\boldsymbol{h} = f^{(1)}(\boldsymbol{x}; W, \boldsymbol{c})$$
$$y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$$

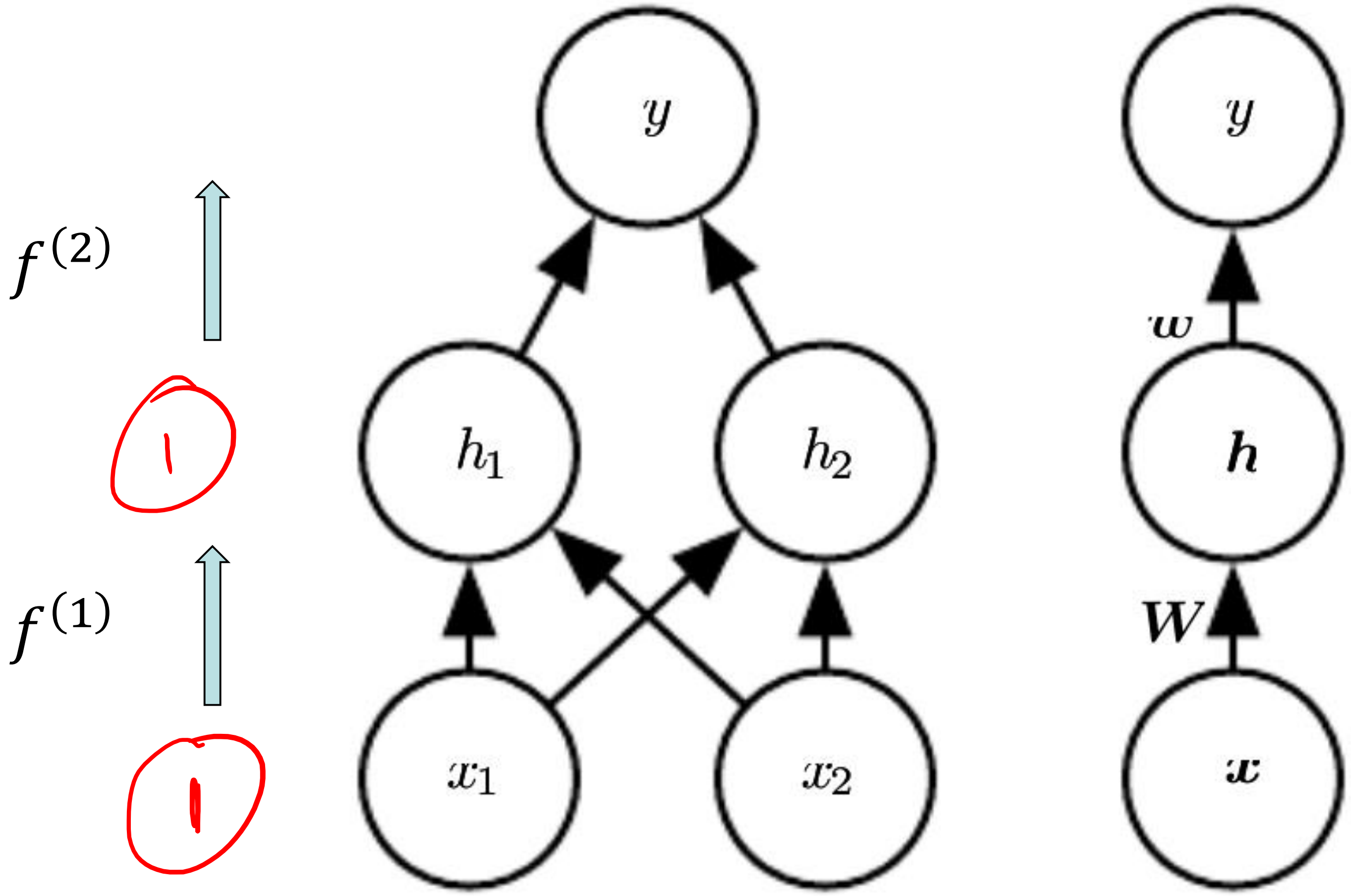
The vector $\boldsymbol{h}$ is called a vector of **hidden units**

Its dimension is m, also known as the *width* of the hidden layer

The combined model is

$$f(\boldsymbol{x}; \boldsymbol{\theta}) \equiv f(\boldsymbol{x}; W, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}\left(f^{(1)}(\boldsymbol{x}; W, \boldsymbol{c}); \boldsymbol{w}, b\right)$$

Graph Representation of Neural Networks



Activation Functions

We have not specified how to choose $f^{(1)}$ and $f^{(2)}$

Simplest choice: linear functions

$$f^{(1)}(\boldsymbol{x}; W, \boldsymbol{c}) = W\boldsymbol{x} + \boldsymbol{c} \rightarrow h$$
$$f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b) = \boldsymbol{w}^T\boldsymbol{h} + b$$

$$\begin{aligned} f^{(2)}\left(f^{(1)}(x)\right) &= \boldsymbol{w}^T(W\boldsymbol{x} + \boldsymbol{c}) + b \\ &= \underbrace{(W^T\boldsymbol{w})^T}_{\boldsymbol{w}'}\boldsymbol{x} + \underbrace{(\boldsymbol{w}^T c + b)}_{b'} \end{aligned}$$

We need some form of **nonlinearity**!

We pick the simplest type of nonlinearity

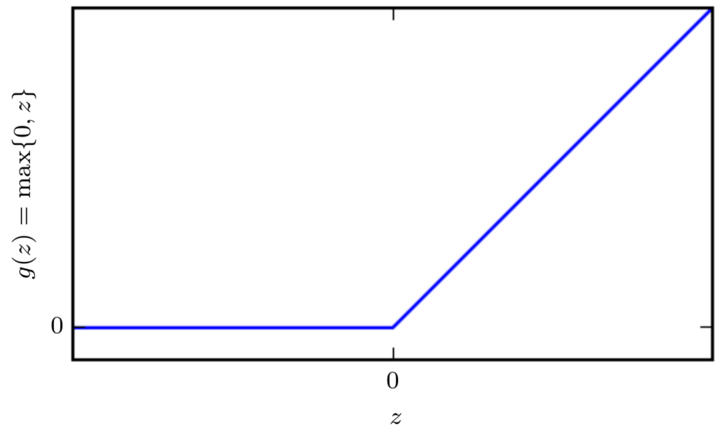
$$\boldsymbol{h} = f^{(1)}(\boldsymbol{x}; W, \boldsymbol{c}) = g(W\boldsymbol{x} + \boldsymbol{c})$$
$$y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b) = \boldsymbol{w}^T\boldsymbol{h} + b$$

The function $g: \mathbb{R} \to \mathbb{R}$ is called an **activation function**, and is applied **element-wise** to a vector

$$g(\boldsymbol{z})_i = g(z_i)$$

Simplest choice of activation:

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



A Solution to the XOR Problem

Neural network model:

$$f(\boldsymbol{x}; W, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max(0, W\boldsymbol{x} + \boldsymbol{c}) + b$$

$x = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow Wx + c = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow f = (1 \quad -2)\begin{pmatrix} 1 \\ 0 \end{pmatrix} + 0 = 1$

$x = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \Rightarrow Wx + c = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow f = 1$

Let

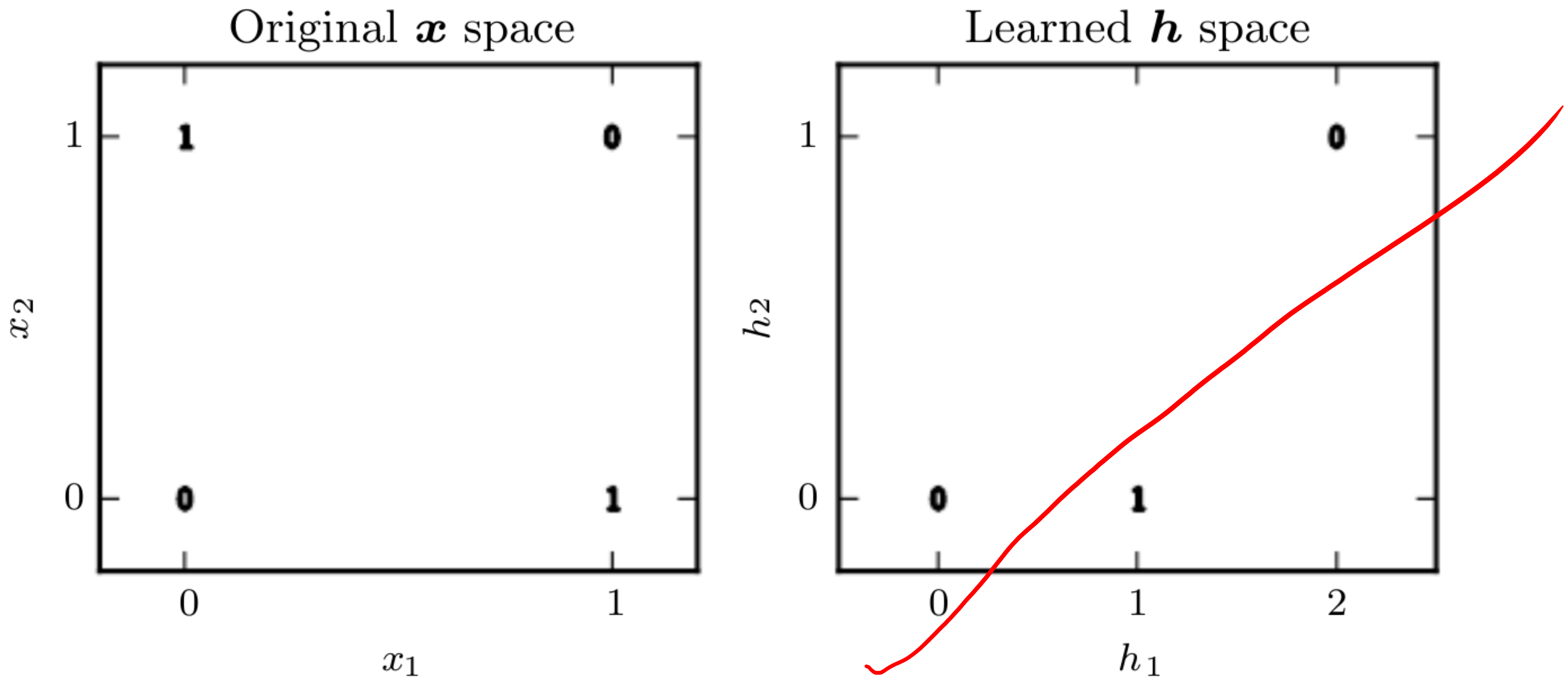
$$W = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad \boldsymbol{c} = \begin{pmatrix} 0 \\ -1 \end{pmatrix} \quad \boldsymbol{w} = \begin{pmatrix} 1 \\ -2 \end{pmatrix} \quad b = 0$$

$x = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \Rightarrow Wx + c = \begin{pmatrix} 0 \\ -1 \end{pmatrix} \Rightarrow f = (1 \quad -2)\begin{pmatrix} 0 \\ 0 \end{pmatrix} + 0 = 0$

Then

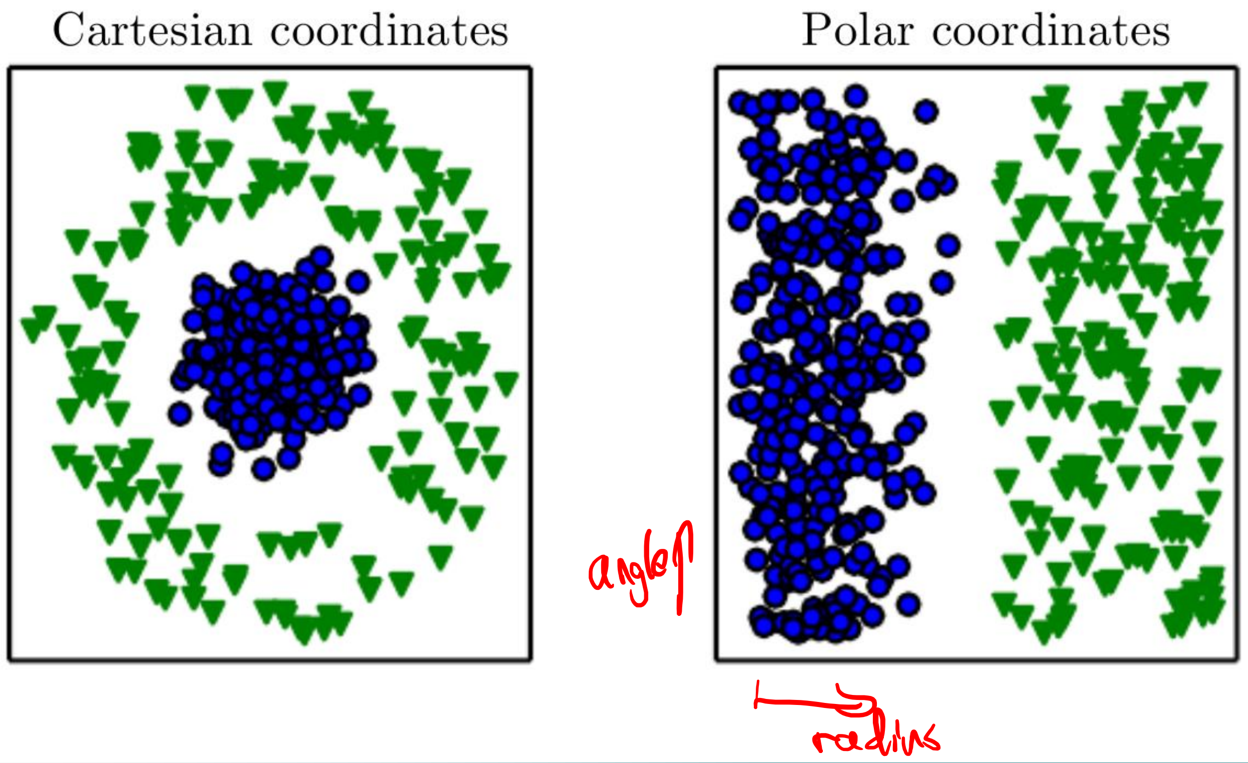
$$f(X; W, \boldsymbol{c}, \boldsymbol{w}, b) = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = f^*(X)$$

What is the role of the hidden units?



Feature Space

$$\begin{tikzcd}\boxed{\text{Input } (\boldsymbol{x})} \arrow[r, "f^{(1)}"] & \boxed{\text{Feature } (\boldsymbol{h})} \arrow[r, "f^{(2)}"] & \boxed{\text{Output } (y)}\end{tikzcd}$$



General Shallow (1-hidden-layer) Neural Networks

General neural networks for regression

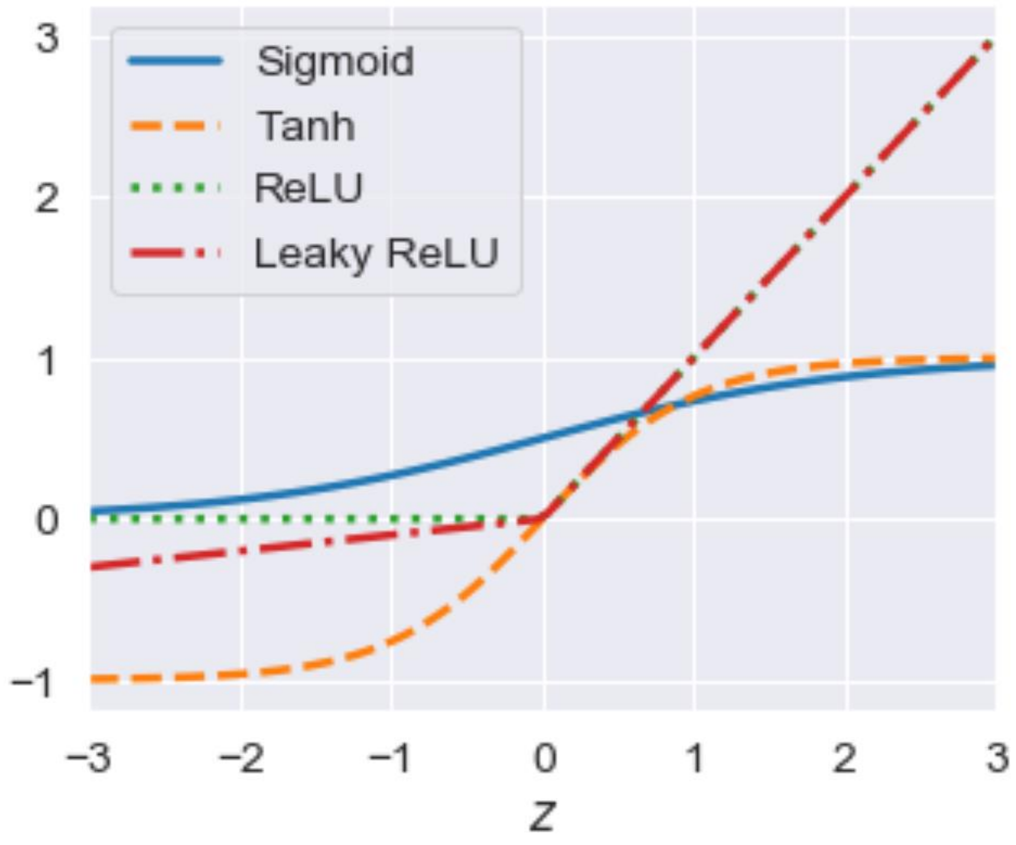
$$f(\boldsymbol{x}; W, \boldsymbol{c}, \boldsymbol{w}, b) = w^T \underbrace{g(W\boldsymbol{x} + \boldsymbol{c})}_{\boldsymbol{\phi}(\boldsymbol{x};\boldsymbol{\theta})} + b$$

Variables: $W \in \mathbb{R}^{m\times d}, \boldsymbol{c} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m, b \in \mathbb{R}$

- W and $\boldsymbol{w}$ are called **weights**
- $\boldsymbol{c}$ and b are called **biases**
- g is the activation function

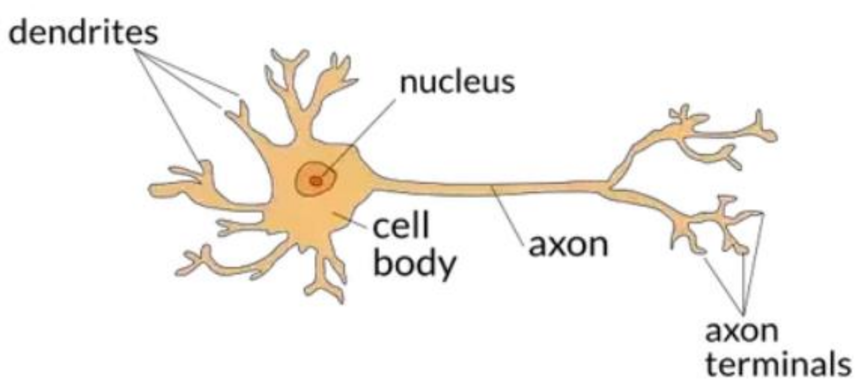
Activation Functions

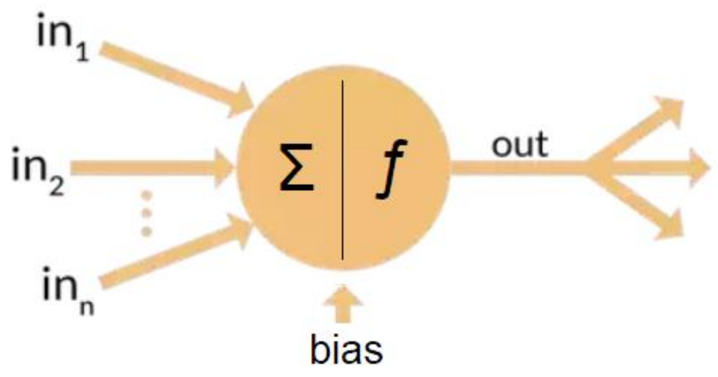
- Rectified Linear Unit (ReLU)
  $$g(z) = \max(0, z)$$
- Sigmoid
  $$g(z) = \frac{1}{1 + e^{-z}}$$
- Tanh
  $$g(z) = \tanh(z)$$
- Leaky-ReLU
  $$g(z) = \begin{cases} z & \text{if } z \geq 0 \\ \delta z & \text{if } z < 0 \end{cases}$$



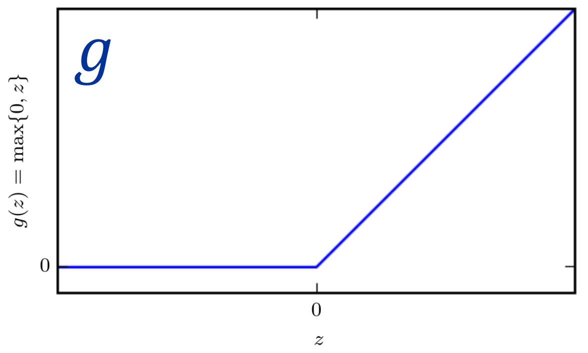
Historical Motivation: Modelling Neurons

Neural networks originated from an attempt to model collective interaction of neurons





Neuron: $g(W_i \cdot \boldsymbol{x} + c_i)$



Picture: https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7

Universal Approximation Theorem

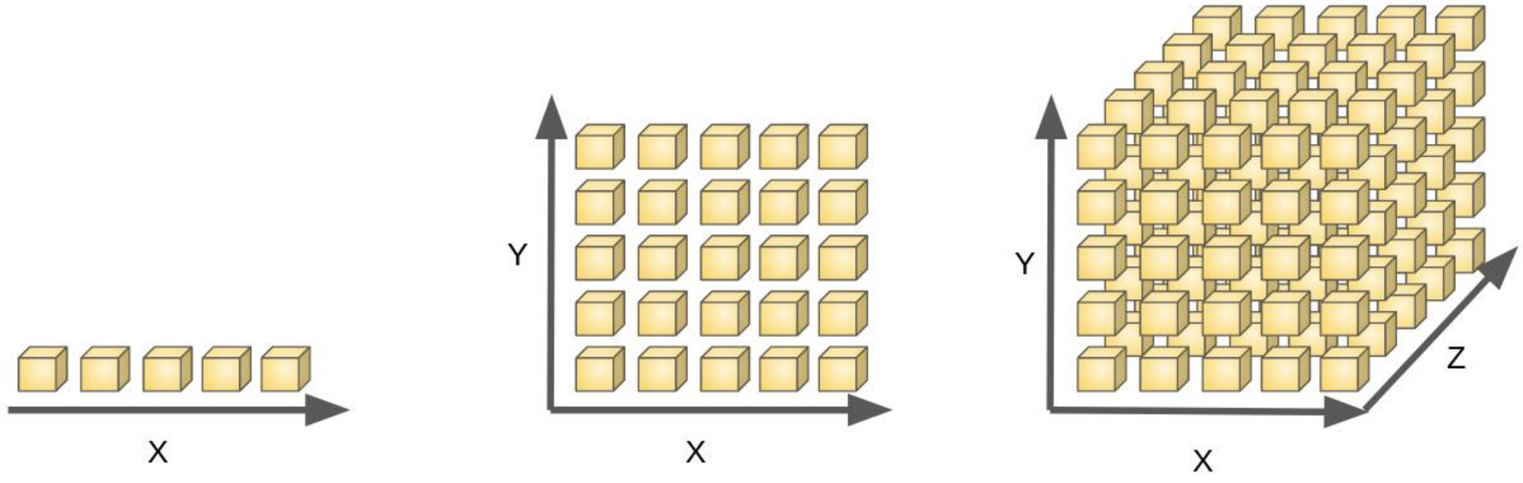
One of the foundational results for neural networks is the **universal approximation theorem**.

In words, it says the following:

__Any__ continuous function f^* on a compact domain can be approximated by neural networks to __arbitrary precision__, provided there are enough neurons (m large enough).

Why NNs over linear basis models?

Curse of Dimensionality



$\{x^j : j = 0,1,\ldots\}$ $\quad$ $\{x_1^{j_1} x_2^{j_2} : j_1, j_2 = 0,1,\ldots\}$ $\quad$ $\{x_1^{j_1} x_2^{j_2} x_3^{j_3} : j_1, j_2, j_3 = 0,1,\ldots\}$

Under some technical assumptions, for any continuous (+ other conditions) function $f^*: [0,1]^d \to \mathbb{R}$, there exists a width-$m$ neural network f_m such that

$$\|f^* - f_m\|^2 \le \mathcal{O}(m^{-1})$$

This result is first proved in [Baron, 1993]

This is a **tremendous** improvement over linear basis models, where we usually have [Jackson, 1912]

$$\|f^* - f_m\|^2 \le \mathcal{O}\left(m^{-\frac{2\alpha}{d}}\right)$$

The constant α measures the smoothness of f^*

Neural Networks for Classification

Neural networks can be modified to handle classification problems, exactly in the same way as linear models

$$f(\boldsymbol{x}; W, \boldsymbol{c}, \boldsymbol{w}, b) = w^T \underbrace{g(W\boldsymbol{x} + \boldsymbol{c})}_{h} + \underset{\downarrow \in \mathbb{R}}{b}$$

$$\Big\Downarrow$$

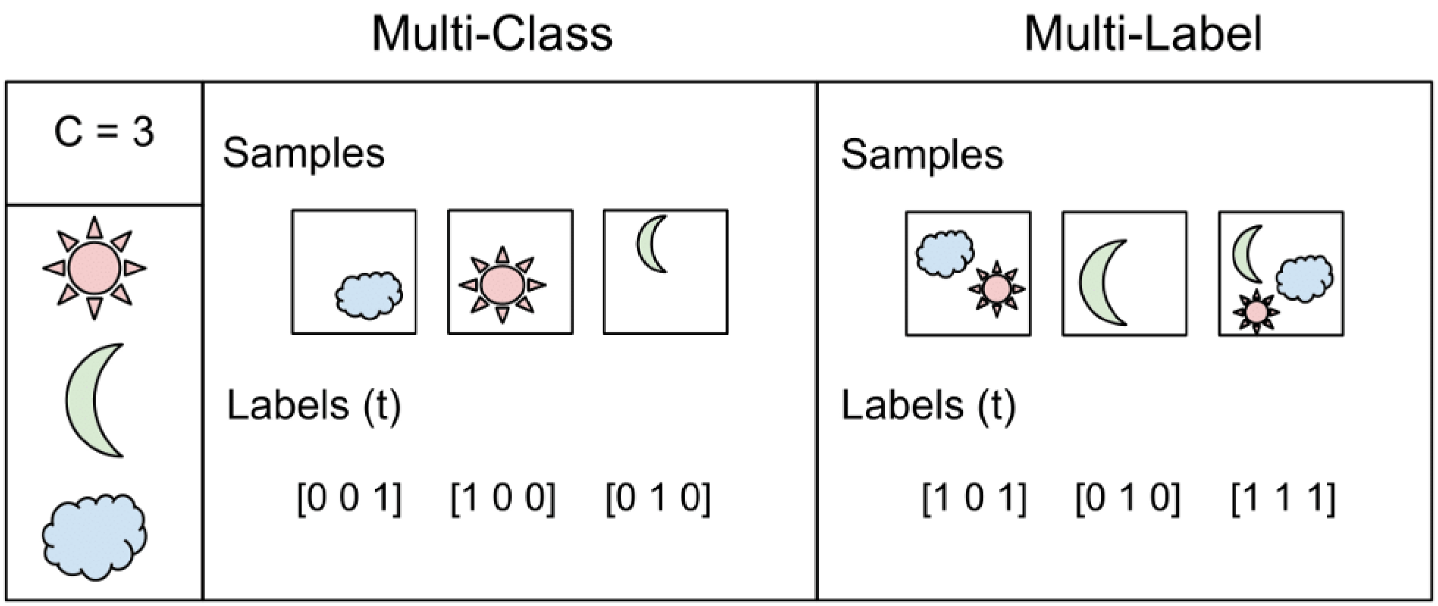
$$f(\boldsymbol{x}; W, \boldsymbol{c}, \boldsymbol{w}, b) = \sigma(V \underbrace{g(W\boldsymbol{x} + \boldsymbol{c})}_{h} + \underset{\downarrow \mathbb{R}^K}{\boldsymbol{b}})$$

$$V \in \mathbb{R}^{K \times m},\ \boldsymbol{b} \in \mathbb{R}^K,\ \sigma = \text{softmax}$$

Choice of Output Units

The soft-max output unit is appropriate for multi-class classification problems

There are also multi-label classification problems…



In multilabel classification, a sigmoid output unit is more appropriate

$$\text{Softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$ $(0.1 \quad 0.1 \quad 0.8)$

$$\text{Sigmoid}(\boldsymbol{z})_i = \frac{1}{1 + \exp(-z_i)}$$ $(0.9 \quad 0.1 \quad 0.8)$

Loss Functions

Empirical risk minimization requires the definition of loss functions

$$R_{\text{emp}}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} L\big(f^*(\boldsymbol{x}_i), f(\boldsymbol{x}_i, \boldsymbol{\theta})\big)$$

For regression problems, we usually pick the square loss (why?)

$$L(y, y') = \frac{1}{2}(y - y')^2$$

Loss Functions for Classification

Multiclass classification problem

- Here, $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ and $\boldsymbol{y}' = f^*(\boldsymbol{x})$ are probability vectors
- Usually pick the **cross-entropy loss**

$$L(\boldsymbol{y}, \boldsymbol{y}') = -\sum_j y_j' \log y_j$$

Multilabel classification problem

- Here, each $\boldsymbol{y}, \boldsymbol{y}'$ is a vector whose coordinates are in $[0,1]$
- Usually pick the **binary cross-entropy loss**

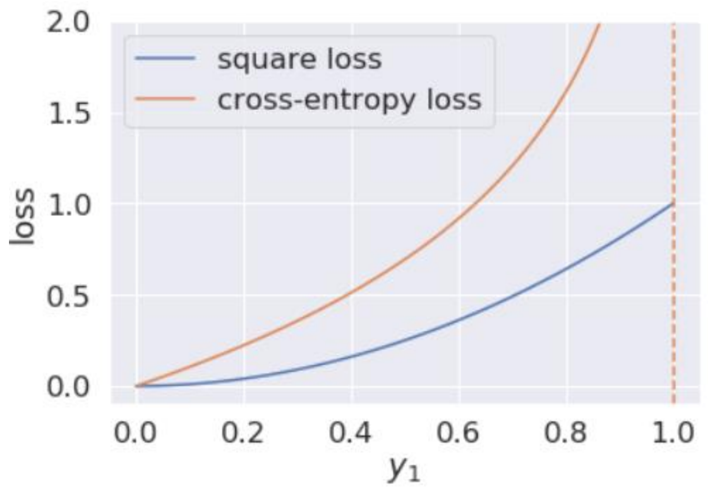
$$L(\boldsymbol{y}, \boldsymbol{y}') = -\sum_j \left(y_j' \log y_j + \left(1 - y_j'\right) \log\left(1 - y_j\right)\right)$$

Why cross-entropy over square?

There are some good reasons to choose cross-entropy over the square loss for classification problems

- Better statistical interpretation
- Better numerical stability

2-Class case:
MSE: $L(y, y') = (y_1 - y_1')^2$
CE: $L(y, y') = -y_1' \log y_1 - (1 - y_1') \log(1 - y_1)$



Surrogate Losses

Importantly, for classification problems, whatever loss you choose, they are often **surrogates** for the true loss, which is the **accuracy**:

$$L(y, y') = \mathbb{I}_{y \neq y'}$$

This is known as the **zero-one loss**.

Why is this usually not used for optimization?

Gradient Based Learning

Empirical Risk Minimization

Essentially, empirical risk minimization is an optimization problem

$$\min_{\boldsymbol{\theta}} R(\boldsymbol{\theta})$$

Recall that in linear regression, we solved the problem by setting

$$\nabla R(\boldsymbol{\theta}) = 0$$

However, very often there are no easy way to solve this equation

Iterative Methods

In this case, we will resort to iterative methods, where we make the loss smaller successively

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + F(\boldsymbol{\theta}_k)$$

And hopefully we achieve

$$R(\boldsymbol{\theta}_{k+1}) \leq R(\boldsymbol{\theta}_k)$$

Gradient Descent as an Iterative Method

Let us consider the optimization problem

$$\min_{\boldsymbol{\theta}} R(\boldsymbol{\theta}) \qquad R\colon \mathbb{R}^p \to \mathbb{R}$$

Given $\boldsymbol{\theta}$, we want to make a small change to $\boldsymbol{\theta}$ in some appropriate direction $\boldsymbol{\phi}$, so that

$$R(\boldsymbol{\theta} + \epsilon \cdot \boldsymbol{\phi}) \leq R(\boldsymbol{\theta})$$

Here, ϵ is a small positive number

Taylor Expansion:

$$R(\boldsymbol{\theta} + \epsilon\boldsymbol{\phi}) = R(\boldsymbol{\theta}) + \epsilon\boldsymbol{\phi}^T \nabla R(\boldsymbol{\theta}) + \mathcal{O}(\epsilon^2)$$

We want the term $\epsilon\boldsymbol{\phi}^T \nabla R(\boldsymbol{\theta})$ to be as negative as possible, so we should take

$$\boldsymbol{\phi} \propto -\nabla R(\boldsymbol{\theta})$$

In other words, we want to push $\boldsymbol{\theta}$ in the negative gradient direction

$$\boldsymbol{\theta} \to \boldsymbol{\theta} - \epsilon\nabla R(\boldsymbol{\theta})$$

The Gradient Descent Algorithm

The direction $-\nabla R$ is called the **steepest descent direction**

This gives rise to a simple iterative algorithm for minimization

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon \nabla R(\boldsymbol{\theta}_k)$$

This is called **Gradient Descent** (GD)

The number ϵ is called the **step size** or the **learning rate**

Can we take ϵ as large as we like?

Example

Consider minimizing a quadratic function

$$R(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|^2$$

$\hat{\theta} = 0$

Gradient descent iterates

$\nabla R(\theta_k)$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon\underset{\uparrow}{\boldsymbol{\theta}_k}$$

$$\boldsymbol{\theta}_k = (1-\epsilon)^k\boldsymbol{\theta}_0$$

Example (Iterative Solution of Linear Regression)

Recall linear regression problem

$$\min_{\boldsymbol{w}} R(\boldsymbol{w}) = \frac{1}{2}\|X\boldsymbol{w} - \boldsymbol{y}\|^2$$

Gradient descent iterates

$$\begin{aligned}\boldsymbol{w}_{k+1} &= \boldsymbol{w}_k - \overbrace{\epsilon X^T(X\boldsymbol{w}_k - \boldsymbol{y})}^{\nabla R} \\ &= (I - \epsilon X^T X)\boldsymbol{w}_k + \epsilon X^T \boldsymbol{y}\end{aligned}$$

Suppose $\boldsymbol{w}_k \to \boldsymbol{w}_\infty$ for some $\boldsymbol{w}_\infty$, what is $\boldsymbol{w}_\infty$?

$$\boldsymbol{w}_\infty = (I - \epsilon X^T X)\boldsymbol{w}_\infty + \epsilon X^T \boldsymbol{y} \;\Rightarrow\; \boldsymbol{w}_\infty = (X^T X)^{-1} X^T \boldsymbol{y}$$

Convergence Analysis (Optional)

When does $\boldsymbol{w}_k \to \boldsymbol{w}_\infty$?

To see this, we can rewrite the GD iterations as

$$\boldsymbol{w}_{k+1} - \boldsymbol{w}_\infty = A(\boldsymbol{w}_k - \boldsymbol{w}_\infty) \quad \text{where} \quad A = (I - \epsilon X^T X)$$

Denote by $\boldsymbol{e}_k = \boldsymbol{w}_k - \boldsymbol{w}_\infty$ the error vector, then

$$\boldsymbol{e}_k = A^k \boldsymbol{e}_0$$

Let $\{\lambda_1(A), \dots, \lambda_d(A)\}$ be the real eigenvalues of A (why real?)

Then, if $\lambda_i(A) < 0$ for all i,

$$\boldsymbol{e}_k = A^k \boldsymbol{e}_0 \to 0$$

This is ensured by $\epsilon \le \frac{1}{\max_i \lambda_i(X^T X)}$ (Is this necessary?)

Suppose we take

$$\epsilon = \frac{1}{\max_i \lambda_i(X^T X)}$$

Then,

$$\|\boldsymbol{e}_k\| \leq \left(1 - \frac{1}{\kappa(X^T X)}\right)^k \|\boldsymbol{e}_0\|$$

The quantity

$$\kappa(X^T X) := \frac{\max_i \lambda_i(X^T X)}{\min_i \lambda_i(X^T X)} \geq 1$$

is called the **condition number** of $X^T X$. The bigger the condition number, the slower the convergence.

Convergence for General Losses

In general, one can prove that if ∇R is Lipschitz and $\nabla^2 R$ is uniformly bounded, then for small enough ϵ we have

$$\|\nabla R(\boldsymbol{\theta}_k)\| \to 0 \qquad \text{as} \qquad k \to \infty$$

Does this mean that $\boldsymbol{\theta}_k \to \boldsymbol{\theta}_\infty$ for some $\boldsymbol{\theta}_\infty$?

Steepest Descent to Optimize Neural Networks

Recall that the empirical risk minimization problem for NNs can be cast as an optimization problem

$$\min_{W,\boldsymbol{c},\boldsymbol{w},b} R(W,\boldsymbol{c},\boldsymbol{w},b) = \sum_{i=1}^{N} L\big(f^*(\boldsymbol{x}_i), f(\boldsymbol{x}_i; W,\boldsymbol{c},\boldsymbol{w},b)\big)$$

And hence can be solved* iteratively by gradient descent

$$W_{k+1} = W_k - \nabla_W R(W_k,\boldsymbol{c}_k,\boldsymbol{w}_k,b_k)$$
$$\vdots$$

Demo: Building Neural Networks in Keras

Tensorflow

Tensorflow is a open source deep learning framework developed by Google.

Main features

- Automatic differentiation
- Same code can be run on GPU (CUDA)
- Deployable to all sorts of devices (e.g. mobile phones)
- Eager execution starting from Tensorflow v2.0

Further Information and Tutorials

https://www.tensorflow.org/

TensorFlow

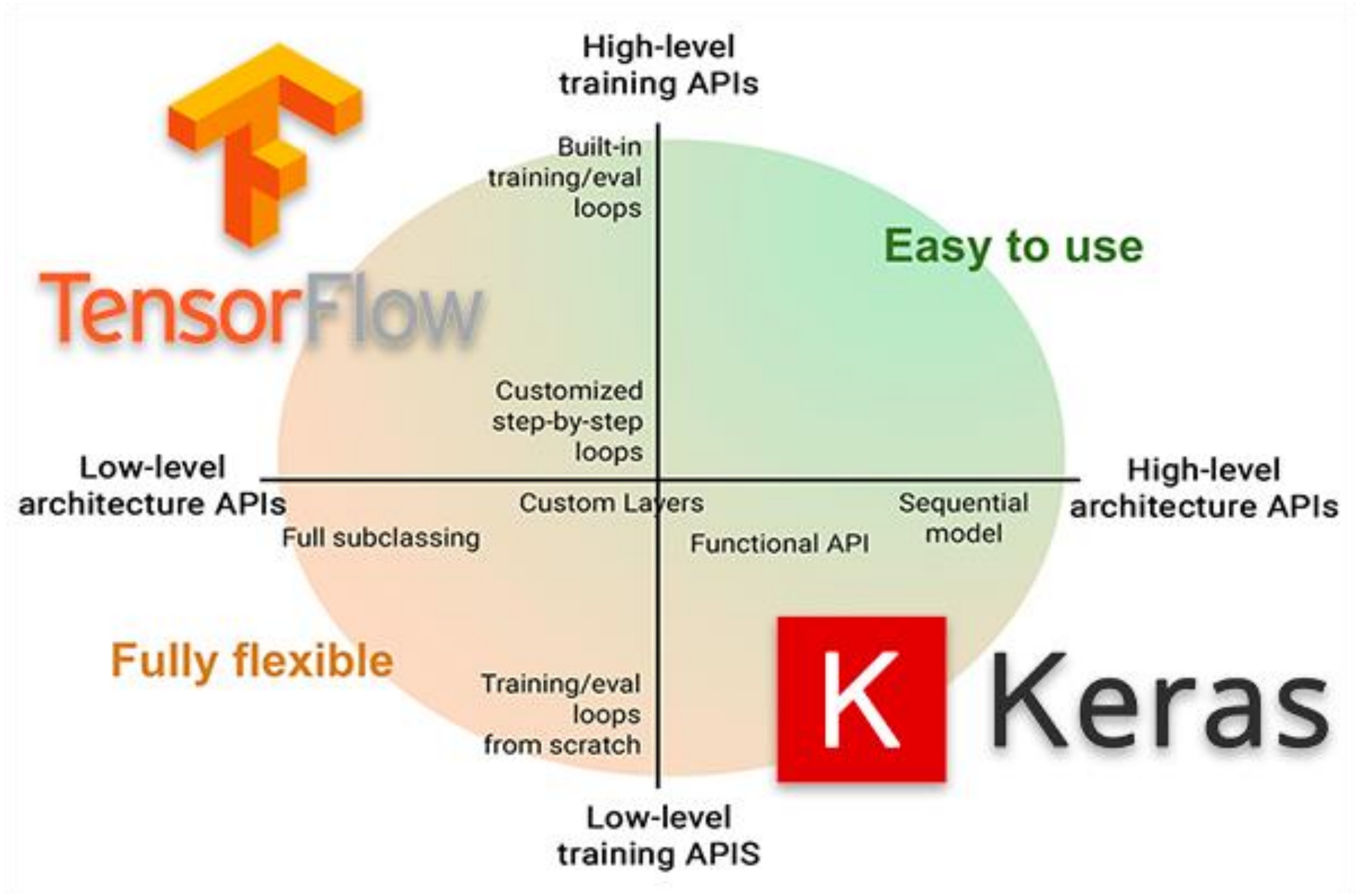
Keras

Keras is a high-level neural network API that makes model building and prototyping easier

- Uses Tensorflow (or CNTK, Theano) backend
- High-level abstraction (layers, models)
- “Disadvantage”: may need some lower level (tensorflow) knowledge to do some meaningful tinkering

K Keras

Keras vs Tensorflow



https://www.pyimagesearch.com/2019/10/21/keras-vs-tf-keras-whats-the-difference-in-tensorflow-2-0/

Alternatives

The main alternative to the tensorflow/keras framework is **pytorch**

- Similar API and behavior to numpy
- Flexible for model and algorithm development
- See: https://pytorch.org/

PYTORCH

Computational Resources

For your project, you may require GPU computing resources for application-heavy type of problems

There are two sources for computational resources available to you:

- University HPC cluster (https://nusit.nus.edu.sg/hpc/)
- National Supercomputing Center (NSCC) (https://www.nscc.sg)