Bios 740: Deep Learning Methods for Biomedical Applications with Pytorch

Hongtu Zhu, Ph.D. Professor of Biostatistics

Course Overview

Description: This course provides an in-depth exploration of deep learning methods applied to biomedical data using PyTorch. It covers foundational neural network architectures, advanced models like convolutional neural networks (CNNs), recurrent neural networks (RNNs), transformers, and BioBERT, with applications in medical imaging, genomic data analysis, and disease prediction.

Goals:

- Develop expertise in deep learning methodologies tailored for biomedical applications.
- Gain hands-on experience implementing, training, and evaluating models using PyTorch.
- Tackle real-world challenges through project work and interactive discussions.

Instructor:

Dr. Hongtu Zhu (Email: htzhu@email.unc.edu), a renowned expert in biostatistics, biomedical AI, and medical imaging analysis.

Teaching Assistant:

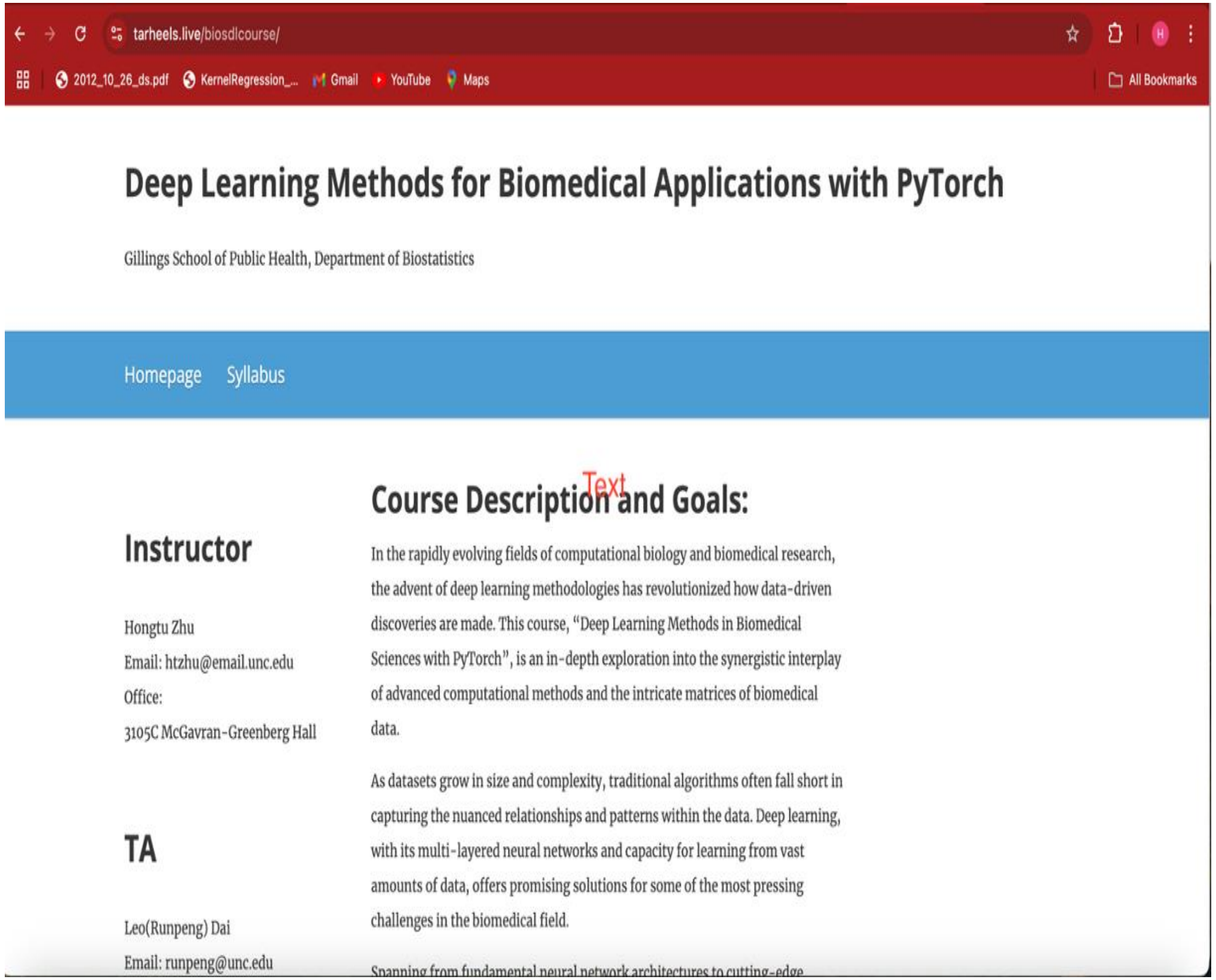
Mr. Runpeng Dai (Email: runpeng@unc.edu)

Course Format and Structure

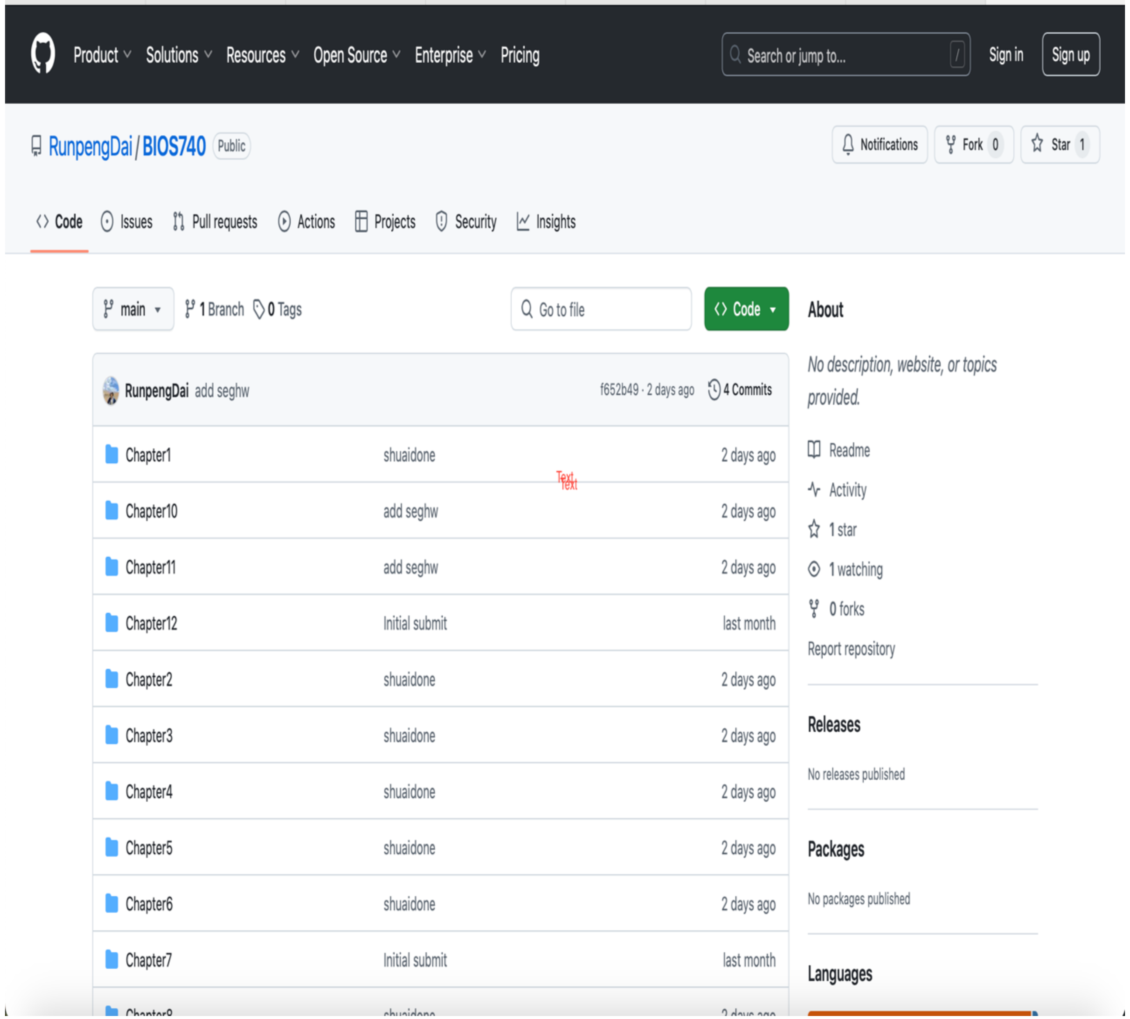
- **Schedule:** Tuesdays and Thursdays, 9:30–10:45 AM.
- **Location:** 1304 MCG.
- **Format:** Seminar-style sessions with supplemental readings, interactive discussions, and case studies.
- **Prerequisites:** Basic Python programming; familiarity with linear algebra, probability, and calculus is recommended.
- **Grading:**
    - Homework Assignments: 75% (5 assignments, each worth 15%).
    - Final Project: 25%.
    - No midterms or final exams.
- **Help Sessions**: Weekly help sessions are available to support students with homework, project work, and technical questions related to PyTorch and deep learning concepts.
- **Office Hours:** Friday 12:00pm-1:00pm at 3105C MCG.
- **AI Policy:** Generative AI is allowed, provided its usage is documented.

Course Websites

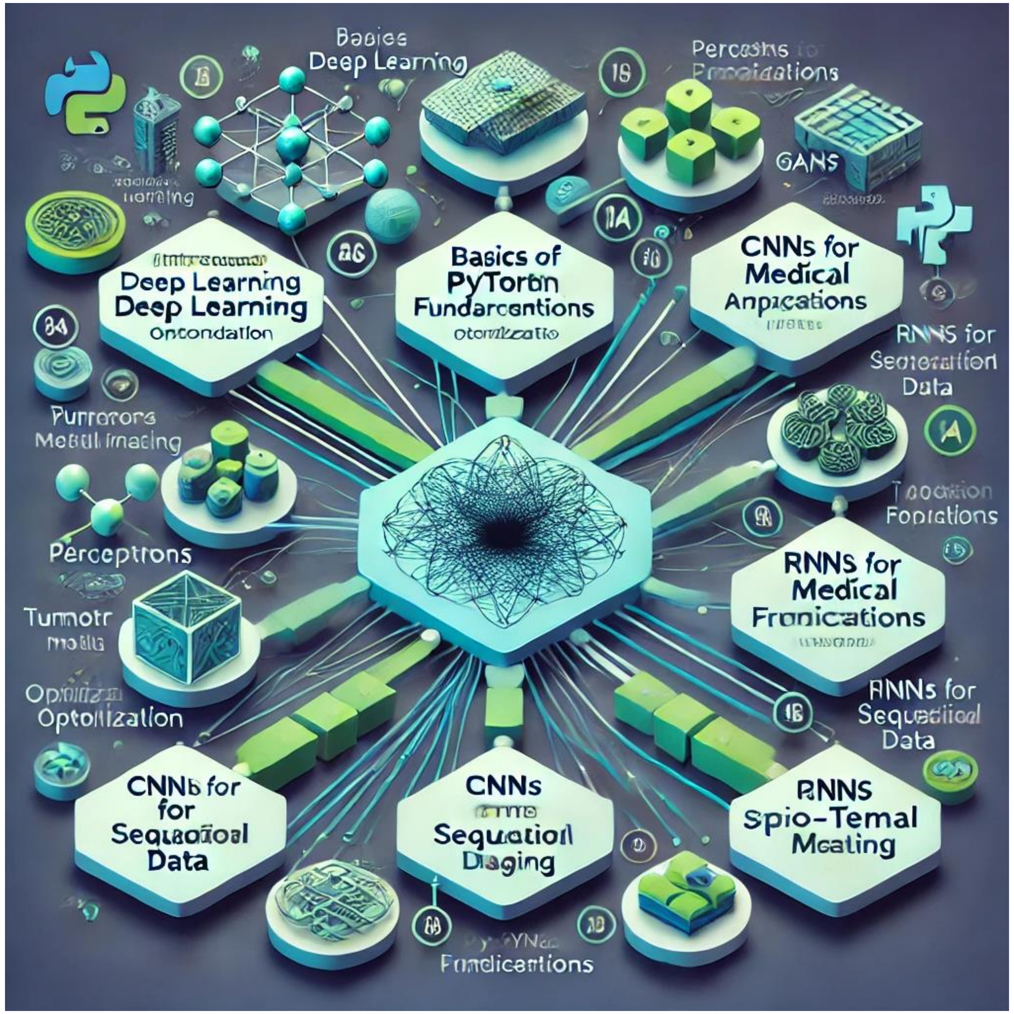
https://tarheels.live/biosdlcourse/



https://github.com/RunpengDai/BIOS740



Key Modules



Introduction: Basics of deep learning, supervised/ unsupervised learning, and PyTorch fundamentals.

1. Neural Networks: Perceptrons, optimization techniques, and activation functions.

2. Advanced Topics:

- CNNs.
- RNNs and LSTMs
- GANs/ Diffusion Models
- Transformers
- BioBERT.

3. Applications: Segmentation, Registration, Tumor localization, Disease spread prediction, Biomedical text mining, and Drug discovery.

Final Project

Objective: Apply deep learning concepts and tools to address a biomedical challenge. Projects can follow one of two tracks:

- **Applications Track:** Solve a practical problem using deep learning models.
- **Models Track:** Develop or improve a deep learning model for biomedical tasks, potentially leading to publishable work.

Deliverables:

- **Report:** 4 pages, adhering to the provided template.
- **Submission:** Final project report and supplementary materials (e.g., source code, visualizations).
- **Structure:**
    - Title, Abstract, Introduction, Related Work, Data, Methods, Experiments, Conclusion, Writing/Formatting.

For more details, visit the course website or homework repository. Let me know if you'd like additional refinements!

Bios 740- Chapter 1. Introduction to Deep Learning and Computing Resources

Acknowledgement: Thanks to Miss Jiarui Tang and Mr. Runpeng Dai for preparing some of the slides.

Content

1 Introduction to Deep Learning

2 Introduction to PyTorch

3 Introduction to UNC Research Computing Resources

4 Introduction to Basic Algorithm

Content

1 Introduction to Deep Learning

2 Introduction to PyTorch

3 Introduction to UNC Research Computing Resources

4 Introduction to Basic Algorithm

Deep Learning

Deep

Many hidden layers

Learning

Supervised, semi-supervised, unsupervised
Learn adaptive parameters

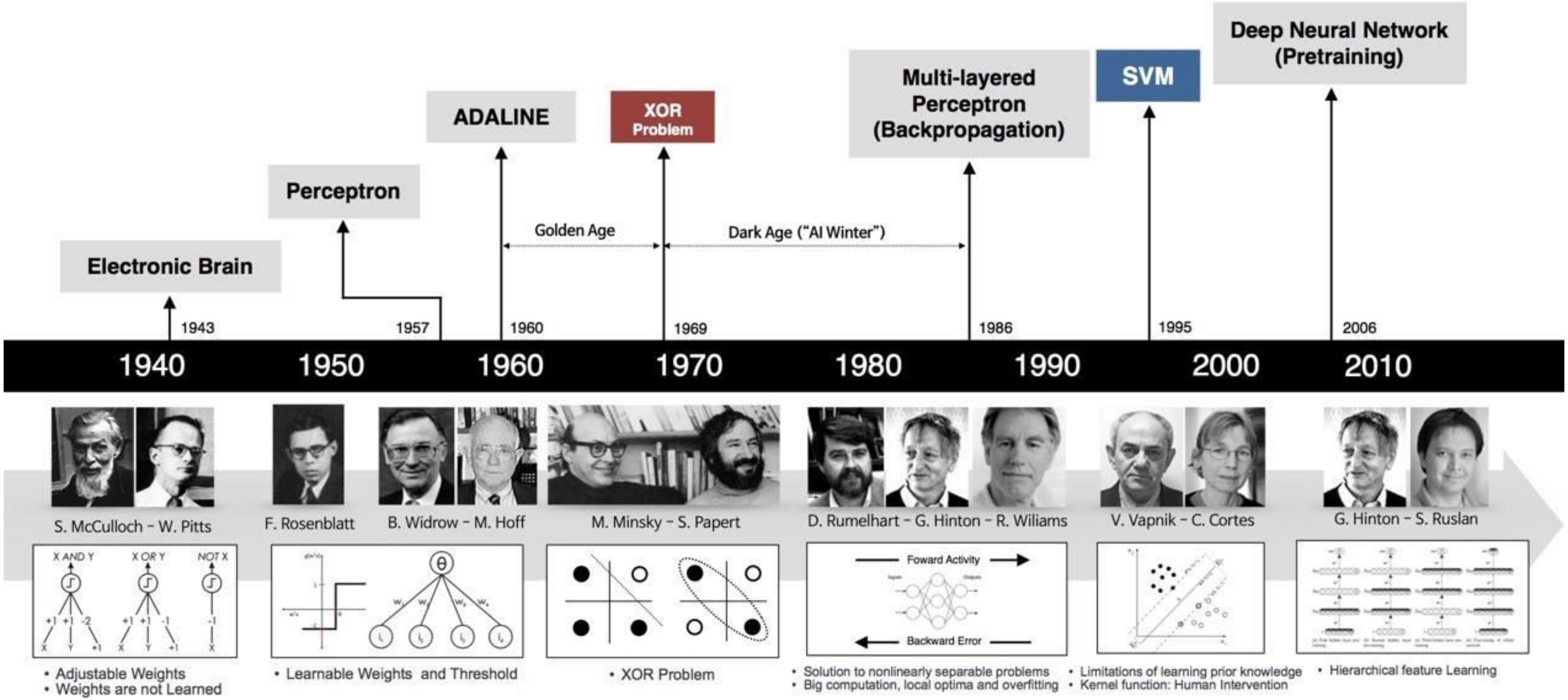
- Use a cascade of multiple layers of nonlinear processing units for feature extract and transformation
- Learn in supervised and/or unsupervised manner
- Learn representations in different level of abstraction

Why popular?

- Chip processing ability
- Increased size of data for training
- Advances in machine learning and signal/information researches

→ Deep models to efficiently exploit complex, compositional nonlinear functions to learn distributed and hierarchical feature representations, to make best use

Historical Summary



Deep Learning Explosion

Downloaded from the NSF website and the medium.com

Deep Learning Platforms

TensorFlow

Chainer

Microsoft CNTK

PaddlePaddle

PYTORCH

theano

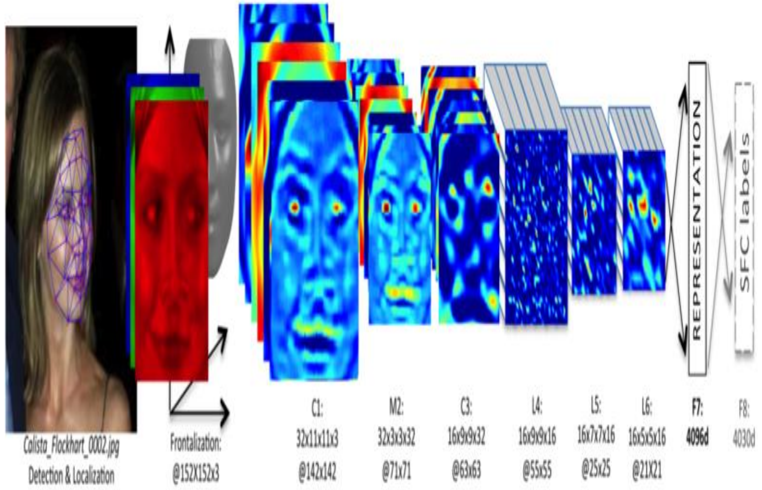
Keras

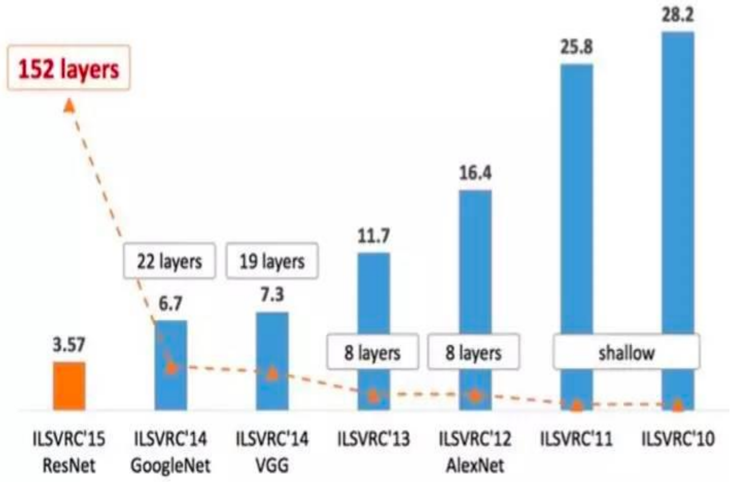
Caffe2

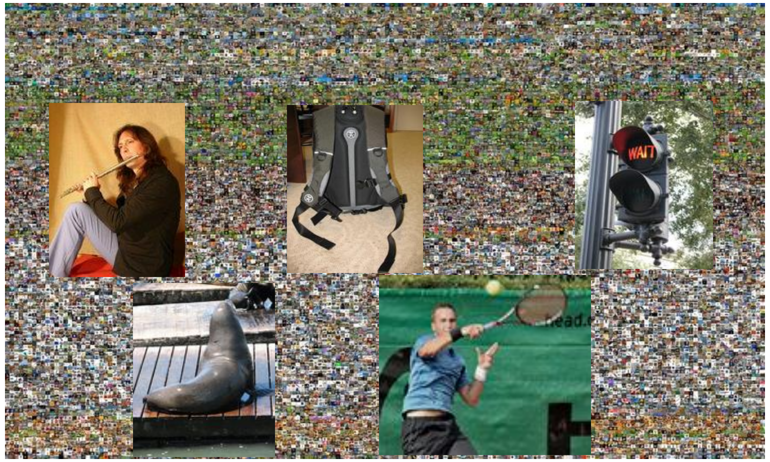
APACHE mxnet

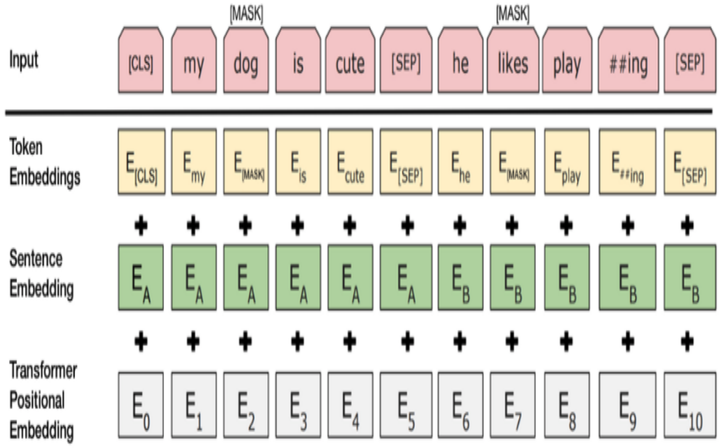
Applications - Vision

IMAGENET

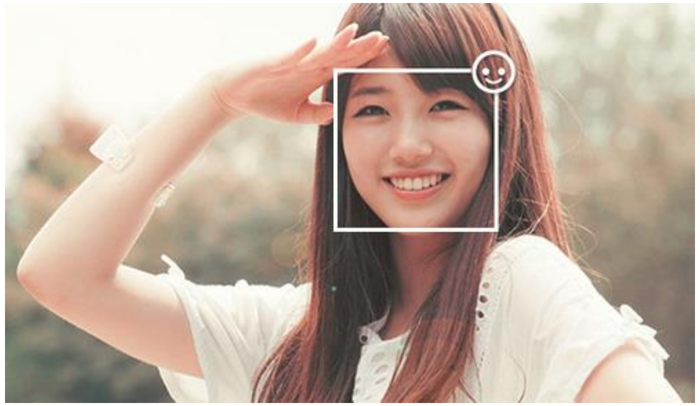


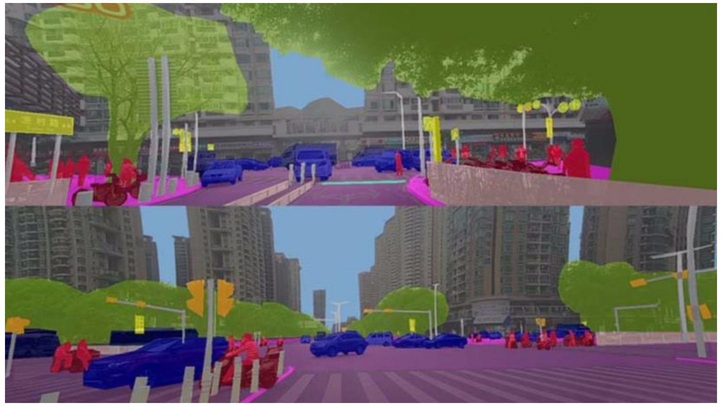


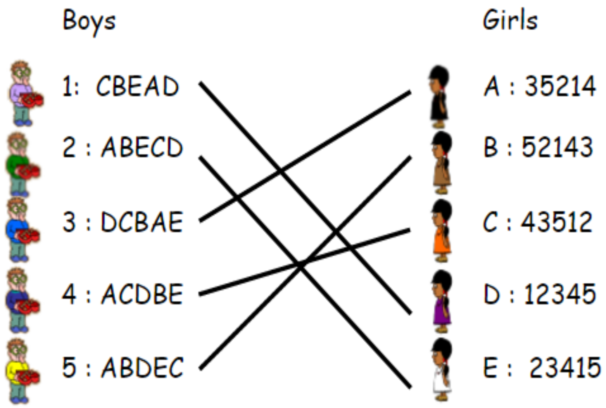




Applications - Vision







Applications - Vision

Disease Detection in Healthcare and Medicine

Deep learning can be utilized for early and more accurate detection of diseases like cancer, Alzheimer's, and heart diseases through image analysis.

High quality image generalization

DALL·E 2

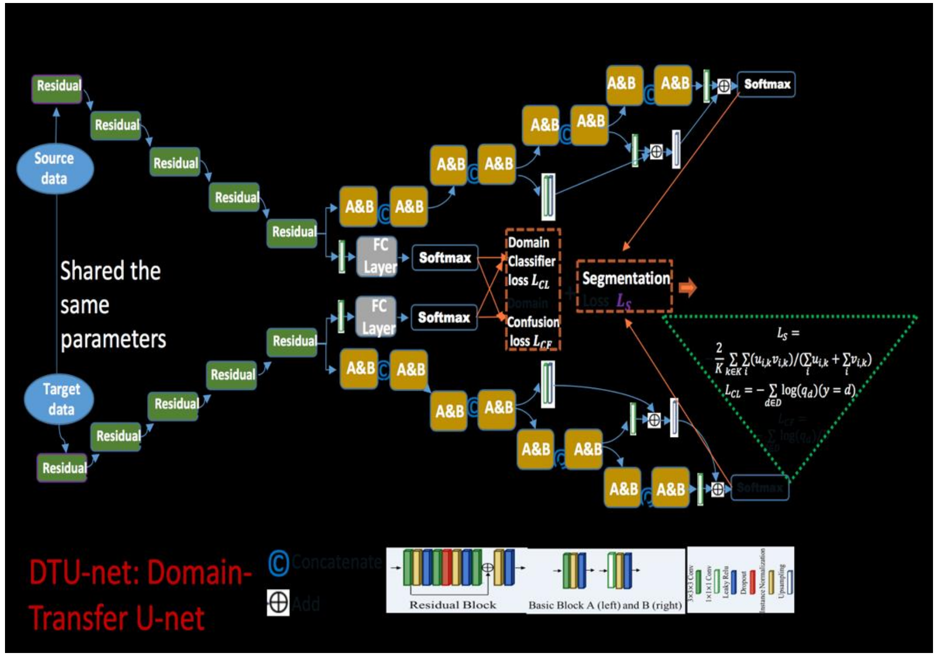
“a teddy bear on a skateboard in times square”

[“Hierarchical Text-Conditional Image Generation with CLIP Latents”](#)
Ramesh et al., 2022

Applications – Medical Imaging

Segmentation Annotation

U-Nets



Liu, Q., Xu, Z., Bertasius, G., & Niethammer, M. (2023). SimpleClick: Interactive Image Segmentation with Simple Vision Transformers. ICCV., 22290-22300. 2023.

Azad *et al.*, "Medical Image Segmentation Review: The success of U-Net." arXiv, Nov. 27, 2022.
Minaee, Shervin, et al. "Image segmentation using deep learning: A survey." *IEEE PAMI* 44.7 (2021): 3523-3542

Application - Language

Language Translation in Natural Language Processing

Deep learning enhances real-time, accurate translation of languages, as seen in tools like Google Translate. The following picture shows the translation of a webpage from English to Chinese.

Application - Language

Large language models

Large language models can perform various tasks such as answering questions, generating creative content, summarizing text, translating languages, and engaging in conversations. It's designed to understand and generate text in a coherent and contextually relevant manner.

Application - Decision

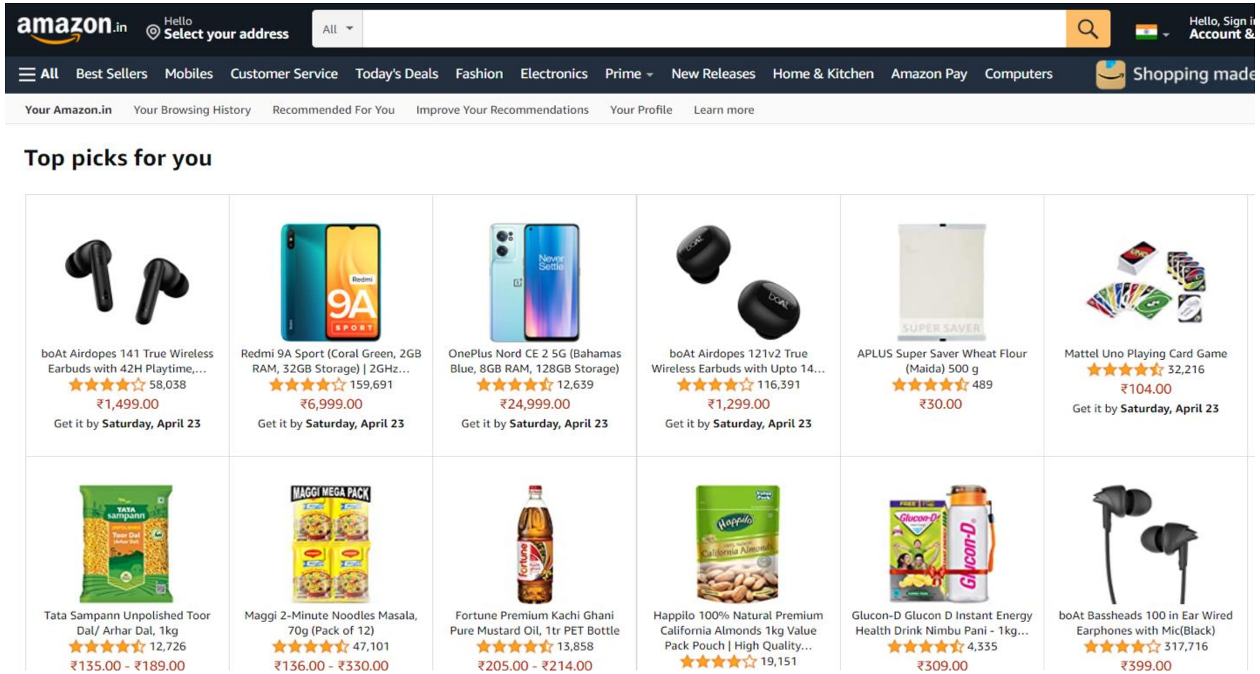
March 2016, AlphaGo made headlines by defeating Lee Sedol.

Reinforcement learning methods have shown priority in video games.

Applications - more

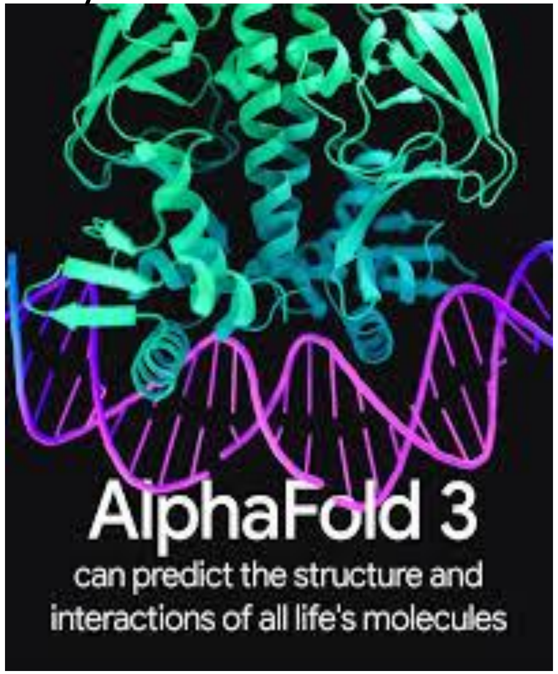
Personalized Shopping Experience in Retail and E-Commerce

Deep learning is leveraged to provide personalized recommendations and targeted advertising to customers based on their shopping behavior.



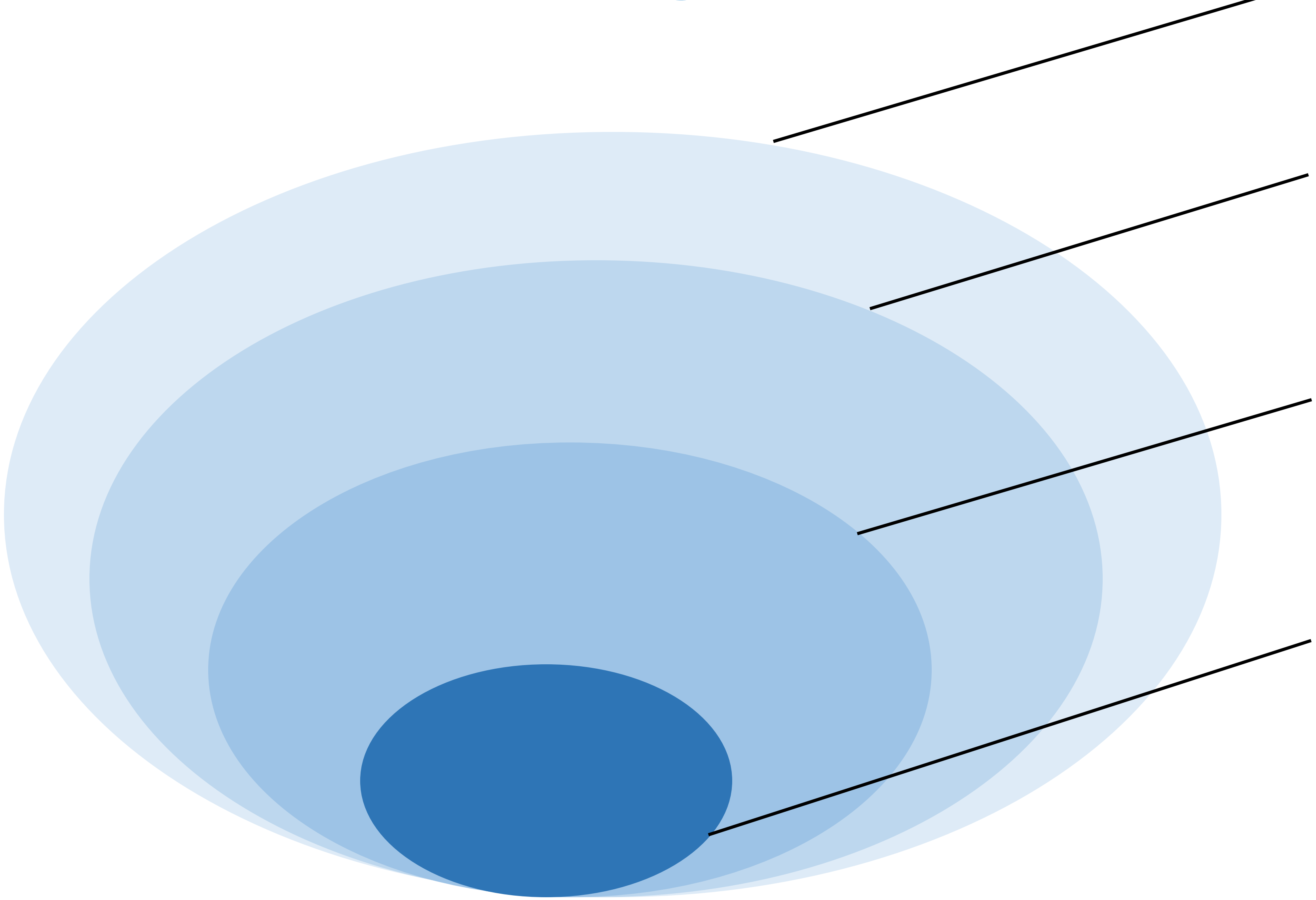
Predict the folding and 3D structure of protein

AlphaFold aims to solve the protein folding problem, which involves predicting a protein's three-dimensional structure based solely on its amino acid sequence. Understanding protein structures is crucial for biological research and drug discovery.



What Exactly is Deep Learning?

Key Terminologies



Artificial Intelligence

Simulates human intelligence in machines for tasks like decision-making and language translation.

Machine Learning (ML)

A subset of AI where algorithms learn from data to make predictions or decisions without being explicitly programmed for each scenario.

Deep Learning (DL)

A branch of machine learning using multi-layered neural networks, effective in processing large amounts of unstructured data like images and speech.

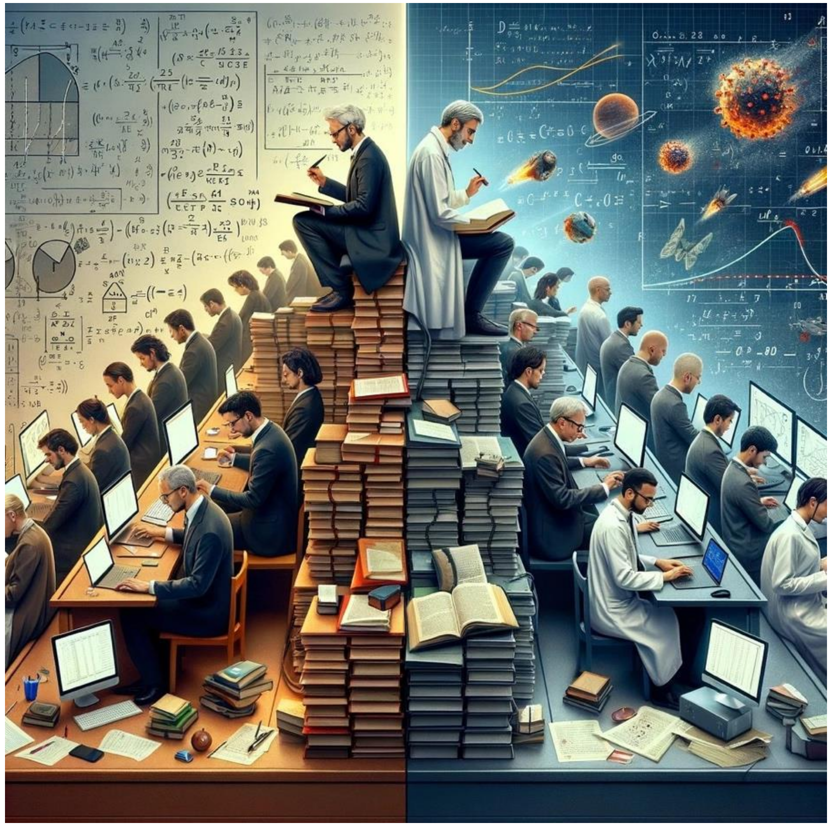
Generative AI

AI algorithms that generate new, original content (like text or images) based on existing data, using techniques like Generative Adversarial Networks (GANs).

Statistical Modeling: The Two Cultures

Statistics is the discipline that concerns the collection, organization, analysis, interpretation, and presentation of data. **https://en.wikipedia.org/wiki/Statistics**

Leo Breiman (2001). *Statistical Science*. There are two cultures in the use of statistical modeling to reach conclusions from data. One assumes that the data are generated by a given stochastic data model. The other uses algorithmic models and treats the data mechanism as unknown. The statistical community has been committed to the almost exclusive use of **data models**. This commitment has led to **irrelevant theory, questionable conclusions, and has kept statisticians from working on a large range of interesting current problems.** **Algorithmic modeling**, both in theory and practice, has developed rapidly in **fields outside statistics**. It can be used both on large complex data sets and as a more accurate and informative alternative to data modeling on smaller data sets. *If our goal as a field is to use data to solve problems, then we need to move away from exclusive dependence on data models and adopt a more diverse set of tools."*



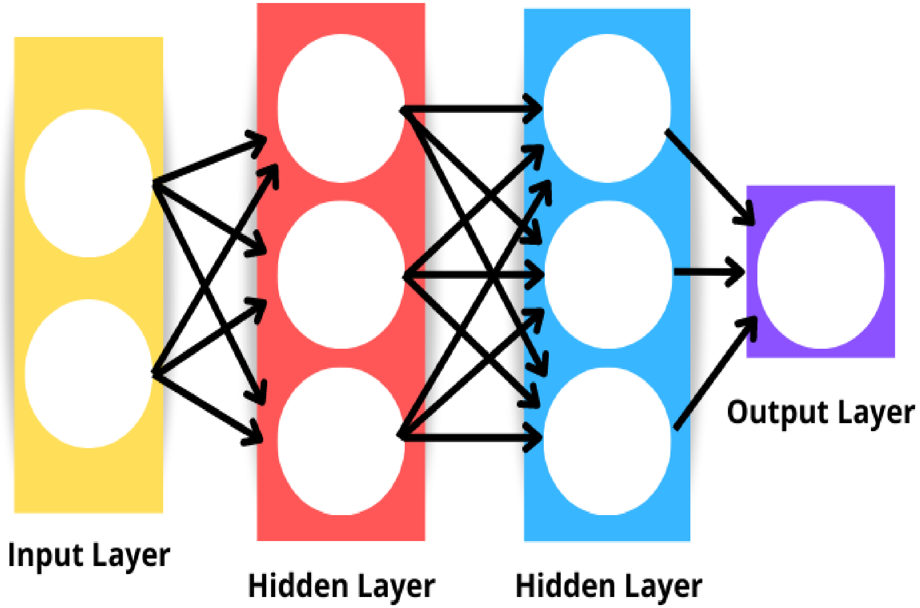
Deep Learning

- Deep learning is a subset of machine learning that focuses on training **algorithmic neural networks** to perform tasks. Its algorithms were inspired by the working of the human brain.
- It's characterized by the use of **multiple layers (deep architectures)** that allow networks to learn hierarchical representations of data and to learn to complete specific tasks.
- In contrast to traditional machine learning/data models, which often requires manual feature extraction, deep learning can automatically learn features from raw data, which you can think of as patterns that occur within the data.
- Deep learning can be used for supervised, unsupervised, self-supervised, semi-supervised, generative, contrastive, few-shot, as well as reinforcement learning.

Objective: teaching computer how to **learn a task** directly from **raw data**

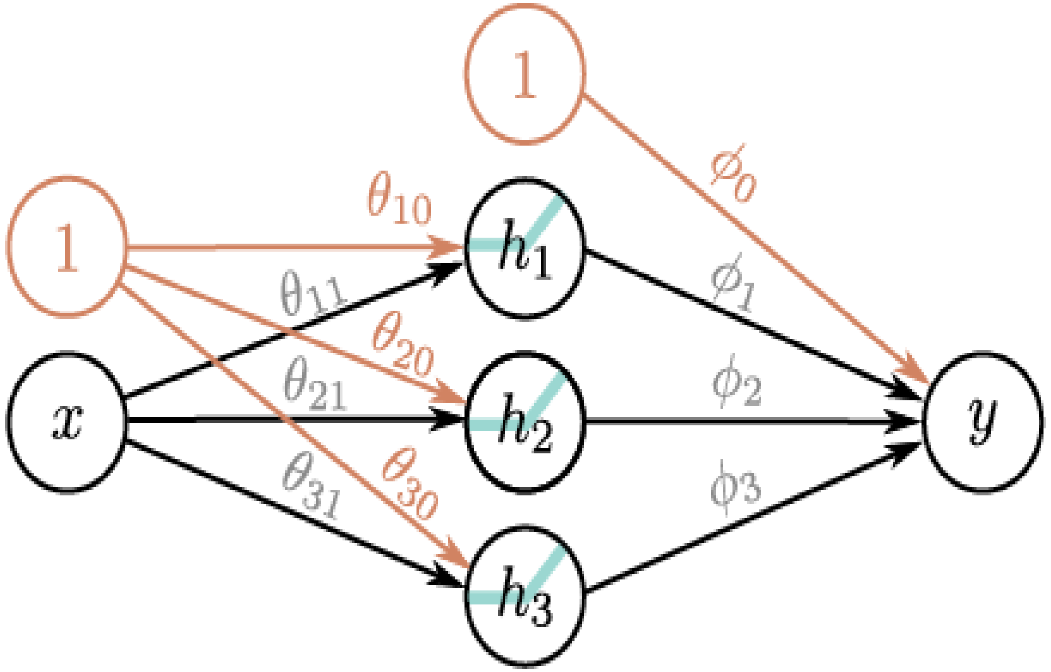
Backbone of DL - Neural Networks

- Neural networks, also called artificial neural networks (ANNs) or simulated neural networks (SNNs), are a **subset of machine learning** and are the **backbone of deep learning algorithms**.
- The neural network is inspired by the human brain's interconnected neurons. They are called "neural" because they mimic how neurons in the brain signal one another.
- It consists of layers: an **input layer**, one or more **hidden layers**, and an **output layer**.
- The **"deep"** in deep learning refers to the depth of layers in a neural network.
- Usually, a neural network of more than three layers, including the inputs and the output, can be considered a deep-learning algorithm.



Further details on neural networks will be in upcoming courses.

Deep Learning Basics



Neurons (Nodes) receive input signals and perform computations and produce an output.

Channels (connections) are associated with a **weight** value that determines the strength of the connection.

Bias is conceptually similar to the intercept in linear regression, accounting for potential deviations from the ideal relationship between inputs and outputs.

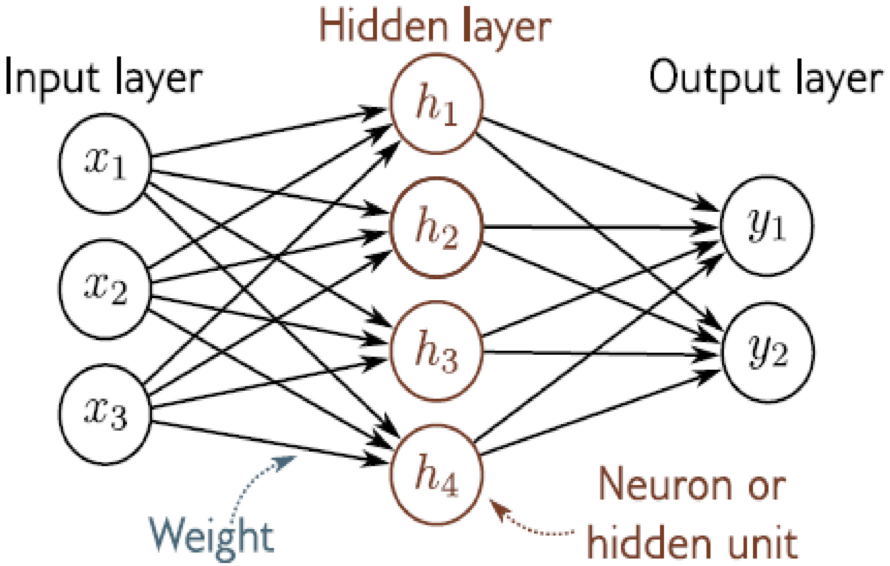
Activation function are threshold values that introduce non-linearities into the neural network, determining if the particular neuron will get activated or not.

Shallow Neural Network

Universal Approximation Theorem

Cybenko (1989) and Hornik (1991)

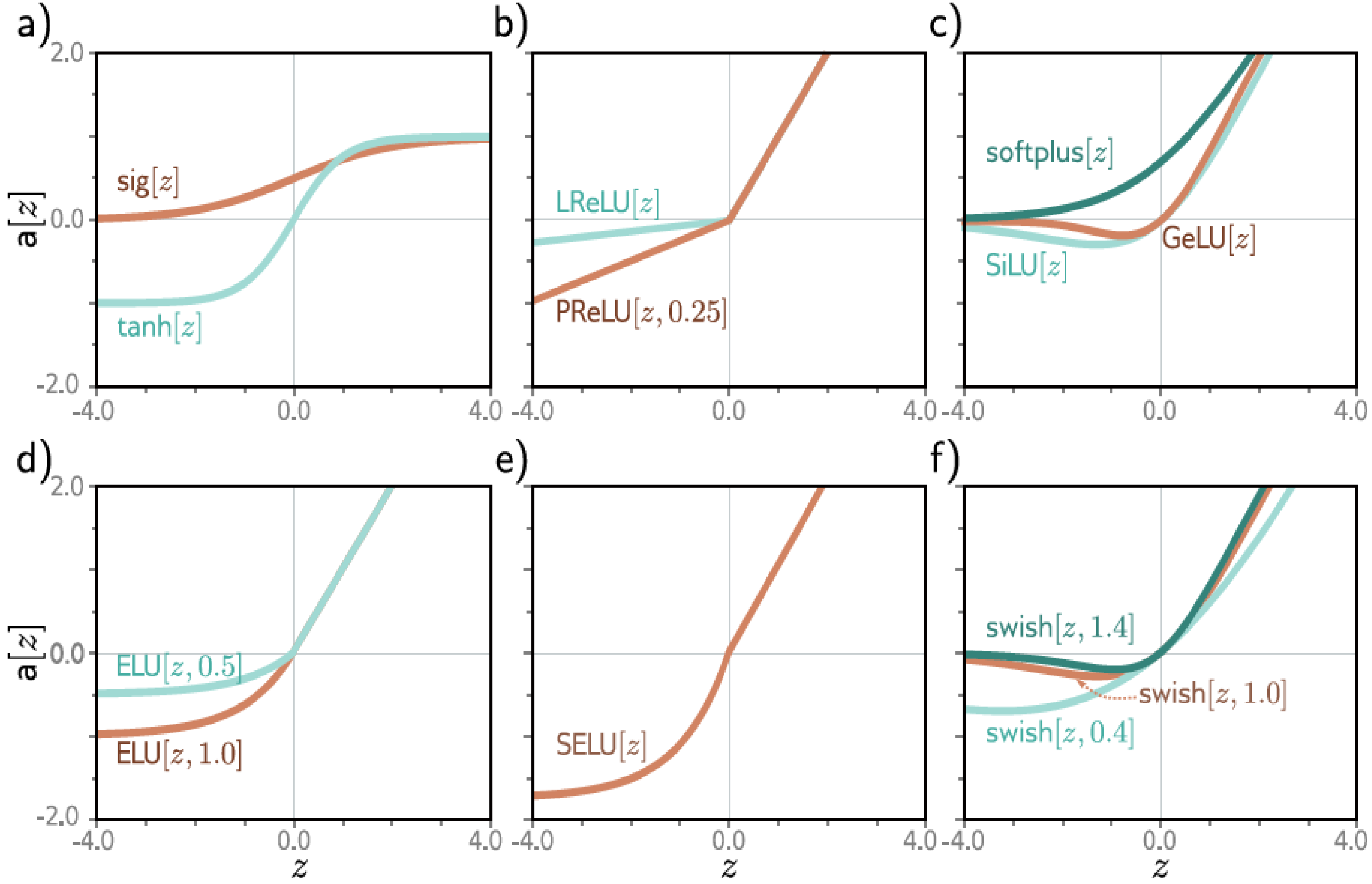
A feed-forward network with **a single hidden layer** containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function.



$$h_d = \mathrm{a}\left[\theta_{d0} + \sum_{i=1}^{D_i} \theta_{di} x_i\right],$$

$$y_j = \phi_{j0} + \sum_{d=1}^{D} \phi_{jd} h_d,$$

Activation Functions

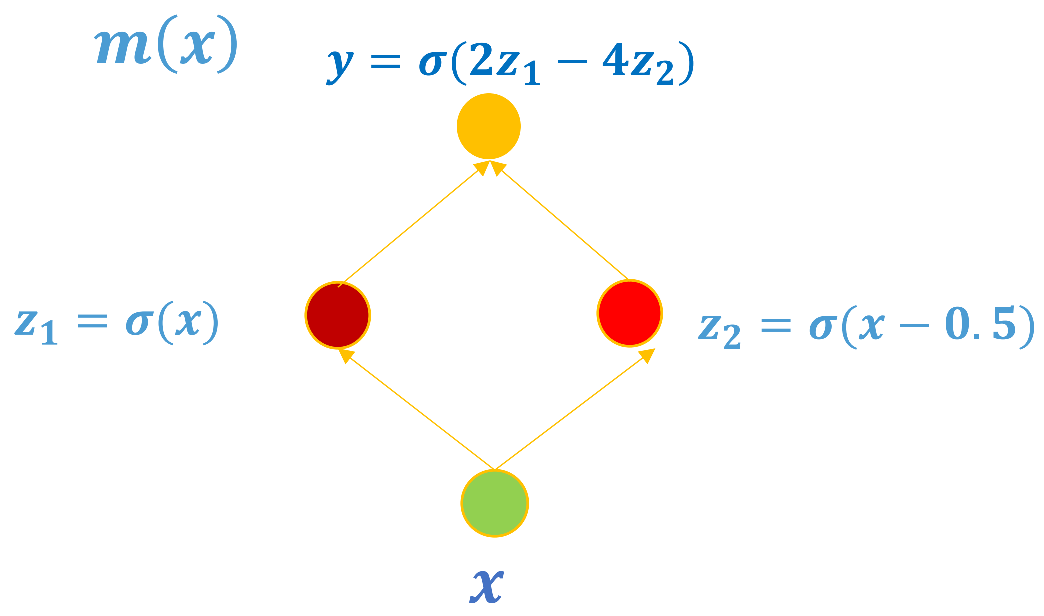


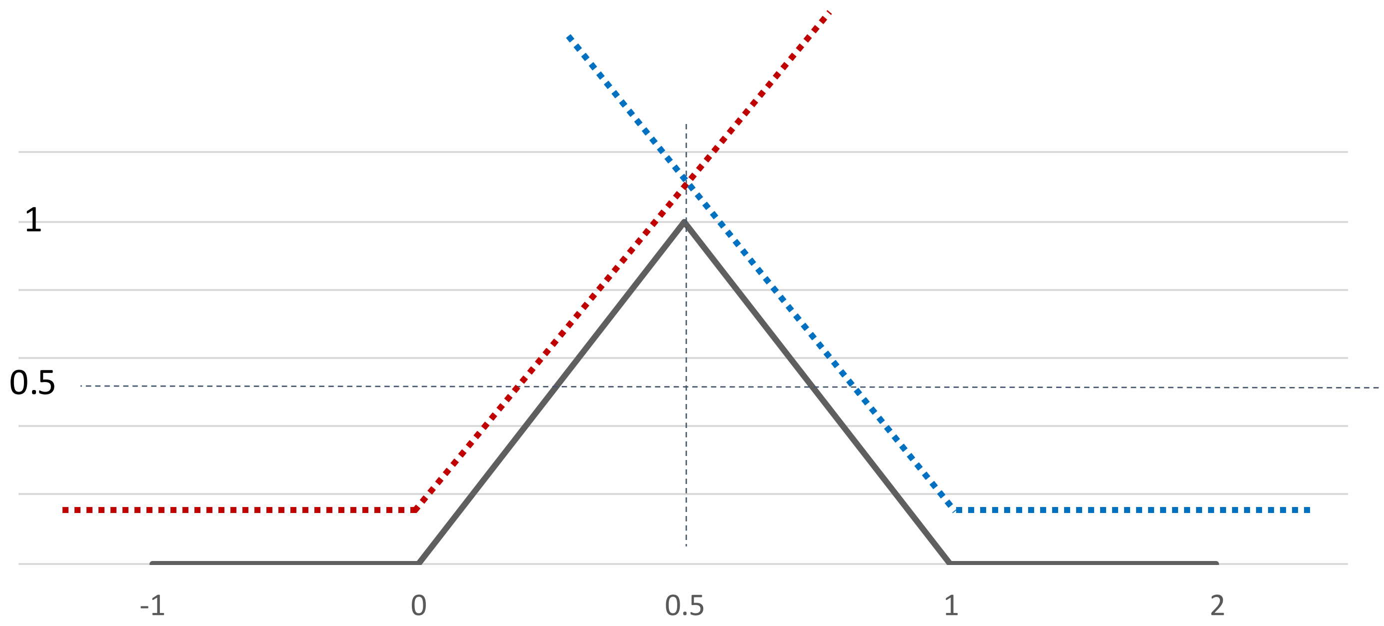
a) Logistic sigmoid and tanh functions.
b) Leaky ReLU and parametric ReLU with parameter 0.25.
c) SoftPlus, Gaussian error linear unit, and sigmoid linear unit.
d) Exponential linear unit with parameters 0.5 and 1.0.
e) Scaled exponential linear unit.
f) Swish with parameters 0.4, 1.0, and 1.4.

Motivation for Deep Learning

Consider a piecewise linear function $m(x) = \begin{cases} 2x, & x \in [0,0.5] \\ 2-2x, & x \in [0.5,1] \\ 0 & otherwise \end{cases}$

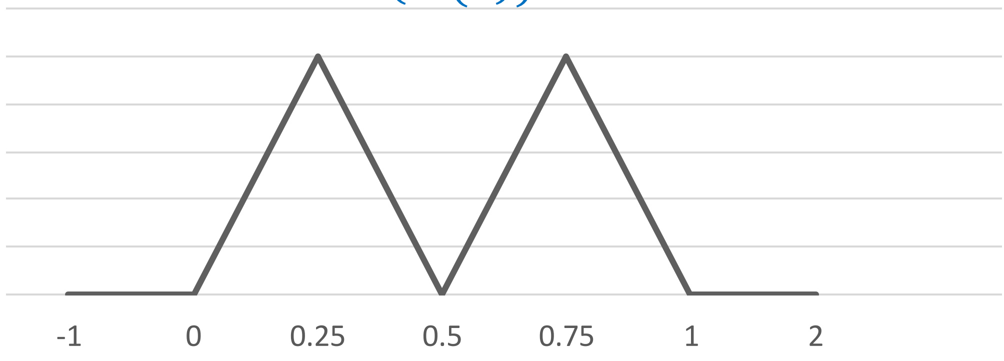
Define $\sigma(x) = \max(0, x)$



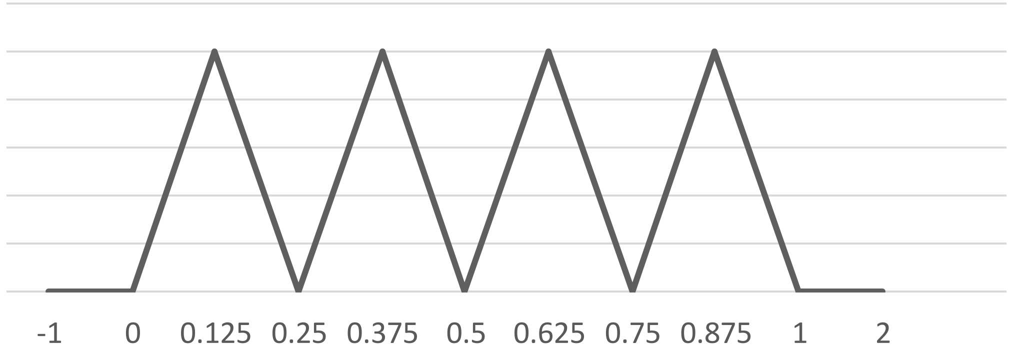


Motivation for Deep Learning

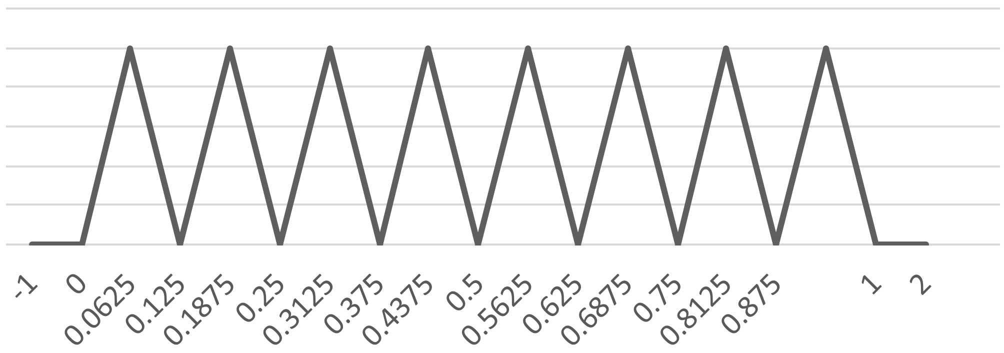
$m(m(x))$



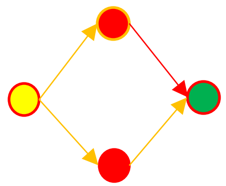
$m(m(m(x)))$



$m(m(m(m(x))))$



Generate a deep network:



Depth
- Multiplicatively

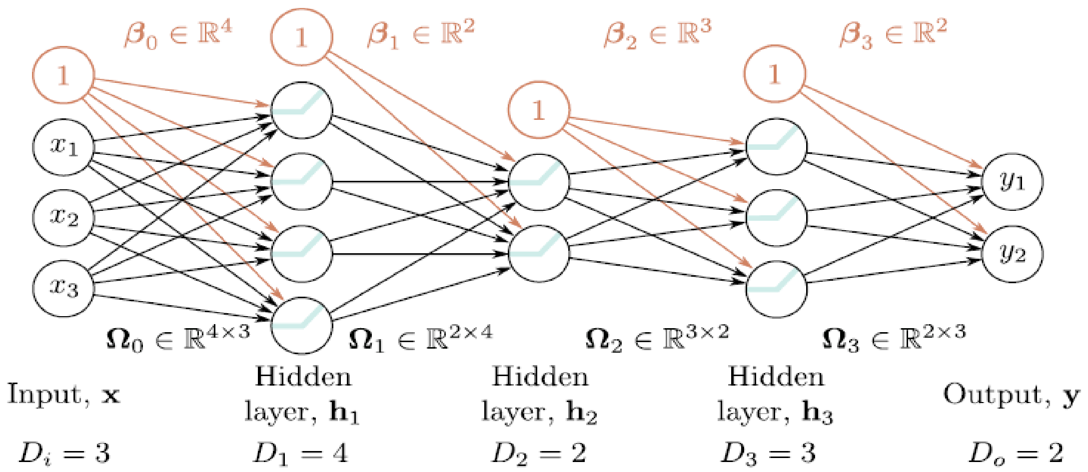
3n+1 nodes to represent the $m^{(n)}(x)$

with width of each layer ≤ 2

Width
- additively

If we generate a shallow one, we need 2^n nodes

Deep Neural Network



Shallow vs deep networks

- ❖ both networks can approximate any function given enough capacity,
- ❖ deep networks produce many more linear regions per parameter,
- ❖ some functions can be approximated much more efficiently by deep networks,
- ❖ in practice, the best results for most tasks are achieved using deep networks with many layers.

$$\mathbf{y} = \boldsymbol{\beta}_K + \boldsymbol{\Omega}_K \mathbf{a}\left[\boldsymbol{\beta}_{K-1} + \boldsymbol{\Omega}_{K-1}\mathbf{a}\left[\ldots \boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2\mathbf{a}\left[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1\mathbf{a}\left[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0\mathbf{x}\right]\right]\ldots\right]\right].$$

- ❖ The number of hidden units in each layer is referred to as the ***width*** of the network, and the number of hidden layers as the ***depth***. The total number of hidden units is a measure of the network's ***capacity***.

$$\begin{aligned}
\mathbf{h}_1 &= \mathbf{a}[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0\mathbf{x}] \\
\mathbf{h}_2 &= \mathbf{a}[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1\mathbf{h}_1] \\
\mathbf{h}_3 &= \mathbf{a}[\boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2\mathbf{h}_2] \\
&\vdots \\
\mathbf{h}_K &= \mathbf{a}[\boldsymbol{\beta}_{K-1} + \boldsymbol{\Omega}_{K-1}\mathbf{h}_{K-1}] \\
\mathbf{y} &= \boldsymbol{\beta}_K + \boldsymbol{\Omega}_K\mathbf{h}_K.
\end{aligned}$$

- ❖ The *depth version* of the universal approximation theorem (Lu et al., 2017): There exists a network with ReLU activation functions and at least D_i+4 hidden units in each layer can approximate any specified D_i-dimensional Lebesgue integrable function to arbitrary accuracy given enough layers.

Fitting DL Models

AS=Applied Statistics

Define a set of functions/models

⬇

Find a criterion/measurement of goodness – loss(+ regularization)

⬇

Get the best model for the problem

AS=Applied Statistics

Design the neural network

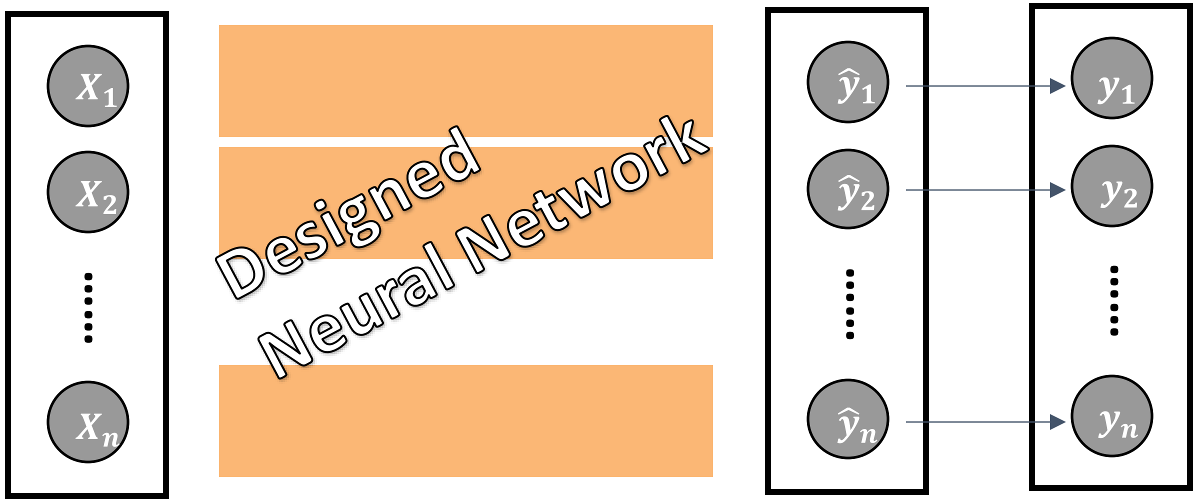
⬇

Find a criterion/measurement of goodness – loss(+ regularization)

⬇

Get the best model for the problem

Fitting DL Models



$$
\begin{aligned}
\mathbf{h}_1 &= \mathbf{a}[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0 \mathbf{x}] \\
\mathbf{h}_2 &= \mathbf{a}[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1 \mathbf{h}_1] \\
\mathbf{h}_3 &= \mathbf{a}[\boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2 \mathbf{h}_2] \\
&\vdots \\
\mathbf{h}_K &= \mathbf{a}[\boldsymbol{\beta}_{K-1} + \boldsymbol{\Omega}_{K-1} \mathbf{h}_{K-1}] \\
\mathbf{y} &= \boldsymbol{\beta}_K + \boldsymbol{\Omega}_K \mathbf{h}_K .
\end{aligned}
$$

Design the neural network

Find a criterion/measurement of goodness

Get the best model for the problem

A **loss function** is needed here,
to measure the difference between the output and truth

Total loss: $$L = \sum \ell(\hat{y}_i, y_i)$$

$$\hat{\mathbf{y}}_i = \boldsymbol{\beta}_K + \boldsymbol{\Omega}_K \mathbf{h}_K(\mathbf{x}_i; [(\boldsymbol{\beta}_0, \boldsymbol{\Omega}_0), \cdots, (\boldsymbol{\beta}_{K-1}, \boldsymbol{\Omega}_{K-1})])$$

Find the network parameters to minimize the loss

Types of Deep Learning

Deep Learning

Unsupervised Learning

The neural network learns to discover the patterns or to cluster the dataset based on unlabeled datasets. There are **no target variables**.

Generative Model

Autoencoders

...

Reinforcement Learning

DDPG

Deep Q Network

...

An agent learns to make decisions in an environment to **maximize a reward signal**. The agent interacts with the environment by taking action and observing the resulting rewards.

Supervised Learning

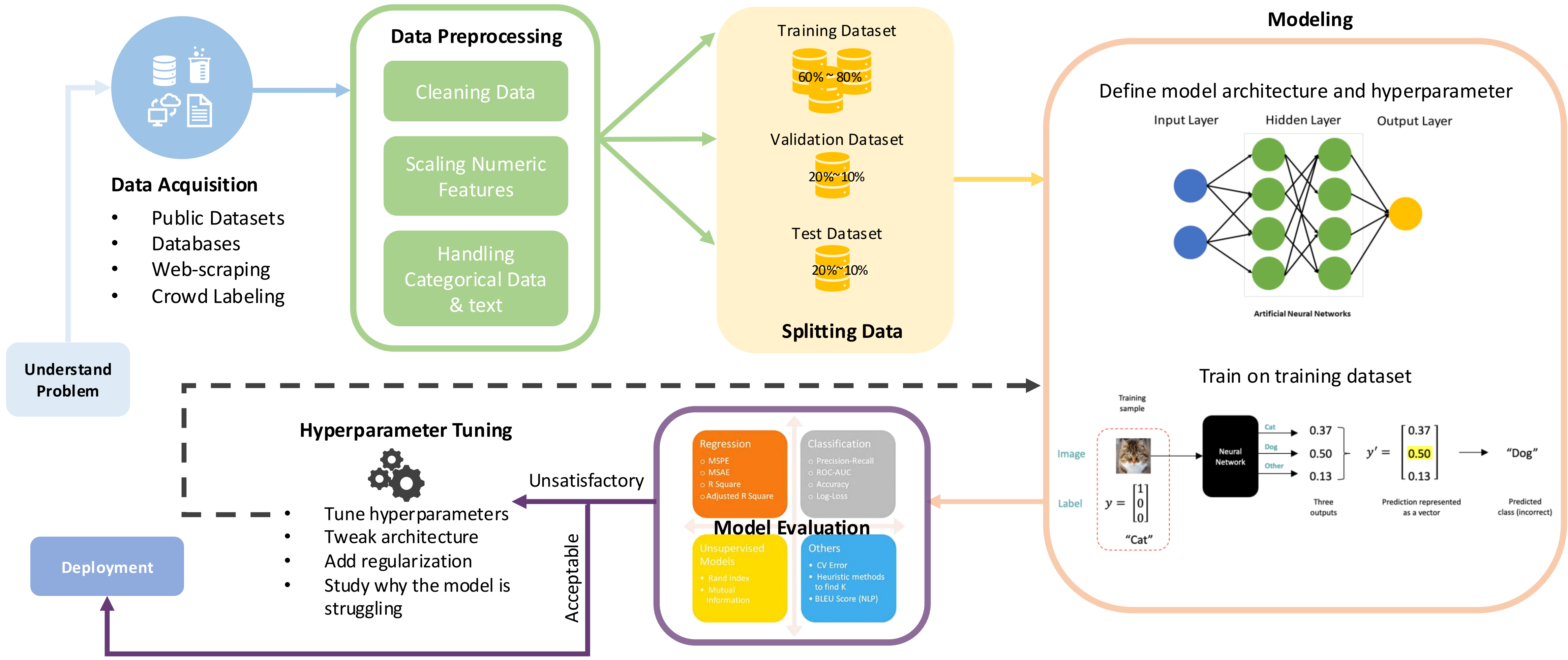
CNN

RNN

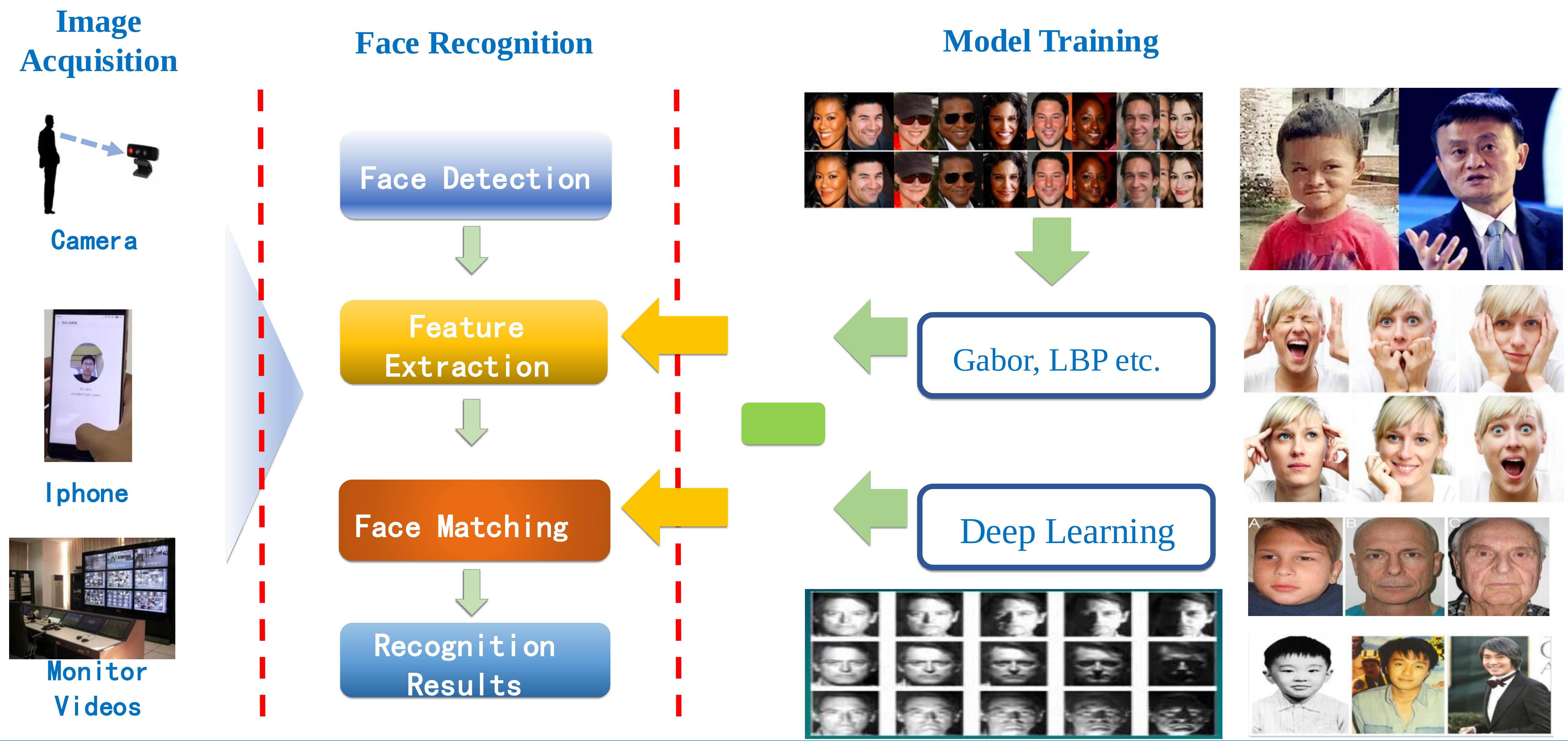
...

Neural network learns to make predictions or classify data **based on the labeled datasets**.

Workflow of a Typical DL Project

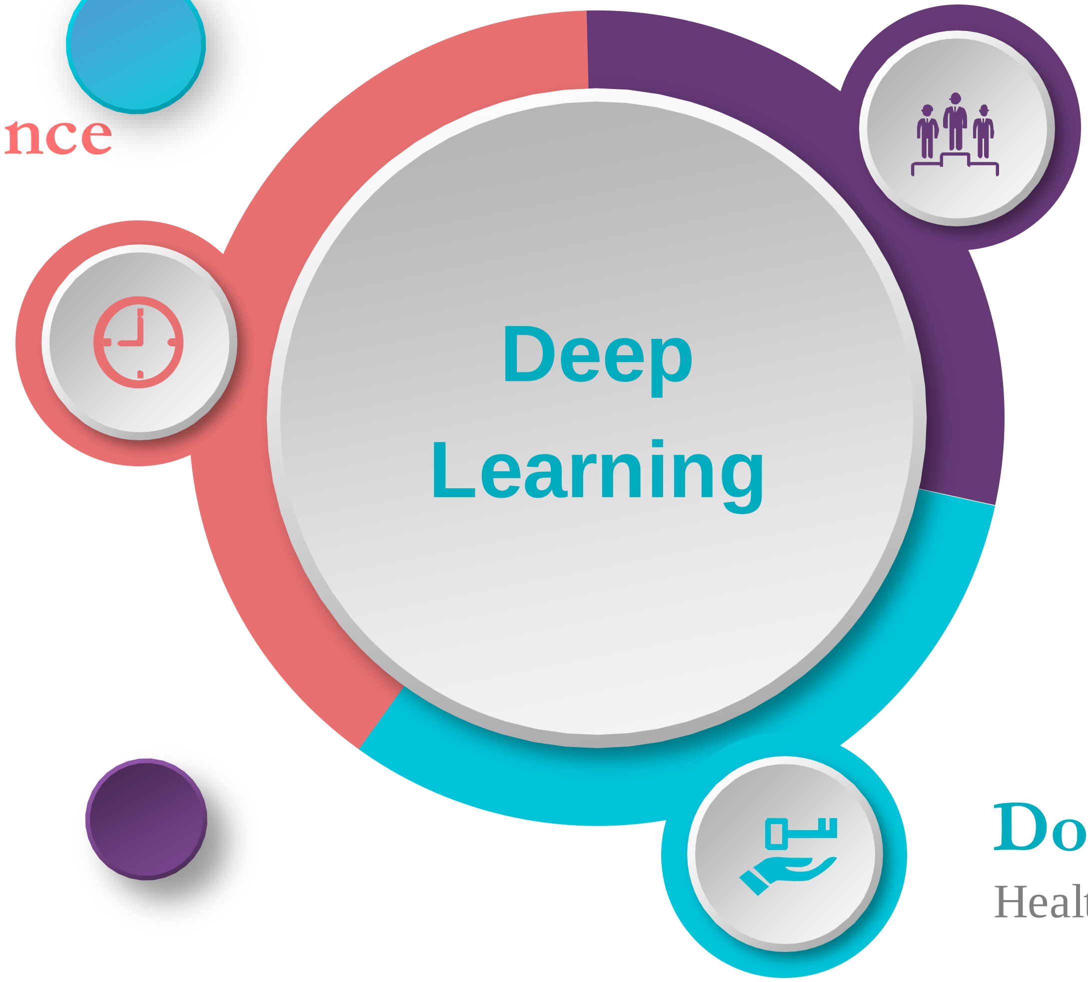


Face Recognition System



Skill Sets

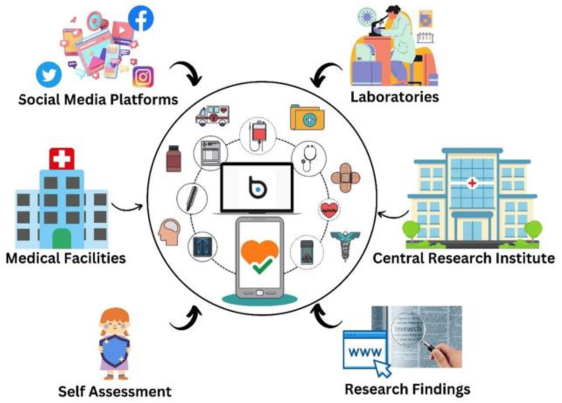
Computer Science
Programming



Math & Statistics
Probability, Linear algebra

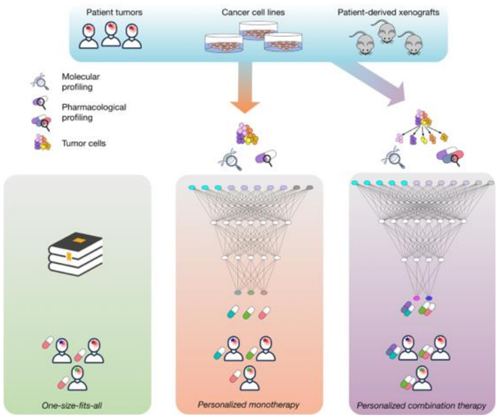
Domain Knowledge
Health science, medical

Some Future Directions in DL for Biostatistics



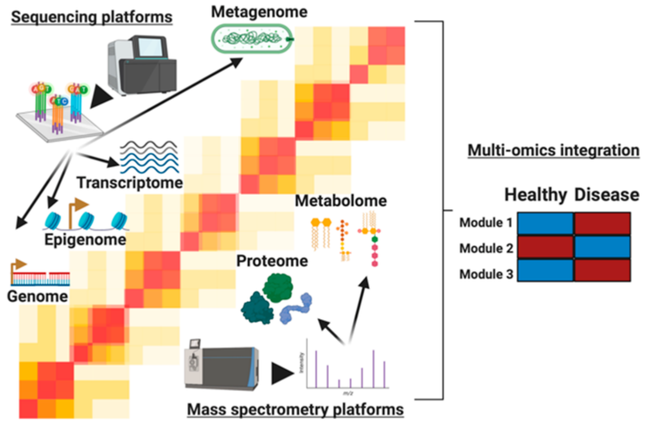
AI-driven Public Health Interventions

Utilizing deep learning models to analyze large-scale public health data for informed decision-making and policy development. Allowing better resource allocation, and more effective epidemic control strategies.



Advanced Drug Response Modeling

Using deep learning to model and predict individual responses to drugs, considering genetic, environmental, and lifestyle factors. Developing more effective personalized treatments.



Integrative Analysis of Multi-omic Data

Leveraging deep learning to integrate and analyze data from genomics, proteomics, metabolomics, and other omics fields for a comprehensive understanding of biological processes and disease mechanisms.

Generalist Medical Artificial Intelligence

- **Foundation Models in Medicine**: These models leverage large-scale datasets and generalizable architectures to address diverse medical tasks, moving beyond task-specific AI systems.
- **Generalist AI**: Unlike traditional models, foundation models aim to function across multiple domains, such as imaging, text, and genomics, enabling integration of multimodal data for holistic medical insights.
- **Challenges**:
  - Data heterogeneity: Medical data comes in varied formats, requiring harmonization.
  - Privacy and ethics: Ensuring secure, unbiased AI while maintaining patient confidentiality.
  - Interpretability: Providing clinicians with actionable insights from AI outputs.
- **Applications**:
  - Diagnostics: Detecting diseases across imaging modalities (e.g., radiology).
  - Prognostics: Predicting patient outcomes using integrated data.
  - Personalized medicine: Tailoring treatments based on multimodal patient profiles.
- **Future Directions**:
  - Collaboration between AI experts and clinicians to co-design models.
  - Development of robust validation frameworks for clinical adoption.
  - Advancing explainability and trust in AI-driven medical decisions.

Moor, M., … ., Rajpurkar, P. (2023) *Nature*.

Perspective

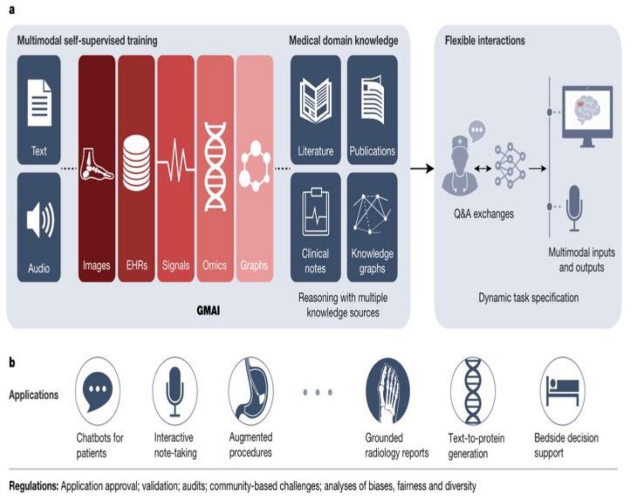


Fig. 1 | Overview of a GMAI model pipeline. a, A GMAI model is trained on multiple medical data modalities, through techniques such as self-supervised learning. To enable flexible interactions, data modalities such as images or data from EHRs can be paired with language, either in the form of text or speech data. Next, the GMAI model needs to access various sources of medical knowledge to carry out medical reasoning tasks, unlocking a wealth of capabilities that can be used in downstream applications. The resulting GMAI model then carries out tasks that the user can specify in real time. For this, the GMAI model can retrieve contextual information from sources such as knowledge graphs or databases, leveraging formal medical knowledge to reason about previously unseen tasks. **b**, The GMAI model builds the foundation for numerous applications across clinical disciplines, each requiring careful validation and regulatory assessment.

Content

1 Introduction to Deep Learning

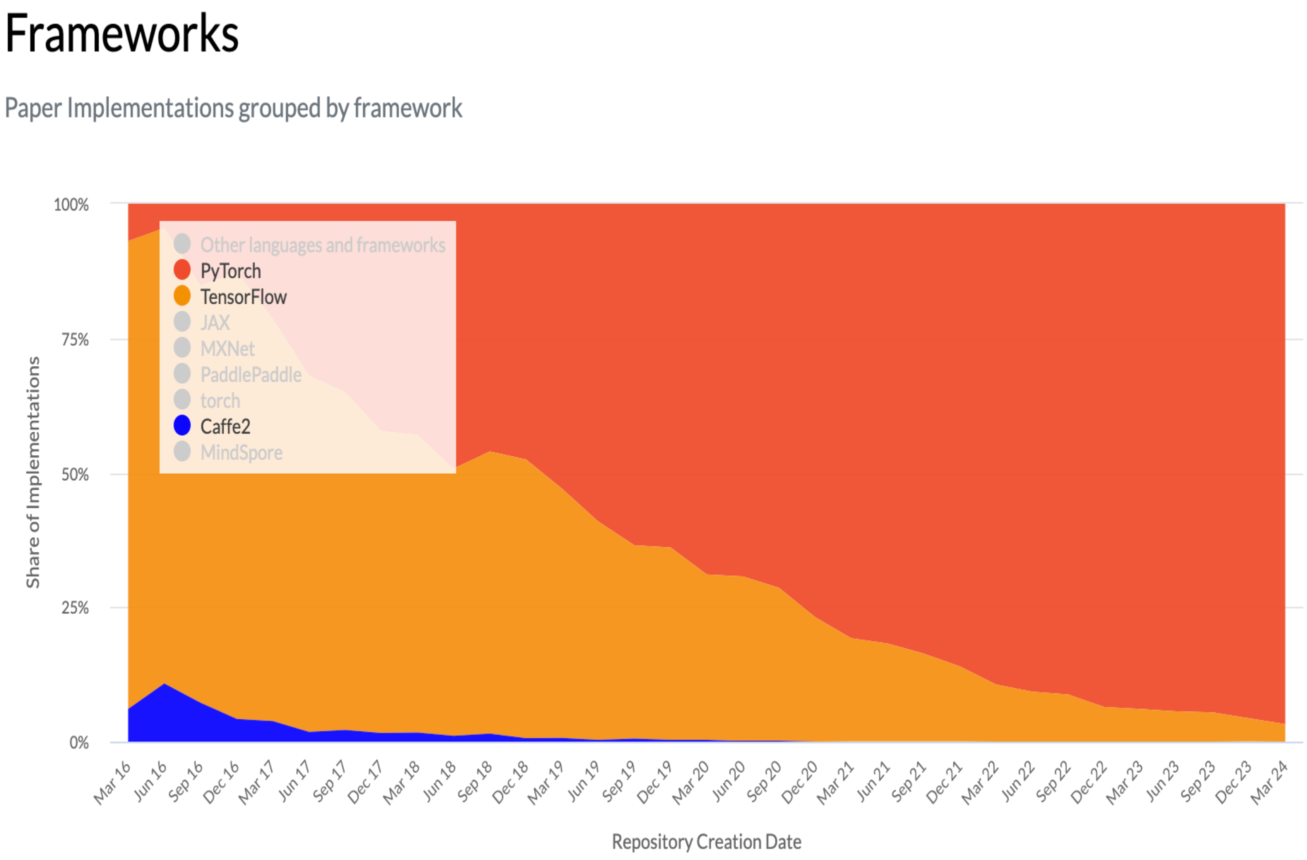
2 Introduction to PyTorch

3 Introduction to UNC Research Computing Resources

4 Introduction to Basic Algorithm

Why Torch?


NLTK
Spark
theano
Numpy
Keras
Pytorch
TensorFlow
mxnet
Scikit-learn
Pandas


Frameworks
Paper Implementations grouped by framework
Other languages and frameworks
PyTorch
TensorFlow
JAX
MXNet
PaddlePaddle
torch
Caffe2
MindSpore
100%
75%
50%
25%
0%
Share of Implementations
Mar 16
Jun 16
Sep 16
Dec 16
Mar 17
Jun 17
Sep 17
Dec 17
Mar 18
Jun 18
Sep 18
Dec 18
Mar 19
Jun 19
Sep 19
Dec 19
Mar 20
Jun 20
Sep 20
Dec 20
Mar 21
Jun 21
Sep 21
Dec 21
Mar 22
Jun 22
Sep 22
Dec 22
Mar 23
Jun 23
Sep 23
Dec 23
Mar 24
Repository Creation Date

What is PyTorch?

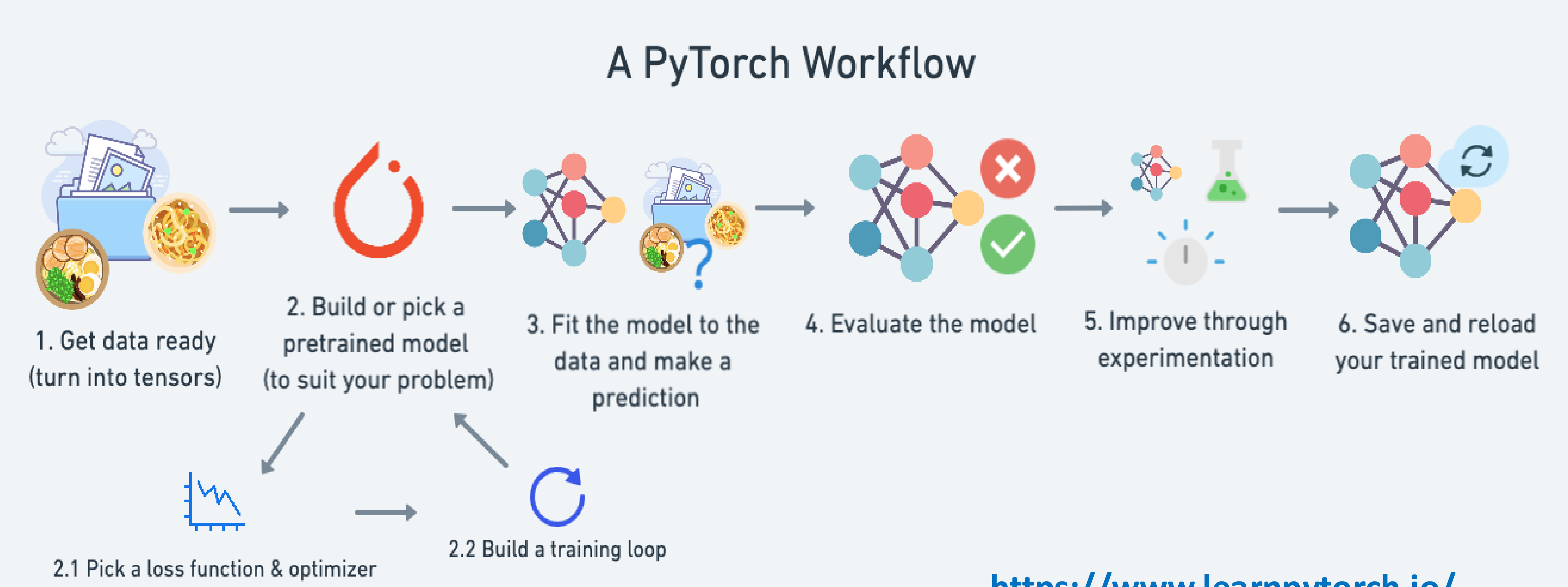
- **PyTorch** is a Python-based machine learning library designed to provide flexibility and efficiency for developing deep learning models. It is widely used in academia and industry due to its intuitive and dynamic design.

  https://pytorch.org/tutorials/beginner/basics/intro.html

- **Key Features**:
  - **Dynamic Computation Graphs**: Modify the model's architecture during runtime, making it easier to debug and experiment.
  - **GPU Acceleration**: Seamlessly integrate GPU computations for speedup.
  - **Extensive Ecosystem**: Includes libraries such as **torchvision** (computer vision), **torchtext** (NLP), and **torchaudio** (audio processing).
  - **Integration with Python**: PyTorch operates natively in Python, allowing access to Python libraries and tools.

- **Common Use Cases**:
  - Building neural networks for image recognition, natural language processing, and generative models.
  - Research and experimentation due to flexibility in designing and debugging models.
  - Production-level deployment using tools like TorchScript and TorchServe.

A PyTorch Workflow



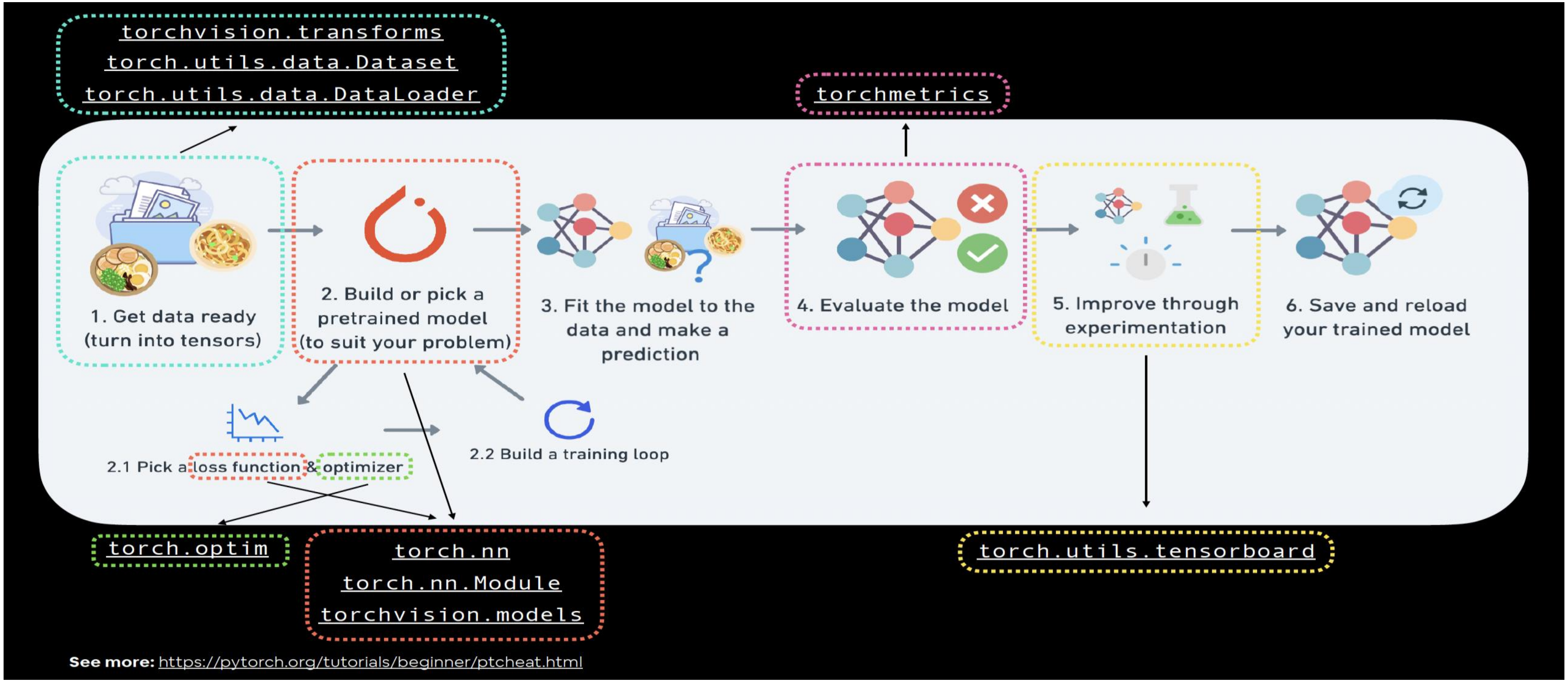
https://www.learnpytorch.io/

PyTorch essential building modules

PyTorch module	What does it do?
`torch.nn`	Contains all of the building blocks for computational graphs (essentially a series of computations executed in a particular way).
`torch.nn.Module`	The base class for all neural network modules, all the building blocks for neural networks are subclasses. If you're building a neural network in PyTorch, your models should subclass `nn.Module`. Requires a `forward()` method be implemented.
`torch.optim`	Contains various optimization algorithms (these tell the model parameters stored in `nn.Parameter` how to best change to improve gradient descent and in turn reduce the loss).
`torch.utils.data.Dataset`	Represents a map between key (label) and sample (features) pairs of your data. Such as images and their associated labels.
`torch.utils.data.DataLoader`	Creates a Python iterable over a torch Dataset (allows you to iterate over your data).

See more: https://pytorch.org/tutorials/beginner/ptcheat.html

A PyTorch Workflow with Modules



https://pytorch.org/tutorials/

A Sample Torch Code

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# Define a simple neural network
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.dropout(x)
        x = self.fc2(x)
        return F.log_softmax(x)

# Create an instance of the network
net = Net()
```

```
# Data loading
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32,
shuffle=True)

# Define a loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001)

# Training loop
for epoch in range(10):
    for inputs, targets in train_loader:
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

Working with Tensors

- Tensors are the foundation of PyTorch, enabling efficient numerical computations. They are similar to NumPy arrays but optimized for GPUs. Tensors are used to encode inputs and model weights.
- **Creating Tensors:**
  - Scalars, vectors, matrices, and higher-dimensional tensors:

Numpy style operations

```
tensor = torch.ones(4, 4)
print(f"First row: {tensor[0]}")
print(f"First column: {tensor[:, 0]}")
print(f"Last column: {tensor[..., -1]}")
tensor[:,1] = 0
print(tensor)
```

Out:

```
First row: tensor([1., 1., 1., 1.])
First column: tensor([1., 1., 1., 1.])
Last column: tensor([1., 1., 1., 1.])
tensor([[1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.]])
```

Capability of CPU/GPU computing

```
# We move our tensor to the GPU if available
if torch.cuda.is_available():
    tensor = tensor.to("cuda")
x = torch.tensor([1.0]).to('cuda')  # Move to GPU
y = x.to('cpu')  # Move back to CPU
```

Common Operations:

- Element-wise operations: +, -, *, /.
- Matrix operations: torch.matmul, torch.mm.
- Reshaping: .reshape(), .squeeze(), .unsqueeze().

Datasets and DataLoader

PyTorch provides a structured approach to handling datasets through Dataset and DataLoader.

Dataset:

The Dataset class allows you to define how data samples are accessed and prepared.

Custom datasets can be implemented by subclassing torch.utils.data.Dataset and defining __getitem__and __len__ methods.

Built-in Datasets: PyTorch provides ready-to-use datasets such as MNIST and CIFAR-10 through torchvision.datasets.

DataLoader:

- Combines datasets into batches, shuffles data, and handles multiprocessing for loading data efficiently.

Loading FashionMNIST

```
from torchvision import datasets
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor())
```

Building a Dataloader

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(training_data, batch_size=64,
shuffle=True)
train_features, train_labels = next(iter(train_dataloader))
```

In practice, many researches opt to write their own data processing and sampling code to gain greater flexibility.

Network Structure

- The neural network is defined by subclassing nn.Module, the layers are initialized in __init__ and the operations on input data is in forward method.
- After instantiate the network, the network do forward by passing the input data.

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

Key Components:

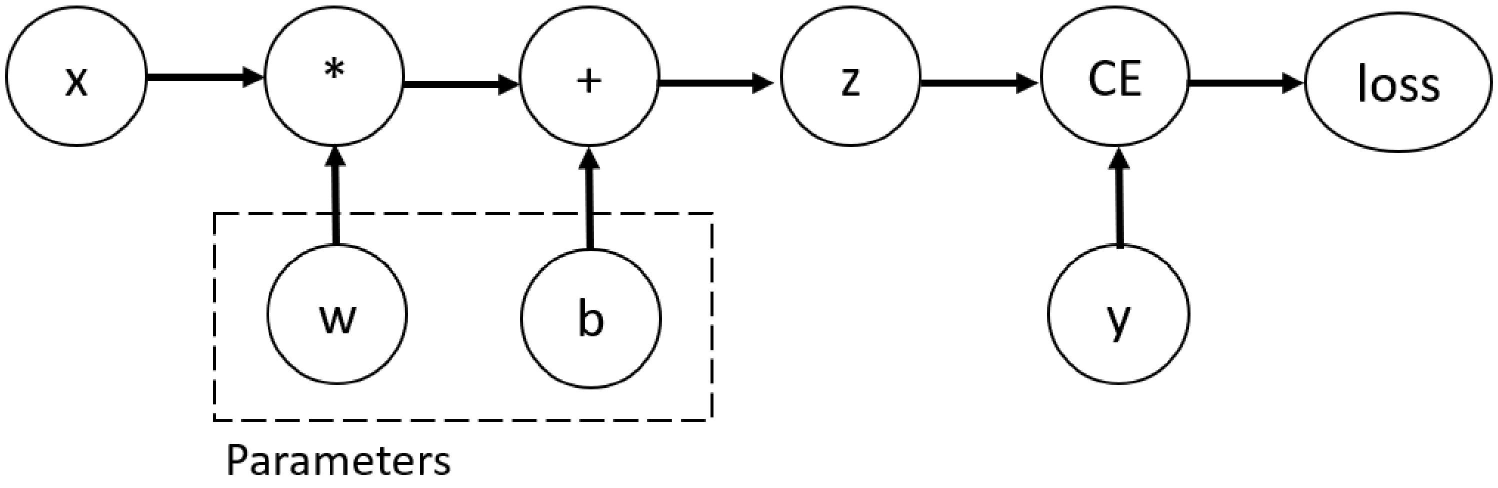
- **Modules:** Predefined layers like
- nn.Linear, nn.Conv2d, nn.LSTM.
- **Sequential Models:** Simplify model definition:
- **Custom Models:** Subclass nn.Module to define custom architectures:
- **Activation Functions:** PyTorch provides various activation functions such as ReLU, Sigmoid, and Tanh.

Instantize and send weights to GPU

```
model = NeuralNetwork().to(device)
X = torch.rand(1, 28, 28, device=device)
logits = model(X)
```

Auto-differentiation

- When training neural networks, the most frequently used algorithm is gradient descent (and its variants). Pytorch has a built-in engine called autograd for calculating gradients.




```
x = torch.ones(5)  # input tensor
y = torch.zeros(3)  # expected output
w = torch.randn(5, 3, requires_grad=True)
b = torch.randn(3, requires_grad=True)
z = torch.matmul(x, w)+b
loss = torch.nn.functional.binary_cross_entropy_with_logits(z, y)
```

- Pytorch have the notion of computation graph.

- Each time do calculations, pytorch store the graph structure (how the final output is related to the tensors that require grad).

- When you use backward to calculate the grad, pytorch calculates the gradient of each component that requires_grad and store it.

Optimization

- Optimization algorithms define how model parameters adjust to reduce model error in each training step.
- Many optimization algorithms (e.g. Adam) are available in torch.optim.

```
# Define the optimizer
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

# Within each training step, first reset the gradient of parameters
# Then calculate the gradients of the loss w.r.t. each parameter.
optimizer.zero_grad()

# Adjust the parameters using gradients
optimizer.step()
```

Training and Testing Loops

Training a neural network involves iterative optimization to minimize a loss function.

- **Steps in Training**:
  - **Initialize the Model**: Define the network and its parameters.
  - **Define the Loss Function**: Choose a loss function (e.g., nn.MSELoss, nn.CrossEntropyLoss).
  - **Select an Optimizer**: Use optimizers like torch.optim.SGD or torch.optim.Adam.
- **Training Loop**:

```
for epoch in range(n_epochs):
    model.train()  # Set model to training mode
    for x, y in dataloader:
        optimizer.zero_grad()  # Reset gradients
        output = model(x)  # Forward pass
        loss = criterion(output, y)  # Compute loss
        loss.backward()  # Backward pass
        optimizer.step()  # Update weights
```

Testing Loop: Use .eval() mode and disable gradient calculations:

```
model.eval()
with torch.no_grad():
    for x, y in test_loader:
        output = model(x)
```

Content

1 Introduction to Deep Learning

2 Introduction to PyTorch

3 Introduction to UNC Research Computing Resources

4 Introduction to Basic Algorithm

UNC Longleaf

Before begin:

- Establish a VPN connection with UNC.
  - [For VPN instructions, search https://help.unc.edu for VPN.]
- If you do not have longleaf account yet, request it now:
  - https://help.rc.unc.edu/request-a-cluster-account/
- *If there's time, download…*
  - *and install the ssh app longleaf prefers:*
    - *https://help.rc.unc.edu/getting-logged-on/*
  - *These Using Longleaf course slides:*
    - *https://its.unc.edu/research-computing/research-computing-presentations/*

UNC ITS Computing Resources

Provides "computing infrastructure as well as other technology tools and capabilities to support the research needs of University faculty and staff"

Resource examples:

- VCL: Virtual Computing Lab
- **Longleaf Cluster (today)**
- Dogwood Cluster
- DGX Cluster
- Secure Research Workstation
- Access to Xsede Campus Champions program

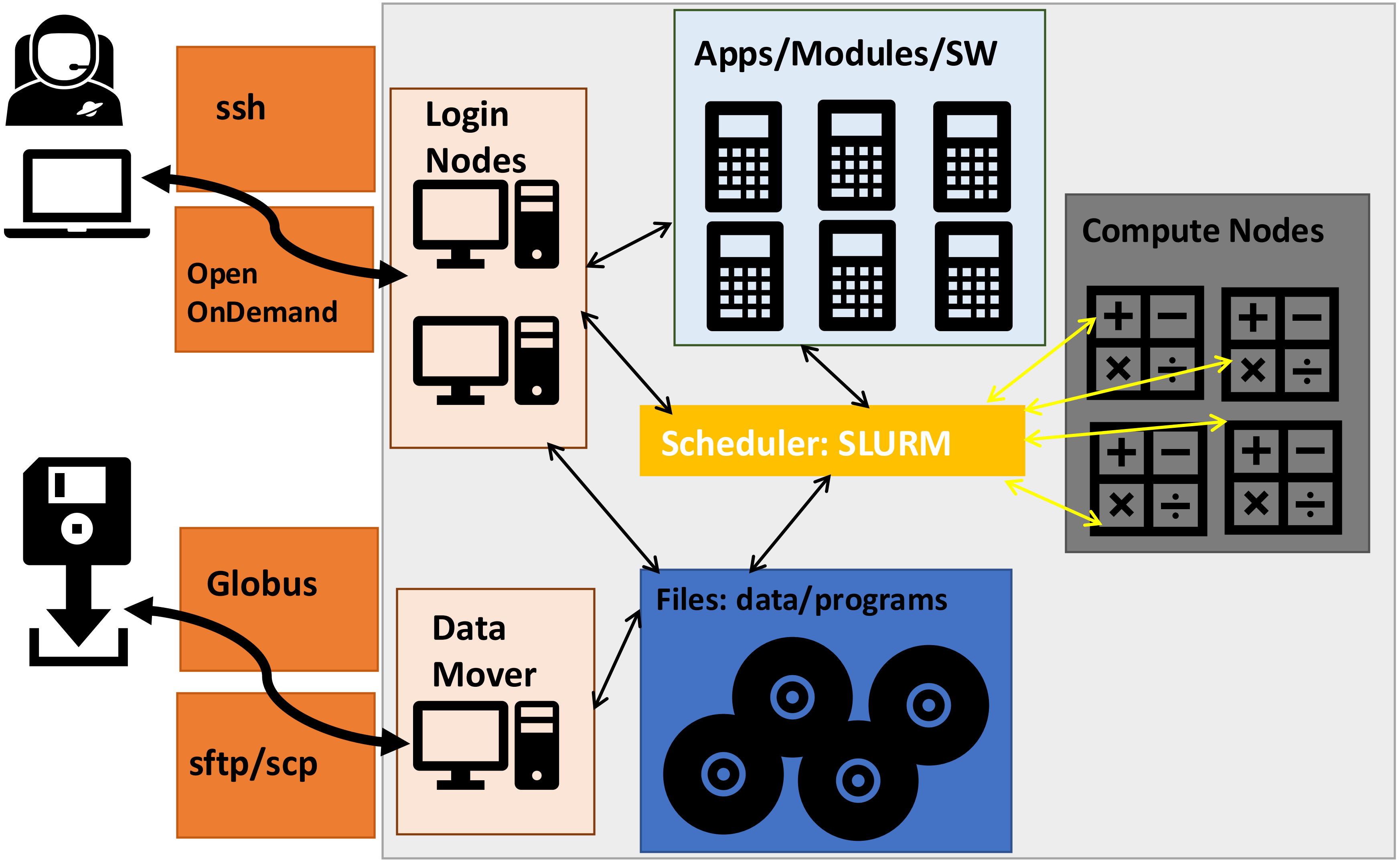
Need help?

https://help.rc.unc.edu

research@unc.edu answered during business hours, M-F

List grows all the time...including some cloud computing.

Longleaf (LL) Components



Before Computing on Longleaf

https://island-climb-ea6.notion.site/Longleaf-GPU-guidance-1723d297a55f803daccff5e46f77659e

Properly

- **Connect** to Longleaf (log on)
- **Move** your project's data & programs to/from storage areas in longleaf
- Pick and **manage** your software "modules"
- **Schedule** jobs to run on longleaf's compute nodes
- **Monitor** the status of jobs, their output and results
- Navigate a Linux operating system *(not in this course)*

THEN you can have fun doing your project on longleaf.

Access Longleaf

Two ways to connect:

1. Via an **ssh connection**: ssh –X <onyen>@longleaf.unc.edu

   *Requires an ssh program on your computer, and **not** the one your computer came with.*

   If connecting from off-campus, your computer is required to establish a VPN connection to UNC before connecting to longleaf. Search help.unc.edu for VPN.

2. Via **Open OnDemand (OOD)**: https://ondemand.rc.unc.edu

   ****New in 2020****

   Yes any browser will do! (But not your cell phone)

   Limited access to longleaf, but enough for many users.

   DUO authentication required.

Read More: https://help.rc.unc.edu/getting-logged-on/

Content

1 Introduction to Deep Learning

2 Introduction to PyTorch

3 Introduction to UNC Research Computing Resources

4 Introduction to Basic Algorithm

Case Study: Logistic Regression in PyTorch

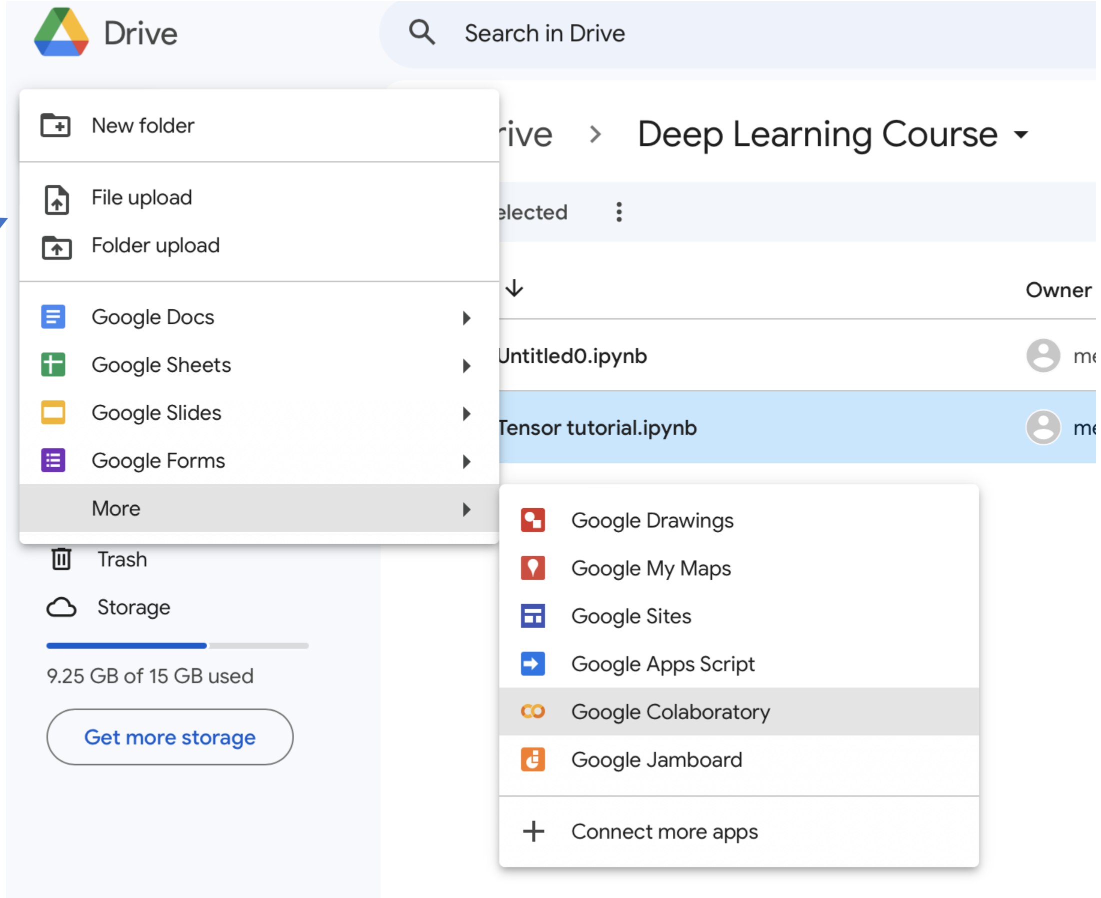
Identifying handwritten digits using Logistic Regression in PyTorch.ipynb

Case Study: Logistic Regression in PyTorch

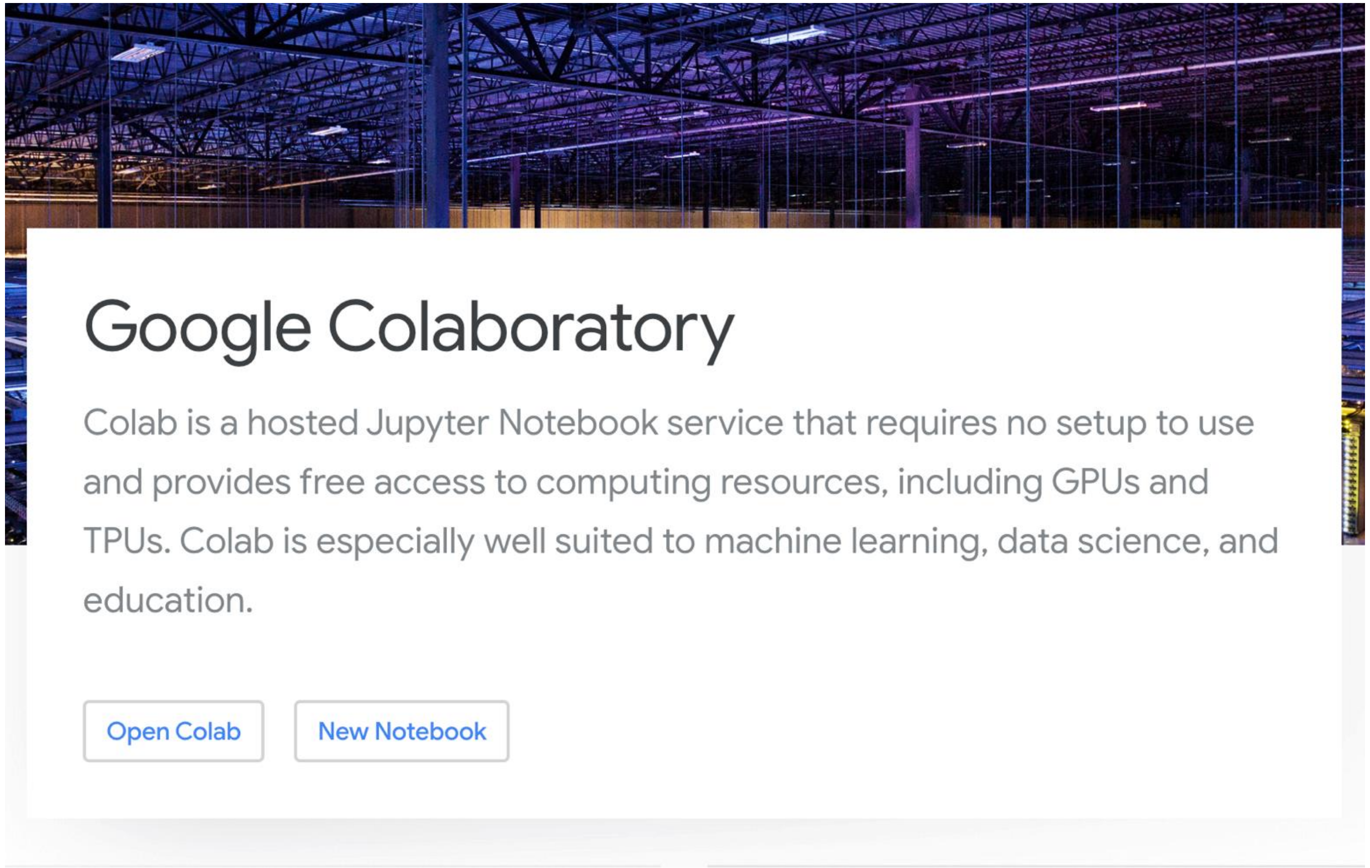
Running example code in Google Colab – a good start for beginners

First step: How to access to Google Colab?

1. Log into Google Drive.
2. In Google Drive, create a new Google Colab notebook.
3. Start coding then.

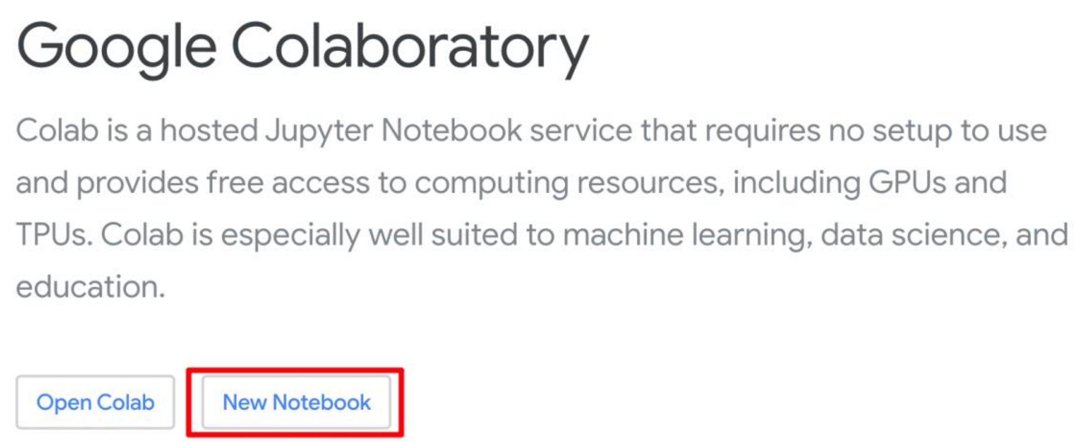


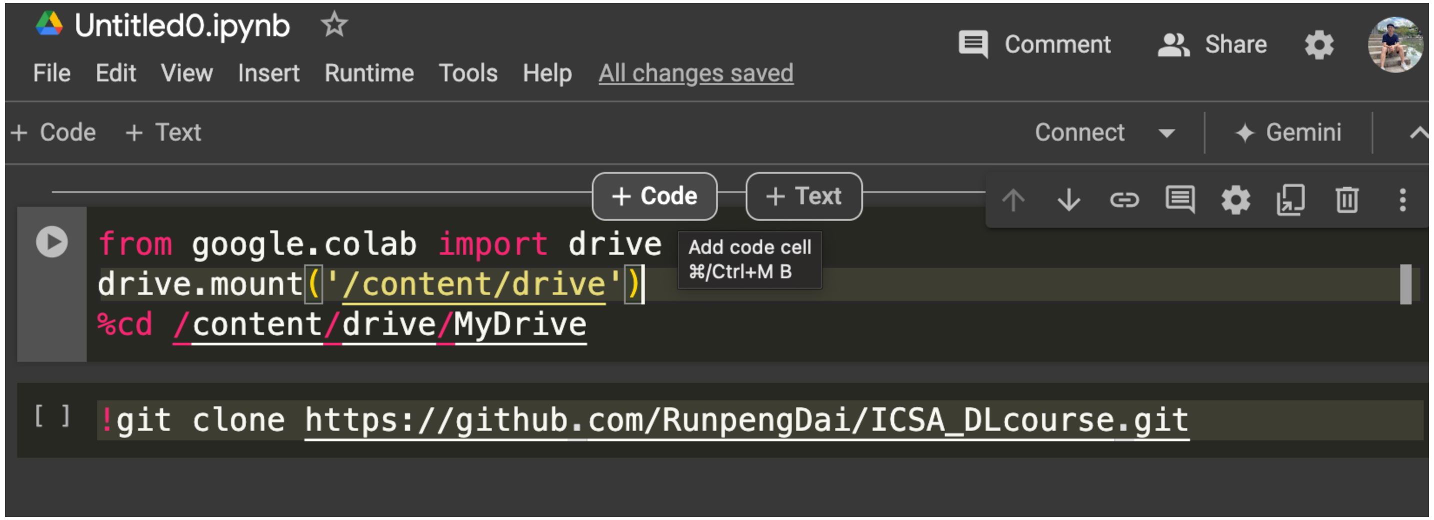
Google Colab



- Google Colab, short for Colab, is a free, cloud-based platform provided by Google Research. It allows users to write and execute Python code through a web browser.
- One of Colab's key features is the use of Jupyter Notebooks
- Colab provides free access to powerful hardware accelerators, including GPUs and TPUs
- Colab can mount google drive.

Setup



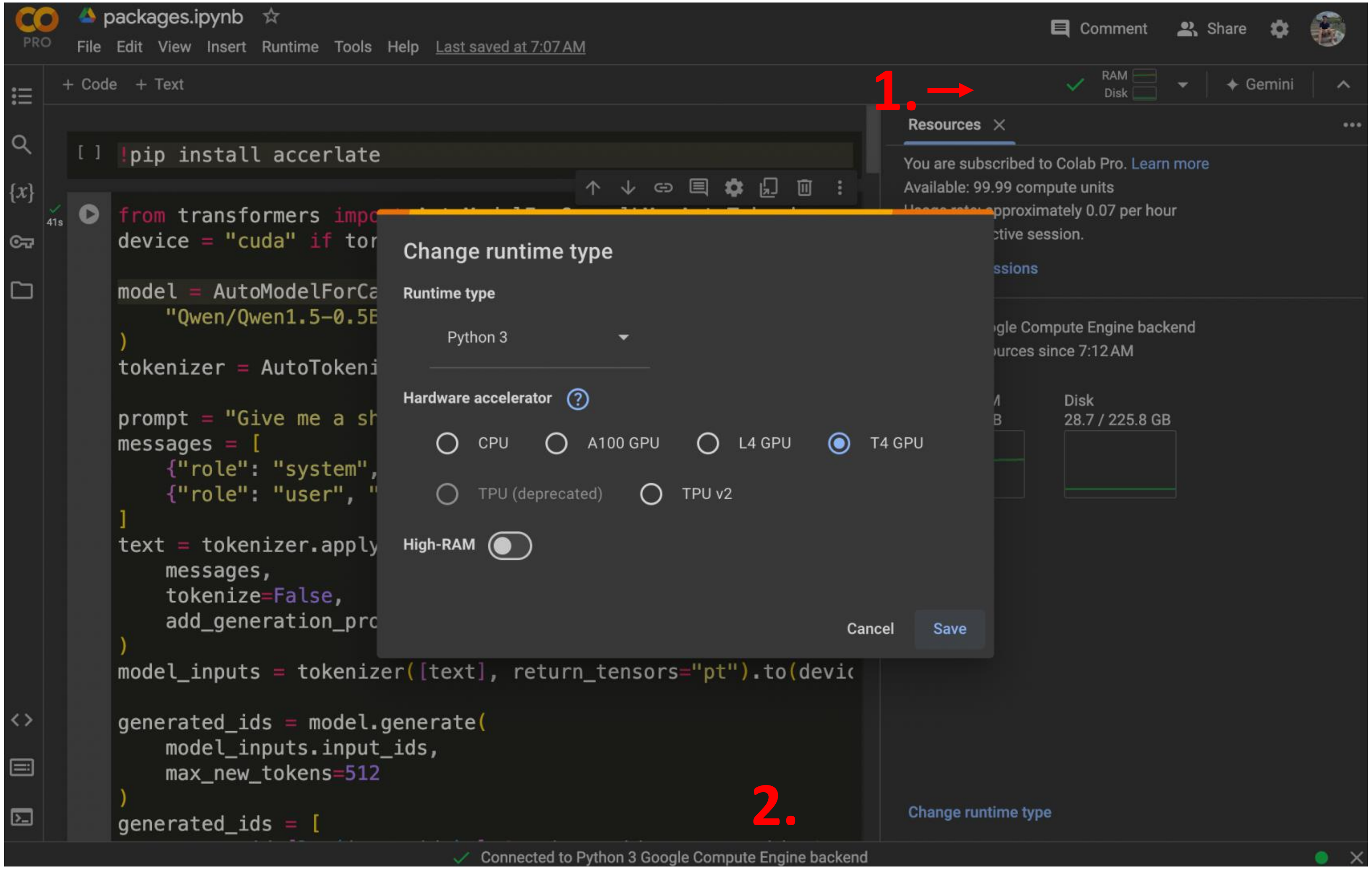


- We offer multiple coding sessions in the short course.
- We are going to clone the git repo into Goole drive using commands in colab.
- First, create a empty Notebook through https://colab.google/.
- Then execute the following code, to mount Google drive to colab and clone our repo to Google drive.

```
from google.colab import drive
drive.mount('/content/drive')
%cd /content/drive/MyDrive
```

```
!git clone
https://github.com/RunpengDai/ICSA_DLcourse.git
```

Code session – Pytorch Basics



- First open packages.ipynb from the intro folder within Google drive.
- Change the runtime type to T4 GPU to have access to GPU computing resource.

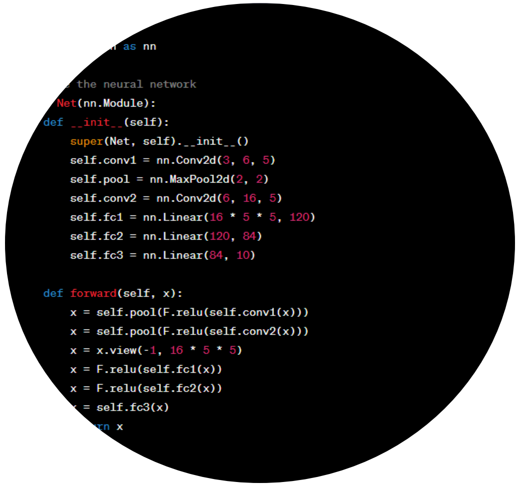
References

Prince, S. J. D. (2023). Understanding Deep Learning.

Shen, G. (2024). Exploring the Complexity of Deep Neural Networks through Functional Equivalence. International Conference on Machine Learning 2024

Suh, N. and Cheng, G. (2024). A Survey on Statistical Theory of Deep Learning: Approximation, Training Dynamics, and Generative Models

How to succeed in this course?



Practice

Explore

Visualize

Discuss

Ask