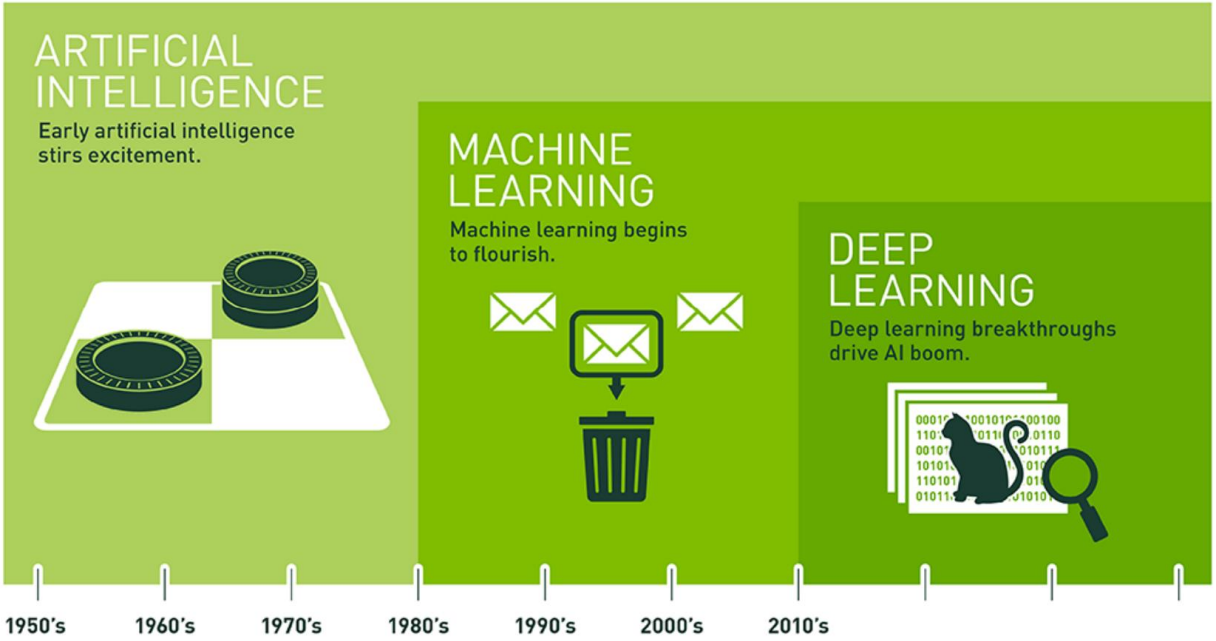
Advanced Deep Learning

Lec 1: Introduction and Advances in DL

STAT4744/5744

Xin (Shayne) Xing • Virginia Tech

AI, Machine Learning, and Deep Learning



Artificial Intelligence
Artificial Intelligence is human intelligence exhibited by machines

Machine Learning
the practice of using algorithms to learn from data, and then make a determination or prediction about something in the world.

Deep Learning
Using deep neural networks to implement machine learning

Historical Context

1950s-1960s: Early AI excitement

1990s-2000s: Machine learning flourishes

2010s-present: Deep learning breakthroughs

Source: adapted from nvidia.com

Supervised Learning

Concept

Have both covariates **X** and response (label) **Y**

Goal: Learn mapping from X to Y using training data, then predict Y for new X

Two Types

Regression (Y continuous)

Linear Regression, Neural Networks

Classification (Y discrete)

Logistic Regression, Random Forest, CNNs

Application Example

Protein Function Classification

- Input: DNA sequences (A, T, C, G)
- Labels: Protein functions (enzyme, structural, etc.)
- Model predicts functions of new proteins
- Accelerates scientific discovery

Example: AlphaFold for protein structure prediction

Training data: X (features) → Y (labels) | Test data: X_new → Y_new = ?

Unsupervised Learning

Concept

Only have covariates **X** (no labels)

Goal: Discover hidden patterns and structure in data

Two Main Tasks

Clustering

Group similar data points together

Methods: K-means, Hierarchical Clustering

Dimension Reduction

Reduce feature space while preserving information

Methods: PCA, t-SNE, UMAP

Deep learning approaches: Autoencoders, VAEs, Self-organizing Maps

Application Example

Single-Cell Clustering

- Measure gene expression in individual cells
- Data: thousands of genes × thousands of cells
- Cluster cells to identify cell types
- Reveals cellular heterogeneity in tissues

Example: Identifying T-cells, B-cells, NK-cells in immune system analysis

Reinforcement Learning

Concept

Agent learns through interaction with environment

State: Current situation

Action: What agent can do

Reward: Feedback from environment

Goal: Maximize cumulative reward

Framework

Markov Decision Process (MDP) for discrete state space

Example: Cart-Pole Balancing

State:

Cart position, velocity, pole angle, angular velocity

Actions:

Push left or push right

Reward:

+1 for each step pole stays balanced

Done:

When pole deviates more than 15° from vertical

Real-World Applications

Robotics, game playing (AlphaGo, OpenAI Five), autonomous vehicles, resource management

Deep RL combines neural networks with RL algorithms (DQN, PPO, A3C)

Deep Learning: Overview

What is Deep Learning?

Uses deep neural networks that mimic the human brain to process data for object detection, speech recognition, language translation, and decision making

Deep Learning Approaches

Supervised: CNNs, RNNs, Transformers

Unsupervised: Autoencoders, VAEs, GANs

Reinforcement: Deep Q-Networks, Policy Gradients

Why Deep Learning Works

- Automatic feature learning
- Scales with data and compute
- Hierarchical representations
- End-to-end optimization

Computer Vision | **NLP** | **Speech Recognition** | **Game Playing**

Deep learning excels with large datasets and sufficient computational resources

Deep Learning vs Traditional ML

Traditional ML

- Requires manual feature engineering
- Different model for each task
- Works well with small-medium datasets
- More interpretable
- Performance plateaus with more data

Examples

Linear Regression, SVM, Random Forest, k-NN

Deep Learning

- Automatic feature learning
- One architecture for multiple tasks
- Requires large datasets
- More complex (black box)
- Performance improves with more data

Examples

CNNs, RNNs, Transformers, GANs, VAEs

Deep learning dominates when: (1) Large datasets available, (2) Complex patterns, (3) High-dimensional data

Model Flexibility: Traditional ML vs Deep Learning

Traditional Machine Learning

Different Models for Different Tasks

Image Classification
→ SVM with HOG features

Text Analysis
→ Naive Bayes with TF-IDF

Time Series
→ ARIMA models

Tabular Data
→ XGBoost, Random Forest

Deep Learning

One Architecture, Multiple Tasks

Image Tasks
→ CNN for all vision tasks

Text Tasks
→ Transformer for all NLP

Sequential Data
→ RNN/LSTM for sequences

Multimodal
→ Handles images + text

Limitations

- Each task needs specialized model
- Manual feature engineering
- Limited transfer learning

Advantages

- Universal architecture
- Automatic feature learning
- Easy transfer learning

Key Insight: Deep learning provides unified frameworks that adapt to different data types and tasks

Performance Scaling

Why DL Scales Better

- Learns complex patterns
- High parameter capacity
- End-to-end optimization

Traditional ML Limit

- Fixed features
- Lower capacity
- Plateaus quickly

DL Advantage

- Better with more data
- Scalable capacity
- Complex patterns

Critical Insight: With small data, traditional ML can compete. With big data, deep learning dominates.

One Neural Network Architecture, Multiple Tasks

CNN (Convolutional Neural Network)

Same architecture, different tasks

Image Classification
Cat vs Dog → 2 output neurons

Object Detection
Same CNN + bounding box head

Semantic Segmentation
Same CNN + pixel-wise output

Medical Diagnosis
Same CNN on X-rays/MRI scans

Transformer

Universal sequence processor

Machine Translation
English → French, Chinese → English

Text Generation
GPT: stories, code, essays

Question Answering
BERT: reading comprehension

Sentiment Analysis
Same transformer, different fine-tuning

RNN/LSTM

Handles sequential data

Time Series Forecasting
Stock prices, weather prediction

Speech Recognition
Audio waveform → text

Music Generation
Compose melodies note by note

Video Analysis
Process frame sequences

The Power of Transfer Learning

Pre-train Once

Train on large dataset (ImageNet, Wikipedia) to learn general features

Fine-tune Many Times

Adapt same model to medical images, satellite imagery, art classification, etc.

Shared Knowledge

Low-level features (edges, textures) transfer across all vision tasks

Key Advantage: Learn the architecture once, apply everywhere with minimal modification

Big Data & Programming Tools

The Data Explosion

"We are drowning in information but starved for knowledge." - John Naisbitt

We need computing tools to make sense of massive data

Data Sources

Social media, IoT sensors, scientific instruments

Medical imaging, genomics, satellite imagery

Video, audio, text from billions of users

Programming Languages

Python (76.3%)

Most popular for ML/DL

R (59.9%)

Strong for statistics

Deep Learning Frameworks

PyTorch, TensorFlow, Keras, JAX

High-level frameworks abstract complexity, enabling rapid experimentation

The Transformer Revolution (2017)

Key Innovation

Self-Attention Mechanism: Models can focus on relevant parts of input simultaneously

Parallel Processing: Unlike RNNs, transformers process all tokens at once

Scalability: Architecture scales efficiently with more data and parameters

Why It Matters

- Replaced recurrent architectures in NLP
- Enabled training on massive datasets
- Foundation for all modern LLMs
- Extended to computer vision (ViT)

"Attention Is All You Need" (Vaswani et al., 2017)

Large Language Models (LLMs)

GPT-3 (2020)

175B parameters

Few-shot learning

ChatGPT (2022)

RLHF training

Conversational AI

GPT-4 (2023)

Multimodal

Advanced reasoning

Key Capabilities

Natural Language Understanding: Context, intent, nuance

Code Generation: Multiple programming languages

Text Generation: Creative writing, summaries, translations

Reasoning: Multi-step problem solving, analysis

Content Creation

Customer Support

Education

Research

Other major models: Claude (Anthropic), Gemini (Google), LLaMA (Meta)

Multimodal AI: Vision + Language

Vision Understanding

- Image classification & object detection
- Visual question answering
- Scene understanding & captioning
- Document analysis & OCR

Key Models

CLIP - Image-text understanding

GPT-4V - Vision-enabled GPT-4

Gemini - Native multimodal model

Real-World Applications

Medical Imaging

Diagnostic assistance from X-rays, MRIs

Autonomous Vehicles

Scene understanding for navigation

Accessibility

Image descriptions for visually impaired

Content Moderation

Automated detection of harmful content

Multimodal models bridge the gap between visual and textual understanding

Generative AI: Image Generation

GANs (2014)

Generator vs Discriminator

StyleGAN, BigGAN

Diffusion Models (2020)

Iterative denoising process

DALL-E, Stable Diffusion, Midjourney

VAEs

Latent space encoding

Continuous generation

Diffusion Models: How They Work

Forward Process: Gradually add noise to training images over many steps

Reverse Process: Learn to remove noise step-by-step to generate images

Text Conditioning: Guide generation with natural language prompts

Quality: State-of-the-art photorealism and artistic control

Art & Design

Creative workflows

Marketing

Ad generation

Gaming

Asset creation

Education

Visual learning

Stable Diffusion, DALL-E 3, and Midjourney democratized AI-generated imagery

AI Agents & Future Directions

AI Agents: Beyond Conversation

Tool Use: Call APIs, search databases, execute code

Memory: Maintain context across sessions

Planning: Break down complex tasks into steps

Autonomy: Make decisions and take actions

Technical Frontiers

- Longer context windows (1M+ tokens)
- Better reasoning and planning
- More efficient training methods
- Improved interpretability
- Multimodal integration (audio, video)

Societal Impact

- Transforming knowledge work
- Democratizing expertise
- Safety and alignment challenges
- Ethical considerations
- Regulatory frameworks

We are in the early stages of an AI revolution that will reshape how we work, create, and learn

The pace of AI development continues to accelerate rapidly

Setting Up Google Colab in VS Code

Why Use Colab with VS Code?

- Free GPU/TPU access from Google
- VS Code's powerful editor features
- Pre-installed deep learning libraries
- No local setup required

Prerequisites

- VS Code installed on your computer
- Google account (free)
- Stable internet connection

Useful Extensions

- **Python:** Microsoft's official extension
- **Jupyter:** Native notebook support
- **GitHub:** For version control

Setup Steps

1. Create a Colab Notebook

Go to colab.research.google.com and create a new notebook

2. Enable Colab Connection

In Colab: Click "Connect" → Select runtime type (GPU/TPU if needed)

3. Open in VS Code

Method 1: In Colab, click the "Open in VS Code" button ( )
Method 2: Use the "google-colab-vscode" extension

4. Authenticate

VS Code will prompt you to sign in with your Google account

5. Start Coding!

Edit your .ipynb files with full VS Code features + Colab's compute power

Tip: Save notebooks to Google Drive for easy access. Runtime disconnects after ~12 hours of inactivity.

Introduction to Vibe Coding

What is Vibe Coding?

A modern, AI-assisted approach to programming that emphasizes exploration, iteration, and natural interaction

- Code with AI assistants (ChatGPT, Claude, Copilot)
- Focus on high-level ideas, not syntax details
- Iterate rapidly through experimentation
- Learn by doing and asking questions

Traditional vs Vibe Coding

Traditional

- Memorize syntax
- Plan everything first
- Debug alone
- Linear process

Vibe Coding

- Ask AI for syntax
- Explore and iterate
- Pair with AI
- Experimental flow

Vibe Coding Workflow

1. Start with Intent

"Build a CNN for image classification"

2. Ask AI for Help

Get code scaffolds, explanations, debugging

3. Experiment & Iterate

Run code, see results, refine approach

4. Understand & Learn

Ask "why" and "how" to deepen knowledge

Perfect for Deep Learning!

- Complex architectures made approachable
- Quick prototyping of model ideas
- AI explains hyperparameters
- Focus on concepts, not boilerplate

Remember: AI is a collaborator, not a replacement. Understanding fundamentals is still essential!

Python for Deep Learning

Why Python?

- Simple, readable syntax
- Rich ecosystem of libraries
- Large community support
- Interactive development
- Excellent for prototyping
- Industry standard

Core Libraries

NumPy: Numerical computing

Pandas: Data manipulation

Matplotlib: Visualization

Scikit-learn: Traditional ML

Deep Learning

PyTorch: Dynamic graphs, research

TensorFlow: Production, deployment

Keras: High-level API

JAX: Auto differentiation

Object-Oriented Programming: Classes enable code reusability and organization

Review Python classes, inheritance, and methods to prepare for framework usage

Python Classes & Objects

What are Classes?

Classes bundle data and functionality together

- Create a new **type** of object
- Define **properties** (data)
- Define **methods** (functions)

```
class Dog:
    "This is a dog class"
    age = 5
    def get_color(self):
        print("grey")
```

Objects (Instances)

Create specific examples from a class

Class: Dog

Blueprint for all dogs

Object 1

Bulldog, age 4

Object 2

Husky, age 6

```
bull_dog = Dog()
print(Dog.age) # 5
Dog.get_color() # grey
```

Classes begin with **class** keyword, similar to functions which begin with **def**

Creating Child Classes (Inheritance)

Inheritance

Create a class that inherits from another by passing the parent class as a parameter

```
# Parent class
class Dog:
    age = 5
    def bark(self):
        print("Woof!")

# Child class
class SmallDog(Dog):
    age = 1
```

Using Child Class

SmallDog inherits methods from Dog:

```
puppy = SmallDog()
print(puppy.age)
# Output: 1
puppy.bark()
# Output: Woof!
```

Child class inherits all methods and can override properties

Key Benefit

Reuse code without duplicating - extend existing functionality

In Deep Learning

Extend nn.Module (PyTorch) or Model (Keras) for custom networks

Pass parent class in parentheses: class ChildClass(ParentClass):

Creating Objects & Calling Methods

Creating Objects

Use the class name followed by parentheses

```
dog1 = Dog()
dog2 = Dog()
dog3 = Dog()
```

Each object is a separate instance with its own data

Accessing Attributes

Use dot notation to access properties:

```
print(dog1.age)
# Output: 5
```

Calling Methods

Use dot notation to call functions:

```
dog1.bark()
# Output: Woof!
```

Important: The self Parameter

When you call a method like **dog1.bark()**, Python automatically passes **dog1** as the **self** parameter

This:

`dog1.bark()`

Is equivalent to:

`Dog.bark(dog1)`

You never explicitly pass self when calling methods - Python handles it automatically!

Constructors: The __init__ Method

What is __init__?

- Special method called a **constructor**
- Automatically runs when creating an object
- Used to initialize object attributes
- Double underscores = special function

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def bark(self):
        print(f"{self.name} says Woof!")
```

self refers to the instance being created - allows each object to have its own data

Using the Constructor

Pass arguments when creating objects:

```
max = Dog("Max", 3)
bella = Dog("Bella", 5)
# __init__ automatically called
print(max.name) # Max
max.bark() # Max says Woof!
bella.bark() # Bella says Woof!
```

Default Values

```
def __init__(self, name="Unknown"):
    self.name = name
dog = Dog() # name = "Unknown"
```

Complete Python Example

Class Definition with Inheritance

```
class Animal:
    def __init__(self, name):
        self.name = name

class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)
        self.breed = breed

    def info(self):
        return f"{self.name} is a{self.breed}"
    def bark(self):
        print("Woof!")
```

Using the Classes

```
# Create objects
max = Dog("Max", "Labrador")
bella = Dog("Bella", "Poodle")

# Call methods
print(max.info())
# Max is a Labrador

max.bark()
# Woof!
```

Key Concepts Review

- **Class:** Blueprint for objects
- **__init__:** Constructor
- **self:** Current instance
- **Inheritance:** Extend classes
- **super():** Parent methods

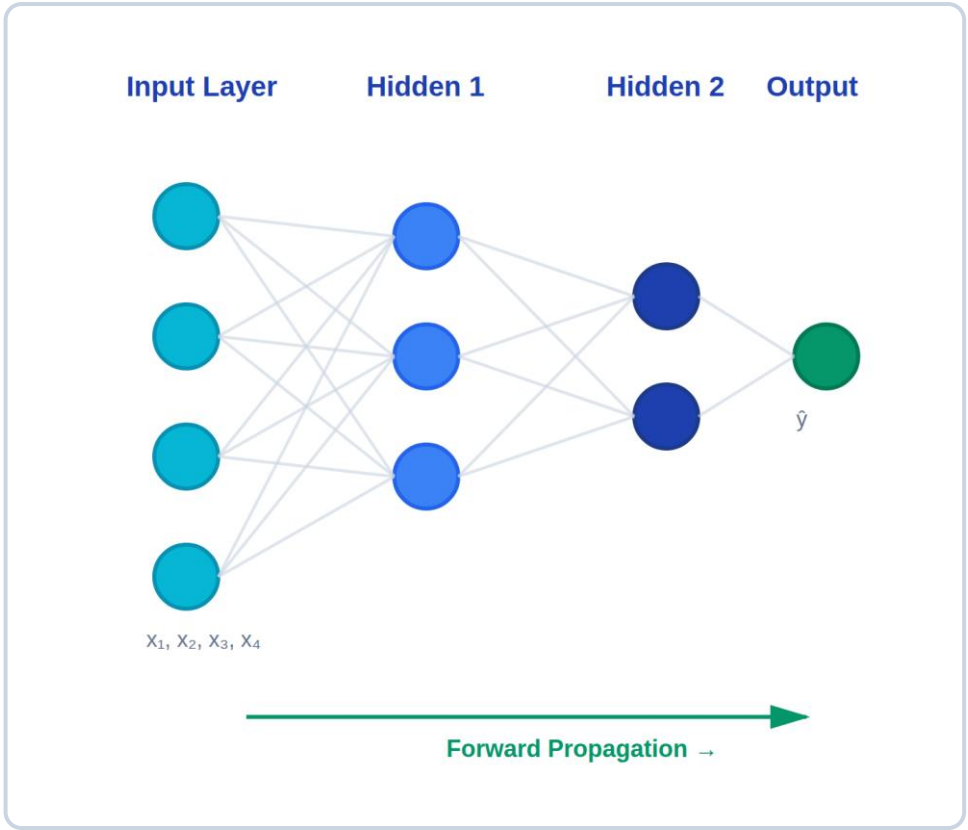
These concepts are essential for building neural networks in PyTorch and TensorFlow!

Fully Connected Neural Network

Architecture

Every neuron in one layer connects to every neuron in the next layer

- **Input Layer:** Receives data
- **Hidden Layers:** Process information
- **Output Layer:** Produces predictions



How it Works

1. **Forward Pass:** Data flows through network
2. **Activation:** Non-linear transformations
3. **Output:** Final prediction

Each connection has a **weight** (learned during training) and each neuron has a **bias**

Each layer transforms data: h = activation(W·x + b) where W = weights, b = bias