Introduction to Machine Learning and PyTorch

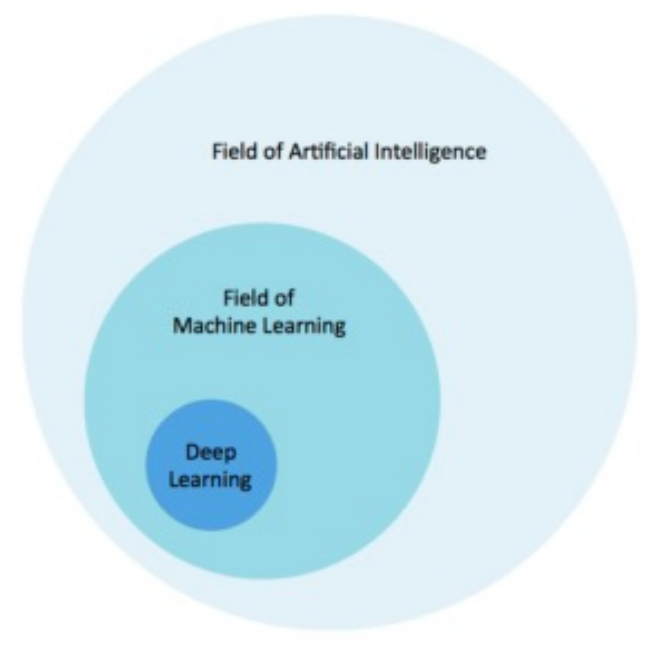
A Guide to Getting Started with AI

Mingzhi Ye

Overview of Presentation

- **Introduction to Deep Learning**
  - Understanding AI, Machine Learning, and Deep Learning
  - The significance of Deep Learning and its applications
- **Introduction to PyTorch**
  - What is PyTorch?
  - Features and Advantages
  - Comparison with Other Frameworks
- **Core Components of PyTorch**
  - Tensors and Autograd System
  - Building Blocks: Neural Networks, Optimizers, Loss Functions

What is Deep Learning?



- **Artificial Intelligence (AI):** The broader concept of machines being able to carry out tasks in a way that we would consider "smart".
- **Machine Learning (ML):** A subset of AI, where enable machines to learn from data, then use algorithms to generalize to unseen data, so that it can perform tasks without being explicitly programmed for those specific outcomes.
- **Deep Learning:** A further subset of ML, utilizing neural networks with many layers. It enables learning from vast amounts of data, mimicking the way humans gain certain types of knowledge.

Introduction to Deep Learning

Why Deep Learning?

- **Advancements:** Deep learning performs much better than traditional methods in many fields, like image recognition, natural language processing, speech recognition, and prediction of protein structure.
- **Impact:** Thanks to deep learning, medical images can be analyzed with high accuracy, assisting in detecting diseases early and saving lives. Search engines are also significantly improved in functionality and efficiency, they are more personalized and do better in understanding user intent as well as improving search relevance.
- **Famous Applications:** AlphaFold2, ChatGPT, Sora.

Introduction to Deep Learning

Key Concepts in Deep Learning

- **Neurons**: The basic building blocks of neural networks, inspired by biological neurons in the brain.
- **Neural Networks:** Comprise layers of neurons that transmit signals from input data and learn to make predictions or classifications.
- **Activation Functions:** Functions like ReLU or Sigmoid that help determine the output of neural network nodes.
- **Loss Functions:** Measure how well the model's predictions match the actual data. Examples include Mean Squared Error for regression tasks and Cross-Entropy for classification tasks.
- **Backpropagation:** A method used to update the model's weights effectively, minimizing the loss function by adjusting weights in the direction that reduces error.

Introduction to PyTorch

What is PyTorch?

- **Introduction:** PyTorch is an open-source machine learning library developed by Facebook's AI Research lab (FAIR), primarily focused on deep learning and artificial intelligence projects.
- **History:** It originated from Torch, a scientific computing framework, and was officially released in 2017. PyTorch quickly gained popularity for its ease of use, flexibility, and dynamic computational graph approach. It's widely adopted in both academia and industry for research and development nowadays.
- **Place in Research and Industry:** PyTorch is renowned for its significant role in facilitating cutting-edge AI research, contributing to advancements in fields such as natural language processing, computer vision, and reinforcement learning. In the industry, it's used by companies for developing models for fraud detection and content recommendation system.

Introduction to PyTorch

Features of PyTorch

- **Dynamic Computation Graphs:** Unlike static graphs in other frameworks, PyTorch uses dynamic computation graphs (also known as define-by-run approach), which allows for more flexibility in building complex models. This feature makes PyTorch intuitive and adaptable, suiting the needs of researchers and developers for rapid prototyping and testing.
- **GPU Acceleration:** PyTorch provides seamless support for CUDA to leverage GPU acceleration, significantly speeding up computations necessary for large-scale neural networks.
- **Ecosystem:** PyTorch has a rich ecosystem comprising libraries such as TorchVision for image processing, TorchText for text and natural language processing, and TorchAudio for audio processing. This ecosystem simplifies the development process and enhances the library's functionality.
- **Tools:** The framework includes comprehensive tools for deep learning, such as PyTorch Lightning for simplifying boilerplate code, and TensorBoard for visualization, facilitating an efficient and streamlined development process.

Introduction to PyTorch

PyTorch vs. Other Frameworks

- **TensorFlow:** Developed by Google, TensorFlow is known for its robust production capabilities and extensive ecosystem. While TensorFlow uses static computation graphs by default (with the introduction of Eager Execution for dynamic graphs), Compared to this, PyTorch's dynamic nature is often preferred for research and development that requires flexibility.

- **Keras:** Keras is a high-level neural networks API running on top of TensorFlow, designed for easy and fast prototyping. While Keras simplifies model development, PyTorch provides a more granular level of control over model design, which can be quite helpful for custom model architectures.

- **Comparison:** PyTorch is often praised for its simplicity and ease of use, especially when it comes to debugging and dynamic computation graphs. TensorFlow uses static graph capabilities, provides better computational efficiency. Keras, as part of the TensorFlow ecosystem, focuses on accessibility, but may lack the lower-level control desired by researchers.

Core Components of PyTorch

Tensors in PyTorch

- **Definition:** Tensors are multidimensional arrays, similar to NumPy arrays, but with additional capabilities for GPU acceleration.
- **Usage:** Fundamental data structure for all computations in PyTorch, supporting operations for deep learning models. All vectors, matrixes are in this format.

```
In [5]:
1 import torch
2
3 x= torch.tensor([[1,2,3],[4,5,6]])
4 y= torch.tensor([[7,8,9], [10,11,12]])
5
6 f= 2*x + y
7 print(f)
```

```
tensor([[ 9,  12,  15],
        [ 18,  21,  24]])
```

Core Components of PyTorch

Autograd System

- **Definition:** Autograd is PyTorch's automatic differentiation engine that powers neural network training. It automates the computation of backward passes in neural networks.
- **Key Points:**
  - Enables automatic calculation of gradients for tensors and model parameters, essential for the optimization process.
  - Facilitates complex computations in the dynamic computational graphs.
  - Simplifies the implementation of backpropagation through its automatic differentiation capabilities.

Core Components of PyTorch

Neural Networks with torch.nn

- **Definition:** The torch.nn module provides the building blocks for creating neural networks. It includes a wide range of layer types, activation functions, and utilities.
- **Key Points:**
  - Offers predefined layers like convolutional, recurrent, linear, etc., to build neural network architectures.
  - Supports custom layers for specialized needs.
  - Provides utilities for weight matrix initialization and other model parameter management.

Core Components of PyTorch

Optimizers and Loss Functions

- **Definition:** Optimizers and loss functions are crucial for training neural networks, guiding the update of model weights based on the output of loss functions.
- **Key Points:**
  - Loss functions quantify the difference between the model predictions and the actual data. Common examples include MSE (Mean Squared Error) for regression tasks and Cross-Entropy for classification.
  - Optimizers like SGD (Stochastic Gradient Descent), Adam can effectively adjust model weights to minimize the loss.

Core Components of PyTorch

GPU Acceleration with CUDA

- **Definition:** PyTorch integrates with CUDA, NVIDIA's parallel computing platform, enabling dramatic increases in computing performance by using the power of GPUs.
- **Key Points:**
  - Simple API calls to move tensors between CPU and GPU.
  - Enables faster model training and inference times.
  - Essential for working with large datasets and complex models.

Core Components of PyTorch

Custom Datasets and Data Loaders

- **Definition:** torch.utils.data provides utilities for loading data, making it easier to feed data into models for training and inference.
- **Data Loaders for Parallelization:** torch.utils.data.DataLoader wraps a dataset and provides an iterable traverse over the dataset with support for batching, sampling, shuffling, and multiprocess data loading. This is key for efficient training, especially when dealing with large datasets that cannot fit into memory.

Advanced Components of PyTorch

Transfer Learning and Pre-trained Models

- **Leveraging Pre-trained Models:** PyTorch offers access to a wide range of pre-trained models, such as VGG, ResNet, and BERT, through its torchvision, torchaudio, and torchtext libraries. These models, trained on large datasets like ImageNet, can be fine-tuned for specific tasks, significantly reducing the time and resources required for training.

- **Transfer Learning Strategies:** Based on pre-trained model, do the training on new dataset to adjust for new task or new scenario. People can choose to only modify the final layers or retrain the entire model (or a significant portion of it) on the new dataset.

Advanced Components of PyTorch

Distributed Training and PyTorch Lightning

- **Distributed Training:** With torch.distributed, PyTorch supports various methods of parallelism, including data parallelism, model parallelism, and hybrid approaches. This facilitates training across multiple GPUs and nodes, reducing training time for large models.

    - **PyTorch Lightning:** PyTorch Lightning is a lightweight wrapper around PyTorch that simplify lots of the boilerplate codes for training loops, validation, logging, and checkpointing. It enables cleaner and more modular code, making advanced techniques like distributed training more accessible.

PyTorch Community and Resources

Community Support, Forums, and Tutorials

- **Robust Ecosystem:** The PyTorch ecosystem includes not only the core library but also a rich collection of tools, libraries, and extensions for tasks ranging from computer vision to natural language processing. Examples include torchvision for image processing, torchaudio for audio data, and torchtext for text-related tasks.
- **Community and Learning:** The PyTorch community is supported by an extensive network of forums, user groups, and discussion channels, including the official PyTorch Discussion Forums, Stack Overflow, and Reddit. These platforms facilitate knowledge sharing, troubleshooting, and collaborative problem-solving.
- **Educational Resources:** PyTorch provides a wealth of tutorials, documentation, and online courses that cover various levels, from beginner to advanced. This includes the official PyTorch tutorials, and also deep learning courses on platforms like Coursera and Udacity, and numerous community-contributed resources and blogs.

Future of PyTorch

- **Continuous Innovation:** PyTorch continues to evolve, with contributions from both academia and industry focusing on ease of use, performance improvements, and support for the latest AI research. Upcoming features and enhancements are regularly discussed in community forums and GitHub, reflecting the community-driven development process.

- **Growing Adoption in Industry and Academia:** PyTorch's flexibility, performance, and ease of use have led to its growing adoption for research projects, commercial applications, and educational purposes. Its dynamic nature makes it particularly suited for rapid prototyping and experimentation, a key factor in its widespread popularity.

- **Collaborations and Integrations:** The future of PyTorch includes deeper integrations with cloud services, GPUs, and other AI ecosystems, making it easier to deploy PyTorch models at scale and in production environments. Collaborations with hardware manufacturers for optimized performance and with academic institutions for cutting-edge research are essential in its roadmap.

Q&A

Thanks