Reading Summary Week 1

Aakash Agrawal
HDSI
University of California San Diego
San Diego, CA, 92092
aaa015@ucsd.edu

1 The Rise of Machine Learning Systems

1.1 AI: Transforming Every Aspect of Life

The author begins by highlighting key applications of AI across various facets of human life. He describes the 21st century as the era of the AI Revolution, drawing parallels to the transformative impact of the Industrial Revolution in the 19th century and the Digital Revolution in the 20th century. He emphasizes that developing reliable, scalable, and safe AI systems that align with human values and interests will play a pivotal role in shaping this revolution.

1.2 The Evolution of AI

The reading outlines key developments and milestones in the evolution of AI through a timeline, highlighting early innovations such as the Perceptron and modern breakthroughs like GPT-4. He identifies five distinct thematic groupings in the evolution of AI:

- **Symbolic AI**: Comprised of well-defined, rule-based approaches that could only handle inputs that exactly matched a pre-defined pattern or sets of rules, thereby lacking generalizability.
- **Expert Systems**: These systems utilized domain-specific knowledge to address niche application areas, such as medicine. However, their development was labor-intensive and time-consuming, with significant challenges in maintenance and handling uncertainty.
- **Statistical Learning**: A transformative phase in AI characterized by the availability of massive datasets, the development of advanced algorithms, and exponential growth in computational power. A common example is the detection of spam emails using the Naive Bayes algorithm.
- **Shallow Learning**: Comprised of computationally efficient foundational methods such as k-NNs, Decision Trees, and regression-based models. These approaches produced interpretable and reproducible results, serving as the cornerstone of the early machine learning era.
- **Deep Learning**: A fundamentally distinct approach to data modeling inspired by the structure of the human brain that autonomously extract features and learn hierarchical representations, much like how humans learn. A key milestone in this field was the introduction of AlexNet, a groundbreaking model that demonstrated the practical potential of deep learning.

Building on this foundation, deep learning entered an era of unprecedented scale, driven by the realization that larger datasets and deeper neural networks could address increasingly complex problems. However, this progress also brought significant challenges in system design, prompting practitioners to explore ways to bridge the gap between theoretical advancements and practical implementations. This shift ultimately paved the way for the emergence of the machine learning systems engineering field, which framed **ML systems** more broadly, considering not only algorithms but also data and computational power.

Quoting from the reading:

> *It's not enough to have a brilliant algorithm if you can't efficiently collect and process the data it needs, distribute its computation across hundreds of machines, serve it reliably to millions of users, or monitor its performance in production.*

1.3 Components of an ML System

ML systems consist of three interrelated components, each dependent on the others for effective operation:

- **Model/Algorithms**: The model architecture determines the compute requirements for both training and inference, as well as the volume and structure of data needed.
- **Data**: The scale and complexity of the data dictate the infrastructure required for storage, processing, management, and serving for both training and inference.
- **Compute**: The hardware and software infrastructure that supports the scalability of models and efficient processing of the datasets.

These interconnected components give rise to an integrated ML Systems Lifecycle, which includes several stages of development with feedback loops throughout:

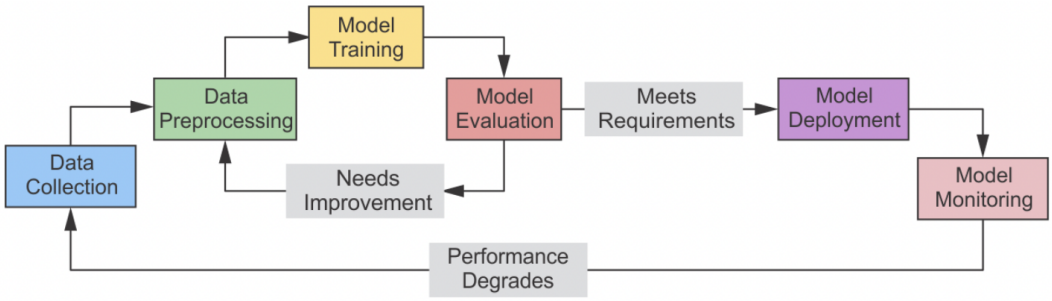

Figure 1: The typical lifecycle of a machine learning system. Source: `https://mlsysbook.ai/`

1.4 Real-world Case Studies

The author also discusses several real-world ML system case studies and their impact, including:

- **FarmBeats**: Demonstrates the potential of distributed ML systems in transforming resource-constrained environments like agriculture. These systems deliver AI-driven insights to farmers, helping reduce water usage and waste while boosting crop yields through optimized resource allocation. They employ edge computing capabilities and a range of deep learning techniques for computer vision, personalization, and the integration of diverse, distributed data sources.
- **AlphaFold**: Showcases the use of ML systems to tackle complex scientific challenges, such as predicting the 3D structure of proteins. It utilizes multifaceted datasets and leverages advanced neural network architectures like attention mechanisms, for modeling and relies on clusters of GPUs and TPUs for training. This application has the potential to transform fields like drug discovery and computational biology.
- **Waymo**: The autonomous driving application showcases the practical implementation of complex ML systems that utilize a wide range of computing infrastructure, from embedded systems to cloud platforms. These systems leverage an ensemble of modeling techniques, including CNNs and reinforcement learning, while also considering interpretability aspects. Waymo's impact highlights the potential of ML systems to revolutionize transportation and urban planning, paving the way for safer, more efficient mobility solutions.

1.5 Challenges and Considerations

Building and deploying ML models in production incurs a plethora of challenges and considerations:

- **Data Challenges**: ML systems face challenges not only in processing massive volumes of data but also in cleaning and representing it effectively for modeling, requiring sophisticated data cleaning and validation processes. Additionally, issues such as data drift during inference can render models ineffective. To remain reliable, ML systems must detect these drifts and adapt dynamically.
- **Model Challenges**: Modern ML models, often consisting of billions of parameters, present significant challenges in training and maintenance. Their large size and high computational requirements make inference power-intensive, complicating deployment in production. Additionally, these models may struggle to generalize effectively to real-world scenarios. Ensuring that ML models operate efficiently, accurately, and reliably in production is critical, particularly in high-stakes domains such as medicine and autonomous vehicles.
- **System Challenges**: Encompasses operational challenges such as monitoring, effective debugging, and implementing sanity checks to ensure the end-to-end pipeline functions reliably.
- **Ethical and Societal Considerations**: Many deep learning models, such as large language models (LLMs), are often considered as "black boxes" due to their limited interpretability and explainability. This opacity can lead to ethical concerns, including issues of bias, fairness, and transparency, especially in critical areas like law and healthcare.

2 DNN Architectures from the lens of System Design

The chapter centers on exploring how different deep learning architectures align with system resources and the ways in which these demands can be optimized. The author delves into the computational patterns of architectures such as MLPs, CNNs, RNNs, and Transformers, examining how they correspond to and utilize system resources. This analysis considers several critical factors, including memory access patterns, computational requirements, and data movement.

2.1 Multi Layer Perceptrons (MLPs)

The MLP architecture manifests fundamental computational patterns that appear throughout the realm of deep learning. They act as universal function approximations and, hence, are quite versatile in nature. A key feature of MLPs is **Dense Pattern Processing**, which enables unrestricted feature interactions, learned feature importance, and adaptive representation based on the data. This is achieved through a series of **multiply-accumulate** steps, a fundamental pattern that drives key system design decisions:

- **Memory Requirements**: Learning an MLP requires the system to store and access weights, inputs, and some intermediate outputs. Here, memory access is optimized through careful data organization and reuse. CPUs can leverage their cache hierarchies for data reuse, while GPUs can employ specialized memory hierarchies for high-bandwidth access.
- **Computation Needs**: Learning an MLP involves dense matrix multiplication operations. Both CPUs and GPUs can make use of cache locality by tiling the matmul operation to maximize data reuse.
- **Data Movement**: Each multiply-accumulate operation requires input values, weights, and the running sum. The system needs to move these data from memory to computational units creating substantial data transfer demands. CPUs can make use of prefetchers and cache hierarchy, while GPUs can make use of massive threading for data transfer and staging optimization.

MLPs introduced many fundamental concepts that became the cornerstone of deep learning: the power of layer stacking, the importance of non-linear transformations, and the basic feedforward computational pattern.

2.2 Convolutional Neural Networks (CNNs)

CNNs are a special type of deep learning architecture used for modeling spatial relationships, i.e., how neighboring pixels relate to each other in image data. A key feature of CNNs is the **Spatial pattern processing** that enables hierarchical feature representations in the network. These convolutions offer several advantages, such as parameter sharing and sparsity of connections, that also help contain overfitting. Let's see how this distinct pattern (again composed of multiply-accumulate patterns) influences system design and allows room for optimization:

- **Memory Requirements**: During learning, the system is required to store two main components: filter weights and the feature maps (and inputs). An optimized memory access pattern involves reusing filter weights and careful feature map management and storage. Both CPUs and GPUs can use their cache hierarchies for filter reuse across different spatial positions.
- **Computation Needs**: The primary computation of CNN involves repeatedly applying filters (local multiply-accumulate operation) across spatial positions. We can benefit from GPUs' massive parallelism that divides convolution tasks across many threads, whereas CPUs can use SIMD instructions to process multiple filter positions simultaneously.
- **Data Movement**: Here, the system must stream the input features through the computation unit while keeping the filter weights stable. The repetitive and predictable nature of these convolutions allows processors to optimize data movements. CPUs can use cache to keep filters stable while streaming through the input/feature maps. GPUs can employ memory architectures optimized for spatial locality and provide hardware support to make sliding window operation more efficient.

2.3 Recurrent Neural Networks (RNNs)

RNNs are a special type of deep learning architecture used for modeling sequential relationships between elements over time. A key feature of RNNs is the **Sequential pattern processing** that enables the model to handle variable length inputs and maintain and update an internal context over time. This sequential nature leads to computational patterns quite different than CNNs and MLPs.

- **Memory Requirements**: The system is required to store two sets of weights (input-to-hidden, hidden-to-hidden), along with an internal hidden state. CPUs can use their cache hierarchies to carefully manage hidden states and maintain weight matrices in the cache, while GPUs can make use of specialized memory architectures for maintaining states. Deep learning frameworks can also help optimize memory access via Batching.
- **Computation Needs**: The core computation involves applying weight matrices (again, a multiply-accumulate operation) across time steps. The sequential dependency prevents us from parallelizing across time steps. Despite this constraint, GPUs can batch multiple sequences for parallel processing.
- **Data Movement**: Each time step must load the previous hidden state, access weight matrices, and store the newly computed hidden state. CPUs maintain weight matrices in caches while streaming through the sequence elements and managing the hidden state. GPUs employ specialized memory to maintain hidden state information and allow for processing multiple sequences in parallel.

2.4 Attention Mechanisms and Transformers

2.4.1 Basic Attention Mechanisms

Attention mechanisms is a special method used in deep learning models that enable dynamic relationships between elements in the input. These form the foundation of a peculiar processing paradigm known as **dynamic pattern processing**, which has the flexibility to modify its dataflow graph based on the input itself. Here, the architecture is not fixed but rather emerges from the data itself.

- **Memory Requirements**: The system is required to store the attention weights, key-query-value projection matrices, and intermediate feature representations. The dynamic nature of these weights leads to significant memory usage.

- **Computation Needs**: The core computation involves generating attention weights and applying them to values (multiply-accumulate operations); others include computing projection matrices and applying softmax. The overall computation incurs a $O(N^2)$ time complexity.
- **Data Movement**: Each attention operation involves projecting and moving the query-key-value vectors for each input, storing and frequently accessing the attention matrix, and coordinating the movement of value vectors for computing the attention output.

2.4.2 Transformers and Self-Attention

Transformers represent a powerful dynamic pattern processing architecture that uses an evolution of attention mechanisms called Self-Attention. Let's explore the implications of this dynamic architecture on system design:

- The self-attention mechanism processes all elements in the sequence simultaneously (in parallel), allowing for significant speedups during training. Each input in the sequence can attend to all other inputs at once without relying on the order of processing, which enables highly efficient parallel computation on modern hardware like GPUs.
- The attention weight matrix computation incurs $O(N^2)$ time complexity, which can become a significant bottleneck when processing long sequences.
- The multi-head attention mechanism efficiently runs many self-attentions in parallel. This incurs a $O(N)$ linear computation load, N being the number of heads.
- The core computation in self-attention is still the same multiply-accumulate operation. Although they contribute to the major computational cost of the model, these intensive matrix multiplications can greatly benefit from hardware like GPUs.

2.5 System-level Building Blocks aka Primitives

The system-level requirements of these deep-learning architectures can be thought of as being made up of some fundamental primitive operations. We can have different types of primitive operations:

1. **Core Computational Primitives**: This encompasses operations that form the fundamental building blocks of deep learning. These primitives include **matrix multiplication**, **sliding window operations**, and **dynamic computation**, which are indivisible into smaller components. The implementation of these primitives determines how data is stored, accessed, and moved within the system.
2. **Memory Access Primitives**: This encompasses access patterns that dominate deep learning architectures: **sequential** access, **strided** access, and **random** access. While sequential access aligns well with modern memory systems via supports such as burst mode in DRAM, prefetchers, etc., hardwares support strided access through pattern-aware caching strategies and specialized memory controllers. Because of their unpredictable access patterns, random access incurs the greatest challenge for system efficiency.
3. **Data Movement Primitives**: This consists of primitives that characterize how information flows through the system. Often these operations can be more time and energy consuming than the computation itself. These primitives include four fundamental access patterns: **broadcast**–sending the same data to multiple destinations simultaneously; **gather**–collecting data from multiple sources; **scatter**–distributing a subset of computation across different destinations; and **reduction**–combining different values into a single result.

These primitives form the core requirements that shape the design of deep learning systems. They drive advancements in specialized hardware, such as TPUs and tensor cores in GPUs, and the development of sophisticated memory hierarchies like high-bandwidth memory, on-chip memory hierarchies, etc, all aimed at enhancing the efficiency of deep learning systems.