Week 1 Reading Summaries: Chapter 1 and 4

Amogh Patankar
University of California, San Diego
3869 Miramar Street Box #3037, La Jolla, CA- 92092
apatankar@ucsd.edu

Abstract

The concept of Machine Learning Systems is a special concept that is the bridge between theoretical foundations of AI/ML, and the hands-on, practical engineering required to implement those theories. These notes serve as an introductory piece, and a deep dive into DNN Architectures.

1 Chapter 1: Introduction

1.1 AI is Everywhere

AI is a superhuman force that is transforming how we live, from infrastructure and healthcare, to more scientific research. The AI Revolution is advancing at an unprecedented pace, and the potential is massive- potentially life changing at a global scale. This rapid evolution is an engineering challenge, i.e. developing systems that can achieve superhuman capabilities in many domains.

1.2 Understanding AI/ML

Intelligent systems are the key to AI, and the replication of human behavior, learning, adapting, and reasoning, is the fundamental problem at hand. ML is used to build these systems, and the relationship with AI is similar to the connection between theory and practice. This progression drives the creation of systems that are similar to intelligent behavior.

1.3 Evolution of AI

AI has changed in the past 7 decades, starting with the perception in 1957, all the way to GPT4 in 2023. The point being, the early AI systems don't even compare to the model ML systems that do a wide variety of applications and tasks at the present day.

1.3.1

AI was coined at the Dartmouth Conference in 1956, based on the idea that intelligence is equal to symbol manipulation, thanks to the STUDENT system. STUDENT and similar AI could only process input that they stored. The limitation revealed a deep problem, that AI at the time could only match patterns.

1.3.2

MYCIN was developed at Stanford, to diagnose blood infections in the 70s. Again, we saw challenges- that domain knowledge and converting it to rules was difficult, and doctors for example, could process better than the model. The challenges of knowledge and handling uncertainty was something that is still a concern in modern ML.

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

1.3.3

Statistical learning was built off the premise of Moore's Law, that algorithms like SVMs and NNs could learn patterns because of a larger amount of data and compute. Statistical learning changed the approach towards building AI due to: quality/quantity of data, evolutional, and precision and recall tension.

1.3.4

Shallow learning (a precursor to DL) was due to powerful algorithms like decision trees and KNNs process data well, but was limited to small model sizes. Yet, they were relevant for nearly a decade.

1.3.5

Deep learning was the crux of modern ML, relying on larger layers of neurons, with each layer transforming data to different representations. Breakthroughs like AlexNet and others led to more and more powerful models being developed, off the premise of complex, deep networks. This foundation led to unprecedented scaling, with models 1000x the size of AlexNet. The progression from symbolic reasoning to DL allowed scaling, and for applications to become more complex.

1.4 Rise of MLSysEng

The shift in the past 10 years built on the idea that engineering was required to build strong ML, i.e. ML systems. The idea of advancing HW, SW, EE, and ML together created MLSysEnd, i.e. the interplay between algorithms and engineering to advance ML.

1.5 Definition of MLSys

A ML System is defined as an integrated system constituting of data, learning algorithms, and computing infra. Each aspect enhances/dictates the other two, and the interdependency of these aspects indicates no independence.

1.6 MLSys Lifecycle

The lifecycle of ML Systems consists of data collection and preprocessing, model training and evaluation, deployment and, finally, monitoring. These interconnected stages allow for smooth and reliable improvements in model performance, and systems. Each piece limits the improvement of the system, and problems can compound.

1.7 Spectrum of MLSys

ML Systems range from tinyML systems running on microcontrollers, and scale all the way up to cloud-based LLM Systems in data centers. Regardless, there is a broad spectrum of ML systems, with various capabilities depending on size, compute requirements, resources, and limitations of applications.

1.8 MLSys Implications on ML Lifecycle

A bottle neck of the ML lifecycle is ML Systems- hardware often time determines architectural decisions, and resource management varies drastically. Resources considerations influence both model and system architecture, and complexity increases with system distribution. The tradeoffs of compute, architecture, complexity, and systems requires a delicate balance.

1.8.1 Emerging Trends

There is innovation in application, as well as system architecture. App-level innovation can handle system changes, allowing advancements. System architecture changes require resource efficiency to advance, and hardware such as AI accelerators to advance as well. Both advancements allow systems to grow sustainably.

1.9 Real-World Applications

There are diverse ML system applications.

1.9.1 FarmBeats

ML in non-tech industries requires lots of specifications to succeed. For example, in FarmBeats, the data ecosystem is diverse, the ML models are tailored to agricultural applications, and the compute allows models to run on [small] IoT and edge devices. All in all, FarmBeats showed great deployment of ML systems in real-world environments.

1.9.2 AlphaFold

In the case of AlphaFold, the same aspects (data, algorithm, and infra/compute) were a point of success. A complex data pipeline with preprocessing, novel NN architectures, and exceptional parallel computing resources allowed for true innovation.

1.9.3 Autonomous Vehicles

For Waymo, the data was of a different manner- namely, data from LiDAR sensors, generating 1 Tb/hour. The structured and unstructured nature of this data was the unique aspect, along with its algorithm and compute, that allowed Waymo to be effective, and to potentially revolutionize transportation and urban planning.

1.10 Challenges + Considerations

There are many challenges in ML Systems that are important.

1.10.1 Data Challenges

Data management in ML Systems poses a unique challenge- as high quality and large quantity data is hard to obtain, leading to techniques like synthetic data generation. ML Systems require large amounts of data, and this cale poses storage/processing challenges as well, alongside data drift.

1.10.2 Model Challenges

ML models, which can be as complex as hundreds of billions of parameters, create problems such as situations with limited resources. The training process has trade-offs which are a challenge, especially in real world conditions.

1.10.3 System Challenges

System challenges include training and serving systems being up to par, meaning that monitoring and updating must also be smoothly carried out.

1.10.4 Ethical + Social Considerations

Fairness and transparency in ML Systems is a long-standing problem that researchers have only started to dive into and make advances in. Privacy, especially in the high tech modern world, is a major concern, especially with data and systems themselves.

1.11 Future Directions

The future of MLSystems is bright, and trends such as democratization of AI, efficient ML, and autonomous ML systems, are the future of the field. As such, the limitations posed by AI and ML systems are difficult to tackle, leading to more research in the future.

2 Chapter 4: DNN Architectures

2.1 Overview

Deep learning architectures are how models that we use in deep learning are organized- the depth, breadth, complexity, and structure. Neural networks have evolved from simple pattern recognition, all the way to generating text, audio and video, due to more complex architectures. Architectures are usualyl discussed w.r.t. their alrogithmic basics, but each pattern maps differently to HW resources. Specifically, memory access patterns, computation characteristics, data movement, and resource allocation.

2.2 Multi-Layer Perceptrons

MLPs are the "easiest" deep neural net architecture, consisting of mathematical computations (matrix multiplications), and systematically progressing through layers.

2.2.1 Pattern Processing Needs

DL Systems run into problems where input is directly influential on output, and the solution is dense pattern processing. This allows for output to be somewhat independent of input combinations, and feature importance is introduced, allowing systems to determine the most important connections. Moreover, adaptive representations lets the network(s) reshape representations based on input data.

2.2.2 Algorithmic Structure

MLPs have fully connected layers, and this translates to MatMul operations. Essentially, the formula looks like this: $\mathrm{h}^l = f(\mathrm{W}^l n^{l-1} + b^l)$

This creates patterns to be handled effciently by computer systems.

2.2.3 Computational Mapping

There are a few layers to computational mapping. The first computation is the simple mathematical equation in the previous section, i.e. the matmul of X and W, with a bias, fed into an activation. The second computation involves batch processing, computing neuron outputs, and passing them through an activation function. This translation from mathematics to computation shows the desnity of matmul operations.

2.2.4 System Implications

Three fundamental dimensions exist for computation patterns- memory requirements, computational needs, and data movement, which enables analysis of system design decisions. With respect to memory requirements, the weights, inputs, and results impact memory requirements. Optimization opportunities present themselves through data organization/reuse. For computation, MAC (multiply-accumulate) operations are nested; this lends itself to optimization strategies, such as BLAS. Finally, data movement allows for data staging and transfer optimization.

2.3 CNN: Spatial Pattern Processing

CNNs are the best models for encapsulating spatial relationships, namely, pixels. As such, CNNs have different needs and approaches than MLPs.

2.3.1 Pattern Processing Needs

Spatial processing is specifically useful for data depending on relative positions, such as an edge in an image. So, we require the patterns to retain their structure/"shape", i.e. pattern localization and pattern recognition, independent of position. CNNs use local connections to achieve this, known as convolution (set of weights applied).

2.3.2 Algorithmic Structure

$$\mathbf{H}^{(l)}_{i,j,k} = f(\sum_{di}\sum_{dj}\sum_{c}\mathbf{W}^{(l)}_{di,dj,c,k}\mathbf{H}^{(l-1)}_{i+di,j+dj,c} + \mathbf{b}^{(l)}_{k})$$

Figure 1: With respect to CNNs, the core operation looks like this.

This operation process local regions, reuses the same weights, and maintains spatial structure, differing from the MLP architecture. This convolution operation moves throughout the image as a filter.

2.3.3 Computational Mapping

The first map is a convolution operation, i.e using kernel, input, bias, and an activation. The second map applies the same weight filters to local regions. The result is a m x m filter, with n output channels, with significantly less MAC ops than MLP.

2.3.4 System Implications

Here we look at- memory requirements, computational needs, and data movement. CNNs use reusable filters, unlike MLPs, and hence require much less memory, as we reuse weights and optimize in that manner. With regards to computational needs, the load remains large due to spatial repitition despite fewer ops than MLPs. Finally, the window of convolutions and reused weights allows for streaming input.

2.4 RNN: Sequential Pattern Processing

While MLPs are tailored to handle arbitrary relationships, and CNNs handle spatial, RNNs are meant for sequential data [patterns].

2.4.1 Pattern Processing Needs

For RNNs, the primary use case is NLP, meaning that a word's meaning is dependent on the previous sequence. That is, the challenge is maintaining context, and RNNs have variable-length sequences.

2.4.2 Algorithmic Structure

RNNs use a different structure, by using "recurrent connections", allowing RNNs to maintain a state that gets updated. Sequential processing uses a high-dimensional vector for each word, and uses an equation like the one that follows:

$$\mathbf{h}_t = f(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_h)$$

Figure 2

2.4.3 Computational Mapping

Computational mapping for RNNs is a single time step, with input, hidden state, and two weight matrices, passed through an activation function. Nested loops then process each sequence (batch parallelism), previous hidden states, new information, and biases + activations.

2.4.4 System Implications

For RNNs, the memory requirements, computation needs, and data movement differ than that of a MLP or a CNN. We store two sets of weights, and a hidden state, meaning memory usage is heightened. By extension, computational needs cannot parallelize across steps due to sequential nature, and similarly, weights can be reused for data flow.

2.5 Attention Mechanism

The attention mechanism was specifically built for language understanding, i.e. to capture relationships between words rather than just position.

2.5.1 Pattern Processing Needs

The attention architecture can dynamically process relationships between two words, and systems must be able to compute relationships, weigh them, and use weights to combine information.

2.5.2 Basic Attention Mechanism

1. **Algorithmic Structure:** The real power of the attention mechanism is computed by using queries, keys, and values, representing learned projections of input. The calculation is Attention(Q, K, V) = softmax$((QK^T * \mathrm{d}_k{}^{-1/2}) * V$. The Q, K, V projection get calculated, an N x N attention matrix is formed, and we combine value vectors, using attention weights, generating an output.

2. **Computational Mapping:** The mapping is the mathematical abstraction described above, with nested loops for sequence processing, attention compute, key comparison, and attention score.

3. **System Implications:** For the attention architecture, there must be storage for attention weights, KQV projections, and feature representations. Similarly, the system performs an enormous amount of MAC operations, and by extension, dynamically computed weights must move frequently, requiring high data movement.

2.5.3 Transformers and Self-Attention

Transformers extend the idea of attention by applying it for a single sequence, resulting in one element "attending" to all others.

1. **Algorithmic Structure:** The innovative aspect of transformers is due to self-attention layers, in which QKV are determined from the same input, meaning the weights can be calculated while encoding positions. This allows it to capture long-range dependencies; transformers generally use multi-head attention, with multiple sets of QKV projections.

2. **Computational Mapping:** The computational mapping of attention is increased drastically, due to the introduction of self-attention, and multi-head attention simultaneously.

3. **System Implications:** Self-attention allows for parallel processing for all positions; the attention score calculation has quadratic complexity w.r.t sequence length, which is a massive bottleneck. Moreover, the multi-head attention is a parallel run of self-attention ops, which linearly increases the power. And finally, the computations themselves, and intermediate results are all very memory heavy and dominant.

2.6 Architectural Building Blocks

2.6.1 Perceptron to MLP

The transition from perceptron to MLP was a new idea that is in every new modern architecture, such as the MLP feed forward style networks in transformers. The idea of data transformation through non-linear layers and backprop, is a building block of modern ML.

2.6.2 Dense to Spatial Processing

CNNs revolutionized data processing, with the ideas of parameter sharing, and skip connections. These ideas allowed for more efficient networks, but also direct paths for gradient flow, and of course, batch normalization.

2.6.3 Evolution of Sequence Processing

LSTMs and GRUs allowed gating, which eventually lead to a pattern in architecture design, with the idea that networks could process variable-length inputs by reusing weights, and eventually led to attention.

2.6.4 Modern Architectures: Synthesis and Innovation

Modern architectures are built from an evolution of NNs, from MLPs to CNNs to RNNs and finally using attention in the modern era. The ideas of self-attention (allowing for dynamic processing) and FF with skip and normalization, has changed modern language modeling.

And of course, with change, comes various utilization ideas, such as this:

Primitive Type	MLP	CNN	RNN	Transformer
Computational	Matrix Multiplication	Convolution (Matrix Mult.)	Matrix Mult. + State Update	Matrix Mult. + Attention
Memory Access	Sequential	Strided	Sequential + Random	Random (Attention)
Data Movement	Broadcast	Sliding Window	Sequential	Broadcast + Gather

Figure 3

2.7 System-Level Building

2.7.1 Memory Access Primitives

With the newest architectures, we must also look at various primitives for memory access. Sequential access allows for memory like DRAM to read quickly, while strided access for CNNs requires software frameworks to change this data to sequential access. Rnadom access posing the largest threat for efficiency is the reason why LMSys is such a large domain to work and research in.

2.7.2 Data Movement Primitives

When we talk about data movement, we look for on-chip vs off-chip, as one requires much more energy than the other. Broadcast, scatter, gather, and reduce are key to distribute and collect data across computational units. Broadcasting- sends data to many destinations, scatter- distributes different data to different locations, gather- collects from multiple sources, and reduce combines multiple values to one result.

2.7.3 System Design Impact

The effect of all these primitives leads us to expore different designs for systems in ML. Operations like matmul, convolution, and MAC lead to changing of processing units. Balancing the tradeoff needs us to evaluate the workload and deployment, to choose the right memory and data primitives to make good decisions.