CPE504

Artificial Neural Networks (ANNs)

5.0. DEEP LEARNING COMPUTE:

PROBLEMS AND PERFORMANCE OF NEURAL NETWORK ALGORITHMS

What you will learn

- Function Approximators
- Memory and Computation in Digital Computers
- Performance
- Computational Complexity: Time and Memory
- Computational Complexity: MLPs
- The Problems of Hidden Layers
- Deep Learning: Classifiers
- Summary

Function Approximators

- In **computing**, we can view everything as a **function**.
- **Software: ML models** like **ANN architectures** have been justified to be useful based on **two notions**:
- **Universal function approximation (to arbitrary precision).** *Note this with high suspicion*
- **Ease of parameterization**
- An important question is: for such models, **how do we find the control-input parameters called weights** (`W`)?
- We have seen that we usually start with a **loss function** parameterized in a **least-squares sense** expressing the **performance index** of our model w.r.t to the learned `W`. Then we attempt to **minimize** that index (**zero-finding**) in an **iterative** fashion, hoping for quadratic **convergence** of the model to the **true distribution** of the training data samples.
- The system (or model) identification process above has been called **training a model.**
- **MLPs** which underpins other advanced architectures is a *function transformation* of **real-valued input vector spaces to real-valued output vector spaces.**

Function Approximators

- Further, due to the hidden layers that may be stacked together, we can also view **MLPs** as **function compositions by matrix multiplication**. This principle generalizes to almost all **ANNs.** The study of matrices is fundamental in linear algebra.
- Let the input features be $\mathbf{x} = \mathbf{y_0}$. Assume there are $\mathbf{n_k}$ nodes in each layer.

input to nodes in layer k : $\mathbf{v}_{\boldsymbol{k}} = \mathbf{W}_{\boldsymbol{k}}\mathbf{y}_{\boldsymbol{k-1}}$ **matrix multiply ops.**

output of nodes in layer k : $\mathbf{y}_{\boldsymbol{k}} = \boldsymbol{\sigma}(\mathbf{v}_{\boldsymbol{k}})$ **vector ops.**

- **So, what is a matrix?** A matrix is a 2D array or tensor, that is an array with **row space dimension and a column space dimension.**
- $\mathrm{A} \in \mathrm{R}^{\mathrm{i} \times \mathrm{z}}$ $\mathrm{B} \in \mathrm{R}^{\mathrm{i} \times \mathrm{z}}$
- Then $\mathrm{C=AB} \Rightarrow \mathrm{c_{ij}} = \sum_{\mathrm{z}} \mathrm{a_{iz}}\, \mathrm{b_{zj}}$
- Each element is a **sum of product** of size **z** carried out **ij** times.
- This makes a total of **ijz** *operations* for **matrix multiply** and **ij** *operations* for **element-wise vector multiply**.

Function Approximators

- Today, the great success of deep learning is a result of **increase in computational power** obtained from digital computers.
- Interestingly, as the size of the **MLP increases** either in terms of **nodes per layer** (horizontal **scaling**) or **number of layers** (vertical **scaling**), its compute demand increases **but** its function approximation performance (inference) does not necessarily scale.
- This knowledge then leads us to: how do we **analytically measure** the computational performance of these algorithms. How **efficient** are they in terms of **run-time** and **memory-storage use**? *Asymptotic Measures for Computational complexity*.
- A common **upper-bound** (**worst-case**) asymptotic measure is the **big-O notation**. It gives the worst-case estimate for the **number of arithmetic steps** or the **number of bytes** an algorithm will take compared to the **size of the input or target data** being processed.
- To continue, it is useful to review some basics

Memory and Computation in Digital Computers

- The basic unit of digital storage (memory) is 1 bit (binary digit). 1 byte (B) = 8 bits
- Today, we have RAMs in GBs (10^9 B), Disk-drives in TBs (10^12 B).
- Big data storage are in EBs (10^18 B) - ZBs (10^21 B) range and beyond
- A common question are digital computers designed to represent real numbers.
- There is the IEEE-754 standards describes **floating-point number systems** which are function abstractions for **approximating** real numbers (decimals) and **performing** floating-point operations (FLOPs) on a digital computer within certain machine precisions (e.g.: 32, 64 128 bit precisions).

Recommended: Study the following articles on **FLOPs**.

- https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html
- https://arxiv.org/abs/2012.02492
- https://web.stanford.edu/class/ee380/Abstracts/170201-slides.pdf
- https://randomascii.wordpress.com/2013/07/16/floating-point-determinism/
- https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/

Memory and Computation in Digital Computers

Some **notes**:

- **Overflow** can cause unsafe code (crashes)
- **Underflow**, on the other hand is safer, as it silently yields zeros or denormalized numbers.
- **Catastrophic cancellation** can occur when two numbers of very much different magnitudes are algebraically added by the digital computer
- All these problems occur because what we know as real numbers are represented by bits through a floating-point model on the digital computer.
- When dealing with exponentiation e^-x is safer than e^x

- **Avoid overflows as much as possible**
- **Avoid adding small numbers to numbers with vey large magnitude.**
- **Avoid subtracting two numbers that are nearly equal**
- **Avoid division by zero.**

Memory and Computation in Digital Computers

- So, we go on to ask **Turing's question** again: Can Machines **Think**? We know this digital beasts can **process and manipulate numbers at super-human speed**. But like Turing said, it all depends on **how we define machines thinking**.
- Interestingly, our deep learning algorithms **think with the use of floating-point (FP) number systems**.
- Unfortunately, these floating-point systems are sometimes error prone and leave much to be desired in terms of performance of accuracy. Researchers are thinking of better ways to approximate real numbers.
- One newer real-number representation system that is gaining traction is universal number systems (**unums**) called **Posits.** *The 3-decade work of one researcher.* *https://posithub.org/about*
- A key advantage of **Posits** is that we can get **more precision** and **dynamic range** using **less bits** than **IEEE-754 FPs**. *This is a better way to think with digital computers, hence an increase in the speed of our deep learning algorithms, and reduction in energy-use costs.*
- There is no fabricated chip using posit arithmetic to date. https://posithub.org/docs/PDS/PositEffortsSurvey.html **The big question: when will chips designed to use Posits arithmetic be produced in large-scale.** **https://www.nextplatform.com/2019/07/08/new-approach-could-sink-floating-point-computation/**

Memory and Computation in Digital Computers

Rethinking floating point for deep learning

- https://arxiv.org/abs/1811.01721
- https://engineering.fb.com/2018/11/08/ai-research/floating-point-math/
- https://www.sigarch.org/posit-a-potential-replacement-for-ieee-754/
- **Digital Computer systems** are **machines designed** to perform **information processing and computation.** Their **performance** is typically measured by **how much information processing they can accomplish per unit time.**
- Deep Learning algorithms have high computational requirements (demands) on the digital computer in terms of speed and memory storage.
- See: https://www.sigarch.org/the-new-bottlenecks-of-ml-training-a-storage-perspective/
- Therefore, an efficient number system for high-performance thinking (computing) can then not be overemphasized.

Performance

- We have in **CPE507** discussed some **key properties of software:** Correctness, clarity, maintainability, testability, reliability, portability, etc.

However the real currency of software is **performance in terms of speed and memory efficiency**. Today, the printing press of computing performance is **clock frequency**.

- High computing is done with big parallel multi-core processors with complex caches, wide vector units, hyperthreading, dynamic frequency scaling and so on to perform parallel executions of matrix operations.

How do we **write software to utilize modern hardware effectively**.

- 1. make effective use of memory through linear algebra libraries like BLAS (LAPACK)
- 2. use divide and conquer algorithms a much as possible
- 3. write native vectorized code as much as possible (avoids expensive temporary memory allocations as much as possible)

A lot of **software libraries** (Tensorflow, Torch, etc.) use these hacks to instruct the compiler to take advantage of vector instructions and processor intrinsics in order to compile vectorized code, that is take advantage of processing data in a **SIMD** fashion (**Single Instruction Multiple Data Stream**), and take advantage of vectorized hardware: TPUs, GPUs, DPUs, etc.

Matlab is at its core an high-level language wrapper for BLAS.

- *Very specifically, we are concerned with the* **amount of storage memory required** *when an algorithm executes on a problem size n (***space complexity***) and the* **number of key operations executed** *by an algorithm in* problem size n.

Computational Complexity: Time and Memory

- An **Algorithm** is loosely defined as a **function**: a set of instructions for doing something. It must be definite, have an i/o, and be effective.
- **Classical algorithms** typically come with **strong** general guarantees. These invariances can be stated as preconditions, postconditions and *how time-space complexity scales with input size*
- Unfortunately, **deep learning algorithms** come with **weak** guarantees on the pre-conditions and post-conditions
- The basic unit for measuring algorithm efficiency is FLOPs: Floating-point operations per seconds.
- Floating-point operations are enabled by **FPUs** (*floating-point units*) on a **CPU chip**.
- The *independent* **CPUs** in a single computing component (die or chip) are known as **Cores**.
- **Fast memory accessible** locations associated with CPUs on a chip are called **Caches**. (on-chip memory for easier **Data movement)**
- **FPUs** are *basic ALU hardware units* that deal with **floating-point ops**: add, subtract, multiply, divide, compare and the overhead fetch and store operations. Very accurate FPU hardware have what is called *fused multiply-add units* **(FMA) .**
- $\mathbf{c \longleftarrow c + a{*}b}$
- A single **fused** *multiply* operation followed by an *add* operation (**1 FMA)** can be counted as 1 *mult* plus 1 *add***.** Looping through the operation above for vectors of length **n**, we have **n** *mult* and **n** *add* making
- $\mathbf{n\ FMA \equiv 2n\ FLOP}$
- **Peak performance in FLOPS = cores x clock x FLOPs/cycle.** E.g : A 2.9GHz PC with 2 processor chips, 4 cores per chip, 8 flops per cycle DP (FMA) = 2*4*2.9*G*(8*2) = 856GFlops

Computational Complexity: Time and Memory

- Let’s say the two double precision (8B) floating-point vectors are of size n = 375000.
- Data-size for the two vectors: 2*(8B*n) = 6MB.
- For a cache of 6MB and peak memory transfer speed of 25.6GB/sec, the two vectors can fit in the cache.
- **Time** to move vectors from memory to cache is 6MB/25.6GB/s = 0.23ms
- **Time** to multiply two vectors (dot product) is: 2n flop / 856Gflop/sec = 0.9us
- **Max Time** = **max**(0.01us,0.23ms) = 0.23ms
- *We see that memory access time is a bottle-neck.*
- To measure asymptotic run-time efficiency of algorithms, an **upper-bound measure** in the worst-case is the **big-O notation**.
- For instance T(n) = O(g(n)): there exists two real-valued positive constant c and n, such that T(n) is at most as large as c g(n) for all n that is at least as big as g(n). This means given a problem size n, the number of steps it takes the algorithm to run is proportional to another function g(n) which is the upper-bound limit (big-O) of that algorithm. **T(n) <= c g(n)**
- E.g an algorithm that runs in 100n time, has a **big-O of n**, that is: g(n) = n, because: 100n <= **c** n. Where **c** can be interpreted as the **number of flops.**
- In general,
- **Vector-vector multiply ab** is 2n flops **O(n)** flops
- **Matrix-vector Ab** is 2mn or 2n^2 flops **O(n^2)** flops
- **Matrix-Matrix AB** is 2mpn or 2n^3 flops **O(n^3)** flops

Computational Complexity: Time and Memory

- In practice, we want to avoid quadratic-time O(n^2) and above running algorithms. Slower. They are not practical when dealing with large input size (large datasets).
- **Orders of Computational Complexity**

Exponential-time (NP-hard, inefficient)

- **Factorial** O(n!)
- **Exponential** O(b^n)

Polynomial-time (average)

- **Polynomial** O(n^q) q > 1 – doable, nested loops
- **Linearithmic** O(n log n) – fast, divide and conquer algorithms (Quicksort, FFT)

Sublinear (**super fast**)

- **Linear** O(n)
- **Logarithmic** O(log n) – binary search
- **Constant** O(1)

- Note. Initially sort and DFT functions were **O(n^2) operations**. It was John Tuckey that developed two of the most celebrated and important algorithms in use today: **Quick sort** and **FFT** functions, **which have asymptotic run-time of O(nlogn)**

Computational Complexity: MLPs

MLP:

Let the input features be $\mathbf{x} = \mathbf{y_0}$. Assume there are $\mathbf{n_k}$ nodes in each **k** layer.

- **input to nodes in layer k :** $\mathbf{v}_k = \mathbf{W}_k\mathbf{y}_{k-1}$ *matrix multiply ops.* **O(n^3)**
- **output of nodes in layer k :** $\mathbf{y}_k = \sigma(\mathbf{v}_k)$ *vector ops.* **O(n)**
- total asymptotic run-time of forward propagation**: O(n^4) –** *polynomial time*

GD Learning by BP: at each iteration

- $\mathbf{e}_k = \mathbf{W}_k\boldsymbol{\delta}_{k-1}$ *matrix multiply ops.* **O(n^3)**
- $\boldsymbol{\delta}_k = \frac{\mathrm{d}\mathbf{y_k}}{\mathrm{d}\mathbf{v_k}}\mathbf{e}_k$ *vector multiply ops.* **O(n)**
- $\Delta\mathbf{W}_k = \alpha\,\boldsymbol{\delta}_k\,\mathbf{y}_{k-1}$ *matrix (vector-vector) multiply ops.* **O(n^3)** becomes **O(n^2)**
- total asymptotic run-time of backpropagation: **O(n^6) –** *polynomial time*

Computational Complexity: MLPs

- Therefore, we see that the *BP run-time is more than the typical MLP' forward runtime by* **a worst-case factor of O(n^2).** This implies the storage space required also increases.
- In other words, *learning (training) neural nets is far slower than the inferencing with neural nets*. See. https://slideslive.com/38922815/efficient-processing-of-deep-neural-network-from-algorithms-to-hardware-architectures?ref=recommended-presentation-38923183
- This means when we double the training data size n, we expect the run-time to increase by a factor of 2. Again, this means when we **increase the hidden layers we also expect the run-time to increase.**
- The best or least run-time for ANNs is till an **active research area** in computing: *algorithm efficiency, provably approximately correct* (PAC) models of computational learning (that is the tractability of a learning problem).
- Most of the success of deep learning today, are **due to incremental improvements** but still **no really critical technologies**.
- It was the problem of hidden layers that made NNs to go out of favour, until the success of CNNs in the mid 2000s, yielding dazzling levels of performance, ever since. However, there are still many grey areas and controversies.
- **The core problem of hidden layers is that Stacking layers of MLPs together increase the number of matrix multiplications, hence make higher demands on limited computational resources.**

The Problems of Hidden Layers

- This poor performance was attributed to poor training:

Vanishing gradients in the hidden layers

Overfitting of the weights in the hidden layers

High compute demand due to the stacked hidden layers

- What's the point of stacking layers when the backpropagated gradients become approximately zero, and so can't learn. T**o solve this** the logistic function was *eliminated as the activation of the hidden nodes* and the rectified linear unit (RELU) became popular. The **RELU** is a sharp approximation (half-wave rectification) of the integral of the logistic function, which has been called **soft-plus** (full-wave rectification).
- RELU: $\mathrm{y} = \max(0, \mathrm{v}),\ \frac{d\mathrm{y}}{d\mathrm{v}} = \begin{cases} 1\ \mathrm{v} > 0 \\ 0\ \mathrm{v} \leq 0 \end{cases}$
- Other hacks for reducing vanishing gradients: **residual connections**. Today we have other varied MLP architectures that have motivations to reduce vanishing gradients like: **Convolution Networks** CNNs, **Recurrent Networks** RNNs (LSTMs), **Attention Networks** (Transformers), etc. *The fundamental ideas of the MLP underpins all these architectures, which have also been used as* **autoencoders** for the purpose of learning representations
- An **active research area** on neural architectures in the MLP sense is **Network Architecture Search** (NAS) typically using evolutionary optimization algorithms.

The Problems of Hidden Layers

- Complicated structures are prone to overfitting.

Regularization techniques: Make the massive network simple as possible during learning.

- **Dropouts** at each epoch,
- **Batch-normalization**,
- **Residual connections**,
- **Norm regularizers**.
- **Early Stopping**
- Introduced by **Hinton**, Dropout using a particular ratio can be implemented as **a random permutation multiply mask.**
- **For example: 50% drop-out ratio:** $\mathbf{y} = \begin{bmatrix} 2 \\ 1 \\ 1.2 \\ 3 \end{bmatrix} \mathbf{m} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \mathbf{yd} = \begin{bmatrix} 0 \\ 1 \\ 1.2 \\ 0 \end{bmatrix}$
- The use of **more** training data, for example by **data augmentation** helps **reduce the bias** in the learning process. However, it leads to **higher demand** on computing resources. Today an **active research area** is *how to build architectures that use small data but generalize well*.

Deep Learning: Classifiers

- We can view **classification** as choosing a **probability distribution**. This can be binary-class or multi-binary class. A common way of encoding the classes is to use **one-hot encoding.**
- **For example A 3-class distribution encoding:**
- Class 1: 0 0 1
- Class 2: 0 1 0
- Class 3: 1 0 0

Soft-max

- The extension of the simple logistic function to output normalized vectors that relatively sum to 1 is styled the **soft-max** function. It is still the logistic function (*normalized for maximum input argument takes all*).
- Heavily used today in most architectures.
- $\mathbf{y} = \frac{\mathbf{e^{v}}}{\sum \mathbf{e^{v}}} = \frac{\mathbf{1}}{\mathbf{1} + ((\Sigma \mathbf{e^{v}}) - \mathbf{e^{v}}) \, \mathbf{e^{-v}}}$
- $\mathbf{v} = \begin{bmatrix} 2 \\ 1 \\ \text{-}1 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 0.71 \\ 0.26 \\ 0.03 \end{bmatrix}$

Deep Learning: Classifiers

- An ANN model in development that takes one month to run or train can only be modified twelve times in a year. **This makes useful research/experimentation limited or impossible in certain cases**.

- **Deep Learning Architectures** as classifiers have however dominated as the *SOTA (state-of-the-art)* in certain areas of *ML or Signal Processing*:
- **Image recognition**
- **Speech recognition**
- **Natural Language/Text recognition**

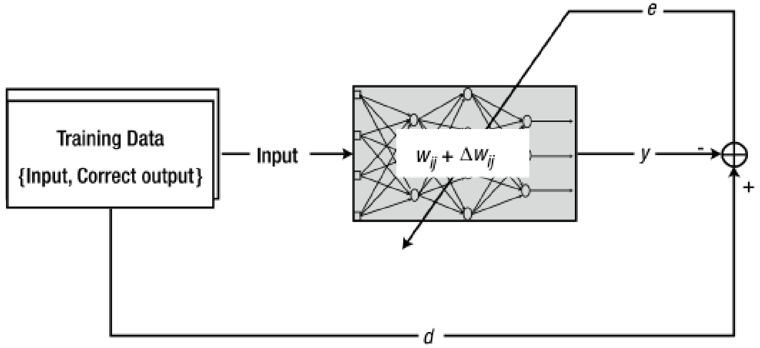
- You can always replace **recognition** with **processing** or **classification**.

- **TODO**: Develop an MLP that can recognize the MNIST digits.
- *Take note of your design choices and trade-offs, training-time speed. Did you obtain a good model? How did you arrive at this conclusion?*

Summary

Recap:

- **ANNs** *are a critical AI technology. Efficient Processing of these algorithms is a very important area of research in terms of* **latency, throughput, power and energy costs** *when the digital computer is used.*
- https://slideslive.com/38922815/efficient-processing-of-deep-neural-network-from-algorithms-to-hardware-architectures?ref=recommended-presentation-38923183
- https://www.sigarch.org/posit-a-potential-replacement-for-ieee-754/
- https://www.sigarch.org/the-new-bottlenecks-of-ml-training-a-storage-perspective/



Tools

Recommended Languages

- MATLAB (Fast Matrix Prototyping)
- Julia (Fast Prototyping and more)
- JavaScript (Language of the Web)

Instructions to Student

- **All ANN functions will be written from scratch in form of custom libraries**
- **Learn to transfer maths to software.**
- **Copying of another person's code (or work) will be heavily penalized.**

Recommended Texts

Main Texts

- **MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence** by Phil Kim
- **Pattern Recognition and Machine Learning** by Christopher M. Bishop
- **Understanding Machine Learning: From Theory to Algorithms** by Shai Shalev-Shwartz and Shai Ben-David
- **PATTERNS, PREDICTIONS, AND ACTIONS: A story about machine learning** by Moritz Hardt and Benjamin Recht
- **Mathematics for Machine Learning** by Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong

Good luck!