Memory Optimizations for Machine Learning

Tejas Chopra

Agenda

- Introduction
- Memory footprint in ML
- Data Quantization
- Model Pruning
- Efficient Mini-batch selection
- Hardware considerations
- Future directions and research

Introduction

- Sr. Software Engineer, Netflix
- Co-Founder, GoEB1
- Tech 40 under 40, BCS Fellow
- TedX speaker: Cloud, Blockchain
- ex-Box, Datrium, Samsung, Cadence

Memory Footprint in ML

- ML is transforming industries: healthcare, finance, ecommerce
- Models are growing in complexity
  - Memory usage is increasing
- Optimizations for memory enable
  - Training larger models with less resources
  - Deploying models on resource constrained devices
  - Reducing infra costs and energy consumption
- Challenges
  - Balancing memory efficiency and model performance
  - Diverse hardware platforms & architectures

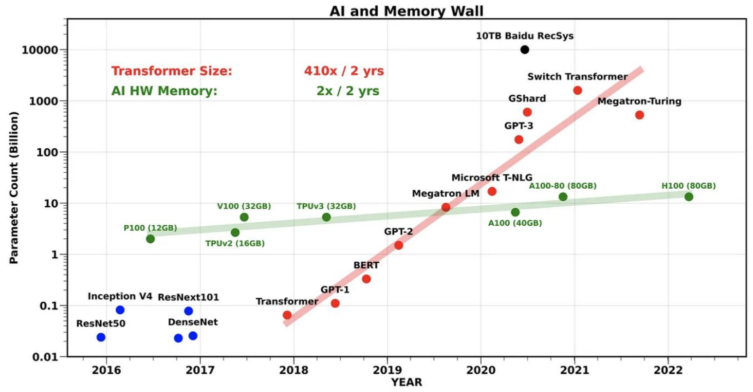
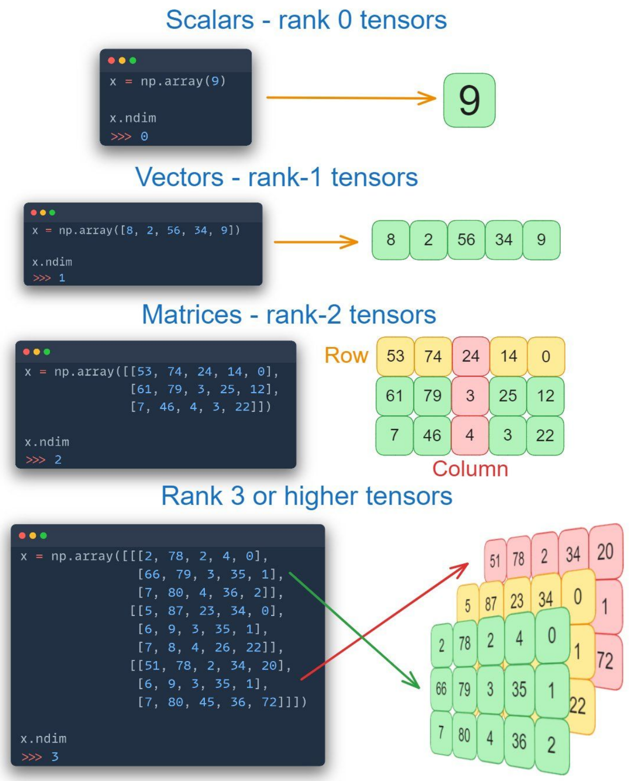


Figure 2: The evolution of the number of parameters of SOTA models over the years, along with the AI accelerator memory capacity (green dots). The number of parameters in large Transformer models has been exponentially increasing with a factor of 410x every two years*², while the single GPU memory has only been scaled at a rate of 2x every 2 years.*³ [Download This Image]

Tejas Chopra, Sr. Engineer, Netflix

Memory Footprint in ML

- ML models consist of data structures like tensors and matrices
  - Example: A 2D tensor of shape (1000, 1000) with 32-bit floats requires 4 MB of memory
- Memory consumption during training:
  - Storing model parameters, gradients, and optimizer states
  - Intermediate activations and backpropagation buffers
- Memory allocation and deallocation:
  - Dynamic memory management during forward and backward passes
  - Potential for memory leaks and fragmentation
- Factors impacting memory footprint:
  - Model architecture, depth, and width
  - Batch size and input data dimensions



```
x = np.array(9)

x.ndim
>>> 0
```

```
x = np.array([8, 2, 56, 34, 9])

x.ndim
>>> 1
```

```
x = np.array([[53, 74, 24, 14, 0],
              [61, 79, 3, 25, 12],
              [7, 46, 4, 3, 22]])

x.ndim
>>> 2
```

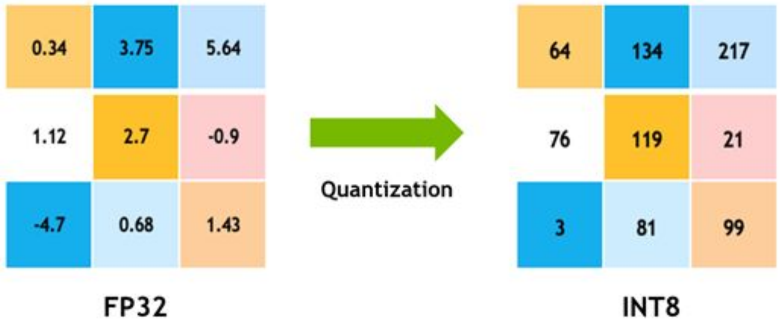
```
x = np.array([[[2, 78, 2, 4, 0],
               [66, 79, 3, 35, 1],
               [7, 80, 4, 36, 2]],
              [[5, 87, 23, 34, 0],
               [6, 9, 3, 35, 1],
               [7, 8, 4, 26, 22]],
              [[51, 78, 2, 34, 20],
               [6, 9, 3, 35, 1],
               [7, 80, 45, 36, 72]]])

x.ndim
>>> 3
```

Tejas Chopra, Sr. Engineer, Netflix

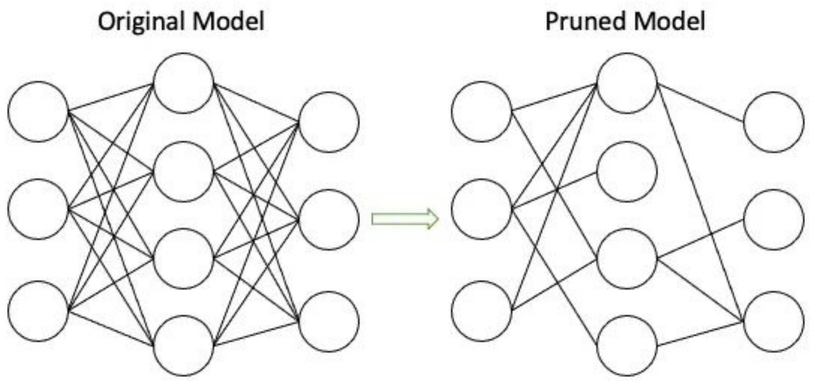
Data Quantization

- Data quantization: Reducing the precision of data representations
  - Example: Converting 32-bit floats to 8-bit integers
- Benefits of quantization:
  - Reduces memory footprint by 50-75%
  - Faster computation and inference times
- Quantization techniques:
  - Uniform quantization: Equal-sized intervals
  - Non-uniform quantization: Variable-sized intervals based on data distribution
- Post-training quantization vs. quantization-aware training
  - Post-training quantization: Quantizing trained models
  - Quantization-aware training: Incorporating quantization during training



Model Pruning

- Model pruning: Removing unnecessary or redundant model parameters
- Pruning techniques:
  - Magnitude-based pruning: Removing weights with small absolute values
  - Structured pruning: Removing entire neurons, filters, or channels
- Iterative pruning and fine-tuning:
  - Gradually pruning the model over multiple iterations
  - Fine-tuning the pruned model to recover performance
- Benefits of pruning:
  - Reduces memory footprint by up to 90%
  - Faster inference times and energy efficiency



Efficient Mini Batch Selection

- Mini-batch selection: Dividing training data into smaller subsets
- Impacts of batch size on memory usage:
  - Larger batch sizes require more memory for intermediate activations and gradients
  - Smaller batch sizes have lower memory requirements but may impact convergence speed
- Strategies for efficient mini-batch selection:
  - Dynamic batch size adjustment based on available memory
  - Gradient accumulation: Performing multiple forward and backward passes before updating weights
- Hardware considerations:
  - Optimal batch sizes differ for CPUs, GPUs, and specialized accelerators
  - Memory hierarchy and bandwidth impact batch size selection
- Best practices:
  - Profile memory usage for different batch sizes
  - Experiment with different batch sizes and gradient accumulation settings
  - Consider trade-offs between memory efficiency and training speed

Hardware Considerations

- Memory optimization techniques vary across hardware platforms
- CPUs:
  - Leverage cache hierarchy and data locality
  - Vectorization (SIMD) for parallel processing
  - Memory alignment for efficient access patterns
- GPUs:
  - Utilize high-bandwidth memory (HBM)
  - Coalesced memory access for efficient data retrieval
  - Maximize occupancy and minimize data transfer between CPU and GPU
- Specialized accelerators (e.g., TPUs, FPGAs):
  - Leverage on-chip memory for fast access
  - Optimize dataflow and computation graphs
  - Exploit low-precision arithmetic and structured sparsity

Example use cases

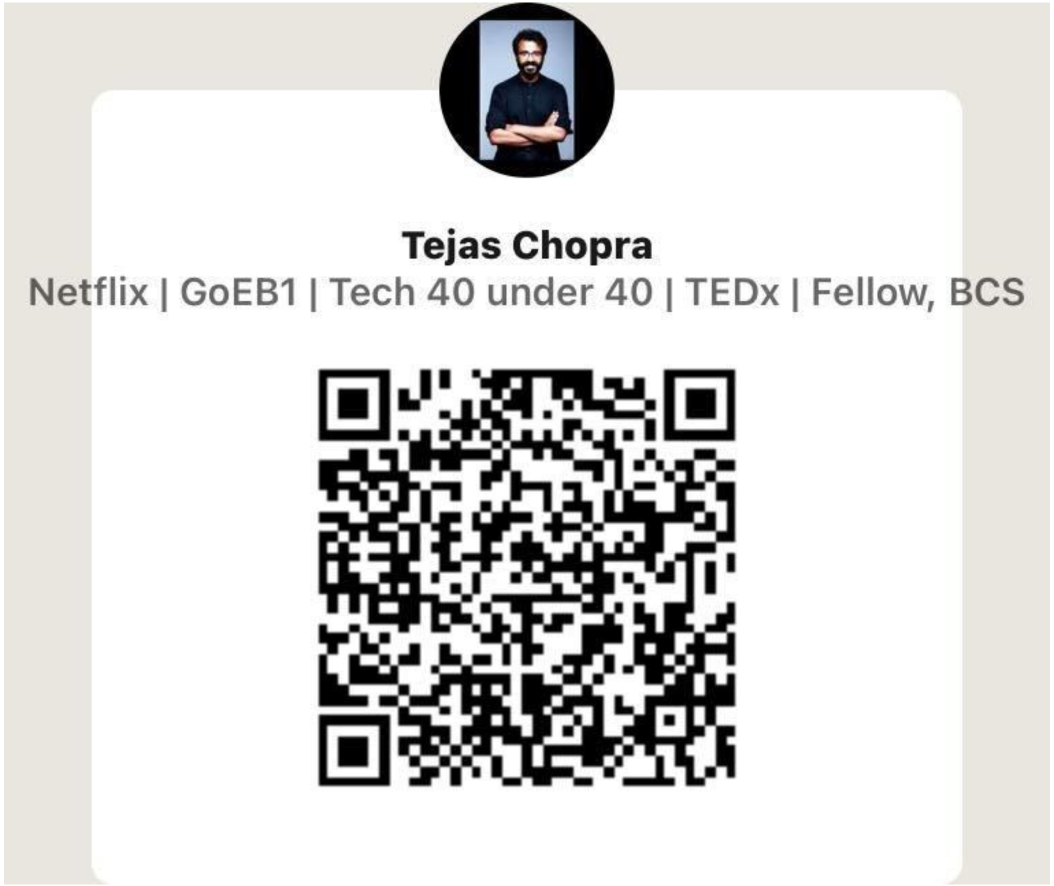
1. Image Classification with MobileNetV2:
    - Applied 8-bit quantization to reduce memory footprint by 75%
    - Achieved 70.2% top-1 accuracy and 89.5% top-5 accuracy on ImageNet
    - Improved inference speed by 21% on a Qualcomm Snapdragon 845 processor
2. Language Translation with Transformer Model:
    - Pruned 80% of weights while maintaining BLEU score within 0.1 of the original model
    - Reduced memory footprint from 512 MB to 102 MB
    - Enabled deployment on resource-constrained devices for real-time translation
3. Recommendation System with Collaborative Filtering:
    - Applied mixed-precision training with 16-bit floats and 32-bit accumulation
    - Reduced memory usage by 50% without impacting accuracy
    - Scaled to handle larger datasets and provide personalized recommendations to millions of users

Future Directions & Research

- Neural Architecture Search (NAS) for memory-efficient models
  - Automated search for optimal architectures balancing performance and memory usage
  - Promising results in discovering novel memory-efficient architectures
- Quantization-aware training (QAT)
  - Jointly optimizing model parameters and quantization parameters during training
  - Improved accuracy compared to post-training quantization
- Sparse representations and computation
  - Leveraging sparsity for memory optimization
  - Techniques like sparse matrix multiplication, sparse convolutions, and sparse attention mechanisms
- Hardware-software co-design
  - Jointly optimizing hardware architectures and software algorithms for memory efficiency
  - Custom accelerators tailored for low-precision arithmetic and structured sparsity
- Memory-efficient transfer learning
  - Adapting large-scale pre-trained models to resource-constrained environments
  - Techniques like model compression, knowledge distillation, and parameter sharing

Conclusion

- Memory optimization is crucial for efficient and scalable ML deployments
- Techniques like data quantization, model pruning, and efficient mini-batch selection can significantly reduce memory consumption
- Hardware-aware optimization is essential for maximizing memory efficiency on diverse platforms
- Real-world case studies demonstrate the impact of memory optimization in various domains
- Future research directions offer promising avenues for further advancing memory optimization in ML
  - Neural Architecture Search, quantization-aware training, sparse computation, hardware-software co-design, and memory-efficient transfer learning



Tejas Chopra

Netflix | GoEB1 | Tech 40 under 40 | TEDx | Fellow, BCS