Scaling to Meet the Growing Needs of AI
Pradeep Dubey (Intel) and Dr. Amir Khosrowshahi (Nervana)

Artificial Intelligence (AI) is a simple vision where computers become indistinguishable from humans; to think and behave like one of us. AI needs computers, but it has primarily relied on human-crafted focus, insights, and heuristics. It has been human-centric, but even that human focus has centered specifically on human experts. The reliance on experts (especially experts in the fields initially used in AI research such as medicine) has been the challenge in growing AI, because experts do not scale.

AI now relies on machine learning, which is a class of algorithms that improve over time. This happens through data crunching and improvements to hardware and methodology. It’s always been a principle of AI but wasn’t practiced because the necessary large quantities of data were not available to be crunched and developed. Now, the amount of data doubles every year, growing at a rate faster than compute power. This is the real reason why we are discussing AI and the reason why machine learning is an effective tool for realizing the power of AI.

Artificial Intelligence, Machine Learning, and Neural Networks
As neural networks are a class of algorithms included in the scope of machine learning within the AI field, there is a group within neural networks, a “sub-subclass” of deep neural networks. These have more than two hidden layers of input and output. Figure 1 shows a single layer of input and output, with the relationship weights connecting them. In practice, deep neural networks have over a hundred layers of input and output.

![Figure 1. A single layer of a neural network](image_url)

The subclass of algorithms within machine learning known as neural networks are the algorithms that receive the most attention. The focus of this paper will be on machine learning and the subclass of machine learning algorithms that are neural networks. The relationship is shown in Figure 2.
Give the deep neural network an image, and it will find the person you are looking for, labeling them with a box (or contour). This is when the network is properly trained. The task of going from input to output is called “forward propagation” and the term used is “inferencing.”

For this to work, neural networks must be trained. The training is the challenge.

**Training**

What does it take to train a network? You start, go through forward propagation, and look at the results. You look at the difference between what it should have said and what it did say, take the difference, and propagate the difference back (Figure 3). This is called backward propagation. The backward propagation algorithm is the hardest part. Weights at each edge are adjusted. This is done carefully, layer by layer, for multiple training scenarios (images, in this case).
There are two primary challenges with current state-of-the-art training schemes: a) they rely on labelled data (supervised training), and b) the algorithmic parallelism is very limited. Required labelling of data for training, even though labor intensive, does not require experts, rather ordinary humans, often drawing a box or counter around the target object. Algorithmic parallelism is mostly limited to the batch of images that are processed together to learn average attributes at a point. Large batch sizes do not work very well on training data, hence the batch size is generally kept limited to around a thousand.

Across the large numbers of input data (images) and the depth of the network, training proceeds mostly serially a layer at a time, and the network itself is very large (with millions of different weights to be determined), so training time can be long. For the most use-cases, the length of training (weeks, even months) is unacceptable. Since the target task of inferencing cannot even begin without a reasonably trained model, we want quicker turnaround for training and hence the time-to-train becomes the primary metric of goodness.

Since the metric is the time it takes to train the model, how do we reduce the time to train? Machine learning itself cannot begin until we have a complete model.

Good training leads to a good, compact model. As a comparison, imagine a caricaturist who can describe a person in five brushstrokes. There are an infinite number of brushstrokes available. Which five brushstrokes, where they are applied, and how they are applied, is the result, and even an expert artist often can’t really explain how they did it, so it cannot be reproduced readily. The machine learning algorithms, once trained with enough data, can learn the right compact representation without needing any help from a human expert, which can be used for recognition or recreation of the original image.

Self-driving cars are an excellent example of the machine learning process. Within the vehicle there are sensor processing, and sensor data capture, path planning, and driving control functions. At the data center, there is the vehicle endpoint management, vehicle simulation and validation, and captured sensor data analytics across a fleet of such cars. Machine learning can happen on either end. There can be car-specific processing on the car side, and wider, cross-vehicle processing on the data center side. Primarily, most of the inferencing and processing happen on edge devices and the cloud, but it must be done end-to-end as well, so that cross-device learning happens.
Neural Network Layers

A fully connected layer is a layer where all inputs are connected to all outputs in the layer (outputs being the results). There is input, weights, and output in a layer. Fully connected layers are simply matrix processing, using matrix mathematics. Even other non-fully connected, convolutional layers, in a neural network are often processed using matrix multiplications. Matrix multiplication is where more than 90 percent of the calculations are done, and it is mostly dense linear algebra. In other words, the core kernels of neural networks are quite compute-friendly and any architecture designed to do dense linear algebra should do very well.

Ultimately, the matrix calculations required are as shown in Figure 4. Forward propagation – input matrix, weight matrix and the calculation of the output matrix. Backward propagation looks at output matrix, weight matrix, and calculates the input matrix right back. For the weight update you compare input and output deltas and calculate the new weights (Figure 4).

\[ I \in \mathbb{R}^{K \times N} \quad \text{Input} \]

\[ W \in \mathbb{R}^{M \times K} \quad \text{Weights or model} \]

\[ O \in \mathbb{R}^{M \times N} \quad \text{Output or activations} \]

Forward propagation: \((M \times K) \times (K \times N)\)

Backward propagation: \((M \times K)^T \times (M \times N)\)

Weight update: \((M \times N) \times (K \times N)^T\)

Figure 4. Matrices in a single layer and the matrix calculations

Parallelism

The types of parallelism are data parallelism, model parallelism, and hybrid parallelism.
In data parallelism, you split the data in two to run through two different nodes using the same weights in parallel. The weights are unchanged, but the input and output data is split to run in parallel.

Model parallelism is the same idea as data parallelism, but applied to model (weights). The weights are split and the data is run through half of the weights to run in parallel. This typically happens with fully connected layers, where model size is much larger \( (n^2) \) compared to the amount of data \( (n) \).

Hybrid parallelism (Figure 5) is when you combine model and data parallelism.

![Hybrid Parallelism Diagram](image)

*Figure 5. Hybrid parallelism showing both data and weights split to run in parallel*

How do you know when one system of parallelization is better? As you split the data or the model, the matrix changes and becomes more difficult to work with. For example, we may start with a large and regular (comparable dimension) matrix, and after few splits land up with a tall-skinny matrix (many rows, very few columns) after few splits. Former is friendly to blocked data fetches in processor cache or memory, and wide SIMD parallelism, but latter is not. Sometimes calculations for the same node are split, resulting in additional work to ensure same-node calculations are kept together. Inter-node communication and skewed matrix format become a problem as we continue to parallelize finer-grain for highly parallel architectures.

So, generally:

- Use data parallelism when activations (output) > weights (model).
- Use model parallelism when weights > activations.

The implications of these splits:

- Data parallelism at scale makes activations much smaller than the weights.
- Model parallelism at scale makes weights much smaller than the activations.
- Compute efficiency is reduced because of the skewed matrices (10x4k matrices chew up processor time regardless of the cache, for instance).
Scaling
Communication time starts dominating total compute time as we parallelize to large-scale. As a result, increasing computational power of a node has decreasing benefit for the workload performance measured in terms of overall time-to-train. Therefore, we need to go beyond naïve parallelization schemes to be able to benefit from large computation resources (as in a public cloud) for reducing the time to train large models. Two such optimizations are: use of hybrid parallelism and limiting/managing inter-node communication. These two methods allow high-efficiency scaling. Hybrid parallelism helps prevent the matrixes from becoming too skewed. Further, to limit inter-node communication, we propose creating node groups such that we do activation transfers within each group, and weight transfers across groups.

Communication Patterns in Deep Learning
Let us take a deeper look at the types of inter-node communication patterns involved. When you parallelize across nodes, you have need for inter-node communication. For example, if you are multiplying-adding for dot products, where only one element of the matrix is available on one node then partial activations from two different nodes need to be communicated and combined to get the desired dot product. If this data is not in the next layer cannot be computed. More details of such internode communication is shown in Figure 6:

![Figure 6. MPI collectives in Deep Learning](image)

The green lines (Allreduce) can wait, the red lines (Alltoall et al) are necessary so the next layer calculations can be made. Green lines represent communication that is less critical since the data is needed for the next forward propagation phase. Red lines represent more time-critical communication since the data is needed immediately for the next layer computation, and any delay is more likely to
create processing pipeline bubbles, or loss of efficiency can be handled in a different parallel from each other, but the red lines cannot. Therefore, red-line messages need to be scheduled with higher priority.

To summarize, these are the necessary steps for efficient communication patterns:

1) Optimize performance of various inter-node communication primitives so that they take minimum processing time
2) Overlap communication with compute wherever possible to minimize the performance impact of communication
3) Schedule time-critical internode communication messages with higher priority (e.g., red vs. green).

Figure 7 shows the performance improvement (with respect to current Intel MPI implementation) resulting from optimizing the Allreduce primitive.

![MPI Allreduce performance on Intel® Xeon Phi™ processor 72xx series](image)

**Deep learning specific optimizations result in 3X speedup for the Allreduce collective (average for the message profile of 16KB – 16MB floats)**

*Figure 7. MPI Allreduce performance*

Efficiency happens when fewer cycles are spent on communication and more on compute. Figure 8 and Figure 9 illustrate how effective these optimizations are.
Combination of these optimizations leads to overall better scalability, and higher performance for various neural network topologies. Figure 10 shows the scaling results for four popular topologies. These are shown up to 128 nodes, but we are working on scaling to thousands.
After the Time to Train is Met

Once we have developed enough and reached a point where the time to train is acceptable, what happens? The model is compact enough and has speed and accuracy. The trained model now gets deployed on some edge device to run and deliver accurate inferences on actual field data. The scorecard of inferencing is then sent back to the server; next, the server gets enough new data to retrain itself, make an even better model, and deploy to even more devices. Better the inferencing accuracy of the model, more likely it is to be deployed on more edge devices, and hence more feedback to the server to retrain and make the model even better. This is the virtuous cycle of compute (Figure 11).
Better the inferencing accuracy of the model, more likely it is to be deployed on more edge devices, and the more places it is deployed, the more inferencing data it can send back to the server (regarding when it predicts correctly versus incorrectly), which then in turn improves server’s ability to refine and improve the inferencing accuracy of the model. This in turn means the model is now likely to be deployed on even MORE devices...and the cycle continues. This is the virtuous cycle of compute (Fig 11) that we are preparing for.

However, the loop is not complete until we actually send it to the devices. Sending to the device requires slightly more work, because networks are always constrained. Real-time, fast retraining of a model on a datacenter infrastructure becomes harder with long latency and limited bandwidth of the network connecting the edge devices to data center. As the hardware, network, and wireless connections improve, this problem will become easier to tackle.

Further, it’s important to know what the edge device is. Most often the limitations are in compute power or available memory, limiting model size. The tool requirements and the software support needs
must be met end-to-end. Our Deep Learning toolkit takes such model deployment considerations into account.

There are three high-level tradeoffs in model deployment on the edge devices: compression, accuracy, and throughput (Figure 12). There are devices that might not be capable of holding the model, requiring you compress, or shrink, or somehow reduce the model (Figure 13). How can you compress the model? By sparsifying it.\textsuperscript{1, 2}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure12.png}
\caption{Three-axis model performance base}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure13.png}
\caption{Sparsifying FC layers (e.g., Deep Compression\textsuperscript{*}) -> mobile}
\end{figure}

In specific circumstances, one may have the need for higher inferencing throughput, and may be willing to sacrifice the inferencing accuracy for the sake of higher throughput. (Figure 14).
In other words, the requirements of the device and the use context determine the precise tradeoffs you are willing to make.

**Nervana**

In 2016, Intel acquired Nervana, a machine learning industry leader and a platform for machine intelligence in hardware engineering, systems software, machine learning, and cloud.

Nervana’s goal is to build a platform for machine intelligence. This means using computers to create and process large datasets and make inferences on them. The goal is to accelerate the process by optimizing Deep Learning and other algorithms.

However, the purpose of machine learning is to provide solutions for human problems. So we answer the questions, how do people want to use Deep Learning? Where can we use it? The following are some places we can immediately use our machine learning platform:

- **Healthcare** – Medical imaging is one of the biggest areas. Volumetric imaging with MRIs and CT scans, even single images can produce issues. Some static medical images are as much as 200,000 pixels per side; one image alone can be larger than the benchmark dataset. So the computer problems are enormous and we must be able to efficiently scale.
- **Agriculture** – Genomic problems and climate modeling, as well as robot vegetable harvesters, which selectively harvest crops, are used. These require special needs of scaling at the edge as well as in the cloud where you need low-latency inference.
• Finance – There are many use cases here, such as the vast IT problems a financial institution would have. Exchanges for various kinds of financial instruments traded in different ways can use Deep Learning to better focus trading times and methods. These also are used to anticipate potential fraud to protect the exchange against adverse events.

• Automotive – Speech recognition, driver assist, automated driving. These all have massive datasets, collecting even more data. The scale is massive, requiring a full solution that is not only processing on the edge in the car but also at the data center.

Therefore, Deep Learning thus is core technology. The ‘Google Brain’ model is applied by Nervana at each client, with a central core working on all related information (Figure 16).

Figure 16. Nervana’s Deep Learning model versus the Google Brain model

This method of viewing the solution, the engineers on the solution, the deployment, and everything about creating a Deep Learning product, helps customers understand how to look at Deep Learning as a core technology.

The additional focus in Nervana’s Deep Learning approach is cloud deployment. The cloud allows for the fastest deployment of Deep Learning to a client. Exploratory data science and training data models is easier with high-bandwidth, low-latency connections to an elastic, scalable cloud. This is the most digestible form of Deep Learning available. It’s better adopted with a clean interface, low download requirements, and easy submission of information to a cloud service.

Intel’s Role in Driving Machine Learning

Intel is taking care of adoption and improvements in both the hardware and software roadmaps and the frameworks update.
Hardware
Intel® Xeon Phi™ processors, the densest compute solution today, is the best platform for Deep Learning because of the highly parallel architecture (Figure 17). It is also just like Intel® Xeon® processors. All development done for the Intel Xeon processor is directly applicable to Xeon Phi, because the two share the same compute model. With further integration of the memory subsystem and fabric, performance improvements are quite significant.

**Intel® Xeon Phi™ Processor Family for Performance**

Enables shorter time to train

- Breakthrough Highly-Parallel Performance
  - Up to ~6 SGEMM TFLOPS\(^2\) per socket
  - 1.36x better scaling efficiency resulting in lower time to train for multi-node
  - Eliminates add-in card PCIe* offload bottleneck and utilization constraints

- Removes Barriers through Integration
  - Integrated Intel® Omni-Path fabric (dual-port; 50 GB/s) increases price-performance and reduces communication latency for deep learning networks

- Better Programmability
  - Binary-compatible with Intel® Xeon® processors
  - Open standards, libraries and frameworks

*Figure 17. Intel® Xeon Phi™ processor value in Deep Learning performance*

Ease of programming, ability to amortize programming across Intel Xeon and Xeon Phi processors, and parallel architecture deliver performance AND productivity for the Intel® Xeon Phi™ processor 72xx series.

Software
Machine scaling of training and inferencing across multiple nodes was hardware. How well do we do on each node? Intel® Math Kernel Library (Intel® MKL) is the most important library for the node-level optimization. With Intel MKL the improvements are as much as 24x (Figure 18).

*Caffe*/AlexNet **single node** training performance

*Figure 18. Single-node training performance improvements using Intel® Math Kernel Library (Intel® MKL)*
This improvement is only from using the Intel MKL. No recoding was necessary for the Intel MKL. Improvements on multiple nodes would be even greater. All popular deep learning frameworks will be supported and scale across multiple nodes. But there is no need to wait; you can get started now with Caffe* and Intel MKL.

Intel is also releasing a Deep Learning toolkit, with tools to accelerate design, training, and deployment of Deep Learning solutions.

**Summary**

The vision and power of AI is to remove mental drudgery of our life. Machine learning will unlock the true power of AI. It is the key enabler for a new virtuous cycle of compute triggered by an explosion of digital data and ubiquitous connectivity. Machine learning can vastly expand the reach of computing applications like self-driving automobiles, agriculture, health, and manufacturing. These are fields where simple decision making tasks are left to humans. With recent advances in machine learning and AI, these fields will be improved, transformed, and changed for the better.

Compute requirements of deep learning need to be tackled with a sense of urgency. We must scale to bring down the training time from months and weeks to days and hours for complex enough models. Machine learning compute infrastructure must both perform and be productive for developers. It must leverage the efficiency of the cloud.

Scaling distributed machine learning is challenging as it pushes the limits of available data and model parallelism, as well as internode communication. Intel’s new Deep Learning tools (with the upcoming integration of Nervana’s cloud stack) are designed to hide/reduce the complexity of strong scaling time-to-train and model deployment tradeoffs on resource-constrained edge devices without compromising the performance need.

**Notes**


**Notices**

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at [intel.com](http://intel.com).

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.
Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel, the Intel logo, Intel Xeon Phi, Intel Xeon, and the Intel inside logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

© 2016 Intel Corporation