What is So Special About a Vector Anyway?

INTRODUCTION

Why is artificial intelligence (AI), and neural networks specifically, so popular these days? The math and the science behind these algorithms were developed decades ago, but it's only in recent years that neural network-powered AI has taken off. So, what happened that enabled neural networks to succeed, where they fell short in the past? As is often the case, there is a key event that catalyzes a phase transition from "niche" to "transformational" for any piece of technology. The re-imagining of Graphics Processing Units (GPUs) from graphics devices to general computational machines was the nucleus around which the current AI revolution crystalized. GPUs enabled the acceleration of a specific set of mathematical operations: vector and matrix transformations. They gave us the ability to process information in a practical amount of time, but GPUs alone wouldn't have brought us here. The AI revolution happened because along with GPUs providing the horsepower, there was an infrastructure in place—the internet and the open-source developer community— which facilitated the collaboration of thousands of researchers and developers and created the collection of tools and libraries which are ubiquitous in AI development today. The GPU-powered breakthrough for accelerating neural networks would have remained niche if not for the army of researchers and developers who created the libraries that abstract away the nuances and difficulties of directly programming a GPU. This worked well for solving challenges being faced across a wide range of use cases. However, these libraries don't follow safety critical development guidelines which is increasingly becoming a fundamental requirement in many applications.

When we look at autonomy, and the embedded industry which relies on safety, determinism, and reliability, we find ourselves back at the start of the phase transition. We have GPUs, and with The Khronos Group's Vulkan® SC we have an API that supports both safety critical graphics and compute for programming GPUs . Now we need a complementary collection of libraries sitting on top of the Vulkan SC safety layer that follow safety-critical guidelines and standards to take us from niche to mainstream. CoreAVI has jumpstarted the revolution with the introduction of ComputeCore™. ComputeCore is CoreAVI's implementation of the linear algebra (BLAS) APIs. Linear algebra, and vectors in particular, are the fuel that feeds all AI algorithms. Pick your favorite AI engine today. That engine runs fast because it has a BLAS library accelerating all vector operations. ComputeCore does exactly this, but it is implemented to meet functional safety certification standards in avionics, automotive and industrial markets (DO-178C, Level A, ISO 26262 ASIL D and IEC61508 SIL 3). Vectors and matrices are the heart of AI and data processing. This is mentioned often, but it is almost never explained. Let's take a look.

WHAT IS DATA PROCESSING?

When we think of data processing, that is, data processing in the broad sense of taking some information and transforming it in some way, there is one data structure that is central to any and all processing, and that is the vector. What is so special about a vector anyway? When we speak of data processing, we mean the ability to take information and modify it. In some cases, we want to transform the information so that we can extract knowledge from it. This is the case for machine learning. We might take an image and transform it in a way that can tell us what objects are present in the scene. We do this for computer vision, enabling self-driving cars. We can also transform an image by blurring it, or sharpening it, or extracting edge information about the structures in the scene. These operations are common steps in a robust vision pipeline.

Data may also be non-visual. Consider a system for weather forecasting where we want the system to predict the maximum and minimum temperature for a given day of the year. In this case the input into the system might be a collection of historical data samples describing temperature ranges for each day of the year, for a given location, for the past decade. In any case, a data sample is a unit of information. When analyzing images, a data sample is a single image. When analyzing the clinical history of a patient in an attempt to predict the risks of heart disease for that patient, the data sample is a single patient. A data sample is composed of a list of *features* that describe the sample in some way. In the case of an image the features are the pixels that make up the image. In the case of a medical patient, the features are the attributes that describe that patient: age, sex, smoking habits, history of heart disease in the family, and other information related to their clinical history. When we look at each data sample as a collection of features, this view aligns very well with the concept of vectors.

CONVERTING DATA SAMPLES INTO VECTORS

How do we convert images and patients into vectors? For images, we interpret the colour of each pixel as a numerical value and construct a list of pixel values. For example, an image that's 2 pixels in width by 2 pixels in height would result in a vector that has 4 values and might look like this: [10,0,245,50], where each value represents the intensity of the colour of that pixel. Images of higher resolution result in vectors of higher dimension, for example a 256x256 pixel image results in a vector of 65,536 values. Patient information can also be vectorized by encoding the meaning of each feature using numbers, for example 1 = male, 0 = female, 1 = history of heart disease in the family, 0 = no heart disease, etc. Converting data samples into vectors is useful in two very important ways: it helps to encapsulate a list of features in a way that's easy to relate to each individual sample, and secondly, interpreting vectors geometrically reveals information about how samples in a dataset relate to each other. For example, suppose that we want to create an algorithm that can learn to differentiate between images of cars and pedestrians. We start by interpreting our images as vectors. Geometrically, a vector can be thought of as an arrow in space starting from the origin of our coordinate system and traveling through space to reach the point defined by the value of the vector in each dimension, see Figure 1.

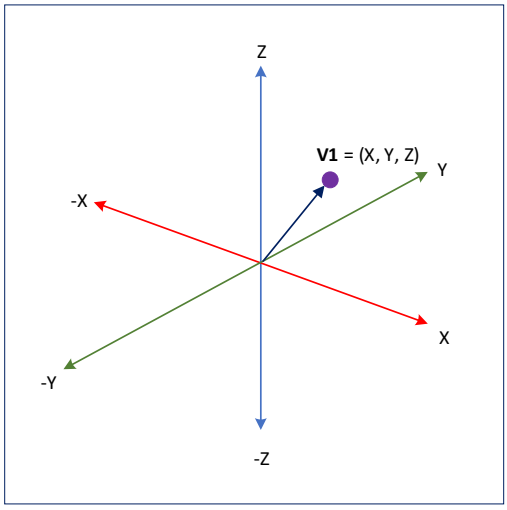

Figure 1: This figure illustrates 3D vector V1 in the cartesian coordinate system. The vector is the line that starts from the origin and ends at the point defined by its 3 components: x, y, z.

While we cannot visually represent vectors in dimensions higher than 3D, mathematically we can still manipulate them using the same equations. And what makes vectors so special is that when we interpret them geometrically, vectors which point in a similar direction have similar properties. For example, when we interpret images of cars and pedestrians as vectors in some multi-dimensional space (called a hyperspace), although we can't visualize the direction that that vector is pointing to, we can calculate it, and we can compare vectors belonging to images of cars and vectors belonging to images of pedestrians. It turns out that vectors of images of cars point roughly in the same direction, and vectors of images of pedestrians point in a similar direction to other images of pedestrians, but away from the car vectors. To create an algorithm that can learn to automatically differentiate between cars and pedestrians means manipulating those vectors mathematically to understand the direction they are pointing in and finding the line—or hyperplane—which bisects the space where cars and pedestrian vectors live. Once we find that hyperplane the algorithm can classify a brand-new image simply by knowing on which side of the line its vector falls, the side with all the car vectors, or the side with all the pedestrian vectors. This is why artificial intelligence algorithms all end up performing copious amounts of vector operations. It is because the input data is typically represented as vectors, because it is geometrically advantageous to do so. The other data structure that's used quite often in AI is the matrix. The reason for this should not come as a surprise. A vector is a collection of values. A matrix is simply a collection of vectors stacked one on top of the other. In some cases, it is useful to combine vectors as a matrix, so that we can process a set of vectors—for example a set of input samples—simultaneously.

COMPUTATION TOOLS – THE WHAT, WHY AND HOW

At this point we have established the need for vectors and matrices in artificial intelligence. Now we need a method to perform these calculations quickly. When we think of automation - for example, cars driving unassisted on a busy highway, or robots working alongside humans in a warehouse or factory - we think of systems executing in real-time, with real-time consequences. When a self-driven car fails to detect a pedestrian with enough time to avoid a collision, the AI algorithm itself may not be the source of the fault. Indeed, the algorithm may have correctly detected a pedestrian, but the detection may have just taken a few milliseconds too long. This example helps illustrate a couple of things. First, these systems need to perform vector and matrix math as fast as possible. Second, in safety critical applications these operations need to occur in a deterministic amount of time. That is, we need to be able to calculate the worst-case execution time for the executing algorithm, so that we can determine for every foreseeable circumstance if the system will have enough time to react to the results of executing that algorithm. It would be of little use to detect an obstacle if the window of opportunity to avoid it has already passed.

To solve these problems, application and solution developers need two things. First, they need a tool that can accelerate the computation of vector and matrix operations. For non-safety critical use cases there are a plethora of tools and frameworks freely available from the open-source community. These include frameworks like Tensorflow, Pytorch, Caffe, and Scikit-learn. All of these frameworks and libraries rely on a key component known as a BLAS library, mentioned above, which provides a set of functions to accelerate vector and matrix operations. Unfortunately, these frameworks are not implemented following functional safety standards. To that end CoreAVI has introduced ComputeCore, which is accelerated through CoreAVI's VkCore® SC graphics and compute driver, which is an implementation of the Khronos safety critical Vulkan® SC API. Aside from BLAS functionality to power the vector and matrix operations, ComputeCore also provides a full Fast Fourier Transform (FFT) API implementation to help accelerate signal processing use cases.

ACCELERATION

We have so far discussed the need for processing vectors and matrices, and for doing so in a deterministic manner. We have also said that to perform these operations quickly, we need a tool—like ComputeCore—which performs the acceleration. What do we mean by acceleration? Typically, when we think of software executing, we think of the CPU as being the end device where programming instructions are executed. Most of the software in a system runs on the CPU. Indeed, we could implement BLAS using the C programming language and perform the calculations on the CPU. The upside of this approach is that most programmers in the embedded space—real time devices, cars, robots—are familiar with the C programming language and there are standard practices for developing safety critical software in C. The downside of this approach is that the CPU is great at executing sequential operations quickly, but it's not great at parallelizing work. Unfortunately, the tasks involved in manipulating vectors and matrices for most AI algorithms are known as "embarrassingly parallel problems". Therefore, it is in our best interest to execute these calculations on machines that can perform these calculations in parallel, as opposed to sequentially. ComputeCore solves this problem by trying to meet the best of both worlds. The ComputeCore API—for BLAS and FFTs—is implemented using the C programming language which we all know and love. But the executing engine which performs the actual computations, leverages the Vulkan SC compute pipeline to perform the vector and matrix operations using an accelerator hardware, such as a GPU or a dedicated FPGA, which is capable of parallelizing the work in order to increase the volume of calculations performed per second. By accelerating the mathematical operations using an accelerator device—GPU or other Vulkan SC supported device such as a neural processing unit (NPU)—and by sitting on top of a safety certifiable open-standard API such as Vulkan SC, ComputeCore is the first truly certifiable engine for AI and machine learning operations in the safety critical industry.

CONCLUSION

The past decade has been great for artificial intelligence. The research community, and the open-source engineering community have produced a plethora of tools that power many of today's non-safety critical products. Products like Amazon's Alexa, or Google's Home Assistant are powered by state-of-the-art Natural Language Processing (NLP) algorithms. The breakthrough research into these algorithms was made possible by an ecosystem of tools which enabled engineers and scientists to develop and train complex neural networks, with relative ease, and accelerate the operations on GPU devices. These tools include neural network frameworks like TensorFlow and PyTorch, statistics and data-analytic tools like the Python scientific library scikit-learn, and high-performance compute frameworks such as CUDA® and SYCL™. Image processing libraries like OpenCV make an engineer's life easier by providing built-in functionality for commonly used algorithms and techniques. In fact, these available frameworks are so successful and ubiquitous that as we move into the safety critical domain many people have not yet realized that those tools are not available where determinism and safety matters. This is why CoreAVI has been working with Khronos to develop the Vulkan SC specification, to serve as the foundation of safety critical compute.

But just as the research community uses an eco-system of tools beyond just CUDA or OpenCL™, the safety critical industry needs to build an ecosystem of safety critical tools and frameworks. ComputeCore, as CoreAVI's safety-critical implementation of BLAS and FFT, is a step in the right direction, but we need our partners and the entire embedded and autonomous community to help define the tools and safety critical APIs that will power all reliable devices of the future. The open-source community has shown what can be achieved and has provided a great platform for prototyping our products and ideas, but we know that Python libraries don't execute deterministically. Autonomy implies responsibility. Responsible systems must be safe and reliable. The sooner the industry realizes this, the sooner we can leave the sandbox and build an infrastructure based on the rigor of safety standards and guidelines that facilitates the mass deployment of safe autonomy.

AUTHOR

Ken Wenger
Principal Software Engineer

Ken Wenger is a Principal Software Engineer at CoreAVI. He has developed a number of safety critical products including graphics drivers and compute libraries. His current area of focus is research in the application of safety critical principles and guidelines in autonomous systems, using neural networks and other advanced machine learning algorithms. He is currently leading a research project into the use of semi-supervised learning algorithms for medical image diagnosis.

This white paper was first published Dec., 2021 in Embedded Computing Design.