Electronic Design

By WILLIAM WONG, Editor,
Electronic Design

Understanding the Art of Machine Learning

Though neural-network-based machine learning is escalating in popularity, the mechanics behind it tend to be misconstrued or simply not known at all.

Machine learning (ML) is a hot topic when it comes to almost anything related to computing, from analyzing data in the cloud, to self-driving cars recognizing people and things, to detecting defective PCBs or chips. Like artificial intelligence (AI), ML is a very broad subset of AI that's often mischaracterized even by people technical backgrounds.

Deep learning is a term that's bandied about these days, but what does it really mean? Typically, it's shorthand for a discussion about deep neural networks (DNNs). We will get into more detail about neural networks but first a comment about current ML use.

We recently finished up our local Mercer Science and Engineering Fair, which I help manage. As you might guess, a lot of projects employed ML models for various tasks. This is a challenge from numerous points of view.

All of the students were using a predefined ML model to perform a specific task, like identifying a problem. Some actually trained a model, but the general knowledge about ML, their model, and its implications were something neither the students or judges really had a handle on. I wound up giving a number of judges an overview on ML and how they should view it with respect to the projects.

Essentially, the student's use of ML was as a tool and not some cutting-edge AI advance. In general, the understanding of the tool, how it worked, and how to use it was significantly lacking. However, that should not be unexpected given how much students are learning when working on a project that's very new to them. It's great to discover how they can get things done. Still, developers and engineers who plan on building commercial solutions need much more insight because of the responsibility they have for the resulting products and their use.

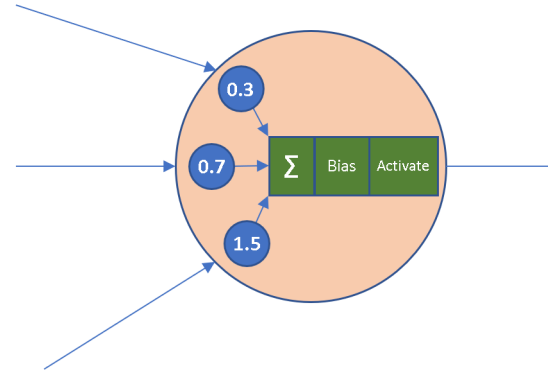
What I hope to accomplish here is to reveal the types and capabilities of some of the more important machine-learning technologies so you can figure out what to investigate and what level of interaction is needed to take advantage of DNN ML technology. It can range from using preconfigured models to creating new models for an application.

The amount of effort between those two is significant, from an afternoon's worth of work to decades of man hours. Many will simply use a product like a smart camera that employs ML without ever dealing with the technology directly. Nonetheless, it helps to know how the technology works and what the limitations are because we don't have positronic brains around the corner or the robots to match—yet.

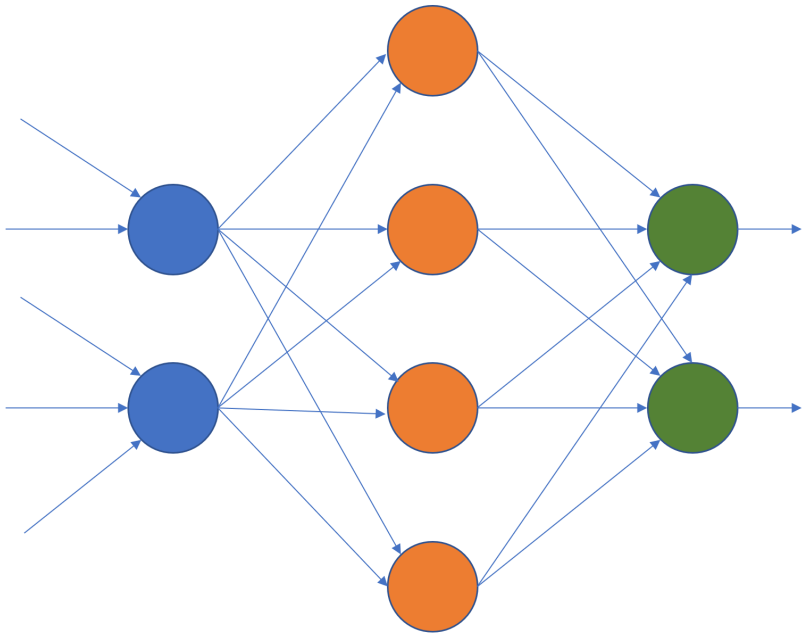
Machine-Learning Basics

Two of the main ML technologies include rule-based expert systems that are sort of state machines on steroids, and neural networks.

Expert systems are still in use and a viable solution for many problems. They're normally created using explicit rules to build behavior-based systems that respond to input. Self-trained systems like this haven't worked well, but expert systems can be very fast, accurate, and efficient when properly programmed. Unfortunately, translation of expertise into an expert system is often time-consuming.



1. A perceptron multiplies weight values with inputs that are then added together and combined with bias and activation functions to generate an output.



2. Perceptrons are combined in multiple layers to form machine-learning models. The middle layers are often referred to as hidden layers, and there may be multiple hidden layers.

On the other hand, neural networks (NNs) are modeled after the brain, although at a logical level. The approach has been around for decades. However, new hardware made this approach more practical, allowing for the implementation of complex AI/ML models.

A perceptron is the model of a biological neuron *(Fig. 1)*. In general terms, input values are combined with weights, summed with a bias and activation function included in the mix to generate an output. This is kind of like discussing logic circuits, op amps, and transistors. We're doing a bit of hand waving here, but it's enough to provide the basics for our discussion.

The next step is to combine perceptrons *(Fig. 2)*. Typically, multiple layers are involved in a model that needs to be designed based on the number of inputs, outputs, and functionality of the system. A model is the combined architecture along with the weights used within the system. These weight values are usually obtained by training a model. The number of layers can be large, hence the term deep neural networks, or DNNs, that really describes the general magnitude of the models rather than a specific approach, which we will get to later.

The outputs are probabilities. A system will often have thresholds for acceptable identifications depending on the purpose of the model. The level of detail in the analysis of an image, for example, can range from an object to an animal to a cat to a Persian cat.

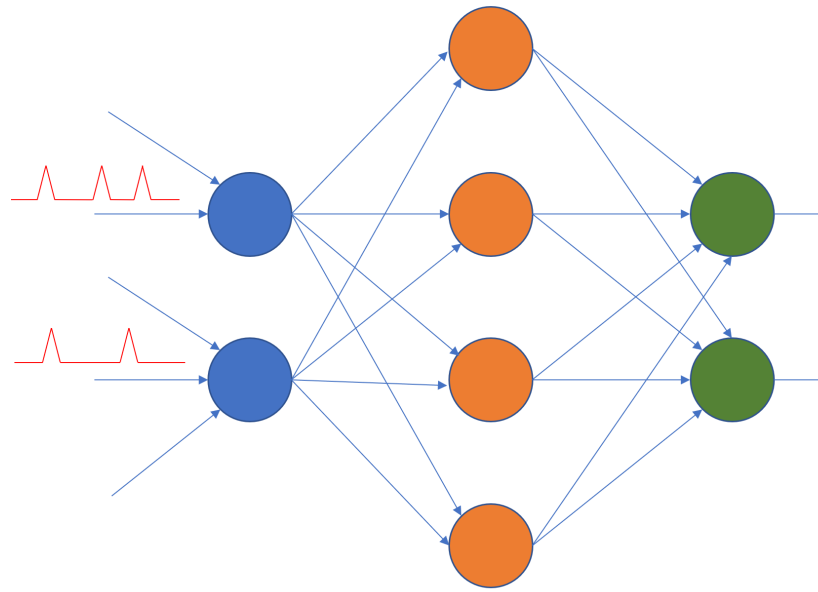
Dealing with images can greatly increase the complexity of a model. Even a small, 320- × 240-pixel image translates to 76,800 inputs. If color instead of grayscale is used as inputs, then a red-green-blue (RGB) encoding bumps this by a factor of three.

On the other hand, working with fewer inputs enables simpler models to be implemented, often in software on a microcontroller. For example, a motor-control application may have half-a-dozen inputs to support a preventive-maintenance model.

The Many Neural Networks

There are many different types of NNs, including artificial neural networks (ANNs) and spiking neural networks (SNNs). ANNs use a parallel approach to presenting the inputs to the network with outputs available after all of the data flows through the network.

SNNs are more akin to biological neurons—they are



3. SNNs are similar to biological neurons because input data is a series of time-related blips of data.

asynchronous working off time-related blips of data *(Fig. 3)*. An SNN neuron emits a spike when a membrane threshold is reached. An SNN can be implemented in many ways, but the most popular is a leaky integrate-and-fire approach that mimics its biological cousins.

SNNs tend to have less overhead than other neural networks, but their accuracy is usually lower. Keep in mind that all of these networks are dealing with thresholds and probabilities and 100% accuracy is only a goal. Also, keep in mind that SNNs can often be trained in the field, while training for other neural nets is typically done in the cloud where additional data is accumulated. SNNs are able to do this in part because of their lower implementation overhead and due to the technology itself.

A number of other neural-network types are out there, and many models can fit under these broad categories. They can perform functions such as identification and classification. Let's take a closer look at three major categories:

- Convolutional networks (CNNs)
- Recurrent neural networks (RNNs)
- Generative adversarial networks (GANs)

A CNN is a DNN that's typically used for image analysis. It can be applied in facial-recognition systems, parsing documents, and image segmentation. CNNs are also a type of space invariant artificial neural networks (SIANNs).

A CNN model essentially implements filters via the training process. The bias and activation operations include a Frobenius inner dot product. The model includes convolutional and pooling layers. The number of inputs and outputs of a convolutional layer are typically the same. The pooling layers reduce the number of outputs. This is often accomplished by getting an output, which involves taking the maximum input value or average of a group of inputs.

An RNN's perceptrons take their output as the input to the next set of data *(Fig. 4)*. A dataflow implementation would typically include latched inputs so that the output could be used in the subsequent calculation. Oftentimes, RNNs are used to analyze audio streams, text, and time series information.

The GAN is a neural network designed to generate an output that's similar to the input. For example, a GAN model could change a cat into a dog that would have the same pose, etc. This type of model is able to unpaint pictures or upscale video.

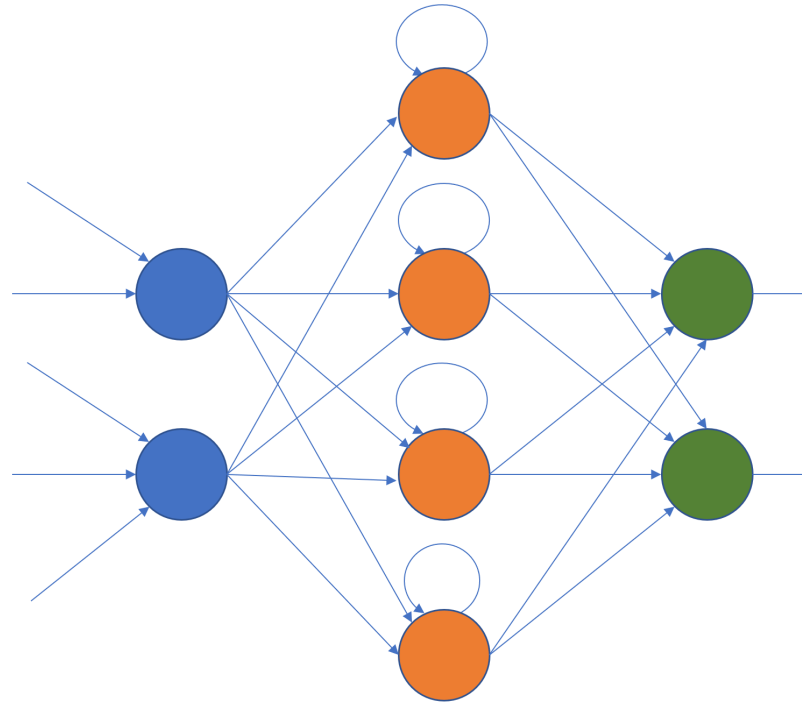
A GAN consists of a generator network and a discriminative network. The former creates candidates based on the inputs and the latter evaluates the new data. Each must be trained with the discriminative network being trained with inputs, much like how a CNN would be trained. The generator is typically a deconvolutional neural network that's trained to increase the error rate of the discriminative network.

Multicore vs. Data-Flow Acceleration

Neural-network models are like applications. They're specifications that can be implemented in a number of ways, from a software application to a hardware implementation. Hardware implementations are typically faster and use less power, thus providing more performance such as the ability to analyze video streams in real-time.

A very, very large number of calculations is the reason that hardware acceleration is often required in AI applications, but this is related to the input size and model complexity. Video applications typically benefit from hardware support.

Hardware-acceleration approaches can be divided into data-flow or application-specific support, targeted multicore, and instruction set augmentation. Most ML/AI chips go with data-flow or targeted-multicore implementations. The latter is often a DSP or single-instruction, multiple-data (SIMD) architecture that's customized for AI/ML models. An example of instruction set augmentation is Arm's Cortex-M55, which adds instructions that improve support



4. Recurrent neural networks (RNNs) provide each perceptron with a feedback loop.

for AI/ML-related numeric encodings and computations such as matrix manipulation.

AI/ML models can be implemented using double-precision floating point, but that tends to be very inefficient for various reasons. Using more compact formats like 8- and 16-bit floating point, and 8-bit fixed point, significantly reduces storage requirements, calculation hardware, and calculation time, thereby lowering power requirements while improving performance.

Another aspect deals with sparse networks. It turns out that in many cases, weights are often zero or very close to zero. This enables those calculations and the data movement to be eliminated by optimization. Much of it is hidden behind model compilers that accept models from standard frameworks like TensorFlow and Caffe2.

Another consideration with respect to hardware involves multiple models and splitting models across different types of hardware. Many times, multiple models are used to provide more functionality based on the same input or to address different aspects of the overall application.

For example, an electric car may have an advanced driver-assistance system (ADAS) or fully autonomous-driving system that uses object recognition and other models to handle preventative maintenance for the electric motors. Targeted or sophisticated system-on-chip (SoC) solutions are often designed to handle this mixture.

The splitting of models makes sense because the hardware support for different parts of the model can vary. Multicore microcontrollers like Eta Compute's ECM3532 includes a Cortex-M4 and NXP CoolFlux DSP. The latter is augmented to handle AI/ML models, but the Cortex-M4 can help.

Making Decisions for Applications

Choosing models and hardware platforms to support them can be complex. They can probably use an AI model if someone would make it. On the plus side, this type of support can be approached in different ways if your application would benefit from AI/ML support.

The first way is easy: Use a product with AI support that requires minimal or no training. Many robotic sensors targeting the educational market fall into this category. They are essentially smart sensors that provide feedback to a host, such as "red obstacle 15 cm ahead."

The majority of uses fall into the next category: Utilizing predefined and often pretrained models on hardware of your choice. This may require choosing a model, and usually the available models will be based on the hardware choice or vice versa. This is the advantage of using platforms like TensorFlow or Caffe, since hardware vendors have chosen to support these. Likewise, performance numbers are often available, so you can gauge what platform may be suitable for your application.

If a model is available and you have the data to train it, then the problem often reverts to a standard engineering challenge of balancing and optimizing cost, performance, power, and capacity. You may have much more work to do if the models that exist don't quite match your needs.

Creating a new model is only for those with big bucks, lots of time, or lots of expertise. This is where the experts come in—even companies and organizations are using AI to create models. You can also wait until someone develops a model that you can use, although working with an unsupported, open-source model may not be the best choice for a commercial product. Like many open-source software projects, you may have to support yourself.

The challenge these days is that the technology and availability of AI/ML solutions is changing so quickly. Compiler improvements can sometimes double performance, making a lower-end platform capable of handling your application when it couldn't do so with the earlier version. Likewise, many new chips with AI enhancements or dedicated chips are becoming available, and new processor designs are regularly including instruction set extensions and data formats amenable to AI/ML models.