These articles will look at a topic, explain some of the background, and answer a few questions that we’ve heard from the MATLAB community.

This article’s topic is GPUs for deep learning. I’ll summarize the topic and then take a look at three questions:

1. When people say it “speeds up training,” how big a speedup do they mean?
2. Do I need to buy a (really) fast GPU to be able to train a neural network in MATLAB?
3. What are my options for deep learning without a GPU?

The incorporation of GPUs—primarily NVIDIA® GPUs—was some of the fuel that powered the big deep learning craze of the 2010s. When working with large amounts of data (thousands or millions of data samples) and complex network architectures, GPUs can significantly speed up the processing time to train a model. Prior to that, many of today’s deep learning solutions would not be possible.

Yes, GPUs are great, but what are they exactly?

GPUs or graphics processing units, were originally intended for graphics (as the name implies). GPUs can perform many computations in parallel, making them very good at handling large simple tasks like pixel manipulation.

The primary use case for GPUs is image classification, but signal data can also benefit from this rapid calculation. In many cases, “images” are created from signals using data preprocessing techniques that convert the signal into a 3D
time-frequency representation of this signal (read more about deep learning for signal processing with MATLAB). These images are then used for deep learning training, where features are learned directly from the time frequency map (image) rather than the raw signal. For even more speed, we also can use GPUCoder™ to create CUDA code that runs directly on NVIDIA GPU.

Unlike CPUs, which often have four or eight powerful cores, GPUs can have hundreds of smaller cores that work in parallel. Each GPU core can perform simple calculations, but by itself it isn't very smart. Its power comes from brute force; putting all those cores to work on deep learning calculations like convolution, ReLU, and Pooling.

If you want to learn more, see what MATLAB support for GPU computing looks like, but for now let’s get to the questions!

Q1
I see a lot of hype around using a GPU to speed up deep learning training, but very few details. I don’t want to waste my time arguing for budget for a GPU if I can’t promise a real speed increase. So, how much of an increase can I reasonably expect?

Here’s the thing—it’s really going to depend. There are some factors that influence how significant an increase you will see:

- Large input data size: the more complicated the dataset, the more a GPU can speed up training
- Complex network structure: the more convolutions and calculation you do, the longer this will take
- Hardware: what you started with and what you are moving to

The situation would be rare in which a GPU does not speed up the training, but there are cases where a GPU might be overkill, such as 1D input data, vector data, or small input data. Take this simple deep learning classification example, in which the images are small (28 x 28 px) and the network only has a few layers. This dataset takes only a few minutes to train with a CPU, so a GPU wouldn’t make much difference at all.

Fun fact: If you have a GPU, you can use the MATLAB function `gputimeit` to measure the average time functions take to run on a GPU. Also, this blog post is from 2017 but it’s still a great resource for measuring the speed of your GPU and comparing CPUs and GPUs for deep learning.

This should all intuitively make sense: if I have a smaller input size, and ask the network to perform less calculations (using less layers), there isn’t as much opportunity for parallelization and speedup the GPU offers.
The best advice I can give you is to see if you can borrow a GPU or sign up for some cloud-based GPU resources and measure the difference in training time. Actual measurements might be more persuasive in arguments than "expected" or "predicted" benefits anyway!

Lastly, each new GPU model is faster than the last, just like CPUs aren’t all staying the same over the years. Check out NVIDIA performance data.

Q2
I’m a MATLAB user and want to train a neural network. Do I need to buy a fast GPU?

There are two words I want to pick out in this question: "need" and "fast." Need implies necessity, and that is a question only you can answer. Do you have a mandate from management to have a neural network ready to go in production on a tight deadline? Then, sure! You need one. Will whatever you’re training work without a fast GPU? Eventually! So, it’s really up to you.

Now, do you need a "fast" GPU? As with "need," this goes back to what your actual requirements are—but we’re past the technicalities so let’s assume you have some sort of time pressure and take this question as, "How do I know which GPU I need?"

Like computer hardware in general, GPUs do age over time, so you want to keep track of what the current research is using when training models. Similar to the last question, results may vary based on your answers to these questions:

- How much data do you have?
- How many training classes are there?
- What is the structure of the network?

Even your laptop has a GPU, but that doesn’t mean it can handle the computations needed for deep learning.

A while back, I hit my own patience threshold. I had a deep learning model I was trying to run, and it was taking forever. I saw a developer friend of mine and thought I’d pick his brain about what the problem might be. We went through the complexity of the network (ResNet-Inception based), the number of images (a few hundred thousand), and the number of classes (about 2000). We couldn’t understand why training would take longer than a few hours.

Then we got to hardware. I mentioned I was using a Tesla K40 circa 2014 and he literally started laughing. It was awkward. And slightly rude. But once he got tired of hardware shaming me, he offered me the use of his. Speed improvements ensued and there was peace throughout the land. The moral of this story is that hardware advances move quickly and a friend who shares their Titan X is a friend indeed.

Here’s a more documented example: my colleague Heather Gorr (@HeatherGorr) ran this video classification example from documentation—the same data and network on two different hardware setups resulted in some significant differences in processing time. Read more about her experience.
NVIDIA Quadro M2200

Original model (50 classes): 12.6 hrs, Acc: 66.7%
Small model (8 classes): 90 mins, Acc: 83.16%

Linux Desktop with GPU
NVIDIA TITAN XP

Original model (50 classes): 2.7 hrs, Acc: 67.8%
Small model (8 classes): 26 min 29 sec, Acc 80%

Just to note: Both tests had training plots enabled for monitoring and screenshot purposes. The number of classes is not the culprit here; it’s that using fewer classes uses fewer input samples. The part you can affect that’s going to have a tangible impact on training time is the amount of data in each class.

I’ve compiled a list of GPUs from very expensive to very not expensive and a few standard specs:

<table>
<thead>
<tr>
<th></th>
<th>Quadro GV100</th>
<th>Titan RTX</th>
<th>GeForce RTX 2080</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA Parallel-Processing Cores</td>
<td>5120</td>
<td>4608</td>
<td>2944</td>
</tr>
<tr>
<td>GPU Memory</td>
<td>32 GB HBM2</td>
<td>24 GB G5X</td>
<td>8GB GDDR6</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>870 GB/s</td>
<td>672 GB/s</td>
<td>448 GB/s</td>
</tr>
<tr>
<td>Price</td>
<td>$8,999</td>
<td>$2,499</td>
<td>$799</td>
</tr>
</tbody>
</table>

Note: These prices are correct as of 4/2/2020 and are subject to change.

Prices go down as hardware ages, so although we laughed earlier at my Tesla K40 story—it’s $500. If you don’t have the money, don’t be fooled into the latest and greatest. Every year, GPU manufacturers will continue to pump out the fastest GPUs we’ve ever seen, which will make the older models less desirable and expensive. In fact, take a look at the RTX 2080. Not a bad little GPU for under $1K.

I don’t have access to a GPU. What can I do?

Well, the good news is you still have options.

First Up: Cloud Resources

For example, with NVIDIA GPU Cloud (NGC) and cloud instances, you can pull 4, 8, or more GPUs to use in the cloud and run multiple iterations in parallel; you can also distribute the training across multiple GPUs. This should help
speed things up, and the use of cloud resources ensures that your GPUs won’t be as dated as something you bought and ages over time. Cloud ≠ free, so while it should be a smaller up-front cost, there is still a fee.

Next: Optimize for CPUs

You can run multicore CPU training. You will still benefit from a low-performing GPU over multiple CPU cores, but they are better than nothing.

In addition to this, you can switch your algorithm. Instead of training, you could perform "activations" from the network. Gabriel Ha talks about this in his video on using feature extraction with neural networks in MATLAB. You can also follow an example showing the use of activations.

Transfer learning tends to take less time than training from scratch. You can take advantage of features learned in prior training and focus on some of the later features in the network to understand the unique features of the new dataset.

Last: Borrow a GPU then Test with a CPU

Say you’ve managed to train your network; CPUs work very well for inference! The speed differences become much more manageable compared with GPUs, and we’ve improved the inference performance of these networks on CPUs.