Published in Towards Data Science

Tivadar Danka Follow
Jun 23, 2020 · 7 min read · · Listen

Photo by Nana Dua on Unsplash

How GPUs accelerate deep learning

The embarrassingly parallel nature of neural networks

Neural networks and deep learning are not recent methods. In fact, they are convolutional networks were introduced in 1989 in the landmark paper Backpropagation Applied to Handwritten Zip Code Recognition by Yann LeCun et al.

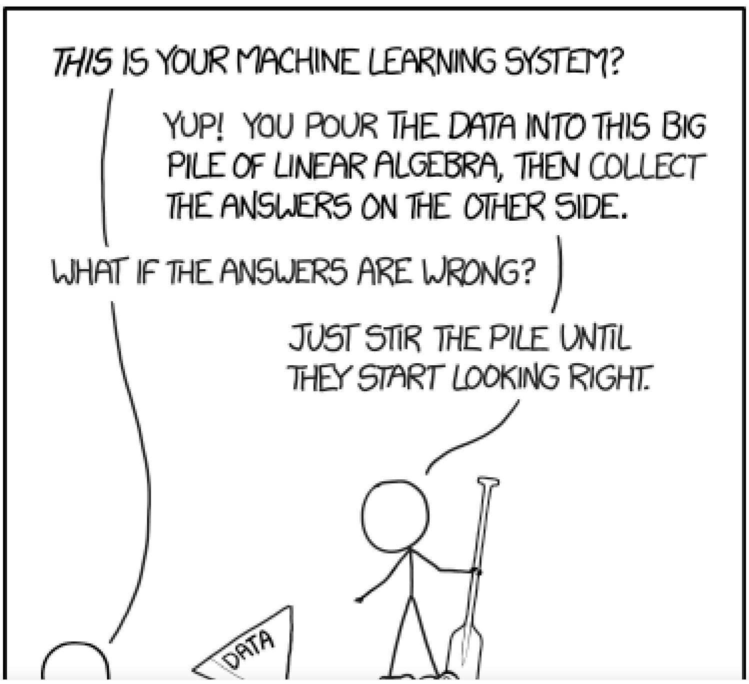
Why did the deep learning revolution had to wait decades?

One major reason was the computational cost. Even the smallest architectures can have dozens of layers and millions of parameters, so repeatedly calculating gradients during is computationally expensive. On large enough datasets, training used to take days or even weeks. Nowadays, you can train a state of the art model in your notebook under a few hours.

There were three major advances which brought deep learning from a research tool to a method present in almost all areas of our life. These are *backpropagation*, *stochastic gradient descent* and *GPU computing*. In this post, we are going to dive into the latter and see that neural networks are actually *embarrassingly parallel* algorithms, which can be leveraged to improve computational costs by orders of magnitude.

A big pile of linear algebra

Deep neural networks may seem complicated for the first glance. However, if we zoom into them, we can see that its components are pretty simple in most cases. As the always brilliant xkcd puts it, a network is (mostly) a pile of linear algebra.



Source: xkcd

During training, the most commonly used functions are the basic linear algebra operations such as matrix multiplication and addition. The situation is simple: if you call a function a bazillion times, shaving off just the tiniest amount of the time from the function call can compound to a serious amount.

Using GPU-s not only provide a small improvement here, they supercharge the entire process. To see how it is done, let's consider activations for instance.

Suppose that φ is an activation function such as ReLU or Sigmoid. Applied to the output of the previous layer

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n,$$

the result is

$$\varphi(\vec{x}) = \begin{pmatrix} \varphi(x_1) \\ \varphi(x_2) \\ \vdots \\ \varphi(x_n) \end{pmatrix}.$$

(The same goes for multidimensional input such as images.)

This requires to loop over the vector and calculate the value for each element. There are two ways to make this computation faster. First, we can calculate each $\varphi(x_i)$ faster. Second, we can calculate the values $\varphi(x_1), \varphi(x_2), \ldots, \varphi(x_n)$ simultaneously, in *parallel*. In fact, this is *embarrassingly parallel*, which means that the computation can be parallelized without any significant additional effort.

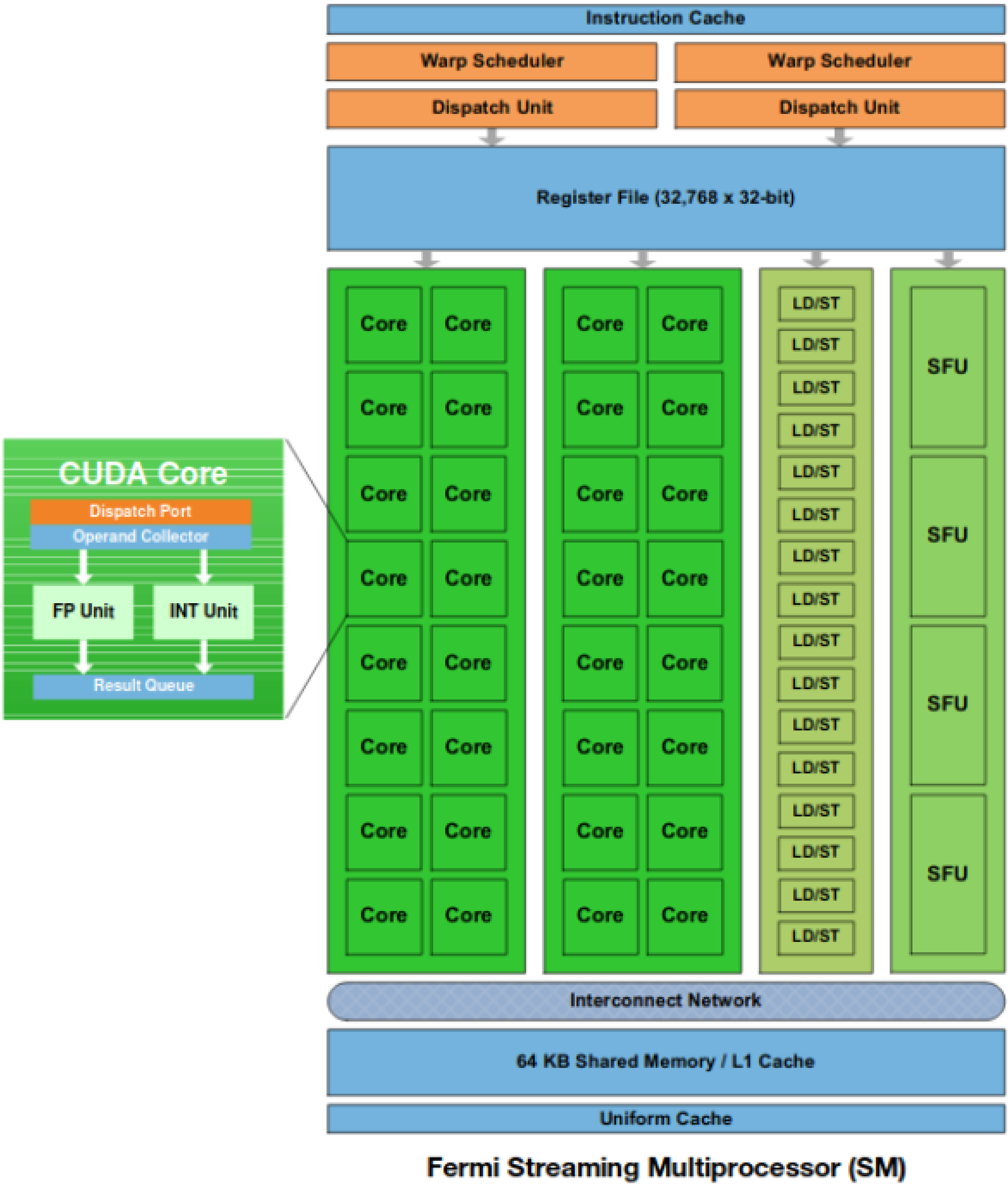
processor design has reached a point where packing more transistors into the units has quantum-mechanical barriers.

However, calculating the values in parallel does not require faster processors, just more of them. This is how GPUs work, as we are going to see.

The principles of GPU computing

Graphics Processing Units, or GPUs in short were developed to create and process images. Since the value of every pixel can be calculated independently of others, it is better to have a lot of weaker processors than a single very strong one doing the calculations sequentially.

This is the same situation we have for deep learning models. Most operations can be easily decomposed to parts which can be completed independently.



NVIDIA Fermi architecture. There has been many improvements to this, but it illustrates the point well. Source: NVIDIA Fermi architecture whitepaper

To give you an analogy, let's consider a restaurant, which has to produce French fries on a massive scale. To do this, workers must peel, slice and fry the potato. Hiring people to peel the potatoes costs much more than purchasing many more kitchen robots capable to perform this task. Even if the robots are slower, you can buy much more from the budget, so overall the process will be faster.

When talking about parallel programming, one can classify the computing architectures into four different classes. This was introduced by Michael J. Flynn in 1966 and it is in use ever since.

1. **S**ingle **I**nstruction, **S**ingle **D**ata (SISD)
2. **S**ingle **I**nstruction, **M**ultiple **D**ata (SIMD)
3. **M**ultiple **I**nstructions, **S**ingle **D**ata (MISD)
4. **M**ultiple **I**nstructions, **M**ultiple **D**ata (MIMD)

A multi-core processor is MIMD, while GPUs are SIMD. Deep learning is a problem for which SIMD is very well suited. When you calculate the activations, the same exact operation needs to be performed, with different data for each call.

Latency vs throughput

To give a more detailed picture on what GPU better than CPU, we need to take a look into *latency* and *throughput*. Latency is the time required to complete a single task, while throughput is the number of tasks completed per unit time.

Simply put, a GPU can provide much better throughput, at the cost of latency. For embarrassingly parallel tasks such as matrix computations, this can offer an order of magnitude improvement in performance. However, it is not well suited for complex tasks, such as running an operating system.

CPU, on the other hand, is optimized for latency, not throughput. They can do much more than floating point calculations.

General purpose GPU programming

In practice, general purpose GPU programming was not available for a long time. GPU-s were restricted to do graphics, and if you wanted to leverage their processing power, you needed to learn graphics programming languages such as OpenGL. This was not very practical and the barrier of entry was high.

This was the case until 2007, when NVIDIA launched the CUDA framework, an

GPU computing for deep learning

So, we have talked about how GPU computing can be used for deep learning, but we haven't seen the effects. The following table shows a benchmark, which was made in 2017. Although it was made three years ago, it still demonstrates the order of magnitude improvement in speed.

TABLE 7. COMPARATIVE EXPERIMENT RESULTS (TIME PER MINI-BATCH IN SECOND)

		Desktop CPU (Threads used)				Server CPU (Threads used)						Single GPU		
		1	2	4	8	1	2	4	8	16	32	G980	G1080	K80
FCN-S	Caffe	1.324	0.790	**0.578**	15.444	1.355	0.997	0.745	**0.573**	0.608	1.130	0.041	**0.030**	0.071
	CNTK	1.227	0.660	**0.435**	-	1.340	0.909	0.634	0.488	**0.441**	1.000	0.045	**0.033**	0.074
	TF	7.062	4.789	2.648	**1.938**	9.571	6.569	3.399	1.710	0.946	**0.630**	0.060	**0.048**	0.109
	MXNet	4.621	2.607	2.162	**1.831**	5.824	3.356	2.395	2.040	**1.945**	2.670	-	**0.106**	0.216
	Torch	1.329	0.710	**0.423**	-	1.279	1.131	0.595	0.433	**0.382**	1.034	0.040	**0.031**	0.070
AlexNet-S	Caffe	1.606	0.999	**0.719**	-	1.533	1.045	**0.797**	0.850	0.903	1.124	0.034	**0.021**	0.073
	CNTK	3.761	1.974	**1.276**	-	3.852	2.600	1.567	1.347	**1.168**	1.579	0.045	**0.032**	0.091
	TF	6.525	2.936	1.749	**1.535**	5.741	4.216	2.202	1.160	**0.701**	0.962	0.059	**0.042**	0.130
	MXNet	2.977	2.340	2.250	**2.163**	3.518	3.203	2.926	2.828	**2.827**	2.887	0.020	**0.014**	0.042
	Torch	4.645	2.429	**1.424**	-	4.336	2.468	1.543	1.248	**1.090**	1.214	0.033	**0.023**	0.070
RenNet-50	Caffe	11.554	7.671	**5.652**	-	10.643	8.600	6.723	**6.019**	6.654	8.220	-	**0.254**	0.766
	CNTK	-	-	-	-	-	-	-	-	-	-	0.240	**0.168**	0.638
	TF	23.905	16.435	10.206	**7.816**	29.960	21.846	11.512	6.294	**4.130**	4.351	0.327	**0.227**	0.702
	MXNet	48.000	46.154	44.444	**43.243**	57.831	57.143	54.545	54.545	**53.333**	55.172	0.207	**0.136**	0.449
	Torch	13.178	7.500	**4.736**	4.948	12.807	8.391	5.471	4.164	**3.683**	4.422	0.208	**0.144**	0.523
FCN-R	Caffe	2.476	1.499	**1.149**	-	2.282	1.748	1.403	1.211	1.127	**1.127**	0.025	**0.017**	0.055
	CNTK	1.845	0.970	0.661	**0.571**	1.592	0.857	0.501	0.323	**0.252**	0.280	0.025	**0.017**	0.053
	TF	2.647	1.913	1.157	**0.919**	3.410	2.541	1.297	0.661	0.361	**0.325**	0.033	**0.020**	0.063
	MXNet	1.914	1.072	0.719	**0.702**	1.609	1.065	0.731	0.534	0.451	**0.447**	0.029	**0.019**	0.060
	Torch	1.670	0.926	**0.565**	0.611	1.379	0.915	0.662	0.440	0.402	**0.366**	0.025	**0.016**	0.051
AlexNet-R	Caffe	3.558	2.587	**2.157**	2.963	4.270	3.514	3.381	**3.364**	4.139	4.930	0.041	**0.027**	0.137
	CNTK	9.956	7.263	**5.519**	6.015	9.381	6.078	4.984	**4.765**	6.256	6.199	0.045	**0.031**	0.108
	TF	4.535	3.225	1.911	**1.565**	6.124	4.229	2.200	1.396	1.036	**0.971**	**0.227**	0.317	0.385
	MXNet	13.401	12.305	12.278	**11.950**	17.994	17.128	16.764	**16.471**	17.471	17.770	0.060	**0.032**	0.122
	Torch	5.352	3.866	**3.162**	3.259	6.554	5.288	4.365	**3.940**	4.157	4.165	0.069	**0.043**	0.141
RenNet-56	Caffe	6.741	5.451	**4.989**	6.691	7.513	**6.119**	6.232	6.689	7.313	9.302	-	**0.116**	0.378
	CNTK	-	-	-	-	-	-	-	-	-	-	0.206	**0.138**	0.562
	TF	-	-	-	-	-	-	-	-	-	-	0.225	**0.152**	0.523
	MXNet	34.409	31.255	**30.069**	31.388	44.878	43.775	**42.299**	42.965	43.854	44.367	0.105	**0.074**	0.270
	Torch	5.758	3.222	**2.368**	2.475	8.691	4.965	3.040	**2.560**	2.575	2.811	0.150	**0.101**	0.301
LSTM	Caffe	-	-	-	-	-	-	-	-	-	-	-	-	-
	CNTK	0.186	0.120	**0.090**	0.118	0.211	0.139	0.117	0.114	**0.114**	0.198	0.018	**0.017**	0.043
	TF	4.662	3.385	1.935	**1.532**	6.449	4.351	2.238	1.183	0.702	**0.598**	0.133	**0.065**	0.140
	MXNet	-	-	-	-	-	-	-	-	-	-	0.089	**0.079**	0.149
	Torch	6.921	3.831	**2.682**	3.127	7.471	4.641	3.580	**3.260**	5.148	5.851	0.399	**0.324**	0.560

Note: The mini-batch sizes for FCN-S, AlexNet-S, ResNet-50, FCN-R, AlexNet-R, ResNet-56 and LSTM are 64, 16, 16, 1024, 1024, 128 and 128 respectively.

CPU vs GPU benchmarks for various deep learning frameworks. (The benchmark is from 2017, so it considers the state of the art back from that time. However, the point still stands: GPU outperforms CPU for deep learning.) Source: Benchmarking State-of-the-Art Deep Learning Software Tools

How modern deep learning frameworks use GPUs

Programming directly in CUDA and writing kernels by yourself is not the easiest thing to do. Thankfully, modern deep learning frameworks such as TensorFlow and PyTorch doesn't require you to do that. Behind the scenes, the computationally intensive parts are written in CUDA using its deep learning library cuDNN by NVIDIA. These are called from Python, so you don't need to use them directly at all. Python is really strong in this aspect: it can be combined with C easily, which gives you both the power and the

Is NumPy really faster than Python?
Yes, but only if you know how to use it.
towardsdatascience.com

Do you need to build a deep learning rig?

If you want to train deep learning models on your own, you have several choices. First, you can build a GPU machine for yourself, however, this can be a significant investment. Thankfully, you don't need to do that: cloud providers such as Amazon and Google offer remote GPU instances to work on. If you want to access resources for free, check out Google Colab, which offers free access to GPU instances.

Conclusion

Deep learning is computationally very intensive. For decades, training neural networks was limited by hardware. Even relatively smaller models had to be trained for days, and training large architectures on huge datasets was impossible.

However, with the appearance of general computing GPU programming, deep learning exploded. GPUs excel in parallel programming, and since these algorithms can be parallelized very efficiently, it can accelerate training and inference by several orders of magnitude.

This has opened the way for rapid growth. Now, even relatively cheap commercially available computers can train state of the art models. Combined with the amazing open source tools such as TensorFlow and PyTorch, people are building awesome things every day. This is truly a great time to be in the field.

If you love taking machine learning concepts apart and understanding what makes them tick, we have a lot in common. Check out my blog, where I frequently publish technical posts like this!

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.