The Big Trend in Informatics

and how it came to be

Hasan Altan Birler
IEL '16
altan.birler@tum.de

informatics /ˌɪnfəˈmatɪks/ *n*: The science of processing data for storage and retrieval; information science.[1]

Oxford Dictionaries

ABSTRACT

An analysis of what machine learning is and how it became one of the most fundamental shifts in how we work with data.

1. INTRODUCTION

What computers do and what is researched in informatics can be simplified as transforming data into new and more useful data.

For example, let's say you have the ages of people who live in Turkey. Let's call this data $\boldsymbol{A}$.

$$\boldsymbol{A} = \begin{bmatrix} 15 \\ 48 \\ 3 \\ \vdots \\ 72 \end{bmatrix}$$

Every row in $\boldsymbol{A}$ represents the age of a person who lives in Turkey. There are about 78,665,830 [2] rows in $\boldsymbol{A}$, and there is no way $\boldsymbol{A}$ can be useful to us without some kind of processing.

Let's say that we have a function $median(\boldsymbol{X}) = \boldsymbol{Y}$ that takes some data as input, and returns the median of that data as output. Lets use $median(\boldsymbol{X})$ on $\boldsymbol{A}$.

$$median(\begin{bmatrix} 15 \\ 48 \\ 3 \\ \vdots \\ 72 \end{bmatrix}) = \begin{bmatrix} 30.1 \end{bmatrix}$$

By using the function $median(\boldsymbol{X})$, we have transformed our data into more useful and understandable data. Now that we have the median age in Turkey rather than only the raw data of the ages of every single person living in Turkey, we can better understand and interpret our data.

In informatics you try to find faster and more useful ways of transforming data, and you try to describe these methods of transformation as sets of instructions. These sets of instructions are called *algorithms*.

algorithm /ˈalɣərɪð(ə)m/ *n*: A process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.[3]

Oxford Dictionaries

Let's define $\boldsymbol{B}$ as the exam results of an imaginary class of size 5.

$$\boldsymbol{B} = \begin{bmatrix} 93 \\ 87 \\ 98 \\ 63 \\ 72 \end{bmatrix}$$

What we want to do is to transform $\boldsymbol{B}$ into more useful information, such as the arithmetic mean of the exam results. How can we develop a set of instructions that will allow us to find the arithmetic mean?

Here is an example:

Algorithm 1: Taking the arithmetic mean

Data: exam results: $\boldsymbol{B}$
Result: the arithmetic mean of exam results
let $sum = 0$;
for ***each*** b_i *in* $\boldsymbol{B}$ **do**
  add b_i to sum;
end
let $size$ = number of elements in $\boldsymbol{B}$;
let $mean = sum/size$;
return $mean$;

Simply put, we add up all the elements in $\boldsymbol{B}$ to generate a sum, and we divide this sum by the number of elements in $\boldsymbol{B}$ to get the mean. Pretty simple right?

2. THE BIG TREND

Well, not every problem is so simple. Some problems are very hard to solve algorithmically, and some are even difficult to understand or to boil down to simpler terms.

Figure 1: What is the next best possible move? Answer: I am not good at backgammon.

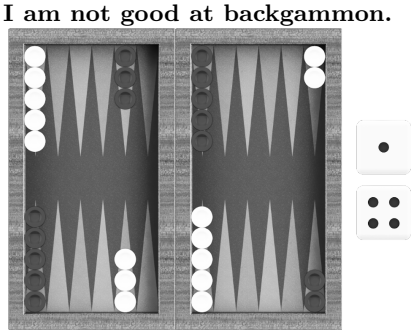


Figure 2: How handsome are the guys in this photo? Answer: Very.

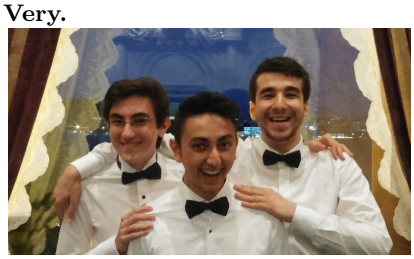
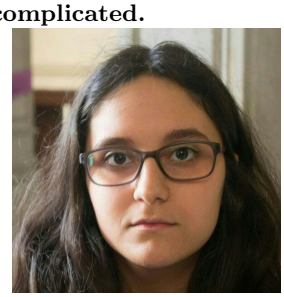


Figure 3: Is the person on this picture happy or sad? Answer: It's complicated.



The big breakthrough in solving these kinds of problems was to allow the computer to learn to solve these problems by itself. This concept is called *machine learning*.

> **machine learning** n: The capacity of a computer to learn from experience, i.e. to modify its processing on the basis of newly acquired information.[4]
>
> *Oxford Dictionaries*

Let's go back to our "transformation of data" definition. What we want the machine to be able to do is to transform some form of data into new and useful data, and we want the machine to be able to learn to do this by itself.

Lets look at the case of backgammon. We can try to teach the computer to approximate the probability of white winning given a board position. We can represent this approximation as the function $eval(\text{Board})$ which takes in a Board position, and returns the probability of white winning given the board position.

$$eval(\text{Board}) \approx P(\text{white wins given Board})$$

How do we find the next best possible move? If we have a really good evaluation function, we can calculate all the following legal board positions (board positions achieved after a legal move was made) and pick the one with the highest probability of winning. If $max_f(S)$ is a function that returns the maximum element in set S according to the value function f then:

$$choosenext(\text{Board}) = max_{eval}(possiblemoves(\text{Board}))$$

Pretty cool right? Don't worry if you got a little bit confused in the last part. What's important is that you get the general ideas behind transforming data and utilizing the result.

In this example, $eval(Board)$ is an approximation of $P(\text{white wins given Board})$. We do not know how to formulate P. We might have some idea on what affects P, like having your checkers close to the finish will most probably result in a higher P. However, if we have machine learning in our tool belt, we can feed the computer with millions of matches of backgammon, and let it learn based on how some positions led to much better results when compared to other position.

This process sounds complicated and is complicated in many ways. However, we can still try to understand some of the fundamental ideas behind it.

In many cases of machine learning, you have some learning data that consists of inputs and usually hand-generated outputs. For example, if we want to teach a computer to recognize faces, we would like to have many example images of people's faces and their corresponding names. Here the face images are the inputs, and the corresponding names are what we call the correct transformations of the inputs. We would then use this learning data to teach the machine.

Now let's look at an example of how a machine can be taught to transform data correctly. Let's say the function f is what correctly transforms our data. The function f is what we want to achieve, but we do not know what f is or how it is formulated. However we are given many examples of correct transformations:

$$\begin{aligned} f(3) &= 15 \\ f(7) &= 35 \\ f(-2) &= -10 \\ &\vdots \\ f(5) &= 25 \end{aligned}$$

Let's define a general use transformation function for numbers:

$$g(x) = c \times x$$

We are going to try to teach $g(x)$ to approximate $f(x)$ by manipulating the constant c. Let's just give a random value to c. I feel like 3 would be a good number to start with.

$$g(x) = 3 \times x$$

To teach $g(x)$ to approximate on $f(x)$ we are going to adjust c according to the correct transformation examples that we have.

Let's take $f(3) = 15$

Currently $g(3) = c \times 3 = 3 \times 3 = 9$

If we increase c by some value, for example $\epsilon = 1$, $g(x)$ is most probably going to get closer to $f(x)$. c is currently 3, and by adding 1 our constant becomes 4.

After the adjustment $g(3) = c \times 3 = 4 \times 3 = 12$

12 is much closer to 15 then 9! So we are definitely in the right direction. Let's formulate our process into an algorithm.

Algorithm 2: Approximating constant in linear function

Data: sample of correct transformations (input, output): ***S***
Result: the approximated constant

```
let c = 3;
let ε = 1;
define g(x) = c × x;
for each input, output in S do
    if g(input) < output then
        set c to c + ε;
    end
    if g(input) > output then
        set c to c − ε;
    end
end
return c
```

If we execute this algorithm on enough samples, we will find out that $f(x)$ is most probably $5 \times x$. This could of course be wrong, $f(x)$ could be a polynomial with an incredibly high degree. However, the more samples we have, the more we can be confident in our approximation of $f(x)$.

You might already be seeing some problems with our algorithm. What would happen if we had set ϵ too high? We most probably couldn't have reached a solution. What if we had set ϵ too low? Then we would need many more samples to reach a good enough solution. What if $f(x)$ was a different function? Then we would have needed another algorithm to find our answer. If $f(x)$ was a polynomial to the second degree, we could have manipulating c_0, c_1 and c_2 in $g(x) = c_2 \times x^2 + c_1 \times x + c_0$ to find us a decent enough approximation.

Even in our modest attempt at machine learning, we have found many aspects in our algorithm which requires further development. This is a really huge and complex area of research, but it is not all rocket science. Every step of the way there are some fundamental and intuitive ideas that, when understood, provide a great understanding of the concepts.

Machine learning isn't new. However it took some time for it to find its place. Here is a good overview of its development throughout the years:

<1950s	Statistical methods are discovered and refined.
1950s	Pioneering machine learning research is conducted using simple algorithms.
1960s	—
1970s	'AI Winter' caused by pessimism about machine learning effectiveness.
1980s	Rediscovery of backpropagation causes a resurgence in machine learning research.
1990s	Support vector machines and recurrent neural networks become popular.
2000s	Deep learning becomes feasible and neural networks see widespread commercial use.
2010s	Machine learning becomes integral to many widely used software services and receives great publicity.

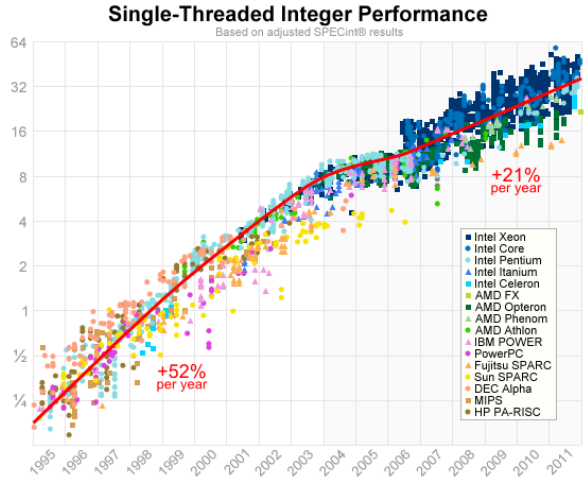
[5]

In the following sections we are going to understand how machine learning got so popular, and how trends in hardware and trends in the availability of data have enabled this technology to be utilized in a much greater scale.

3. TRENDS IN HARDWARE

Computers are mostly designed to do one thing. Execute instructions, fast. That's what CPUs (Central Processing Unit [of a computer]) have been doing since their inception, and they have been getting faster and faster every year.

Let's look at a graph showing how fast CPU's perform at executing sequential instructions on numbers:



[6]

On the graph we see the performance of a single processing core. Performance is certainly growing, but the growth has significantly slowed down. As a matter of fact, it is getting increasingly harder to increase the speed of CPU cores. So what do CPU manufacturers do to solve this issue?

They add more cores.

When you add more cores, the CPU gets the ability to calculate in parallel. So if there are some independent calculations that need to be performed, you won't have to wait to start on the second one until the first one finishes.

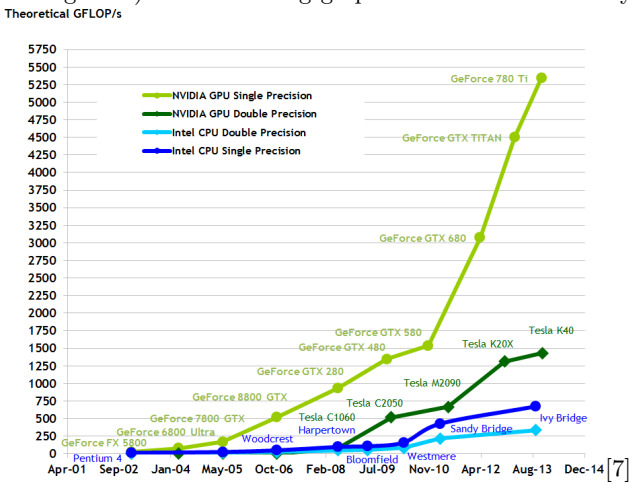
While most algorithms require a significant amount of sequential (non parallel) processing, there are many ways a parallel architecture can help you with many tasks.

You have to execute the same function on multiple data? Share the data among the cores and let them execute the same instructions on multiple data simultaneously.

You have to execute different functions on the same data? Share the functions among the cores and let them execute the assigned functions on the same data simultaneously.

Both of these methods are great for machine learning because there are many cases in machine learning where you do SIMD (single instruction, multiple data) or do MISD (multiple instruction, single data). For example in our approximation of the linear function f, we could have split up the task of checking $g(input) < output$ to different cores. In our example, $g(input) < output$ is a fairly cheap operation so it wouldn't benefit much from parallelization, but on different examples it is easy to see why parallelization could benefit the performance greatly.

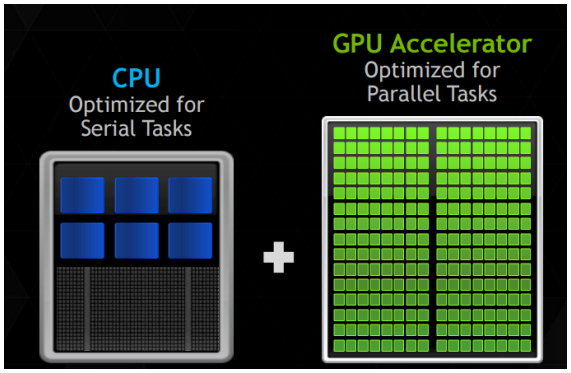
Talking about performance and parallelization, I feel that it is really important to mention GPUs (Graphical Processing Unit). The following graph will make it clear why:



[7]

I want you to focus on the light green line which represents GPU performance on primitive decimals and the dark blue line which represents CPU performance on primitive decimals. GPUs destroy CPUs when it comes to the sheer number of operations they can perform on decimals each second. They are able to do this due to a fundemental difference in their architecture.

GPUs have a lot of cores that are by themselves relatively weak.



[8]

A GPU core is slower than a CPU core (a high end Nvidia GPU's base clock speed is around 4 times slower than an Intel CPU's base clock speed) and supports less types of optimized operations. However, while 16 cores is a high number of cores for a CPU, it is normal to talk about a 1000 core GPU.[9][10]

GPUs are inherently focused on parallel processing while CPUs are designed to be first and foremost good at sequential processing.

This architectural difference in GPUs make them an incredible fit for many machine learning tasks. Affordable GPUs allow such tasks to be accelerated far more than many of the most expensive CPUs. This hardware trend is one of the factors that makes machine learning on big data sets so prevalent today.

4. ADDITIONAL INFO ABOUT GPUS

Why are GPUs designed to be good at parallel tasks? Well, graphical processing is usually a compute intensive task (a task that requires many computations to perform) that can easily be parallelized.

Images on computers are stored as a grid of colors. Each cell in a grid is called a pixel.

> **pixel** /ˈpɪks(ə)l/ *n*: A minute area of illumination on a display screen, one of many from which an image is composed.[11]
>
> *Oxford Dictionaries*

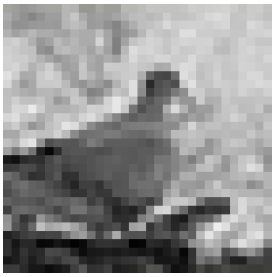
The following picture of a bird consists of a grid of 32×32 pixels:

If we want to turn this image into a grayscale image, we have to transform each colored pixel into a corresponding

gray toned pixel. A grayscaling operation on one pixel is independent of the operations performed on the other pixels, and the operation itself isn't really cost intensive[12]. The issue is that we have to perform it on every single pixel, and even on this low quality image that amounts to $32 \times 32 = 1024$ grayscaling operations.

When performed on a CPU, the grayscaling of the whole image might take some time to perform sequentially on each an every pixel:

However, it is a trivial task for a GPU, that processes all the pixels in parallel:

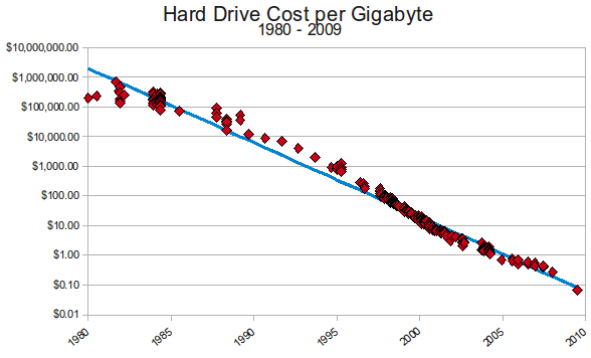


5. TRENDS IN DATA

For machine learning to work well, we need two things:

1. Great computational power
2. Lots of sample data

We talked about how developments in hardware allowed for great computational power to be available. Now let's talk about data.



[13]

The cost of digital storage has been falling in an exponential manner for quite a while now and this development has allowed data to be stored and backed up in amounts that were neither feasible nor believable before. Now companies and organisations store more data then they could possibly need, with the idea that the data may somehow be used someday.

This near infinite source of data has allowed machine learning to take place on a whole new scale.

- The Large Hadron Collider produces 25 petabytes of data every year. [14]
- 4.5 billion likes generated daily on Facebook as of May 2013. [15]
- Photo uploads total 300 million per day on Facebook. [15]
- 300 hours of video is uploaded to Youtube, every minute. [16]

So it should come as no surprise that Facebook has one of the best facial detection software in the world, powered by machine learning.[17][18]

6. APPLICATIONS

There are infinitely many applications of machine learning, here are some examples. Use the citations are further reading material if you are interested!

- Facial recognition [19]
- Handwriting recognition [20]
- Diagnosing patients [21]
- Speech to text [22]
- Translation [23][24]
- Game Artificial Intelligence (Chess, Backgammon, Go) [25][26]
- Showing ads (via predicting Click-Through-Rate, the probability that person A will click on ad B) [27]
- Colorization of black and white images [28]
- Self driving cars [29]

7. SUMMARY

In this article we discussed the main goal of informatics, transforming data into new useful data. We explored about how machine learning on big data sets fundamentally work and broadened our knowledge on its history and development. We learned about the fundamental principles behind CPUs and GPUs, and how this influences the performance of different algorithms. We saw how developments in hardware and the availability of data helped machine learning, a relatively old research topic, explode in the last decade. Finally, we looked at some applications of machine learning that are already in our lives.

I hope that some things you learned from this article have been useful in helping you make sense of this "hot topic", and maybe even be useful to you in the future.

8. REFERENCES

[1] Informatics. http://www.oxforddictionaries.com/definition/english/informatics, 2016.

[2] Turkey: Population. http://data.worldbank.org/indicator/sp.pop.totl?locations=tr.

[3] Algorithm. http://www.oxforddictionaries.com/definition/english/algorithm, 2016.

[4] Machine learning. http://www.oxforddictionaries.com/definition/english/machine-learning, 2016.

[5] Timeline of machine learning. https://en.wikipedia.org/wiki/timeline_of_machine_learning.

[6] Jeff Preshing. Preshing on programming. http://preshing.com/20120208/a-look-back-at-single-threaded-cpu-performance/.

[7] Cuda c programming. http://docs.nvidia.com/cuda/cuda-c-programming-guide/.

[8] Larry Brown. Deep learning with gpus. http://www.nvidia.com/content/events/geoint2015/lbrown_dl.pdf.

[9] Geforce gtx 1080 graphics card. http://www.geforce.com/hardware/10series/geforce-gtx-1080.

[10] IntelSupport. Intel core i7-6700k processor specifications. http://ark.intel.com/products/88195/intel-core-i7-6700k-processor-8m-cache-up-to-4_20-ghz.

[11] Pixel. http://www.oxforddictionaries.com/definition/english/pixel, 2016.

[12] John D Cook. John d. cook.

[13] matt komorowski. a history of storage cost, Sep 2009.

[14] Worldwide lhc computing grid. https://en.wikipedia.org/wiki/worldwide_lhc_computing_grid.

[15] Top 20 facebook statistics. https://zephoria.com/top-15-valuable-facebook-statistics/, Jul 2016.

[16] 135 youtube statistics. http://expandedramblings.com/index.php/youtube-statistics/, 2016.

[17] Inside facebook's biggest artificial intelligence project ever. http://fortune.com/facebook-machine-learning/.

[18] Mathew Ingram. Facebook's new algorithm can recognize you even if your face is hidden. http://fortune.com/2015/06/23/facebook-facial-recognition/, 2015.

[19] Openface. https://cmusatyalab.github.io/openface/.

[20] Michael Nielsen. Using neural nets to recognize handwritten digits. http://neuralnetworksanddeeplearning.com/chap1.html, Jan 2016.

[21] Filippo Amato. Artificial neural networks in medical diagnosis. http://jab.zsf.jcu.cz//11_2/havel.pdf, Jan 2013.

[22] @googleresearch. The neural networks behind google voice transcription. https://research.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html, Aug 2015.

[23] Steve Dent. Google is using neural networks to improve translate. https://www.engadget.com/2016/03/11/google-is-using-neural-networks-to-improve-translate/.

[24] Keith Stevens. Neural network models and google translate. http://ufal.mff.cuni.cz/mtm15/files/11-neural-network-models-and-google-translate-keith-stevens.pdf.

[25] Td-gammon. https://en.wikipedia.org/wiki/td-gammon.

[26] Alphago. https://en.wikipedia.org/wiki/alphago.

[27] Kevin Markham. Beginner's guide to click-through rate prediction with logistic regression. https://turi.com/learn/gallery/notebooks/click_through_rate_prediction_intro.html.

[28] Richard Zhang. Colorful image colorization. http://richzhang.github.io/colorization/.

[29] Mariusz Bojarski. End-to-end deep learning for self-driving cars. https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/, Aug 2016.