META LEARNING

How To Learn Deep Learning And Thrive In The Digital World

RADEK OSMULSKI

Table of Contents

For Lily and Rose

Introduction

I wrote this book because on my long and winding deep learning journey I learned some things that I believe can be useful to others. Many of the insights that I share can accelerate your progress several-fold.

In 2012, I couldn't program. I had a corporate job that provided a steady stream of income, but it was utterly boring. There had to be more to life than thinking of human beings as resources. I read somewhere about these things called MOOCs, and I decided to take a look.

By 2018, I had done enough deep learning to win a Kaggle competition,[1] just a few months after a blog post I wrote while taking a fast.ai course got published by KDnuggets.[2]

In 2019, I received an invitation to visit my colleagues at Curai, a healthcare startup in San Francisco. Out of around 20 engineers, I was their only team member from overseas.

Figure 1. Flying to visit my colleagues in the US

As I write these words, I have the most amazing job I could imagine. I get to work on decoding non-human communication as part of the Earth Species Project with some of the most amazing, interesting people I have ever met.

The reason why I believe I have something worthwhile to share with you is not, however, related to the aforementioned achievements.

Over the years, I've learned a lot about learning and acting effectively. This change – and the ideas that led to it – are the essence of what I would like to tell you about.

At the beginning of my journey, I moved very slowly. I studied for months, only to discover that my deep learning abilities had not progressed much. I couldn't chart a path that would take me from where I was to where I wanted to be. Then, on one of the many detours I took, I became a web developer because I had started to believe machine learning might be out of reach for me.

What allowed me to persevere was my belief in learning. Even if I wasn't learning quickly, I was still learning, and that was something.

I started to reflect more and more on what worked for me and what didn't. I also managed to keep an open mind, always ready to accept advice (though that might have been more out of desperation than anything else). Often, I would write my thoughts down and share them with others.

Figure 2. A tweet from a member of the fast.ai community. Thank you, Kevin!

This process of introspection was key in motivating me to continue. Through it, I began to employ new ideas, and refine my approach.

I had to cover a lot of distance, as I don't have a college degree. I only started to learn to code when my first daughter was about to be born. This gave me the

focus I had lacked earlier, but it also placed some rather significant constraints on my time.

If my approach to learning hadn't undergone such an immense change, I would not be writing these words to you today.

In this book, I share the life-changing ideas I picked up along the way. They will allow you to learn faster and to accomplish more in a shorter period of time. They can alter the trajectory of your life, just as they have mine.

[1] The iMaterialist Challenge (Fashion) at FGVC5 was a multi-label, fine-grained computer vision problem. The competition was fierce, but I managed to come in first place by a wide margin. You can read about my solution here.

[2] How to do machine learning efficiently - this article describes how to structure your machine learning project to achieve the best results in the least amount of time.

Who is this book for?

I wrote this book for those who want to learn deep learning but have access to overwhelmingly impractical advice.

For people who are passionate about learning fast and learning well.

For those entrenched in the academic mindset and who value understanding theory above all else. I have suffered greatly from this affliction myself. You cannot learn the theory well if you don't put practice first. In this book, I provide a cure.

For those who didn't start to program when they were 12. For those who had to teach their parents how to use the computer, not the other way around.

For the timid dreamers who suspect that technology can change their lives but don't have all the answers yet.

How to read this book

> "You can’t connect the dots looking forward. You can only connect them looking backwards"
>
> — Steve Jobs, 2005 Stanford Commencement Address

The chapters of this book are loosely related and can be read in any order. The best approach would probably be to read them in sequence, and then return and review chapters of interest over time.

To help the ideas I describe permeate your life, talk about them with your friends or loved ones. Tweet about a thought with which you agree or disagree. Maybe write a blog post elaborating on one of the concepts. I continue to experience firsthand how powerful the processes of introspection and sharing with others can be.

Change is gradual, and will not happen overnight. Pick an area you would like to work on, give it some time, and, only after a while, move to another.

Above all, be gentle with yourself. Most of the time, I feel like I’m standing still. When you view your life from the perspective of days or weeks, it can be really hard to notice progress. But, to start getting exponentially different results, you need to start doing just a few things differently on a regular basis.

No superhuman effort is required. Do a few things differently, and, very soon, you will be able to look back and start connecting the dots that brought you to where you ended up.

From not being able to program to deep learning expert

Before you can learn to do deep learning, you must become a developer. Being a developer is not only about programming. In the larger scheme of things, being able to write code is just a small part of what a developer can do.

If I were to start all over, I would optimize this part of the journey for fun. I would want to get into the tinkering mindset and start feeling comfortable messing around on my computer. There is no better way to achieve this than by working on something you truly enjoy.

I would take it easy. When it comes to learning, consistency beats intensity every time. When you are just starting something, there is no better way to stay consistent than to not exert yourself too much. I would still make quick progress, because this time around, I would know what to focus on. This is what the beginning of this chapter will cover.

My first stop would be to complete one of the programming MOOCs. They don't teach you a lot about being a developer, but they do teach you some things about writing code. I would probably go for Harvard CS50 on edx. And, as I want to have fun, I might choose the game programming track.

Harvard CS50 has one great thing going for it: On top of teaching you how to program, it also teaches you about the computer and how it works! Two birds with one stone. Still, regardless of which MOOC or programming book you chose, it is hard to go wrong. Programming has been taught for many generations, and many good teachers are out there. The biggest mistake you can make at this stage is to do more than a single beginner programming course. Striving to learn any particular programming language really well would be a mistake.

It doesn't make sense to invest all of your time into learning calligraphy if you have nothing to write about! Likewise, it doesn't make a lot of sense to work on building a table but focus all your effort on a single leg and then complain that the table will not stand!

All we need at this stage is some familiarity with programming concepts – enough so that we can perform meaningful online searches. The programming skills we need are understanding the value of Stack Overflow, documentation, and how to reach both. This might sound tongue in cheek, but these skills are genuinely at the core of a programming career. Add knowing a few basic things about programming, and you have a very firm platform on which to build.

The next leg of the table we are assembling is our ability to move effectively in code. This is not something that most programming courses cover, but it is an essential skill to have.

This boils down to learning to use a code editor really well. Any major code editor will do. Some people swear by textmate, sublime text, atom, vscode, emacs, or vim. It doesn't matter which one you choose. Focus the same attention on learning it as you would to writing code and you will be alright. If you are absolutely lost as to which one to pick, chose vscode. Do not worry about the choice too much – it is not one you must stick with forever. I do not know of a single developer who has not tried multiple editors in the course of their career. However, I also don't know a single developer worth their salt who doesn't believe that learning to use an editor is an important skill.

To work on our next leg, we move to version control. Code is written in small iterations, often by more than one person. Additionally, code is so expensive to produce that we want to make sure we don't lose our work and that we can explore new directions without destroying the work we have done up to now. All this is very hard to achieve without specialized tools. Thankfully, we have a

software solution that addresses all the needs listed above and then some!

The tool goes by the name of git. It is a small command line program that can change any directory into a git repository. In the process, it endows the directory with superpowers, such as an ability to travel back in time or create alternate timelines!

Git comes with an online counterpart, called GitHub. While git excels at managing the source code on your computer, GitHub is great for code sharing and collaborating with others. Pushing your code to GitHub also provides you with a backup. If some accident should befall your PC, your work will be preserved at another location.

Git is a marvel of software engineering but, unfortunately, the API it exposes is not very human-friendly. Thankfully, GitHub created its own command line tool, the gh cli. It has a much gentler learning curve than git.

The last leg of our four-legged table is probably the hardest to construct. The challenge lies solely in the lack of good learning resources. This part deals with learning how to use the computer in the context of writing code. For deep learning, this goes beyond your local setup. How do you spin up a cloud VM? How do you ssh into it? How do you move data around? How do you build your own home rig for experimentation? The good news is that you do not have to learn all of this at once. A great starting point might be to learn how to navigate the file system on the operating system of your choice. As you get more comfortable using the command line, you can gradually add more commands to your arsenal.

The one resource that stands out for learning how to use a computer is The Missing Semester of Your CS Education by MIT.

Maybe a day will come when a single course will cover all the basics of being a

developer. For now, unfortunately, we must piece together our own curriculum. The good news is that it is enough to get just a taste of each of the four disciplines I outline above; there is no need to become an expert at any of them.

All we have been doing thus far is preparing ourselves for taking the "Practical Deep Learning for Coders" course by fast.ai. If you have a rudimentary understanding of programming and can move around on your computer, the course can take you from there.

And why a fast.ai course and not something else? No other resource out there will teach you everything you need to know to do machine learning at a high level. Many good offerings focus on a particular set of tools or some aspect of the machine learning pipeline, but no other complete package exists[3]. This is the only course I am aware of that will teach you the end-to-end process of arriving at a good solution to a machine learning problem.

In fact, for many, the course goes beyond showing one how to be a good deep learning practitioner. Through taking the fast.ai courses, I learned how to learn. I also started to share my work and became an active member of the deep learning community. The courses not only taught me nearly everything practical that I know about machine learning but also showed me how to thrive in the digital age.

If you give the lectures a proper go and become an active member of the fast.ai forums or fast.ai discord, you will be alright. Nonetheless, the courses propose a way of learning that might be new to many. While the course explains the approach, a written summary of it, from my perspective, with the entire progression explained in one place, might still be beneficial. This is what we will be looking at in the remainder of this chapter.

It all begins with watching a lecture. The next step is to open the lecture notebook and figure out how all the pieces fit together. The idea is to run each

line of code and look at the outputs. If you come across a function you have not previously encountered, this is a good time to read its documentation. How will the performance differ if you pass in different arguments or slightly tweak the hyperparameters?

Once you understand the bigger picture, it's time to reproduce the results. To do so, open a new notebook and try to recreate the training pipeline that the lecture demonstrated. It's an open-book exercise – it's okay if you need to look back at the lecture notebook, but the less you do so, the better. On the other hand, there are no limitations on reading the documentation or searching online. Both of these activities are encouraged and they closely resemble what programming feels like when it's done completely on your own.

Once everything is up and running, and you have obtained a result similar to that in the lecture, it is time to go a step further. This time, we search for a similar dataset and test drive the technique we just learned about. Your imagination is the only limit here. You could seek out a dataset online or you could put one together yourself.[4] Alternatively, the fast.ai library provides access to many datasets used for research. You can download them to your machine with a single command.

The idea is to systematically grow your skills. You start in a controlled setting of the lecture notebook, learning the ins and outs of the technique that the lecture discussed. With every new exercise, more of the training wheels come off, to the point where you're doing everything on your own from start to finish.

This approach to learning works like a charm. There are no hard and fast rules here and you are more than welcome to modify it as you go. Being creative and coming up with exercises that are challenging and interesting to you is a very valuable skill in its own right; acquiring it will take your learning to the next level. The result is always the same: to arrive at a place where you can

comfortably apply the techniques you learn in the lectures to problems you haven't previously seen.

While practice is the only way to improve your ability to solve real-life machine learning problems, moving toward this way of learning does not require you to revolutionize your life. A couple of minutes a day, over multiple days, is all you need to get started. With every consecutive day of learning, you build momentum. Soon you start to notice the results, and the ability to do things you thought were out of reach not too long ago creates a lot of excitement. Very soon, you will find yourself losing track of time as you explore this or that technique.

From the broader picture, going through lecture materials is a starting point. It gives us a platform upon which we can build. Even before you finish the course, it is a good idea to start coming up with and working on mini-projects of your own. If you are low on inspiration, you can check out this amazing forum thread with projects that fast.ai students worked on during one of the courses. There, you will find an assortment of ideas: blog posts, howtos, forum posts, GitHub repositories, explainer videos, and Kaggle competitions.

The way of learning I describe above was completely new to me. However, after experiencing what it can do, I have never ceased being a fast.ai student. Any other approach feels like a waste of time.

It wasn't always like that. Initially, I thought that the fast.ai way of learning could not work. The exercises proposed in the lectures didn't feel like learning to me. How could I make any progress spending so much time *doing*? To me, learning was this process in which you sat down with a book for an hour or two and came out on the other end being able to discuss something highly complex and abstract.

I was very lucky, as at the time I found fast.ai I was at the end of my rope. Nothing was working, so I was ready to try something completely new.

Stumbling across fast.ai turned out to be the adventure of my life.

[3] In the initial phase of my Deep Learning journey, only the fast.ai lectures and the forums existed. I talk to them because this is how I learned. But in 2020 a new contender arrived - the much anticipated Deep Learning for Coders with Fastai and PyTorch: AI Applications Without a PhD by Jeremy Howard and Sylvain Gugger. This is a spectacular book and it is a treasured companion I keep on my desk. My recommendation would be to study it as dilligently as the lectures.

[4] Creating a dataset is a great way to learn in general! Figuring out what data to include and how to assign labels makes you think through many important considerations for training your model. It is a great idea to include dataset creation at some point in your learning journey.

Theory vs practice

For most of my life, I didn't realize that machine learning existed. Then, one day, I stumbled across Andrew Ng's machine learning course on Coursera and my life changed forever.

I viscerally remember the sense of awe I felt when I watched a couple of fully connected layers converge to form a solution. I implemented them from scratch in Octave and the operations were fully vectorized. This was the most amazing thing I had ever seen a computer do. What other tricks did it have up its sleeve? I had to find out.

Not knowing anyone who was a developer, I began searching online. Very quickly, a curriculum emerged. According to the Internet, all I had to do was learn some calculus, a bit of linear algebra, and some probability theory and I would be done. Easy peasy.

Calculus seemed like a good starting point. A lot of new and exciting MOOCs were being launched in this space. Plus, lectures from some of the best universities in the world were only a click away on YouTube.

But lo and behold! To understand some of the more advanced calculus proofs, you really had to have the construction of real numbers figured out. Without this, some of the higher math becomes hand-wavy. So, I needed to venture into real analysis for a while. No problem at all. "Principles of Mathematical Analysis" by Walter Rudin and another set of lectures to the rescue!

But what is this? Oh, dear. It now appears I have to learn the set theory to fully understand what is going on.

This part was fun. I got to know Mr. Cantor and got to observe firsthand how

science is done. Turns out even something as abstract as set theory evolved over millennia! It is really hard to appreciate the modern result without understanding the path that humankind took to arrive at it.

Through all of this, my main goal was to learn deep learning and I believed I was on the right path. I could derive the equations to backpropagate the loss through all the layers in a CNN in my sleep. I would implement scientific papers in NumPy[5] (I didn't realize PyTorch or TensorFlow existed), and that felt like making progress.

Then, one day, I signed up on Kaggle. I looked around, and worked through the Titanic competition, but other than that, I was lost beyond belief. All I had ever worked with were toy datasets. How do you feed all this peculiarly formatted, often dirty, data into your model? I looked at the forums and could barely understand the language the people spoke there.

I poured so much of my heart and energy into understanding machine learning. Why was I that useless?

The problem with theory

> "I have not failed. I've just found 10,000 ways that won't work."
>
> — Thomas A. Edison on his path to inventing the light bulb

Overwhelmingly, theory follows practice. If you look at the history of inventions, you'll see that very rarely is a new discovery made by someone extending a theory and using it to conjure something new.

One of the most fascinating examples of the theory first approach is the periodic

table. In 1869, Russian chemist Dmitri Mendeleev, had an epiphany. Instead of organizing elements into a system based on their properties, which is what his predecessors attempted, he decided to use atomic mass as the organizing principle.

This approach worked like a charm. Using these new theoretical underpinnings, he was able to predict the structure and properties of elements still to be discovered. For instance, in 1871 he predicted gallium, which wasn't discovered until 1875.

This example of the theory first approach is so famous precisely because of how unique it is! An overwhelming majority of the time, it is the Great Tinkerer that arrives at something, via experimentation; only later does theory follow. Sometimes the discovery and the theory are many years apart.

A good example of this is Batch Normalization[6], developed jointly by Sergey Ioffe and Christian Szegedy. Both authors are highly acclaimed researchers working at Google Brain. Szegedy is also the inventor of the inception architecture[7] and the discoverer of adversarial examples[8] — two very important contributions that have moved forward the entire field of deep learning.

As for Batch Normalization, it is an astonishing technique in its own right. It cut the number of steps needed to train a state-of-the-art image classifier by 14 times, which is an unbelievable result! Not only did it allow for much faster training, but it also made training neural networks easier. Using this novel technique, we no longer had to be as careful as we were before about initialization.

How did the authors explain this revolutionary discovery? In the paper, the performance improvement is attributed to alleviating the covariate shift.

Covariate shift refers to the changes in the input distribution during training. Consider this: If we tweak the weights of the first layer of our model, the inputs into the second layer will no longer have the same mean and standard deviation. As the new inputs flow up our model, this effect will get amplified.

This variability of inputs slows down the training of our network. Intuitively, not only will the model be learning to do something useful on our data, but each layer will have to engage in a considerable amount of computation to continue learning how to adapt to changing distributions of its inputs. If we can solve the problem of changing inputs, we should be able to train faster and better. This was a brilliant insight and research direction! But, as it turns out, it wasn't until several years after the publication of the Batch Normalization paper that the reason why it works so well was more fully formalized. Turns out, more is amiss than the inventors of the method originally suspected there was!

The point is this: Ioffe and Szegedy didn't arrive at this discovery by working out the theoretical underpinnings. They were guided by their stellar understanding of deep learning and their vast scientific knowledge. However, even in this highly algorithmic, mathematical domain, theory followed practice.

And this is not even the most extreme example of practice taking the leading role. You may have heard about the ULMFIT paper[9] by Jeremy Howard and Sebastian Ruder. This is another seminal paper that shifted the trajectory of an entire field. Discriminative learning rates, progressive unfreezing, transfer learning across many complex layers! Where was the theory for all of this before the paper got published? Where is the theory now? All of these techniques are deeply rooted in a very good intuition of how an architecture will perform and a very good understanding of applications across multiple domains. These were all arrived at through practice and experimentation, and so a new and exciting era in NLP was born.

In case you aren't fully convinced of the value of practice, here are another couple of examples for your consideration.

Ian Goodfellow is a giant of the field and the inventor of generative adversarial networks. One night in 2014, he went out for drinks to celebrate his friend's graduation. Over beers, a couple of his friends brought up a problem they didn't know how to solve: How can we get the computer to generate images? Goodfellow suggested a novel solution, though his friends were skeptical. What did he do next? He went home, fired up his computer, and began to work. As the morning approached, he had his implementation of the novel architecture ready. He tested it, and it worked.[10] In another example of an unparalleled ability to experiment, an entirely new field of research was born.

The examples could go on and on. To finish off this inquiry, let's consider Geoffrey Hinton, one of the undisputed fathers of modern deep learning. In the eighties and throughout the nineties, no one really knew how to train deeper architectures. Would we need to resort to training neural networks one layer at a time using Boltzmann machines?[11] What can neural networks learn, anyhow?[12] How do you make sense of competing theories, such as the localist and distributed theories of neural representation?[13] How do you navigate this new land that, according to many academics who had solid theoretical grounding, was not worth giving attention to? Reading Hinton's papers one can get a sense of how empirical he was in his approach. Regardless of whether the inspiration came from physics[14], or optimization theory[15], or biology[16], the mechanism of expanding the frontier of what we know was always the same — that of experimentation.

The purpose of this discussion is not to belittle theory. Of course theory is useful and even critical to doing anything worthwhile in machine learning. But even if you aim to reach the highest echelons of deep learning research, you first and foremost need to be a great practitioner!

If theory is not the end all be all as many would portray it, how do you put it to good use in learning and discovery? Where did I go wrong in my relationship to theory that rendered me incapable of making progress?

For best effects, use one cup of theory, one cup of practice. Rinse and repeat

My main shortcoming was that I didn't know what theory I had to learn! I was learning but what I learned was not aligned with where I wanted to go.

Another, darker effect, is at play here. We might be reading papers, we might feel our understanding grow. But without training actual models, without the experience of applying what we are learning to real-life problems, we are missing a very important feedback loop. This feedback is essential, as without it we have no way of knowing whether our recently gained understanding is, indeed, correct. We have no good way of identifying the blind spots.

How should I have proceeded as a learner? Say I wanted to learn about image classification. I should have began with downloading a relatively small dataset but, ideally, one that had some benchmarks I could refer to. Imagenette by fast.ai [17] would be a great candidate, as it comes with a leaderboard and references working implementations. I could then implement a basic model and evaluate the results. In the case of Imagenette, the classes are balanced and the data is cleanly labeled, so maybe I would come to the conclusion that my best bet for improving the results would be to work on my architecture. Maybe I would choose to add identity paths to my model.[18] Or maybe I was already using a resnet architecture, but wanted to go a step further and potentially make improvements to the input stem.[19]. Or maybe I would feel that I had to better understand how to optimize the training process itself.[20]. I would follow any of these paths and learn the related theory in the process.

Once I would start getting decent results and feeling comfortable, maybe it would be time to move onto another setting. Maybe I would come across a Kaggle computer vision competition that would spark my interest. Chances are, this setting would produce a different set of challenges. Maybe the classes would not be balanced and I would need to learn more about addressing this.[21] Or maybe the labels would be noisy.[22]

Through this entire process, I would be running experiments, which, in its own right, is a very good skill to grow. I would be guided in what to learn by direct feedback from the model and I could put to test everything that I was learning. I wouldn't necessarily need to fully understand what improvements I could apply to address any given problem, as I could research the issue in the moment. fast.ai courses teach you how to approach this, and if you spend some time on fast.ai or Kaggle forums, you will pick up on the major pain points that people encounter. The specifics might change from dataset to dataset, but the main categories of problems to address do not.

Through it all, I would also be building a nice set of results I could talk about in any potential invitations to professional collaboration.

How much faster would I have progressed had I taken this approach as opposed to taking a random walk through various fields of mathematics?

Practice wins every time

> "There's just a way of learning that applies to most domains, and once you sort of find that to be successful in one domain, it's not that hard to just basically say 'copy paste'."
>
> — David Heinemeier Hansson, An interview with a real-world superlearner

The notion of learning what you need as you go is not exclusive to deep learning. Many high achievers from various fields would attest to its prowess. But, as I believed in the primacy of understanding arrived at through studying theory, this notion didn't appeal to me. As a result, I progressed much slower than I could have otherwise.

I now see where I was wrong. You cannot achieve proficiency in a field, no matter how theoretical it seems, without making practice your highlight.

[5] 10 Neural Networks - an implementation of 10 neural networks in Matlab and numpy based on Geoffrey Hinton's papers

[6] Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

[7] Going Deeper with Convolutions

[8] Intriguing properties of neural networks

[9] Universal Language Model Fine-tuning for Text Classification

[10] The GANfather: The man who's given machines the gift of imagination

[11] Boltzmann Machines

[12] Parallel distributed models of schemata and sequential thought processes.

[13] Learning distributed representations of concepts — this is one of my favorite papers, of all time

[14] A learning algorithm for Boltzmann machines.

[15] RMSprop

[16] Dropout: A Simple Way to Prevent Neural Networks from Overfitting

[17] Imagenette is a subset of 10 easily distinguishable classes from ImageNet (tench, English springer, cassette

player, chain saw, church, French horn, garbage truck, gas pump, golf ball, parachute).

[18] Deep Residual Learning for Image Recognition

[19] Bag of Tricks for Image Classification with Convolutional Neural Networks

[20] A disciplined approach to neural network hyper-parameters: Part 1 — learning rate, batch size, momentum, and weight decay

[21] A systematic study of the class imbalance problem in convolutional neural networks

[22] Does label smoothing mitigate label noise?

Programming is about what you have to say

My first programming book was the pickaxe. It's a legendary manual on the syntax and features of the Ruby language. I spent hours poring over it, as I believed that understanding a language in great detail was what made one a good programmer. Oh, how mistaken I was!

Your ability as a developer is measured by the utility of the things you can express in a language. The ability to construct, from crude stone, buildings that withstand the test of time has much more value than the ability to chip stones into exquisite shapes that you are unable to weave together.

Additionally, languages are invented to make programming easier, to serve some purpose. That means if you are at the start of your journey as a programmer, the only way you can really understand a piece of syntax is to observe it as part of a whole—where an expression carries some meaning and integrates in some fashion into a whole.

How, then, should one learn to program or learn a new programming language? The best way is by reading and writing a lot of code. This is how you become fluent. For reading, what usually works very well is seeking out small, maintained repositories on GitHub. For starters, anything around 100-200 lines of code is perfect, but anything under 1000 lines of code will do. As for writing, write to your heart's content. If you are stuck, search for small programming projects in a language of your choice.

Even if you learn a language well, that is at most a starting point—a nice tidbit of information you can share with your friends. There used to be a time when I knew how to write a bit of Haskell, and that served as a nice talking point when having coffee.

The truth is, no one cares! What really matters is what you can say in a language—whether you have enough domain knowledge to say anything interesting.

Maybe you want to become a web developer. You study how web applications work and, along the way, you naturally pick up some Ruby or JavaScript. Or maybe you want to be a deep learning practitioner. You study neural networks extensively and learn some Python and PyTorch.

The big secret is that domain knowledge comes first. There is an art to writing clean, maintainable code, no doubt about that. But you cannot put the cart before the horse. You can study naming, refactoring, dependency inversion, and so on only once you have a lot to say. Only then will the more advanced features of a language make sense. And only then will they become important.

The fastest way to learn to program is to learn how to say something useful.

The secret of good developers

You sit down to write code. If the code is particularly challenging, many minutes will pass before you put down the first character.

First, you might need to check where a function you want to call is defined and exactly what it does. What tests do you have for the part of the codebase that you are working on? It might be worth reading them to figure out what is tested and how things fit together. At some point, you may recall a blog post you read that offers a nice conceptual framework for what you are about to implement. You pull it up and start reading.

Finally, very cautiously, you write the first line of code. And then another. An hour passes and you have two lines of code but that is good progress. Now you are in your zone; you have everything you need and have reached your peak productivity. Additional lines of code can start to flow.

The loading of information into your brain is a valuable and important process. However, it is not cheap. It takes time and energy. Something similar happens even when you are working on a task so straightforward that you sit down and immediately start pushing one line of code after another. You create a mental image of how each line of code that you write connects to the one before it, and how it might relate to the lines of code that come next. This requires a lot of mental energy.

A similar process takes place when you are studying something complex. As you continue to read about the subject, you begin to hold an intricate structure in your mind. So many ideas connecting to one another!

After an hour or two of such effort, you can look back with admiration at what you have done.

But what happens if you get interrupted?

It doesn't matter how trivial the interruption is. Maybe it is a message from your friend that you respond to. Maybe it is a piece of news about your local football team that you read.

The moment you consider something other than your main activity, your mind switches contexts. It drops the information you were holding in the air and starts loading information pertinent to the distraction. After just a couple of minutes, when you return to your main activity, some of the information you originally held in your mind will be gone. You might have to read the code you wrote earlier or go back to the definition of a function you rely on and read it again. Either way, you will not program or learn as effectively as you did before the interruption. To return to the original state, you will pay a price in terms of time and energy.

Developers refer to what happens when you change focus as "context switching." Many of the best developers believe that the price of this operation is extremely high.

If you want to make progress on learning or writing, be it text or code, I do not know of a better technique than long, uninterrupted sessions.[23]

Give it a shot for a while. Put away the phone. Close your browser or move it to another workspace. You will be surprised at how much you will achieve.[24]

[23] This concept is sometimes referred to as "deep work." It is so powerful that entire books have been written on this single idea alone!

[24] A single session of deep work might last anywhere from an hour to two-and-a-half hours. One session a day, if you can afford it, will enable fast progress on any project. Two sessions is believed by many to be the max that is humanly possible. I don't think that writing code for five hours a day for several days straight is manageable. Even a single day can become a challenge. As we become tired, our work is not only of lower quality but might be counterproductive in the long run. The same applies to learning. Note that this pertains to focused work. I don't include writing emails, attending meetings, or filling out paperwork in that category. In most professional

settings, a developer will be very lucky if they can get in one uninterrupted session of deep work per day. Long live working from home!

The best way to improve as a developer

> "You don’t sharpen your skills with resources, books, or articles. You sharpen your skills with practice. If you want to get better, go do the thing."
>
> — Jason Fried, a tweet

Contrary to popular opinion, there are only two methods that are guaranteed to take you from not knowing much about writing code to being a great developer. They certainly do not include passively watching YouTube videos, taking MOOC after MOOC, or binging on computer science textbooks. While all of these are good side dishes, they make for a poor main course.

The two time tested practices that work in this regard are reading and writing code. They lie at the core of what being a developer is all about.

You can find a lot of code to read on GitHub. I would lean toward smaller, well-maintained repositories. Do not be discouraged, as initially, reading open-source code will feel like drinking from a firehose. But give it a few weeks and you will be surprised at how many things start to make sense.[25]

Writing code is even more straightforward. You come up with an idea for a mini-project and bring it to some state of completion.[26], you can also check out some of the projects that I worked on.

There is no need to work on anything fancy. Besides, many impressive projects started with a few lines of code thrown together to explore an interesting idea.

If you are absolutely stuck for ideas, you can do a quick online search. Sharing mini-project designs is a theme beloved by bloggers and YouTubers alike.

And that is all there is to it. The key to becoming a great developer is reading and writing a lot of code. The quality of the code you produce along the way doesn't matter as long as you continue to read and write.

[25] RailsConf 2014 - Reading Code Good by Saron Yitbarek offers additional insights on reading code. It's a highly recommended watch.

[26] The skill of building mini-projects, which take anywhere from a couple of hours to a couple of weeks to build, is extremely valuable. This is how great developers continue to learn. As an example of a very impressive mini-project, check out miniGPT by Andrej Karapthy or some of his other work. Also, take a look at the many projects launched by fast.ai. Some of them look very impressive right now, but all of them started with just a few lines of code. Some of the repositories are still in the infant stage. Going through them, one can see how each project is about both delivering value and exploring a new idea or approach. On my my personal website

How to use your tools to achieve a state of flow

You are in a state of flow when you are immersed in your work — when nothing separates you and your work and nothing else enters your awareness. You act and the medium you are working in provides an immediate response.

To lose yourself in your work like this is extremely gratifying. The sensation is great and you produce high quality work at impressive speed. You transcend being a developer and become an artist.

Achieving a state of flow is not a given, but there are ways to make it more probable. We can organize our work to remove obstacles, so that it will flow from our hands.

This is the reason why many programmers spend a lot of time studying their editors and swear by the ones that don't require lifting their fingers from the keyboard. When you reach for your mouse, your flow is interrupted. You introduce a delay between your intention and the results appearing on the screen. The clunkiness of operating the mouse comes between you and your work. Ideally, you want nothing to separate the two.

Additionally, there are many other distractions that we are likely to encounter when we write deep learning code. We might be getting disconnected from our VM. We might be spending an inordinate amount of time troubleshooting a subtle bug. Our wireless keyboard might be running low on batteries and require charging.

It does not matter how good we are; we are bound to keep encountering these situations. But through planning, leveraging tricks such as creating toy datasets to run experiments on, and learning our tools well, we can move toward the

ideal.

And now comes the really good part. We are never fully in a state of flow — and, conversely, we are never completely not in it. We are always somewhere on a continuum between these two extremes. That means we do not have to worry too much about achieving the full experience of flow. All we can do is take small actions that bring us toward the ideal. Any change to your work process or environment counts, even if it's a small one.

Making small adjustments and seeing how they work is all it takes to start experiencing flow on a more regular basis. This can take our productivity to a whole new level.

Use reality as your mirror

I tinkered on and off with Ruby on Rails[27] for around two years. I even used it at work to build an application for generating custom reports, though my role was in management at that time. I didn't dream of getting a job as a developer — how could I possibly get one with no education? To me, programmers were mythical creatures, possessing sage wisdom not attainable by a mere mortal — or at least not a mere mortal who would spend his free time creating toy applications on his laptop.

Luck had it that I came down with a bad case of flu and was stuck at home. Not having much to do I scrolled through my Facebook feed and noticed an ad from a local Rails shop. They were looking for someone to join them.

Without too much thought, I responded that I was interested. I got an invite for an interview very quickly and soon another company reached out to me as well. There was nothing to lose, so when I got better, I went for both interviews.

Both of the interviews ended in a job offer on the spot. I was blown away. Living in my shell of ideas I had no clue how starved the market was for Ruby on Rails developers. I also didn't realize how little people in the industry knew about what I thought was important and how strong they were across other dimensions. Turns out you could create multimillion startups writing Ruby on Rails without peering into its source code. But you had better be very good at being able to churn out code quickly and make the website look professional. I had a completely wrong mental model of what being employable meant.

In the end, I got lucky. By sheer accident I learned a framework that was en vogue, I had other characteristics sought after in an employee, and the rest is history. But the lesson remains — if you want to live a different life tomorrow than you are living today, you have to put your beliefs to the test. And going

from not being a deep learning professional, to being one, might require questioning a lot of assumptions. This is what this book is about. Each chapter describes an idea or set of ideas that I held earlier and how they changed with experience.

You can help this process by actively noticing where you jump to conclusions and are not opening yourself up to feedback from the world. The stronger the emotions involved in a situation, the higher the chance you are not seeing things clearly.

As an example, let's take a look at the fear I continue to experience before tweeting. I worry a lot that people will read what I write and think that I am stupid. Or that people whom I respect will unfollow me. Every tweet feels like it might put an end to a lot of things I spent a lot of time building and that I care about deeply. All this creates a lot of stress.

And then I take a moment to step back. I breathe in slowly and look at the situation.[28] What are the most likely outcomes if I publish the tweet? The most probable outcome is that very few people will notice. Even if quite a few people unfollow me, in the longer run, that probably will not make much of a difference.

But what is important to me is having a voice and helping others. I want to be better at communicating, which can come only with experience. Realizing this, I go ahead and press the tweet button.

Even with something as seemingly straightforward as tweeting, which I have done quite a bit of, I still need to talk myself through my emotions to start seeing things more as they are. But where this approach really shines is in opening yourself to new experiences.

I have never written a book. I have never shared so many of my thoughts so

openly. Do I have anything worthwhile to say? Will I be able to make my writing clear enough given that I am not a native English speaker? Will anyone buy the book at all? How do I pay taxes on any sales I might make?

The reality is that I don't have a clue, and that is okay. You cannot know any of these things in advance. Sure, you can read about the experience of others, which can be helpful, but in the end nothing changes you remotely as much as experiencing things yourself.

Transitioning into a new field, you enter a realm that operates in mysterious ways. Nothing in my prior life prepared me for what learning online or finding a programming job (let alone having one!) would be like.

But if you act and keep an open mind, soon some of the mystery fades away.

[27] Ruby on Rails is a web development framework created by David Heinemeier Hansson. GitHub, Twitter, Shopify, Instacart, and many other web applications were built using it. Hansson is a successful businessman, programmer, amateur race car driver and photographer. He often shares insights on learning and running a business.

[28] This approach is extremely powerful and there is an entire field of psychotherapy — rational emotive behavior therapy — based on it. The premise is straightforward: We hold erroneous beliefs about situations we are involved in, and when we dispute and change them, we become a tiny bit saner. Quite often, as we learn to navigate the real world better, we become significantly more effective.

Do genuine work (it compounds)

> "Reading a book without taking notes is like discovering a new territory and forgetting to draw a map."
>
> — Julian Shapiro

Genuine work moves the atoms in the universe.

Thinking about writing a blog post for a couple of days is just a tiny bit of work. Sitting down to write an outline is better. Completing a draft and sharing it with your friends is better still.

Genuine work compounds. You think about something, even if you do so very deeply, and the next day you wake up and nearly nothing remains. A couple of days of thinking like that is like running on a treadmill when you are attempting to move from point A to point B.

On the other hand, a small set of notes is something you can build on. You can add to them the next day or, based on the experience of writing them down, decide that you do not want to work on this any longer. You are in motion and making progress.

Feedback from the world is even better. It might come from your friends or like-minded people you met online. It allows you to see what works and what does not. You also become better at what you will probably want to do down the road — that is, sharing your work freely. There is little point in keeping genuine work to yourself.

This works for every type of activity: writing text, writing software, studying.

You can learn passively by watching a lecture or reading a book. This constitutes some learning and will compound very slowly. You can take notes while watching a lecture. Now we are starting to get somewhere. You can review your notes the next day. You move even faster.

If you add some exercises to the mix, your speed increases even more. If your exercises morph into mini-projects that you share with the world, you are moving very fast.

The more atoms you move the better the feedback you receive and the greater the extent to which you can reflect on what you are learning.

The hidden game of machine learning

At the core of all machine learning lies the ability to generalize to unseen data. What makes it possible to train our model on some amount of data and have it be useful in other contexts? To what extent can we trust the results? How do we create an environment in which we can safely run our model on unseen data and act on the results?

These are the questions that matter the most. Being able to answer them makes you an adept practitioner. Understanding what goes into coming up with the answer makes you a great one. Missing an answer to either question by just a tiny bit can lead to tragic consequences.

The best starting point for gaining a deeper understanding of the ability to generalize to unseen data is a blog post by Rachel Thomas, How (and why) to create a good validation set. The article is very practical, yet it covers many subtleties to a far greater extent than any of the other resources available on the Internet.

For venturing into the theory, I highly recommend Learning From Data: A short course by Yaser S. Abu-Mostafa. The lectures from the associated MOOC are also fantastic.

For deploying the models in the wild, no better resource covers all the bases than the fast.ai v4 part 1 lecture 3. A great book to read on this is Building Machine Learning Powered Applications: Going from Idea to Product by Emmanuel Ameisen.

How to structure a machine learning project

> "The only way to maintain your sanity in the long run is to be paranoid in the short run."
>
> — Radek Osmulski, How to do machine learning efficiently

A machine learning project is like a flower. It has the potential to spring from a small seed into a magnificent entity, assuming the conditions are right. But it cannot be willed into existence. Fail to supply one of the necessary ingredients and no matter how much effort you put into it, it will not grow.

The main condition for a healthy machine learning project is a good train—validation—test split. This consideration is so fundamental, I devote an entire chapter to it.

With the fertile ground of a good validation strategy in place, we are ready to construct our baseline. This is the seed from which our entire solution will grow.

What to aim for in a baseline? The smaller and simpler, the better. Essentially, we are looking for the simplest transformation of our train data that will move the needle on our metric as measured on the validation set. For example, let's imagine we would like to predict the price of a car based on its features, such as age, color, horsepower and so on. One thing we could do is to split the examples into two groups based on age. One group would contain cars built before the year 2010 and the other would contain cars built after that. We would then take the mean price of each of the groups and use it as our prediction. When we see a new example, we would look at which age group it falls into, and predict its price to be the mean price of that group. When compared to a prediction of all

zeros, this should lead to increased performance on our validation set.

Why is having a baseline so crucial? It provides us with an indication of what is possible. Further, in the process of constructing our baseline, we are likely to learn things about the problem we are solving that we haven't known before. It is also a great way to get our bearings on whether we are moving in the right direction. Should we train our first model — especially on more complex problems with unbalanced classes — without a baseline, it might be very hard to interpret the results.[29].

Additionally, a baseline is a good way of validating whether what we have put in place has a good chance of being bug free. Even a simple baseline will require a non-trivial amount of code. We need to read our examples from disk and assign labels to them, likely by deciphering a metadata file. We also need to implement a means of evaluating the results on the validation set. Given how simple the baseline is, we can have some idea of what the results should look like. If we get something that roughly makes sense, we are good to move onto the next phase after a quick look through our code. If we get a nonsensical result, we probably have a bug somewhere or we don't understand something important about the problem domain. Both of these issues become easier to address the less code we have written.

Machine learning code is notoriously hard to write. It is extremely easy to introduce subtle bugs. Our code might still run and we might get a decent result, but it won't be as good as it could otherwise be.[30] Further, it is really hard to determine whether the result we are getting is good if we don't have anything to compare it against. A baseline is a good way of addressing both of these concerns. This way of thinking is a very powerful tool one that will accompany us as we continue working on our project. With the baseline in place, we are ready to apply the same approach to growing our solution organically, bit by bit, as we continue validating our work every step of the way.

We next shine the light of our attention on a component of the pipeline that we feel might benefit most from tweaking. We implement the change and run our entire pipeline. If we see a substantial decrease in performance or if the pipeline doesn't complete, we likely have an issue that we need to address. We should have written only a few lines of code since the last time we ran our pipeline and, thus, the changes should be localized. This gives us a fighting chance of addressing the problem without much hassle.

However, it is important that we not run the experiments solely to tweak the hyperparameters, especially early in the training. We want to invest our time where it matters. That usually means exploring a larger set of architectures and developing diagnostic code. The more we learn about how our models are performing, the better. Additionally, trying out different architectures can be a great source of insight on what is possible.

As we build on our baseline, the idea is to keep moving in small steps. Just as we didn't want to go from having no pipeline to training a complex model, neither do we want to jump straight to it now. Before we train an elaborate, state-of-the-art deep learning model, we might want to try random forest[31] or a simple model consisting only of fully connected layers.

Extending this way of thinking to the implementation of our most complex model, we probably don't want to go about building it in one sweep. We might want to begin by implementing the loss and bolting it onto a simple model consisting of only fully connected layers. The next step might be to implement some set of layers and pass a batch from our dataloader through it. Ensuring we get something other than all zeros and an output of the correct shape is a very valuable check.[32]

The idea is to always move in small increments, using simpler models as a stepping stone to more complex ones.

Another consideration worth addressing is that growing our solution in this fashion will require running our entire pipeline regularly. Waiting for it to complete can be a horrible time sink. We also risk getting distracted in the process. As we work on growing our project, we do not care about precise measurements of performance. All we need is some indication of whether we might be moving in the right direction. Above all, we want to ensure that our code still runs with the changes we introduce. Constructing a smaller train set might be a great means of achieving both of these ends. Instead of training on the entire dataset, maybe just 1% or 5% of data would be enough. Assuming random sampling of our examples would be appropriate, switching to running on a subset of our data might require changing just a single line of code. With a bit of experimentation, this approach can usually be leveraged even for more complex problems. While we might not be able to tell whether our model has enough capacity to make use of all the information in our dataset, running our model on a subset of data, with appropriate diagnostic code, might still provide valuable insights into how it stacks up against other models that we have been working on.

The last condition necessary for growing a good machine learning solution is time. With enough experience, you will likely arrive at a decent solution quickly. But as you venture into the unknown and push the boundary of what can be done, you embark on a creative process. Just like learning, it cannot be crammed. You spend some time at the computer screen. You then take your dog for a walk or go jogging. You busy your mind preparing a meal. Through it all, your mind continues to work on the solution and analyze what to do next. This process can be particularly potent if we strengthen it by reading as much as we can about the problem domain. All this takes time and it is humanly impossible to accelerate this beyond a certain point. We have to leave time for things to click into place and for us to develop key insights.

Plant the seed of a good baseline in the fertile ground of a good validation

strategy, water it daily with inspiring reads, shine your attention on each component of the pipeline, and grow it systematically, and very soon you will have an intricate, well-performing machine learning solution. The key to a satisfactory outcome is rarely any particular architecture or method that you chose to adopt along the way. Rather, it is the process of arriving at the solution that can make all the difference.

[29] To learn more, consult this fastai lecture

[30] This is a strong argument for using frameworks such as the fastai library. I know I can implement my training loop manually. I love the freedom that doing so offers. But even more than that, I appreciate that the code in a library has thousands of people running it and a lot of eyes looking at it. Plus, it should be thoroughly tested. Making our code simpler by using high-level libraries is another great way of maintaining our sanity.

[31] Random Forest is a great algorithm to reach for, given the chance! It is very quick to train and it doesn't overfit. On many datasets, it can match the performance of more complex methods. It also lends itself extremely well to interpretation. For an example in which a Random Forest classifier matched the performance of a neural network, see this repository. Additionally, it enabled a deeper understanding of the factors at play, which would be very hard or impossible to get at using the RNN!

[32] This can be taken further, where we look at the means and standard deviations of intermittent layers, monitor for how many dead ReLus we have, and so on. In many instances, just a single check will suffice as it provides a great value for the time invested.

How to win at Kaggle

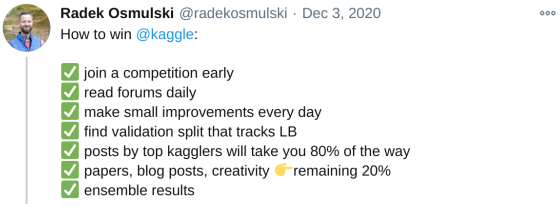


Figure 3. Steps needed to win at Kaggle summarized as a list.

The first step to winning at Kaggle is to join a competition early. Getting set up with a competition can take a while. Depending on your Internet connection, downloading the dataset can take a couple of days. Then comes the process of figuring out how to read in the data and feed it to our model. Depending on your level of experience and available hardware, this part alone can take a while.

More importantly, even once you start training your models, arriving at a good solution will still require time. This is true for any competition, even those with tabular data for which the training of a single model can take as little as a couple of minutes. Working on a machine learning problem is a process of learning about the problem domain and data and responding to feedback on what works and what doesn't. There is a great chance that you will not arrive at a great model from the get-go. With time, Kaggle competitions are becoming more demanding, and some can be very complex. Doing well in a competition, reaching the silver or gold medal level, will require running a lot of experiments. Writing diagnostic code and identifying where your model is struggling also requires time.

But the main reason for joining a competition early is that solving machine learning problems at a high level is a creative process. Cookie-cutter solutions can get you decent results. However, while they will usually be good enough in a business context, they will not get you very far on Kaggle. Going beyond them is where the fun and the challenge begin.

Human beings, unlike the 486 line of PCs, do not have a turbo button. Creativity takes time. You cannot force yourself to sit for eight hours and command yourself to be creative. Finding new ways of going about things needs to flow naturally. You read relevant research papers and you try some things in code. You might be hit with a bolt of insight while driving or taking a shower. If you read the write-ups of well-performing solutions that are posted after a competition finishes, you will find that they are extremely varied. This is a testament to the creativity required to do well. Most likely, to find yourself in the golden medal range, you might need quite a few moments of inspiration.

Here is where the Kaggle forums come into play. They are a resource that cannot be overrated. First, if you are struggling, they, combined with the notebooks that people share, will get you going. Also, having an opportunity to discuss what you are seeing and to compare with it to the perspectives of others is a great source of inspiration. Because of this dynamic, combined with the fact that people want to be genuinely helpful, make their mark on the community, and, thus, share very liberally, simply following the Kaggle forums will take you 80% of the way toward a very good solution. Not only is this a very effective strategy for doing well but it's also a great way to learn.

One thing to bear in mind, though, is that people, including top Kagglers, will share a lot, but they will not share everything. This is understandable. Often, they will hint at what works, and you might need to do some digging of your own. Because of the speed at which new information will come in, and the amount of follow-up you might need to do, reading the forums daily is the only way to keep up. It also gets you into a habit of thinking about the competition daily, and making small tweaks every day is the best way to improve.

If Kaggle forums are that useful, what are other sources of information you must leverage to win at Kaggle? First and foremost, research papers. There is a skill to reading them. You do not need to understand each paragraph. In fact,

attempting to do so while exploring a new area would be extremely time consuming. But it is important to get an idea of what a paper is talking about, what its essence is and whether the described technique has a chance of being useful in the problem you are solving. This is a skill that comes from scanning and reading a lot of papers. There is no magic to it. Some papers will be more approachable than others and that is also okay. If you have a hard time understanding a certain paper, it is best to read related papers first and then take a look at the initial paper only after a while. If you feel the paper is relevant to the competition, another thing you can do is to start a thread on the forums and ask people for their help in understanding it. Threads like this are great, as both the person asking the question and the people who help them can learn a lot.

Research papers have their younger siblings: the blog posts. Often, they are the superior place to start, though finding good blog posts on more arcane subjects can be tough. Nonetheless, they can still be great learning resources. The quality of blog posts varies greatly, but some are as accurate as the papers, and reading them can be a great way of building intuition on any given subject.

Now that we have a framework for doing research, we must work on our solution daily. The idea is to make small tweaks, and the effects will compound over time. Jumping to the top of the leaderboard might seem impossible at any given moment, but improving our solution just slightly, trying out a new method, is within our reach. This is where most of our energy should go. It is also very useful, especially early in the competition, to focus not on tweaking the hyperparameters too much but, rather, on trying new architectures. We want our exploration to be meaningful and cover as much ground possible. I distinctly recall participating in a competition that included tabular data and images. I rushed to create a model utilizing both types of data and ultimately finished fifth. I feel that I could have performed better if I had spent more time building more varied models. Post-competition, it turned out that there was more signal

in the tabular data than I assumed! Had I not jumped directly to deep learning models but, instead, spent some time trying out various architectures just on the tabular data, I would have learned much more about the problem domain. This would have been a better way of working on the problem and would likely have led to a better result.[33]

The idea is to run a lot of experiments, but how do we know we are moving in the right direction? To answer this, we need to go back to the beginning. In a competition, once we start writing code (which should be on the day we decide to join), our first order of business is to create a simple baseline. In the context of Kaggle, a baseline is usually a Jupyter notebook where we download the data, figure out how it's organized, and make a submission. The submission can consist of all zeros; it does not matter. It is useful, though, to have all the mechanics in place before we begin working on our model.

With everything prepared, we are ready to start training. The initial objective is to find a validation split that will track the public leaderboard. Ideally, the results you get locally will align with the results on Kaggle. However, for a large portion of competitions, this might not be attainable. It is enough for your local results to move in the same direction as the results on the leaderboard.

To achieve this, you must study how to best split your data. Is random sampling enough? Do you need to perform stratified splits based on classes? Or maybe the data is temporal and should be split on a timestamp. There might also be other considerations specific to a given competition that need to be accounted for. A proper validation split lies at the heart of any good Kaggle submission.

So, how do we measure whether we got it right? We train at least two models and submit them to the leaderboard. The results on the leaderboard should track the local results. If the first model does better than the second model, this should be reflected in both places. Two models is the bare minimum, though it

might be enough to get started. Still, as you work on the competition, you must keep an eye on whether the leaderboard is tracking local results. If not, chances are that you should revisit your validation split or something else worth investigating might be at play with the test data.

The final piece of the puzzle is ensembling. The idea behind it is very straightforward. We want to combine predictions from multiple models in the hopes that their errors are not correlated. If model A makes different errors than model B, some of them will cancel out each other. Using a diverse set of models helps.

A related technique is training with cross-validation. We train multiple models on our data, withholding different parts of it for validation and combining the results. Both techniques are extremely powerful, and it is hard to imagine a winning Kaggle solution that would not leverage both. Many great resources on both of these techniques are available online.[34]

And that's all there is. Join a competition early. Read the forums and work on your solution daily. You are guaranteed a win. The real prize, the one worth fighting for, is what you learn. Finishing on the top of a Kaggle leaderboard is a fun experience. The prizes are also nice. But in the longer run, taking satisfaction and financial aspects into account, what you learn is much more valuable. This is the ultimate prize, and if you follow the formula I outline above, you are guaranteed a win every time.

[33] Google AI Open Images - Visual Relationship Track is the competition I am referring to here.

[34] One of my favorite blog posts on ensembling is this one. I imagine quite a few top Kagglers began their journey with this blog post.

The best hardware for deep learning

Quite a few people out there love discussing hardware. In the process, they produce an unbelievable amount of content. For every such conversation, for every thread that you read, you pay with the most valuable currency available to you: your time.

You can put a new architecture to the test and blog about it, watch a lecture, or continue researching the perfect configuration for your home rig eking out 2% of additional performance and saving $20. Most likely, a single research session will not suffice. You will trade many lectures and many blog posts to evaluate the many alternatives.

As such, my advice is to explore hardware only to the extent that you find it entertaining to do so. Apart from this scenario, the reasoning is very simple. If you do not know whether deep learning is for you, seek out credits for AWS or GCP (I find GCP to be more user-friendly, especially for beginners) and use a cloud vm to find out. In the process, you will learn many useful things that will become extremely valuable in a professional setting — how to move code from your local machine to the vm, how to connect to it, how to access the Jupyter notebook server, etc. To keep things simpler initially, you might want to start with Google Collab or Kaggle Kernels.

If you know that deep learning is for you, a home rig is the most cost-effective option. My preferred setup is a laptop running Ubuntu Desktop and a headless desktop with a GPU running Ubuntu Server. I ssh into the desktop and access Jupyter notebooks on a local network. Another option would be to get a desktop with a GPU, install Ubuntu Desktop, and do the work directly on it. However, I find it more convenient to have two separate machines. Ubuntu Server seems to

be more stable than Ubuntu Desktop, and I also found it easier to install all the necessary CUDA and CUDNN libraries on it.

This brings us to the crux of the matter — what GPU should you get? The advice here is to get the biggest GPU[35] you can afford.[36] By biggest, I mean in terms of RAM.[37]

If new hardware is outside of what you can comfortably spend, a previous generation GPU or going back two generations is perfectly acceptable. It really doesn't matter what hardware you have. What matters is how you use it.

As you continue to work on deep learning projects, you will start to learn what is important. Maybe you are working on large tabular datasets and could benefit from a faster CPU or more PC RAM. Or maybe training computer vision models is your cup of tea and you are dying to train on higher resolution images. In this scenario, a multi-GPU setup might be the way to go. Or maybe you are a GAN artist and have a lot of results and model weights you want to curate? Adding another drive — one that is bigger but not necessarily fast — might be an option.

But you will not know any of this until you start shipping a project after a project. The first GPU rig is meant to get you to this stage. After a couple of projects, you will have learned enough useful things about hardware from the sheer act of keeping an eye on your resource usage[38] during training that picking the components for your next rig will become an afterthought. You will also know enough to avoid sweating the inconsequential minutia, such as how many PCIe lanes your CPU supports or what the clock speed is of your memory.

At the same time, if you find learning about hardware to be fun, as many techies do, by all means, go down that rabbit hole! But this is not a prerequisite. You can be an amazing deep learning practitioner and give hardware very little thought.

The valuable bit to learn is how to optimize the usage of the hardware that you

have. Are you loading the data that you are accessing over and over again into RAM or are you reading it from disk? Are you reading the data in sequence or is there a lot of jumping around? Are you using appropriate hardware for the use case?[39] If you need to do data preprocessing, are you offloading the heavy lifting and performing it once before training or are you incurring the cost every time you load an example? As you train, can you tell where your bottleneck is? Are you able to feed the GPU to bring it close to 100% utilization? Is there anything you could do to make the training run faster? These are the questions worth knowing an answer to.

Having a flashy car does not make you a good driver. And being a good driver does not require being a good car mechanic. But it does entail being able to push whatever car you may be driving to its limits.

[35] Just make sure it is supported by the deep learning library you are planning to use! At the time of writing, only NVIDIA GPUs qualify and even then a card may be too ancient or exotic to work without requiring a complex setup. If you are unsure, a quick online search will give you the answer. Anything that came out in the last several years is nearly guaranteed to work.

[36] If you really want to geek out on getting the ideal GPU for your budget, there is no better resource to consult than the blog of Tim Dettmers

[37] How long a training runs is not all that important, though the quicker you can run experiments, the better. Should your GPU have too little RAM, you might not be able to use certain models at all, or the training will become very cumbersome and inefficient. The approach I use now is to pick a GPU and then go onto YouTube to see whatever combination of motherboard and CPU people use with it. I grab the model combinations and head over to an online store that will assemble the PC for me.

[38] When running any computation, it is worth monitoring CPU utilization, RAM usage, GPU utilization, GPU RAM usage, and disk IO utilization. On Linux, I use `htop`, `nvidia-smi -l` (I prefer it vs another popular option, `nvidia-smi dmon`) and `iotop` (only when running disk IO heavy operations). To monitor disk space utilization, you can run `df -h` every now and then.

[39] HDDs offer cheaper storage but reading from them is slower. This is especially true if you are accessing many small files at random.

Debugging with ease is a superpower

Machine learning code is notoriously hard to write. Once a bug is introduced, it can be very hard to diagnose. This is especially true if we don't have proper tools at our disposal!

Wouldn't it be great to run a single command and be immediately transported to where an error occurred? Among other things, this could allow us to look at the variable values at the moment when the error was triggered. We could also attempt some modifications in place to see if that fixes the problem.

Fortunately for us, Jupyter Notebooks allow us to do all of the above, and more. In this tweet thread, I describe these life-saving capabilities and provide examples of their uses. And here is this information, with some additional comments, presented as a Jupyter notebook.

Time yourself

It is useful to keep track of how much time we devote to various activities. Often, we might be very surprised to find out where our time is going!

This is true with ML code as well. We deal with thousands or millions of examples, and even a small speed-up per example can translate into enormous time savings.

When working in Jupyter Notebooks, it is good to get into the habit of using the `%%timeit` cell magic. This can often lead to interesting insights.

Here is one such example.

```
%%timeit
df.target_fn[np.random.randint(df.target_fn.shape[0])]
```

```
40.8 µs ± 603 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

```
%%timeit
df.target_fn.sample(1)
```

```
79.5 ms ± 2.09 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Dropping down to using `numpy` instead of `pandas` offered a nearly 2000x performance increase!

Timing your code is a very useful technique across the board, but especially when you are constructing datasets and dataloaders. The only way to make full use of your GPU is to write efficient code!

You can’t learn a profession by studying a textbook

A blacksmith is a professional who mixes metals in various proportions, creating alloys and using them to create useful objects.

A quick online search revealed to me that blacksmiths use at least 20 types of hammers.[40] On top of that, they are encouraged to create their own, special-purpose hammers. Then come various power hammers that can be steam-powered, mechanical, or screw.

I imagine that the majority of blacksmiths use a subset of these hammers, with artisanal blacksmiths using a broader set of tools. There might also be blacksmiths who devise and create hammers for other blacksmiths to use! But the hammers are barely a start. We then have the anvils, grips, chisels, gouges, punches, and so on.

The tools are probably not even that important. There is likely a specific technique to using each. This is what counts. How do you hammer at a thing for several hours and not hurt yourself? I have no clue!

If blacksmithing were like deep learning, blacksmiths would give off the impression that learning metallurgy really well will immediately transform you into a great blacksmith! In fact, blacksmiths would carry the knowledge of metallurgy as a badge of honor and would look down on people who do not know it as well. However, I do not think that knowing the properties of different metals is all that useful when you are presented with a hammer and asked to smash at things in a useful way!

What makes you a good blacksmith are all the things a more senior colleague could explain to you in the course of a day. What should you look out for? Why is

it better to use this tool than that for this particular purpose? How do you progress from not knowing how to hold a hammer to building the muscle strength and finesse of the grip necessary to do the job well?

And, yes, I suspect that metallurgy can be important. It might be one of the ingredients that, if combined with extensive practice, will take you to the artisanal level. However, if all you ever know is metallurgy, if you channel all of your energy into learning it, I don't believe you stand a chance of ever getting to that level.

[40] 104 Blacksmithing Tools: The Ultimate Guide The types of hammers include top fuller, cheese fuller, metalsmithing hand hammer, cross peen, German square faced cross peen, Swedish cross peen, French cross peen, Czech hammer, ball peen, straight peen, diagonal-peen, chisel-peen, clipping, planishing, Japanese blacksmithing, rounding, filemaker, sledge, blacksmith flatter, and top butcher.

On finding a job

Employability is one of the hardest subjects to talk about. There is simply no straightforward answer.

The best approach is to learn how others have done it. The paths will be as diverse as the people who took them.

However, some themes are common to many of the stories. The world has changed. The resume has lost much of its power. And for people without the desired educational background[41] or names that sound "white"[42], it never had a lot of efficacy in the first place. There are better ways that can take you to more interesting places.

A better approach is to meet people who can make or influence hiring decisions where they hang out. That is on the forums, on social media, at conferences, and meetups or through the text of your blog posts.

There is only one name to this game and that is credibility. By showing your work, by helping others, you not only exert a positive impact on the world but also demonstrate that you can do something non-trivial. Something that has a high chance of being valuable in a professional setting. You are projecting an aura of expertise.

Additionally, with everything that you share, you are expanding the circle of people who know that you exist. It will not happen overnight, but sooner or later people will start reaching out to you about opportunities that you likely would not hear about otherwise and that the general public might not be made aware of.

Building your digital estate is a great strategy. However, a more active approach

can also be beneficial.

If you are actively looking for work, make sure to tell the world about it openly. Add this piece of information to your name on Twitter. Include it in your Twitter bio. Put it at the bottom of your blog posts. Reach out to your acquaintances via email.

The best way to get what you want is to reach out to the people who know you and who have the power to help. The goodwill of people is immesurable.

[41] Hiring practices are often more about establishing whether you have the desired pedigree than testing job-related skills. This tweet by the creator of homebrew is a good example of this phenomenon. In the interview, he was asked to complete a task that was unlike anything he is likely to do on the job. Programmers do not spend their time writing out information they can find on Wikipedia on the whiteboard. The skill to do so is acquired in a university setting. His inability to perform the task outweighed his undeniable programming expertise. The interview was not testing his job-related skills but, rather, whether he had acquired his skills in a particular setting.

[42] Here are two pieces of research on the bias inherent in the hiring process that considers just the names: Are Emily and Greg More Employable than Lakisha and Jamal? A Field Experiment on Labor Market Discrimination, New CSI research reveals high levels of job discrimination faced by ethnic minorities in Britain. This issue is well researched and many results have been made available online.

The deep learning party is on Twitter

Twitter is where the deep learning community hangs out. There, you will find the founders of the field, among them, Geoffrey Hinton, Yann LeCun, and Jürgen Schmidhuber. Many of the researchers whose papers you are likely to read are also there. Then come Kaggle grandmasters, professionals with a lot of experience, and people who might be earlier in their journey but who often have very interesting and valuable insights to share.

This is a great environment to be a part of. You get to listen to people who are doing amazing things. By reading their messages, you can get a glimpse into their world. Additionally, it is a great way of staying in touch with research and trends. The system is not perfect, but many of the most promising developments are likely to surface on Twitter and be brought to your attention.

How do you get started with Twitter? This is not an easy question to answer. Many aspects of Twitter are not obvious. The GUI is unintuitive and it might take quite a bit of time to learn how people interact.

The first thing to do after you sign up is to pick a couple of people to follow. Both Jeremy Howard and Rachel Thomas are great to include in your initial picks. Most researchers will tweet only about their work, but Jeremy and Rachel go out of their way to amplify voices that deserve to be heard.

At this point, you might be tempted to jump in and join the conversation. For the most part, tweet authors welcome comments and will be happy to answer your questions. In the worst-case scenario, your comment will be left unanswered.

You are now also in a position to communicate with your friends and colleagues. If you follow them, there is a good chance that they will follow you back and will

want to hear what you have to say.

The good news is that at this point, you are done. You are now using Twitter in a way that delivers the most value for the least amount of energy invested.

It might be good to stop here. Growing your following beyond your real-life social circle, while rewarding, can be very taxing. There is no space in this book for me to show you how to go about this, but should you choose to use Twitter more actively, let me at least share a couple of thoughts that hopefully can keep you safe.

Twitter, just like any other social media company, employs dozens of highly skilled engineers whose sole purpose is to keep you glued to the screen. The more time you spend on Twitter, the more ads you will see and the more money the company will make.

To reach this goal, Twitter utilizes state-of-the-art algorithms and massive amounts of data that it has collected on your preferences. It will not shy away from showing you content that will elicit an emotional response, and will do so without concern for whether the content is good for you. Many of the mechanisms it employs in the process have been designed specifically to target your psychological blind spots.

When we use social media, we are effectively pitting ourselves against supercomputers with unlimited access to data and we wage our well-being in the process. Doing so is a recipe for disaster. Let's take a look at what we can do to limit the harm.

My first recommendation is to disable the algorithmic timeline. I still remember what using Twitter felt like before I made this change. Each day, I would spend upwards of 45 minutes scrolling through a feed I genuinely found interesting. I would learn what this or that politician had to say on this or that matter that was

important to me. It also seemed that the most intriguing stories from all over the Internet magically converged on my timeline. I felt entertained and exhausted. I could scroll for hours and have nothing to show for the effort. I was not learning anything worth knowing in the long run, and being plugged into the 24-hour news cycle was demoralizing.

After I switched to the chronological timeline, much of the nuisance went away. There are days when I spend just five minutes on Twitter and I'm done. No longer am I not shown tweets from people whom I follow just because Twitter decided they weren't engaging enough. I regained a lot of mental space that I could give to more valuable pursuits. My attention is finite and I am grateful for every additional minute I get to spend with my kids or on activities such as writing this book — minutes that, earlier, I would waste fighting the never-ending onslaught of content.

My second recommendation is to disable notifications. Otherwise, Twitter will transform each vibration of your phone into a very addictive dopamine hit.[43]

It is also a good idea to make sure no Twitter tabs are open while you work. Each distraction will wear you down and significantly cut into your ability to deliver high quality work.

Understanding many of the tactics used by social media does not limit the impact they have on your emotions. Having observed how using Twitter affects me, I now go a step further. I no longer have Twitter on my phone and I don't know my Twitter password. This way, I cannot check Twitter when I am on the go. I also try to delay the first time I look at Twitter after I wake up, and I try to do as much of my creative work as possible before I do so. These certainly feel like tiny changes, but they make a world of a difference.

In summary, Twitter has been amazing to me. I have immensely changed my way of thinking by reading the thoughts of some of the most experienced and

interesting people I could imagine. This is invaluable.

But despite its many wonderful aspects, Twitter must be handled with care. Left unchecked, it will consume your life.

[43] The first two episodes (episode 1, episode 2) of the "Your Undivided Attention" podcast are great for learning more about what goes into making social media so addictive.

Share your work

> "We do not learn from experience... we learn from reflecting on experience."
>
> — John Dewey

The actions that you perform in public weave a net around you. Whenever you tweet, publish a blog post, or share a GitHub repository, you add a new segment that starts transmitting your credibility into the world. But the net also works in the other direction, allowing good things to travel back to you. Praise that can build your confidence. Exposure to new ideas. Maybe even invitations to collaborate that will put food on your table or a nicer car in your garage.

This effect is known as personal branding.[44] It is a force to be reckoned with. For many, including myself, it is a concept that is not easy to embrace, but its utility cannot be denied.

And then there are other reasons to share your work.

The moment you decide to make something public, you better understand what "done" will look like. You have a milestone to work toward. "I will be done with this phase of the project when I share this and this, and in order for me to comfortably do so, this and this needs to be completed." It's much easier and quicker to finish something when you have a vision of the end state.

Sharing also naturally forces you to reflect on your work. You organize your thoughts, and very often, you discover things you wouldn't have otherwise. You strengthen your personal network of ideas.

You learn to communicate better.

You increase the clarity of how you write and organize your code. The only way to maintain your sanity while writing deep learning code is to adopt practices that minimize the chance of introducing bugs. Writing cleaner, more concise code, is a cornerstone of this.

You become a better coder. Code is meant to be read by people. By writing with this intention in mind, you gravitate toward becoming a better developer without effort.

Last but not least, the *homo economicus* is long dead. He continues to have a life only in outdated economic textbooks. We evolved to live in tight groups and, for the most part, we are altruistic.[45] The man invented to give mathematical rigor to economics and justify capitalism, one who is governed by his own self-interest at all times, is not us, for the most part.[46] Only a psychopath experiences infinite greed in the face of scarce resources.

Cooperative inclinations are one of our strongest instincts. When we share our work and explain to others how something can be done, we are being helpful and we give into our true self. This can lead to increased satisfaction with our lives and a stronger sense of purpose. Now, who wouldn't want more of that?

[44] Personal branding can be very powerful. No one explains this concept better than Rachel Thomas in one of her blog posts, Making Peace with Personal Branding.

[45] See Explaining Altruistic Behavior in Humans by Gintis et al.

[46] For a fascinating treatment of this, juxtaposing economics against physics, rewriting economics in terms of evolution, see The Origin of Wealth: The Radical Remaking of Economics and What it Means for Business and Society by Eric D. Beinhocker. This is one of the best books I have ever read.

When to start sharing your work

The sooner, the better!

Being new to something gives you superpowers. You notice what is challenging. You learn to overcome it. This gives you an endless source of things to write about!

There are articles you literally are unable to write at a later time. Here is one such example from me.

When I was starting to use PyTorch, I encountered many things that, to me, seemed unintuitive. I figured out how all of it worked and communicated it in a language that was understandable to people with my level of experience. Writing the article allowed me to reflect on my experience and understand PyTorch at a deeper level. It helped me learn and retain information. On top of that, several people found the article useful and took interest in my work.

There is another important dimension to this process of creation and a strong argument for starting early.

With every piece that we produce, we hone our communication skills, which can be very valuable.

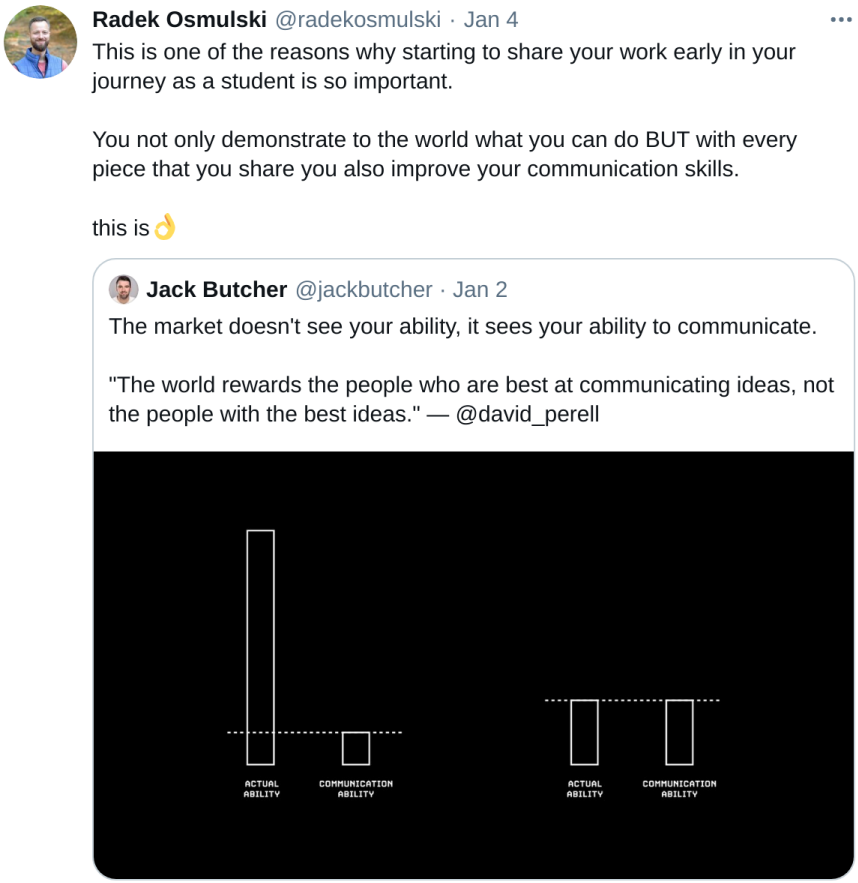


Figure 4. Me commenting on a very insightful tweet by Jack Butcher.

You are also likely to discover that there are very few, if any, negative consequences associated with sharing your work. None of the fears we often associate with doing things publicly ever manifest. The most likely negative outcome that we might face is people simply not caring about what we poured a lot of heart into. That is unpleasant, but a long shot from some of the negative consequences we might imagine!

I am scared to share my work! Help!

That's okay, we all are.

This is how the best marketers look at publishing:

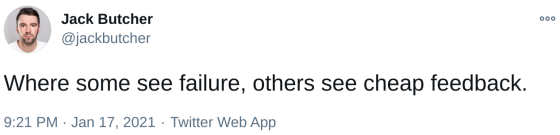


Figure 5. A tweet by Jack Butcher.

We don't have to be even remotely that good for what we are attempting.

But the same principles apply. The emphasis is on *cheap*. The feedback we get for whatever we share with the world costs us nothing. All the reputational costs we imagine that we are likely to suffer exist only in our heads.

If you post something that is not very good, or that people fail to notice, the outcome in 99.999% of cases is silence. It is unpleasant, but that's as far as consequences go.

Figure 6. Another very insightful tweet by Jack Butcher.

Sharing your work requires emotional energy. But whenever you feel low on it, or whenever you are not happy with what you shared, there is a solution. Simply delete it. No one will notice. No one will mind. This is something that people do, and they do a lot. I do it all the time.

Do whatever you need to do to make not only the actual costs of experimentation low but the emotional costs low as well. In the process, collect the benefits of telling the world about your work.

What to focus on in sharing your work

Speak to your experience. This is one of the most valuable things that we all have to offer. By showing people what you did and how, as well as how you felt along the way and what results you achieved, you draw a map that others can follow and demonstrate what is possible. This is immensely valuable.

You do not have to speak to topics outside your expertise. You can, but you do not have to. You do not need to venture into politics, or speak to what you had for lunch, or tell us about your kids.

You can focus all of your emotional energy on speaking to this one topic that is currently important to you professionally or that is something you are attempting to learn. It can be Deep Learning. It can be GANs or maybe recent advances in image segmentation. You can make it as narrow as you would like.

You can start from a very safe, impersonal space and go from there. You can share about yourself as much as you choose, but you do not have to share anything. In fact, especially in the beginning, and the more you speak to your experience, the more you speak to the results you obtain and nothing else, the easier and likely more effective your online journey will be.

Don’t lose sight of what is important

Recognition on social media can be intoxicating. The kudos you receive for doing well in a Kaggle competition might feel like you’ve made it.

But don’t ever lose sight of why you are going to all this effort. The main reason for any of this is to learn.

Social media is particularly tricky in this regard. You might see all these accounts out there, tweeting five times a day, with a bazillion followers. Is this something to aspire to?

Well, it might be if being a *content creator* is your thing. But using social media in this way is often a full-time job and requires a lot of specialized skills. More importantly, articles and tweets that might be great for learning will probably be very different from content geared toward increasing your follower count!

You can reap all the benefits of everything I describe in this book by having just a handful of followers and getting only a couple of likes — or not using social media at all! To convince yourself of this, please reread any chapter of this book and see if you need a large audience to apply any of the techniques I outline. If I did my job well, the answer should be a resounding no.

Social media is particularly dangerous. It throws everything it can at us to convince us that vanity metrics (likes, follower count) are all that matter.[47] It goes to extreme lengths to twist our view of the world around us.

When you share your content on social media, you are playing with fire, and you need to keep reminding yourself of how the world really operates. You do not need to make it big on social media to get your first Deep Learning job. You do

not need to have followers to take advantage of learning in public.

Figure 7. a tweet on a characteristic shared by nearly all of the best DL engineers I know.

Share your work, connect with the community, establish your credibility and above else learn in the process.

But do not chase meaningless bits in a computer — number of likes, number of followers — when, instead, you could be chasing your dreams.

[47] To achieve this, social media leverage UI elements found in casinos, in particular slot machines. You can learn more about this by listening to this episode of the Your undivided attention podcast.

Make mental space for what matters

> "Peace of mind is the most important prerequisite for creative work."
>
> — Richard Feynman

The decision to quit Facebook was one of the best I ever made.

Figure 8. Me making one of the best decisions ever.

You need mental space to make progress in any worthy endeavor. If you let engaging but junk-quality thoughts enter your mind, they will crowd out everything else.

Facebook is very dangerous because it features one of the most advanced algorithmically created feeds. It excels at delivering content that seems interesting to you at the moment but that leaves no positive long-term impact.

It also gives you a very limited ability to curate what you see. Will you unfriend your uncle who embraces conspiracy theories? Or your co-worker who just needs to share how cute their baby is and does so 10 times a day?

Twitter is better, as it gives you some ability to tailor your feed. Crucially, there is very little social stigma around unfollowing someone. You own what you see. Regularly pruning whom you follow and removing anyone who doesn't add value to your day is not a nice-to-have, it's a must.

But even that might not be enough. Social media causes excitement, worry about what others will think, and endless decision-making. Should I react to this post by a colleague of mine? What do I tell this person who just commented on what I shared yesterday?

All this wears you down. It depletes the energy you could give to what is important to you. Your mind turns into mush and you lose the ability to think long-term.

I learn a lot through Twitter, and it provides undeniable value to me. But I want to minimize the price I pay for this.

One strategy is to delay when you look at Twitter for the first time in a day.

I wake up with a fresh mind. I do my most creative work first. Only then do I use Twitter. I have now lost some of my creativity and emotional energy. I get to derive value from being on Twitter and I have minimized the damage. But even past this point, I need to make sure I am not on Twitter all the time. Maybe I will check Twitter five times a day. On a really good day, I get closer to one.

And then, during the night, my mind resets and I am ready to start another day.

To engage afterburners, find a mentor

A good mentor is someone very good at something you care about. They also need to be willing to explain to you, in detail, how to do said thing, using a language that will be easy for you to follow.

The mentor doesn't need to know you or even be aware that you exist. Having a direct relationship with your mentor is great, but often that would limit you to the people in your immediate circle of friends or professional acquaintances. It might be more beneficial to seek someone who is doing something at a world-class level and who is willing to share their knowledge with you. Such a person, depending on what you would like to learn, might not be living on the same continent as you. Or that person might have passed away a long time ago.

In general, we want to balance two competing aspects. On the one hand, it is good when your mentor is like you in some important ways. This will make it easier for you to understand what your mentor has to say. They will also be able to speak to your circumstances much better if they were once on a similar path.

On the other hand, you want to open yourself to new ideas. The greater our repertoire of ideas, the more effectively we can deal with any given situation. Listening to people who are not like us is the fastest route toward this.

One thing to be cognizant of is that the more successful a person is, the more people want to talk to them. Many successful people want to be of help. But they also want to live normal lives, spend time with their families and to do day-to-day things like sleeping, eating, and going for walks. They wouldn't be able to do any of that if they engaged in conversations with every person who reached out to them!

Thankfully, that is not the problem. Many potential mentors go out of their way to create materials that take a decent amount of time to create, but that can be consumed by however large a number of people without burdening the author. These may be courses, books, Twitter threads or interviews. This is how you can best engage with your mentor.

This is not to say that it is not okay to reach out to people and ask them for help or advice. But do go to extreme lengths to make your message as concise and clear as you can and to reduce the burden on the person you are reaching out to.

Additionally, if you do interact with your mentor, it might be good to do so via forums or some other public medium. AMAs work well for this purpose. This way, others will also have a chance to benefit from the answers you receive.

In summary, mentors may communicate via books or lectures, GitHub issues, or tweets. All it takes is for us to do the work, to look through the material pertinent to our situation. We live in an era when information can travel in milliseconds across the globe, and when more and more people are free to pursue their passions (which might be teaching others), and this is a very good time for finding a mentor who is world-class at what they do, even if the only way they learn about your existence is from a thank-you note that you write them. Or not even then.

The biggest regret of fast.ai students

What is the regret that one hears most often from fast.ai students when they complete the course? "I wished I spent more time coding and experimenting and less time studying in a more traditional sense."

I discuss the considerations surrounding this in the "From not being able to program to deep learning expert" chapter. But given how important this is, it deserves another look from a different perspective. Consider my tweet below,

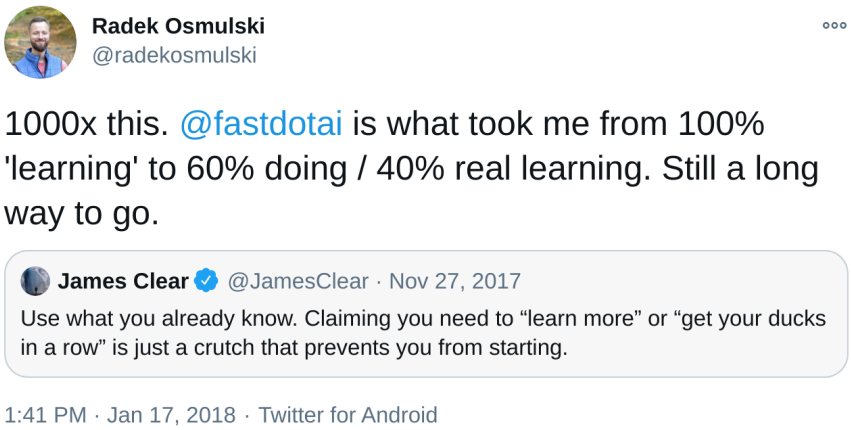


Figure 9. Something I tweeted while taking one of the fast.ai courses.

The observation by James Clear on what might be preventing us from practicing more is priceless!

And do note the language I used — 60% doing vs 40% 'real learning'! By the time I wrote this, I had already made some progress, but I was still very much stuck in the old paradigm that the school system had drilled into me. The language I used provided insight into how I was thinking.

I still love to learn and understandng how things work. Satisfying my curiosity is a driving force behind many of the things I do. The world is exciting and I want

to get to know it better!

But I now know that the shortest path to understand how something works leads through practice. Without it, you might be building castles of sand. Only when you act do you get to put your understanding to the test. Also, a setup in which you can receive direct feedback on your actions makes for a wonderful learning environment!

How would I rephrase my tweet today? To learn something new, I would aim to spend 80% of my time doing and 20% reading theory. Both activities constitute real learning!

Persistence is everything

In his wonderful book (co-authored with Sylvain Gugger) Deep Learning for Coders with Fastai and PyTorch: AI Applications Without a PhD, Jeremy Howard writes:

> There is a lesson here for all of us! On your deep learning journey you will face many obstacles, both technical, and (even more difficult) posed by people around you who don't believe you'll be successful. There's one *guaranteed* way to fail, and that's to stop trying. We've seen that the only consistent trait amongst every fast.ai student that's gone on to be a world-class practitioner is that they are all very tenacious.

Howard has made similar statements in multiple interviews, among others, here.

I am in my eighth year of this journey. I won a Kaggle competition and I am writing a book about machine learning. But most of the time, I feel like a beginner. The code I write has bugs. I imperfectly execute on the tactics I describe in this book. I have probably said more stupid things over these eight years than things worth remembering.

But I continue to get better results. I don't know a person who would be more surprised by this than me!

I look around at the people I met along the way. They come from various backgrounds, had various starting points. My assumption is that if you came to know them better, or came to know me better, you would be surprised by how ordinary we are. When it comes to me, ordinary is not even a good word. Fallible, misguided, caring about things that have no relevance in the longer run — all these would work better to describe me.

And yet here we are, where you are reading the words that I wrote. Where I finally have a job that I am very happy about in an industry well known for how well it pays.

I can’t help but agree with Jeremy Howard. He met way more people than I have and has a sharper mind. The only apparent distinguishing factor is how persistent one is.

If you have a good map of the terrain, which I hope this book can provide, you can arrive at interesting outcomes much sooner than I did. Nonetheless, one thing is certain. At any given moment, as you put in the work, you can barely notice a difference in your life. But the longer you stay the course, the more rewarding the journey becomes. Learning compounds and you need to give it time before you start seeing the exponential results.

Combine persistence with community involvement and you cannot be stopped.

Change is about what not to do

We are creatures of habit.

To adopt better ways of doing things, the challenge lies not only in taking on the new approach but, even more so, in ceasing how we have done things up to now. The latter is often harder than the former.

Or, to put it in other words, to welcome the new, we need to say goodbye to the old.

How does this translate to learning? Let's imagine we would like to focus more on practice and less on learning theory. If, up until now, you have spent an hour a day reading textbooks, and the rest of your time watching TV, the solution is not to add more to your day. You have only a limited reservoir of energy and time. You need to first take things out of your day so that you can replace them with something new.

It is so hard to leave the tracks we are on, so to turn toward a new destination, it helps to make a list of what you will not do anymore. Just the act of realizing what behaviors are no longer helpful to pursuing your goals can be extremely valuable.

I accidentally stumbled across this idea while taking one of the fast.ai courses. Focusing on what not to do worked really well for me. You can read more about what I did here.

Here is essentially the same concept but discussed from the perspective of another fast.ai student, Sanyam Bhutani.

Learning might just be enough

In the 6 years from when I decided to leave my corporate career to the time when I got my first freelance assignment, my strategy consisted of 100% learning.

There are many other activities I could have focused on. Networking. Blogging. However, when I was starting out, all of them seemed scary and unreasonable.

Along my learning journey, I got to know a person or two. I also wrote a couple of posts. However, the emphasis has always been on learning. It just so happened that, in the process, a few people learned about what I was working on or that writing a post ended up being a great way to learn.

What I am trying to say is this: Even if you focus solely on learning, whatever learning means to you, there is a great chance you will be alright.

A good strategy toward learning, one that I have followed very imperfectly, is this:

- observe whether you are getting the results that you are after
- if you are not, change your approach

Investing your energy into this approach might be the only thing you need to do to be okay in the longer run, regardless of anything else you might or might not do.

More perspectives on mentoring

Being mentored, learning directly from people who are much more experienced, being taught by them, receiving direct help — all lie at the core of a successful, fulfilling career.

There is no doubt that the above statement is true. We can do our best, but without people who have a strong network, skills, or capital extending their hand to us, it will be much harder to capitalize on our efforts.

In this mini-chapter, I would like to share three perspectives on mentoring that are extremely valuable. In situations in which you play the role of a student, they can show you an effective way forward. Where you are the teacher, they can show you what positive impact on the lives of others is within your reach.

I share one perspective on mentoring in the "To engage afterburners, find a mentor" chapter. Nonetheless, this is a nuanced issue in which listening to additional voices is to our advantage. Here they are.

When you are looking for a mentor, one way to get started is the Permissionless Apprenticeship. You receive value by giving value first. This might seem like a groundbreaking truth, but it lies at the core of every relationship.

Is there another way to attract the attention of like-minded people? Can you open yourself up to being taught by the best in the industry without asking them directly? Yes, you can. You can read about this idea, codified under the name Learn in Public, here.

Just how powerful is this idea?

Figure 10. Jeremy Howard on the ideas presented in the Learn in Public blog post.

Now, how would a relationship as outlined above unfold in practice? Here is one possibility.

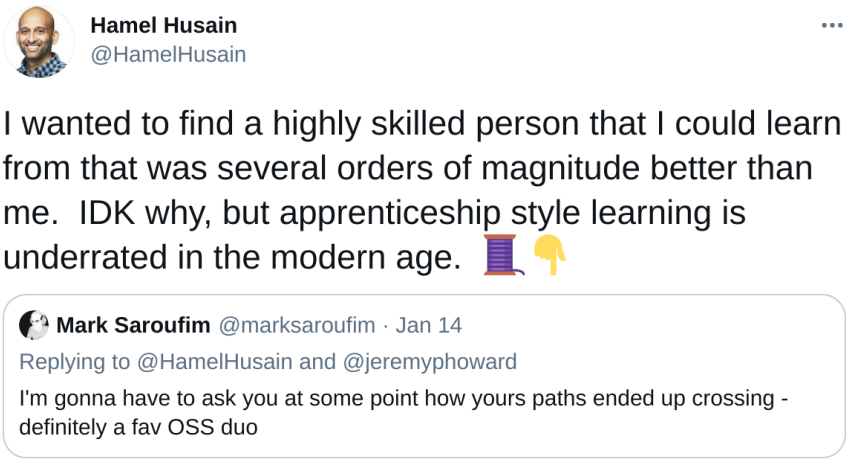


Figure 11. Hamel Husain sharing his experience with receiving mentoring through open source.

And here is probably the best, most succinct description of the repertoire of actions available to a mentor:

Figure 12. What a dedicated mentor can do by Chris Albon.

In the end, it all boils down to one person lifting up another.

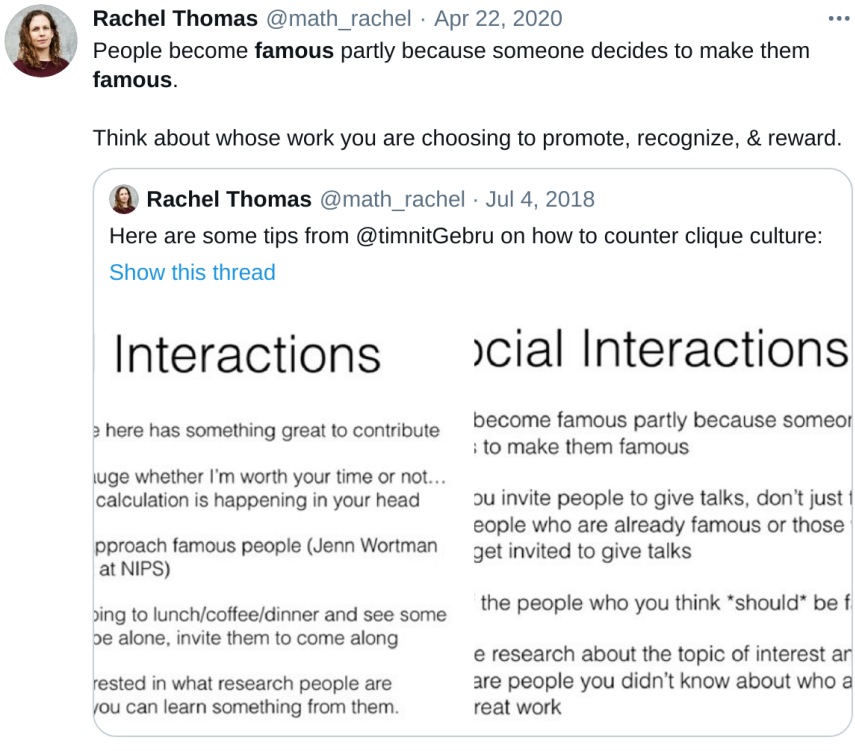


Figure 13. People become famous partly because someone decides to make them famous by Rachel Thomas.

Tap into the power of the community to learn faster

We learn the fastest when we are part of a community of learners. We observe how others do things and pick up on subtle but game-changing details we would never learn from a book. Instead of having to reinvent the wheel on everything that we do, we adapt blueprints fine tuned by many other learners that came before us.

There is pure magic to observing someone who is just like you and who does the things you want to be able to do. You become inspired. You no longer have to grasp in the dark for ways to approach things; instead you can adapt strategies that demonstrably work.

And the best part is that we get all of these benefits with minimum amount of efforts. We evolved to live as part of a community. It is the natural environment for us. Sure, online communities do not mimic the offline groups we are accustomed to exactly. Depending on where you live, there might also be cultural differences at play. But for the most part, being part of a welcoming group is motivating and fun. Many of the friendly online communities will welcome you regardless of how active you are. You can share as much as you are comfortable with or you can just hang around and read (though generally the more active you are, the more value you will derive).

You hang around a friendly group of people and absorb many invaluable things without effort. What's there not to love?

Now, it is true that some online communities are toxic. But this is not true for the machine learning communities I have come in contact with. In that regard, as machine learning practitioners, I feel we are very lucky.

On a personal note, I am an introverted person who would always stress out about talking to people. I had never anticipated that I could benefit from being part of a group. When I was starting to learn to program and do machine learning, my idea was to solely focus on acquiring knowledge and to not enter into any interactions. And then I started asking questions on the fast.ai forums. I slowly began to open up and talk with people who were very much like me, who were learning the same things I was learning and encountering the same obstacles. With time, I started to see the value of these exchanges and my approach changed.

Despite my current appreciation of the role of the community, it hasn't always been like that. And by all means, I am not an expert on this subject. And so I decided to reach out to community leaders, people whom I admire, to share their insights that we can all learn from. Here is what they told me.

Energize

In this chapter, I share a part of my story that gave me the energy to write this book.

However, before we dive in, I would like to address two things. First, I am not a medical professional. While what I write is grounded in personal experience and research, none of it is medical advice. If you plan on making significant lifestyle changes, please consult your doctor.

Second, neither the specific circumstances nor the solution I outline is very important. However, the overarching theme is. While your situation might be vastly different from mine, it is quite likely that you could nonetheless benefit from learning how I formed a new fitness habit and became more conscious of what I ate.

The most empowering experience in my deep learning journey has been the discovery of fast.ai courses. What I outline below has the potential to match it.

The good old days

Not too long ago, I had all the energy in the world. I studied during my commute and put in a full day's work of coding. I then returned home, ate dinner, and spent time with my family. Once everyone in my household was fast asleep, I grabbed a bottle of beer or a glass of whiskey, and the evening study session began. I loved every second of this.

Writing these words, I feel like I am describing someone else, not me. I wouldn't want to work that much anymore. But even if I did, I wouldn't be able to do it.

As months passed, I began experiencing more stress. I started sleeping poorly

and had to drink more coffee. After nearly 30 years spent in front of the computer screen, my back started to hurt. Soon enough, I suffered a serious back injury when lifting something very heavy.

I felt miserable and was in pain throughout the day. It took several weeks to recover. Even then, I had a hard time sitting for longer than an hour. This lasted for months. On some days, I could barely force myself to do two hours of work. My thinking was sluggish and everything felt out of focus. All of my energy went into getting by.

Turning the tide

I tried a couple of things but neither worked too well. The first thing I did was limit the extent to which I was hurting my body. I got a standing desk and a sitting ball and I lifted my monitor.[48] However, while I stopped causing myself too much harm, I still felt drained.

I tried several activities, such as running, swimming, yoga classes, and training with a personal trainer. The effects were minimal. In hindsight, I suspect that part of the issue was that it was impossible to make meaningful progress with one or two training sessions a week.

The big turn came as I reached for something different. I decided to do yoga each day, Monday through Friday, at home. I reserved the weekends for recovery and outdoor activities with my family. I also felt ready to try a completely new mental approach.

Forming a habit

I started each day by prioritizing yoga. Everything else came second. Did I have some catching up to do at work? Yoga came first. Was I taking care of the kids for a couple of days on my own while my wife traveled for training? So long as the

kids were fed and happy, the first few minutes went to yoga.

It wasn't easy to convince myself to take this approach. However, I was on the ropes and willing to try anything.

I employed another key component: I stopped pushing myself. I made it as easy as possible to avoid failing. On some days, I practiced for five minutes — and that was okay! On other days, I went for 10 minutes of practice. Yet those days led to weeks like the current one. The shortest session was 20 minutes but a 45-minute session didn't feel like much of a burden.

Something amazing started to happen. Moving your body daily seems to have the same effect as studying daily. And, it turns out that if you make it easy for yourself to succeed, you will.

Today I completed my 16th week of not missing a single day. I now sleep better and no longer experience back pain. Making this practice the core of my day has changed my life. My optimism, clarity of thought, and energy levels have slowly returned. However, I recently came across yet another practice that gave me an unbelievable, direct boost across these three dimensions.

What and when you eat matters

Having more energy is nice. Not suffering from back pain is nice. But a tiny part of me still treasures mental acuity above all else.

If somebody approached me today and asked, "Radek, what is the best way to complete the highest amount of mental work over the next 48 hours?", my answer would be simple: "Stop eating."

We didn't evolve to digest something throughout all our waking hours. There were no kiosks with snacks in the forest. There was no fridge a couple of steps

away from where we slept.

Our bodies are extremely complex, self-regulating mechanisms. However, if we don't play by the rules, we put a wrench in the works.

A lot of recent science supports my statements, and the results are fascinating. While I don't have room in this book to cover it all, I will describe the approach that has been working for me and will also refer you to resources where you can learn more.

What works for me

I wake up and drink water or coffee. I eat only when I feel hungry, which sometimes doesn't happen until noon or 1 pm. Either way, I do not eat my first meal before 10 am and I try not to eat after 6 pm.

This is known as time-restricted eating and its primary benefit is to restore hormonal balance. It allows your body to rest from digesting and to stop pumping insulin into your blood.

The second tiny change I made was to limit my sugar intake. Sugar produces rapid insulin response; unfortunately, research demonstrates that artificial sweeteners might be even worse in this regard. Fructose is another compound to look out for. It bypasses your hormones altogether and goes directly to the liver, where it is stored as fat. Beyond that, I try to avoid processed carbs, mainly white flour. That's it.

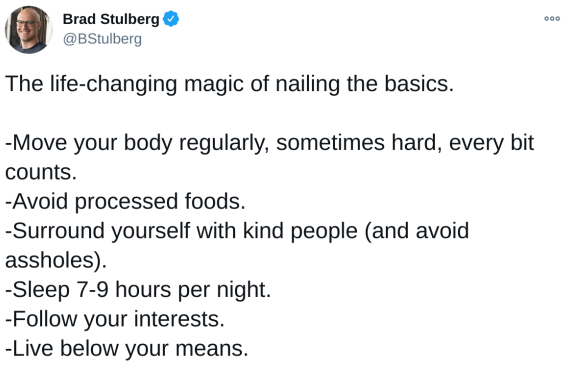
For the most part, I follow these rules but certainly not always. I have a long drive coming up this Saturday and I am quite certain I will eat at McDonald's. When watching a movie with my family at 9 pm, I will indulge in whatever they might be snacking on. Sticking to the rules most of the time is enough.

Whenever I eat less, I feel ...high. I can focus on work longer and I don't feel as tired at the end of the day.

Where to go from here

Two books that my doctor recommended to me are the fast 800 by Dr. Michael Mosley and The Obesity Code by Dr. Jason Fung.

The science behind nutrition is fascinating, but I still see myself as a fairly down-to-earth person. To this end, I would like to leave you with a tweet by a great performance coach, Brad Stulberg.



Namaste!

[48] Essential Work-From-Home Advice: Cheap and Easy Ergonomic Setups is an article by Rachel Thomas detailing what a great ergonomic setup might look like, regardless of your budget.

Thank you

First of all, thank you to you, dear Reader. It is because of you that I get to explore one of my dreams.

Above all, I would like to thank my greatest teacher, Jeremy Howard. Nearly everything practical I know about machine learning, I learned from him. Additionally, he inspired me to study how we learn. He encouraged me to write and share my work. Without concern for his own reputation, he shared my various pieces, which often weren't that good. The interest in my work that this generated was instrumental to me staying the course. His support has been life-changing. It is impossible to enumerate the many ways he has helped and inspired me.

I would also like to thank Rachel Thomas, who shared key insights with me through her extremely instructive blog posts and tweets and without whom I wouldn't have learned about fast.ai in the first place.

My gratitude also goes out to my parents. They instilled in me the belief that learning, even in the inferior form of formal education, is important. My father, who has passed away, had an entrepreneurial spirit. Likely, he shared some of his inclinations with me, for which I am grateful.

Last but not least, I would like to thank my family. My wife continues to breathe love into our small group and creates the nourishing environment where writing a book like this is possible. My kids make writing this book, and many other things, worthwhile.