Meta learning how to learn deep learning and thrive in the digital world notes

How to read this book

1. Read in sequence
2. Return and review chapters that interest you

Talk about ideas with others
Write about ideas that you agree or disagree with

Pick an area of interest and focus on that area of interest for a while

From not being able to program to deep learning expert

Starting over

- Optimize for FUN
- Tinkering mindset
- Become comfortable working with the computer

Consistency beats intensity over time
Know what to focus on as indicated by this book

Step 1.
	Complete one of the programming MOOCs
I believe I am all set with this because I have experience coding
I have enough familiarity with coding concepts

Step 2.
	Assembling our ability to move effectively in code
Can you use the code editor very well?

- In my case I am working with Visual Studios Code
- Learn it well because it is critical to being a good developer

Step 3.
	Version control

Code is written in small iterations - often by more than one person
Code is expensive - do you remember spending hours on tiny bits of code?
Git is the small command line program that will change any directory in to a git repository

- Essentially gives the directory superpowers
  - Ability to go back in time
  - Create alternate timelines (in code)

Git = manage code in computer
Github = code sharing + collaboration

- Pushing code to github = backup
- If PC dies, Github will have code

Apparently Github has its own command line tool: gh cli

- This is called gh cli

Step 4. (final)

How to use the computer in the context of writing code?

In the case of deep learning, how do we make the cloud VM? How do we ssh into it? How do we move data around? How do we build our own home rig?

- Lucky for us, we don't have to make all of this at once

Learning how to use the computer

- https://missing.csail.mit.edu/

How to use fast.ai to learn

1. Watch the lecture
2. Open lecture notebook and figure out how parts of each line of code work
   a. Run each line of code and look at the outputs
      i. If function not familiar - read documentation on it
   b. How will code differ in function when different inputs or arguments are tweaked?
3. Reproduce results
   a. Open new notebook
   b. Recreate the training pipeline
4. Once you have running results, search for a similar dataset
5. Test drive the techniques that we learned
   a. Find some online dataset or put one together yourself
      i. Fastai has a lot of data sets
   b. Download them to your machine

Main idea: systematically grow your skills

- Start in the controlled setting like the lecture notebook
- Learn the ins and outs of techniques in the lecture
- Little by little, move towards implementing things yourself from scratch

Main goal: Be able to apply the techniques you learn in unfamiliar situations

Start coming up with mini projects as you work through the tutorials

Theory vs practice

Theory follows practice

One cup of theory and one cup of practice

Practice wins every time

Programming is about what you have to say

Your ability as a developer = utility of things you can express in a language

Learn programming languages by reading and writing

- Read code that tends to be 100 - 1000 lines long
- Write to your heart's content

Learn how to say something useful first

The secret of good developers

1. Check where a function you want to call is defined and what it does
   a. What tests do you have for the part of the codebase that you are working on?
   b. Read what is tested and how things fit together

Progress on learning or writing = long, uninterrupted sessions

- Deep work
- 1 hour to two and a half hours

The best way to improve as a developer

Read smaller maintained repositories on github
Read and write a lot of code

How to use your tools to achieve a state of flow

State of flow = immersed in your work

- Nothing separates you and your work
- Nothing else in your awareness

Obstacles of flow

1. Reaching away from your keyboard
    a. Introduction of a delay between intention and the results appearing ona the screen
2. Disconnected from our VM
3. Troubleshooting subtle bug
4. Wireless keyboard needs recharging

Use reality as your mirror

Do genuine work (it compounds)

The hidden game of machine learning

Core of machine learning = ability to generalize to unseen data

Most important questions

- What allows us to train our model on some amount of data and have it be useful for some other data?
- To what extent can we trust our results?
- How do we create an environment in which we can safely run our model on unseen data + act on results

Answer = adept practitioner
Understand what goes into coming up with answer = great practitioner

Best starting point for gaining a deeper understanding of ability to generalize to unseen data

- Blog post by Rachel Thomas
- How (and why) to create a good validation set
- https://www.fast.ai/posts/2017-11-13-validation-sets.html

Theory for understanding the ability to generalize to unseen data

- Learning From data: a short course
- https://www.amazon.com/Learning-Data-Yaser-S-Abu-Mostafa/dp/1600490069

Resource for deploying models in the wild

- Fast.ai v4 part 1 lecture 3
    - https://www.youtube.com/watch?v=5L3Ao5KuCC4&feature=youtu.be
- Building machine learning powered applications: going form Idea to product
    - https://www.amazon.com/Building-Machine-Learning-Powered-Applications/dp/149204511X

How to structure a machine learning project

Machine learning ~ flower
- Can spring from small seed to great entity if the conditions are right
- Cannot be willed into existence

Main condition for healthy machine learning project = good train-validation-test split

What is the aim for a baseline? - smaller + simpler = better

- Simplest transformation of our training data that will move the needle on our metric as measured on the validation set

If we predict care price based on features
- Age, color, horsepower, so on
- Split the examples into two groups based on age
    - On group = cars before 2010
    - One group = cars after 2010
- Mean price of each group and use it as our prediction
    - Whenwe see new example, look at which group it falls into and predict its price to be the mean price of that group
- If we compare this prediction to a prediction of all zeros, this should increase the performance on our validation set

        - Makes sense to compare to a relatively close average than to some outlier like 0 that is also arbitrary

Why is the baseline so critical?

- The baseline provides an indication of what is POSSIBLE
- By constructing our baseline, we are likely to learn things about our problem that we are solving that we had not considered or known before.
- It is a good way to get a bearing of whether we are moving in the right direction

Should we train the first model - especially on more complex problems with unbalanced classes - without a baseline, it might be very hard to interpret the results

Baseline = good way of validating whether what we have put into the place has a good chance of being bug free

Simple baseline requires a non-trivial amount of code

1. Read the examples from disk and assign labels to them
    a. Decipher a metadata file
2. Implement a means of evaluating the results on the validation set
    a. Given how simple the baseline is, we might have some idea of what the results should look like
3. If the results make sense, then we should be good to move to the next phase, after we should look through out code
    a. Else, we should figure out where the bug lies or if we don't actually understand something that is important about the problem domain

ML code = hard to write

- Easy to introduce subtle bugs
- Code might run and we might get a decent result, but we might not have the most optimal result
- Really hard to determining if we are getting good results if we don't have something to compare it with
    - Solution to this is the baseline

Baseline = good way of addressing concern about not getting optimal results and concern about difficulty of getting good results if we don't have something to compare it with

Baseline in place = ready to apply same approach to growing our solution, organically, bit by bit, as we continue validating sour work every step of the way

Our attention now arrives at the component of the pipeline that we feel might benefit the most from tweaking

- Implement a change and run our entire pipeline
- If we see substantial decrease in performance, or if pipeline doesn't complete, we probably have an issue that we need to address
- We should have only written a few lines of code since we last ran our entire pipeline
    - The changes should be localized
    - This is good because we actually should have a chance or debugging without too much of a hassle

Important: don't run the experiments solely to tweak the hyperparameters, especially early in the training

- Invest time where it matters the most

Where does time investment matter the most then?

- Usually exploring a larger set of architectures
- Developing diagnostic code

More we learn about how our models are performing, the better

Trying different architectures = great source of insight onto what is possible

As we build on the baseline, keep moving in small steps

- Remember that we didn't jump from having no pipeline to having one that works

Before training an elaborate state of the art deep learning model, see how the performance is on a random forest or a simple model with only fully connected layers.

- This will prevent us from implementing the most complex models from the start
    - Maybe begin by implementing the loss and bolting it into a simple model with only fully connected layers
    - Next step might be to implement some sets of layers and pass a batch from our dataloader through it
    - Ensuring that we don't get only zeros as an output and the correct shape is very valuable
        - Remember we are working with massive tensors or matrices like objects

Idea: always move in small increments, using smaller and simpler models to begin as stepping stones towards more complex ones

Another consideration: Problems with growing our solution in a fashion described above:

- requires running our entire pipeline regularly
- Waiting for completion of the pipeline = incredible time sink
- Risks getting us distracted in the process

We do not care about the precise measurements of performance as we work on growing the project

- What we need is some indication that we are moving in the right direction
- We want to ensure that our code still runs with the changes we introduce

How do we achieve the above two points?

- Constructing a smaller training set might be a great means of achieving both.
- Don't train on the entire dataset, maybe just 1% or 5% of the data might be enough

Assume that the random sampling of the examples will be appropriate (remember statistics?), switching to running on a subset of data might require changing a single line of code

- More experimentation might allow us to leverage smaller samples for more complex problems

By using smaller samples, we might not be able to tell if our model has enough capacity to make use of all the information in the dataset, but running our model on a subset of data (the smaller samples) might still provide insights into how our model might stack up against other models we have been working on (with the right diagnostic code of course)

Last consideration necessary for machine learning growth: Time

With enough experience, you will likely arrive at a decent solution quickly

- You cannot venture into the unknown and push the boundary of what can be done without embarking on the creative journey

You cannot cram the development of good machine learning solutions

We can strengthen the process by reading as much as we can about the problem domain
We need to leave time, however, for things to click into place

Shine attention on each component of the pipeline, grow it systematically, soon you will have an intricate, well-performing machine learning solution

The key to a satisfactory outcome is rarely a particular architecture or method that you choose to adopt along the way: it is the process of arriving at the solution that can make all the difference

How to win at Kaggle

Step 1: join a competition early

- Getting set up with a competition can take a while
- Depending on internet connection, downloading a dataset can take a couple of DAYS
- Then you have to figure out how to read in the data and feed it to the model
    - This depends on your level of experience and your hardware
- Once you start training your models
    - Arriving at a good solution takes TIME

Working on a machine learning problem = process of learning about the problem domain and data + responding to feedback on what works and what doesn't

- Good chance you don't arrive at a good model from the start

Kaggle competitions are becoming more demanding + complex

- Will require a lot of experiments
- Writing diagnostic code + identifying where the model struggles requires time

Main reason for joining a competition early = solving a machine learning process at a high level = creative process

- Cookie cutter solutions only get you so far
    - Good enough for business, not enough for high ranking on Kaggle

Creativity takes TIME

Find new ways of going about things requires time

- Reading relevant research papers

After a competition

- Read write-ups of well performing solutions that are posted
- You will see that the solutions vary

Kaggle forums = golden resource

1. If you are struggling, the kaggle forums + notebooks that people share = keep you going
2. Having opportunity to discuss what you are seeing + compare with the perspectives of others = great source of inspiration
3. People genuinely want to help + leave their mark on the community
4. Good way to learn

Keep in mind - the best don't share all their tricks

Also, the amount of information that is being shared is enormous, therefore reading the kaggle forums daily = key

- Gets you into the habit of thinking about the competition daily and making small tweaks every day = best way to improve

What are other useful sources of information?

1. Research papers
    a. Requires skill to read them. Don't try to understand every paragraph.
    b. Get an idea of what the paper is talking about
    c. What is the essence of the paper and is the described technique viable or useful for the problem I am trying to solve?
    d. Some papers will be more approachable than others
    e. If you can't understand a paper, it might be best to read the related papers first and then come back
    f. If paper is relevant, then start a thread in the forums and ask for help understanding what is going on there
2. Blog posts
    a. Superior place to start often
    b. Finding good blog posts on arcane subjects can be hard though
    c. Great learning resources because they are often as accurate as the papers
        i. Good for getting intuition on a subject

Work on the solution daily

- Make small tweaks and the effects will compound over time

Early in the competition, focus on trying new architectures, rather than tweaking the hyperparameters too much.

- Exploration should be meaningful and cover as much ground as possible

Trying out varying architectures might take more time early on, but later the performance might thank you.

Run a lot of experiments: how do we know we are moving in the right direction?

- Start from the beginning
    - We should start writing code on the day we decide to start a competition.
    - We should first create a simple baseline

1. Create a simple baseline
    a. Usually a jupyter notebook where we:
        i. Download the data
        ii. Figure out how the data is organized
        iii. Make a submission
            1. Submission = usually all zeros
            2. Make sure that our mechanics are in place before we work on the model itself
2. Start the training
    a. Find a validation split that will track the leader board (public)
    b. Results should ideally align with that of the Kaggle

        c. Likely unattainable for the majority of the competition
        d. Good enough if your results move in the same direction as the results of the leaderboard
3. Study how to best split your data
    a. Is random sampling enough?
    b. Do you need stratified splits based on classes?
    c. Is the data temporal and need to be split on a time stamp?
    d. Proper validation is key to good submission
4. How do we measure whether we got it right or not?
    a. Train at least two models and submit them to the leaderboard
        i. If first model does better than the second model, this should be reflected in both places
    b. Two models = minimum for getting started
    c. Keep your eye out for whether the leaderboard is tracking local results
        i. If not, you should revisit the validation split or test data
5. Ensembling
    a. We want to combine predictions from multiple models in hopes that their errors are not correlated
        i. If model A makes different errors than model B, then some will cancel eachother out
        ii. Using a diverse set of models helps
6. Cross validation
    a. Train multiple models on our data, withholding different parts of it for validation and combining the results

Summary:

1. Join competition early
2. Read forums and work on solution daily
3.

Best hardware for deep learning

Advice: explore hardware only to the extent that you find it entertaining to do so

- Reason = if you don’t know if deep learning is for your, seek credits for AWS or GCP + use cloud VM to find out
- You will learn many useful things in the process that will be valuable for a professional setting
    - How to code from your local machine to the VM
    - How to connect to the VM
    - How to access the Jupyter notebook server
    - Etc
    - Start with google collab or Kaggle Kernels to start
- If you know that deep learning is for your
    - Home rig = most cost effective option

        - Laptop on ubuntu desktop
        - Headless desktop with a gpu running Ubuntu server
        - Ssh into the desktop and access jupyter notebooks on a local network
    - Option 2
        - Desktop with GPU, install ubuntu desktop, do work directly on it
        - Less convenient to working on two separate machines
        - Ubuntu server = more stable than Ubuntu Desktop
        - Easier to install all CUDA and CUDNN libraries
- What GPU should you get?
    - Get the biggest in terms of ram that you can afford
    - Important is how you use it
- While working on the Deep learning projects, you will start to learn what is important
    - Certain datasets or tabular datasets might benefit from faster CPU or more PC ram
    - Training computer vision models might need higher resolution images, which require a multi-gpu setup
    - Maybe a GAN artist with a lot of model weights and results to curate, which might benefit from adding another drive, that is bigger but not necessarily faster might be good
- You need to learn more about hardware that will be effective for your purpose
- Avoid thinking about the minute

Learning how to optimize hardware usage is important: ask yourself the following:

1. Are you loading the data you are accessing over and over again into the ram or are you reading it from the disk?
2. Are you reading the data in sequence or is there a lot of jumping around?
3. Are you using appropriate hardware for the use case?
4. If you need to do data processing, are you offloading the heavy lifting and performing it once before training or are you incurring the cost every time you load an example?
5. As you train, can you tell where your bottleneck is?
6. Is there anything you could do to make the training faster?

Debugging with ease is a superpower

Machine learning code = hard to write
Bugs are hard to diagnose

Jupyter notebooks allow us to do the following

1. Run a single command and be immediately transported to where an error occurs
2. Allows us to look at variable values at the moment when the error occurs
3. Attempt modifications in place to see if that fixes the problem

Time yourself

Keep track of how much time you devote to various activities

Good to get into habit of using %%timeit
Example: the case of using numpy vs pandas to code

```
%%timeit
df.target_fn[np.random.randint(df.target_fn.shape[0])]
% took only 40,8 us
```

```
%%timeit
df.target_fn.sample(1)
% took 80 ms, which is nearly 2000x longer than it took when using numpy!
```

Timing code = useful technique across the board

- Especially when making datasets and dataloaders

You can’t learn a profession by studying a textbook

On finding a job

Employability = hard
Best approach = learn how others did it
Common story = meet people who can make or influence hiring decision where they hang out

- Forums
- Social media
- Conferences
- Meetups
- Blog Posts

Biggest thing = credibility

- Show yoru work, help others, positively impact the community and also demonstrate that you can do non-trivial work
- Project your expertise

Sharing = more people know you exist

Deep learning party is on twitter

Listen to what the experts have to say
Hear about what is going on

Learn about research trends

FOLLOW THE FOLLOWING PEOPLE

1. Jeremy Howard
2. Rachel Thomas

Disable the following

1. Algorithmic timeline
2. Notifications
3.

Turn on the chronological timeline

- 5 to 10 minutes is enough

Share your work

Personal branding

When to start sharing your work

The sooner the better
Some articles are literally not possible to write at a later time

- What did you learn?
- What are your biggest difficulties?
- What were the resources that you found to be really helpful?

No negative consequences with sharing your work

I am scared to share my work! Help!

Feedback = priceless

What to focus on in sharing your work

Speak to your experience

- This is the most valuable thing we have to offer

Showing people what you did, how you did it, how you felt along the way, and what you achieved allows others to get an idea of the map of the path ahead of them as well

- This is immeasurably valuable

Focus your energy on speaking to this one topic that you are interested in or attempting to learn

Don't lose sight of what is important

Why are you doing this?

- To learn

Make mental space for what matters

Avoid social media junk
Wakeup with a fresh mind

- Do most creative work first
- Then use twitter for ideas
    - This removes some creativity and emotional energy , but it minimizes the damage while maximizing the benefits

To engage in afterburners, find a mentor

Good mentor = good at what you care about
Willing to explain to you, in detail, how to do said thing, using a language that is followable for you

Balance two competing aspects for a mentor

1. Good to have a mentor who is like you in some important ways
    a. Easier for you to understand what your mentor has to say
    b. Easier for them to speak to your circumstances much better if they are on a similar path
2. Open yourself to new ideas
    a. More ideas available = more effective problem solving
    b. Listening to similar people is not the best route for this

Remember, more successful people = more people want to talk to them

Go to extreme lengths to make messages as concise and clear for them as possible to reduce the burden that you are placing on the person that you are reaching out to
Interact through forums or media

Summary of mentor interactions

1. Communicate via books or lectures
2. Github issues
3. Tweets
4. Do the work, look through the pertinent material

Biggest regret of fast.ai students

“I wish I spent more time coding and experimenting and less time studying in a traditional sense.”

80% doing and 20% learning
Shortest path to understanding how something works is by practice

Persistence is everything

Tenacity is key
Good map of the terrain is key

Change is about what not to do

We are creatures of habit
The challenge of doing things better requires stopping how we did things in the past

DROP WHAT IS SLOWING YOU DOWN

Learning might just be enough

Strategy to learning

1. Observe whether you are getting the results that you are after
2. If you are not, change your approach

More perspectives on mentoring

Permissionless apprenticeship

- Receive value by giving value first
    - https://twitter.com/jackbutcher/status/1261139777061113858?s=20
    - Give yourself a job working for theme for free
        - Write your own job description
        - Video editor? Cut up their best talks into sharable clips
        - Writer? Organize their tweets into a book outline
        - Designer? Illustrate their best ideas (proof of work)
    -
- Core of every relationship

Learn in public

- https://www.swyx.io/learn-in-public
-

Two types of mentors

1. Gives advice
2. Get you gigs, hire you, offer to lend you money, let you use their network, invite you into rooms that matter

Tap into the power of the community to learn faster

We learn fastest when we are part of a community of learners
Observe how others do things and p;ick up on the subtle but game changing details that we would never learn in a book

- Don't reinvent the wheel, adapt the blueprints fine tuned by others that came before us

Listening to others who are like you and does things that you want to be able to do is magical

1. You become inspired
2. You no longer have to grasp in the dark for ways to approach things
3. You can adapt strategies that work

Energize

The good old days

Turning the tide

Big turn = do something each day at home, then reserve the weekends for recovery and outdoor activity with family

Forming a habit

Prioritize one thing - run before all else
Stop pushing yourself

What and when you eat matters

What works for me

Where to go from here