2021 Fast.ai Community Course Week 1: Your first models

Fast AI 2021, with Qld AI

Welcome

I hope you've had a good week of learning.

The **week 0** Jeremy video has vital tips based off many thousands of students.

You should be ready to code!

Chapter 1 review

Let’s review Fastbook Chapter 1.

Go to notebooks now.

Chapter 1 Questionnaire

Let’s review the questionnaire.

Are the points of the questionnaire clear?

Post questions into the discord for this chapter.

Week 1: Data Science Problem Solving

Fast AI 2021, with Qld AI

Data transforms

We engineer data flow transforms.

Data (type, shape, amplitude) is converted step by step.

Optimising our choices will involve a strategy around *risk trade-offs*.

Balancing explore vs exploit.

Discovery vs creation.

Be excited by challenge

*“I need **you** to create the new machine learning model - on this private, sensitive, enormous, unusual, biased, dataset. No excuses this time!”*

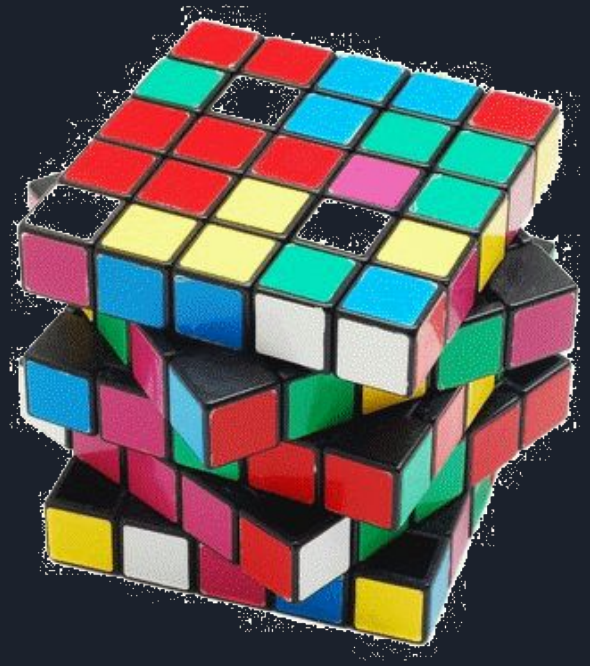
“Great! How exciting.”

Defeating complexity

Neural networks have a very *simple optimiser*, but learn to solve *complex* problems - like recognising a car.

We can apply a similar simple paradigm to our learning.

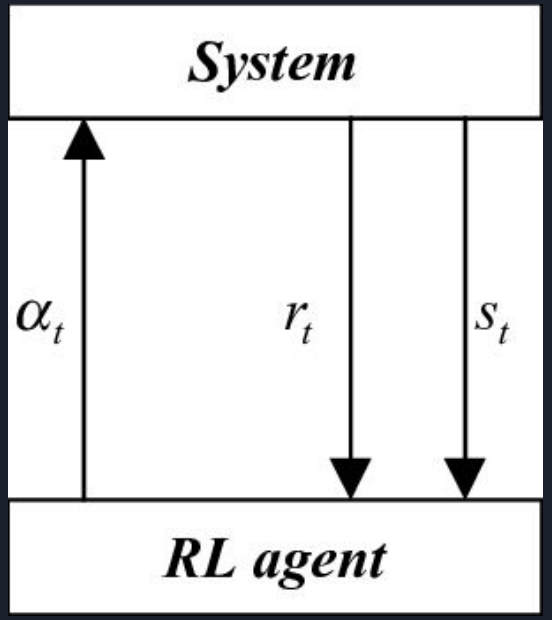
Step through complexity by creating a *benchmark* and then *incrementing*, to isolate errors.



We are a Reinforcement Learner agent

- Expectation of future reward is controlled by a single molecule - dopamine
- actions - pipeline engineering steps
- state - current data pipeline infrastructure

Attempts to take action, or validate state, can fail.



Dopamine - Expectation of future reward

Dopamine controls our *expectation of future reward* (sense of opportunity), and our future-focused control system (creative imagination & strategic planning).

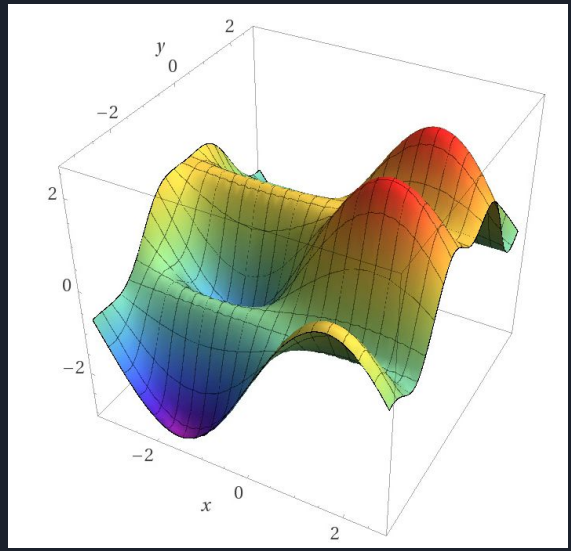
Dopamine dampened mice gave up on tasks in half the time of healthy mice.

Boosting dopamine provides *strategic energy* - the right stimuli will help task persistence, and creative imagination.

What makes a good optimiser?

Broadly - a good optimiser *never gets stuck* while improving effectiveness.

1. Good initialisation.
2. Traction - Iterate quickly.
3. Momentum - don’t get stuck in local minima.
4. Fuel - Cross the long flats (of tedium)
5. Tools - Cross the high mountains (of challenge)
6. Risk managed exploration - recognise time sinks, flow around obstacles, try many routes, back track.



Getting Started

If you invent a self limiting belief you'll never begin.

To get started - requires a *bias* for optimism, to compensate against a likely existing bias about avoiding complexity.

A bias for confidence will boost dopamine - increasing available creative energy.

Be authentic/accountable - don't rehearse excuses - this is a waste of your energy.

Good Initialisation

Be ready to work.

Be open to information. Minimise distraction.

Dispel ignorance.

Be optimistic.

Maximise authenticity.

Be organised. Upgrade tools.

Deconstruct complexity. Defuse risk.

Cross the desert. Avoid the time sink.

You need tenacity to cross a desert of tedium and failure.

You are not ‘cursed’, no matter how long the road. Keep going!

Your brain will continue to suggest new tasks that are easier/more-gratifying...

Recognise and *dodge* the *time sink*.

Cross the mountains of challenge

How fast can you *recover* from failure?

Tip to doom yourself - hate the minutia, and complain.

Mario Kart strategy for failure.

Risk mitigation pervades every process

Balancing *exploit* vs *explore* - utilising known libraries, and existing coding patterns, vs seeking out new libraries, languages, environments, code patterns, class designs.

Overly cautious: It is vital to have a *bias* towards *optimism* or else you will be trapped in a local minima.

Overly confident: We can save time, skipping validation steps, but *recognising risk* is critical for project success.

Recognise and defuse risk

- You have just *copied* and *pasted* ten lines code, and then *sliced* a dataset. How do you feel?
- The process to install a new library will involve at least 4 stages of compilation. Are you about to waste a day?
- You are about to trigger a cluster training loop that will go for a *week*. What is important?

Discovery

Most of the solution for a fault, is in the foundational discovery of what the fault is.

A data scientist needs to *be* a search engine.

Discover the nexus of information.

Don’t let anything **block** all paths at once.

Should you read the *whole* paper?

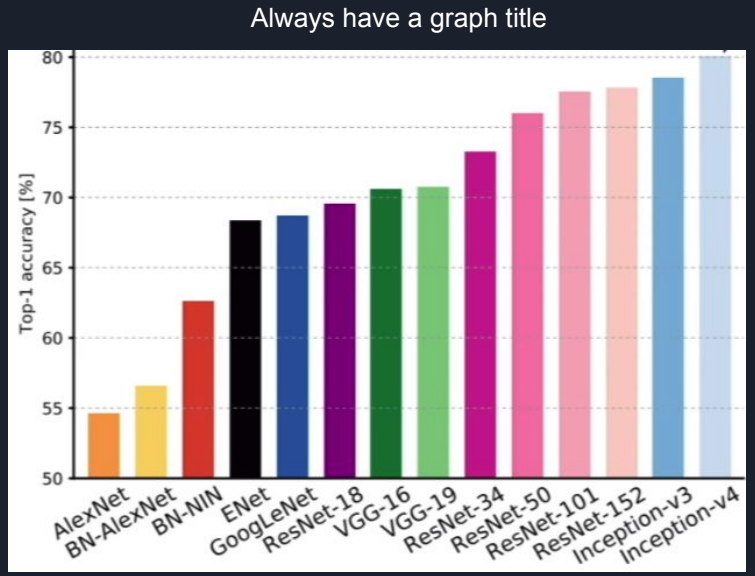
How fast can you get through a 20 minute video?

Discovery - views

Logging and views, defuse risk, by making information *accessible*.

Maximise screen *informational density* - colors can communicate much faster than numbers.

Maximise your informational openness (attentional availability).



Problem Solving code errors

So, the *operation* doesn't have *identity*. ???
Not every problem can be resolved just by an internet search.
Begin by deconstructing complexity - *isolate* the error. Simplify the question.

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-45-b090dbc2fde6> in <module>
      2 clas_pred,bbox_pred,sizes = output[0][idx], output[1][idx], output[2]
      3 bbox_tgt, clas_tgt = output[0][idx],output[1][idx]
----> 4 bbox_tgt, clas_tgt = unpad(bbox_tgt, clas_tgt)

<ipython-input-42-f345468f0a49> in unpad(tgt_bbox, tgt_clas, pad_idx)
      1 def unpad(tgt_bbox, tgt_clas, pad_idx=0):
----> 2     i = torch.min(torch.nonzero(tgt_clas-pad_idx))
      3     return tlbr2cthw(tgt_bbox[i:]), tgt_clas[i:]-1+pad_idx

RuntimeError: operation does not have an identity.
```

Deconstruct complexity.

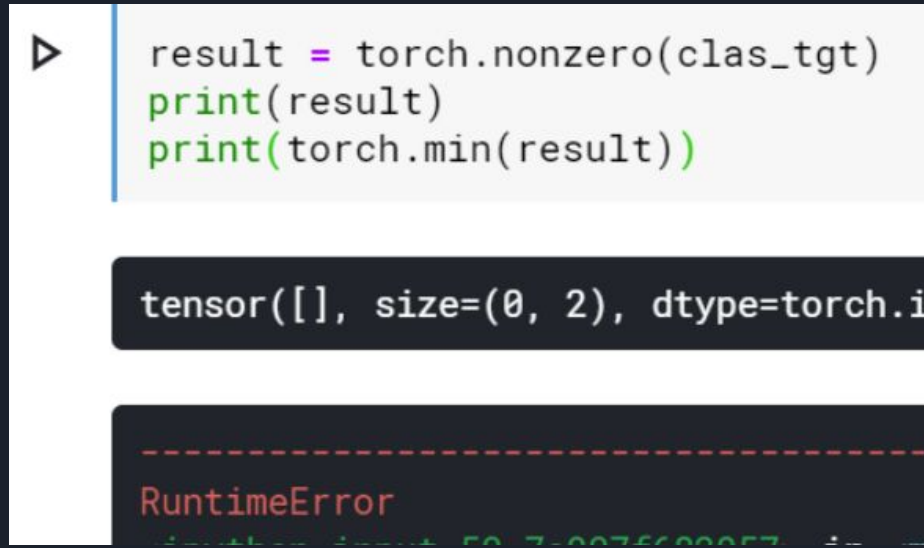
Decouple code/architecture.

‘Shrink the step size’

Reproduce problem.

Validate types, shapes and amplitudes.

Create benchmark working simpler version.



Reducing Complexity - functions/classes

An application has no maximum to potential complexity.

The primary purpose of classes and functions is to *constrain complexity* - through encapsulating functionality into black boxes.

Decoupled code makes it is easy to establish fault & responsibility.

Recognise the risk in poor scoping of variables.

Functions should be short enough that their purpose is obvious.

Maximise *informational density* on screen with good naming.

Deconstruct Complexity

A critical-path class is giving us an error we can’t decipher on instantiation!

What can we do? (other than comb the internet)

```
chest_db = DataBlock(blocks=(ImageBlock, BBoxBlock, BBoxLblBlock),
                     splitter=RandomSplitter(),
                     getters=getters,
                     item_tfms=item_tfms,
                     batch_tfms=batch_tfms,
                     n_inp=1)
```

Deconstruct Complexity

We already have the whole answer.

The primary solution is in isolating the error.

Establish the benchmark - create simpler version. Use a different dataset if you need to.

Step the simple working version towards the complicated implementation that doesn’t work.

Incrementally validate code-portions before aggregation.

You know your homework.

Complete the week 2 material before the next session.

Good luck, and stay excited.