AI Research, Replicability and Incentives

Denny Britz

2020-06-17

More people than ever are working on Deep Learning. Journal and arXiv submissions have increased by ~5x over the past few years. If you consider practitioners who are not publishing their results in academic paper format, the number of people working on something related to AI has probably gone up by 10-25x. Here is a brief analysis I did on my arXiv subscriptions a while ago:

Deep Learning has produced some amazing results in Image Recognition, NLP, generative models, games, and more. It’s hard to argue with such progress. But as researchers are racing to beat each other’s state-of-the-art results, it has become increasingly difficult to decide if a paper is proposing something useful, overfitting a test set, or trying to achieve a flashy outcome for PR reasons. Survey papers and experience reports show that many papers are not replicable, statistically insignificant, or suffer from the narrative fallacy. I will link to some of these throughout the post. On social media, I regularly see posts (e.g. here or here) by practitioners complaining about not being able to get the same results a paper claimed to get.

Quantifying progress is hard. Most empirical subfields of Deep Learning have received some kind of criticism around papers making outlandish claims. One of the more popular examples is a paper Musgrave, Belongie, and Lim (2020) on Metric Learning that re-implemented techniques on equal footing and found:

The trend appears to be a relatively flat line, indicating that the methods perform similarly to one another, whether they were introduced in 2006 or 2019. In other words, metric learning algorithms have not made the spectacular progress that they claim to have made.
Other subfields have seen similar, though perhaps not quite as drastic, criticisms around replicability. **Deep Reinforcement Learning** has probably received the most attention with a whole range of papers Henderson et al. (2017) Khetarpal et al. (2018) Islam et al. (2017) Engstrom, Ilyas, Santurkar, Tsipras, Janoos, et al. (2020) and blog posts Irpan (2018) investigating the high variance of some results.

Trying to understand which results are trustworthy, significant, and generalize to real-world problems has become kind of a guessing game. In this post I want to go over some of the issues around replicability, how open source and academic incentives come into play, and what possible solutions may look like. I’m far from the first one discussing this, but I think that my take on some of the issues is a bit different from what I’ve seen other’s proposing. Another good overview of these trends is Lipton and Steinhardt (2018). At the end of the post, you can find some more papers investigating issues in specific subfields.

Reproduction vs. Replication

I think it’s important to get some terminology out of the way first. **Reproducibility** and **Replicability** are often confused or used interchangeably, but they refer to different things. Depending on which scientific field you’re in, you may see slightly different definitions. Some fields even have swapped the meaning of the two terms. It’s confusing, to say the least. The standard definitions adopted by the ACM (after a correction and swapping the terms at first, hehe) are roughly as follows:

- **Reproducibility** is running the same software on the same input data and obtaining the same results.
- **Replicability** is writing and then running new software based on the description of a computational model or method provided in the original
publication, and obtaining results that are similar enough to come to the same conclusion.

In the age of open source software, reproducibility is pretty straightforward: Just run the code. Sometimes "just running the code" is easier said than done, but still, it’s usually possible if the code is out there and you put in some effort. Replicability on the other hand is almost never easy. But more on that later. In this post I’m only concerned with replicability - arriving at the same high-level conclusion based on the argument a paper is making. This doesn’t necessarily mean getting the exact same results as a paper, but getting similar enough results to believe its argument.

Why replication is hard

Software frameworks

One would think that implementing the same model in two different frameworks would lead to identical results. But that’s not the case. Subtle differences in framework implementations, insufficient documentation, hidden hyperparameters, and bugs can cascade and lead to different outcomes. If you go through the Github issues and forums of popular Deep Learning frameworks, you can find many examples of researchers obtaining unexpected results (like here or here or here or here or here or here). From what I have seen, high-level frameworks like Keras that hide low-level implementation details and come with implicit hyperparameter choices already made for you, are the most common source of confusion.

Subtle implementation differences

Due to the complexity of Deep Learning algorithms and pipelines, papers don’t describe every little implementation detail. Doing so would distract from the overall narrative and overwhelm the reader. Authors may also just have forgotten a few details about a large codebase they’ve been iterating on for several months when writing a paper.

Using the paper format for communicating high-level ideas and the most important decisions seems like a good idea. But it’s not always obvious which parts are important. Sometimes it’s quite counterintuitive. Researchers may not know themselves, and may not even be aware that they don’t know. Small implementation details can sometimes have a huge impact. And since they are not highlighted in the paper, very few people would even consider those details in further experiments.

We find that much of the observed improvement in reward brought by PPO may come from seemingly small modifications to the core algorithm which we call code-level optimizations Engstrom, Ilyas, Santurkar, Tsipras, Janoos, et al. (2020)
Researchers often stick with their favorite random seed and don’t re-run their experiments a sufficient number of times to obtain confidence intervals. This isn’t always done with malicious intent. Authors may be unaware of just how large the variance can be, and sometimes it’s not feasible to re-run experiments. When models take days, weeks, or months to train and experiments cost tens to hundreds of thousands of dollars (the training of GPT-3 costed an estimated $10-20M), it makes sense that not everyone has the luxury of repeatability. Error bars are not requirement for paper acceptances, nor should they necessarily be. However, it’s problematic when changing random seeds can lead to significantly different outcomes:

We demonstrate that the variance between runs is enough to create statistically different distributions just from varying random seeds Henderson et al. (2017)

It is possible to get training curves that do not fall within the same distribution at all, just by averaging different runs with the same hyper-parameters, but random seeds. Islam et al. (2017),
Hyperparameters

Parameters that control the training process, but are not directly part of the model, are called hyperparameters. This usually includes choices like optimizer, learning rate schedule, model size, activation functions, and so on. Studies have shown that proper tuning of hyperparameters of simpler models can lead to better results than using supposedly more powerful models. For example, Melis, Dyer, and Blunsom (2017) showed that a well-tuned LSTM baseline can outperform more recent models. In GAN research,

We find that most models can reach similar scores with enough hyperparameter optimization and random restarts. This suggests that improvements can arise from a higher computational budget and tuning more than fundamental algorithmic change. Lucic et al. (2018)

Figure 5: TRPO on HalfCheetah-v1 using the same hyperparameter configurations averaged over two sets of 5 different random seeds each. The average 2-sample t-test across entire training distribution resulted in $t = -9.0916$, $p = 0.0016$. 
In addition to the issue of excessive tuning, the choice of what is considered a hyperparameters is arbitrary. In the strictest sense, everything not changed by the optimization algorithm that’s not part of the model (or inductive bias) is a hyperparameter. This would include every little decision in your code that could’ve been different, but is not central part of your argument. For example, you would not consider the network architecture a hyperparameter if you are proposing a specific extension, such as a new attention mechanism, to that architecture. However, you could consider the architecture a hyperparameter if you were proposing a new Reinforcement Learning algorithm, as Henderson et al. (2017) showed.

There are thousands of such tiny decisions, as we’ve briefly touched on in the implementation differences section above. By using high-level frameworks you are also introducing additional hyperparameters that may be hidden from you because they are set as the default. It’s infeasible to find all hyperparameters and vary. It’s also not obvious what their range should be, or how they interact with one another. Most authors make educated guesses about what parameters are important, separating them from what they believe the inductive biases to be. But these are mostly guesses based on gut feeling, and sometimes they’re wrong.

Path dependency and the winning lottery ticket

When reporting benchmark results, papers typically claim to have found an algorithm that generalizes to unseen data. Generalization is the goal after all. Often, the findings are accompanied by some convoluted story, discovered after the fact, for why the algorithm works. No matter how smart, we’re all suckers for good stories, especially our own.

In reality, the researchers may have found something more like a lottery ticket that happened to work on a specific problem under a specific set of hyperparameters. It may not work so well on a slightly different data distribution Geirhos et al. (2020). In the published paper and code, you only hear about the winning ticket, not the 100 losing tickets that did not work - architecture and hyperparameter variations the authors tried but failed, probably while evaluating them on the
test set. If you knew about these 100 failed variations, wouldn’t this make you skeptical about the generalization ability of the winning variation?

Research is path-dependent, but you only get to see the final results in the form of a polished paper. It’s like receiving a winning lottery ticket from last week - pretty amazing that someone won, but not very useful for the future. In an ideal world, you would know the generative process of a paper and its code - the full path the researchers took in to arrive at their conclusion. But that’s not practical, nor would it make the authors look good in peer reviews.

During my time working in finance I found that trading companies were well aware of these pitfalls when it comes to backtesting (evaluating a model on historical data) and try to adjust results based on the path that was taken. For example, in some hedge funds, each access to a test set is logged, and the final performance metrics are mathematically adjusted to account for the number of times the data was touched by a research team.

**Computational budget**

Learning curves can be deceiving (in addition to being ADDICTING). It’s not obvious when training process converges. You may be stopping too early, especially when your computational budget for an experiment is constrained.

Computational budgets also affect the space of hyperparameters you can search. In Sivaprasad et al. (2019) the authors found that optimizer hyperparameters were crucial when they evaluated the effect of larger computational budgets on optimizer performance. If you are not using the Adam optimizer, which seems to be easiest to tune but is not always optimal, a larger tuning budget can lead to quite different results:
Despite that, computational budgets used for experiments are rarely reported in papers.

Some research is so compute-intensive that very few labs can do it at all. This is true for gigantic models like GPT-3, but also for subfields like Neural Architecture Search:

Recent advances in neural architecture search (NAS) demand tremendous computational resources, which makes it difficult to reproduce experiments and imposes a barrier-to-entry to researchers without access to large-scale computation Ying et al. (2019)

**Evaluation protocols**

Evaluation protocols can also be ambiguous. Authors may nudge evaluation criteria to be slightly in their favor. For example, in NLP different tokenization schemes and implementations of metrics such a BLEU may be used without explicitly mentioning such details. In Reinforcement Learning, performance can be reported in a number of ways,

Some authors report performance during the learning phase itself after every n rollouts, while others report average score over k once training is stopped, whereas some report scores on variations of the environment not seen during training Khetarpal et al. (2018)
Metric choices and baseline comparisons are also somewhat arbitrary. In some fields, researchers can pick whatever comparison makes their model look best—and there may be dozens of metrics to choose from.

Few papers compare to one another, and methodologies are so inconsistent between papers that we could not make these comparisons ourselves. For example, a quarter of papers compare to no other pruning method, half of papers compare to at most one other method, and dozens of methods have never been compared to by any subsequent work. In addition, no dataset/network pair appears in even a third of papers, evaluation metrics differ widely, and hyperparameters and other confounders vary or are left unspecified. Blalock et al. (2020)

**Test set leakage**

Most Machine Learning practitioners understand that the data should be split into training, validation, and test sets, and that one should not use the test set to inform training decisions. In practice, the story is more complex than that. While the test set should not be used by the training algorithm, it’s unclear to what extent it can be used by the researchers themselves. The researchers are just an extension of the learning algorithm tuning model hyperparameters. They are the meta-learner, and their learning process is something jokingly referred to as graduate student descent.

Because benchmark performance is reported on the test set, researchers must eventually check to make sure their algorithms works well enough to justify publishing a paper. And since the test set is publicly available, nothing prevents researchers from continuously peeking at it. My own experience, as well as that of at least one other person is:

> I have seen ML systems designed by researchers at top tech companies that display test set error during training. People do peek. [...] Don’t get too bogged down by sacrilegious notions. Hypotheses are not made in a vacuum; you have received signal from your test set by even reading other research that reports results on it. The test set is simply a tool, and the better it is protected the more effective it is. Use it to the best of your ability to learn something about your model.

In the field of metric learning, researchers found that several code bases trained directly with test set feedback:

> [...] during training, the test set accuracy of the model is checked at regular intervals, and the best test set accuracy is reported. In other words, there is no validation set, and model selection and hyperparameter tuning are done with direct feedback from the test set. Some papers do not check performance at regular intervals, and instead report accuracy after training for a predetermined number of
iterations. In this case, it is unclear how the number of iterations is chosen, and hyperparameters are still tuned based on test set performance. This breaks one of the most basic commandments of machine learning. Training with test set feedback leads to overfitting on the test set, and therefore brings into question the steady rise in accuracy over time, as presented in metric learning papers. Musgrave, Belongie, and Lim (2020)

Everyone peeks in one way or another. You are peeking just by reading researcher papers reporting results. The question is, how much peeking is appropriate? Once at the end? After each hyperparameter tuning run? Once a day? When are you peeking so much that you should be adjusting your results to account for the test set feedback? And how exactly are you going to adjust for it? As mentioned in the path dependency section above, benchmark results in quantitative finance are sometimes adjusted based how often the test data was touched, but such a system would be difficult to implement in open research.

It’s also unclear to what extent such test set leakage actually matters. Researchers found Recht et al. (2019) Engstrom, Ilyas, Santurkar, Tsipras, Steinhardt, et al. (2020) that image recognition models generalize pretty well to unseen data, despite years of researchers fitting on the same test set.

So at least on CIFAR-10 and ImageNet, multiple years of competitive test set adaptivity did not lead to diminishing accuracy numbers. Recht et al. (2019) and Engstrom, Ilyas, Santurkar, Tsipras, Steinhardt, et al. (2020)

**Data pipelines, pre-processing and augmentations**

Data is another place where replicability issues can arise. Shuffling of training examples can have an impact on performance particularly when the data is not fully independent and identically distributed (iid). Researchers may run experiments on distributed infrastructure, so that even with identical random seeds there will always be stochasticity in the order the data is coming in, leading to slightly different results.

As approaches using self-supervised objectives and data augmentations, e.g. contrastive learning, are becoming more mainstream, more complexity may be moving into the data pipelines themselves. This can create yet another source of variability as researchers tend to focus more on model details as opposed to the data processing in their writeup. Image augmentations may move directly into the data pipeline, but implementation details may differ slightly. In NLP, this was the case with tokenization schemes. It was sometimes unclear which tokenizer and tokenization scheme was used during pre- and post-processing.
Bugs

In traditional software engineering bugs are typically obvious. Your program crashes or produces unexpected output. When implementing complex Deep Learning models, bugs are more subtle. Sometimes they are invisible. Your model may learn just fine and the bug may only slightly affect the convergence rate and final results. Sometimes it can even act as a regularizer! You never know! This recent paper Islam et al. (2019) studied the most common types of bugs in Deep Learning models and frameworks.

Open Source is misunderstood

Open source has had a huge effect on the research community. Conferences and workshops are starting to require code as part of paper submissions, and sites like paperswithcode aggregate open source results. Many (including me) believe that open source code should be a requirement for all publications. While open source is great, I don’t think it solves quite as many issues as some people think it does.

First, open source code provides reproducibility, but not replicability. Code includes hyperparameters and inductive biases from the authors. As discussed above, these code-level optimizations may turn out to be crucial for an algorithm’s performance without ever being mentioned in the paper. Only by forcing researchers to replicate results from the ground up can we find such issues.

Publishing code doesn’t do anything until someone reads and understands it. This may sound obvious, but a lot of published code is probably never read by someone other than its original authors. At least not in detail. Open source doesn’t mean there are no bugs, or that authors didn’t train on test data. I remember a few occasions where people found bugs or test set leakage in published code. Sometimes these resulted in paper retractions, but sometimes such evidence was ignored by the authors, who had already moved on to other projects. Nobody really cared.

Of course, that bugs were found in published code is great - it shows that publishing code is a good thing. But it also raises the question of how many bugs are still sitting there in open source repos that will never be discovered by anyone. Research code, which is often difficult to comprehend and tends to lack in code quality and documentation, is only read by a few select people who don’t even have a strong incentive to find and report bugs.

Open source code also encourages a copy-and-paste SOTA chasing mentality. When you have a piece of code that gets close to state-of-the-art, it’s easy to copy it, tweak a few hyperparameters and architecture choices, and with a bit of luck you’ll get good enough results to publish a paper. This is graduate student descent. With such an approach you’re more likely to overfit
to benchmarks and discover incremental improvements that don’t generalize. If code had not been available, you may have been forced to start from first principles, think deeply about the problem, and come up with new insights.

As discussed previously, code never tells the full story. The code you see published is just the final setting that the author, who is the meta learner, has found to work well. You don’t know the learning process that lead the author to that code. **Code itself is just a hyperparameter setting**, fundamentally not so different from a learning rate you may find in a configuration file. The only difference it’s a hyperparameter setting for the meta learner, while that learning rate is a setting for the learner.

I am not trying to make an argument against open sourcing research code. The positives outweigh the negatives. But it’s important to acknowledge the shortcomings. Where public code truly shines is when it acts as additional documentation for a paper. When a paper is ambiguous, you no longer need to email the author. Instead, you can look at the code to clear up the confusion.

Open source code is also a good negative filter. A result that comes without open source code should raise red flags immediately. But the existence of such code does not say anything about the replicability, validity, or generality of the result.

**Incentives and skin in the game**

Most issues discussed here seem to have straightforward technical solutions. You may be wondering, why don’t researchers just work off a common shared code-base that implements standardized data loading, tracking, and benchmarking, contribute with forks and pull requests, and use held-out test data that is kept private similar to how Kaggle does it? Isn’t that pretty easy to build? Yup. The reason nobody does this is due to a lack of **incentives** and the existing academic **culture**. It’s not so much about technical feasibility.

Researchers in industry and academia are often evaluated (for job applications and promotions) using metrics such as the number of published papers, citation count, publication venues, h-index, etc. This encourages the rapid production of papers, maximizing the chances of journals acceptances. Sometimes this is done by overfitting benchmarks to get state-of-the-art results, which seems to be a proven way to get past skeptical human reviewers. It’s hard to argue against numbers that come packaged with a good story after all. Unless these incentives change, technical solutions are unlikely to be effective and would probably be ignored by the community. We’ve all seen things like this happening:
Another problem may be the lack of skin in the game. Publishing papers has little downside risk, but researchers keep the upside. If research doesn’t make outlandish claims but is slightly flawed or overfit to the test set, there aren’t any consequences for the researchers. It may still get cited or published by a journal. Even when mistakes are found, they are often ignored because fixing an already published paper is not worth the effort, and retractions are rare. While some downside protection is important for innovation (science is hard and we don’t want to discourage risky ideas), close to zero downside can result in a lot of unnecessary noise and outlandish claims. It encourages people to throw random stuff at the wall arXiv.

Research as Art

Before talking more about incentives, let’s take a step back and think about what research is about. Most issues I’ve talked about are specific to empirical, benchmark-driven research. Such research is important, and I believe that competing on good standardized benchmarks was one of the key drivers behind Deep Learning’s rapid progress. But there are other kinds of research that do not rely on benchmarks and are just as important: Theoretical results, meta research and surveys, and novel ideas that don’t work well enough (yet). Let’s talk a bit about the last one.

The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them. ~Sir William Bragg

Replicability and reproducibility are neither sufficient nor necessary conditions for impactful research. Novel ideas, even without experiments or good results, can have a big impact. Neural Networks did not work for a long time, and papers using them were largely ignored. Perhaps we could’ve discovered their
potential earlier if researchers had been incentivized to continue working on them? Similarly, as everyone is working on NNs to beat the latest benchmarks now, we are probably overlooking a lot of potential in other areas of AI that could make use the huge datasets and new infrastructure we’ve created for Deep Learning. Jürgen Schmidhuber, most famous for his work on LSTMs, is known for publishing interesting ideas without extensive experiments or empirical results. Such research is closer to art than empirical science, which is obsessed with quantifiable facts and numbers. Both types of research are needed, but current incentives seem to favor benchmark hill climbing over novel ideas.

Peer Review is partly to blame for this - it’s difficult to evaluate crazy ideas and assign numerical scores to them. Reviewers also have their own ego to fight with. It’s comfortable and low-risk to rely on standard benchmarks metrics when writing reviews. This has been true for other scientific fields throughout history, where groundbreaking ideas have sometimes been rejected by the standard peer review process:

> You could write the entire history of science in the last 50 years in terms of papers rejected by Science or Nature - *Paul Lauterbur*

Max Planck described his editorial philosophy for *Annalen der Physik*, where Einstein published his papers and which had an acceptance rate of ~90%, as:

> To shun much more the reproach of having suppressed strange opinions than that of having been too gentle in evaluating them.

Forcing everyone to adhere to replicability standards could push academia even further into that direction - resulting in less artistic and audacious research. Perhaps the solution is the opposite - to stop being obsessed with measures and treat research as more of an art. Let the free market of ideas decide what bubbles up. This also requires changing incentives and culture, perhaps even more so than forcing replicability.

**Top-down incentive change**

One way to improve academic incentives is to force top-down change. This requires buy-in from various market participants: Journals could change guidelines around peer review and acceptance criteria. Universities could change admission, graduation and promotion processes. Companies could change the way credentials and publications are viewed during and hiring and promotion. Influential academic leaders could take on some risk and promote a new system. The recent changes to EMNLP reviewing policies are one example of such change:

Another example of top-down change is distill.pub, which is an academic journal dedicated to publishing modern research that is not well-suited for the PDF format. I categorize distill.pub as top-down because it requires buy-in from industry leaders to drive change. People in charge of hiring and promotion should value publications on distill.pub the same as publications in other journals when
making decisions. If this does not happen, there are few incentives for researchers to change their behavior.

Top-down change can be slow. To make a startup analogy, top-down change is like doing enterprise sales - you must convince the management team of some big old-school organization that your new way of doing things creates value. Not necessarily value for the customers or ecosystem in the long term, but value for the company and the people on the management team while they are still around to benefit from it. Nobody likes to buy stuff that adds risk to their portfolio but yields benefits only to the next generation of people that come long after they’re gone. They’d rather stick with the status quo.

Bottom-up incentive change

Another way to fix incentives is from the bottom up. Let’s stay with the startup analogy. In this case, you skip enterprise sales and provide a product that is useful to individuals or small teams. Once it reaches a critical mass, larger organizations are simply forced to adopt it. In the developer space, quite a few products followed this pattern. One example is Github. Many organizations used to have their own internal version control system, but individual employees found Github so much better that they started to use it internally, even without explicit permission from the top. Now, those organizations pay Github through an enterprise plan, but Github did not start out making deals with the management teams trying to convince them. Old-school companies had no choice but to adopt it due to cultural pressure.

In the case of academia, changing incentives from the bottom up means providing people with alternatives to exit or circumvent the traditional academic system. This is already happening in various ways. In addition to publishing papers, a few researchers are now choosing to promote their research through blogs, project pages with interactive visualizations, YouTube videos, Github code, live demos, or other mediums. As this HN user put it, this allows them to reach a much wider audience and build an personal portfolio:

The research community really should do a better job of engaging with the wider world, irrespective of whether it increases citations. For example, I had this paper in Hotnets 2019, and about 80 people saw my presentation. But I also spent some time, turned it into a little video and put it on YouTube. 360,000 people have seen it there. Now the subtleties were probably lost on many of those people, but if only a few of those people got something out of it, then it will likely have had more impact than the original paper.

In today’s environment, popular projects, even without academic papers and citations, can lead directly to job offers in industry labs. Less so in university settings, which rely more heavily on traditional metrics. This is the start of an alternative system. Open Source has done something similar for Computer
Science. Few companies these days have a hard requirement for a Computer Science degree - real world projects can take that place.

The way research is discovered and disseminated has also been changing. In the past, researchers relied heavily on journals, conferences and peer connections to discover relevant research. Now that everyone is publishing on arXiv, discovery and curation also happens on social media, like Reddit and Twitter, and independent sites such as arxiv-sanity. While most researchers’ goal is still to publish their work in journals, few seem to rely on (or trust) them to curate and discover research. They are now acting more like a stamp of approval - similar to university degrees.

Another trend that I hope we will be seeing more of are independent researchers. The traditional system is highly political and favors researchers in large labs and universities. Independent researchers like gwern have proven that you don’t need to have an affiliation to do interesting things. Unfortunately, there seem to exist few structured ways to make a living doing independent research.

It would be an interesting exercise to design a new academic incentive system from the ground up for the 21st century. Many of the current processes come from an era where the internet did not exist or was not mainstream. Personally, I am curious if something like an academic prediction market that provides in centives for critical discussion and review of results could work. I have a few thoughts on this that I may write up in a separate post. This one is already way too long!


