Reproducibility in Machine Learning - From Theory to Practice

Koustuv Sinha
McGill University / Mila / Facebook AI Research
https://cs.mcgill.ca/~ksinha4

MICCAI Hackathon,
October 2nd, 2020
Agenda

1. Why do we need Reproducibility in Science
2. Case study on the need of reproducibility in Machine Learning.
3. What is the ML community doing about it.
4. How can you perform reproducible research?
Reproducibility in Science

“Reproducibility refers to the ability of a researcher to duplicate the results of a prior study….”

Using the same materials as were used by the original investigator.

Reproducibility is a minimum necessary condition for a finding to be believable and informative.”

Bollen et al.
National Science Foundation, 2015.
Reproducibility in Science

Data
- Same
  - Reproducible
  - Robust
- Different
  - Replicable
  - Generalisable

Code & Analysis
- Same
- Different

https://github.com/WhitakerLab/ReproducibleResearch
Reproducibility Crisis in Science

Why care about Reproducibility

Is there a Reproducibility Crisis?

1,576 researchers surveyed

- 52% Yes, a significant crisis
- 38% Yes, a slight crisis
- 7% Don't know
- 3% No, there is no crisis
Reproducibility Crisis in Science

Why care about Reproducibility

Source: https://www.nature.com/news/1-500-scientists-lift-the-lid-on-reproducibility-119970
Findings
Reproducibility is easy, right?

Same code

Same amount of data. Same amount of computation.
Quantifying Reproducibility Issues in ML

- 255 papers from 1984 to 2017
- 63.5% of papers were reproducible
- Significant factors which affect reproducibility:
  1. Pseudo-code
  2. Hyperparameters
  3. Readability: Equations, Tables,
  4. Compute Needed
  5. Primary Topic

---

### Table 1: Significance test of which paper properties impact reproducibility. Results significant at $\alpha \leq 0.05$ marked with *.

<table>
<thead>
<tr>
<th>Feature</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year Published</td>
<td>0.964</td>
</tr>
<tr>
<td>Year First Attempted</td>
<td>0.674</td>
</tr>
<tr>
<td>Venue Type</td>
<td>0.631</td>
</tr>
<tr>
<td>Rigor vs Empirical*</td>
<td>$1.55 \times 10^{-9}$</td>
</tr>
<tr>
<td>Has Appendix</td>
<td>0.330</td>
</tr>
<tr>
<td>Looks Intimidating</td>
<td>0.829</td>
</tr>
<tr>
<td>Readability*</td>
<td>$9.68 \times 10^{-25}$</td>
</tr>
<tr>
<td>Algorithm Difficulty*</td>
<td>$2.94 \times 10^{-5}$</td>
</tr>
<tr>
<td>Pseudo Code*</td>
<td>$2.31 \times 10^{-4}$</td>
</tr>
<tr>
<td>Primary Topic*</td>
<td>$7.039 \times 10^{-4}$</td>
</tr>
<tr>
<td>Exemplar Problem</td>
<td>0.720</td>
</tr>
<tr>
<td>Compute Specified</td>
<td>0.257</td>
</tr>
<tr>
<td>Hyperparameters Specified*</td>
<td>$8.45 \times 10^{-6}$</td>
</tr>
<tr>
<td>Compute Needed*</td>
<td>$8.75 \times 10^{-5}$</td>
</tr>
<tr>
<td>Authors Reply*</td>
<td>$6.01 \times 10^{-8}$</td>
</tr>
<tr>
<td>Code Available</td>
<td>0.213</td>
</tr>
<tr>
<td>Pages</td>
<td>0.364</td>
</tr>
<tr>
<td>Publication Venue</td>
<td>0.342</td>
</tr>
<tr>
<td>Number of References</td>
<td>0.740</td>
</tr>
<tr>
<td>Number Equations</td>
<td>0.004</td>
</tr>
<tr>
<td>Number Proofs</td>
<td>0.130</td>
</tr>
<tr>
<td>Number Tables*</td>
<td>0.010</td>
</tr>
<tr>
<td>Number Graphs/Plots</td>
<td>0.139</td>
</tr>
<tr>
<td>Number Other Figures</td>
<td>0.217</td>
</tr>
<tr>
<td>Conceptualization Figures</td>
<td>0.365</td>
</tr>
<tr>
<td>Number of Authors</td>
<td>0.497</td>
</tr>
</tbody>
</table>

---

A Step Toward Quantifying Independently Reproducible Machine Learning Research

Edward Raff
Booz Allen Hamilton
raff.edward@bah.com
University of Maryland, Baltimore County
raff.edward@umbc.edu
Subfield: Computer Vision

What the computer sees

- 82% cat
- 15% dog
- 2% hat
- 1% mug
Subfield : Computer Vision
Different models vary differently with varying seeds

Subfield: Computer Vision

Unreproducible Research is Reproducible

Xavier Bouthillier, César Laurent, Pascal Vincent

Generative Adversarial Networks (GANs) are a way to make a generative model by having two neural networks compete with each other.

The **discriminator** tries to distinguish genuine data from forgeries created by the generator.

The **generator** turns random noise into imitations of the data, in an attempt to fool the discriminator.
Generative Adversarial Networks (GAN)

- Training is extremely sensitive to hyperparameter choices

Subfield: Reinforcement Learning (RL)

- Robotics
- Video games
- Conversational systems
- Medical intervention
- Algorithm improvement
- Crop management
- Personalized tutoring
- Energy trading
- Autonomous driving
- Prosthetic arm control
- Forest fire management
- Financial trading
- Many more!

Learn $\pi = \text{strategy to find this cheese!}$

- Very general framework for sequential decision-making!
- Learning by trial-and-error, from sparse feedback.
- Improves with experience, in real-time.
Robustness of RL algorithms

Deep Reinforcement Learning that Matters

Peter Henderson¹, Riashat Islam¹², Philip Bachman², Joelle Pineau¹, Doina Precup¹, David Meger¹

¹ McGill University, Montreal, Canada
² Microsoft Maluuba, Montreal, Canada

Consider Mujoco simulator:

Same algorithm - different results!
Different results in different hyperparameter choices
Let us look a little closer

Same algorithm, same code, same hyperparameters, different seeds
Even number of trials per experiment is not standardized

<table>
<thead>
<tr>
<th>Work</th>
<th>Number of Trials</th>
</tr>
</thead>
<tbody>
<tr>
<td>(et al. 2016)</td>
<td>top-5</td>
</tr>
<tr>
<td>(et al. 2017)</td>
<td>3-9</td>
</tr>
<tr>
<td>(et al. 2016)</td>
<td>5 (5)</td>
</tr>
<tr>
<td>(et al. 2017)</td>
<td>3</td>
</tr>
<tr>
<td>(et al. 2015b)</td>
<td>5</td>
</tr>
<tr>
<td>(et al. 2015a)</td>
<td>5</td>
</tr>
<tr>
<td>(et al. 2017)</td>
<td>top-2, top-3</td>
</tr>
</tbody>
</table>
Even number of trials per experiment is not standardized

Top-3 results

Baseline to beat

Strong positive bias, variance appears much smaller!
Different methods have distinct set of hyperparams
- Different methods exhibit variable sensitivity to hyperparams
- What method is best often depends on data / compute budget

From **fair** comparisons...

to **robust** conclusions.
What we are doing about it
The Open Science Movement

Community efforts to increase transparency in science

- Journal for Reproducibility: ReScience
- Checklists
- Code Submission Policies
- Reproducibility Challenges
Disseminating Reproducibility Reports

The ReScience Journal

- Platform to publish reproducibility reports
- Extensive review process
- Reproducibility reports of annual ML Reproducibility Challenge published (ICLR 2019, NeurIPS 2019)

Reproducible Science is good. Replicated Science is better.

ReScience C is an open-access peer-reviewed journal that targets computational research and encourages the explicit replication of already published research, promoting new and open-source implementations in order to ensure that the original research is reproducible.

https://rescience.github.io/
The ReScience Journal

- Open reviewing, on Github
- Accepts reproducibility reports on a vast array of fields
- Large team of editors for rolling reviews over the year

https://rescience.github.io/
The Machine Learning Reproducibility Checklist

- Introduced by Joelle Pineau, 2018
- Minimal information that must be included in the manuscript
- Not necessarily exhaustive
- V1.2 deployed during NeurIPS 2019 review process
- Part of NeurIPS, ICML, ICLR submission guidelines

The Machine Learning Reproducibility Checklist

The Machine Learning Reproducibility Checklist (v2.0, Apr 7, 2020)

For all models and algorithms presented, check if you include:
- A clear description of the mathematical setting, algorithm, and/or model.
- A clear explanation of any assumptions.
- An analysis of the complexity (time, space, sample size) of any algorithm.

For any theoretical claim, check if you include:
- A clear statement of the claim.
- A complete proof of the claim.

For all datasets used, check if you include:
- The relevant statistics, such as number of examples.
- The details of train / validation / test splits.
- An explanation of any data that were excluded, and all pre-processing steps.
- A link to a downloadable version of the dataset or simulation environment.
- For new data collected, a complete description of the data collection process, such as instructions to annotators and methods for quality control.

For all shared code related to this work, check if you include:
- Specification of dependencies.
- Training code.
- Evaluation code.
- Pre-trained model(s).
- README file includes table of results accompanied by precise command to run to produce those results.

For all reported experimental results, check if you include:
- The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results.
- The exact number of training and evaluation runs.
- A clear definition of the specific measure or statistics used to report results.
- A description of results with central tendency (e.g., mean) & variation (e.g., error bars).
- The average runtime for each result, or estimated energy cost.
- A description of the computing infrastructure used.

Surprisingly, 36% of papers judge error bars are applicable to their results, while 87% see clear value in defining metrics and statistics used!
The Open Science Movement

Code Submission Policies

Values populated from Author response, may not be the actual number.
The Machine Learning Reproducibility Checklist

Does the checklist affect acceptance rates?

As of now, No.

Do reviewers find checklists useful?

Yes!

Improving Reproducibility in Machine Learning Research (A Report from the NeurIPS 2019 Reproducibility Program)

Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d’Alché-Buc, Emily Fox, Hugo Larochelle

The ML Code Completeness Checklist

- Introduced by Robert Stojnic (PapersWithCode), 2020
- Enforces open source code released by papers to follow the guidelines.

1. **Dependencies** — does a repository have information on dependencies or instructions on how to set up the environment?

2. **Training scripts** — does a repository contain a way to train/fit the model(s) described in the paper?

3. **Evaluation scripts** — does a repository contain a script to calculate the performance of the trained model(s) or run experiments on models?

4. **Pretrained models** — does a repository provide free access to pretrained model weights?

5. **Results** — does a repository contain a table/plot of main results and a script to reproduce those results?
Checklists - The ML Code Completeness Checklist

1. **Dependencies** — does a repository have information on dependencies or instructions on how to set up the environment?

2. **Training scripts** — does a repository contain a way to train/fit the model(s) described in the paper?

3. **Evaluation scripts** — does a repository contain a script to calculate the performance of the trained model(s) or run experiments on models?

4. **Pretrained models** — does a repository provide free access to pretrained model weights?

5. **Results** — does a repository contain a table/plot of main results and a script to reproduce those results?

- NeurIPS 2019 repositories with 0 ticks had a median of 1.5 GitHub stars.
- In contrast, repositories with 5 ticks had a median of 196.5 GitHub stars.
The Open Science Movement

Community efforts to increase transparency in science

- Checklists
- Code Submission Policies
- Reproducibility Challenges
Code Submission Policies

- ICML 2019 and NeurIPS 2019 rolled out explicit code-submission policies
- Many concerns regarding Dataset confidentiality, Proprietary software, Computation infrastructure, Replication of mistakes...
- NeurIPS 2019/2020 code submission policy leaves significant time and flexibility - “expects code only for accepted papers, and only by the camera-ready deadline”
The Open Science Movement

Community efforts to increase transparency in science

- Checklists
- Code Submission Policies
- Reproducibility Challenges
The ML Reproducibility Challenge

- Inaugural edition at ICLR 2018
- Till date 3 editions: ICLR 2018, ICLR 2019 and NeurIPS 2019
- Choose a paper, and reproduce full / parts of the paper
- In ICLR 2018-19 editions, we chose papers from submitted list
- Starting from NeurIPS 2019, we choose papers from accepted list

The ML Reproducibility Challenge

Conference → Claim a paper → Track → Submit → Review → Journal publication

Baselines
Ablation
Replication

OpenReview.net

RESCIENCE C
The ML Reproducibility Challenge

NeurIPS 2019

63 Universities

10 Institutions

5 ML Courses

173 Papers claimed

84 Reports submitted

10 Reports published in ReScience

https://openreview.net/group?id=NeurIPS.cc/2019/Reproducibility_Challenge
Related work in community

- EMNLP Reproducibility Challenge (Jesse Dodge et al.)
  - University of Washington, Winter 2020
- Workshops
  - ICML 2017, ICML 2018, ICLR 2019 (Rosemary Nan Ke et al.)
How can you perform Reproducible research
Let us start with a simple example

- Say you have an awesome research idea - Transfer Learning on MNIST dataset
Setting the basics - Training and test pipelines

- You start with the basics:
  - Set up Dataloaders
  - Set up training loop
  - Set up test loop
  - Run the experiment

```python
dataloaders = ClassIncrementalMNIST(loc="/path/to/loc")
model = FancyResNetModel(args)

for task_loader in dataloaders:
    model.train()
    model.test()
```

```
python train.py --hidden_dim 100 --batch_size 32 --num_tasks 10 --dropout 0.2
    --with_mask --epochs 20 --log_interval 100 --learning_rate 0.001 --optimizer sgd
    --scheduler_type plateau --scheduler_gamma 0.9 --weight_decay 0.9
```
You start with the basics:
- Set up Dataloaders
- Set up training loop
- Set up test loop
- Run the experiment

So many arguments!!
Practice 1: Use config files

- For easy management and ease of mind, use config files with all hyperparameters
  - Use either .json or .yaml file
  - .yaml better for comments
Practice 1: Use config files

- For easy management and ease of mind, use config files with all hyperparameters
  - Use either .json or .yaml file
  - .yaml better for comments
  - Running the code is simple

```python
python train.py --config exp
```
Practice 1: Use config files

- For easy management and ease of mind, use config files with all hyperparameters
  - Use either .json or .yaml file
  - .yaml better for comments
  - Running the code is simple
- Have separate config file for each experiment!

Checkout Hydra!
https://hydra.cc/
Inference - checking your model performance

- Ideally, you first infer on your validation set.
- You can also write a separate inference script to evaluate on held out test set.

```python
import classyvibe

dataloaders = ClassIncrementalMNIST(loc="/path/to/loc")
model = FancyResNetModel(args)

for task_loader in dataloaders:
    train_loader, val_loader = task_loader
    model.train(train_loader)
    model.test(val_loader)
```
Inference - checking your model performance

- Ideally, you first infer on your validation set
- You can also write a separate inference script to evaluate on held out test set
- But, for this script, you need to save your model!
Practice 2: Effective checkpointing

- Save as much as your resources permit!
- Save model checkpoint on last epoch and best performing epoch
- If your model is long running, consider saving within epochs
Practice 2: Effective checkpointing

- Save as much as your resources permit!
- Save model checkpoint on last epoch and best performing epoch
- If your model is long running, consider saving within epochs
- If your model fails, you can load from a previous checkpoint!

```python
args = load_args("exp.yaml")
dataloaders = ClassIncrementalMNIST(loc="/path/to/loc")
model = FancyResNetModel(args)
for task_loader in dataloaders:
    train_loader, val_loader = task_loader
    model.train(train_loader)
    score = model.test(val_loader)
    model.save(args.save_loc)
```

```
$ ls -lhtr
-rw-r--r-- 20 user group 200M Aug 01 2020 checkpoint_epoch=40.pt
-rw-r--r-- 20 user group 200M Aug 01 2020 checkpoint_epoch=50.pt
```
Checking your model performance
- monitor training

- Your model seems to not give good scores ... how does the training look like?
- Maybe you need to check the training loss?

```python
args = load_args("exp.yaml")
dataloaders = ClassIncrementalMNIST(loc=/path/to/loc)
model = FancyResNetModel(args)

for task_loader in dataloaders:
    train_loader, val_loader = task_loader
    model.train(train_loader)
    score = model.test(val_loader)
    model.save(args.save_loc)
```
Practice 3: Logging

- Log all features of the training and evaluation
- Ideally, save the logs in a log file
  - Use python `logging` module

```python
dataloaders = ClassIncrementalMNIST(loc="/path/to/loc")
model = FancyResNetModel(args)

for epoch in range(args.epochs):
    for task_loader in dataloaders:
        train_loader, val_loader = task_loader
        train_loss, train_acc = model.train(train_loader)
        val_loss, val_acc = model.test(val_loader)
        print(f"Epoch: {epoch}, Train loss: {train_loss}, Train Acc: {train_acc}"
        print(f"Val loss : {val_loss}, val acc : {val_acc}"
```
Practice 3: Logging

- Better, use logging services:
  - TensorBoard

```python
dataloaders = ClassIncrementalMNIST(loc="/path/to/loc")
model = FancyResNetModel(args)
logger = FancyLogger(args)

for epoch in range(args.epochs):
    for task_loader in dataloaders:
        train_loader, val_loader = task_loader
        train_loss, train_acc = model.train(task_loader)
        val_loss, val_acc = model.test(task_loader)
        metrics = {
            "train_loss": train_loss,
            "train_acc": train_acc,
            "val_loss": val_loss,
            "val_acc": val_acc,
            "step": epoch
        }
        logger.log_metrics(metrics)
```
Practice 3: Logging

- Better, use logging services:
  - TensorBoard
  - Weights & Biases
Practice 3: Logging

- Better, use logging services:
  - TensorBoard
  - Weights & Biases
  - and many more ...
    - Comet.ml
    - Visdom
    - MLFlow
Practices 1-3: Experiment Management

- Setting up boilerplates for practices 1-3 are an useful practice, but can get cumbersome!
- Consider using experiment management tools:
  - Pytorch Lightning
  - Sacred
  - MLFlow
All is good, but something is odd!

- Multiple runs of the same experiment shows different results!
- You might have forgot to set the seed
Practice 4: Set the seed, forget about it!

- Before running an experiment, set the seed
- Make sure seed is set for all randomization sources
- Set the seed and save it in config
- Do not optimize the seed!
- Report an average of different seeds for model variance
- Keep an eye out for GPU reproducibility *

Wait, I forgot what I did to make this work?

- You are deep in your research, but you forgot exactly which line of code you changed last night while being caffeinated.
- Now, you are unable to reproduce your own results!!
Practice 5: Version your code!

- Use Git to version your code
- Commit early, Commit often!
- Add descriptive commit messages
  - You can also add the raw result in the commit message so that you remember what point you left your work
Practice 5: Version your code!

- Use Git to version your code
- Commit early, Commit often!
- Add descriptive commit messages
  - You can also add the raw result in the commit message so that you remember what point you left your work
- Github is your friend!
  - Tag versions of your project to separate or major decisions
  - Have separate branch for small proof of concepts
Mind your Data!

- In the transfer learning setup, you decided it would be good idea to mix certain classes of Fashion MNIST with MNIST digits
Mind your Data!

- In the transfer learning setup, you decided it would be good idea to mix certain classes of Fashion MNIST with MNIST digits
- You went a bit too deep down the rabbit hole...
Practice 6: Track your data too!

- Easiest way to keep track of your data is to add it to Git version system
- However, due to large size it is sometimes not feasible
- Compute md5 hash of the data directory and save it in config file
  - Assuming config files are added to git
- In each experiment, compare the md5 hash
Practice 6: Track your data too!

- Easiest way to keep track of your data is to add it to Git version system
- However, due to large size it is sometimes not feasible
- Compute md5 hash of the data directory and save it in config file
  - Assuming config files are added to git
- In each experiment, compare the md5 hash
- Backup data periodically using Google Drive / AWS S3 buckets

```python
def md5_update_from_dir(directory, hash):
    assert Path(directory).is_dir()
    for path in sorted(Path(directory).iterdir(), key=lambda p: str(p).lower()):
        hash.update(path.name.encode())
        if path.is_file():
            with open(path, "rb") as f:
                for chunk in iter(lambda f.read(4096), b")
                    hash.update(chunk)
        else:
            hash = md5_update_from_dir(path, hash)
    return hash
```

```python
def md5_dir(directory):
    return md5_update_from_dir(directory, hashlib.md5()).hexdigest()
```
Practice 6: Track your data too!

- If you are releasing your own data, consider adding a datasheet!

Datasheets for Datasets

TIMNIT GEBRU, Google
JAMIE MORGENSTERN, Georgia Institute of Technology
BRIANA VECCHIONE, Cornell University
JENNIFER WORTMAN VAUGHAN, Microsoft Research
HANNA WALLACH, Microsoft Research
HAL DAUMÉ III, Microsoft Research; University of Maryland
KATE CRAWFORD, Microsoft Research; AI Now Institute

You have results, time to make some shiny plots / analysis!

- You have nice results which you want to discuss with your supervisor
- You find a cool plotting function and generate shiny plots ...
- Supervisor asks to modify the plot
- You can’t seem to find your plotting script, nor the output file!

Plots taken from the paper "What makes a good conversation? How controllable attributes affect human judgments", See et al, 2019
Practice 7: Maintain notebooks!

- Maintain a set of Jupyter notebooks in a separate folder in your code
- Have separate notebooks for data analysis, result analysis, plot generation, table generation
- Use notebooks to share intermediate results
  - Use Github to render notebooks in your discussions!
- Supercharge your notebooks with Jupyter Contrib Extensions!

Practice 8: Report results with error bars

- Once the pipeline is done, run your experiment with multiple seeds and report the variance in the results
- Add the seed in the config file for reproducible runs
- Choose arbitrary seeds - do not run grid search on it!
Practice 8: Report results with error bars

- Once the pipeline is done, run your experiment with multiple seeds and report the variance in the results
- Add the seed in the config file for reproducible runs
- Choose arbitrary seeds - do not run grid search on it!
- Plot with error bars and add confidence intervals in table
  - Define the criteria:
    - Multiple seeds
Practice 8: Report results with error bars

- Once the pipeline is done, run your experiment with multiple seeds and report the variance in the results
- Add the seed in the config file for reproducible runs
- Choose arbitrary seeds - do not run grid search on it!
- Plot with error bars and add confidence intervals in table
  - Define the criteria:
    - Multiple seeds
    - Multiple datasets

![Accuracy vs Relation Length](image)

<table>
<thead>
<tr>
<th>Models</th>
<th>Unstructured models (no graph)</th>
<th>Structured model (with graph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>BiLSTM - Attention</td>
<td>BiLSTM - Mean</td>
</tr>
<tr>
<td>Clean</td>
<td>0.58 ±0.03</td>
<td>0.53 ±0.06</td>
</tr>
<tr>
<td>Supporting</td>
<td>0.76 ±0.02</td>
<td>0.64 ±0.02</td>
</tr>
<tr>
<td>Irrelevant</td>
<td>0.7 ±0.05</td>
<td>0.76 ±0.02</td>
</tr>
<tr>
<td>Disconnected</td>
<td>0.49 ±0.08</td>
<td>0.45 ±0.08</td>
</tr>
<tr>
<td>Supporting</td>
<td>0.67 ±0.06</td>
<td>0.66 ±0.07</td>
</tr>
<tr>
<td>Irrelevant</td>
<td>0.51 ±0.06</td>
<td>0.52 ±0.06</td>
</tr>
<tr>
<td>Disconnected</td>
<td>0.57 ±0.07</td>
<td>0.57 ±0.06</td>
</tr>
<tr>
<td>Average</td>
<td>0.61 ±0.06</td>
<td>0.59 ±0.06</td>
</tr>
</tbody>
</table>
Practice 9: Manage dependencies

- In your final release, collect all dependencies that have been used in the project
- Add requirements.txt file in your repository

$ pip freeze > requirements.txt
Practice 9: Manage dependencies

- In your final release, collect all dependencies that have been used in the project
- Add requirements.txt file in your repository
- For better reproducibility, consider using Docker or Singularity containers!

```bash
$ pip freeze > requirements.txt
```
Practice 10: Open Source your Research

- After your paper release, considering open-sourcing your work!
  - Adds visibility to your paper
  - Encourages reproducible research

https://paperswithcode.com/
Practice 10: Open Source your Research

- Pre-release checklist:
  - Squash commits in public branch (master) to a single commit before making your repository public
  - Make sure your code doesn’t contain any API keys (for loggers)
Practice 10: Open Source your Research

- Pre-release checklist:
  - Squash commits in public branch (master) to a single commit before making your repository public
  - Make sure your code doesn’t contain any API keys (for loggers)
  - Keep an eye out for hardcoded file locations
  - Format your code to improve readability
  - Document your code!
Practice 11: Effective communication with README.

- Following the ML Code completeness checklist, add these info in your readme:
  - Dependencies
  - Training scripts
  - Evaluation scripts
  - Pre-trained models
  - Results

Source: https://medium.com/paperswithcode/ml-code-completeness-checklist-e9127b168501
Practice 11: Effective communication with README.

- Following the ML Code completeness checklist, add these info in your readme:
  - Dependencies
  - Training scripts
  - Evaluation scripts
  - Pre-trained models
  - Results
- Good to have:
  - Contributing guide
  - Blog Post

Source: [https://medium.com/paperswithcode/ml-code-completeness-checklist-e9127b168501](https://medium.com/paperswithcode/ml-code-completeness-checklist-e9127b168501)
Practice 12: Test and validate

- Additionally, test your setup on a separate machine to ensure reproducible build
- Use Google Cloud (GCP) / AWS / Azure to spawn a small environment
- Test loading and inference of your model
  - Makes sure dependencies are ok
  - Check if no hardcoded paths exist in your code
Takeaways

● Reproducibility in Machine Learning is extremely important for advancement of the field
● The ML Community is coming up with innovative ways to encourage Reproducibility
● Commit to reproducible research early on in your workflow!
Takeaways

- Reproducibility in Machine Learning is extremely important for advancement of the field
- The ML Community is coming up with innovative ways to encourage Reproducibility
- Commit to reproducible research early on in your workflow!

*Science is not a competitive sport!*

- Joelle Pineau, NeurIPS 2018

---

**ML Reproducibility Tools and Best Practices**
August 5, 2020

Koustuv Sinha and Jessica Zosa Forde

Participate in the ML Reproducibility Challenge 2020!

ML Reproducibility Challenge 2020

for papers published in:

https://paperswithcode.com/rc2020
Thanks for listening! Questions?

Thanks to many reviews from Joelle Pineau, Shagun Sodhani, Jessica Forde, Matthew Muckley and Michela Paganini from Facebook AI Research.

Reproducibility Challenge
Koustuv Sinha
Jessica Forde
Rosemary Nan Ke
Genevieve Fried
Hugo Larochelle
Joelle Pineau

ML Checklists
Joelle Pineau
Robert Stojnic

NeurIPS 2019 Reproducibility Program
Joelle Pineau
Koustuv Sinha
Phillippe Vincent Lamarre
Vincent Lariviere
Alina Beygelzimer
Florence d’Alche-Buc
Emily Fox
Hugo Larochelle
Zenyu (Sherry) Xue
Hannah Wallach

ReScience
Nicolas P. Rougier
Konrad Hinsen
Olivia Guest
Benoit Girard

OpenReview
Pam Mandler
Melisa Bok
Mohit Uniyal
Michael Spector
Andrew McCallum

CodeOcean
Xu Fei

Thanks to many reviews from Joelle Pineau, Shagun Sodhani, Jessica Forde, Matthew Muckley and Michela Paganini from Facebook AI Research.

koustuv.sinha@mail.mcgill.ca
https://www.cs.mcgill.ca/~ksinha
@koustuvsinha