Deep Learning Research

Hands on

Deep Learning Research

Challenges

Challenges

- **Reproducibility**

Challenges

- Reproducibility
- Engineering Learning Curve

Challenges

- Reproducibility
- Engineering Learning Curve
- Analysis

Challenges

- Reproducibility
- Engineering Learning Curve
- Analysis
- Model Deploy

Reproducibility

Reproduce the previous experiment

Reproducibility

Reproduce **exactly** the previous experiment

Reproducibility

Reproduce an experiment:

- Code & Environment

Reproducibility

Reproduce an experiment:

- Code & Environment
- Random initialization

Reproducibility

Reproduce an experiment:

- Code & Environment
- Random initialization
- Training data

Reproducibility

Reproduce an experiment:

- Code & Environment
- Random initialization
- Training data NEW

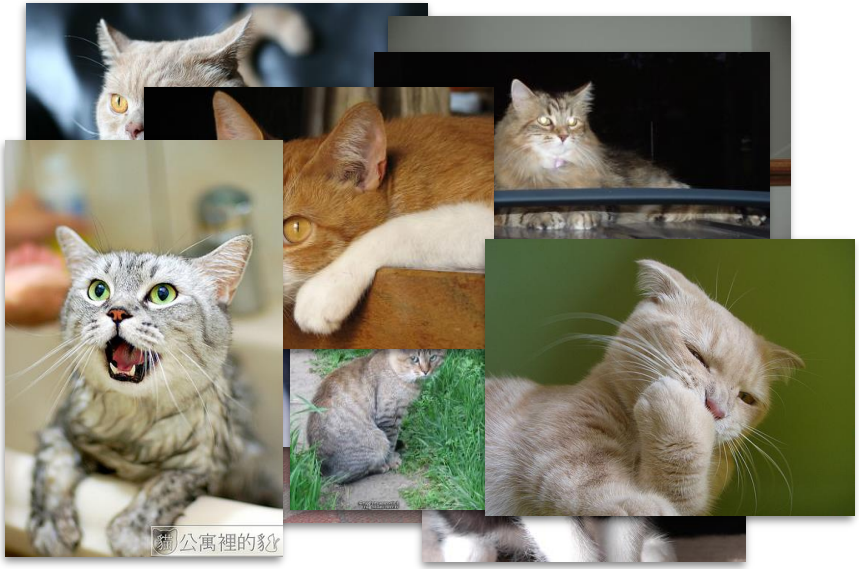
Code versioning

Pro Git book https://git-scm.com/book/it/v2

Getting Git Right https://www.atlassian.com/git

Data versioning

Dataset



0100
0011
1001

Trained models

model weights #1

model weights #2

Data versioning

Dataset

0100
0011
1001

Trained models

model weights #1

model weights #2

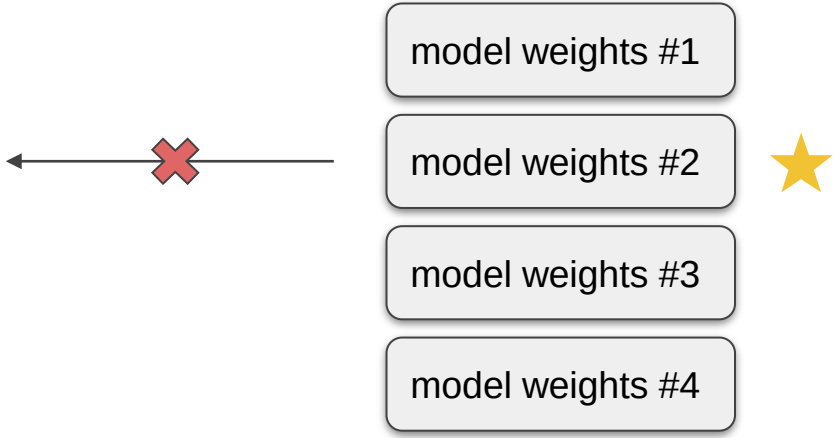
model weights #3

Data versioning

Dataset

0100
0011
1001

Trained models



...and model versioning

Origin of the means and stds used for preprocessing? #1439

Closed **pmeier** opened this issue on Oct 9, 2019 · 17 comments

---

pmeier commented on Oct 9, 2019 • edited Collaborator

Does anyone remember how exactly we came about the channel `mean`s and `std`s we use for the preprocessing?

```
transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
```

I think the first mention of the preprocessing in this repo is in #39. In that issue **@soumith** points to https://github.com/pytorch/examples/tree/master/imagenet for reference. If you look at the history of main.py the commit pytorch/examples@`27e2a46` first introduced the values. Unfortunately it contains no explanation, hence my question.

Specifically, I'm seeking answers to the following questions:

- Are these values `round`ed, `floor`ed, or even `ceil`ed?
- Did we use only the images in the training set of `ImageNet` or additionally the images of the validation set?
- Did we perform any kind of resizing or cropping on each image before the calculations were performed?

nizhib commented on Oct 10, 2019

You need to go deeper ;)

https://github.com/facebook/fb.resnet.torch/blob/master/datasets/imagenet.lua

```
-- Computed from random subset of ImageNet training images
local meanstd = {
   mean = { 0.485, 0.456, 0.406 },
   std = { 0.229, 0.224, 0.225 },
}
```

👍 7 | 😄 2

nizhib commented on Oct 10, 2019

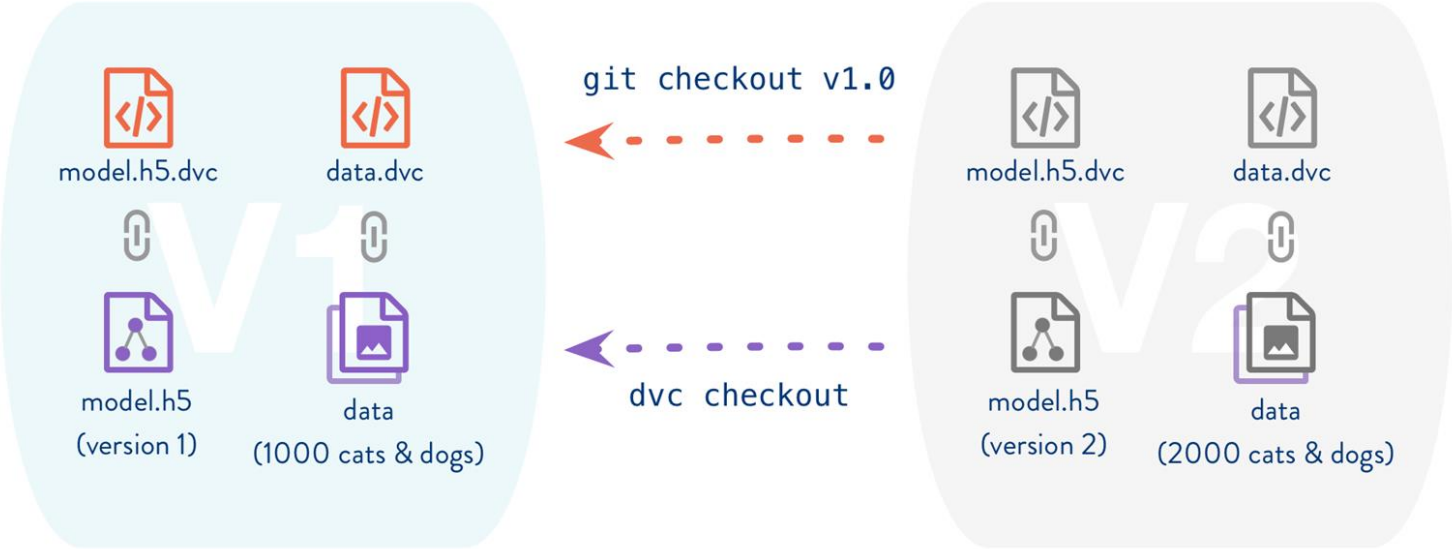
You need to go deeper ;)

Unfortunately, the concrete *subset* that was used is lost. For more information see this discussion or these experiments.

```
-- Computed from random subset of ImageNet training images
local meanstd = {
   mean = { 0.485, 0.456, 0.406 },
   std = { 0.229, 0.224, 0.225 },
}
```

👍 7 😄 2

Data versioning



Challenges

- Reproducibility
- Engineering Learning Curve
- Analysis
- Model Deploy

Engineering Learning Curve

- Manage configuration
- Hyperparameters tuning
- Organize codebase
- Decouple Data and Models
- Handle different types of hardware (CPU, GPUs, TPUs)
- Enforce determinism
- Logging to e.g. Wandb
- Checkpointing
- Resume previous trainings
- EarlyStopping
- ...

Manage configuration: Hydra

config.yaml

```yaml
db:
  driver: mysql
  user: omry
  password: secret
```

Manage configuration: Hydra

config.yaml

```yaml
db:
  driver: mysql
  user: omry
  password: secret
```

my_app.py

```python
from omegaconf import DictConfig, OmegaConf
import hydra

@hydra.main(config_path=".", config_name="config")
def my_app(cfg):
    print(OmegaConf.to_yaml(cfg))

if __name__ == "__main__":
    my_app()
```

Manage configuration: Hydra

config.yaml

```yaml
db:
  driver: mysql
  user: omry
  password: secret
```

my_app.py

```python
from omegaconf import DictConfig, OmegaConf
import hydra

@hydra.main(config_path=".", config_name="config")
def my_app(cfg):
    print(OmegaConf.to_yaml(cfg))

if __name__ == "__main__":
    my_app()
```

```
$ python my_app.py
db:
  driver: mysql
  user: omry
  password: secret
```

Manage configuration: Hydra

config.yaml

```yaml
db:
  driver: mysql
  user: omry
  password: secret
```

my_app.py

```python
from omegaconf import DictConfig, OmegaConf
import hydra

@hydra.main(config_path=".", config_name="config")
def my_app(cfg):
    print(OmegaConf.to_yaml(cfg))

if __name__ == "__main__":
    my_app()
```

```
$ python my_app.py db.user=root db.password=1234
db:
  driver: mysql
  user: root
  password: 1234
```

Manage configuration: Hydra

Directory layout

```
├── conf
│   ├── config.yaml
│   ├── db
│   │   ├── mysql.yaml
│   │   └── postgresql.yaml
│   ├── schema
│   │   ├── school.yaml
│   │   ├── support.yaml
│   │   └── warehouse.yaml
│   └── ui
│       ├── full.yaml
│       └── view.yaml
└── my_app.py
```

Code Organization: PyTorch Lightning

A powerful framework to:

- Organize your code

PyTorch Lightning

Code Organization: PyTorch Lightning

A powerful framework to:

- Organize your code
- Automate most of the engineering code

PyTorch Lightning

PYTORCH

```
# models
self.encoder = nn.Sequential(nn.Linear(28 * 28, 64), nn.ReLU(), nn.Linear(64, 3))
self.decoder = nn.Sequential(nn.Linear(3, 64), nn.ReLU(), nn.Linear(64, 28 * 28))

encoder.cuda(0)
decoder.cuda(0)

# download on rank 0 only
if global_rank == 0:
    mnist_train = MNIST(os.getcwd(), train=True, download=True)

# split dataset
transform=transforms.Compose([transforms.ToTensor(),
                              transforms.Normalize(0.5, 0.5)])
mnist_train = MNIST(os.getcwd(), train=True, download=True, transform=transform)

# train (55,000 images), val split (5,000 images)
mnist_train, mnist_val = random_split(mnist_train, [55000, 5000])

# The dataloaders handle shuffling, batching, etc...
mnist_train = DataLoader(mnist_train, batch_size=64)
mnist_val = DataLoader(mnist_val, batch_size=64)

# optimizer
params = [encoder.parameters(), decoder.parameters()]
optimizer = torch.optim.Adam(params, lr=1e-3)

# TRAIN LOOP
model.train()
num_epochs = 1
for epoch in range(num_epochs):
  For train_batch in mnist_train:
    x, y = train_batch
    x = x.cuda(0)
    x = x.view(x.size(0), -1)
    z = encoder(x)
    x_hat = decoder(z)
    loss = F.mse_loss(x_hat, x)
    print('train loss: ', loss.item())

    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

# EVAL LOOP
model.eval()
with torch.no_grad():
  val_loss = []
  for val_batch in mnist_val:
    x, y = val_batch
    x = x.cuda(0)
    x = x.view(x.size(0), -1)
    z = encoder(x)
    x_hat = decoder(z)
    loss = F.mse_loss(x_hat, x)
    val_loss.append(loss)
  val_loss = torch.mean(torch.tensor(val_loss))
  model.train()
```

PYTORCH LIGHTNING

Turn PyTorch into Lightning

Lightning is just plain PyTorch

Challenges

- Reproducibility
- Engineering Learning Curve
- Analysis
- Model Deploy

Analysis

Log everything:

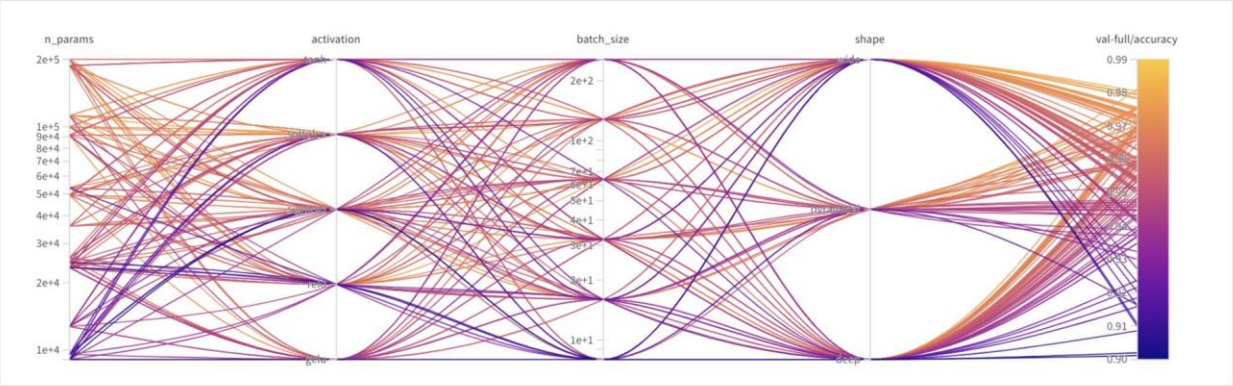
- Hyperparameters
- Losses
- Metrics
- Project-specific data (e.g. generated images)

...then compare and analyze

Sweep: shape_sweep

Results of Hyperparameter Sweep 5

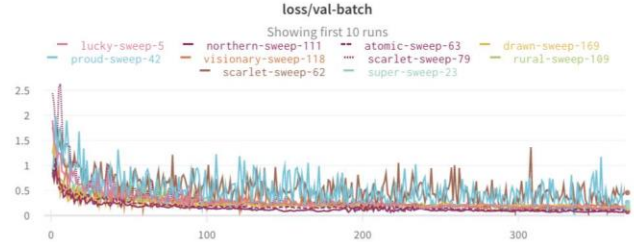
Name (156 visual	val-full/a
lucky-sweep	0.981
northern-sv	0.9804
atomic-swe	0.9795
drawn-swee	0.9793
proud-swee	0.9784
visionary-sv	0.9781
scarlet-swe	0.976
rural-sweep	0.9756
scarlet-swe	0.9756
super-swee	0.9754
morning-sw	0.9754
prime-swee	0.9752
fresh-sweep	0.9752
rose-sweep	0.9751
rose-sweep	0.9748
easy-sweep	0.9747
easy-sweep	0.9737
toasty-swee	0.9734
ruby-sweep	0.9733



Parameter importance with respect to val-full/accuracy

Search | Parameters | Rows per page 10 | 1-10 of 18

Config parameter	Importance	Correlation
n_params		
budget		
parameter_budget		
Runtime		
lr		
batch_size		
activation.value_relu		
activation.value_tanh		
activation.value_sig...		
shape.value_deep		



Best Model by Val Acc

lucky-sweep-5

0.981

Weights & Biases

Challenges

- Reproducibility
- Engineering Learning Curve
- Analysis
- **Model Deploy**

Model Deploy

- In production
- Showcase with a demo

Model Deploy

- ~~In production~~
- Showcase with a demo

Model Deploy

Examples:

- Odeen: https://huggingface.co/spaces/gladia/odeen
- NN Expressivity: https://share.streamlit.io/lucmos/demo-nn-expressivity/main/ui.py
- Bayesian Opt: https://share.streamlit.io/lucmos/demo-bayesian-optimization/main/main.py

Tooling Recap

Tooling Recap (Examples)

Code & data versioning

Code configuration

Code organization

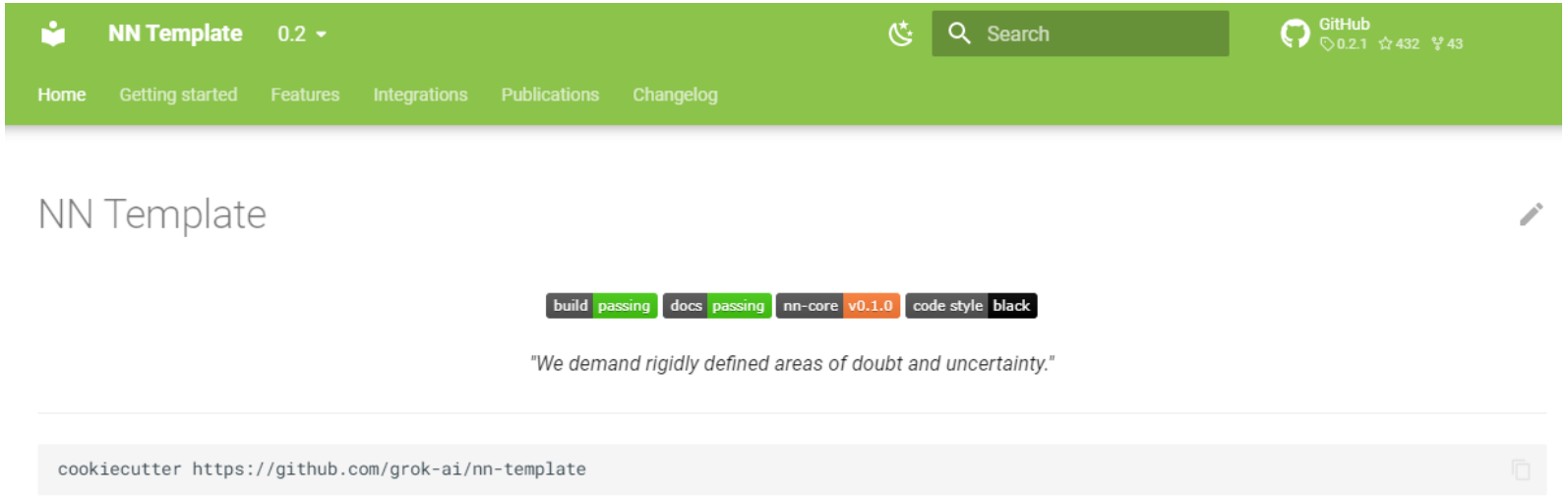
Experiments logging and analysis

Interactive demos

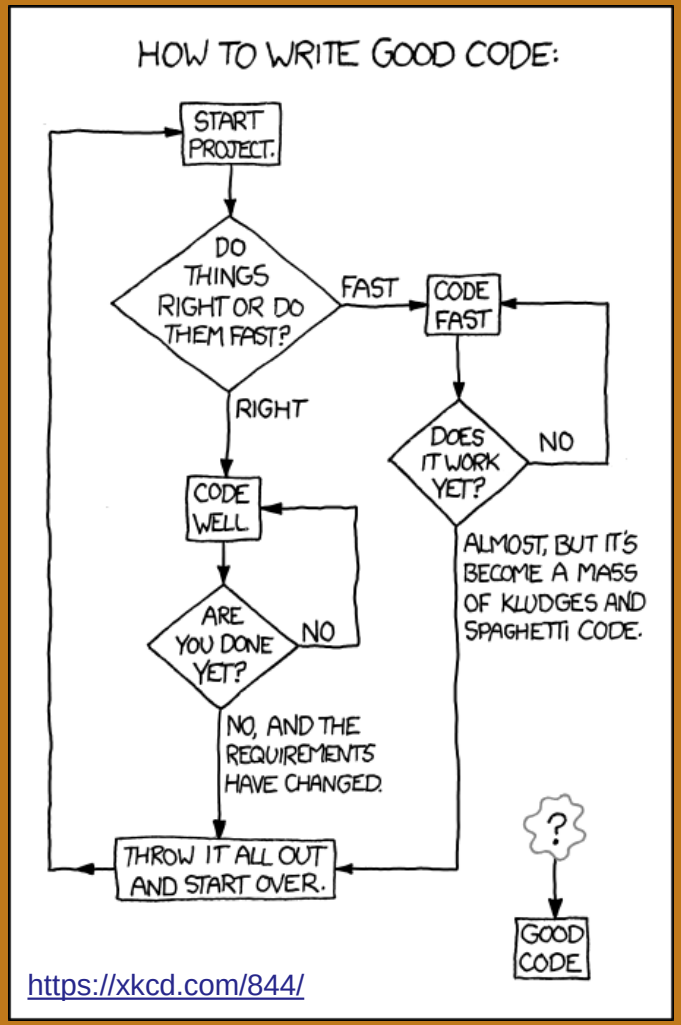
Code Boilerplate

Avoid Code Boilerplate: NN Template

https://grok-ai.github.io/nn-template



Any contribution is welcome!



https://xkcd.com/844/

Questions?