Project Proposals

Group name and members: boinet. Lisa Wimmer, Sven Lorenz, Andreas Klaß.

We are not sure which proposal is best suited as a final assignment, so we include all to have fallback options. So far, our preferences in terms of a ranking are:

1. Top-KAST
2. Uncertainty Surrogates
3. Generic Active Learning

Project Proposal 1  
Top-KAST: Top-K Always Sparse Training

Top-KAST is a generic method that preserves constant sparsity throughout training (in both the forward and backward-passes).

Group name and members: boinet. Lisa Wimmer, Sven Lorenz, Andreas Klaß.

## Paper in a nutshell (<https://arxiv.org/pdf/2106.03517v1.pdf>)

* Motivation: larger sparse networks outperform smaller dense networks[[1]](#footnote-1)
* Idea: select a subset of parameters A ⊂ Θ that correspond to the top-K parameters (by magnitude) for each training step. Apply gradients to a larger parameter subset B ⊃ A (with B ⊂ Θ).
* Method never requires:
  + a forward pass with dense parameters
  + calculation of a dense gradient
* 3 minute summary in a video: <https://crossminds.ai/video/top-kast-top-k-always-sparse-training-606fef9cf43a7f2f827c1681/>

## **Define** the different **building blocks** that need to be implemented

* Macro procedure *(Person A)*:
  + exploratory stage with changing A sets
  + refinement stage with fixed A
* Forward Pass *(Person B)*:
  + Determine set of active weights A
  + Top K selection operation which is applied per layer
    - compute the Top-K entries in parallel on CPU
    - apply the mask every 100 steps
  + Constructing alpha parameter / selection process periodically
* Backward Pass *(Person C)*
  + Compute gradient vector to update weights in superset B ⊃ A
* Auxiliary exploration regularization loss (to encourage the masking to stay adaptable rather than fixating on small suboptimal sparse subsets) *(Person A)*
* Initialization: A as a random subset of weight-indices from θ *(Person B)*

## Clearly define what **requirements** the final **code** must fulfill

* Must work like any optimizer from torch.optim.
* Execute a truly sparse training procedure. Custom sparse kernels are mentioned in appendix D?..
* Training method is generic and should be applicable to at least the benchmark architectures named in the following chapter

## How will the **framework** be **tested**

1. It is yet unclear what unit / integration tests to include.
2. By reproducing the paper results:
   * Apply sparse ResNet50 on ImageNet with the same specifications as described in the paper.
   * If need be: reproduce paper results for a transformer architecture.
   * If need be: Something of our own.  
     Trying out more architectures is possible, but one or two should suffice.

## What do we start with? What code already exist?

* <https://github.com/google-research/rigl> (Tensorflow); similar method which the authors mention
* <https://github.com/varun19299/rigl-reproducibility>; reproduces rigl (=rigging the lottery) and includes (at least) many helper functions / wrappers which we could exploit.
* Pseudocode from appendix

## Particularly **challenging** parts:

* Rewriting the training procedure could become quite low-level => deep dive into the .optimize() and .train() methods from Pytorch
* Writing generic forward-backward passes that can be applied everywhere is challenging.
* Compatibility with generic Pytorch modules is not trivial.

## Main contribution

* Implementation of a generic true sparse-to-sparse training
* We reproduce the results from the paper
* (Could be added to the pytorch library as a new layer module?)

## Further related work

<https://cs.stanford.edu/~matei/papers/2020/sc_sparse_gpu.pdf>

https://www.nature.com/articles/s41467-018-04316-3

## Appendix

Pseudocode:

Algorithm 1 TopKAST ####################

// First perform a Top-K

dense\_params = initialise()

fwd\_params = TopK(dense\_params, X%)

bwd\_params = TopK(dense\_params, Y%)

just\_bwd\_set = set(bwd\_params) - set(fwd\_params)

...

// Output with just the TopK

params output = model(fwd\_params, input)

loss = loss\_fn(output)

// Exploration L2 Loss

loss += l2(fwd\_params) + l2(just\_bwd\_set) / (X/100)

...

// Update only the bwd params

bwd\_params = bwd\_params - grad(loss, bwd\_params)

Project Proposal 2  
Uncertainty Surrogates

Uncertainty surrogates are a simple, inexpensive method for assessment and visual inspection of an image classifier’s uncertainty estimate.

Group name and members: boinet. Lisa Wimmer, Sven Lorenz, Andreas Klaß.

## Paper in a nutshell (<https://arxiv.org/pdf/2104.08147.pdf>)

* Motivation: uncertainty quantification indispensable, but often difficult to implement and potentially costly (and not straightforward to measure and/or interpret)
* Idea: force the network to output in the penultimate layer a feature representation that resembles a pre-defined visual pattern indicative for the respective class label (e.g., a grayscale image spelling the class name)
  + Higher uncertainty visible in deviation from the desired pattern
  + Uncertainty quantifiable via pixel-wise errors w.r.t. the desired pattern (via MSE / cross-entropy loss or – better – a secondary, lightweight CNN)
* Easily integrated as an additive ingredient to the loss function (and thus also acting as a regularizer)

## **Define** the different **building blocks** that need to be implemented

* Construction of basic net architecture *(Person A)*
* Definition of surrogate patterns *(Person A)*
  + Helping confident discrimination as much as possible
  + Depending on size, number of channels, pattern complexity
* Incorporation of surrogate modeling *(Person B)*
  + Extraction of last-hidden-layer representation
  + Definition of custom loss function
  + Adaption of backpropagation
* Computation of uncertainty estimates at test time *(Person B)*
* Experiments *(Persons A & C)*
  + Simulation of epistemic uncertainty via OOD examples (same domain / different domain)
  + Simulation of aleatoric uncertainty via label noise
  + Adversarial attacks
* Potential enhancements *(Person C)*
  + Tuning / scheduling of regularization coefficient
  + Explicit incorporation of adversarial training
  + Special punishment for confidently wrong predictions
  + Uncertainty metric as early stopping criterion

## Clearly define what **requirements** the final **code** must fulfill

* Forward pass must produce twofold output: class label & surrogate pattern prediction
* Prediction on test data must yield class label & surrogate pattern prediction as well as uncertainty estimate
* Pattern layer must be applicable on top of generic CNN architecture
* Training and backpropagation must be compatible with standard optimization routines

## How will the **framework** be **tested**

* Unit test: pattern layer must output pattern prediction of pre-defined size, given some input image
* Overall test: (partly) reproduce paper experiments
  + MNIST for grayscale images
  + CIFAR10 for color images

## What do we start with? What code already exist?

* Basic network architecture (e.g., PreActResNet18 as in experiments)

## Particularly **challenging** parts:

* Finding meaningful surrogate patterns
* Incorporating regularization term into loss function and backward propagation

## Main contribution

* Simple, inexpensive means of uncertainty quantification
* Visual assessment of uncertainty estimates

Project Proposal 3  
SIMILAR: Submodular Information Measures Based Active Learning In Realistic Scenarios

Group name and members: boinet. Lisa Wimmer, Sven Lorenz, Andreas Klaß.

## Paper in a nutshell

* <https://arxiv.org/pdf/2107.00717v1.pdf>
* Use submodular information measures as acquisition functions in active learning
  + Handle different modes by using relationship between SIM

## **Define** the different **building blocks** that need to be implemented

1. Standard training loop for model
2. Similarity Matrix with hypothesized labels + data handling (simulating rare classes, ood etc.)
3. Use different modes of Submodular Functions on Similarity Matrix

## Clearly define what **requirements** the final **code** must fulfill

* Implement different general SIMILAR framework
* Different implemented SMI Functions
* Handle different situations (standard AL, rare classes, out of distribution data and data redundancy)

## How will the **framework** be **tested**

* Test SIMILAR framework against paper results
* Model can be tested against other pretrained models

## What do we start with? Which code parts already exist?

* Standard AL mode similar to BADGE: <https://github.com/JordanAsh/badge>
  + Some computational tricks are the same as well

## Particularly **challenging** parts:

* Make sure computationally feasible using tricks in paper
* Get different modes correctly
* Reduce code redundancies by using Appendix Table 1

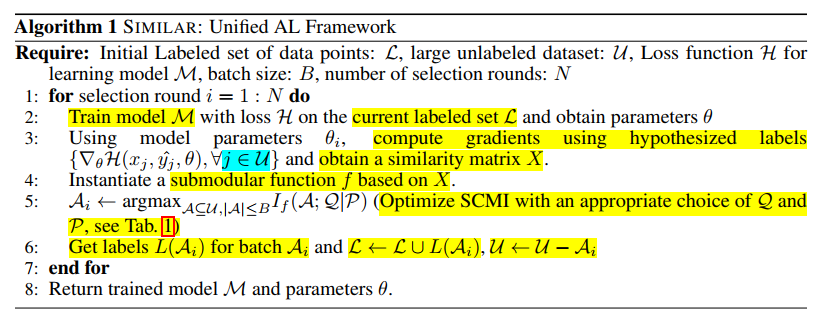
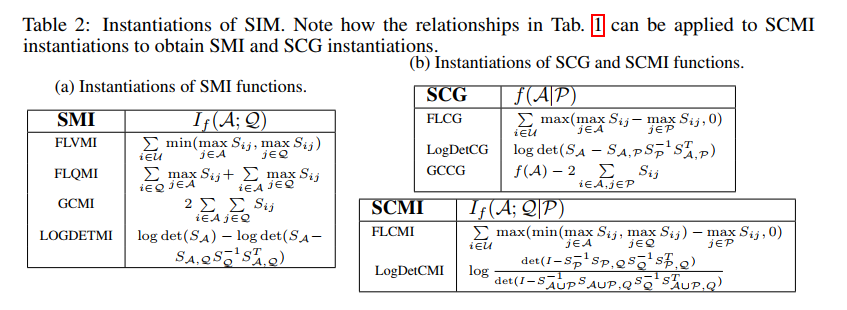
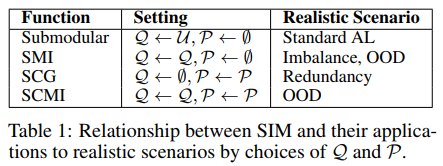
## Main contribution

* Implementing unified active learning approach.

## Related works and references

* <https://github.com/JordanAsh/badge>
* <https://github.com/decile-team/distil>
* https://github.com/gudovskiy/al-fk-self-supervision

## Appendix

1. Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. In International Conference on Machine Learning (ICML), 2018 [↑](#footnote-ref-1)