Project Proposal  
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

* 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

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

* Macro procedure:
  + exploratory stage with changing A sets
  + refinement stage with fixed A
* Forward Pass:
  + 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
  + 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 sparse subsets)
* Initialization: A as a random subset of weight-indices from θ

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

* Supposed to work with:
* Accept which inputs:
* Unit tests for building blocks
* Execute a truly sparse training procedure.

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

By reproducing the paper results:

* Apply sparse ResNet50 on ImageNet with the same specifications as described in the paper.

We Something of our own. Further “architecture tests” are possible, but two should suffice. We would like to start with a CNN.

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

* <https://github.com/google-research/rigl> (Tensorflow); similar method which the authors mention
* https://github.com/varun19299/rigl-reproducibility
* Pseudocode from appendix

## Particularly **challenging** parts:

* Rewriting optimizers directly is very low-level
* Writing generic forward-backward passes that can be applied everywhere is challenging. Compatibility with generic modules is not trivial.
* Figuring what truly sparse means

## Main contribution

* We implement a
* EIGENLEISTUNG
* We reproduced the results from the paper
* Could be added to the pytorch library as a new layer module or option

## Related works and references

<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)

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)