Project report

# Abstract:

CNNs have existed for many years, and with time their size has increased substantially, with recent models reaching hundreds of billions of parameters. Those models are computationally expensive, and take a long time to run after training, making some of them hard to use efficiently. Our study suggests a method to prune such networks using an approximation of the Recoverability metric (TODO: Cite the recoverability paper) to decide which weights (and layers?) can be nullified. By doing so we make the computation faster while minimally hurting the performance. TODO: Results and future work.

# Intro:

The Lottery Ticket Hypothesis (Cite) suggest that neural networks contain subnetworks, that when trained in isolation can reach results as good as the original networks, while reducing the required storage, training time and computation time. In order to find these subnetworks, we would like to use pruning techniques (cite?) which allow us to take a trained networks and start removing redundant weights and even complete layers. Right now, many pruning methods exist and are proven to work, but we would like to try a new pruning method which relies on the Recoverability metric, or at least an approximation of it. Given two vectors and , the Recoverability is defined by , where is a continuous function. Since there are infinitely many continuous functions, we decided to limit the group of functions which f can be chosen from, and then perform calculate the approximated metric exactly.

We believe that by calculating how recoverable a layer’s output is from the input, we will be able to find layers in which certain weights are redundant to the calculation, thus prunable.

# Method:

Using our approximation, we are able to estimate how recoverable is from on all functions in which a single weight is pruned. If we find that the recoverability is high, we zero the weight which we found to be least important repeat the process until the Recoverability reaches a certain threshold that we defined as a hyperparameter.

## Basic pruning method:

As a basis we decided to try and find a way to prune weights in a specific layer when given a specific input vector . The method works as follows:

1. Start with the Layer , input and vector .
2. Define hyperparameters .
3. Define , .
4. Define .
5. Pick , such that is minimal.
6. Save and in a list.
7. If , set and return to step 4.
8. Calculate and return .

The main idea is that we try to find a balance between how much we hurt the performance and how many weights we zero. This balance is determined by the hyperparameter . A high value of means that we prefer zeroing more weights. Since we have no reason to try and zero every single weight, we decided that we need a threshold that once we reach, we stop the algorithm and return the best result found so far. This threshold is defined by the hyperparameter . A higher means we would like to search deeper.

This pruning method is problematic since a single input vector is not a good representative of how important a specific weight is. For that reason we decided to not only consider one image, but many, and thought of the Batch Pruning Method.

## Batch pruning method:

In this method we prune weights in a specific layer when given a batch of vectors . The method works as follows:

1. Start with the Layer , input .
2. Define hyperparameters .
3. Define , .
4. Define .
5. Pick , such that is minimal.
6. Save and in a list.
7. If , set and return to step 4.
8. Calculate and return .

Now we consider multiple images while pruning and get a much better estimate of the Recoverability metric.

# Experiments:

# Conclusion:

Does not work, run as far as possible.

# Future work:

Our pruning method worked in a greedy manner in order to save calculation time and resources, but it most definitely is not perfect. Using a better search method can lead to better results.

Since our research was rather small, we have only shown how the performance changed as a function of the number of weights zeroed. We did not try to scale down the model based on those finding, which we believe is the next step that should be taken when exploring the capabilities pf this pruning method.

# References:

# Weight pruning method:

1. Start with the Layer , input and vector .
2. Define hyperparameters .
3. Define , .
4. Define .
5. Pick , such that is minimal.
6. Save and in a list.
7. If , set and return to step 4.
8. Calculate and return (Rethink this entire algorithm).

Time complexity: , where is the number of weights in , is the maximal pruning we try, and using no parallelization.

Problems:

1. Taking the best step in any iteration and excluding all other steps.
2. The search might not lead to optimal results.
3. Relies on a hyperparameter which we don’t know it’s optimal value.
4. Using an approximation of the recoverability (Not necessarily a bad thing).
5. The argmax function might be a bad fit.
6. Only using 1 X. Possible solution: Choose n random vectors and calculate the distance on all of them, then aggregate (Using average, worst case or voting).

# Questions:

1. Do we need to prune individual weights, layers or both?
2. Ask about the 1000 layer network.
3. Ask for explanation about the kernel method and RKHS. (Previously said not required)
4. Ask for practical help with recoverability. (Show what we have already)
5. Maybe help define the project better.
6. Ask for intuition when checking recoverability of complete layers, do we want to check the recoverability between two constant spots (Like start and end of the model) or on changing spots?

# Sources:

ResNet implementation - <https://towardsdatascience.com/residual-network-implementing-resnet-a7da63c7b278>.

CIFAR-10 turorial - <https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html>.