Representation Aware Pruning with Centered Kernel Alignment

Calvin Higgins

May 2023

1 Introduction

Modern pruning techniques can substantially reduce the number of parameters in neural networks while maintaining their accuracy [1]. Most existing pruning algorithms rely on simple heuristics such as weight magnitude to identify redundant parameters [1]. However, more sophisticated heuristics based on representational similarity measures have not been thoroughly investigated. The latter approach could enable more aggressive pruning resulting in reduced compute requirements during inference.

To explore this area, we propose a novel greedy algorithm for pruning neuron weights. The method involves removes and restores each neuron in a given layer and measuring the similarity between layer activations before and after each removal. The neuron whose elimination results in maximum similarity is pruned. Intuitively, this preserves the representations of the network during pruning.

We evaluate our proposed algorithm against a typical weight magnitude heuristic and find that our approach offers marginal improvements at high rates of pruning (that is, when accuracy is significantly impacted by pruning). Unfortunately, these gains are overshadowed by the significant computational cost of our algorithm.

2 Background

2.1 Centered Kernel Alignment

Centered kernel alignment (CKA) with a linear kernel is a measure of similarity between representations of neural networks [2]. CKA is invariant to orthogonal transformation and isotropic scaling. It is a normalized version of the Hilbert-Schmidt Independence Criterion (HSIC). HSIC is defined as

$$\text{HSIC}(K, L) = \frac{1}{(n-1)^2}\text{tr}(KHLH)$$

where $H_n = I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^T$. Using HSIC, CKA is defined as

$$\text{CKA}(K, L) = \frac{\text{HSIC}(K, L)}{\sqrt{\text{HSIC}(K, K)\text{HSIC}(L, L)}}$$

where $K = XX^T, L = YY^T$ and X, Y are activation matrices. That is, each row of X, Y contain the activations of a layer given some input. Note that X, Y have maximum similarity when $\text{CKA}(K, L) = 1$ and minimum similarity when $\text{CKA}(K, L) = 0$.

2.2 Pruning

Pruning is a technique for reducing the size and complexity of neural networks by eliminating redundant parameters [1]. There are two major types of pruning methods: unstructured and structured pruning. Unstructured pruning removes individual parameters whereas structured pruning removes logical groups of parameters [1]. For example, in unstructured pruning, a single connection to a neuron may be removed, whereas, in structured pruning, the entire neurons may be removed. Unstructured pruning typically results in sparser pruned networks than structured pruning. Unlike structured pruning, unstructured pruning does not produce performance gains since the weight matrices cannot be shrunk.

Pruning is typically applied at a fixed rate and can be performed locally or globally [1]. Local pruning prunes each layer individually while global pruning prunes the network as a whole [1]. For example, local pruning may remove 20% of the parameters from each layer of the network but global pruning may remove 30% of the parameters from the first layer and 10% of the parameters from the second layer.

Pruning can be applied in one-shot or iteratively [1]. In iterative pruning, the network is retrained before applying another pruning pass to remove more parameters [1]. This technique often produces smaller pruned networks compared to one-shot pruning.

3 Methodology

Dataset and training. In our experiments, we utilized the MNIST dataset with a 55000-5000-10000 train-validation-test split. The images were normalized with zero mean and unit standard deviation. All networks were trained with Adam ($\gamma = 0.001, \beta_1 = 0.9$ and $\beta_2 = 0.999$) for maximum of 50 epochs and early stopping on validation loss (three epoch patience) with a batch size of 512. The learning rate was determined via hyperparameter search with WandB. Unless otherwise specified, 50% dropout was applied to the hidden layers.

L1 pruning. L1 pruning was used as a baseline for all experiments. In L1 pruning, weights with low magnitude are pruned. Intuitively, low magnitude weights have low importance. We used the following algorithm:

1. Do the following for each layer.
   (a) Compute the L1 norm of all the neuron weights.
   (b) Prune the weights of the $p\%$ of neurons with the lowest L1 norm of their weights.
2. Train the network to convergence.
3. Repeat for j iterations.

CKA pruning. We designed a greedy algorithm to prune neuron weights with CKA. Intuitively, CKA was used determine the impact of removing any neuron on the layer's representation and neurons with the lowest impact were removed. Note that our algorithm takes $\Omega(n^2)$ time where n is the number of neurons in the layer.

1. Compute the activations of each layer with a mini-batch of 512 samples.
2. Do the following for each layer.
   (a) Do the following for each neuron in the layer.
      i. Set the neuron weights to zero and recompute the layer activation.
      ii. Compute the similarity between the original layer activation and the new layer activation with CKA.
      iii. Restore the weights of the neuron.
   (b) Prune the weights of the neuron whose removal resulted in maximum similarity.
   (c) Repeat until $p\%$ of the neurons in the layer have been pruned.
3. Train the network to convergence.
4. Repeat for j iterations.

4 Experiments

4.1 Iterative Pruning

We considered pruning a standard fully connected neural network with two hidden layers (512 neurons each). We trained the network from nine different random initializations with 50% dropout applied to the hidden layers. Then, the hidden layers of each network were iteratively pruned with $p = 0.2$ for 15 iterations with both L1 and CKA.

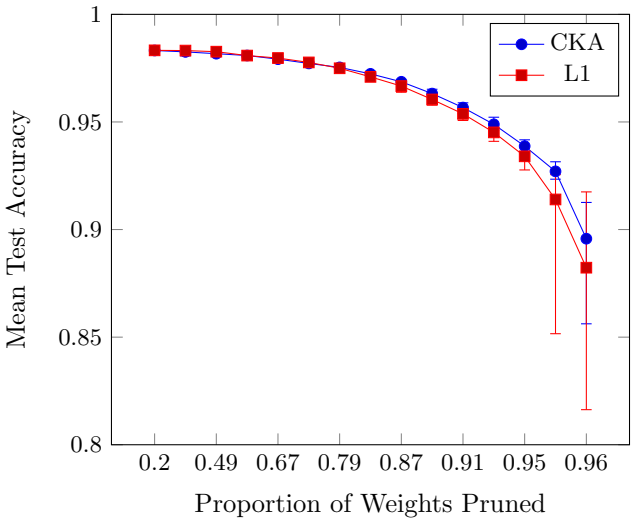

Figure 1: Averaged over 9 trials. Error bars are minimum and maximum values (per-iteration).

As the proportion of pruned weights increases, CKA pruning achieves higher mean, maximum and minimum test accuracy than L1 pruning. This effect is more pronounced at later iterations suggesting that CKA and L1 pruning may agree on which neurons to prune in early iterations.

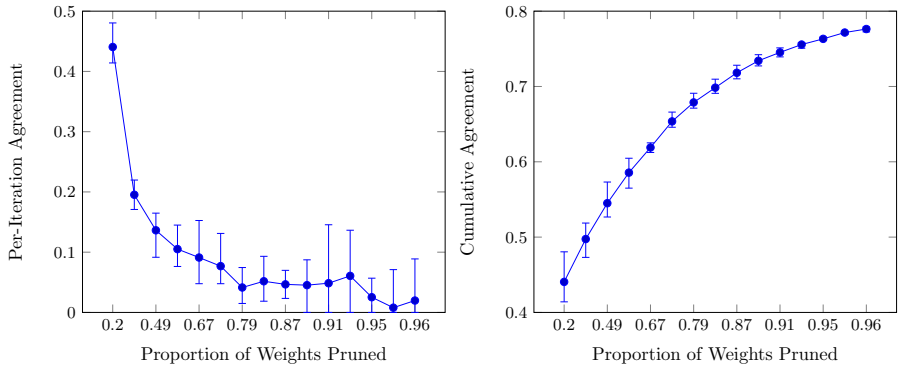

Figure 2: Averaged over 9 trials. Error bars are minimum and maximum values (per-iteration). Per-iteration agreement is the proportion of neurons selected for pruning by both CKA and L1 on the given iteration. Cumulative agreement is the proportion of neurons selected for pruning by both CKA and L1 across the current and all prior iterations.

In fact, this is exactly what we observe. In the first iteration, around 50% of neurons are pruned by both CKA and L1. By the third iteration, this falls to

almost 10%. Across all iterations, the majority of pruned neurons are pruned by both CKA and L1. This indicates that there might be critical neurons that are removed by L1 pruning and not removed by CKA pruning during later iterations.

4.2 Varying Dropout

We considered how varying the dropout rate impacts the relative performance of CKA and L1 pruning. We trained same architecture as before (two hidden layers, 512 neurons each) with 0%, 10%, 30%, 50% and 70% dropout on the hidden layers from nine different random initializations. Then, the hidden layers of each network were iteratively pruned with $p = 0.2$ for 15 iterations with both L1 and CKA.

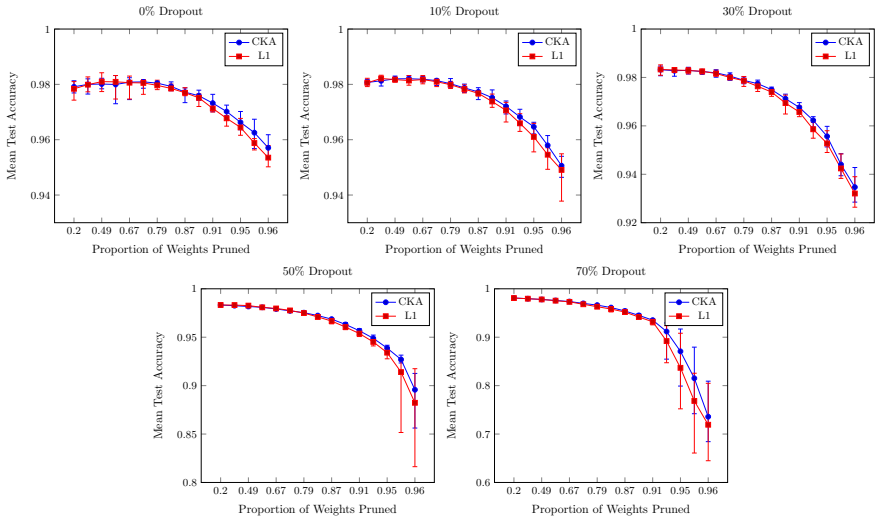

Figure 3: 50% dropout averaged over 9 trials. Other dropout rates averaged over 10 trials. Error bars are minimum and maximum values (per-iteration).

High dropout rates improved the relative accuracy of CKA pruning. Since high dropout rates lead to an effectively small network size during training, the accuracy boost could have been because CKA pruning performs better than L1 pruning when most of the network has been pruned.

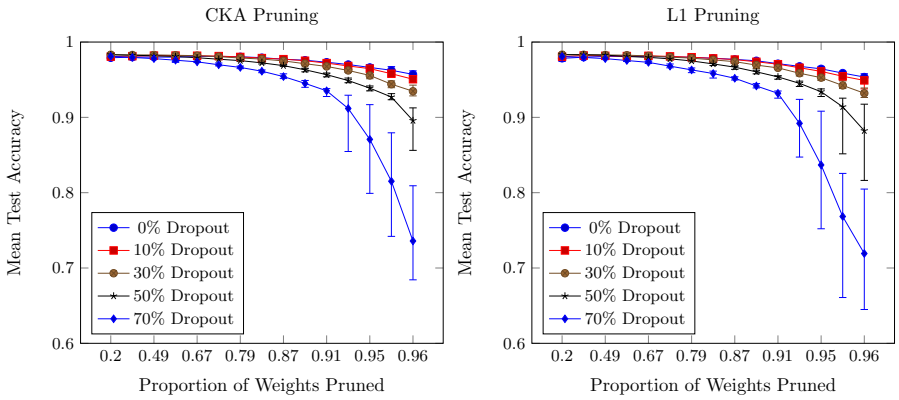

Figure 4: 50% dropout averaged over 9 trials. Other dropout rates averaged over 10 trials. Error bars are minimum and maximum values (per-iteration).

Interestingly, higher rates of dropout appear to harm both CKA pruning and L1 pruning. This might be because highly pruned networks become more sensitive to dropout. This suggests that dropout rates should be reduced as the number of pruning iterations increases.

4.3 Varying Width

We investigated how varying the hidden layer width impacts the relative performance of CKA and L1 pruning. We trained fully connected neural networks with two hidden layers with 128, 256, 512, or 1024 neurons. There was 50% dropout applied to the hidden layers. Then, the hidden layers of each network were iteratively pruned with $p = 0.2$ for 15 iterations with both L1 and CKA.

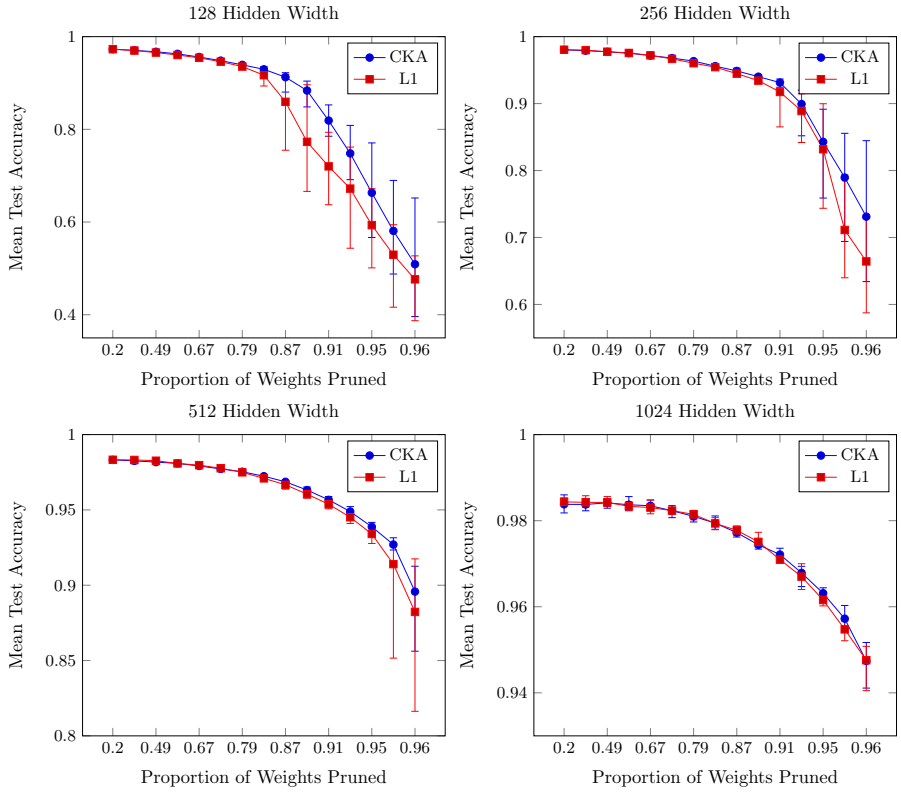

Figure 5: 512 hidden width averaged over 9 trials. Other hidden widths averaged over 10 trials. Error bars are minimum and maximum values (per-iteration).

Note that CKA pruning once again exhibits superior performance exclusively for smaller network sizes. The magnitude of the advantage appears to be inversely proportional to the network size. For example, CKA pruning significantly outperforms L1 pruning for 128 hidden width network but offers no benefits for the 1024 hidden width network.

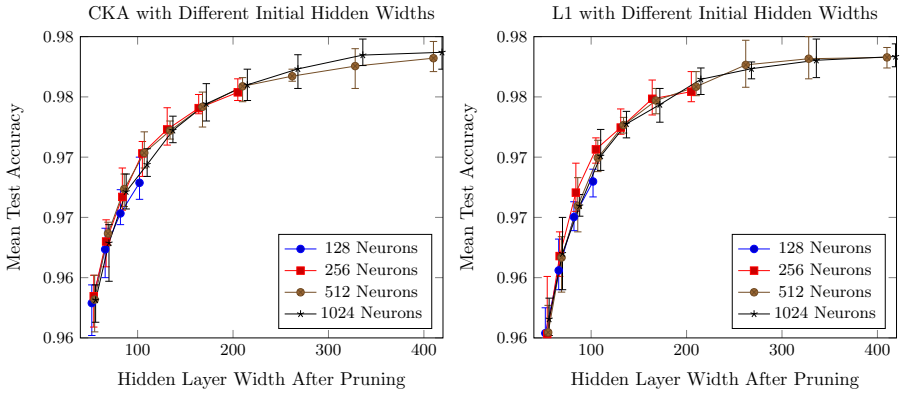

Figure 6: 512 hidden width averaged over 9 trials. Other hidden widths averaged over 10 trials. Error bars are minimum and maximum values (per-iteration).

Networks with large initial hidden widths do not produce more accurate pruned networks of a given size. This suggests that overparameterization does not complement CKA and L1 pruning.

4.4 Varying Depth

We investigated how varying network depth impacts the relative performance of CKA and L1 pruning. We trained fully connected neural networks with one, two and three hidden layers with 512 neurons and 50% dropout. Then, the hidden layers of each network were iteratively pruned with $p = 0.2$ for 15 iterations with both L1 and CKA.

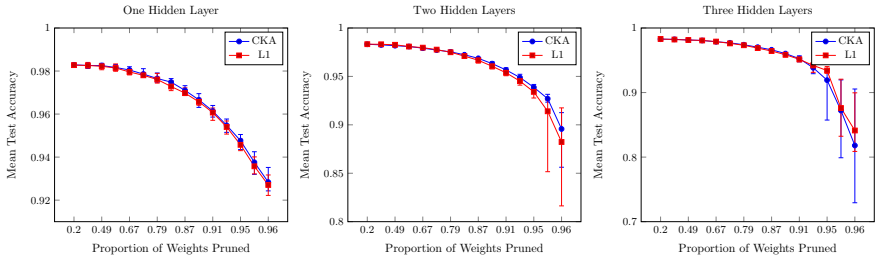

Figure 7: Two hidden layers averaged over 9 trials. Other depths averaged over 10 trials. Error bars are minimum and maximum values (per-iteration).

Surprisingly, L1 pruning outperformed CKA pruning for the three hidden layer network when most of the network had been pruned. This could have been because layer activations were computed once per pruning iteration so representation changes due to pruning in early layers were not accounted for by

pruning in later layers.

Additionally, the accuracy of deeper pruned networks is poor relative to shallower networks. This suggests that the pruned networks might be trapped in local minimia. Pruning might benefit from some sort of random re-initialization.

4.5 Varying Pruning Rate

We investigated how varying the pruning rate p impacts the relative performance of CKA and L1 pruning. We trained fully connected neural networks two 512 neuron hidden layers and 50% dropout. Then, the hidden layers of each network were iteratively pruned with $p = 0.1$, $p = 0.2$, $p = 0.3$ and $p = 0.4$ for 15 iterations with both L1 and CKA.

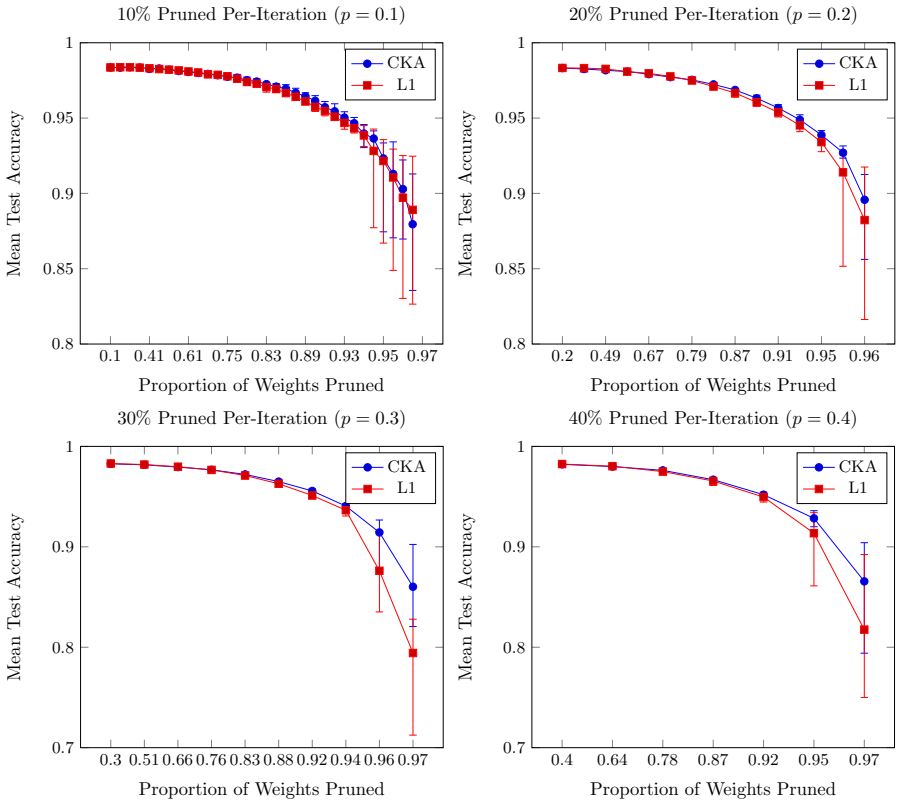

Figure 8: $p = 0.2$ averaged over 9 trials. Other rates averaged over 10 trials. Error bars are minimum and maximum values (per-iteration).

Again, CKA pruning produced more accurate pruned networks than L1 pruning when most of the network was pruned. This advantage became more significant

at high rates of pruning. This suggests that CKA pruning is able to preserve network accuracy even at aggressive pruning schedules. Such schedule enables compute savings during pruning as it avoids retraining the network several times.

5 Conclusions

CKA pruning offers marginal benefits over L1 pruning at high pruning rates and for heavily pruned networks. This suggests that CKA pruning is able to identify important neurons with low weight magnitudes. However, the $\Omega(n^2)$ complexity of CKA pruning renders it infeasible for real-world workloads. Nonetheless, alternative representation aware pruning algorithms could offer similar advantages at a more reasonable computational cost.

6 Limitations and Future Work

This work focused on training a limited number of architectures on a single dataset and pruning with only one measure of representational similarity. Further research could explore if our findings hold for other architectures and datasets, investigate the potential of CKA to determine the optimal pruning rate p, and search for more computationally efficient algorithms.

References

[1] Jonathan Frankle and Michael Carbin. *The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks*. 2019. arXiv: 1803.03635 [cs.LG].

[2] Simon Kornblith et al. *Similarity of Neural Network Representations Revisited*. 2019. arXiv: 1905.00414 [cs.LG].

A Hyperparameter Search

We swept $lr \in \{0.1, 0.01, 0.001, 0.0001, 0.00001\}$ with WandB for all models and found that $lr = 0.001$ was always optimal.

B Representation Aware Pruning Rate

We tried pruning until the similarity score fell below a certain threshold for both CKA and L1 pruning. We found that this process was highly sensitive to threshold choice. If the threshold was too low, no neurons were pruned. If the threshold was too high, almost all neurons were pruned. We failed to find any reasonable threshold values.

C Implementation

All code can be found at `https://github.com/SupurCalvinHiggins/cka-prune`.