# Variants of Minimal Effort Backpropagation (meProp) on a Feedforward Neural Network

Enrique De Alba (edealba), Nithin Kannan (nkannan), Young Han Kim (yhkim99)

## Introduction

Minimal effort backpropagation, or meProp, is a technique for neural network learning that reduces the computational cost of training without sacrificing accuracy. In fact, accuracy is found to improve in most cases, likely as a result of reduced overfitting. This is done by sparsifying each gradient update to only listen to the most influential nodes in a hidden layer. In this project, we explore several experiments on the implementation of meProp.

## The meProp Algorithm

At each layer, neural networks typically store:

- $W \in \mathbb{R}^{m*n}$, our weight matrix
- $x \in \mathbb{R}^{m}$, our input layer
- $y \in \mathbb{R}^{n}$, our output layer

where $y = Wx$. In backpropagation, one would normally calculate the gradient as $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial y} * x^T$.
However, under meProp, we fix an integer $k < n$ and look at the $k$ entries in $\frac{\partial L}{\partial y}$ of greatest magnitude, and only compute $\frac{\partial L}{\partial W_{i*}}$ for the corresponding values of $i$.

That is, we assign $\frac{\partial L}{\partial W} \leftarrow top_k\left(\frac{\partial L}{\partial y}\right) x^T$
where $top_k(v)$ returns a vector where all but the $k$ entries in $v$ of greatest magnitude are set to 0. It is easy to see why this would lead to a linear reduction in the runtime for backpropagation.

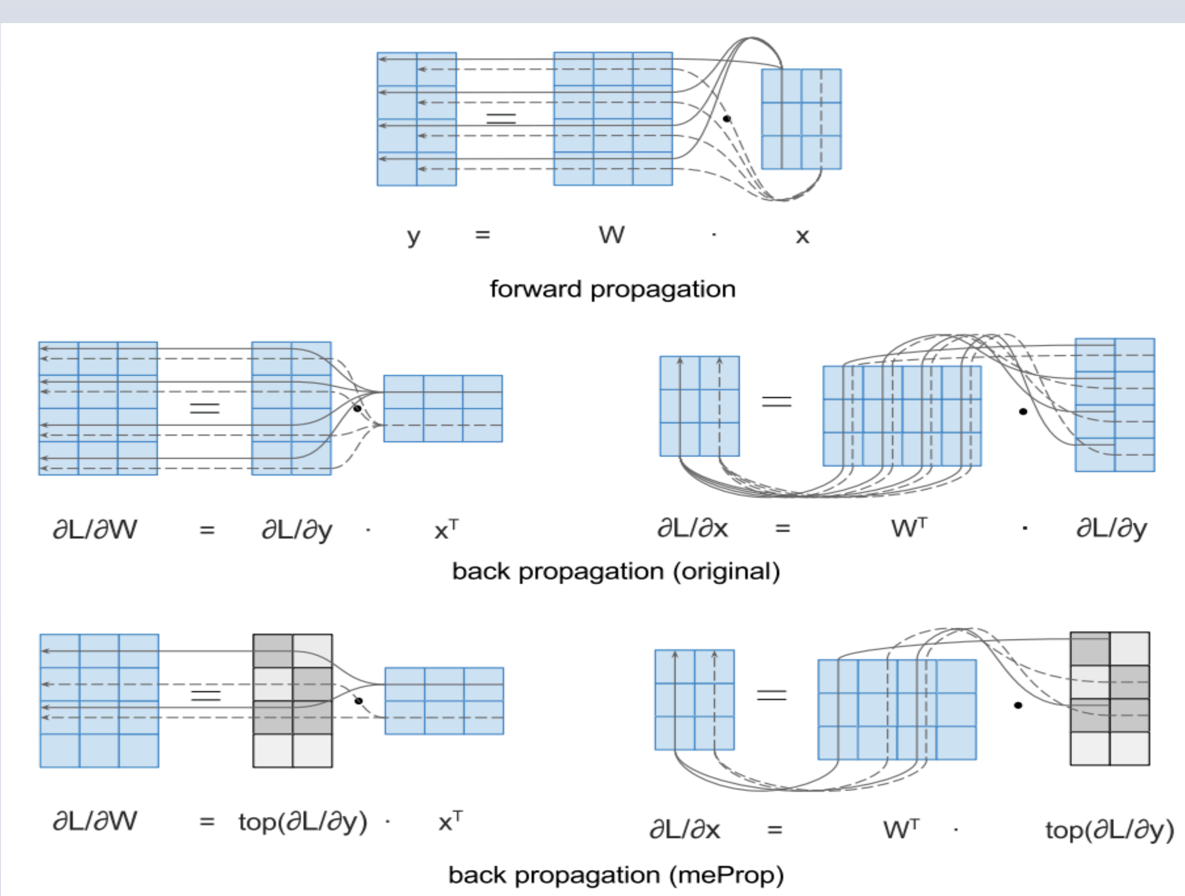


Fig. 1: Reduced matrix calculation for (mini-batch) meProp[1]

## Observations and Approaches

Wang *et al.* (2017), who invented meProp, already addressed concerns regarding simultaneous use with dropout, 'randomized' meProp, and deep neural networks. However, we felt as though there were a few more areas requiring further exploration:

1. Intuitively, the neural network through several epochs learns the significance of weights, so we look at the possibility of decreasing $k$ as it trains.

2. For different training examples, $\frac{\partial L}{\partial y}$ will have a different distribution of magnitudes, for which we shouldn't necessarily assign the same values of $k$. So, we instead fix a threshold $0 < \tau < 1$ and assign for each training example

$$k^{(i)} = \min\{k : \left\|top_k\left(\frac{\partial L}{\partial y^{(i)}}\right)\right\|_1 > \tau \left\|\frac{\partial L}{\partial y^{(i)}}\right\|_1\}.$$

3. If the neural net is multilayered, meProp currently passes the same $k$ value at each layer. It deserves some attention as to whether we should vary $k$ on each layer depending on the layer size. For example, we could define $k$ to be a fixed fraction of the number of nodes in each layer.

## Annealing $k$ with Learning Rate

A common technique used in neural network training is to reduce the learning rate after a sharp increase in dev loss. Since meProp is also useful for overfitting, we tried different variations of annealing $k$ and the learning rate.

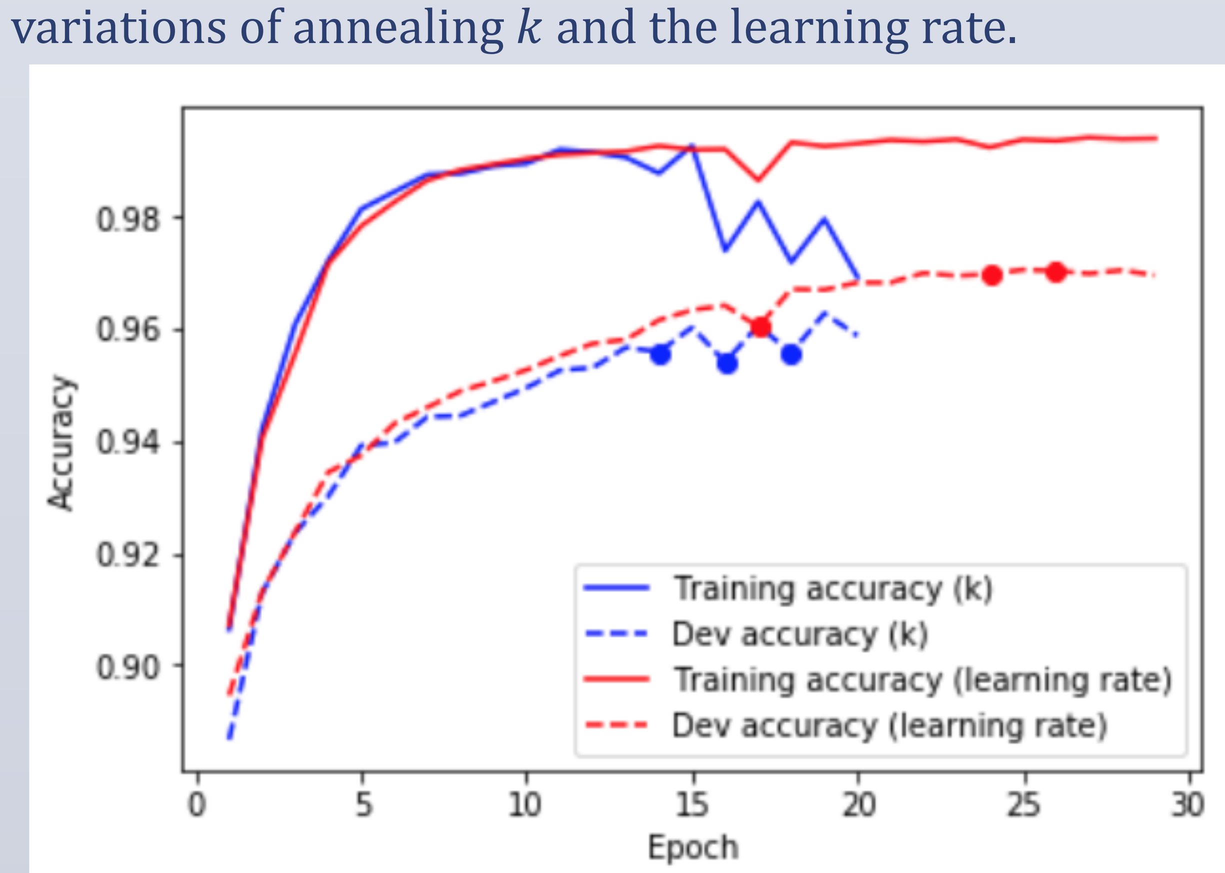


Fig. 2: Variations of annealing $k$ and the learning rate

## Norm-Limited $k$ Selection

Our hope with norm-limited $\boldsymbol{k}$ selection was to pass components of the gradients that were numerically significant, rather than using a fixed $\boldsymbol{k}$ throughout a layer.

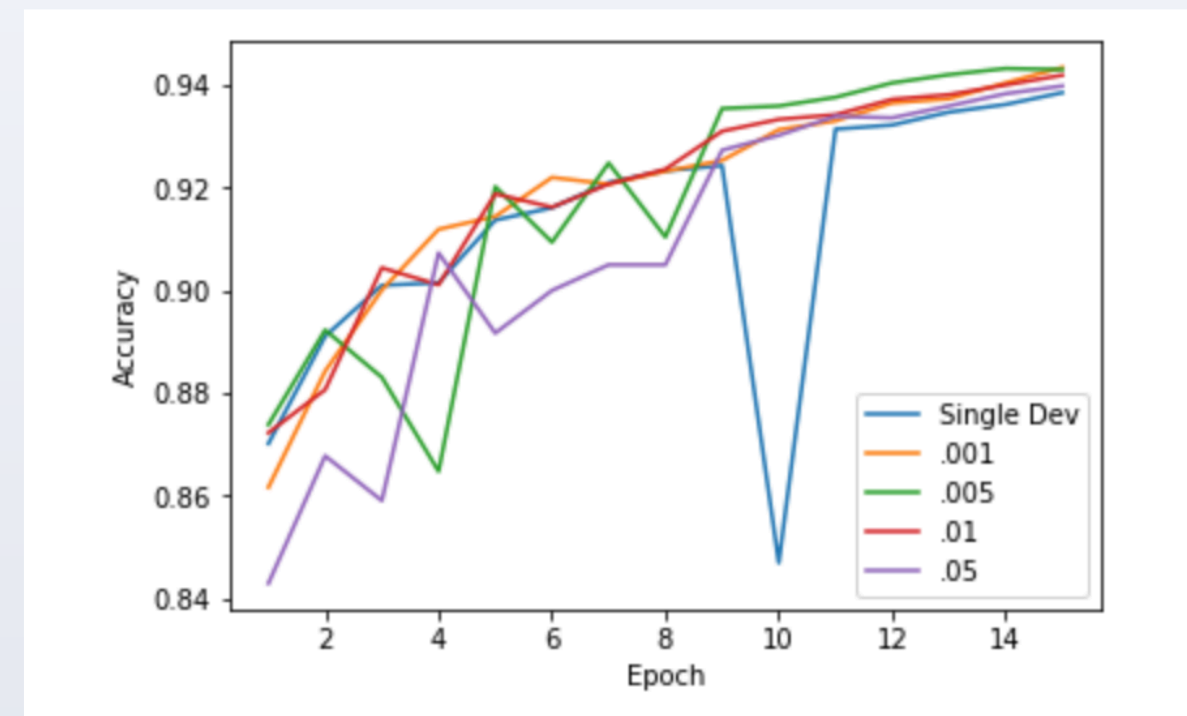


Fig 3: Test accuracies for different norm limits

## Changing $k$ for Hidden Layer Dimensions

Our first approach setting $k$ for both layers to be a certain fixed fraction of our hidden layer size. Our intuition for this approach was to maximize the effect of meProp at each layer by preventing meProp from removing too few or too many of the gradients at any particular layer.

| Ratio | Test Accuracy |
|---|---|
| .0225 | .9715 |
| .0305 | .9763 |
| .0340 | .9675 |

Fig. 4: Some of our local optimal $k$ ratios

Our second idea was to modify values of $k$ depending on how deep into our neural network we were.

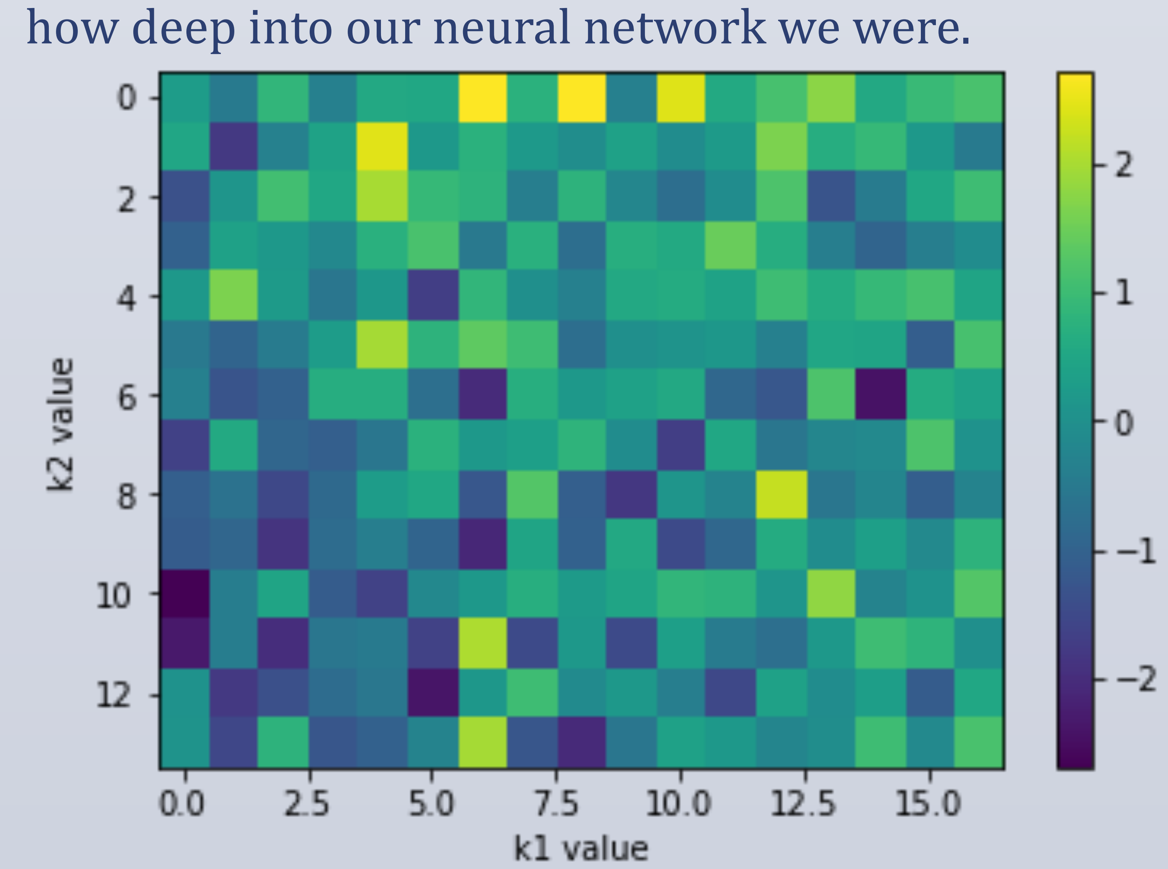


Fig. 5: A visualization of test accuracy based on values of k at different layers

## Results/ Analysis

Overall, from our experiments annealing $k$ and the learning rate, it seems that annealing $k$ upon spikes in the dev loss is not as effective as annealing the learning rate, as the dev accuracy performs worse and the epochs of annealment are clustered closer together.

Norm-Limited $k$ selection seemed to perform the poorest out of our modifications. Regardless of the norm limit, all models converged within error to the test accuracy of the neural network without meProp implemented. It's possible that we didn't test sufficiently high values of $\tau$.

While there was no direct relation between the $k$ ratio and test accuracy, different $k$ values yielded significantly different test accuracies. Treating this ratio as a hyper-parameter may prove to be a useful modification to meProp. In addition, it seemed that earlier layers of neural networks seemed to benefit more from meProp.

## Future Research

One potential future experiment to perform would be to more carefully evaluate the runtime of the norm-limited algorithm, for it may increase the computational cost, though we would anticipate the difference to be slight. Furthermore, as all of these experiment were performed on a feedforward neural network, it is worthwhile to consider generalizing meProp and its modifications to sparsify backpropagation for other models, such as an RNN. A last area of further study would be to explore the relationship between norm regularization and meProp.

## Works Cited

1. Wang et al. "meProp: Sparsified Back Propagation for Accelerated Deep Learning with Reduced Overfitting" arXiv:1706.06197 (2017)
2. Wei et al. "Minimal Effort Back Propagation for Convolutional Neural Networks" arXiv:1709.05804 (2017)
3. LeCun, Yann and Cortes, Corinna. "MNIST handwritten digit database." (2010):