**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:

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

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

That is, we assign \( \frac{\partial L}{\partial y} \leftarrow \text{top}_k \left( \frac{\partial L}{\partial y} \right) \ast x^T \) where \( \text{top}_k \) returns a vector where all but the \( k \) entries in \( y \) 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.

**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 significant 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

\[
(k^{(i)}) = \min \{ k : \frac{\partial L}{\partial y}^{(i)} \geq \tau \frac{\partial L}{\partial y} \}
\]

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.

**Norm-Limited \( k \) Selection**

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

**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.

**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 experiments 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**