Target Propagation- Empirical Investigation

Abhijeet Ghawade [EE15B088]
Anand P S [EE15B088]

January 2019

1 Introduction

Modern deep learning relies on back-propagation as a workhorse for training and credit assignment. As we consider deeper networks– e.g., consider the recent best ImageNet competition entrants (19 or 22 layers)– the composition of many non-linear operations becomes more strongly non-linear derivatives obtained by back-propagation are becoming either very small (most of the time) or very large (in a few places). Though back-propagation has been crucial in obtaining state of the art performance in various problems/ train various architectures, its lack of biological analogies has long been cited as one of the reasons it cannot be a feasible representation of credit assignment in the brain.

These reasons include

- the back-propagation computation is purely linear, whereas biological neurons interleave linear and non-linear operations
- Feedback neurons carrying gradient information would need precise knowledge of the feedback path
- Existence of an information delivery that carries signed error gradients, through the layers, without influencing the existing activation as it is doing so.
- real neurons communicate by (possibly stochastic) binary values(spikes)

2 Spiking Time Dependent Plasticity

Spiking Time Dependent Plasticity is the control of/change in weighting given to synapse, based on the order of spiking or relative spiking times. If the post-synaptic neurons fire significantly earlier than the pre-synaptic neurons, then the synapse is down weighted. This is interpreted as a layer-local training. This gives rise to layer-wise training of neural networks.

3 Target Propagation

Target propagation was proposed by [1]. It uses inverse functions to propagate targets to lower layers. This inverse function is forced to satisfy

$$g(\hat{h}_l) - g(h_l) = \hat{h}_{l-1} - \hat{h}_l$$

where $h_l$is the activation of layer l and g is the inverse that obtains targets for layer $l-1$ from activations of la

3.1 Difference Target propagation

This variant of target propagation sets the last layer targets using a gradient based correction to the current activity. This gradient is still a local gradient, since the interaction is still restricted to consecutive layers. This is still feasible under STDP.

Propagate activity forward
for $l = 1$ *to* L **do**
  Compute activity forward:$h_l \leftarrow f_l(h_{l-1}; \theta)$;
end
Compute target for first layer: $\hat{h}_L \leftarrow h_L - \frac{d}{dh_L} L(y, h_L)$
generate targets for lower layers:
for $l = L - 1$ *to* 1 **do**
  $\hat{h}_l = h_l - g(h_{l+1}; \lambda_{l+1}) + g(\hat{h}_{l+1}; \lambda_{l+1}))$
end
train inverse function parameters:
for $l = L$ *to* $l = 2$ **do**
  generate corrupted activity: $\tilde{h}_{l-1} = h_{l-1} + \epsilon$
  where $\sim N(0, \sigma^2)$
  Update parameters λ_l using SGD on loss L_{inv}
  $L_{inv}(\lambda_l) = h_{l-1} - g(\tilde{h}_l; \lambda_l)^2$
end
Train feed-forward parameters:
for $l = 1$ *to* L **do**
  Update parameters θ_l using SGD on Loss L_{inv}:
  $L_{inv}(\theta_l) = f(h_l; \theta_l) - \hat{h}_{l+1}{}^2$
end

Algorithm 1: Difference Target Propagation

3.2 Simplified Difference Target Propagation

Here the gradient based adaptation of the last layer is achieved by merely loading the labels onto the last layer, to become the targets of the penultimate layer.

Propagate activity forward
for $l = 1$ *to* L **do**
  Compute activity forward:$h_l \leftarrow f_l(h_{l-1}; \theta)$;
end
Compute target for first layer: $\hat{h}_L \leftarrow argmin_{h_L} L(y, h_L)$
generate targets for lower layers:
for $l = L - 1$ *to* 1 **do**
  $\hat{h}_l = h_l - g(h_{l+1}; \lambda_{l+1}) + g(\hat{h}_{l+1}; \lambda_{l+1}))$
end
train inverse function parameters:
for $l = L$ *to* $l = 2$ **do**
  generate corrupted activity: $\tilde{h}_{l-1} = h_{l-1} + \epsilon$
  where $\sim N(0, \sigma^2)$
  Update parameters λ_l using SGD on loss L_{inv}
  $L_{inv}(\lambda_l) = h_{l-1} - g(\tilde{h}_l; \lambda_l)^2$
end
Train feed-forward parameters:
for $l = 1$ *to* L **do**
  Update parameters θ_l using SGD on Loss L_{inv}:
  $L_{inv}(\theta_l) = f(h_l; \theta_l) - \hat{h}_{l+1}{}^2$
end

Algorithm 2: Simplified Difference Target Propagation

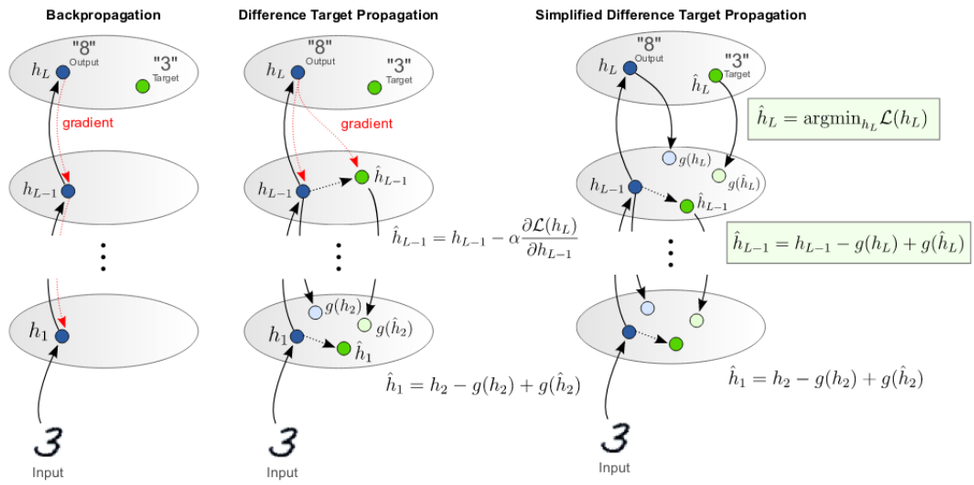


Figure 1: In BP and DTP, the final layer target is used to compute a loss, and the gradients from this loss are shuttled backwards (through all layers, in BP, or just one layer, in DTP) in error propagation steps that do not influence actual neural activity. SDTP never transports gradients using error propagation steps, unlike DTP and BP.

Image taken from [2]

4 Optimizers

We have used 3 optimizers majorly for the purpose of this project

- Stochastic Gradient Descent
- RMS Prop Optimizer
- Adam Optimizer

4.1 Batch Gradient Descent

Batch Gradient descent is a variation of the batch Gradient Descent algorithm(Vanilla Gradient Descent). It computes the gradient of the cost function w.r.t. to the parameters θ for the entire training dataset.
θ The update rule can be written as,
$\theta = \theta - \eta * \nabla_\theta J(\theta)$

for the entire dataset, there is only 1 update. Thus, it can be very slow at times. batch gradient descent divides the train data into a certain number of batches, and calculates the gradient for only one batch at a time.
Stochastic gradient descent is the extreme version of batch gradient descent which used one sample at a time to calculate gradient.
The update equation for a sample x_i, y_i looks like,
$\theta = \theta - \eta * \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$

4.2 RMSProp

let $g_{t,i} = \nabla_\theta J(\theta_{t,i})$
The RMSProp update can be written as,

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$
$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

4.3 Adam

Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t similar to momentum.
We compute the decaying averages of past and past squared gradients m_t,v_t respectively as follows,
$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

m_t, v_t are estimates of the mean and the variance (1st and 2nd moment) of the gradient. The biases in these are counteracted by diving by constants, hence
$\tilde{m^t} = \frac{m_t}{1-\beta_1^t}$
$\tilde{v^t} = \frac{v_t}{1-\beta_2^t}$

The update equations are finally written as,
$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\tilde{v_t}+\epsilon}} \tilde{m_t}$

5 Graphs

5.1 DTP vs back-propagation

Here we compare the performance of the two algorithms we are discussing in this report, DTP and back-prop. As we can see that both of have very similar performances, in terms of accuracy.
We have used 'tanh' activation with 1000 nodes in a single layer for both the networks.
Adam Optimizer is used to update the weights in both the cases. MNIST dataset is used for the performance evaluation.

Both the algorithms reach a peak performance of around 0.966 accuracy.

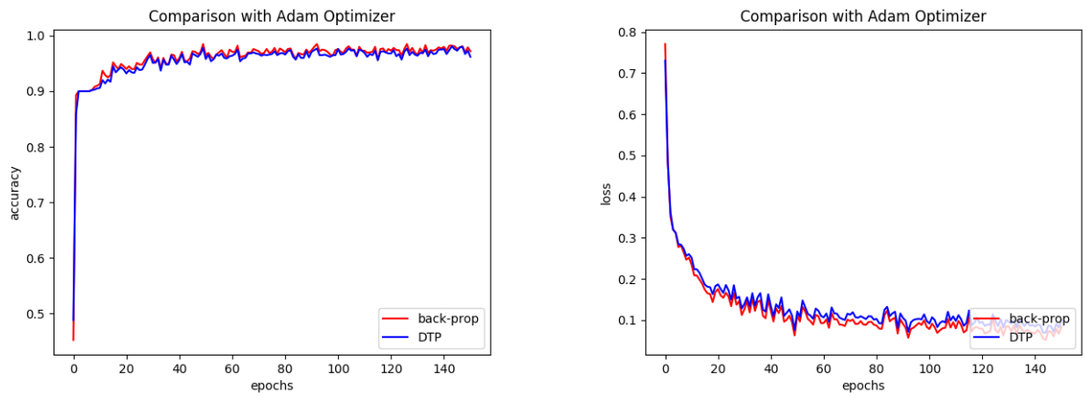


Figure 1:

5.2 Optimization functions (RMSProp/Adam)

We perform comparative numerical experiments on the MNIST dataset, under different architecures and optimization functions i.e. RMSProp[Figure 3], Adam [Figure 2].

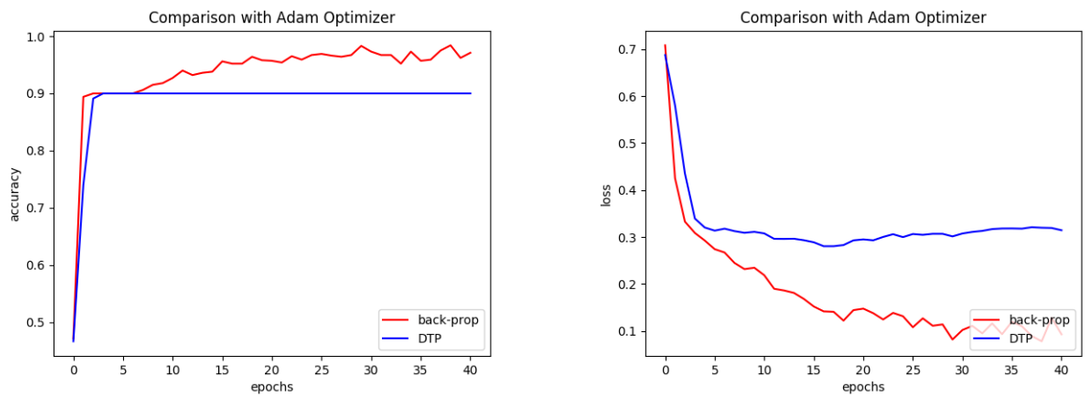


Figure 2: Adam Optimizer: accuracy and loss

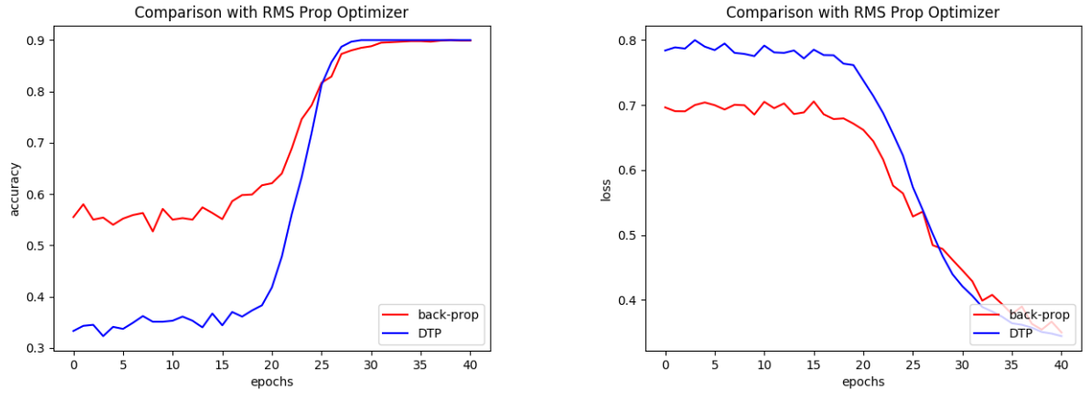


Figure 3: RMSProp Optimizer: accuracy and loss

- From the above comparison, we see that RMSProp[Figure 3] needs a 'burn-in' period, due to the lack of momentum adaptation (change in the momentum coefficient).

- The burn in behaviour is seen only if the forward pass is trained using the RMSProp optimizer.

- Target Prop is still more susceptible to Local minima under Adam, whereas Back Prop does seem to quickly overcome such a minima/saddle point or avoid it altogether. This suggests that the updates are 'more optimal' in backprop.

5.3 Noise Variations

We do not want to estimate an inverse mapping only for the concrete values we see in training but for a region around the these values to facilitate the computation of $g_i(h^i)$ for h^i which have never been seen before. For this reason, the loss is modified by noise injection.

$$L_i^{inv} = ||g_i(f_i(h_{i-1} + \epsilon)) - (h_{i-1} + \epsilon)||_2^2$$
$$\epsilon \sim N(o, \sigma)$$

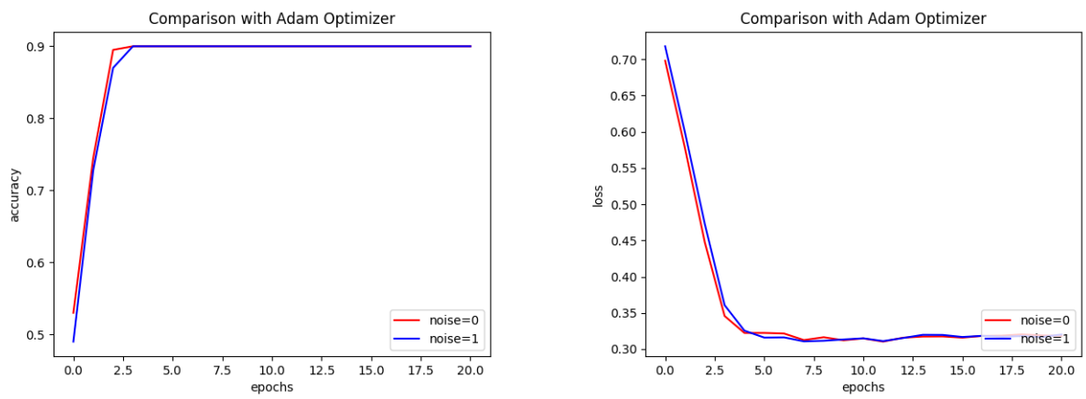


Figure 4:

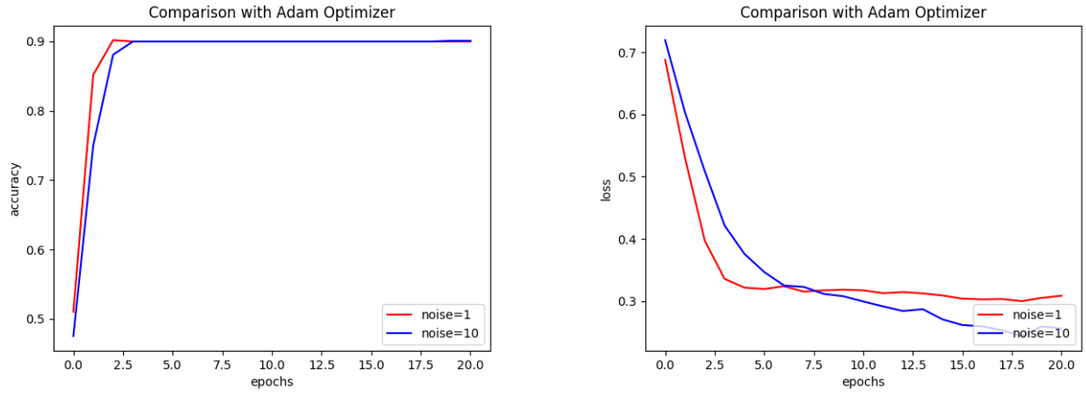


Figure 5:

But the experiments suggest that there is no significant change obtained in the performance due to the addition of noise. The performance is marginally better with zero noise in this particular case in the initial stages, but it converges at the same performance.
This could be due to the small dimensionality of the data. But with increasing data size, and as generalization becomes of importance, noise will play a more important role.

5.4 Alpha variations

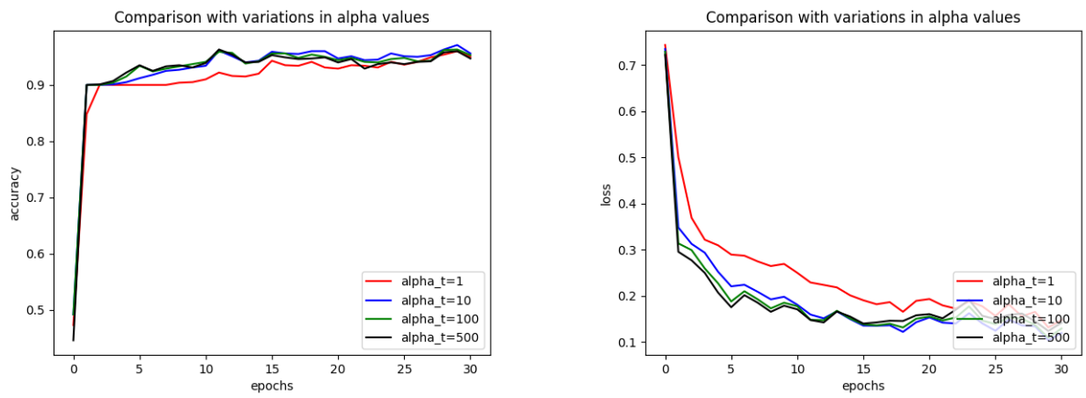
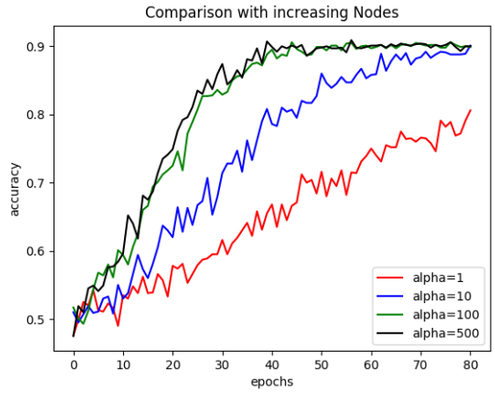


Figure 6:

α functions as a close analogue of the learning rate(for BP) in DTP. SDTP does not have any such parameters(or SDTP can be seen as DTP with the parameter set to zero, while y is set as target). In the graph shown, we obtain accuracy and loss curves for a single hidden layer network, with 1000 parameters. The noise variance is kept at 1.0, and the initial learning rate is set as 3e-4.

On varying α we see that increasing α leads to faster convergence, without any instability. The effect of increasing α is more appreciable when the target dimensions are fewer. This was observed[Figure: 7 and 8] when performing the the same experiment with different network widths(for the same number of hidden layers=1, training on the CIFAR10 dataset) of 200 and 1000. We see in the lower figures(1000 units) small values of α can produce similar speed of convergence, though in the smaller network, α needs to be larger to produce the same rate of convergence.



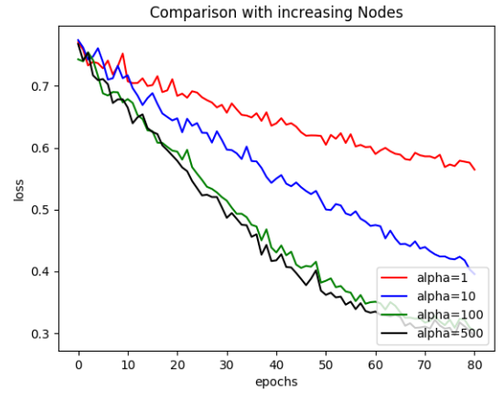
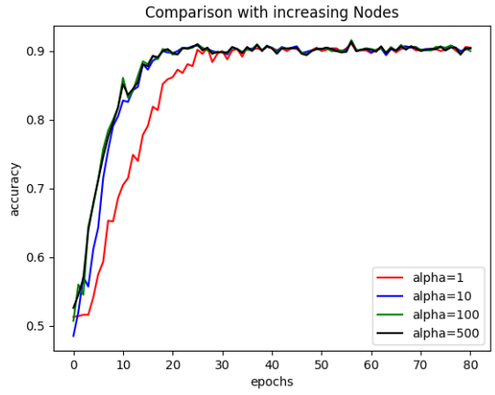


Figure 7:



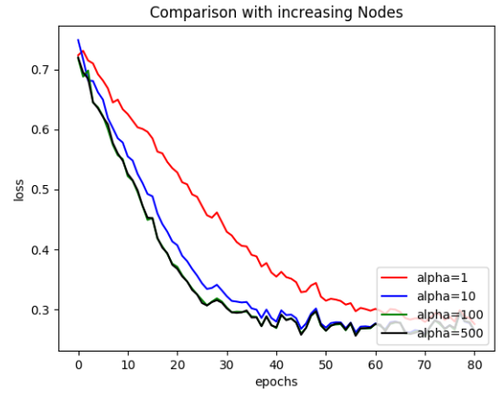
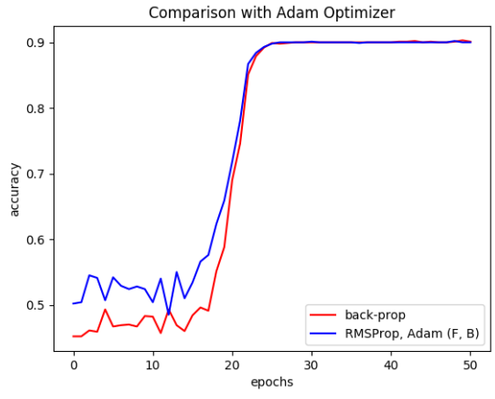


Figure 8:

5.5 Different update rules for forward pass and backward learning



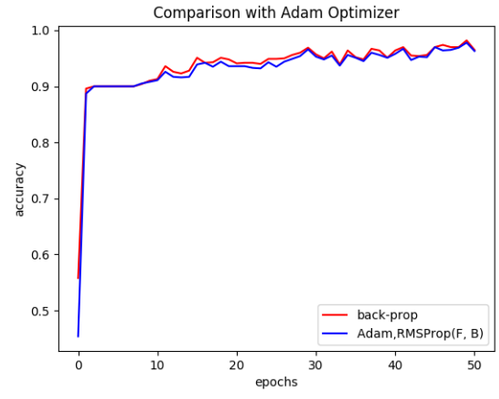
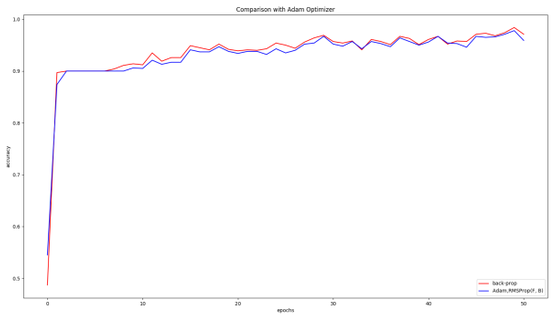


Figure 9:



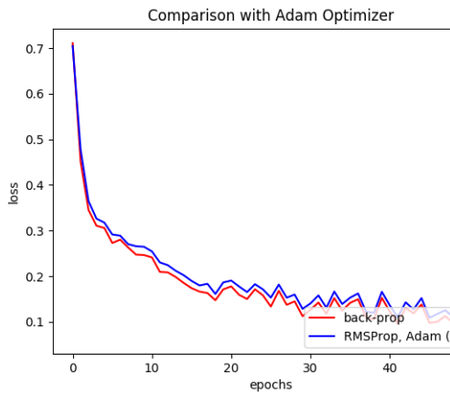


Figure 10:

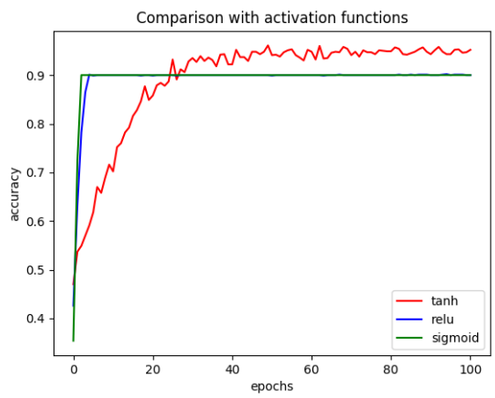
Here we changed the optimizers used to train the forward pass and inverse function parameters(as opposed to the same optimizer for both). We set α as 1.0, learning rate as 3e-4, and the MNIST dataset is used for training.
We find that the loss curve is primarily determined by the optimizer used in the forward parameter training[Fig:9(RMSProp forward Adam back) and 10(Adam Forward and RMSProp Backward)]. This is expected as networks can converge reliably even with very small batch sizes(i.e with highly corrupted oracle gradients).

5.6 Different activation function variations

We have experimented with activation functions to see their corresponding effects on learning performance in DTP. The graphs below represent the comparative performances for them.

5.6.1 MNIST data

We first compare the effect of different activation functions in the forward propagation step. We examine the training of a network with 1 hidden layer of width 1000. we also fix other parameters such as α_t, noise variance and the optimizer used(Adam).



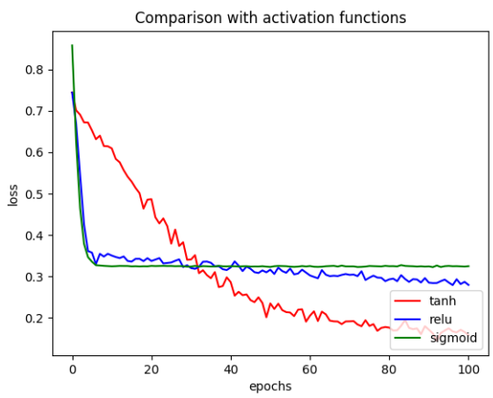
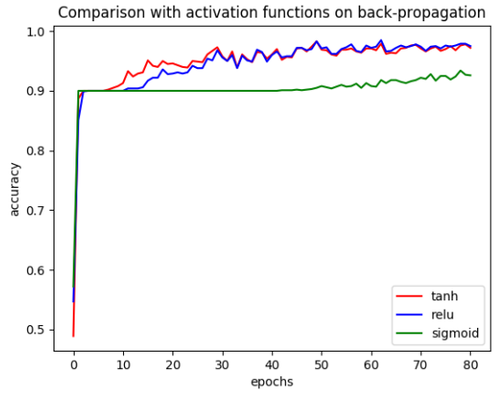


Figure 11:

The better performance of 'tanh' [figure 11] is hypothesized to be due to the variation in sign that *tanh* can achieve. We conduct an experiment to test this using a comparison of relu and leaky-relu(α =0.1)[figure-13]. The results show that a leaky-relu converges faster. However, the local saddle point which gives accuracy 0.9, continues to exist in leaky-relu also.



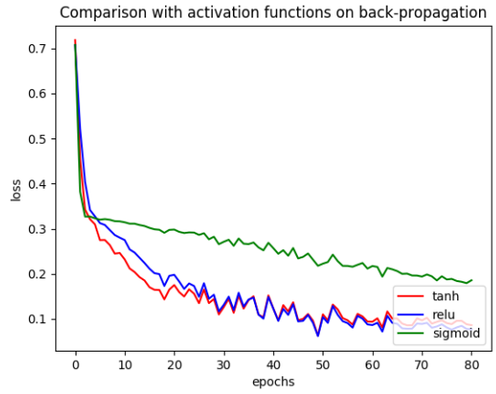
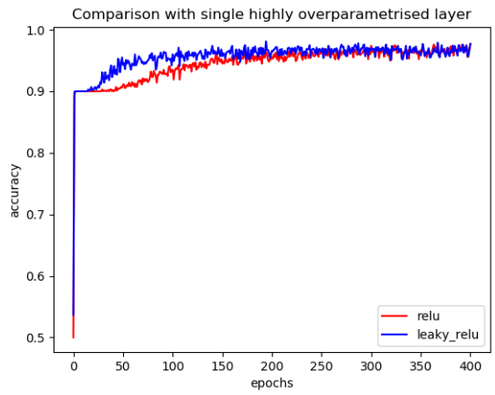


Figure 12:

In case of MNIST data and **BackPropagation** [figure 12], we see that the best performance is achieved by ReLU, similar to tanh activation. Sigmoid performs worse as compared to tanh, because tanh is zero centered.

ReLU performs better in backpropagation because it does not rely on inversion, which can potentially be influenced by the zero centering of the activation function.



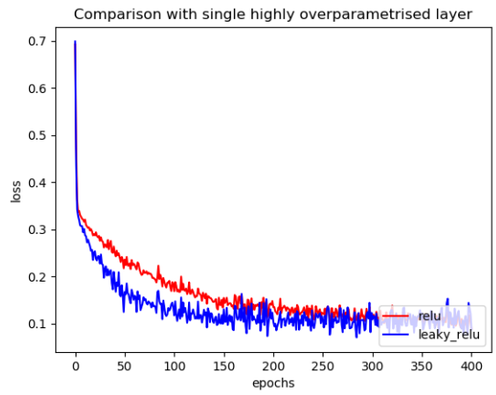
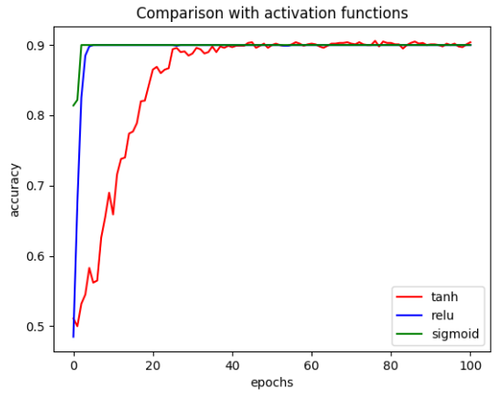


Figure 13:

5.6.2 CIFAR10 data



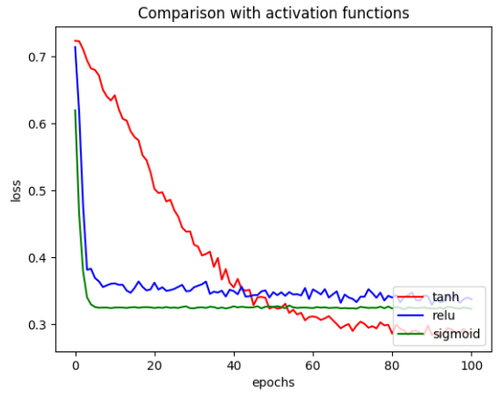


Figure 14:

The performance of different activation functions on the CIFAR10 data is roughly similar to the performance on MNIST data[Figure 14].

5.7 Pre-processing data

We have studied the effects of pre-processing the input data before feeding it to the DTP network. We are using Principal component analysis for dimensional reduction. The 784 input dimensions in MNIST are reduced to 100 using PCA, and then used regularly.

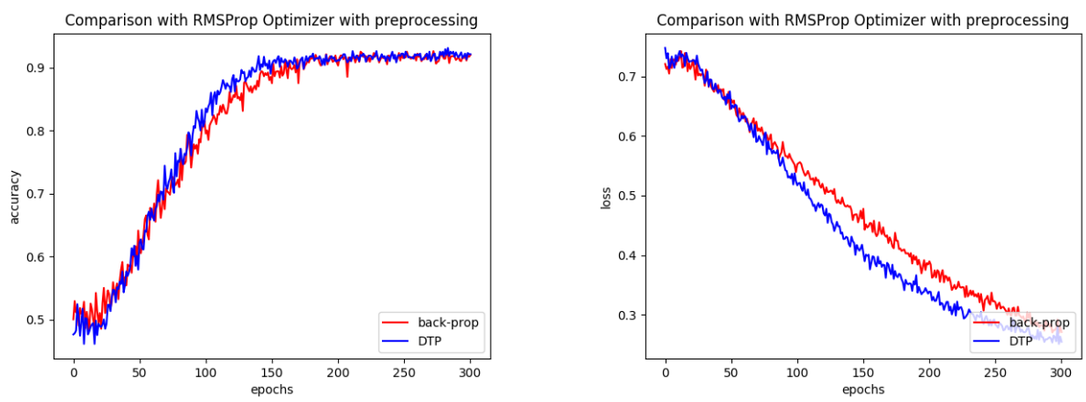


Figure 15:

In the comparison, the network size is kept constant for both the networks, and the learning algorithm is RMSProp in both the cases. It can be seen that DTP performs better than back-propagation when the data is pre-processed. One of the probable reasons for this finding could be that the inverse functions are calculated better with lower dimensional data. But after training for sufficiently large number of batches, both the algorithms converge to similarly.

5.8 Changing the network size

The number of hidden layers are being changed in this experiment, for both the DTP and Back-Propagation algorithm.
The number of layers and the nodes per layer are as follows:

- **1 hidden layer**: 200
- **2 hidden layers**: 200,200
- **4 hidden layers**: 200,200,200,200
- **8 hidden layers**: 200,200,200,200,200,200,200,200

This is the same configuration for both the Back-Propagation algorithm. Both are being trained using Adam Optimizer.
MNIST Dataset is being used for all the runs, with the same initial learning rate for all the different number of layers, for the same algorithm.

5.8.1 Backpropagation

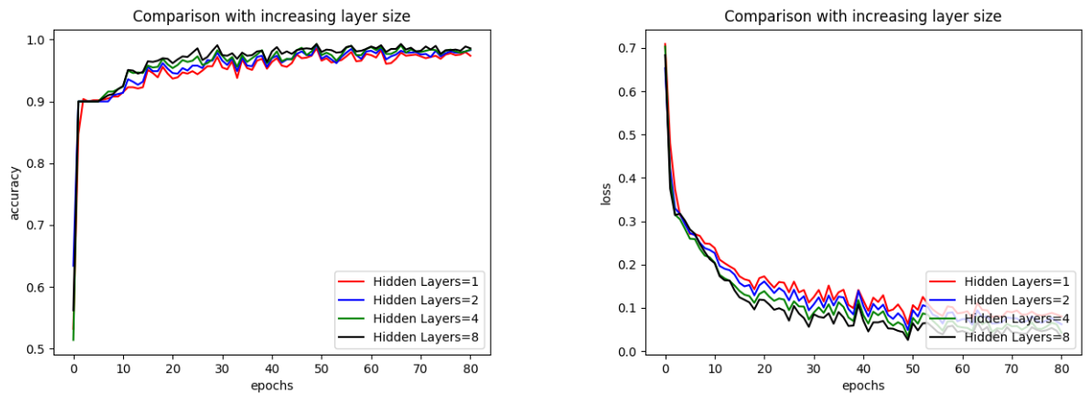


Figure 16:

We can see that the learning curve is almost the same for Back-Propagation, and the network with more number of hidden layer has slightly better accuracy as compared to the network with fewer number of hidden layers.

This is to be expected, as with more number of layers, the network can learn abstraction in the data better. But in case of the MNIST data, not a lot of difference is observed with addition of hidden layers, this could be because MNIST is a fairly small dataset and the abstraction are captured in a single hidden layer.

5.8.2 DTP

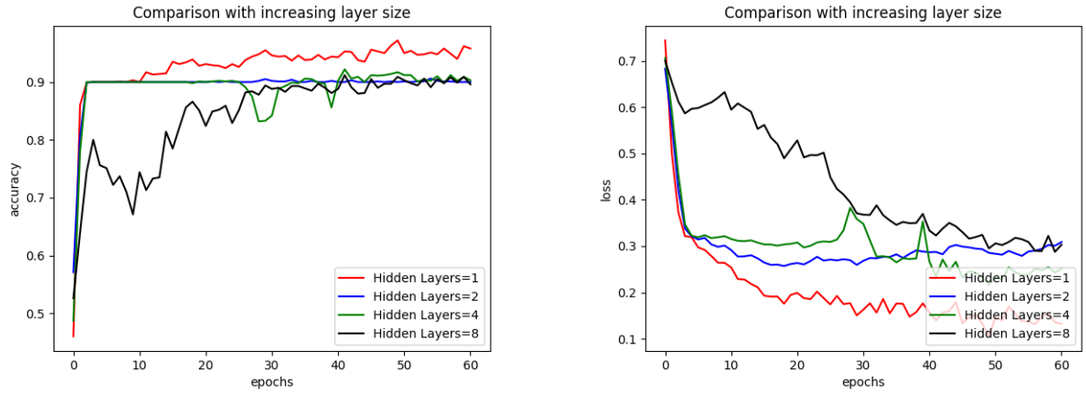


Figure 17:

There is a huge difference when it comes to changing the layers in DTP network with addition of more layers. With one hidden layer, the networks provides accuracy of up to 0.96, but after increasing the number of hidden layers, the performance deteriorates, which is counter intuitive. With addition of 8 hidden layer, the performance suffers the most, especially in the initial training period. This shows that DTP suffers with increasing number of layers.

This also explains why DTP and its variants do very poorly in large datasets like image-net, but performs fine with smaller datasets like MNIST.

5.9 Varying the number of nodes

Only one hidden layer is used in this experiment. The number of nodes are being changed in this experiment, for both the DTP and Back-Propagation algorithm. The number of nodes in the layer are as follows:

- **1** : 20
- **2** : 100
- **3** : 250
- **4** : 500

Data for training is MNIST, and same learning rate for all runs in the same algorithm.
This is the same configuration for both the Back-Propagation algorithm. Both are being trained using Adam Optimizer.

5.9.1 Back Propagation

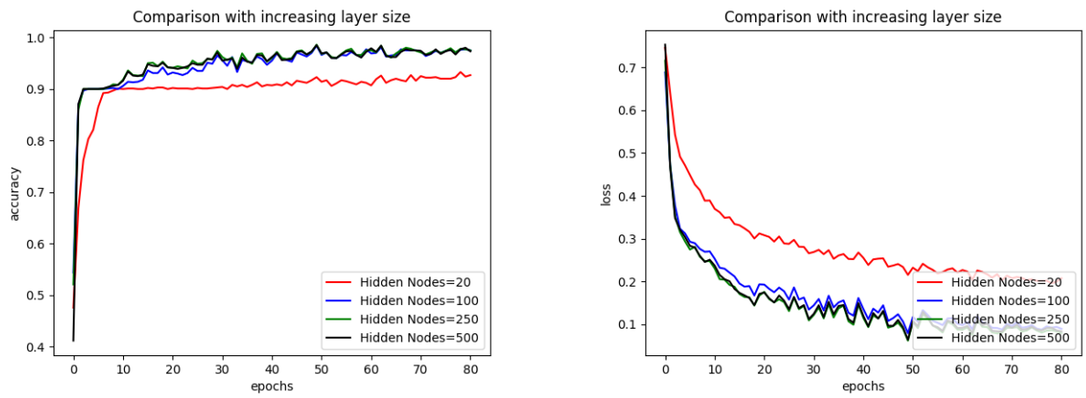


Figure 18:

As we can see, increasing the number of nodes does not change the accuracy or the loss much. It seems to be that around 50-75 number of layers is enough for optimal performance on MNIST dataset. There is a minor performance improvement even when the number of nodes is increased by 1 or 2 orders of magnitude.

The reasoning for this is similar to the one with increasing hidden layers, that the abstraction is learned with fairly low number of neurons.

5.9.2 Difference Target Propagation

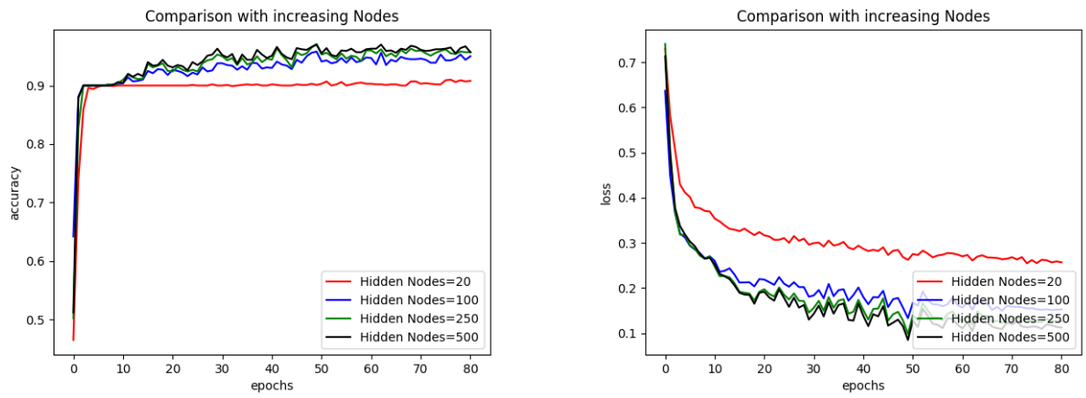


Figure 19:

The graph looks similar to the one obtained in the back-propagation case. This means that DTP does not suffer with addition of excessive nodes, as it does in case of addition of excessive layers. This means that the problem probably arises when we are calculating the inverse for layers away from the output layer, and hence the dip in the performance. The inverse calculation is high due to compounding of errors of already inaccurate inverses from previous layers.

6 Future Work

- DTP and its variants performs comparable to backpropagation, at times even better than it. But it is observed that it fails to perform good in larger networks and on large datasets. This is analogous to problem faced in back-propagation when batch-normalization was not used. There is perhaps a better way to calculate inverse which works better than this model. An analysis with larger dataset like ImageNet might provide the answers to that question.

- Recurrent Neural Networks (RNN) are used for analysis of sequential data. Due to its feedback connection, instead of Back-propagation, its variant Back-propagation through time is used. this has issues like vanishing and exploding gradients. Variants of RNNs like GRU and LSTM have been introduced. DTP is a natural choice of for gradient update, because gradient tensors do not multiply and leading to exploding values, thus training can be done more efficiently.

- Residual networks are important in training state-of-the-art models especially to enable training of deep(150 layer) networks. They do this by reducing the effective depth of the network. Extending DTP to a residual network is likely to solve the scaling issues it faces. It is also likely to make the algorithm ore biologically feasible, since neurons being distributed exactly layer-wise is not supported by experiments. These are very promising areas for future work.

- Looking at the functions of a biological neurons, and neuron models could help provide answers regarding the corrections required to this model in order to scale it up, and make the algorithm more general purpose.

References

[1] Difference Target Propagation
Dong-Hyun Lee, Saizheng Zhang, Asja Fischer and Yoshua Bengio

[2] Assessing the Scalability of Biologically-Motivated Deep Learning Algorithms and Architectures

Sergey Bartunov,Adam Santoro,Blake A. Richards,Luke Marris, Geoffrey E. Hinton,Timothy P. Lillicrap