Forward Thinking

Math People*
Department of Mathematics
Brigham Young University
Provo, UT 84602
seanwade@byu.edu

Abstract

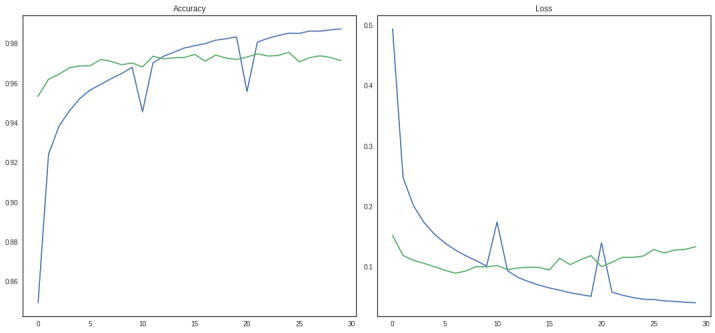
We propose a new method for the layer-wise dynamic training of neural networks without backpropegation. This method begins with a low level classifier and dynamically adds layers until learning does not improve. Removing back propegation also removes the need for predetermined computational graphs. This frees the network to grow the archetecture as it needs. The result is quicker training time as well as higher accuracy.

1 The Method

This is how I implemented the architectures. These methods draw inspiration from the layerwise training of Deep Belief Networks as well as the graph augmentation of the guy tanner said.

1.1 Forward Thinking

The base of forward thinking is feature engineering while training the network. In its most simple form this looks like using the loss function to have a low level learner try to classify by using one hidden layer. This layer can be interpreted as the learned features. When ready to increase learning ability the data is training data is transformed using these features. The next iteration of the algorithm takes the old features and the new features to attempt to learn the problem. A weakness to this method is there is limited transfer of learned knowledge when attempting to learn the next layer. This knowledge only lies in the form of the new features. This can be seen at training time with sharp drops in accuracy at new layers.

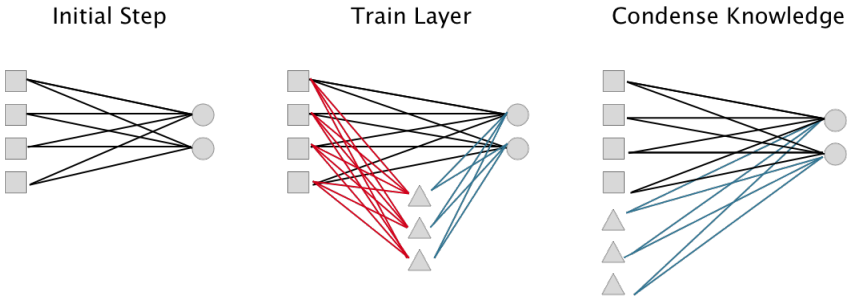


*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

1.2 Pass-Forward Thinking

To overcome the obstacles in the aforementioned section, we introduce information preserving pass-forward thinking. Similar to work by that one guy tanner cited, we intelligently add layers to the model, such that knowledge caries over and there is no drop in accuracy. In the case of standard DNN's this has an elegant representation.



The matrix represented by the black connections can be described as the knowledge matrix learned from previous layers. The red connections are the transformation matrix, the connections that generate the new features. Finally, the blue connections are the new knowledge. With our new features, this information connects directly to the output. In the final stage we flatten both the old and new knowledge to get a single representation.

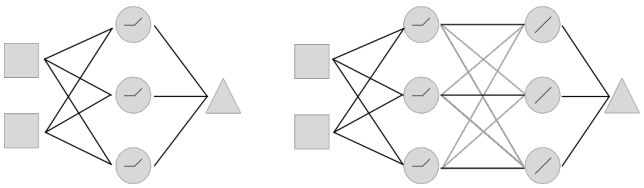
We note that with this method of training we have two options: freeze old knowledge or continue to train it. It seems to be that freezing outperforms. I speculate this is because it serves as a form of regularization and allows the network to not have to focus on tweaking everything at once.

1.3 Push-Forward Thinking

Push-Forward Thinking is method for growing neural networks deeper to improve learning. Our method allows doing so without a loss in accuracy (hopefully). This dynamic topology overcomes the weakness of standard neural networks by not having to predetermine the depth. It also eliminates the need for back propagation.

1.4 Thoughts

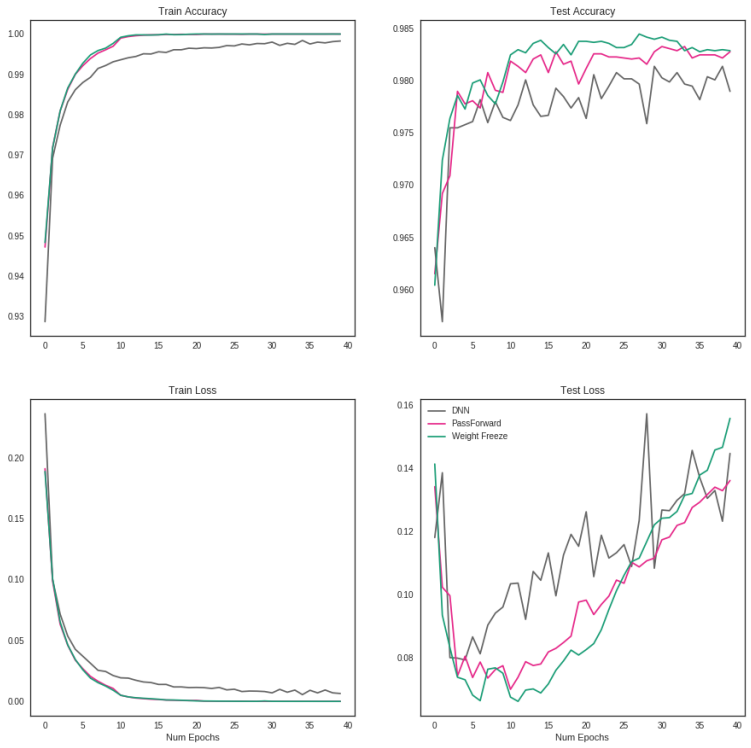
Each of these implementations offer different spins on the idea of forward thinking by pushing the data through while training. Pass-forward allows any number of hidden nodes to be added in the subseqwuent layers. This overcomes the weaknesses of cascase correlation learning, which only adds a single neuron? Whe we did this on MNIST it only was ably to get 96.34% accuracy since the added learners were too weak to gain any meaningful insite on their own. Push through has the limitation of having the same amount of hidden neurons in each layer.



2 Observations

2.1 Freezing Features

Once the network has plateued, the layers are then frozen and more layers are added to increase the compacity to learn. This method of training comes with several benifits as wll as tradeoffs. The first benifit is increased learning speeds. A typical neural network must learn all the weights at once. By starting with fewer weights, these can be learned faster as is seen in the following graphic.



Another benifit of freezing features is that it serves as a natural form of regularization. Once general concepts are learned, layers are frozen to prevent overfitting by hyper tuning the parameters. Now thats a win.

2.2 Greedy Algorithm

A weakness of elimenating backprop is that forward thinking is a gready algorithm. If the data is transformed with each layer then data is lost and it learnes the best it can even though that may not be optimal. We saw this when doing the pure push data through approach.

We were able to combat this by using an approach similar to res-nets. That is that we would still input the primary features into subsequent layers. This improved accuracy and allowed the network to recover from greedy learning later down the line in training.

2.3 Other Random

- Focus on building whole network from scratch, dynamic
- No backprop
- Forward Thinking learns quicker than backprop, the accuracy only catches us when they both run a long time.

3 Results

3.1 Regularization

In working with the MNIST dataset we implemented many different types of regularization. These included l1, l2, as well as dropout. Both the regular DNN and Forward THining implementations would get comparable results to before. With this being such a simple problem there were no noteworthy increases in accuracy in any of the methods. But it is important to note that these results were the same with a plain DNN structure. It is very likely that the benifits of regularization will extend in the same places they do with DNNs. If we can find a good dataset to show this with that would be nice.

Table 1: MNIST without augmentation

Name	Accuracy	Training Time (seconds)
DNN	97.9%	265.93
Forward Thinking	97.9%	174.81
Pass-Forward Thinking	98.28%	239.07
Pass-Forward Thinking (weight freeze)	98.29%	231.42

Table 2: MNIST with augmentation

Name	Accuracy	Training Time (seconds)
DNN	98.78%	slowest
Forward Thinking	98.80%	2 fastest
Pass-Forward Thinking	98.87%	3 fastest
Pass-Forward Thinking (weight freeze)	98.85%	1 fastest

Make the time and accuracy relative

- The weight freezing is significantly faster than not for the augmented data.
- the abovce results are a 3 layer deep network, 150, 100, 50. Adding a 4th layer had no improvment in accuracy
- My code is on the github if there are other things anybody wants to try

4 Future Work

- How to determine width of adding layer in pass-forward
- look at learned features at every layer
- Use similar method for other machine learning models by stacking them and passing through the data
- CNN's
- Optimization methods build specifically with this in mind
- When to best know when to add a layer, is it just accuracy dosnt change for 5 epcohs or is there something better?

References