CSE 569: Mentee Networks

Dhiraj Gurkhe(dgurkhe@asu.edu,1209305002), Himanshu Tyagi(htyagi3@asu.edu,1209283279)

Abstract—Machine learning has been a pioneer in Computer Science and related fields in the recent past. One way of improving performance of any machine learning algorithm is to train the a plothora of models using the algorithm and then find the average of those predictions. But, even with the amount of computation we have today, training and predictings outputs of the different models is a cumbersome task with huge complexity and requires a large amount of computation power. As mentioned by [1] , they were able to achieve good results on MNIST dataset by distilling knowledge of an already trained ensemble model into a single model. This project is based on the experimentation and findings defined in [2] , who dived deeper in using bigger networks to train mid-sized networks and achieved better generalization accuracies as compared to common regularization techniques such as l_2 , l_1 and dropouts

Index Terms—Neural Network, Distillation, MNIST.

1 Introduction

In the present large scale machine learning, difference in requirements of the training stage and deployment stage is not considered. The same model is used in both the scenarios. As stated by [1], tasks like speech and object recognition, training must extract structure from very large, highly redundant datasets but it does not need to operate in real time and it can use a huge amount of computation. Deployment to a large number of users, however, has much more stringent requirements on latency and computational resources. The idea of training an cumbersome model to be an ensemble of multiple trained models, or train a single very large model trained with a very strong regularier such as dropout [1].

It is general accepted notion that the true objective of the user must be reflected by the objective function used for training as closely as possible.Still, models that are usually trained optimize performance on the training data, whereas the real objective is to generalize good on new data. If the cumbersome model generalizes well because, for example, it is the average of a large ensemble of different models, a small model trained to generalize in the same way will typically do much better on test data than a small model that is trained in the normal way on the same training set as was used to train the ensemble.

It has become common practice in the computer vision community to simply fine-tune one of networks such as VGG-19, overfeat, R-CNN for any task [2] We also show that mentee networks are better transferable than the independently learnt baselines and are also a good initializer. We also show that mentee networks can learn good representations from very little data and sometimes even without supervision from a dataset in an unsupervised fashion. We have also tried out variations in the MNIST dataset [3] to check how the model works on datasets the mentor was not trained on.

2 Methodology

We've followed the methodology followed by [2] Generalized mentored learning. If a large mentor network with n layers which is already trained a large dataset D , M_n, the k^{th} neuron activations in the i^{th} layer in the network as $M_n(i,k)$. Consider a smaller mentee network with m layers as S_m. Suppose we want to train S_m on a new dataset d_i, which is a subset of D. They defines a probe as following

$$\varphi(l,j) = \sqrt{\frac{1}{a}\sum_{i=0}^{a}(M_n(l,i) - S_n(j,i))^2}, \tag{1}$$

$\forall$ l $\leq$ n and $\forall$ j $\leq$ m

where a is the minimum number of neurons added to the probe between the mentor and mentee network. The overall network cost is then given by the following equation:

$$e = \alpha_t Ł_s(d_1^b) + \beta_t \sum_{\forall(l,j)\in B} \varphi(l,j) + \gamma_t \varphi(n,m) \tag{2}$$

here, d_1^b is the b^{th} mini-batch of data from the dataset d_1. where $Ł(.)_s$ is the network loss of that mini-batch, $\alpha_t and \beta_t$ weighs the two losses together for balanced training and γ_t is the weight of the probe between the two temperature softmax layers. We in our implementation have used a 1-hidden layer, therefore there is no $\sum$ term in the second probe term $\varphi(l,j)$, this term basically calculated the difference in RMSE of the mentor and mentee hidden layers.

Since M is pre-trained model, the second and third terms in the equation 6 are penalties for the layers in S that donot resemble the activations of the probed layer from M respectively.

The update rule with mentored probes is given as follows:

$$w^{t+1} = w^t - \eta[\alpha_t \frac{\partial}{\partial\omega}Ł_s + \beta_t \sum_{\forall(l,j)\in B}\frac{\partial}{\partial\omega}\varphi(l,j) + \gamma_t \frac{\partial}{\partial\omega}\varphi(n,m)] \tag{3}$$

The last two terms in the above equation act as guided noise parameters, which decrease with the number of iterations.

Softmax classifier function provided us a generalization of multiple classes. Softmax classifiers give us somewhat more intuitive output and also probabilistic interpretation. A general softmax function is defined as follows:

$$f_j(z) = \frac{\exp(z_j)}{\sum_k \exp(z_k)} \tag{4}$$

TABLE 1
Datasets Used

Dataset	Train+Validate+Test
MNIST + background images	10000 + 2000 + 50000
MNIST + background images	10000 + 2000 + 50000
MNIST + random background	10000 + 2000 + 50000
Rotated MNIST digits	10000 + 2000 + 50000

Softmax function converts a vector of arbitrary real-valued scores and squashed them to a vector of values between (0,1) which all add up to 1. This gives us a better view of the corelation between the classes and also finding out the true class becomes easier as only the true class will have the highest value. We have used the temperature softmax as described by [1] defnined below:

$$q_i = \frac{\exp(\frac{z_i}{T})}{\sum_i \exp(\frac{z_i}{T})} \tag{5}$$

3 Implementation Details

3.1 Datasets

We have experimented with the orginial MNIST dataset, and also its following variations as given in [3].The data is available at the following resource [3]

3.2 Mentor Network

The mentor network is built using theano [4] with one hidden layer of 500 neurons with tanh activation. The input layer has 784 neurons which is consistent with the number of input of MNIST dataset. The output layer is a softmax layer. The cost function is given by the log likelihood error with L1 and L2 regularization. The mentor is only trained with the original MNIST dataset. The learning rate is kept at 0.01 with batch size of 20. The best parameters for the mentor network are saved to be used while mentoring the various model.

3.3 Mentee Network

The mentee network has the same architecture as the mentor layer. The probes are made from the single hidden layer and output layer of mentor to mentee network. At each iteration the same data is fed first to the mentor layer the outputs of the hidden and output layer recorder, after which the original data is fed to mentee network. The errors for the for each probe is calculated and is used in the cost function. As we are using just one hidden layer and one output softmax layer as probe our cost function is given by the following function.

$$e = \alpha_t Ł_s(d_1^b) + \beta_t * \varphi(l,j) + \gamma_t \varphi(n,m) \tag{6}$$

And the corresponding update function is given by:

$$w^{t+1} = w^t - \eta[\alpha_t \frac{\partial}{\partial\omega} Ł_s + \beta_t \frac{\partial}{\partial\omega}\varphi(l,j) + \gamma_t \frac{\partial}{\partial\omega}\varphi(n,m)] \tag{7}$$

3.4 Hyperparameters and Learning Rate

The hyper parameters of alpha, beta, gamma are updated every epoch depending on the configuration. The parameters are an interesting set with the its value regularizing both the model and also the learning of the dataset. Three main type of configurations as given by [2] are as follows:

- **Adamant Network:** This network has the configuration where it is learning from both the architecture and the label of the training dataset. The configuration gives more importance to the label dataset and has a consistent annealing function as we move forward in the epochs. The 2 graph shows one such configuration.
- **Obedient Network:** This network has the configuration where it learns from the architecture first more than the labels. It slowly starts learning more as we move forward. The obedient network is an ideal network to understand the real power of mentee network. The configuration is similar to the adamant network it has annealing function starting from a low value for beta and gamma. Alpha starts with a low value and is an increasing function till a point and then starts annealing again. The 1 graph shows one such configuration.
- **Gullible Network:** This network can be looked at in an unsupervised learning perspective as we give no label information. This is a where the mentee just learns from the representation. This can be looked as a good initializer for neural network parameters. The 3 graph shows one such configuration.

These parameters can be interchanged as per the requirement of the training. According to us the obedient network is the smartest as it starts with learning the representations and once it is well initialized it starts learning from the labels. The learning rate is set to low rate at the beginning, we diminish it further by dividing it by 10 after a crossing point. The future sections we will look into the different configurations and dataset we have tried and tested. There can be so many other ways that we can setup the training.

3.5 Temperaturee Softmax

The temperature softmax as shown in [1] is implemented using theano, with configurable temperature. When no temperature is mentioned the default softmax of theno is used.

4 Experimentation and Results

The experimental setup was designed using **Theano V0.1** and the programs were written by ourselves. The experiments were conduction on Windows10 system with NVIDIA GForce 940 and Ubuntu operating system with AMD Radeon. Framework for plotting various graph has been setup with matpolib library. The batch size of the MNIST dataset was 20, and stochastic gradient descent was applied on all the batches. No pre-processing was applied to the data. The motivation of the experiments was to learn about distillation in neural networks, understand how a network trained via mentoring can learn faster and better general features than a network that learns from scratch.

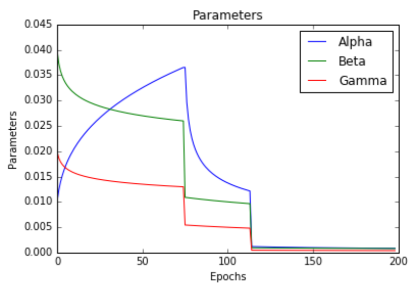

Fig. 1. Parameters : Obedient Mentee

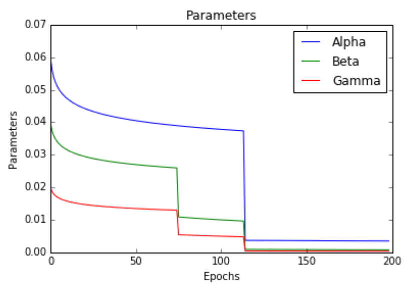

Fig. 2. Parameters : Adamant Mentee

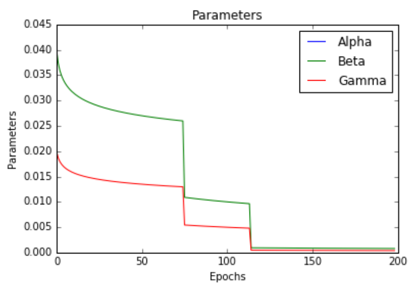

Fig. 3. Parameters : Gullible Mentee

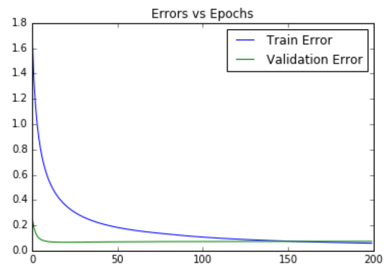

Fig. 4. Error Obedient

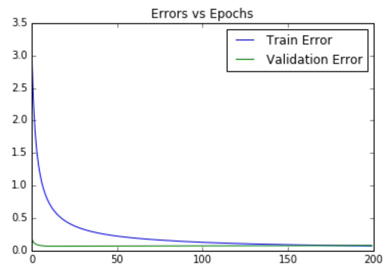

Fig. 5. Error Adamant

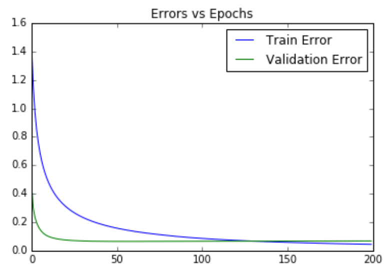

Fig. 6. Error Gullible

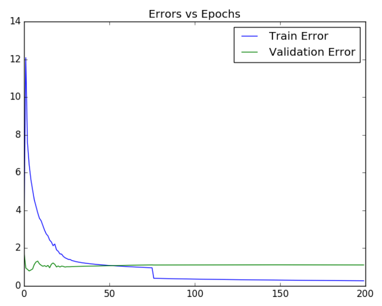

Fig. 7. Error:Adamant+Backround

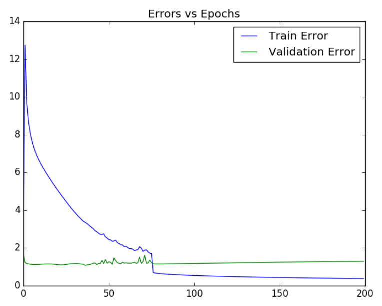

Fig. 8. Error:Adamant+Random Backround

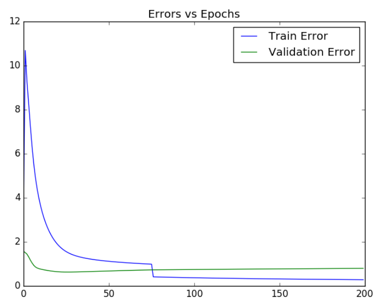

Fig. 9. Error:Adamant+Rotated Backround

As mentioned in [2] softmax layers as we constantly kept getting NaNs, so changing the temperature in the softmax layer as given in [1] helped out. The final temperature was set to 0.7 for all the experiments. The learning rate was initialized to 0.01 and was diminished by a factor of 10 after 115 epochs . We also divide the the learning by 100 when we encounter NaNs and restore the previous epochs parameters. The network architecture for both the mentee and mentor has been same as mentioned before. The details of the experimentation are given in the table.

For the MNIST dataset we observe that it performs better than the mentor in all the three ccofingurations. The best result is for obedient networks which we thought will be the case as our prediction was that it was the smartest of the networks. The gullible network also performs well here this can be accounted to the fact the network of mentor and mentee is the same along with the dataset. The graphs in 5,4,6 show how training and validation score change with epochs, it is important to note here that both of these are mesaured with different cost, the validation being negative log likehood cost and the training being the cost given by equation 2.

TABLE 2
Results

		Test Error	
Dataset	**Configuration**	**Mentee**	**Mentor**
MNIST	Obedient	1.870 %	1.96%
MNIST	Adamant	2.070 %	1.96%
MNIST	Gullible	1.930 %	1.96%
Rotated MNIST	Obedient	19.40 %	15.996 %
Rotated MNIST	Adamant	19.038 %	15.996 %
MNIST + RBG	Obedient	24.576%	22.004 %
MNIST + RBG	Adamant	23.716 %	22.004 %
MNIST + RBG	Gullible	85.934 %	22.004 %
MNIST + BG	Obedient	25.512 %	29.526 %
MNIST + BG	Adamant	25.234 %	29.526 %
MNIST + BG	Gullible	84.708 %	29.526 %

For the variations the mentor for the mentee was trained only on mnist dataset, as we wanted the mentee to be smarter while training. Although the performance on mentor when running on the variation is better than mentee running with its mentor trained on MNIST, we can see that mentee is still performing well. The possible reason for adamant being best for mentee can be that it learns more from labels more from the beginning it predicts better for new datasets. The gullible performs horrible due to the simple fact that the mentor has never seen data like this and himself must be performing poorly. The adamant configurations resutls in the graph plotted in figure 9,7,8.

We can see that most graphs start to converge in the validation set way before the 200th epoch. We have not let the training stop when it converges on the validation sets, and the parameters changing very little. Our observations was that mentee networks used to converge way earlier and faster as compared to the mentor networks. We ran on two different systems so each gave differnet timings but overall the mentee was performing better than mentor.

5 Conclusions and Future Use

We have learned that distilling works very well for transferring knowledge from a large pre-trained neural network which is trained on generalized dataset into a smaller network which is related to solving a smaller or more explicit task. During the course of the project, we realized the difficulties in training a neural network which started running into NaN's if the regularization was not proper. Also, using this approach decreases the total training time for a new network significantly, also leading to better regularization of the learning weights. We feel that this concept of training a large ensemble network and using this network for training a smaller neural network for a specific problem has a lot of potential in tasks such as speech recognition, image analysis, natural language translation etc. In the future we will like to try out other configurations of the hyperparamters as there are combinotorial ways to do them. We can also train the mentee on the varitions of dataset and compare it to mentee using mentor trained on the same data. Other huge thing we can do is make the mentor larger and complex, and mentee smaller and simple and see how the mentee performs. Changing the dataset to different domain data can be done to see the performance

References

[1] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, 2015.

[2] R. Venkatesan and B. Li, “Diving deeper into mentee networks,” *arXiv preprint arXiv:1604.08220*, 2016.

[3] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, “An empirical evaluation of deep architectures on problems with many factors of variation,” in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 473–480.

[4] T. MLP, “Multilayer Perceptron,” http://deeplearning.net/tutorial/mlp.html, 2016, [Online; accessed 28-Oct-2016].