Stacked Training for Overfitting Avoidance in Deep Networks

Alexander Grubb AGRUBB@CMU.EDU
J. Andrew Bagnell DBAGNELL@CS.CMU.EDU
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 USA

Abstract

When training deep networks and other complex networks of predictors, the risk of overfitting is typically of large concern. We examine the use of *stacking*, a method for training multiple simultaneous predictors in order to simulate the overfitting in early layers of a network, and show how to utilize this approach for both forward training and backpropagation learning in deep networks. We then compare this approach to overfitting avoidance with the dropout method for a number of common tasks.

1. Introduction

A deep network is composed of a sequence of layers, or predictors, that are successively applied to inputs to obtain final predictions. Each of these layers typically takes weaker representations from previous layers and then computes a more complex representation from that, allowing for powerful overall representations that are useful for solving difficult prediction problems.

Training these models, however, is a delicate matter, as each layer relies on the accuracy of other layers when optimizing its internal representation. At training time, small inaccuracies in other layers may be exploited to improve overall performance, but when run on unseen test data these inaccuracies can compound and create very different predictions and often poor test set performance.

This problem is widely recognized in the deep networks community, particularly due to the global fine-tuning approaches that are used to train these networks, and many strategies such as regularization have been employed to combat this problem. Other approaches include unsupervised learning techniques and methods for perturbing or manipulating the inputs to generate more training data.

A recent development that has proven very successful is the dropout approach (Hinton et al., 2012). In this approach a neural network is trained by stochastically ignoring hidden units at each training iteration, allowing the trained network to avoid relying on interactions between multiple hidden units which are a common source of overfitting. The performance improvement of this approach has led to a number of extensions, such as maxout networks (Goodfellow et al., 2013), which are networks of hidden units specifically designed to be used in tandem with dropout training.

One related approach to this problem from a different domain is the concept of stacking (Wolpert, 1992; Cohen & Carvalho, 2005). Essentially this method trains multiple copies of a given predictor while holding out portions of the dataset, in a manner similar to cross-validation. Each predictor is then run on the held-out data, mitigating the impact of overfitting on those predictions. This approach has proven to be useful in structured prediction settings (Cohen & Carvalho, 2005; Munoz et al., 2010) such as computer vision, where it is common to build sequential predictors which use neighboring and previous predictions as contextual information to improve overall performance.

It is this stacking approach which we will examine and extend to two common training scenarios for deep network: forward training and backpropagation learning.

2. Stacked Forward Training

In forward training, a deep network is trained using a sequential approach that trains each successive layer iteratively using the outputs from previously trained layers. This approach is common in structured prediction tasks such as vision where iterated predictions are used allow for smoothing of neighboring regions or when the structure of lower level features is selected apriori and trained independently of later layers. This is also a common approach to pre-training many deep

Appearing at the ICML 2013 Workshop on Representation Learning.

network architectures, such as the contrastive divergence learning used in pre-training deep Boltzmann machines (Salakhutdinov & Hinton, 2009).

Assume we are given a dataset $\mathcal{S}^0$ of examples and labels $\{(x_n, y_n)\}_{n=0}^N$.

We model a deep network of K layers as a sequence of predictors $f^1, \ldots, f^K$, with the output of layer k given as

$$x_n^k = f^k(x_n^{k-1}), \tag{1}$$

with the initial input $x_n^0 = x_n$.

Assume that for each layer k, there is a learning algorithm $\mathcal{A}^k(S)$ for generating the predictor f^k for that layer, using predictions from the previous layer x_n^{k-1} and labels y_n. That is, having trained the previous layers $f^1, \ldots, f^{k-1}$, the next layer is trained by building a dataset

$$\mathcal{S}^k = \{(x_n^{k-1}, y_n\}_{n=0}^N, \tag{2}$$

and then training the current layer

$$f^k = \mathcal{A}^k(\mathcal{S}^k). \tag{3}$$

This method is not robust to overfitting, however, as errors in early predictors are re-used for training later predictors, while unseen test data will likely generate less accurate predictions or low level features. If early predictors in the network overfit to the training set, later predictors will be trained to rely on these overfit inputs, potentially overfitting further to the training set and leading to a cascade of failures.

The stacking method (Cohen & Carvalho, 2005) is a method for reducing the overall generalization error of a sequence of trained predictors, by attempting to generate an unbiased set of previous predictions for use in training each successive predictor. This is done by training multiple copies of each layer on different portions of the data, in a manner similar to cross-validation, and using these copies to predict on unseen parts of the data set.

More formally, we split the dataset $\mathcal{S}^{\|}$ in to J equal portions $\mathcal{S}_1^k, \ldots, \mathcal{S}_J^k$, and for each layer f^k train an additional J copies f_j^k. Each copy is trained on the dataset *excluding* the corresponding fold, as in J-fold cross-validation:

$$f_j^k = \mathcal{A}^k(\mathcal{S}^k \setminus \mathcal{S}_j^k). \tag{4}$$

Each of the copies is then used to generate predictions on the held-out portion of the data which are used to build the training dataset for the next layer:

$$\mathcal{S}^{k+1} = \cup_{j=1}^J \left\{(f_j^k(x), y) \mid |(x, y) \in \mathcal{S}_j^k\right\}. \tag{5}$$

Algorithm 1 Stacked Forward Training

Given: initial dataset $\mathcal{S}^0$, training algorithms $\mathcal{A}^k$, number of stacking folds J.
for $k = 1, \ldots, K$ **do**
  Let $f^k = \mathcal{A}^k(\mathcal{S}^k)$.
  Split $\mathcal{S}^k$ into equal parts $\mathcal{S}_1^k, \ldots, \mathcal{S}_J^k$.
  For $j = 1, \ldots, J$ let $f_j^k = \mathcal{A}^k(\mathcal{S} \setminus \mathcal{S}_j^k)$.
  Let $\mathcal{S}^{k+1} = \cup_{j=1}^J \left\{(f_j^k(x), y) \mid (x, y) \in \mathcal{S}_j^k\right\}$.
end for
return $(f^1, \ldots, f^K)$.

The predictor f_k for the current layer is still trained on the whole dataset $\mathcal{S}^k$, as in (3) and returned in the final model. The stacked copies are only used to generate the predictions for training the next layer, and are then discarded.

A complete description of stacked forward training is given in Algorithm refalg:stacked-forward.

3. Stacked Backpropagation

In the previous section we detailed the stacking method for forward training. Many deep networks do not use a purely forward training algorithm, however, as this prevents early stages of the representation from being adjusted to improve overall performance.

In this section we extend the stacking method to the most common approach to training whole networks simultaneously, backpropagation. We now assume that we are given a loss function on the final output

$$L[f^1, \ldots, f^k] = \sum_{n=1}^N l(x_n^K, y_n). \tag{6}$$

Define

$$h^k(x) = f^K(f^{K-1}(\ldots f^{k+1}(x))) \tag{7}$$

to be the result of evaluating everything after layer k. Typical backpropagation learning, given a training example (x_n, y_n) and a layer f^k, computes the current input x_n^{k-1} and the gradient

$$\nabla^k = \frac{\partial}{\partial f^k(x_n^{k-1})} l(h^k(f^k(x_n^{k-1})), y_n), \tag{8}$$

an then uses these values to update the current layer f^k. This is typically an activation function applied to linear weights:

$$f^k(x_n^{k-1}) = z(\theta^{k^T} x_n^{k-1}), \tag{9}$$

Algorithm 2 StackProp: Stacked Backpropagation Update

Given: current estimates for $f^k, f_j^k, \forall j, k$, training example (x_n, y_n)
Let $(x_n, y_n) \in \mathcal{S}_{j^*}$.
for $k = 1, \ldots, K$ **do**
  Compute $\hat{x}_n^{k-1}$ from x_n using $f_{j^*}^1, \ldots, f_{j^*}^{k-1}$ as in (10).
  Compute ∇^k using $f^{k+1}, \ldots, f^K$ as in (11).
  Similarly, compute ∇_j^k for all $j \neq j^*$ as in (12).
end for
for $k = 1, \ldots, K$ **do**
  Update f^k using $\hat{x}_n^{k-1}$ and ∇^k.
  Update f_j^k using $\hat{x}_n^{k-1}$ and ∇_j^k, for all $j \neq j^*$.
end for

which is updated using gradient descent, but other approaches exist for updating networks of arbitrary functions using gradient information (Grubb & Bagnell, 2010).

In the case of stacked backpropagation, we will again attempt to generate unbiased inputs $\hat{x}_n^{k-1}$ for each layer. Like before, we will train a copy of each layer f_j^k for each of the J splits of the training set, and use these copies on held-out data to generate the unbiased inputs.

Split the initial dataset in to J partitions, $\mathcal{S}_1, \ldots, \mathcal{S}_J$. Letting the initial unbiased input $\hat{x}_n^0 = x_n$, we can define the unbiased input to each layer as

$$\hat{x}_n^k = f_j^k(\hat{x}_n^{k-1}), \text{ where } (x_n, y_n) \in \mathcal{S}_j \qquad (10)$$

The gradient is still computed using the normal, *i.e.*unstacked predictors for each layer, with the only modification being the use of the unbiased inputs:

$$\hat{\nabla}^k = \frac{\partial}{\partial f^k(\hat{x}_n^{k-1})} l(h^k(f^k(\hat{x}_n^{k-1})), y_n). \qquad (11)$$

For a given example in $(x_n, y_n) \in \mathcal{S}_{j^*}$, we also have to update all the stacked layers in other partitions $j \neq j^*$, using the input $\hat{x}_n^{k-1}$ and the gradient

$$\hat{\nabla}_j^k = \frac{\partial}{\partial f_j^k(\hat{x}_n^{k-1})} l(h^k(f_j^k(\hat{x}_n^{k-1})), y_n). \qquad (12)$$

Algorithm 2 gives the backpropagation update for a single training example, but the same method can be easily extended for other scenarios such as minibatch training.

	Baseline	Stacking	Dropout
SBD	80.49%	81.86%	80.54%
CamVid	83.53%	84.94%	83.25%

Table 1. Test set accuracy (average across 5 folds) on the Stanford Background Dataset (SBD) and Cambridge Video Dataset (CamVid) using the HIM (Munoz et al., 2010) approach combined with baseline forward training, stacking, and dropout.

4. Experimental Results

We now present results comparing stacked learning to both dropout and a baseline forward training of backpropagation approach.

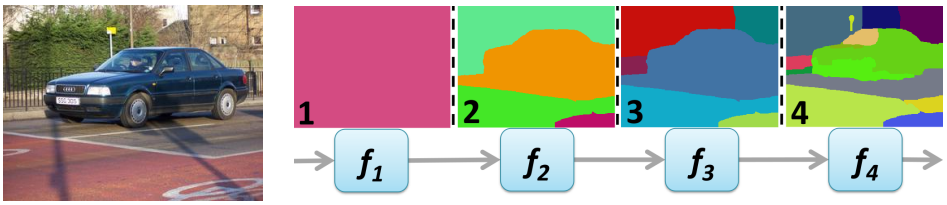
4.1. Forward Training for Structured Prediction

We applied standard forward training, stacked training, and dropout to a structured prediction task in the scene labeling domain. In this domain the goal is to label every pixel of an image with a label, *e.g.*car, building, sky. For this task, we used the sequential network approach of Munoz, *et al.*(2010) known as Hierarchical Inference Machines (HIM). We use the same setup and stacking approach as described in their previous work, detailed briefly in Figure 1.

In this approach a number of predictors are trained in a sequential manner on a hierarchy of segmentations for each input image. Each layer takes as inputs features of the image, along with predictions from the previous layer on the current image region and neighboring regions, and predicts the label for that portion of the image.

For the dropout approach to this problem, we use the same hierarchical network and training setup as in (Munoz et al., 2010), but when building features based off of previous predictions the predictions are dropped out of the feature vector (by setting the feature to 0) with probability $p = 0.1$. In the forward training setting, larger values of p such as the $p = 0.5$ used in the typical backpropagated dropout approach result in too much noise being introduced in to the dataset, as the features from the previous layer are only trained on once, unlike the backpropagation approach where the same example is sampled many times.

Table 1 gives the average test set accuracy for each of the three forward training approaches on the Stanford Background Dataset (SBD) (Gould et al., 2009) and Cambridge Video Dataset (CamVid) (Brostow et al., 2008).



(a) (b)

Figure 1. Hierarchical Inference Machines (Munoz et al., 2010). (a) Input image. (b) The image is segmented multiple times; predictions are made and passed between levels. Images courtesy of the authors' ECCV 2010 presentation.

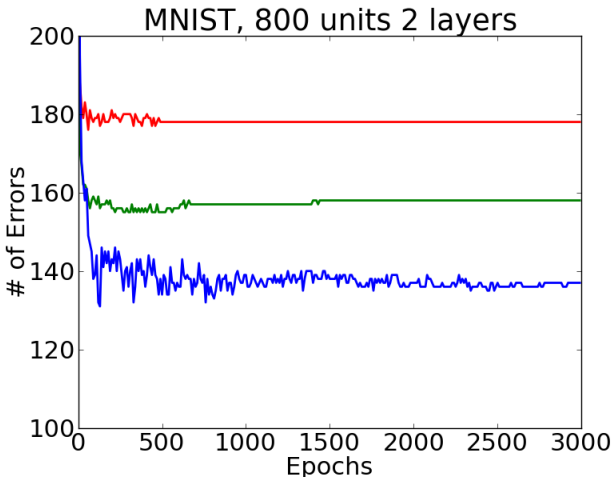


Figure 2. Number of test set errors for the MNIST dataset for baseline backpropagation (red), StackProp (green) and dropout (blue), using an 800 unit, 2 layer network.

4.2. Backpropagated Neural Networks

To analyze the performance of stacked backpropagation, we use it to train a number of standard feed-forward networks for the MNIST dataset and two datasets from the UCI Machine Learning Repository (Frank & Asuncion, 2010), the 'letter' and 'pendigits' datasets.

We train all models and methods using stochastic gradient descent on minibatches of 100 examples each. The specific learning rate and momentum settings are the same as those given in (Hinton et al., 2012) for all models. Namely, the learning rate is decayed using

$$\eta_t = 0.998^t \tag{13}$$

and a momentum parameter that smoothly increases from 0.5 to 0.99 over the first 500 training epochs. We also use the same method of constraining the layer weights using a hard L2 constraint of 15.0 for each unit.

The stacked version uses 10 validation folds for training the stacked layers, and the dropout approach uses a hidden unit dropout probability of 0.5. No dropout is used on the input features for any model.

Figure 3 and Figure 4 give the results for UCI 'letter' and 'pendigits' datasets, respectively, using all three methods with two different networks: a two layer and three layer network, each with a width of 500 hidden units.

Figure 2 gives the results for the MNIST dataset using the three methods (baseline backpropagation, stacking and dropout) with a two layer architecture with 800 hidden units.

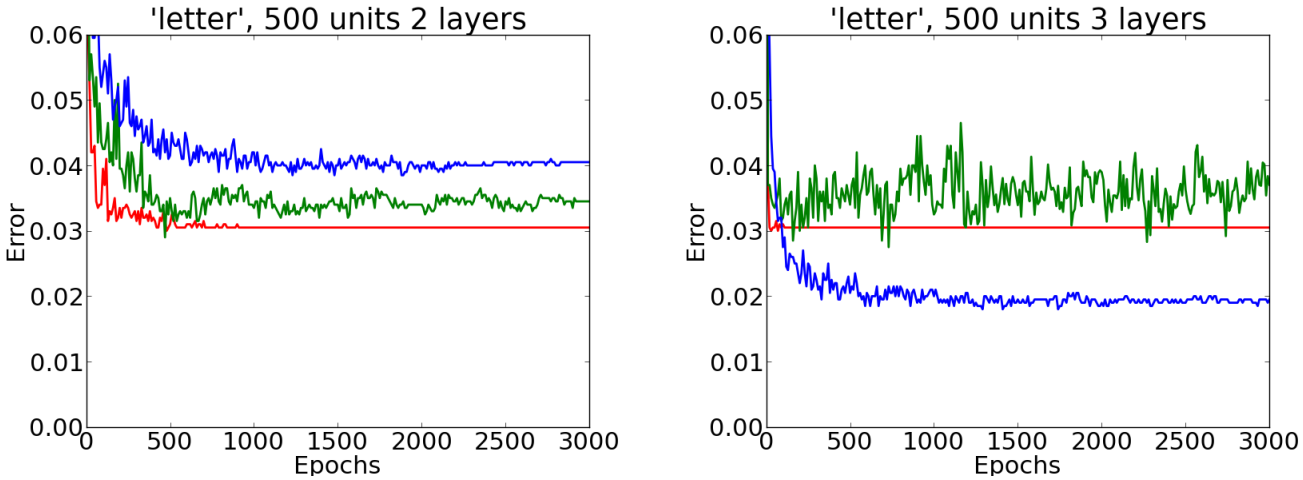
5. Discussion

Overall, we see that stacked training has the most benefit in forward training, where it substantially outperforms the baseline and dropout approaches. This comes as no surprise as stacking is originated as method of overfitting avoidance in structured and sequential prediction problems.

In the backpropagation setting, stacking can sometimes improve the robustness of a network, but it is almost always dominated by the dropout approach. It also seems that the stacking approach to backpropagation detailed here has substantial convergence problems when dealing with deeper networks, as both of the three layer networks results for stacking were worse than their respective baseline approaches, while dropout excelled when used with these networks.

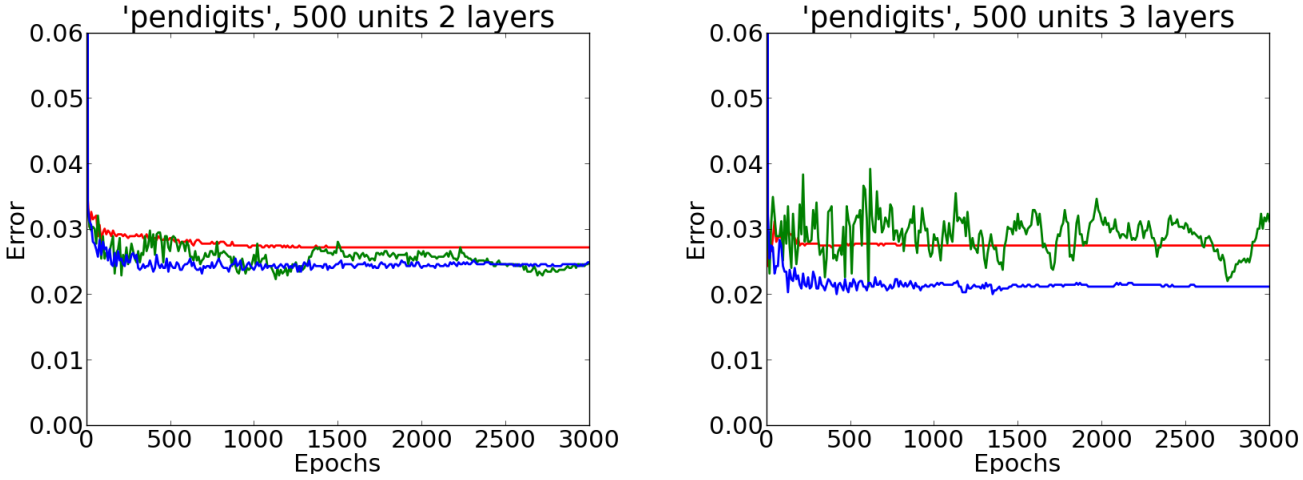
The added complexity of stacked training, both in terms of implementation overhead and added computational cost, coupled with the often poorer performance than dropout, seemingly makes it a poor choice currently for fine-tuning deep networks.

The large benefit seen in forward training leaves open the possibility of future work examining the use of stacking in other forward training settings commonly used for training deep networks, such as in the pre-training stages used for most deep networks. Fur-



(a) (b)

Figure 3. Test set error rate for the UCI 'letter' dataset for baseline backpropagation (red), StackProp (green) and dropout (blue), using a (a) 500 unit, 2 layer network and (b) 500 unit, 3 layer network.



(a) (b)

Figure 4. Test set error rate for the UCI 'pendigits' dataset for baseline backpropagation (red), StackProp (green) and dropout (blue), using a (a) 500 unit, 2 layer network and (b) 500 unit, 3 layer network.

ther study of the impact of overfitting in these early training stages, and whether stacking can alleviate this overfitting, is a promising extension of this work.

Acknowledgements

We would like to thank Daniel Munoz and Stephane Ross for their helpful discussions and feedback.

References

Brostow, Gabriel J., Shotton, Jamie, Fauqueur, Julien, and Cipolla, Roberto. Segmentation and recognition using structure from motion point clouds. In *ECCV*, 2008.

Cohen, William W. and Carvalho, Vitor. Stacked sequential learning. In *IJCAI*, 2005.

Frank, A. and Asuncion, A. UCI machine learning repository, 2010. URL http://archive.ics.uci.edu/ml.

Goodfellow, Ian J., Warde-Farley, David, Mirza, Mehdi, Courville, Aaron C., and Bengio, Yoshua. Maxout networks. *CoRR*, abs/1302.4389, 2013.

Gould, Stephen, Fulton, Richard, and Koller, Daphne.

Decomposing a scene into geometric and semantically consistent regions. In *ICCV*, 2009.

Grubb, A. and Bagnell, J. A. Boosted backpropagation learning for training deep modular networks. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

Hinton, Geoffrey E., Srivastava, Nitish, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

Munoz, D., Bagnell, J. A., and Hebert, M. Stacked hierarchical labeling. In *European Conference on Computer Vision*, 2010.

Salakhutdinov, Ruslan and Hinton, Geoffrey E. Deep boltzmann machines. *Journal of Machine Learning Research - Proceedings Track*, 5:448–455, 2009.

Wolpert, David H. Stacked generalization. *Neural Networks*, 5(2), 1992.