Deep structured modeling of deep learning training convergence with application to hyperparameter optimization

Lev Faivishevsky [1] **Amitai Armon** [1]

Abstract

Modeling the convergence process of machine learning training is an important tool for large scale algorithms acceleration. We introduce a deep structured method based on LSTM for prediction of the test set accuracies after each training iteration. We use it as a tool for large scale hyperparameter optimization. Applied to a real world multinode implementation speeding up task the method achieved promising results.

1. Introduction

The growing scale of modern machine learning methods raised the importance of modeling and acceleration of convergence for iterative training processes. Common strategies for modeling the convergence include extrapolation of sequence of the partial accuracies by weighted probabilistic models (Domhan et al., 2015) and Gaussian processes (Swersky et al., 2014). The model of convergence process is used for early stopping and for hyperparameter optimization tasks (Domhan et al., 2015).

In this paper we propose a novel approach for modeling the convergence of the training process by deep structured prediction based on LSTM. Using this modeling method we introduce a new hyperparameter optimization algorithm, which lends itself well for tuning large scale deep learning tasks for the fastest runtime in the training stage.

2. Deep structured prediction for modeling learning curves

Many modern machine learning algorithms are trained in an iterative manner. This way an initial solution for the training optimization task is gradually improving through a number of iterations. The quality of the solution is assessed during the training iterations by applying the underlying machine learning algorithm on a test set. Training an iterative machine learning algorithm G yields a sequence of testing set accuracy estimates $A_1, A_2, .., A_i .., A_N$, where A_i is the testing accuracy of the algorithm after i training iterations and N is the total number of training iterations. Such sequence of accuracy estimates is dubbed *Learning curve*, see (Domhan et al., 2015). The accuracy of the completely trained algorithm is given obviously by A_N. The prediction of final accuracy A_N is of high importance for practical applications, yet there is an important additional value in modeling the whole learning curve.

Each machine learning algorithm includes a set of configuration parameters (or hyperparameters) $H_1, H_2, ..., H_K$, such that a proper setting of their values influences the accuracy and the running time of the algorithm. For example, training a deep learning algorithm usually requires a proper setting of hyperparameters such as learning rate, batch size and others. As a consequence the learning curve values may be considered as a structured output which is dependent on a set of chosen hyperparameter values $A_1, ..., A_N \leftarrow G(H_1, ..., H_K)$. This motivates applying recurrent neural networks for modeling the sequence of dependent outputs as a function of the hyperparameters.

We employ the deep neural network in the following manner. The hyperparameters $H_1, H_2, ..., H_K$ constitute the input layer, which is followed by a number of fully connected layers. As the modeled dependency is highly nonlinear a number of fully connected neural network layers are used to obtain an optimal representation of the configuration for the subsequent computation. This computation is performed by a recurrent neural network of type LSTM (Hochreiter & Schmidhuber, 1997). The last fully connected layer serves as a projection layer for the LSTM, then the LSTM performs N time steps, so that it outputs accuracy prediction A_i at time step i.

The resulting model, which we dub *DNNLSTM*, predicts accuracies as a differentiable function of hyperparameters which is significant for the applications, see Figure 1.

[1]Intel Advanced Analytics. Correspondence to: Lev Faivishevsky <lev.faivishevsky@intel.com>.

Proceedings of the ICML 17 Workshop on Deep Structured Prediction, Sydney, Australia, PMLR 70, 2017.

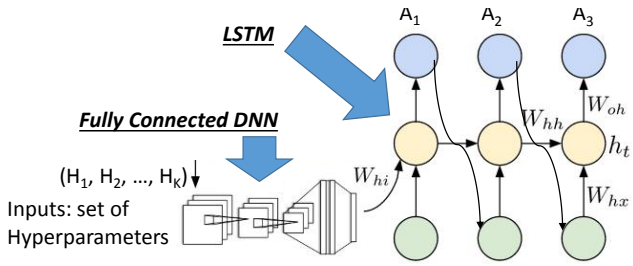

Figure 1. DNNLSTM model for accuracies prediction as a function of hyperparameters.

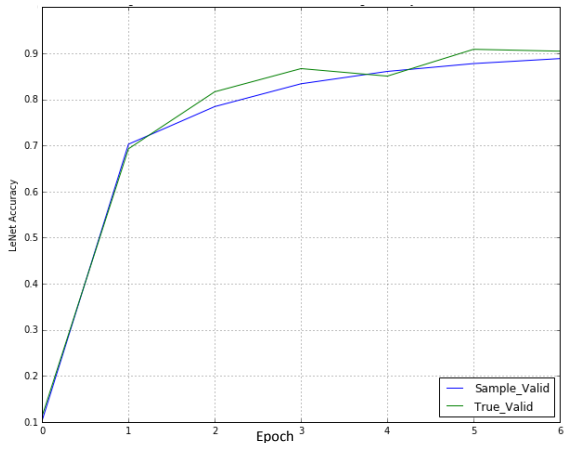

Figure 2. Example: predicting learning curve for LeNet, learning rate = 0.01, momentum = 0.925, weight decay = 0.0005

3. Hyperparameter optimization by learning curves modeling

The proposed DNNLSTM model for structured output prediction $A_1(H_1, ..., H_K), ..., A_N(H_1, ..., H_K)$ serves as a useful tool for the important task of hyperparameter optimization. In the common use case the target is to maximize the final accuracy. In this paper we address hyperparameter optimization in a different setup. We solve the problem of setting the algorithms hyperparameters for the fastest running time while maintaining its desired final accuracy A^* by multivariate optimization. The hyperparameters are considered to be the variables, and the target function to be minimized is the running time of the algorithm required to achieve a fixed value of accuracy. Then we minimize:

$$\begin{aligned} H_1^*, ...H_K^* &= \arg\min_{H_1,...,H_K} t \\ \text{s.t.} \quad A_t &\geq A^* \end{aligned} \tag{1}$$

To the best of our knowledge, this constrained formulation of hyperparameter optimization was only considered before in (Faivishevsky & Armon, 2016). As a first stage to employ this approach we train the DNNLSTM model. Second, we apply an optimization method to Equation 1. Third, we define a scalable version of the algorithm.

3.1. Training DNNLSTM model

As we seek for the functional dependency of a machine learning algorithm hyperparameters $H_1, ..., H_K$ and its test set accuracies $A_1, ...A_N$ where K is usually small, we may define an initial grid of M sets of hyperparameters values $H_1^m, ..., H_K^m$ and run training of G M times in order to produce sequences $A_1^m, ..., A_N^m$. Then we use M pairs $H_1^m, ..., H_K^m; A_1^m, ..., A_N^m$ in order to train the DNNLSTM model. As A_i takes real values the simple choice is to train LSTM regression model with l_2 loss. More elaborate approaches may consider training the LSTM on delta accuracies sequences: $H_1^m, ..., H_K^m; A_2^m - A_1^m, ..., A_N^m - A_{N-1}^m$. The regression task may also be casted as a classification problem by quantization of accuracy values into B bins. Yet another solution may be utilizing mixture output networks that learn probability distributions of output variables by means of Gaussian mixtures (Schuster, 2000)

3.2. Optimization method

The optimization problem 1 requires solving K variables minimization for finding an optimal set of hyperparameters $H_1^*, ..., H_K^*$. A trained DNNLSTM model gives a differentiable functional dependency of accuracies A_i on hyperpameters H_k therefore a natural strategy would be employing a common nonlinear constraint optimization method like Interior point or Trust region (Byrd et al., 2000).

A simpler approach might be fixing a subset of possible epoch numbers $i_1 < ... < i_t < ... < i_T$ and then solving T nonlinear unconstrained problems

$$H_1^{t*}, ..., H_K^{t*} = \arg\max_{H_1,...,H_K} A_{i_t} \tag{2}$$

by a nonlinear unconstrained optimization method like BFGS method (Liu & Nocedal, 1989). The final solution $H_1^*, ..., H_K^*$ corresponds then to smallest t such that $A_{i_t}(H_1^t, ..., H_K^t) \geq A^*$

3.3. Large Scale application

The ability to scale out is a important advantage for a hyperparameter optimization. The reason is that modern cloud facilities include a vast amount computing nodes (often several hundreds of thousands) whereas best multinode implementations for a single training run of a modern deep learning algorithm are often limited up to several dozens of nodes. In this case an ability to utilize parallel training executions of the underlying machine learning algorithm G is

Algorithm 1 Hyperparameter Optimization by Deep Structured Prediction (HODSP), large scale parallelization

Inputs: Algorithm G, accuracy A^*, hyperparameters $H_1, ..., H_K$, grid of M sets $\{H_i^m\}_{m=1}^M$
Parallel initialization of DNNLSTM train set D
for $m = 1$ **to** M **do**
  $\{A_i^m\}_i^N \leftarrow G(\{H_k^m\}_{k=1}^K)$
  $D \bigcup = \{H_k^m\}_{k=1}^K; \{A_i^m\}_{i=1}^N$
end for
repeat
  Train DNNLSTM (Fig. 1) on D
  Solve Eq. 1 to find M^* candidates $\{H_i^m\}_{m=1}^{M^*}$
  for $m = 1$ **to** M^* **do**
    $\{A_i^m\}_i^N \leftarrow G(\{H_k^m\}_{k=1}^K)$
    $D \bigcup = \{H_k^m\}_{k=1}^K; \{A_i^m\}_{i=1}^N$
  end for
until A stopping criterion is satisfied

significant for a hyperparameter optimization method.

The proposed method lends itself well into such a framework. First, the initial execution of M runs for the initial training of DNNLSTM may obviously done in the parallel manner. Then an optimization step 1 may output not only a single optimal point $H_1^*, ...H_K^*$ but a set of M^* potential candidates $H_1^{m^*}, ...H_K^{m^*}$, where m ranges from 1 to M^*. As a validation stage the machine learning algorithm training is then executed using these M^* candidates, so that there are additional M^* pairs of hyperparameters and corresponding sequences of testing accuracies. Obviously, a reasonable next step is to retrain the DNNLSTM model using all available $M + M^*$ input-output pairs. As the training set grows the quality of the DNNLSTM improves. We repeat the above steps of optimization 1, validation by machine learning algorithm parallel training and DNNLSTM retraining until a stopping criterion is satisfied. We summarize the algorithm's pseudocode which we dub *Hyperparameter Optimization by Deep Structured Prediction* (HODSP) in Algorithm 1

4. Experiments

We performed a number of experiments to assess the quality of DNNLSTM model for modeling learning curves of deep learning algorithms. After that we employed the HODSP for the hyperparameter optimization of a multinode implementation of a large scale convolutional network

4.1. Modeling learning curves by DNNLSTM

First we applied the DNNLSTM to modeling learning curves of LeNet (Lecun et al., 1998) on MNIST dataset. We studied the dependency of the test set accuracies on learning rate, momentum and weight decay. We varied the learning rate from 0.001 to 0.01, the momentum from 0.85 to 0.999 and weight decay from 0.0005 to 0.001. Overall we used 100 sets of hyperparameters for training DNNLSTM and then we measured the prediction performance for other 34 hyperparameters sets. To provide a conservative setup we split training and testing sets so that the hyperparameter sets with learning rates 0.0004 and 0.012 are used for the testing only, whereas other learning rates are used only for the training. We trained LeNet 134 times for 7 epochs in order to get the set of pairs $\{H_1^m, H_2^m, H_3^m\}; \{A_1^m, ..., A_7^m\}$

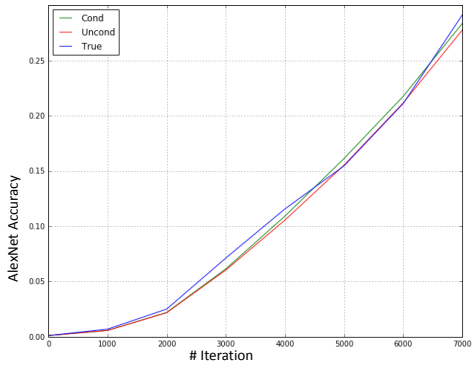

Figure 3. Predicting learning curve for AlexNet, learning rate = 0.007, momentum = 0.89, weight decay = 0.0005

We used the following configuration of the DNNLSTM. DNN had 4 layers with hidden layers of dimensions 50, 15, 5 and ReLU activations, the last hidden projection layer had size 30. The LSTM had hidden state size and cell size of 30. For the training we used Adam optimizer with l_2 loss. We made 1000 epochs with batch size 100 and initial learning rate 3e-3 with learning rate decay of 0.995. After training DNNLSTM we sampled 34 predicted sequences for the validation hyperparameter sets.

To measure the quality of the modeling we measured RMSE for the prediction of LeNet testing accuracy. We obtained RMSE of 0.34% for 100 training sequences and RMSE of 0.48% for 34 validation sequences. We present an example of a modeled learning curve in Figure 2.

Next we applied the DNNLSTM to modeling learning curves of AlexNet (Krizhevsky et al., 2012) trained on Imagenet as functions of learning rate, momentum and weight decay. We varied the learning rate from 0.005 to 0.015, the momentum from 0.89 to 0.915 and weight decay from 0.0001 to 0.001. Overall we used 144 sets for training DNNLSTM and measured the prediction performance for other 66 hyperparameters sets. The hyperparameter sets with learning rates 0.007, 0.01 and 0.013 are used for the validation only, whereas other learning rates are used only

Table 1. Hyperparameter optimization for multinode GoogleNet training. 32 nodes of 2 socket system with Intel Xeon Processor E5-2699 v4 (22 Cores, 2.2 GHz), Intel Caffe, Intel MKL 2017

	FIRECAFFE	HODSP
BATCH SIZE	1024	1536
BASE LEARNING RATE	0.08	0.0898
MAX ITERATIONS	91000	62000
WEIGHT DECAY	0.0002	0.0001
TRAINING TIME, MINUTES	883	**796**

for the training. We trained AlexNet for 7000 iterations and measured test accuracy each 1000 iterations.

The DNN part of DNNLSTM had 5 layers with hidden layers of dimensions 50, 25, 15, 5, the last hidden projection layer had size 10. For the training we used Adam optimizer with l_2 loss. We made 500 epochs with batch size 30 and initial learning rate 4e-3 with learning rate decay of 0.995.

We sampled the DNNLSTM model in two setups. In the first case, which we dub *uncoditional*), we used only hyperparameters to predict sequence of testing accuracies as usual. In the second case, which we dub *conditional*, we predicted the testing accuracy one by one, feeding hyperparameters to the input layer of DNN and plugging true values of $\{A_1, ..., A_{i-1}\}$ to LSTM to predict A_i.

We obtained RMSE of 0.86% for the uncoditional prediction of 66 validation sequences and RMSE of 0.77% for the conditional case. We remark that lower RMSE for the conditional prediction is an expected outcome, because we use more recent information in prediction. We put an example of modeled learning curves of AlexNet in Figure 3.

4.2. Large Scale HyperParameter optimization by HODSP

We employed the proposed method for the real world problem of hyperparameter optimization for 32-nodes implementation of GoogleNet (Szegedy et al., 2015) trained on ImageNet dataset. We initialized a grid of 50 hyperparameters sets and ran in parallel GoogleNet executions. We trained DNNLSTM and used the trained model to optimize the target of Eq. 2 by BFGS and identified 25 candidate data sets for the validation parallel training runs of GoogleNet. After only 3 repetitions of the above process we obtained a significant improvement of 10% training running time needed to achieve state-of-art multinode top-1 accuracy with respect to the baseline training time (Iandola et al., 2015), see Table 1.

5. Conclusion

In this paper we introduced deep structured prediction method for modeling convergence of modern machine learning algorithms. Using it we defined a new hyperparameter optimization technique, which was applied to a real world task. The results demonstrated a significant improvement with regard to the commonly used baseline algorithm.

References

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 2012.

Byrd, R. H., Gilbert, J. C., and J., Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 2000.

Domhan, T., Springenberg, J. T., and Hutter, F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, 2015.

Faivishevsky, L. and Armon, A. Using downhill simplex method for optimizing machine learning training running time. In *NIPS Workshop for Artificial Intelligence for Data Science*, 2016.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Comput.*, 1997.

Iandola, F., K., Ashraf, M.W., Moskewicz, and K., Keutzer. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. *CoRR*, 2015.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, 2012.

Lecun, Y., L., Bottou, Y., Bengio, and P., Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 1998.

Liu, D. C. and Nocedal, J. On the limited memory bfgs method for large scale optimization. *MATHEMATICAL PROGRAMMING*, 1989.

Schuster, M. Better generative models for sequential data problems: Bidirectional recurrent mixture density networks. In *Advances in Neural Information Processing Systems 12*. 2000.

Swersky, K., J., Snoek, and R.P., Adams. Freeze-thaw bayesian optimization. *CoRR*, 2014.

Szegedy, C., L., Wei, Y., Jia, P., Sermanet, S., Reed, D., Anguelov, D., Erhan, V., Vanhoucke, and A., Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.