Learning Compact Networks via Adaptive Network Regularization

Sivaramakrishnan Sankarapandian
Proscia Inc.
siva@proscia.com

Anil Kag
Department of ECE
Boston University
anilkag@bu.edu

Rachel Manzelli
Department of ECE
Boston University
manzelli@bu.edu

Brian Kulis
Department of ECE
Boston University
bkulis@bu.edu

Abstract

Deep neural networks typically feature a fixed architecture, where the number of units per layer are treated as hyperparameters and tuned during training. Recently, strategies for training adaptive neural networks without a fixed architecture have seen renewed interest. In this paper, we employ a simple regularizer on the number of hidden units in the networks, which we refer to as adaptive network regularization (ANR). This method places a penalty on the number of hidden units per layer, designed to encourage compactness and flexibility of the network architecture. This penalty acts as the sole tuning parameter over the network size, increasing simplicity during training. We describe a training strategy that grows the number of units during training, and show on several benchmark datasets that our model yields architectures that are smaller than those obtained when tuning the number of hidden units on a standard fixed architecture. Along with smaller architectures, we show on multiple datasets that our algorithm performs comparable to or better than fixed architectures learned via grid-searching over the hyperparameters. We motivate this model using small-variance asymptotics—a Bayesian neural network with a Poisson number of units per layer becomes our model in the small-variance limit.

1 Introduction

A significant amount of effort is spent determining appropriate parameters of a deep neural network. When the parameters are discrete, determining the correct network architecture for a learning problem is typically performed via grid search. This expensive hyperparameter search involves training networks for each possible architecture and examining the resulting validation loss. Recently, methods that attempt to adapt the architectures of the model to a training set have regained traction, such as constructing a neural network using non-parametric priors [5, 3, 1], deriving a suitable neural network architecture using a non-probabilistic framework [6], and using an auxiliary neural network trained using reinforcement [2].

In this paper, we apply *small-variance asymptotics* (SVA) on a Bayesian neural network (BNN) with a non-fixed Poisson number of units per layer, yielding a simple non-probabilistic model where a loss function over a neural network is augmented with a penalty on the number of units in each layer. The tradeoff between the loss and the penalty are governed by one tuning parameter. This parameter eliminates the need for grid-searching the model architecture, which simplifies training

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

while additionally encouraging compactness and flexibility in the network architecture size. Unlike standard search methods for architecture size, where the number of units is a discrete hyperparameter, our tuning parameter is continuous. As a result, other methods such as Bayesian optimization could be incorporated to tune this hyperparameter. We refer to our method as adaptive network regularization (ANR).

We then empirically demonstrate that our resulting approach yields an algorithm for training a neural network whose architecture is not fixed, while providing an easy method for controlling the compactness. We show that our algorithm achieves performance comparable to or better than models where the architecture has been fixed. Surprisingly, even when we train models whose architectures are the same as those that our algorithm learns (i.e., we retrain using standard backpropagation on the architectures that our algorithm learns), we find that our loss is typically lower, indicating that from an optimization perspective, there may be benefits to training a network by adding neurons via this penalty parameter. We believe that our approach will yield further insights in the future to design richer neural network models inspired by probabilistic counterparts.

2 Adaptive Network Regularization

We motivate our approach to adaptively determining the architecture of a Bayesian neural network via small-variance asymptotics. We discuss the case when the number of units are treated as Poisson random variables.

2.1 SVA for Bayesian Neural Networks

From a generative perspective, the standard Bayesian neural network can be viewed as a generative process where the weights are drawn from the prior, and then the outputs are generated from the likelihood distribution given the weights. We can now consider the case where the weights within each layer ℓ are further conditioned on a random variable $k^{(\ell)}$ corresponding to the number of hidden units in layer ℓ. We will place a Poisson distribution on $k^{(\ell)}$; assuming there are L layers total, this makes the model (in the regression setting) therefore

$$\begin{aligned} p(k^{(\ell)}) &= \text{Pois}(\lambda), \quad \ell = 1, ..., L \\ p(\boldsymbol{w}^{(\ell)}|k^{(\ell)}) &= \mathcal{N}(0, I_{k^{(\ell)}}), \quad \ell = 1, ..., L \\ p(y_i|\boldsymbol{w}, \boldsymbol{x}_i\sigma^2) &= \mathcal{N}(\text{NN}(\boldsymbol{x}_i; \boldsymbol{w}), \sigma^2), \end{aligned}$$

where $\boldsymbol{w}^{(\ell)}$ corresponds to the weights in layer ℓ. The definition of the variables $(k^{(\ell)}, \boldsymbol{w}^{(\ell)})$ imply that the joint probability $p(k^{(\ell)}, \boldsymbol{w}^{(\ell)}) = p(\boldsymbol{w}^{(\ell)}|k^{(\ell)})p(k^{(\ell)})$.

If we make no further assumptions about the rate parameter λ, then in the small-variance limit, the prior over the number of units will vanish, as the prior over the weights vanishes in the standard BNN case, resulting in an ill-defined model. However, if we assume that $\lambda = \exp(-\gamma/\sigma^2)$, then observe that

$$-\log p(k^{(\ell)}|\lambda) = \frac{\gamma}{\sigma^2} \cdot k + \exp(-\gamma/\sigma_p^2) + \log k^{(\ell)}!$$

As $\sigma \to 0$ as in the previous section, the first term of this expression dominates; indeed, one can easily see that the MAP inference problem becomes simply

$$\sum_{i=1}^{n} (\text{NN}(\boldsymbol{x}_i; \boldsymbol{w}) - y_i)^2 + \gamma \cdot \sum_{\ell=1}^{L} k^{(\ell)}, \tag{1}$$

that is, the standard regression error with an additional term that regularizes the number of hidden units per layer. Note that the analysis here need not be restricted to the regression setting, and can incorporate arbitrary loss functions.

2.2 ANR algorithm motivated by SVA

Our algorithm considers starting with a small network of a few nodes per layer. During training, we consider the change in the loss function that occurs if we add some number of nodes dM to the network and train the whole network along with these additional nodes. If the change in the loss

Algorithm 1 Adaptive Network Regularization

Input: $\mathcal{X}$, $\mathcal{Y}$,H_l,$[M_{H_1}, \ldots M_{H_l}]$, γ,prev_E = ∞
for $e = 1$ **to** epochs **do**
  Sample $\boldsymbol{w}$ from $P(\boldsymbol{w}|\mathcal{X}, \mathcal{Y})$
  current_E = $\log(p(\mathcal{Y}|\mathcal{X}, \boldsymbol{w}))$
  if prev_E − current_E $> \gamma$ **then**
    $H^* = U[0, H_l]$
    $M_{H^*} = M_{H^*} + dM$
  end if
  prev_E = current_E
end for

Table 1: Results of our proposed algorithm on regression datasets with Avg RMSE and Std

Data set	N	D	Fixed Architecture	Hidden Units	Proposed Algorithm	Hidden Units Converged	Fixed Architecture with Hidden Units from Proposed Algorithm
Concrete	1030	8	5.977±0.2207	50	7.660±0.2613	25	12.346±1.8695
Kin8nm	8192	8	0.091±0.0015	50	0.0870±0.0017	43	0.091±0.0016
Naval Propulsion	11934	16	0.001±0.0001	50	0.004±0.0000	36	0.004±0.0040
Powerplant	9568	4	4.182±0.0402	50	4.114±0.0051	13	4.140±0.0061
Protein	45730	9	4.539±0.0288	100	4.511±0.0009	44	4.618±0.0559
Wine	1599	11	0.645±0.0098	50	0.609±0.0170	14	0.671±0.4421
MSD	515345	90	8.932±NA	100	8.872±NA	92	8.958±NA

function is greater than γ, the tradeoff parameter from (1), then we add dM units; otherwise, we maintain the network at its current size. When adding new nodes to the network, one should train long enough that a reliable estimate of the change in the loss function is obtained; we train for a full epoch before deciding whether to add new nodes, though it is possible that shorter training times (e.g., a minibatch) may be sufficient for determining the change in the loss. When training multiple layers, we choose uniformly at random a single layer at a time when adding new nodes.

3 Experiments

We evaluate our proposed algorithm on both single layer and multilayer feed-forward neural networks and CNNs. To show the effectiveness of our proposed algorithm, we take datasets from [4] and apply our algorithm to it, along with suitable baselines. We train single layer neural networks five times with $dM = 1$ for all datasets in three different ways: 1) using the same architecture as in [4], 2) using our dynamic algorithm to grow the number of hidden units starting with five hidden units, 3) applying standard backpropagation to the architectures learned by our algorithm. We also train multilayer networks on MNIST and CIFAR-10 with a fixed number of layers, but the hidden units are grown using our proposed algorithm. In all of our experiments on CIFAR-10 and MNIST, we perform a grid search on network size with 8 combinations of [512,1024,2048] units ([24,48,96] filters for CNNs) in each layer for a fixed set of hyperparameters, including learning rate, batch size and number of epochs. Then, we use the same set of hyperparameters and perform a grid search on dM and γ running our proposed algorithm. In both cases, we use validation accuracy as the metric to choose the best set of parameters to obtain results for test set. We see that our proposed algorithm is most likely to find a compact architecture with comparable accuracy or sometimes better than the regular grid search on model architecture. We also observe that if we train the network with the same number of units (or filters for CNNs), we end up in a lower test accuracy than our proposed algorithm. To be fair, we use the same random seed for weights matrices in all of our experiments.

Table 2: Results of our proposed algorithm on CIFAR-10 (left), MNIST using FCNs (right) and MNIST using CNNs (bottom)

Method	Accuracy	$M_{h_0}, M_{h_1}, M_{h_2}$	No.of Parameters
Grid Search	57.71	2048-1024-2048	10M
Proposed Algorithm	58.43	1223-1498-1151	7M
Fixed Architecture	57.42	1223-1498-1151	7M

Method	Accuracy	$M_{h_0}, M_{h_1}, M_{h_2}$	No.of Parameters
Grid Search	98.71	2048-2048-2048	10M
Proposed Algorithm	98.76	633-659-677	1M
Fixed Architecture	98.57	633-659-677	1M

Method	Accuracy	$M_{h_0}, M_{h_1}, M_{h_2}$	No.of Parameters
Grid Search	99.58	48-96-24	66658
Proposed Algorithm	99.58	34-48-25	29911
Fixed Architecture	98.48	34-48-25	29911

References

[1] Ryan Adams, Hanna Wallach, and Zoubin Ghahramani. Learning the structure of deep sparse graphical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 1–8, 2010.

[2] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[3] Marc-Alexandre Côté and Hugo Larochelle. An infinite restricted boltzmann machine. *Neural computation*, 28(7):1265–1288, 2016.

[4] José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

[5] Eric Nalisnick and Padhraic Smyth. Stick-breaking variational autoencoders. In *International Conference on Learning Representations (ICLR)*, 2017.

[6] George Philipp and Jaime G Carbonell. Nonparametric neural networks. *arXiv preprint arXiv:1712.05440*, 2017.