Model Scheduling

Aug 13, 2018 · 15 min read

Models can be built incrementally by modifying their hyperparameters during training. This is most common in transfer learning settings, in which we seek to adapt the knowledge in an existing model for a new domain or task. The more general problem of continuous learning is also an obvious application. Even with a predefined data set, however, incrementally constraining the topology of the network can offer benefits as regularization.

Dynamic Hyperparameters

The easiest incrementally modified models to train may be those in which hyperparameters are updated at each epoch. In this case, we do not mean those hyperparameters associated with network topology, such as the number or dimension of layers. There are many opportunities to adjust the topology during training, but the model often requires heavy retraining in order to impose reasonable structure again, as demonstrated clearly in the case of memory networks[1]. If we instead focus on the weights associated with regularizers and gates, we can gradually learn structure without frequent retraining to accommodate radically altered topologies.

Curriculum Dropout

Hinton et al.[2] describes dropout as reducing overfitting by preventing co-adaptation of feature detectors which happened to perfectly fit the data. In this interpretation, co-adaptive clusters of

neurons are concurrently activated. Randomly suppressing these neurons forces them to develop independence.

In standard dropout, these co-adaptive neurons are treated as equally problematic at all stages of training. However, Morerio et. al.[3] posit that early in training, co-adaptation may represent the beginnings of an optimal self organization of the network. In this view, these structures mainly pose the threat of overfitting later in training. The authors therefore introduce a hyperparameter schedule for the dropout ratio, increasing the rate of dropout as training continues. To the best of my knowledge, this is the only proposal of adaptive regularization published.

Mollifying Networks

Mollifying networks[4] are, to my knowledge, the only existing attempt to combine techniques focused on incrementally manipulating the distribution of data with techniques focused on incrementally manipulating the representational capacity of the model. Mollifying networks incrementally lower the temperature of the data through simulated annealing while simultaneously modifying various hyperparameters to permit longer-range dependencies. In the case of an LSTM, they set the output gate to 1, input gate to $\frac{1}{t}$, and forget gate to $1 - \frac{1}{t}$, where t is the annealing time step. Using this system, the LSTM initially behaves as a bag-of-words model, gradually adding the capacity to handle more context at each time step.

Mollifying networks use a different data schedule for each layer, annealing the noise in lower layers faster than in higher layers because lower-level representations are assumed to learn faster.

Adaptive Architectures

The hyperparameters most difficult to modify during training may be those which dictate the topology of the model architecture itself. Nonetheless, the deep learning literature contains a long history of

techniques which adapt the model architecture during training, often in response to the parameters being learned. Methods like these can help search optimally by smoothing functions at the beginning of training, speed up learning by starting with a simpler model, or compress a model to fit easily on a phone or embedded device. Most of these methods could be classified as either growing a model by adding parameters mid-training or shrinking a model by pruning edges or nodes.

Architecture Growth

Some recent transfer learning strategies have relied on growing architectures by creating entire new modules focused on the new task with connections to the existing network[56]. If the goal is to instead augment an existing network by adding a small number of parameters, the problem bears a resemblance to traditional nonparametric learning, because we need not explicitly limit the model space to begin with.

Classical techniques in neural networks such as Cascade Correlation Networks[7] and Dynamic Node Creation[8] added new nodes at random one by one and trained them individually. On modern large-scale architectures and problems, this is intractable. Furthermore, the main advantage of such methods is that they approach a minimal model, which is an aim that modern deep learning practitioners no longer consider valuable thanks to leaps in computing power in the decades since. Modern techniques for incrementally growing networks must make 2 decisions: 1) When (and where) do we add new parameters? 2) How do we train new parameters?

Warde-Farley et. al.[9] add parameters in bulk after training an entire network. The augmentation takes the form of specialized auxiliary layers added to the existing network in parallel. These layers are trained on class boundaries that the original generalist model struggles with. The class boundaries that require special attention are selected by performing spectral clustering on the confusion matrix of a holdout data set, partitioning the classes into challenging subproblems.

The auxiliary layers are initialized randomly in parallel with the original generalist system, and then are each trained only on examples from their assigned partition of the classes. The original generalist network is held fixed, other than fine-tuning the final classification layer. The resulting network is a mixture of experts, which was shown to improve results on an image classification problem.

Neurogenesis Deep Learning (NDL)[10], meanwhile, makes autoencoders capable of lifelong learning. This strategy updates the topology of an autoencoder by adding neurons when the model encounters outliers that it performs especially poorly on. These new parameters are trained exclusively on those outliers, allowing the existing decoder parameters to update with much smaller step sizes. Existing encoder parameters update only if they are connected directly to the new neuron.

After introducing and training these new neurons, NDL stabilizes the existing structure of the network using a method the authors call “intrinsic replay”. They reconstruct approximations of previously seen samples and train on these reconstructions.

Another system that permits lifelong learning is the infinite Restricted Boltzmann Machine (RBM) [11]. This extension of the classic RBM parameterizes hidden units by unique indices, expressing an ordering. These indices are used to enforce an order on the growth of the network by favoring older nodes until they have converged, permitting the system to grow arbitrarily large. An intriguing approach, but it is not obvious how to apply similar modifications to networks other than the idiosyncratic generative architecture of the RBM.

None of these augmentation techniques support recurrent architectures. In modern natural language processing settings, this is a fatal limitation. However, it is possible that some of these techniques may be adapted for RNNs, especially since training specialized subsystems has been recently tackled in these environments [12].

Architecture Pruning

Much recent research has focused on the possibility of pruning edges or entire neurons from trained networks. This approach is promising not only for the purpose of compression, but potentially as a way of increasing the generalizability of a network.

Pruning Edges

Procedures that prune edges rather than entire neurons may not reduce the dimensional type of the network. However, they will make the network sparser, leading to possible memory savings. A sparser network also occupies a smaller parameter space, and may therefore still more general.

Han et. al.[13] takes the basic approach of setting weights to 0 if they fall below a certain threshold. This approach is highly effective for compression, because the number of weights to be pruned can be easily modified through the threshold.

LeCun et. al.[14] and Hassibi et. al.[15] both select weights to prune based on Taylor series approximation of the change in error resulting from trimming. While these methods were successful for older shallow networks, performing these operations on an entire network requires a Hessian matrix to be computed over all parameters, which is generally intractable for deep modern architectures. Dong et. al.[16] presents a more efficient alternative by performing optimal brain surgery over individual layers instead.

Pruning Nodes

Pruning entire nodes has the advantage of reducing the entire dimensionality of the network. It also may be faster than choosing individual edges to prune, because having more nodes than constituent edges reduces the number of candidates to consider for pruning.

He et al.[17] selects which neuron w_i^ℓ to prune from layer ℓ with width d_ℓ by calculating the importance of each node. They test several importance metrics, finding that the highest performance results from using the 'onorm', or average l_1 norm of the activation pattern of the node:

$$\text{onorm}(w_i^\ell) = \frac{1}{d_{\ell+1}} \sum_{j=1}^{d_{\ell+1}} |w_{ij}^{\ell+1}|$$

Net-trim [18] likewise relies on the l_1 norm to induce sparsity.

Wolfe et al.[19] compares the results of importance based pruning to a brute force method that will greedily select a node to be sacrificed based on its impact on performance. In the brute force method, they rerun the network on the test data without each node and sort the nodes according to the error of the resulting network. Their importance metrics are based on neuron-level versions of the Taylor series approximations of that impact[15].

In the first algorithm tested, they rank all nodes according to their importance and then remove each node in succession. In the second algorithm, they re-rank the nodes after each removal, in order to account for the effects of subnetworks that generate and then cancel. In the second case, they find that it is possible to prune up to 60% of nodes in a network trained on mnist without significant loss in performance. This supports an early observation[20] that the majority of parameters in a network are unnecessary, and their effect is limited to generating and then canceling their own noise. The strength of this effect supports the idea that backpropagation implicitly trains a minimal network for the task given.

Srinivas and Babu[21] prune with the goal of reducing the redundancy of the network, so they select nodes to remove based on the similarity of their weights to other neurons in the same layer. Diversity networks[22], meanwhile, choose based on the diversity of their activation patterns. In order to sample a diverse selection of nodes, they use a Determinantal Point Process. This technique minimizes the

dependency between nodes sampled. They followed this pruning process by fusing the nodes pruned back into the network.

An intriguing difference emerges between the observations in these papers. While Mariet and Sra[22] find that in deeper layers they sample more nodes from the DPP, Philipp and Carbonell [19] prune more nodes by brute force in the deeper layer of a 2-layer network. In other words, diversity networks retain more nodes at deeper layers while greedy brute force approaches remove more from the same layers. These results point to fundamental differences between the respective outcomes of these algorithms and warrant further investigation.

Merging Nodes

Mariet and Sra[22] found that performance increased after their DPP-based pruning if they then merged the pruned nodes back into the network. They achieved this by re-weighting the remaining nodes in the pruned layer to minimize the difference in activation outputs before and after pruning:

$$\min_{\tilde{w}_{ij} \in \mathbb{R}} \left| \sum_{i=1}^{k} \tilde{w}_{ij} v_i - \sum_{i=1}^{d_\ell} w_{ij} v_i \right|_2$$

Because the DPP is focused on selecting an independent set of neurons, it seems likely that pruning will select at least 1 node within any given noise cancellation system to keep, since those cancellation subnetworks are by necessity highly dependent. The merging step in that case would merge the noise canceling components back into the noise generating nodes or vice versa. This would make merging a particular necessity in diversity networks, but it may still present a tractable alternative to retraining after a different pruning algorithm.

Nonparametric Neural Networks

The pruning and growing strategies are combined in only one work, to my knowledge. Nonparametric Neural Networks (NNNs)[23] combine adding neurons with imposing a sparsity-inducing penalty over

neurons. For a feedforward network with N^L layers, authors introduce 2 such regularizers, a “fan-in” and a “fan-out” variant:

$$\Omega_{\text{in}} = \sum_{\ell=1}^{N^L} \sum_{j=1}^{d_\ell} \left(\sum_{i=1}^{d_\ell} |w_{ij}^{\ell+1}|^p \right)^{\frac{1}{p}}$$

$$\Omega_{\text{out}} = \sum_{\ell=1}^{N^L} \sum_{i=1}^{d_\ell} \left(\sum_{j=1}^{d_\ell+1} |w_{ij}^{\ell}|^p \right)^{\frac{1}{p}}$$

In other words, the fan-in variant penalizes the p-norm of the inputs to each neuron, while the fan-out of variant penalizes the p-norm of the outputs from each neuron. In the case of feedforward networks, either of these regularizers can be added to the loss function with any positive weight λ and $0 < p < \infty$ to guarantee that the objective will converge at some finite number of neurons.

NNNs offer a combination of beneficial strategies for adapting the network. In particular with $p = 1$ or 2, induces sparsity by applying pressure to form *zero-valued neurons*, or neurons which have either a fan-in or fan-out value of 0. At intervals we can remove these zero-valued neurons which result. At the same time, we can introduce new zero-valued neurons at different locations in the network, and the regularizer guarantees the objective will converge, so we can stop adding neurons at any point that performance begins to decline.

However, there are clear issues with this approach. The first obvious limitation is that this regularizer cannot be applied in any network with recurrences. This constraint reduces the strategy’s usefulness in many natural language domains where state-of-the-art performance requires a RNN.

Another disadvantage to this method is the choice to insert zero-valued neurons by initializing either the input or output weight vector as 0 and randomly initializing the other associated vector. We therefore retrain the entire network with each interval, rather than intelligently initializing and training

the new node to accelerate convergence. While this approach may converge to an optimal number of nodes, it does nothing to accelerate training or help new nodes specialize.

Finally, this approach adds and removes entire neurons to create a final dense network. It therefore forfeits the potential regularization advantages of the sparser networks which result from instead pruning weights.

Teacher/Student Approaches

It is also possible to produce a larger or smaller model based on an existing network by fresh training. When investigating any adaptive architecture, it is crucial to compare with a baseline which uses the previous state of the network as a teacher to a student network which has the new architecture.

The approach of teacher/student learning, in which the teacher network's outputs layer are used in lieu of or in addition to true labels, was introduced in the particular case of distillation learning [24]. Distillation is a technique for compressing a large ensemble or generally expensive classifier with high performance. A smaller network is trained using an objective that combines a loss function applied to true labels with cross-entropy against the logit layer of the large teacher network. In addition to compression, teacher/student learning is effective for domain adaptation technique [25], suggesting it may be useful for adapting to a new time step in a data schedule.

---

1. Sachan, Mrinmaya, and Eric Xing. "Easy questions first? a case study on curriculum learning for question answering." *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers). Vol. 1. 2016. ^
2. Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR* abs/1207.0580. ^

3. Pietro Morerio, Jacopo Cavazza, Riccardo Volpi, Rene Vidal, and Vittorio Murino. 2017. Curriculum Dropout. *arXiv:1703.06229 [^cs, stat]:*. Retrieved March 22, 2017 from http://arxiv.org/abs/1703.06229 ^
4. Caglar Gulcehre, Marcin Moczulski, Francesco Visin, and Yoshua Bengio. 2016. Mollifying Networks. *arXiv:1608.04980 [^cs]:*. Retrieved October 7, 2016 from http://arxiv.org/abs/1608.04980 ^
5. Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. 2017. A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks. In *arXiv:1611.01587 [^cs]:*. Retrieved November 11, 2016 from http://arxiv.org/abs/1611.01587 ^
6. Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive Neural Networks. *arXiv:1606.04671 [^cs]:*. Retrieved September 14, 2016 from http://arxiv.org/abs/1606.04671 ^
7. Scott E. Fahlman and Christian Lebiere. 1989. The cascade-correlation learning architecture. Retrieved November 30, 2016 from http://repository.cmu.edu/compsci/1938/ ^
8. Ash. 1989. Dynamic node creation in backpropagation networks. In *International 1989 Joint Conference on Neural Networks*, 623 vol.2. https://doi.org/10.1109/IJCNN.1989.118509 ^
9. David Warde-Farley, Andrew Rabinovich, and Dragomir Anguelov. 2014. Self-informed neural network structure learning. *arXiv preprint arXiv:1412.6563*. Retrieved June 9, 2016 from http://arxiv.org/abs/1412.6563 ^
10. Timothy J. Draelos, Nadine E. Miner, Christopher C. Lamb, Craig M. Vineyard, Kristofor D. Carlson, Conrad D. James, and James B. Aimone. 2016. Neurogenesis Deep Learning. *arXiv:1612.03770 [^cs, stat]:*. Retrieved February 27, 2017 from http://arxiv.org/abs/1612.03770 ^
11. Marc-Alexandre Cote and Hugo Larochelle. 2015. An Infinite Restricted Boltzmann Machine. *arXiv:1502.02476 [^cs]:*. Retrieved February 8, 2018 from http://arxiv.org/abs/1502.02476 ^

12. Shlomo E. Chazan, Jacob Goldberger, and Sharon Gannot. 2017. Deep recurrent mixture of experts for speech enhancement. *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*: 359–363. ^
13. Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. *arXiv:1506.02626 [^cs]:*. Retrieved May 26, 2016 from http://arxiv.org/abs/1506.02626 ^
14. Yann LeCun, John S. Denker, and Sara A. Solla. 1990. Optimal brain damage. In *Advances in neural information processing systems*, 598–605. ^
15. Babak Hassibi, David G. Stork, and Gregory J. Wolff. 1993. Optimal brain surgeon and general network pruning. In *Neural Networks, 1993., IEEE International Conference on*, 293–299. ^
16. Xin Dong, Shangyu Chen, and Sinno Jialin Pan. 2017. Learning to Prune Deep Neural Networks via Layer-wise Optimal Brain Surgeon. *arXiv:1705.07565 [^cs]:*. Retrieved from http://arxiv.org/abs/1705.07565 ^
17. Tianxing He, Yuchen Fan, Yanmin Qian, Tian Tan, and Kai Yu. 2014. Reshaping deep neural network for fast decoding by node-pruning. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*: 245–249. ^
18. Aghasi, Alireza, et al. “Net-trim: Convex pruning of deep neural networks with performance guarantee.” *Advances in Neural Information Processing Systems*. 2017. http://arxiv.org/abs/1611.05162 ^
19. Nikolas Wolfe, Aditya Sharma, Lukas Drude, and Bhiksha Raj. 2017. “The Incredible Shrinking Neural Network: New Perspectives on Learning Representations Through The Lens of Pruning.” ^
20. Michael C. Mozer and Paul Smolensky. 1989. Using Relevance to Reduce Network Size Automatically. ^

21. Suraj Srinivas and R. Venkatesh Babu. 2015. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*. Retrieved October 5, 2016 from http://arxiv.org/abs/1507.06149 ^
22. Zelda Mariet and Suvrit Sra. 2015. Diversity Networks: Neural Network Compression Using Determinantal Point Processes. *arXiv:1511.05077 [^cs]:*. Retrieved February 9, 2018 from http://arxiv.org/abs/1511.05077 ^
23. George Philipp and Jaime G. Carbonell. 2017. Nonparametric Neural Networks. In *arXiv:1712.05440 [^cs]:*. Retrieved February 18, 2018 from http://arxiv.org/abs/1712.05440 ^
24. Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the Knowledge in a Neural Network. *arXiv:1503.02531 [^cs, stat]:*. Retrieved September 22, 2016 from http://arxiv.org/abs/1503.02531 ^
25. Jinyu Li, Michael L. Seltzer, Xi Wang, Rui Zhao, and Yifan Gong. 2017. Large-Scale Domain Adaptation via Teacher-Student Learning. *arXiv:1708.05466 [^cs]:*. Retrieved August 26, 2017 from http://arxiv.org/abs/1708.05466 ^