

Supplementary material

Learning Neural Network Architectures using Backpropagation

1 Properties of the method

Here we identify a few properties of our architecture selection method.

1. **Non-redundancy of architecture:** The learnt final architecture must not have any redundant neurons. Removing neurons should necessarily degrade performance.
2. **Local-optimality of weights:** The performance of the learnt final architecture must at least be equal to a trained neural network initialized with this final architecture.
3. **Mirroring data-complexity:** A ‘harder’ dataset should result in a larger model than an ‘easier’ dataset.

We intuitively observe that all these properties would automatically hold if a ‘master’ property which requires both the architecture and the weights be globally optimal holds. Given that the optimization objective of neural networks is highly non-convex, global optimality cannot be guaranteed. As a result, we restrict ourselves to studying the three properties listed.

In the text that follows, we provide statements that hold for our method. These are obtained by analysing widths of each layer of a neural network assuming that depth is never collapsed. In other words, these hold for neural networks with a single hidden layer. Proofs are provided in the Appendix.

Non-redundancy of architecture

This is an important property that forms the main motivation for doing architecture-learning. Such a procedure can replace the node-pruning techniques that are used to compress neural networks.

Proposition 1. *At convergence, the loss (ℓ) of the proposed method over the train set satisfies $\frac{\partial \ell}{\partial \Phi} < 0$*

This statement implies that change in architecture is inversely proportional to change in loss. In other words, if the architecture grows smaller, the loss must increase. While there isn’t a strict relationship between loss and accuracy, a high loss generally indicates worse accuracy.

Local Optimality of weights

The proposed method learns both architecture and weights. What would happen if we initialized a neural network with this learnt architecture, and proceeded to learn only the weights? This property ensures that in both cases we fall into a local minimum with architecture Φ .

Proposition 2. *Let ℓ_1 be the loss over the train set at convergence obtained by training a neural network on data \mathcal{D} with a fixed architecture Φ . Let ℓ_2 be the loss at convergence when the neural network is trained with the proposed method on data \mathcal{D} such that it results in the same final architecture Φ . Then, $\frac{\partial \ell_1}{\partial \theta} < \epsilon$ and $\frac{\partial \ell_2}{\partial \theta} < \epsilon$ for any $\epsilon \rightarrow 0$.*

Mirroring data-complexity

Characterizing data-complexity has traditionally been hard. Here, we consider the following approach.

Proposition 3. *Let \mathcal{D}_1 and \mathcal{D}_2 be two datasets which produce train losses ℓ_1 and ℓ_2 upon training with a fixed architecture Φ such that $\ell_1 > \ell_2$. When trained with the proposed method, the final architectures $\hat{\Phi}_1$ and $\hat{\Phi}_2$ (corresponding to \mathcal{D}_1 and \mathcal{D}_2) satisfy the relation $\|\hat{\Phi}_1\| > \|\hat{\Phi}_2\|$ at convergence.*

Here, \mathcal{D}_1 is the ‘harder’ dataset because it produces a higher loss on the same neural network architecture. As a result, the ‘harder’ dataset always produces a larger final architecture. We do not provide a proof for this statement. Instead, we experimentally verify this in Section 4.2.

2 Proofs of Propositions

Let $E = \ell + \lambda_b \mathcal{R}_b + \lambda_m \mathcal{R}_m$ be total objective function, where \mathcal{R}_b is the binarizing regularizer, $\mathcal{R}_m = \|\phi\|$ is the model complexity term. At convergence, we assume that $\mathcal{R}_b = 0$ as the corresponding weights are all binary or close to binary. Let the maximum step size (due to gradient clipping) for \mathbf{w} and \mathbf{d} be s .

Proof of proposition 1. At convergence, we assume

$$\frac{\partial E}{\partial \phi} < \epsilon, \text{ for some } \epsilon \rightarrow 0_+.$$

$$\frac{\partial \ell}{\partial \phi} < -\lambda_m \frac{\partial \|\phi\|}{\partial \phi} + \epsilon \implies \frac{\partial \ell}{\partial \phi} < -\lambda_m + \epsilon \implies \frac{\partial \ell}{\partial \phi} < 0$$

for some ϵ sufficiently small. □

Proof of proposition 2. Let $\mathcal{R}_b = 0$ at t_1^{th} iteration with architecture Φ_1 . Let Φ_2 be the architecture at iteration $t_2 > t_1$ such that at iterations $t_1 < t < t_2$, architecture is Φ_1 .

$\implies \exists$ an iteration $t_1 < t < t_2$ such that $\mathcal{R}_b > s(1-s) = s_1$, s being the maximum step size.

Let $q = \frac{\partial \ell_2}{\partial \theta_2}$. Let λ_b be parameterized by k as follows.

$$\lambda_b s_1 = \mathbb{E}_{\mathcal{D}}(q) + k\sigma \text{ where } \sigma = \mathbb{E}_{\mathcal{D}}(q - \mathbb{E}_{\mathcal{D}}(q))^2$$

$$\text{If } k \rightarrow \infty \text{ then } \mathbb{P}(q > \lambda_b s_1) \rightarrow 0$$

Hence, for large enough λ_b , $\Phi_2 = \Phi_1$. After $T \gg t$ iterations, we have

$$\frac{\partial \ell_1}{\partial \theta_1} < \epsilon \text{ and } \frac{\partial \ell_2}{\partial \theta_2} < \epsilon \tag{1}$$

for some $\epsilon \rightarrow 0_+$. However, if $\theta_1 \in \mathbb{R}^{d_1}$, then $\theta_2 \in \mathbb{R}^{d_2}$, such that $d_1 < d_2$.

Without loss of generality, let us assume that neurons corresponding to first d_1 weights are selected for, while the rest are inactive. As a result, $\frac{\partial \ell_2}{\partial \theta_2(d)} = 0$, for $d \in [d_1, d_2]$. Hence, the following holds $\frac{\partial \ell_2}{\partial \theta_1} < \epsilon$. This, along with equation 1, proves the assertion. □

3 Hyper-parameter selection

For effective usage of our method, we need a good set of λ s. Here, we describe how to do so practically.

First, we set λ_3 to a low value based on the initial widths and loss values. Recall that this value multiplies with the number of neurons in the cost function. That is, if a network has a layer with n neurons, we get $\lambda_3 \times n$. Hence, if n multiplies by 10, λ_3 divides by 10. We used 10^{-5} for MNIST-network and 10^{-6} for AlexNet. For a given initial architecture, a large λ_3 places more emphasis on getting small models than reducing loss.

Second, we set λ_1 to be about ~ 2 times λ_3 . Using a positive λ_3 shifts the curve to the right. By letting $\lambda_3 = \lambda_1$, the curve shifts to the extreme right with the peak at $x = 1$. Hence if $\lambda_3 = k \times \lambda_1$, we set $0 < k < 1$.

We simply set λ_2 and λ_4 to $1/10^{th}$ of λ_1 and λ_3 respectively.