Targeted Dropout

Anonymous Author(s)
Affiliation
Address
e-mail

Abstract
Neural networks are extremely flexible models due to their large number of parameters, which is beneficial for learning, but also highly redundant. This makes it possible to compress neural networks without having a drastic effect on performance. We introduce targeted dropout, a strategy for post hoc pruning of neural network weights and units that builds the pruning mechanism directly into learning. At each weight update, targeted dropout selects a candidate set for pruning using a simple selection criterion, and then stochastically prunes the network via dropout applied to this set. The resulting network learns to be explicitly robust to pruning, comparing favourably to more complicated regularization schemes while at the same time being extremely simple to implement, and easy to tune.

1 Introduction
There has been a great deal of work on developing strategies to sparsify neural networks [10, 7, 5, 2, 12, 3, 11, 9, 15]. Sparsification involves removing weights (corresponding to setting them to 0) or entire units from the network, while maintaining predictive performance. Sparsity can be encouraged during learning by the use of sparsity-inducing regularizers, like $L^1$ or $L^0$ penalties. It can also be imposed by post hoc pruning, where a full-sized network is trained, and then sparsified according to some pruning strategy. Ideally, given some measurement of task performance, we would prune the weights or units that provide the least amount of benefit to the task. Finding the optimal set is in general a difficult combinatorial problem, and even a greedy strategy would require an unrealistic number of task evaluations, as there are often millions of parameters in a modern neural network architecture. Common pruning strategies therefore focus on fast approximations, such as removing weights with the smallest magnitude [6], or ranking the weights by the sensitivity of the task performance with respect to the weights and removing the least-sensitive ones [10].

Our approach is based on the observation that dropout regularization [8, 4] itself enforces sparsity during training, by sparsifying the network with each forward pass. This encourages the network to learn a representation that is robust to a particular form of post hoc sparsification – in this case, where a random set of units are removed. Our hypothesis is that if we plan to do explicit post hoc sparsification, then we can do better by specifically applying dropout to the set of units that we a priori believe are the least useful. We call this approach targeted dropout. The idea is to rank weights or units according to some fast, approximate measure of importance (like magnitude), and then apply dropout primarily to those elements with low importance. Similar to the observation with regular dropout, we show that this encourages the network to learn a representation where the importance of weights or units more closely aligns with our approximation. In other words, the network learns to be robust to our choice of post hoc pruning strategy. The advantage of targeted dropout compared to other approaches is that it leads to a converged network that is extremely robust to post hoc pruning. It is simultaneously easy to implement, consisting of a two-line change using neural network frameworks such as Tensorflow [1] or PyTorch [13]. Furthermore, it is explicit: the desired level of sparsity is provided by the user, and enforced throughout training.

2 Targeted Dropout

2.1 Dropout

Our work uses the two most popular Bernoulli dropout techniques, Hinton et al.'s unit dropout \[8, 14\] and Wan et al.'s weight dropout (dropconnect) \[17\]. For a fully-connected layer with input tensor \(X\), weight matrix \(W\), output tensor \(Y\), and mask \(M_{i,o} \sim \text{Bernoulli}(\alpha)\) we define both techniques below:

**Unit dropout** \[8, 14\]:

\[
Y = (X \odot M)W
\]

Unit dropout randomly drops units (sometimes referred to as neurons) at each training step to reduce dependence between units and prevent overfitting.

**Weight dropout** \[17\]:

\[
Y = X(W \odot M)
\]

Weight dropout randomly drops individual weights in the weight matrices at each training step. Intuitively, this is dropping connections between layers, forcing the network to adapt to a different connectivity at each training step.

2.2 Magnitude-based pruning

A popular class of pruning strategies are those characterized as magnitude-based pruning strategies. These strategies treat the top-\(k\) largest magnitude weights as important. We use \(\text{argmax}-k\) to return the top-\(k\) elements (units or weights) out of all elements being considered.

**Unit pruning** \[6\]: considers the units (column-vectors) of weight matrices under the \(L^2\)-norm.

\[
W(\theta) = \left\{ \text{argmax}-k \left\| W_o \right\|_2 \mid W \in \theta \right\}
\]  

(1)

**Weight pruning** \[10\]: considers each entry of the weight matrix separately under the \(L^1\)-norm. Note that the top-\(k\) is with respect to the other weights in the same filter.

\[
W(\theta) = \left\{ \text{argmax}-k \left| W_{io} \right| \mid 1 \leq o \leq N_{col}(W), W \in \theta \right\}
\]  

(2)

While weight pruning tends to preserve more of the task performance under coarser prunings \[5, 16, 4\], unit pruning allows for considerably greater computational savings \[13, 11\]. In particular, weight pruned networks can be implemented using sparse linear algebra operations, which offer speedups only under sufficiently sparse conditions; while unit pruned networks execute standard linear algebra ops on lower dimensional tensors, which tends to be a much faster option for practical pruning rates.

2.3 Methods

Consider a neural network parameterized by \(\theta\), and our pruning strategy (defined above in Equations (1) and (2)) \(W\). We hope to find optimal parameters \(\theta^*\) such that our loss \(\mathcal{E}(W(\theta^*))\) is low and at the same time \(|W(\theta^*)| \leq k\), i.e. we wish to keep only the \(k\) weights of highest magnitude in the network. A deterministic implementation would select the bottom \(|\theta| - k\) elements and drop them out. However, we would like for low-valued elements to be able to increase their value if they become important during training. Therefore, we introduce stochasticity into the process using a targeting proportion \(\gamma\) and a drop probability \(\alpha\). The targeting proportion means that we select the bottom \(\gamma|\theta|\) weights as candidates for dropout, and of those we drop the elements independently with drop rate \(\alpha\). This implies that the expected number of units to keep during each round of targeted dropout is \((1 - \gamma \cdot \alpha)|\theta|\). As we will see below, the result is a reduction in the important subnetwork dependency on the unimportant subnetwork, thereby reducing the performance degradation of pruning at the conclusion of training.

Our weight pruning experiments (shown in Tables 1 & 2) demonstrate that the baseline regularization schemes are comparatively weak to their targeted counterparts. We find that targeted dropout applied to the weights results in the network outperforming unregularized performance with only half the total number of parameters. We are able to reach extremely high rates of pruning in Table 2 by
Table 1: ResNet-32 model accuracies on CIFAR-10 at differing pruning percentages and under different regularization schemes. The top table depicts results using the weight pruning strategy, while the bottom table depicts the results of unit pruning (see Sec. 2.2).

<table>
<thead>
<tr>
<th>Prune Percentage</th>
<th>Weight Dropout/Pruning</th>
<th>Unit Dropout/Pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>none</td>
<td>dropout</td>
</tr>
<tr>
<td>0%</td>
<td>0.74</td>
<td>0.74</td>
</tr>
<tr>
<td>10%</td>
<td>0.74</td>
<td>0.74</td>
</tr>
<tr>
<td>20%</td>
<td>0.74</td>
<td>0.74</td>
</tr>
<tr>
<td>30%</td>
<td>0.74</td>
<td>0.74</td>
</tr>
<tr>
<td>40%</td>
<td>0.74</td>
<td>0.74</td>
</tr>
<tr>
<td>50%</td>
<td>0.74</td>
<td>0.74</td>
</tr>
<tr>
<td>60%</td>
<td>0.74</td>
<td>0.74</td>
</tr>
<tr>
<td>70%</td>
<td>0.74</td>
<td>0.74</td>
</tr>
<tr>
<td>80%</td>
<td>0.74</td>
<td>0.74</td>
</tr>
<tr>
<td>90%</td>
<td>0.74</td>
<td>0.74</td>
</tr>
</tbody>
</table>

Table 2: Comparing Smallify to targeted dropout and ramping targeted dropout. Experiments on CIFAR10 using ResNet32. Left: the best of the three targeted dropout runs compared against the best out of six smallify runs; Middle: inspecting higher pruning rates of the best smallify run compared to ramping targeted dropout; Right: inspecting even higher pruning rates of ramping targeted dropouts.

<table>
<thead>
<tr>
<th>Prune Percentage</th>
<th>Weight Dropout/Pruning</th>
<th>Unit Dropout/Pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>none</td>
<td>smallify</td>
</tr>
<tr>
<td>0%</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>10%</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>20%</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>30%</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>40%</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>50%</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>60%</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>70%</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>80%</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>90%</td>
<td>0.37</td>
<td>0.37</td>
</tr>
</tbody>
</table>

3 Conclusion

We propose targeted dropout as a simple and extremely effective regularization tool for incorporating post hoc pruning strategies into the training procedure of neural networks without drastic impact to underlying task performance of a particular architecture. Among the primary benefits of targeted dropout are the simplicity of implementation and intuitive, flexible hyperparameters.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow,


