Seventh Italian Workshop on Machine Learning and Data Mining (MLDM)

From deep to minimalistic neural networks

Authors: S. Scardapane and A. Uncini
Deeplearning networks in 2012

«We trained one of the largest convolutional neural networks to date [...] It contains eight learned layers - five convolutional and three fully-connected. [...] Our network takes between five and six days to train. »

Deep networks in 2018

Xiao, L. et al., 2018. *Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks*. *Proc. ICML 2018*. 
Standard practice today is to **wildly over-parameterize** models, with the only limits being in hardware / data.

Motivated by apparent **lack of overfitting**: « A main puzzle of deep networks revolves around the absence of overfitting despite large overparametrization and despite the large capacity demonstrated by zero training error on randomly labeled data. » (Poggio et al., 2017)²

A few fascinating results
"We demonstrate that there is significant redundancy in the parameterization of several deep learning models. Given only a few weight values for each feature it is possible to accurately predict the remaining values. [...] In the best case we are able to predict more than 95% of the weights of a network without any drop in accuracy." (Denil et al., 2013)
2 - Compressing the parameters

Figure 1: The three stage compression pipeline: pruning, quantization and Huffman coding. Pruning reduces the number of weights by $10\times$, while quantization further improves the compression rate: between $27\times$ and $31\times$. Huffman coding gives more compression: between $35\times$ and $49\times$. The compression rate already included the meta-data for sparse representation. The compression scheme doesn’t incur any accuracy loss.

### Results on standard CNNs

**Table 2: Compression statistics for LeNet-300-100. P: pruning, Q: quantization, H: Huffman coding.**

<table>
<thead>
<tr>
<th>Layer</th>
<th>#Weights</th>
<th>Weights % (P)</th>
<th>Weight bits (P+Q)</th>
<th>Weight bits (P+Q+H)</th>
<th>Index bits (P+Q)</th>
<th>Index bits (P+Q+H)</th>
<th>Compress rate (P+Q)</th>
<th>Compress rate (P+Q+H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip1</td>
<td>235K</td>
<td>8%</td>
<td>6</td>
<td>4.4</td>
<td>5</td>
<td>3.7</td>
<td>3.1%</td>
<td>2.32%</td>
</tr>
<tr>
<td>ip2</td>
<td>30K</td>
<td>9%</td>
<td>6</td>
<td>4.4</td>
<td>5</td>
<td>4.3</td>
<td>3.8%</td>
<td>3.04%</td>
</tr>
<tr>
<td>ip3</td>
<td>1K</td>
<td>26%</td>
<td>6</td>
<td>4.3</td>
<td>5</td>
<td>3.2</td>
<td>15.7%</td>
<td>12.70%</td>
</tr>
<tr>
<td>Total</td>
<td>266K</td>
<td>8% (12×)</td>
<td>6</td>
<td>5.1</td>
<td>5</td>
<td>3.7</td>
<td>3.1% (32×)</td>
<td>2.49% (40×)</td>
</tr>
</tbody>
</table>

**Table 3: Compression statistics for LeNet-5. P: pruning, Q: quantization, H: Huffman coding.**

<table>
<thead>
<tr>
<th>Layer</th>
<th>#Weights</th>
<th>Weights % (P)</th>
<th>Weight bits (P+Q)</th>
<th>Weight bits (P+Q+H)</th>
<th>Index bits (P+Q)</th>
<th>Index bits (P+Q+H)</th>
<th>Compress rate (P+Q)</th>
<th>Compress rate (P+Q+H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>0.5K</td>
<td>66%</td>
<td>8</td>
<td>7.2</td>
<td>5</td>
<td>1.5</td>
<td>78.5%</td>
<td>67.45%</td>
</tr>
<tr>
<td>conv2</td>
<td>25K</td>
<td>12%</td>
<td>8</td>
<td>7.2</td>
<td>5</td>
<td>3.9</td>
<td>6.0%</td>
<td>5.28%</td>
</tr>
<tr>
<td>ip1</td>
<td>400K</td>
<td>8%</td>
<td>5</td>
<td>4.5</td>
<td>5</td>
<td>4.5</td>
<td>2.7%</td>
<td>2.45%</td>
</tr>
<tr>
<td>ip2</td>
<td>5K</td>
<td>19%</td>
<td>5</td>
<td>5.2</td>
<td>5</td>
<td>3.7</td>
<td>6.9%</td>
<td>6.13%</td>
</tr>
<tr>
<td>Total</td>
<td>431K</td>
<td>8% (12×)</td>
<td>5.3</td>
<td>4.1</td>
<td>5</td>
<td>4.4</td>
<td>3.05% (33×)</td>
<td>2.55% (39×)</td>
</tr>
</tbody>
</table>
«If a network can be so compressed, then the function it learned can be represented by a far smaller network than that used during training. Why, then, do we train large networks when we could improve efficiency by training smaller networks instead? » (Frankle and Carbin, 2018)

According to the **lottery ticket hypothesis**:

1. Good performance depends on the *lucky* initialization of one/more subnetworks.
2. Large networks have exponentially more of these subnetworks.
3. **Testable hypothesis**: Retraining pruned network on *same initialization* should provide similar performance.

---

**4b - Testing the hypothesis**

![Graph showing test accuracy over training iterations for different pruning percentages.](image-url)
What happens when we let evolutionary algorithms design the neural networks with limited constraints?

«In principle, there is also no upper limit to any of the parameters. All model depths are attainable, for example. Up to hardware constraints, the search space is unbounded. [...] 
[The best architectures discovered] turn out to be surprisingly simple. » (Real et al., 2017)  

---

6 - Is the architecture everything that matters?

«[We] show that a randomly-initialized neural network can be used as a handcrafted prior with excellent results in standard inverse problems such as denoising, super-resolution, and inpainting. Furthermore, the same prior can be used to invert deep neural representations to diagnose them, and to restore images based on flash-no flash input pairs. » (Ulyanov et al., 2018)\(^5\)

• NNs with *random Gaussian weights* are shown to provide a distance-preserving embedding of the original data.
• Distances are selectively modified based on the angle between data, “pushing away” data from different classes.
• This can also explains the success of architectures where most of the weights are randomly fixed (e.g., reservoir computing).

---

What happens if we also adapt the activation functions, providing a further degree of flexibility to the network? One possibility is the kernel activation function (KAF):

$$g(s) = \sum_{i=1}^{D} \alpha_i \kappa(s, d_i) .$$ (1)

---


### Results on the SUSY benchmark

<table>
<thead>
<tr>
<th>Activation function</th>
<th>Testing AUC</th>
<th>Trainable parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReLU (five hidden layers)</td>
<td>0.8739(0.001)</td>
<td>367201</td>
</tr>
<tr>
<td>ELU (five hidden layers)</td>
<td>0.8739(0.001)</td>
<td></td>
</tr>
<tr>
<td>SELU (five hidden layers)</td>
<td>0.8745(0.002)</td>
<td></td>
</tr>
<tr>
<td>PReLU (five hidden layers)</td>
<td>0.8748(0.001)</td>
<td>368701</td>
</tr>
<tr>
<td>Maxout (one layer)</td>
<td>0.8744(0.001)</td>
<td>17401</td>
</tr>
<tr>
<td>Maxout (two layers)</td>
<td>0.8744(0.002)</td>
<td>288301</td>
</tr>
<tr>
<td>APL (one layer)</td>
<td>0.8744(0.002)</td>
<td>7801</td>
</tr>
<tr>
<td>APL (two layers)</td>
<td>0.8757(0.002)</td>
<td>99901</td>
</tr>
<tr>
<td>KAF (one layer)</td>
<td>0.8756(0.001)</td>
<td>12001</td>
</tr>
<tr>
<td>KAF (two layers)</td>
<td>0.8758(0.001)</td>
<td>108301</td>
</tr>
</tbody>
</table>

**Table 1:** Results on the SUSY benchmark.
In conclusion?
2008: training deep networks was considered very hard / questionable (e.g. universal approximation theorems).

2018: is it possible that training deep networks has become too easy / even a sort of “mental shortcut”?
The incredible success of very deep architectures can be partially attributed to luckily combining:

1. Geometry of the architectures;
2. Initialization of the parameters;
3. Smoothness of the optimization landscape.

None of these is strictly dependent on extremely deep networks.
Ten years ago, an important research question was to design algorithms for deep networks.

Today, it might be more interesting to consider how to design minimalistic neural networks, using as few degrees of freedom as possible.

This is not only an engineering challenge (e.g., mobile), but a theoretical one underpinning our real understanding of these architectures.