For image classification for instance, there has been a trend toward deeper architectures to improve performance.

<table>
<thead>
<tr>
<th>Network</th>
<th>Nb. layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet5 (LeCun et al., 1998)</td>
<td>5</td>
</tr>
<tr>
<td>AlexNet (Krizhevsky et al., 2012)</td>
<td>8</td>
</tr>
<tr>
<td>VGG (Simonyan and Zisserman, 2014)</td>
<td>11–19</td>
</tr>
<tr>
<td>GoogleLeNet (Szegedy et al., 2015)</td>
<td>22</td>
</tr>
<tr>
<td>Inception v4 (Szegedy et al., 2016)</td>
<td>76</td>
</tr>
<tr>
<td>Resnet (He et al., 2015)</td>
<td>34–152</td>
</tr>
<tr>
<td>Resnet (He et al., 2016)</td>
<td>1001</td>
</tr>
<tr>
<td>Resnet (Huang et al., 2016)</td>
<td>1202</td>
</tr>
</tbody>
</table>

“Notably, we did not depart from the classical ConvNet architecture of LeCun et al. (1989), but improved it by substantially increasing the depth.”

(Simonyan and Zisserman, 2014)

A theoretical analysis provides an intuition of how a network’s output “irregularity” grows linearly with its width and exponentially with its depth.
Let $\mathcal{F}$ be the set of piece-wise linear mappings on $[0, 1]$, and $\forall f \in \mathcal{F}$, let $\kappa(f)$ be the minimum number of linear pieces needed to represent $f$.

![Graph of piece-wise linear mappings]

Let $\sigma$ be the ReLU function

$$\sigma : \mathbb{R} \rightarrow \mathbb{R} \quad x \mapsto \max(0, x).$$

If we compose $\sigma$ and $f \in \mathcal{F}$, any linear piece that does not cross 0 remains a single piece or disappears, and one that does cross 0 breaks into two, hence

$$\forall f \in \mathcal{F}, \quad \kappa(\sigma(f)) \leq 2\kappa(f),$$

and we also have

$$\forall (f, g) \in \mathcal{F}^2, \quad \kappa(f + g) \leq \kappa(f) + \kappa(g).$$

Consider a MLP with ReLU, a single input unit, and a single output unit.

$$x_1^0 = x,$$

$$\forall d = 1, \ldots, D, \forall i, \quad \begin{cases} s_i^d &= \sum_{j=1}^{W_{d-1}} w_{ij} s_{i}^{d-1} + b_i^d \\ x_i^d &= \sigma(s_i^d) \\ y &= x_D^1. \end{cases}$$

All the $s_i^d$s and $x_i^d$s are piece-wise linear functions of $x$ with $\forall i, \kappa(s_i^1) = 1$, and

$$\forall l, i, \kappa(x_i^l) = \kappa(\sigma(s_i^l)) \leq 2\kappa(s_i^l) \leq 2 \sum_{j=1}^{W_{l-1}} \kappa(x_j^{l-1})$$

from which

$$\forall l, \max_i \kappa(x_i^l) \leq 2W_{l-1} \max_j \kappa(x_j^{l-1})$$

and we get the following bound for any ReLU MLP

$$\kappa(y) \leq 2^D \prod_{d=1}^D W_d.$$
Although this seems quite a pessimist bound, we can hand-design a network that [almost] reaches it:

So for any $D$, there is a network with $D$ hidden layers and $2D$ hidden units which computes an $f : [0, 1] \rightarrow [0, 1]$ of period $1/2^D$
Given $g \in \mathcal{F}$, it crosses $\frac{1}{2}$ at most $\kappa(g)$ times, which means that on at least $2^D - \kappa(g)$ segments of length $1/2^D$, it is on one side of $\frac{1}{2}$, and

$$\int_0^1 |f(x) - g(x)| \geq \left(2^D - \kappa(g)\right) \frac{1}{2} \int_0^{1/2^D} \left|f(x) - \frac{1}{2}\right|$$

$$= \left(2^D - \kappa(g)\right) \frac{1}{2} \frac{1}{2^{2D}} \frac{1}{8}$$

$$= \frac{1}{16} \left(1 - \frac{\kappa(g)}{2^D}\right).$$

And we multiply $f$ by 16 to get our final result.

So, considering ReLU MLPs with a single input/output:

There exists a network $f$ with $D^*$ layers, and $2D^*$ internal units, such that, for any network $g$ with $D$ layers of sizes $\{W_1, \ldots, W_D\}$:

$$\|f - g\|_1 \geq 1 - \frac{2^D}{2^{D^*}} \sum_{d=1}^{D} W_d.$$ 

In particular, with $g$ a single hidden layer network

$$\|f - g\|_1 \geq 1 - 2 \frac{W_1}{2^{D^*}}.$$ 

To approximate $f$ properly, the width $W_1$ of $g$’s hidden layer has to increase exponentially with $f$’s depth $D^*$.

This is a simplified variant of results by Telgarsky (2015, 2016).
So we have good reasons to increase depth, but we saw that an important issue then is to control the amplitude of the gradient, which is tightly related to controlling activations.

In particular we have to ensure that

- the gradient does not “vanish” (Bengio et al., 1994; Hochreiter et al., 2001),
- gradient amplitude is homogeneous so that all parts of the network train at the same rate (Glorot and Bengio, 2010),
- the gradient does not vary too unpredictably when the weights change (Balduzzi et al., 2017).

Modern techniques change the functional itself instead of trying to improve training “from the outside” through penalty terms or better optimizers.

Our main concern is to make the gradient descent work, even at the cost of engineering substantially the class of functions.

An additional issue for training very large architectures is the computational cost, which often turns out to be the main practical problem.
References


