Deep learning

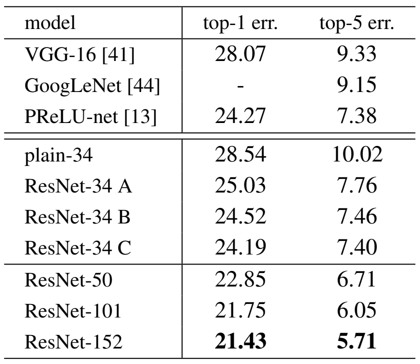
6.1. Benefits of depth

François Fleuret

https://fleuret.org/dlc/

UNIVERSITÉ DE GENÈVE

Using deeper architectures has been key in improving performance in many applications. For instance image classification:



model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

(He et al., 2015)

"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)

Notes

- LeNet5 8 layers (LeCun et al., 1998),
- AlexNet 8 layers (Krizhevsky et al., 2012),
- Inception v1 22 layers (Szegedy et al., 2015),
- Inception v4 76 layers (Szegedy et al., 2016),
- VGG16 has 16 layers (Simonyan and Zisserman, 2014),
- Resnet 34 to 152 layers (He et al., 2015).

Deeper architectures tend to have greater accuracy.

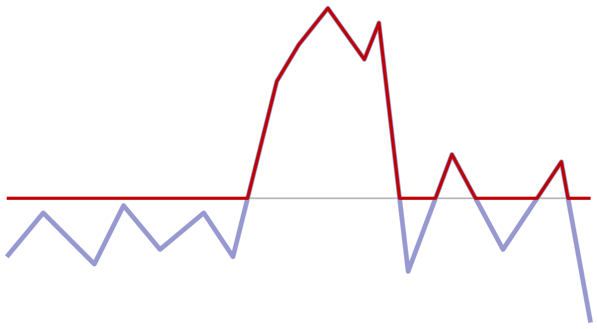
A theoretical analysis provides an intuition of how a network's output "irregularity" grows:

- linearly with its width and
- exponentially with its depth.

Notes

We will see that [a measure of] the complexity of the mapping encoded by a multi-layer model increases exponentially with its depth and linearly with its layers' widths.

Let $\mathscr{F}$ be the set of piece-wise linear mappings on $[0, 1]$, and $\forall f \in \mathscr{F}$, let $\kappa(f)$ be the minimum number of linear pieces in f.



Let σ be the ReLU function

$$\begin{aligned}\sigma : \mathbb{R} &\to \mathbb{R} \\ x &\mapsto \max(0, x).\end{aligned}$$

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

$$\forall f \in \mathscr{F}, \ \kappa(\sigma(f)) \leq 2\kappa(f)\,.$$

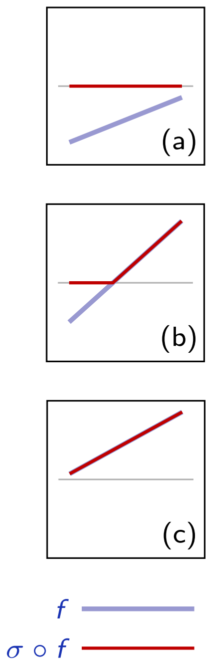
Notes

When we compose σ (ReLU) with f, then three situations can occur for a linear piece:

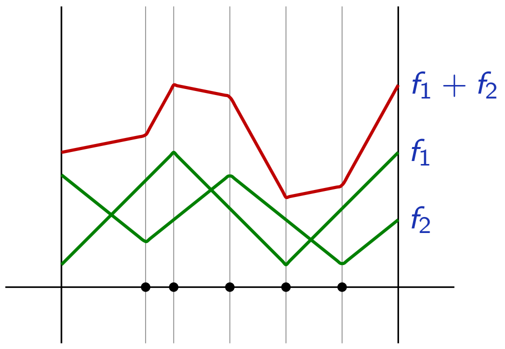
(a) If the linear piece is negative, then it is mapped to 0 by ReLU, and therefore results in at most one linear piece in $\sigma \circ f$.

(b) If the linear piece crosses $y = 0$, then a part of it is positive, and the other is negative, and the latter will be mapped to 0. Therefore this piece for f becomes two pieces for $\sigma \circ f$.

(c) If the linear piece is positive, then it remains unchanged by ReLU, and $\sigma \circ f$ has still one linear piece.



Also, when summing functions, a change of slope in the sum happens only if there was a change of slope in one of the operands.



Hence

$$\forall f_n \in \mathscr{F}, n = 1, \dots, N,\ \kappa\left(\sum_n f_n\right) \leq \sum_n \kappa(f_n)\,.$$

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

$$
\begin{aligned}
& x_1^0 = x, \\
& \forall d = 1, \dots, D, \forall i, \quad \begin{cases} s_i^d & = \sum_{j=1}^{W^{(d-1)}} w_{i,j}^d x_j^{d-1} + b_i^d \\ x_i^d & = \sigma(s_i^d) \end{cases} \\
& y = x_1^D.
\end{aligned}
$$

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

$$\forall d, i, \kappa\left(x_i^d\right) = \kappa\left(\sigma(s_i^d)\right) \leq 2\kappa\left(s_i^d\right) \leq 2 \sum_{j=1}^{W^{(d-1)}} \kappa\left(w_{i,j}^d x_j^{d-1} + b_i^d\right) = 2 \sum_{j=1}^{W^{(d-1)}} \kappa\left(x_j^{d-1}\right)$$

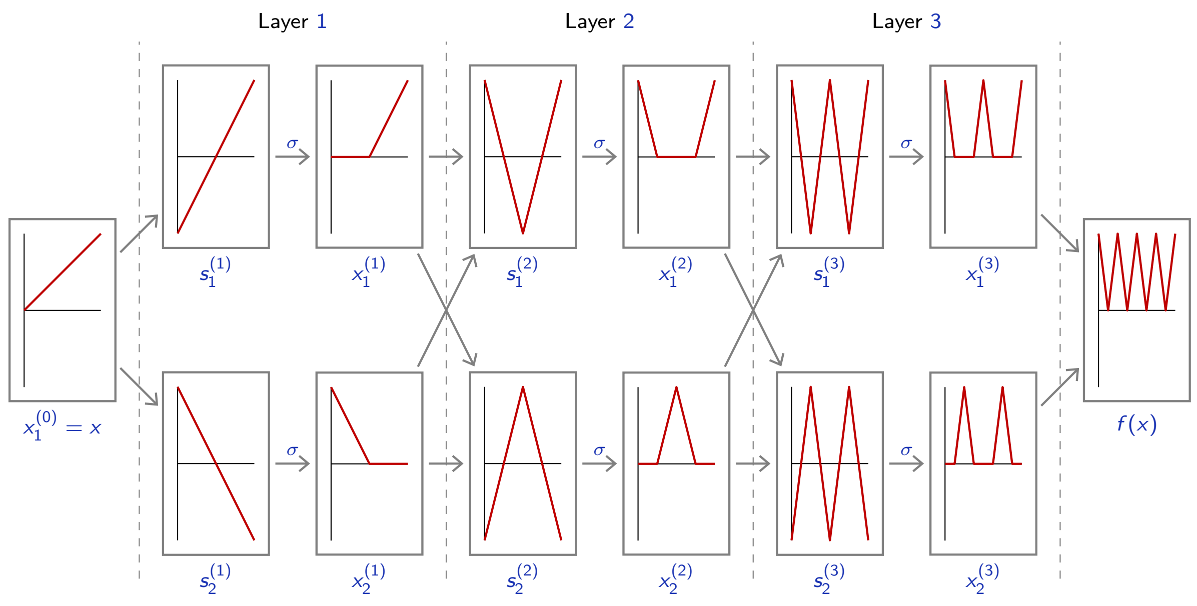
from which

$$\forall d, \max_i \kappa\left(x_i^d\right) \leq 2W^{(d-1)} \max_j \kappa\left(x_j^{d-1}\right)$$

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 pessimistic bound, we can hand-design a network that [almost] reaches it:



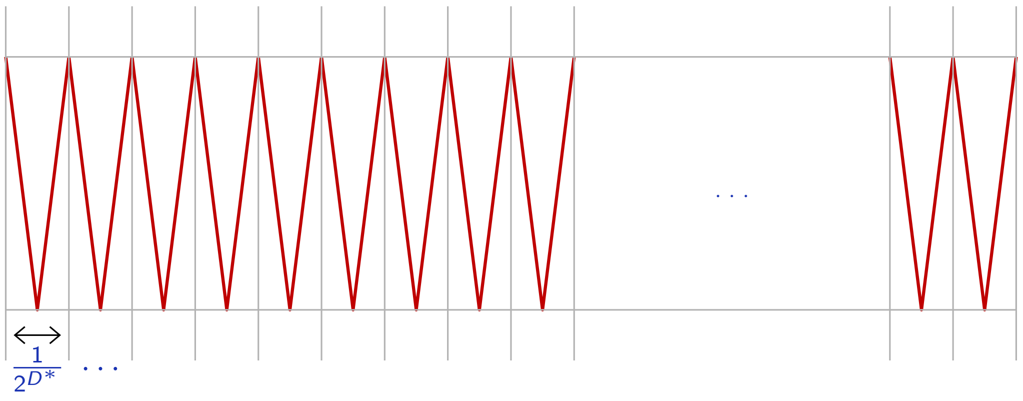
Notes

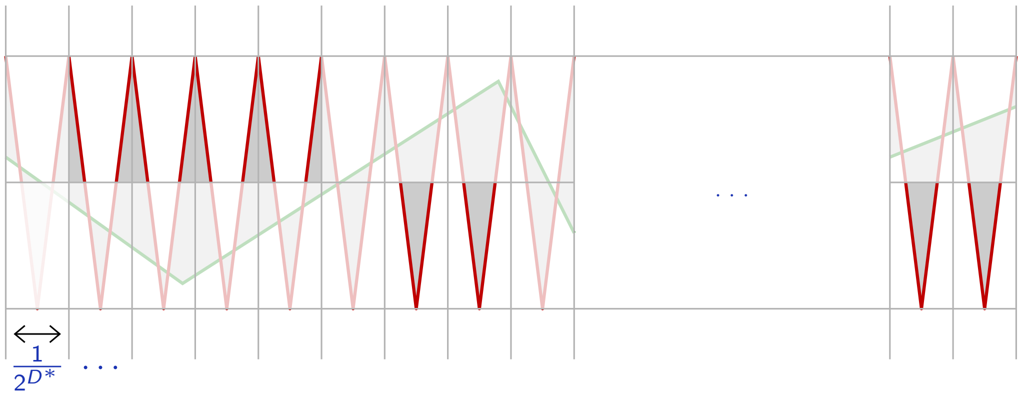
Here we have:

- $s_1^{(1)} = 2x_1^{(0)} - 1$ and $x_1^{(1)} = \max(0, s_1^{(1)})$
- $s_2^{(1)} = -2x_1^{(0)} + 1$ and $x_2^{(1)} = \max(0, s_2^{(1)})$
- $s_1^{(2)} = 2\left(x_1^{(1)} + x_2^{(1)}\right) - 1$
- $s_2^{(2)} = -2\left(x_1^{(1)} + x_2^{(1)}\right) + 1$
- $s_1^{(3)} = 2\left(x_1^{(2)} + x_2^{(2)}\right) - 1$
- $s_2^{(3)} = -2\left(x_1^{(2)} + x_2^{(2)}\right) + 1$
- $f(x) = x_1^{(3)} + x_2^{(3)}$

In the case of 3 layers as depicted here, we have $\kappa\left(f(x)\right) = 8$.

So for any D^*, there is a network with D^* hidden layers and $2D^*$ hidden units which computes an $f : [0,1] \to [0,1]$ of period $1/2^{D^*}$





Given $g \in \mathscr{F}$, 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

$$\begin{aligned}\|f-g\|_1 &= \int_0^1 |f(x)-g(x)| \\ &\geq \left(2^{D^*} - \kappa(g)\right) \frac{1}{2} \frac{1}{2^{D^*}} \int_0^1 \left|f(x) - \frac{1}{2}\right| \\ &= \frac{1}{16}\left(1 - \frac{\kappa(g)}{2^{D^*}}\right).\end{aligned}$$

And we multiply f by 16 to get rid of the $\frac{1}{16}$.

Notes

Here, the f we built is shown in red mapping, and g is a piece-wise linear mapping shown in green.
In any of the slice of width $1/2^{D^*}$, the undimmed gray part has an area of

$$\frac{1}{2}\int_0^{1/2^{D^*}} \left|f(x) - \frac{1}{2}\right| = \frac{1}{2^{D^*}} \frac{1}{8}.$$

The final result on the slide shows that the L_1 distance between f and any function which has a small κ is large.

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 $\{W^{(1)}, \dots, W^{(D)}\}$, since $\kappa(g) \leq 2^D \prod_{d=1}^{D} W^{(d)}$:

$$\|f - g\|_1 \geq 1 - \frac{2^D}{2^{D^*}} \prod_{d=1}^{D} W^{(d)}.$$

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).

Regarding over-fitting, over-parametrizing a deep model often improves test performance, contrary to what the bias-variance decomposition predicts (Belkin et al., 2018).

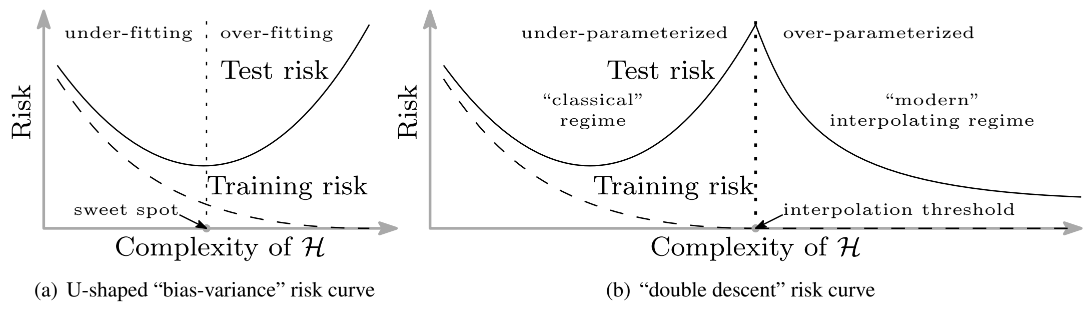


Figure 1: Curves for training risk (dashed line) and test risk (solid line). (a) The classical *U-shaped risk curve* arising from the bias-variance trade-off. (b) The *double descent risk curve*, which incorporates the U-shaped risk curve (i.e., the "classical" regime) together with the observed behavior from using high complexity function classes (i.e., the "modern" interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

(Belkin et al., 2018)

Notes

This results shows that can build more "complicated" models by stacking up layers instead of making a few layers larger.
Surprisingly, those deep architectures reach better test performance in practice. This seems inconsistent with what we have seen with the bias / variance in lecture 2.3. "Bias-variance dilemma": deep architectures should over-fit and the performance decreases in test (left figure).
In the double descent curve (right picture), after the model does perfectly well on the training set (point "interpolation threshold"), the optimization of the parameters is entirely driven by the bias in the model and the regularization mechanism present in the training process (L_2 penalty, SGD, etc.), and the mapping may get better on unseen samples. This interpretation may explain in part why deep networks with billions of parameters do so well although trained with a lesser number of samples.

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

D. Balduzzi, M. Frean, L. Leary, J. Lewis, K. Wan-Duo Ma, and B. McWilliams. **The shattered gradients problem: If resnets are the answer, then what is the question?** CoRR, abs/1702.08591, 2017.

M. Belkin, D. Hsu, S. Ma, and S. Mandal. **Reconciling modern machine learning and the bias-variance trade-off**. CoRR, abs/1812.11118, 2018.

Y. Bengio, P. Simard, and P. Frasconi. **Learning long-term dependencies with gradient descent is difficult**. IEEE Transactions on Neural Networks, 5(2):157–166, Mar. 1994.

X. Glorot and Y. Bengio. **Understanding the difficulty of training deep feedforward neural networks**. In International Conference on Artificial Intelligence and Statistics (AISTATS), 2010.

K. He, X. Zhang, S. Ren, and J. Sun. **Deep residual learning for image recognition**. CoRR, abs/1512.03385, 2015.

S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies, pages 237–243. IEEE Press, 2001.

A. Krizhevsky, I. Sutskever, and G. Hinton. **Imagenet classification with deep convolutional neural networks**. In Neural Information Processing Systems (NIPS), 2012.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. **Backpropagation applied to handwritten zip code recognition**. Neural Computation, 1(4): 541–551, 1989.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. **Gradient-based learning applied to document recognition**. Proceedings of the IEEE, 86(11):2278–2324, 1998.

K. Simonyan and A. Zisserman. **Very deep convolutional networks for large-scale image recognition**. CoRR, abs/1409.1556, 2014.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. **Going deeper with convolutions**. In Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

C. Szegedy, S. Ioffe, and V. Vanhoucke. **Inception-v4, inception-resnet and the impact of residual connections on learning**. CoRR, abs/1602.07261, 2016.

M. Telgarsky. **Representation benefits of deep feedforward networks**. CoRR, abs/1509.08101, 2015.

M. Telgarsky. **Benefits of depth in neural networks**. CoRR, abs/1602.04485, 2016.