

Comments on Sejnowski’s “The unreasonable effectiveness of deep learning in artificial intelligence” (2020).

Leslie S. Smith
Computing Science and Mathematics
University of Stirling
Stirling FK9 4LA
Scotland, UK
email l.s.smith@cs.stir.ac.uk

May 28, 2022

Abstract

Terry Sejnowski’s 2020 paper [arXiv:2002.04806] is entitled “The unreasonable effectiveness of deep learning in artificial intelligence”. However, the paper itself doesn’t attempt to answer the implied question of why Deep Convolutional Neural Networks (DCNNs) can approximate so many of the mappings that they have been trained to model. While there are detailed mathematical analyses, this short paper attempts to look at the issue differently, considering the way that these networks are used, the subset of these functions that can be achieved by training (starting from some location in the original function space), as well as the functions to which such networks will actually be applied.

Terry Sejnowski’s paper is entitled *The unreasonable effectiveness of deep learning in artificial intelligence*, [Sej20]. and he compares and contrasts deep learning with Wigner [Wig60] who marvelled at the limited numbers of parameters in equations. This is in contrast with modern deep learning with its abundance of parameters and extremely high dimensional spaces. While he discusses how we should look to the brain for further optimization, in the paper he does not attempt to answer the question of *why* deep learning is so unreasonably effective. So why can Deep Convolutional Neural Networks (DCNNs) approximate so many of the mappings that they have been trained to model?

There are detailed mathematical analyses, for example [Hor91] [LTR17], [GPEB19] (and the many references within these papers), but this short paper attempts to look at the issue differently, considering the way that these networks are used, the subset of these functions that can be achieved by training (starting from some location in the original function space), as well as the functions that in reality will be modelled.

Deep neural networks have a great many parameters. Firstly, there is the architecture of the network, constrained only by the number of inputs and outputs, and then there are the actual (trainable) parameters of the network itself. For a deep feedforward

network, the number of these is set by the number of layers, and by number of pseudo-neurons (hereafter neurons) in each layer. For DCNNs, it also depends on how the convolutions are implemented. The activation function and output function of each neuron is a parameter, though these are not usually trained. The original topology is also constrained both on the nature of the mapping being approximated, and by how the inputs and outputs for the network have been coded, but these, and the actual internal topology (number of layers, number of neurons per layer, how and where convolution is performed) are not normally optimized during network training: only the weights and biasses are changed.

We write \mathcal{F} for the set of functions that the network can perform, and $f_{\text{init}}^1 \in \mathcal{F}$ for the function that is implemented by the network prior to training. Training will lead to a sequence of functions $f_{\text{init}}^i (i \geq 1)$ eventually converging (assuming that learning is set up so that it does converge) to some $f_{\text{init}}^{\text{final}} \in \mathcal{F}$. Clearly, the sequence of functions will depend on f_{init}^1 , on the actual topology, and on the dataset and precise learning and adaptation rules used to train the network.

Networks are generally trained using a dataset $D = (D_{\text{in}}, D_{\text{out}})$, where $D_{\text{in}} = \{d_{\text{in}}^j : j = 1 \dots t\}$, and $D_{\text{out}} = \{d_{\text{out}}^j : j = 1 \dots t\}$. D is thought of as sampling some underlying classification or function f_D . t is the total number of training examples. d_{in}^j is a vector whose length is the size of the input layer, N_{input} , and d_{out}^j a vector whose length is the size of the output layer, N_{output} . The aim is to be able to map other unseen inputs $d_{\text{in}}^{\text{new}}$ to appropriate $d_{\text{out}}^{\text{new}}$. For classification, there will be a j such that $d_{\text{out}}^{\text{new}} = d_{\text{out}}^j$, but for function approximation, $d_{\text{out}}^{\text{new}}$ may be novel.

For a particular training dataset D , we will normally *choose* an initial topology $\tau = \tau(D)$ that is likely to be able to solve the problem: that is, for which there is likely to be an

$$f_{\text{init}}^{\text{final}} = f_{\text{init}}^{\text{final}}(\tau(D)) \quad (1)$$

which approximates the characteristics of D within some range of error ϵ^1 . We note that if experiment shows that this is not the case, we can choose a different topology τ' until we find one for which this is true.

We note that D is not arbitrary, but has some real-world problem at its root. Thus (for example) we would not be trying to learn a completely random classification of N_{input} binary inputs (which would essentially require 2^N parameters to learn precisely, or rather fewer than that if we choose a nonzero ϵ). But can we say anything else about D , or about the classification or function that D is a sample from (which is more important, since it is generalisation that we are really interested in)? In practice, D is a sample from some underlying function f_D , mapping a set of inputs to a set of outputs, rather than a collection of arbitrary input/output pairs, and it is f_D that we are trying to approximate. In [GPEB19] there is some discussion of the variety of smooth functions for which deep neural networks are good approximators.

¹We are attempting to be general here: the nature of D , and the nature of ϵ will depend on the problem at hand.

The function space \mathcal{F} (which f_D is a member of) can be characterised as

$$\mathcal{F} = \mathbb{R}^{N_{\text{input}}} \times \mathbb{R}^{N_{\text{output}}} \quad (2)$$

where N_{input} is the number of input units and N_{output} is the number of output units. However, because computers are finite devices, the space is much smaller than this, more like

$$F' = (2^{64})^{N_{\text{input}}} \times (2^{64})^{N_{\text{output}}} \quad (3)$$

assuming that both inputs and outputs are coded in 64 bits (as is currently likely to be the case for 64 bit floating point coded inputs).

If we are successful in approximating f_D , this means that there is (and we have found) a τ such that $f_{\text{init}}^{\text{final}}(\tau)$ is close enough to $f_D \in F'$. But what sorts of mappings can the network produce? This clearly depends on the number of layers in the network, the way in which convolving layers are used, and (assuming the usual form of artificial neuron is used), the function used to compute the output from the activation of a neuron, and is the question asked in [GPEB19].

Earlier we noted that f_D was not arbitrary: but if we are to understand how $f_{\text{init}}^{\text{final}}(\tau)$ can be close enough to $f_D \in F'$ we need to refine what *not arbitrary* might mean. We identify two issues here:

Issue 1 the nature of likely f_D 's, defined by the nature of the problem from which D was sampled

Issue 2 the actual domain of the function that D was drawn from

Consider an image classification problem: for such a problem, we start with images (that is, d_{in}^i is a coded image, probably an X by Y pixel image, with each pixel coded in 8 (monochrome) or 24 (colour) bits). These images result from incident light (from many sources, usually) reflecting from (and/or being transmitted by) points in the world, passing through the point spread function of each pixel detector at the camera, resulting in the X by Y vector that we are trying to interpret.

Considering Issue 1 above, there are issues of size and translation invariance in f_D . In addition, we would expect the same classification for an image and a slightly blurred version of that image, and we would expect the classification to remain the same under a range of illuminations. Thus the f_D from which D is drawn is quite tightly constrained. This effect is not limited to image classification: something similar is true for sound classification, in that small alterations in the intensity or pitch of the sound, or in the reverberation of the sound prior to transduction would again not be expected to alter the classification of the sound. Where this becomes more difficult is in the case of categorical perception, as in figure 1: each change is very small, but at some point one needs to change the classification.

A related issue arises with context: see figure 2 (from [KP18]). While the first issue may be soluble using a neural network, the second one requires the use of the context of the symbols on either side, as the central images are identical.

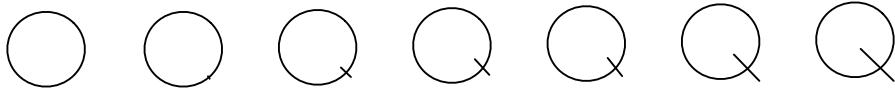


Figure 1: When should an image classifier say that the letter O turns into a letter Q?

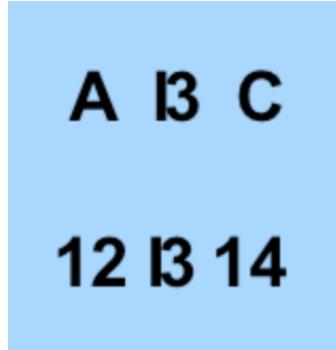


Figure 2: When should an image classifier say that the middle symbol is B and when 13?

For Issue 2 above, one might at first think that any possible vector that is X by Y with 8 (or 24) bit elements might be a possible d_{in}^i . However, this is generally not the case: in general, pixel values do not change suddenly or randomly between adjacent locations. If one considered \mathcal{F} rather than F' , and considered smooth functions, one might be able to restrict it by bounding local partial derivatives. The overall effect is that although f_D is sampled from a function on the input space (whether $\mathbb{R}^{N_{\text{input}}}$ or $(2^{64})^{N_{\text{input}}}$), many (indeed most) elements of this input space will never occur. Thus, testing $f_{\text{init}}^{\text{final}}(\tau)$ on randomly selected elements of the input space may be inappropriate, and lead to unexpected results as in [SZS⁺13], as well as more or less random patterns being misclassified (when they should be marked as unclassifiable).

What do these issues say to *The unreasonable effectiveness of deep learning in artificial intelligence?*

Firstly, we note that the *unreasonable effectiveness* is posited on the selection of an appropriate network $\tau(D)$ by the developer: while the mathematics might suggest that a single very wide hidden layer network might suffice [Hor91] the actual networks used do not look like this, primarily because appropriate generalisation is more important than being able to create a precise function. The arguments in [LTR17] and [GPEB19] are unaffected, but it is clear that the space of functions (particularly classifiers) that require to be approximated is smaller than might have been imagined. There is a sense in which the functions are relatively smooth, and constrained further by the application domain. Further, the neural network will always classify or approximate any input from the input domain, whether that input is possible (in terms of the actual problem input

domain) or not.

Sejnowski discusses some unanswered questions, specifically why it is possible to generalize from so few examples while using so many parameters. Based on what we have explained, because the classifier (or function approximator) is likely to include dimensionality reducing sections within the network (such as relatively narrow layers, or convolution layers), these inputs may be well away from the manifold that f_D was actually trained on (and therefore gives appropriate results for), and may give inappropriate results. However, actual data from the same source as the training data will not suffer from this problem. Thus we conclude that the unreasonable effectiveness while certainly true, is perhaps just a little less surprising.

Do the issues above affect the reachability of an appropriate $f_{\text{init}}^{\text{final}}(\tau)$? Issue 2 above suggests that testing of the function should be on problem-appropriate possible inputs, rather than from inputs drawn randomly from the domain. Thus $\epsilon = \epsilon(f_D, \tau, f_{\text{init}}^1)$ should not be evaluated on randomly selected elements of the input domain. This fits with many test problems, where a dataset is supplied, and is used for training and testing. It is however not clear that this affects reachability directly: however, the user selection of τ (and f_{init}^1) is clearly important here.

Acknowledgement

Thanks to Andrew Abel for useful comments on an earlier draft.

References

- [GPEB19] Philipp Grohs, Dmytro Perekrestenko, Dennis Elbrächter, and Helmut Bölcskei. Deep Neural Network Approximation Theory. *arXiv.org*, January 2019, 1901.02220v1.
- [Hor91] K Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.
- [KP18] Jim W Kay and W A Phillips. Contrasting information theoretic decompositions of modulatory and arithmetic interactions in neural information processing systems. *arXiv.org*, March 2018, 1803.05897v1.
- [LTR17] Henry W Lin, Max Tegmark, and David Rolnick. Why Does Deep and Cheap Learning Work So Well? *Journal of Statistical Physics*, 168(6):1223–1247, July 2017.
- [Sej20] Terrence J Sejnowski. The unreasonable effectiveness of deep learning in artificial intelligence. *Proceedings of the National Academy of Sciences of the United States of America*, vol. VG-1196-G:201907373, January 2020.

- [SZS⁺13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv.org*, December 2013, 1312.6199v4.
- [Wig60] E Wigner. The unreasonable effectiveness of mathematics in the natural sciences . *Communications in pure and applied mathematics*, 13(1), 1960.