Survey on "why does deep and cheap learning work so well?"

Devan Shah
devan.shah@princeton.edu

December 19, 2024

Abstract: This work surveys the 2017 paper "Why does deep and cheap learning work so well" [21], by Henry W. Lin, Max Tegmark, and David Rolnick. We will explore the connections between problems that deep learning can solve and the types of problems that occur in the natural world, and we will reach model expressiveness results motivated by results in physics and a study of Hamiltonians, low-degree polynomials, and compositionality. Our exposition largely follows that in [21; 25], and along the way we will provide a mathematical introduction and introduce relevant recent results.

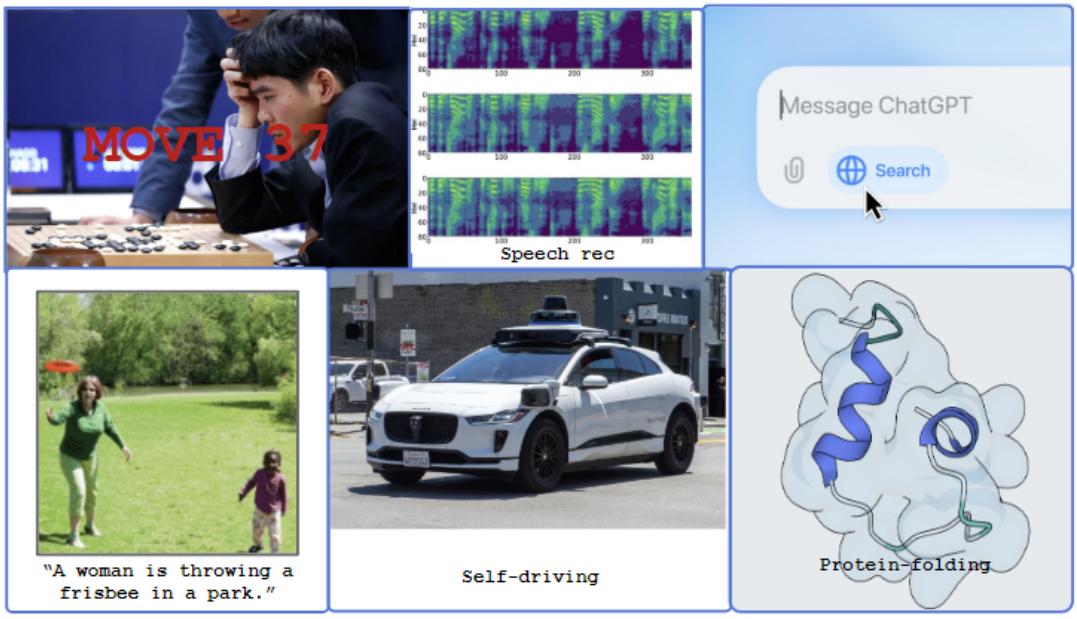

Figure 1: Examples of the success of deep learning including at board games, speech recognition, language generation, image captioning, self driving, and protein folding [1; 2; 3; 12; 17; 22].

1 Introduction

To put it simply, deep learning is incredible. Parameterized models have had great success in tasks ranging from driving cars to generating text to generating videos [5; 6; 7], tasks that involve a complex understanding of the world and, in some cases, such as self-driving and drug discovery, exceed the capacity of a human [21].

Models for image identification and many common tasks can be successfully learned with only a few hundred thousand to million parameters [16], yet there are more functions from images to labels than atoms in the universe [21]. Even though million-parameter neural networks can only represent an incredibly small fraction of the functions that exist from images to labels, they are able to represent the functions we desire. As an arbitrary functions is close to random noise, it is clear that the functions we care about have some sort of structure. The fundamental question we arrive at is then: *for the problems we care about, what are the properties of the underlying systems, and why are neural networks excellent at modeling systems with these properties*?

2 Background

2.1 Neural Networks

Since the AlexNet model surpassed classical machine learning techniques in 2015 [8], deep learning models have surpassed the performance of humans in many fields and are now the norm for machine learning tasks. The most simple neural network, the feed-forward neural network or multi-layer perceptron, models a target function with successive affine transformations and non-linearities, as specified by the following equations:

$$\begin{aligned} f(\mathbf{x}) &= \sigma_L \mathbf{A}_L \cdots \sigma_2 \mathbf{A}_2 \sigma_1 \mathbf{A}_1 \mathbf{x} \\ \mathbf{A}_i \mathbf{x} &= \mathbf{W}_i \mathbf{x} + \mathbf{b}_i \end{aligned} \tag{1}$$

Where $\mathbf{W}$ and $\mathbf{b}$ are parameters learned by a gradient descent-based optimizer. The value L is considered the depth of the network, with layers excluding the input and output considered hidden layers. Generally, a machine learning practitioner will pick a depth L and the hidden layer sizes, where the ith hidden layer is the dimension of the output of $f^{(i)}(\mathrm{x}) = \sigma_i \mathbf{A}_i \sigma_{i-1} \mathbf{A}_{i-1} \cdots \mathbf{A}_1 \mathbf{x}$. This information specifies the amount and size of each of the $\mathbf{W}$ and $\mathbf{b}$ matrices. The function σ is a non-linear function, such as typically sigmoid or relu, which are defined as follows:

$$\begin{aligned} \mathrm{ReLU}(\mathbf{x}) &= \max(0, \mathbf{x}) \\ \mathrm{sigmoid}(\mathbf{x}) &= \frac{e^{\mathbf{x}}}{\sum_i e^{\mathbf{x}_i}} \end{aligned} \tag{2}$$

In the above expression, max is applied coordinate-wise and the exponential function is applied coordinate-wise as well. For instance,

$$\begin{aligned} &\mathrm{ReLU}([2, -3, 4, -1]) = [2, 0, 4, 0] \\ &\mathrm{sigmoid}([2, -3, 4, -1]) = \frac{[e^2, e^{-3}, e^4, e^{-1}]}{e^2 + e^{-3} + e^4 + e^{-1}} = [0.26, 0.00, 0.72, 0.01] \end{aligned} \tag{3}$$

To train neural networks, we specify a loss function, which measures how far the prediction of our function is from capturing the true data. For a dataset of size n,

$\mathcal{D} = (x, y)_{i=1}^{n}$ consists of n input-output pairs representing a dynamic we wish our neural network to capture. In this case, a natural loss function may be mean squared error, as defined below:

$$\mathcal{L}_{\mathbf{W},\mathbf{b}}(\mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2 \tag{4}$$

Once we have computed the loss, we can then update the weights $\mathbf{W}, \mathbf{b}$ to minimize the loss. To do so, for each element p_t of $\mathbf{W}$ or $\mathbf{b}$, we find the updated weight p_{t+1} by shifting p_t in the direction that minimizes the loss, which is mathematically expressed as:

$$p_{t+1} = p_t - \eta \nabla_p \mathcal{L}(\mathcal{D}) \tag{5}$$

It is thus important for the loss function and non-linearity to be differentiable so that this gradient can be computed. We can then efficiently compute the gradients with the back-propagation algorithm [19], and we update our weights by the above expression. The above updating algorithm is traditional gradient descent, but there are many modifications of it. For instance, some parameter update algorithms have a unique η per weight, an adaptive η, only use a subsample of the data per step, or leverage gradient momentum [18].

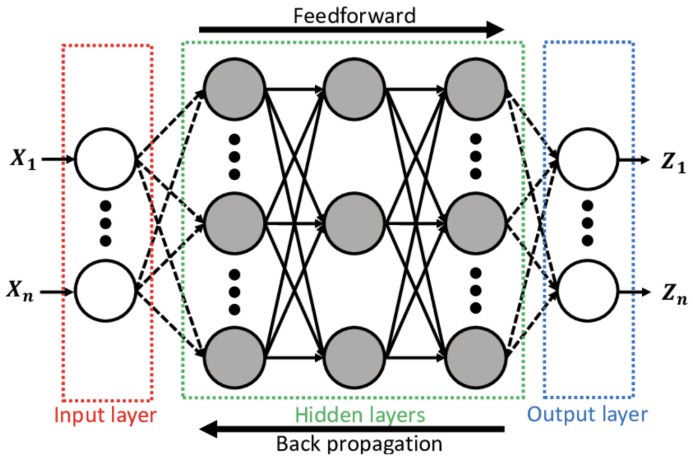

Figure 2: A schematic of a Feed Forward Neural Network showcasing an alternative interpretation of Equation 1. The arrow between node i at layer $\ell - 1$ and node j at layer ℓ represents multiplying node i by W_{ji}^{ℓ} and passing the value to node j. At each node, the inputs are accumulated, the bias term b_j^{ℓ} is added, and the non-linearity is applied [15].

2.2 Deep and Shallow Neural Networks

Neural Networks with few hidden layers (generally $L \leq 3$ and so one hidden layer) are considered shallow whereas neural networks with many hidden layers are considered deep.

By the Universal Approximation Theorem, with reasonably chosen σ (such as ReLU or Sigmoid), the shallow neural network $f(\mathbf{x}) = A_2\sigma_1 A_1 x$ can approximate any continuous function to arbitrarily small error if it has sufficiently large hidden dimension (i.e. the dimension W_1 projects to) [13].

However, in practice, the hidden dimension required to approximate desirable functions by shallow neural networks is prohibitively large. Thus, it begets the question on why

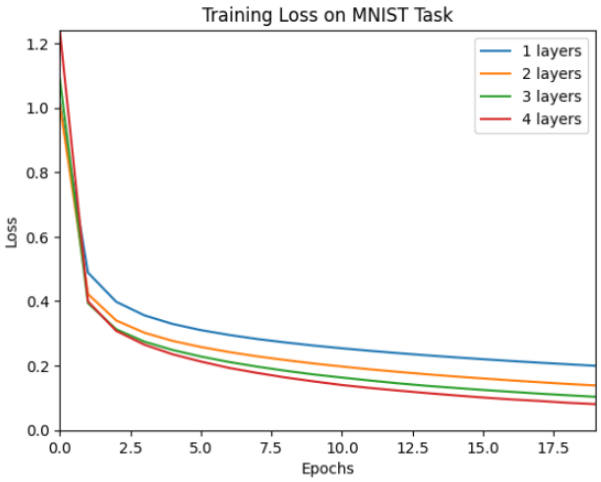

Figure 3: Performance of neural networks with $\tilde{3}500$ parameters and different depth on learning MNIST.

deep neural networks, those with many hidden layers, have drastically greater learning power and more efficient representation than their shallow counterparts. More generally, deep models solve challenging problems rather "cheaply", solving problems with relatively few parameters compared to the complexity of arbitrary functions on the data.

For example, when choosing a model architecture with 3500 learnable parameters to recognize digits from the MNIST dataset, as shown in Figure 3, the deeper models consistently outperform. As we will show, for a large class of problems, deep neural networks are significantly more expressive than their shallow counterparts.

When considering deep learning architectures, we generally care most about the following factors [21]:

1. **Expressibility**: What class of functions can this model express?
2. **Efficiency**: How many parameters/neurons/flops are required to approximate a function?
3. **Learnability**: How quickly can we learn good parameters to approximate a function?

In this paper, as in [21], we will largely focus on the expressibility and efficiency of neural networks.

3 Polynomials in networks and nature

3.1 Neural Networks representing polynomials

Deep learning networks are able to efficiently model polynomials. Restating the corollary from [21],

Corollary: For any given multivariate polynomial and any tolerance $\epsilon > 0$, there exists a neural network of fixed finite size N (independent of ϵ) that approximates the polynomial to accuracy better than ϵ. Furthermore, N is bounded by the complexity of the polynomial, scaling as the number of multiplications required times a factor that is

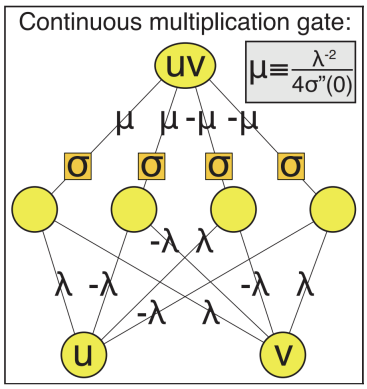

Figure 4: A continuous multiplicate gate from [21].

typically slightly larger than 4 [21].

As opposed to the Universal Approximation theorem [13], this result is in fact constructive. The following construction is from [21]. To prove the corollary, let us first prove a helpful lemma [21]:

Lemma: Let $\mathbf{f}$ be a neural network of the form $\mathbf{f} = \mathbf{A}_2\sigma\mathbf{A}_1$, where σ acts element-wise as a non-linearity. Let the input layer, hidden layer, and output layer have sizes 2, 4, and 1, respectively. Then, $\mathbf{f}$ can approximate a multiplication gate arbitrarily well [21].

Consider the Taylor Expansion of the non-linearity σ chosen:

$$\sigma(u) = \sigma(0) + \sigma'(0)u + \sigma''(0)\frac{u^2}{2} + \mathcal{O}(u^3) \tag{6}$$

Without loss of generality, let $\sigma''(0) \neq 0$ as, since σ is non-linear, $\exists x$ s.t. $\sigma''(x) \neq 0$ and so we can leverage the bias terms in A_1 to shift accordingly.

Thus note that:

$$\begin{aligned} m(u,v) &= \frac{\sigma(u+v) + \sigma(-u-v) - \sigma(u-v) - \sigma(-u+v)}{4\sigma''(0)} \\ &= \frac{\frac{\sigma''(0)}{2}\Big((u+v)^2 + (-u-v)^2 - (u-v)^2 - (-u+v)^2\Big) + \mathcal{O}(u^3+v^3)}{4\sigma''(0)} \\ &= uv(1 + \mathcal{O}(u^2+v^2)) \end{aligned} \tag{7}$$

And thus $m(u,v)$ provides a multiplicative approximation. Moreover,

$$\lim_{\lambda\to 0}\frac{m(\lambda u, \lambda v)}{\lambda^2} = \lim_{\lambda\to 0}\frac{1}{\lambda^2}\cdot\lambda^2 uv\Big(1 + \mathcal{O}(\lambda^2(u^2+v^2))\Big) = uv \tag{8}$$

Thus, by scaling down the bias and weight terms in earlier layers ($\mathbf{A}_1 \to \lambda\mathbf{A}_1$) and scaling up in subsequent layers ($\mathbf{A}_2 \to \lambda^{-2}\mathbf{A}_2$), we can achieve an arbitrarily strong approximation for multiplication, as it shown in Figure 4.

It immediately follows that, for a monomial $p(x_1, x_2, \dots) = cx_1^{r_1}x_2^{r_2}\cdots$ of degree d, we can represent the monomial to arbitrarily high accuracy with a neural network with $2d$ layers by sequentially composing the above multiplication gates. In fact, by parallelizing

the multiplications, we can represent the monic polynomial with a neural network with $2\lceil \log d \rceil$ layers. As it would take a single additional layer to compose all the monic polynomials, we thus have that a polynomial of degree d can be represented by a neural network with $2\lceil \log d \rceil + 1$ layers, and neurons linear in the amount of multiplications.

Thus, we have shown the corollary up to a constant factor, and we have that deep neural networks can exactly represent polynomials if the depth is sufficient. Importantly, we will now show that many problems in nature boil down to well-representing polynomials.

3.2 Polynomials in Nature

The following in this section is largely taken from [21]. First, note that by Bayes Thm., for data x, label y, and alternative labels y' :

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{\sum_{y'} p(\mathbf{x}|y')p(y')} \tag{9}$$

Import terminology from physics, we will consider the Hamiltonian $H_y(x)$:

$$\begin{aligned} H_y(x) &\equiv -\ln p(\mathbf{x}|y) \\ \mu_y &\equiv -\ln p(y) \end{aligned} \tag{10}$$

In physics, the Hamiltonian refers to the energy of the state x given the parameter y, whereas in machine learning it is more common to hear the term surprisal or self-information [21]. In a sense, a large Hamiltonian corresponds to $p \approx 0$ and thus we may be "surprised" to see the event given a prior idea of the distribution. We can thus reformulate Bayes Thm. as:

$$p(y|\mathbf{x}) = \frac{1}{N(x)} e^{-[H_y(x)+\mu_y]} \tag{11}$$

With $N(x)$ defined as the normalizing term:

$$N(x) \equiv \sum_y e^{-[H_y(x)+\mu_y]} \tag{12}$$

Finally, note that, leveraging the softmax function $s_{\max}(\vec{y}) = \frac{e^{\vec{y}}}{\sum_i e^{y_i}}$ and extending $H(x) = [H_{y_1}(x), \ldots, H_{y_n}(x)]$, $\mu = [\mu_{y_1}, \ldots, \mu_{y_n}]$, and $p(\vec{y}|x) = [p(y_1|x), \ldots, p(y_n|x)]$ to compactify the vector operations, we reach the observation that Bayes Thm. is equivalent to:

$$p(\vec{y}|x) = s_{\max}[-H(x) - \mu] \tag{13}$$

Thus, if we wanted a neural network to learn the prediction task, as $s_{\max}$ is a common final activation for neural networks, the remainder of the neural network needs simply to learn the Hamiltonian, with μ learnable as a bias vector.

Critical Observation: "The Hamiltonians that show up in physics are not random functions, but tend to be polynomials of very low order, typically of degree ranging from 2 to 4." [21]

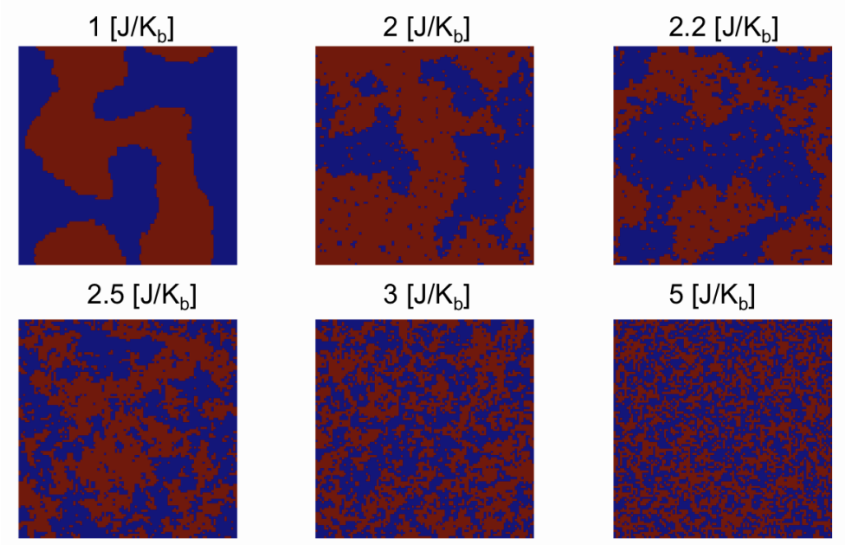

Figure 5: The Ising Model (image from [9]). "For any two adjacent sites $i, j \in \Lambda$ there is an *interaction* J_{ij}. Also a site $j \in \Lambda$ has an *external magnetic field* interacting with it. The energy of a configuration σ is given by the Hamiltonian function: $H(\sigma) = -\sum_{\langle i,j \rangle} J_{ij}\sigma_i\sigma_j - \mu \sum_j h_j\sigma_j$" [4]

.

Thus, if the critical observation is true, based on our prior observations on the power of deep neural networks to represent polynomials, this is a strong insight onto why neural networks are able to represent many of the systems in nature.

But why might this observation be true? [21]:

- The Hamiltonian of the standard model of physics particles has degree $d = 4$. As does many approximations – Maxwell's Equations, Navier-Stokes Equations, Alven Equations, and Ising Models.
- For systems that can be studied perturbatively, Taylor's Theorem suggests a low-order polynomial expansion suffices.
- By the Central Limit Theorem, many limiting distributions are Gaussian, and thus $-\ln(p)$ is quadratic.
- Maximum entropy distributions (i.e. Gaussian) tend to lead to polynomial Hamiltonians with low degree.

Another reason that Hamiltonians tend to be low-degree is for the reason of locality – "things effect what is in their immediate vicinity" [21]. For many settings, i.e. Figure 5, principles of locality limit which variables can interact, often allowing us to bound a Hamiltonian's degree (in graph-modelable systems) by the amount of neighbors a vertex can interact with.

Thus, as many Hamiltonians are polynomial, and modeling these Hamiltonians allows us to model conditional distributions, much of the predictive power of neural networks may arise from their predictive power on polynomial inputs.

3.3 No Flattening Results

As important as showing the representative capacity of deep neural networks is showing why shallow networks cannot achieve similar performance. By the Universal Approximation theorem, shallow networks can represent arbitrary continuous functions arbitrarily well, but the hidden limitation is they often need to be much wider to do so. This introduces us to the field of no-flattening results, which showcase the parameter increase required to represent the same function modeled by a deep neural network on a shallow neural network.

Most notably, we have previously shown that deep neural networks can represent polynomials efficiently. However, [21] proves the that "no [shallow] neural network can implement an n-input multiplication gate using fewer than 2^n neurons in the hidden layer," severely restricting the capacity of a shallow network to model a polynomial. For example, a deep neural network can represent a degree 4 polynomial in 20 variables with $4 \cdot 20 = 80$ neurons, yet a shallow network would require $2^{20} > 1,000,000$ neurons. Thus, as we have just shown the importance of modeling polynomials for modeling natural systems, this provides a severe limitation in the capacity of shallow networks to do so.

4 Hierarchy and Compositionality

Critical Observation: Nature creates datasets compositionally and hierarchically [21].

4.1 Hierarchy

Many processes cannot be modeled with shallow dynamics but rather have more complex hierarchical structure. For example, when writing an article, we are generating character by character of course, but also sentence by sentence and idea by idea. Figure 6 [20] showcases the decay in mutual information across distance in a variety of systems. In a Markov model, a purely sequential system, information decays multiplicatively with each step and thus exponentially in the system. However, we note that more complex systems, such as language, the genome, and the Ising model show approximately linear decay.

Rather than viewing each of these systems as sequential, it is better to recognize the hierarchical structure – as we can model every "step" on the hierarchy as contributing a multiplicative factor of information loss, this system is better represented as having $\mathcal{O}(\log(n))$ hierarchy, and thus elements distance n away have $\mathcal{O}(n)$ mutual information.

Deep neural networks excel at efficient information accumulation and hierarchical processing. The convolutional layer, common in many image processing and prediction tasks, enables efficient local accumulation with deep successive layers of convolutions [16] accumulating information hierarchically. The ResNet architecture assembles exceptionally long convolutional chains to take advantage of the hierarchical structure of images, and 1-dimensional convolution performs similar for sequences [16]. On the ImageNet task, [16] trains convolutional models with up to 152 layers, seeing consistent improvement. In the next section on compositionality, we will show examples of how deeper networks can learn both hierarchical and compositional patterns in their deeper layers.

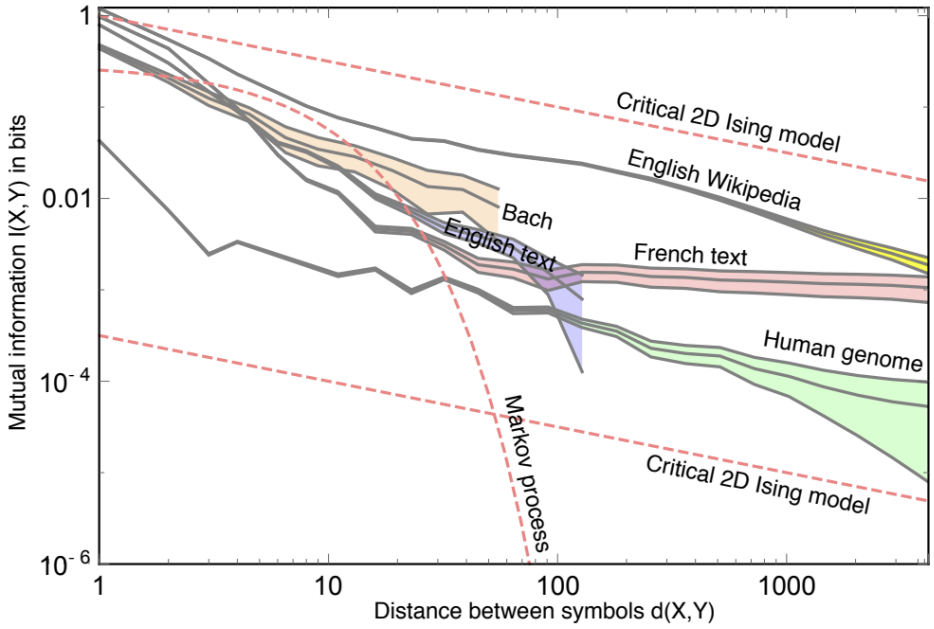

Figure 6: Decay of mutual information (in bits per symbol) as a function of seperation $d(x, y) = |i - j|$ [20].

4.2 Compositionality

Inherent in hierarchy is often composition. For many generative processes in nature, we can decompose them into a series of simpler steps [21]. In Figure 7, we can consider simple example processes that illustrate the compositionality in question.

With complex processes decomposable into multiple simpler steps, it makes sense that a model architecture with many layers, each of which could perform small non-linear steps, would have powerful representative capacities. At the mathematical level, a deep network could learn the minimal sufficient statistic for the Markov operator representing each step, allowing a classification problem based on that generative process to be decomposed into many steps, each effectively inverting the corresponding generative step [21].

Analyzing deep neural networks, they take full advantage of compositional learning, with early layers accumulating information and building new features for subsequent layers to process. For example, in a convolutional architecture like as discussed earlier, when we visualize what features are learned at deeper and deeper layers, we notice a significant hierarchy in complexity, with early convolutions acting as edge detectors and later filters activated by successively more complex features, as is shown in Figure 8.

Another common architecture for classification and generative models is the transformer, which powers much of the latest wave of generative technology such as Chat-GPT and Dall-E [6], which consists of many successive attention and multi-layer perceptron layers. At a base level, the attention module accumulates information and relevant context to build a contextful representation of each token (word, grid of pixels, etc.) in the input [26]. The module itself is incredibly interesting and we point readers to [24] for more details. When visualizing, in the BERT language model, what attention heads in subsequent layers recognize, we can view a successive increase in the complexity of task, with early attention heads attending to perhaps only the next token, and attention heads deep in the model can attend to complex grammar [11]. Some examples of what the different attention heads attend to are included in Figure 9.

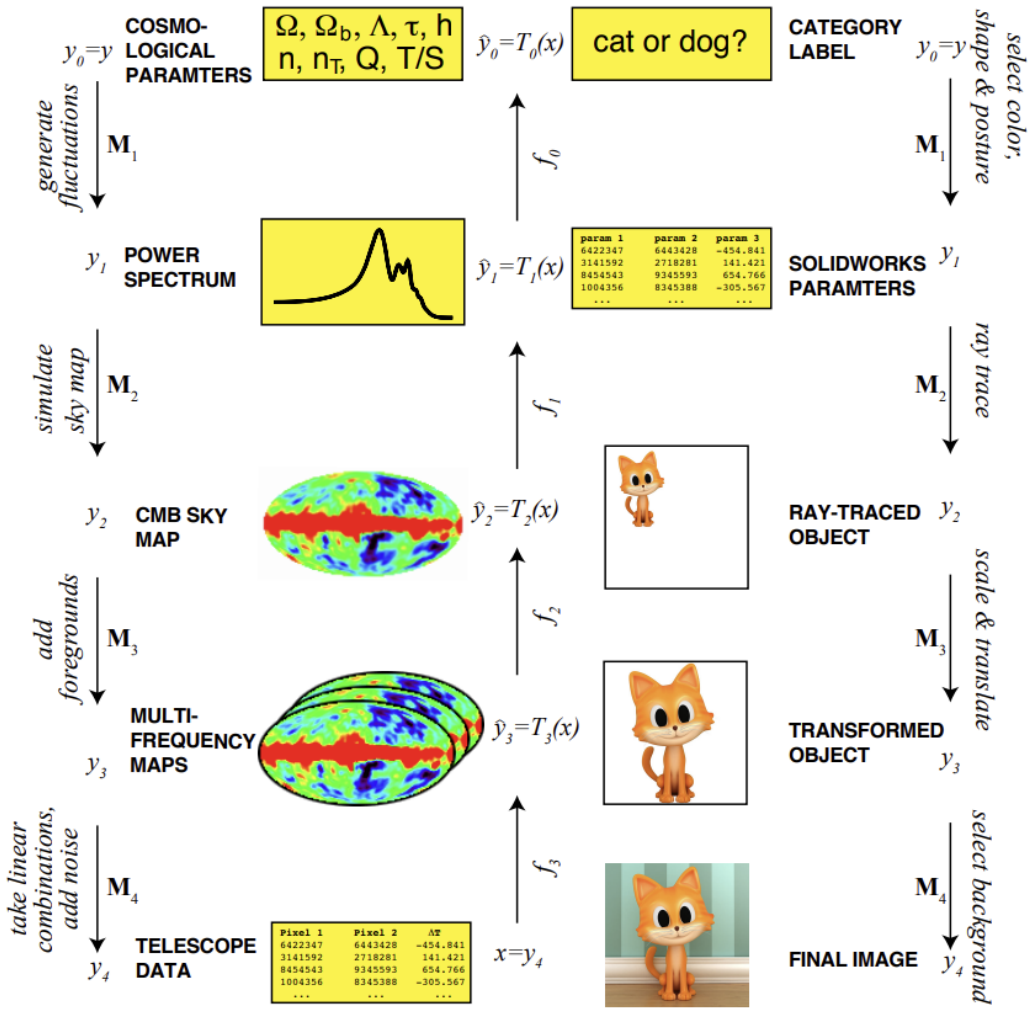

Figure 7: Causal hierarchy examples relevant to physics (left) and image classification (right) [21].

So in deep neural networks, we view the benefits of compositionality in action, with deeper layers able to attend to more interesting patterns due to the preprocessing of earlier layers.

5 Discussion and Conclusion

5.1 Limitations

There are some limitations of the work in [21], and our discussion of the limitations stems from the discussion in [21] itself. Most notably, [21] focuses on the efficiency and expressibility of deep neural networks, but not the learnability of the functions at hand. For instance, in the multiplication gate detailed in Figure 4, it may prove challenging for a gradient descent based optimizer to learn λ and λ^{-2} for small λ, which is required for the Taylor expansion-based approximation to hold. For example, in training such a gate with the Adam optimizer, we were unable to learn better than $\lambda \approx 0.10$. However, we found that, with a slightly deeper network with 75 parameters, we were able to learn multiplication well, and so we still agree that few-parameter learnable multiplication

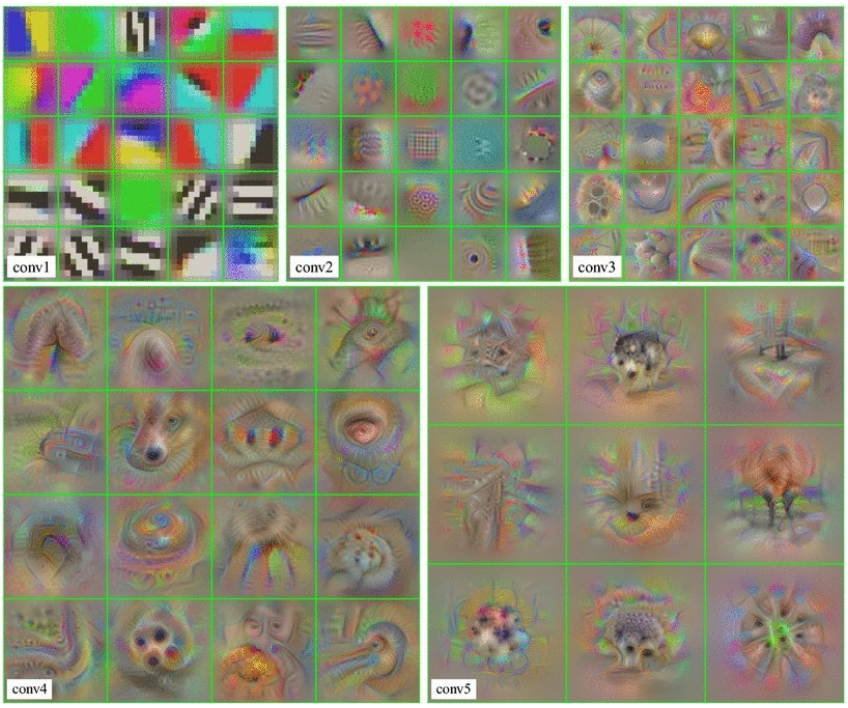

Figure 8: Images maximizing the first filters of each convolutional layers in VGG-M [23].

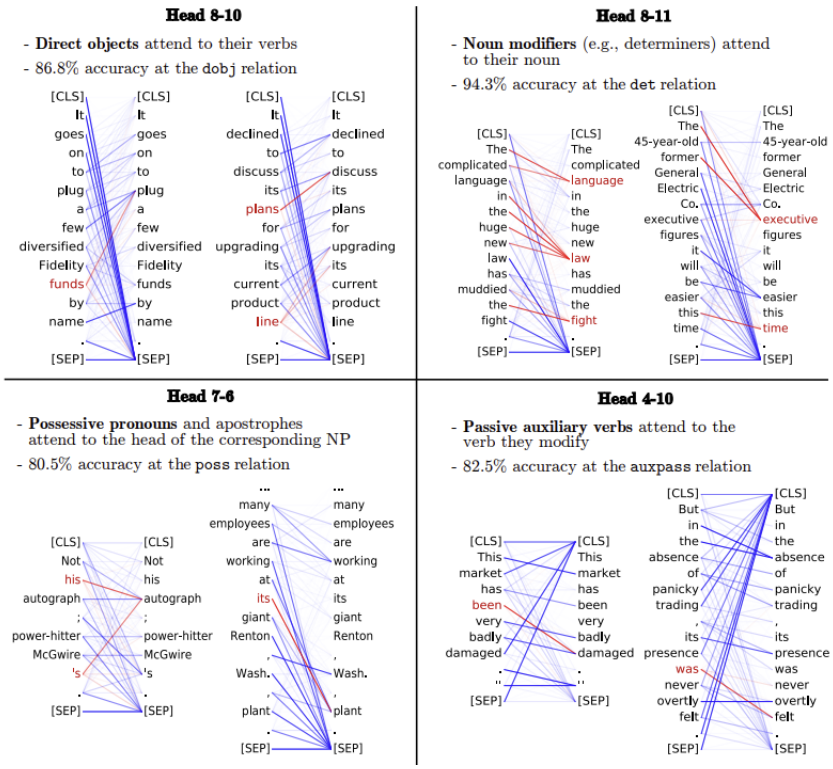

Figure 9: BERT attention heads that correspond to linguistic phenomena. The lines correspond to the strength of the attention weight. These examples indicate how syntax-sensitive behavior and complex grammatical reasoning can emerge in later layers of a transformer neural network [11].

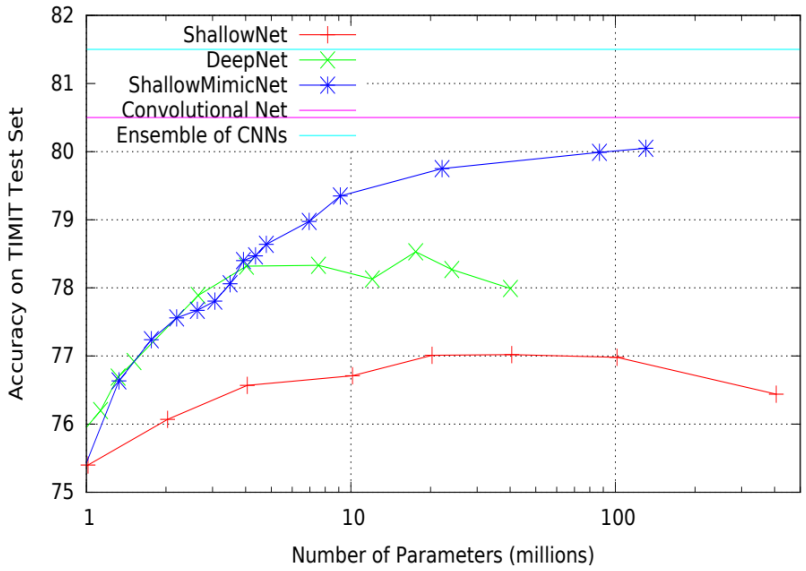

Figure 10: Performance of Shallow vs. Deep multi-layer perceptrons, with a CNN model and CNN ensemble as baselines. ShallowMimicNet is trained with teacher-student distillation from a deeper network [10].

gates exist, but we believe more care should be given to whether certain gates are indeed learnable.

As a separate comment, an important element of the polynomial argument is that many hamiltonians are low-degree. Although we agree this is true, they are notably only low-degree if you have access to the right variables. In many systems, you may not have access to the ground-truth variables that govern the system, and instead the neural network must learn a significantly more complex function. For example, if suppose you wanted to predict crop growth, you could do so easily given the correct variables regarding the soil quality and climate, but if you do not have access to that information, instead only knowing how other crops grew in prior years, now the prediction task becomes significantly more complex and likely requires a much more advanced model.

A related question to how neural networks can represent the functions we care about is, even presuming the existence of good representations, how can training converge to those representations when we are optimizing over a very high-dimensional parameter space [14]. This still remains unsolved and ties into the following remark.

Since 2017, there has been reason to doubt that expressibility and efficiency are the primary reasons that deep neural networks outperform shallow neural networks. In [10], researchers show that shallow convolutional networks can approach and exceed the performance of deep convolutional networks if the shallow networks are trained carefully via distillation. The results of their experiments are shown in Figure 10. Thus, it suggests that learnability of complex functions plays a critical role in differentiating shallow and deep networks, and a more precise argument may be that, under the same learning algorithm, shallow networks underperform deep networks, but this analysis will critically rely on the learning algorithm at hand.

5.2 Conclusion

Lin, Tegmark, and Rolnick leverage insight from physics and the phenomena underlying the generative processes in nature to offer an explanation for why neural networks can model the functions we, as practitioners, care about. From their results [21], the core reasons that deep neural networks perform well can be summarized as:

- The probability distributions common in physics, as so in nature, are simple – typically of the form $p(x) = \frac{1}{N(x)}e^{h(x)}$, where h is a low-degree polynomial.
- Neural networks (with any activation) excel at modeling polynomials.
- Most generative processes are hierarchical and compositional – and we can learn information-maintaining inverses of these processes.
- Deep models are more expressive than shallow models via the no-flattening theorems.

Yet, there are still many questions on the success of deep neural networks, and a full theory remains elusive. With a better understanding of why deep neural networks work well, we can improve the models that currently exist. I look forward to future work that continues questioning why neural networks work and perhaps with some insight into how our cognition works as well.

References

[1] Chai Discovery. `https://www.chaidiscovery.com/`, 2024.

[2] ChatGPT: Get answers. Find inspiration. Be more productive. `https://openai.com/chatgpt/overview`, 2024.

[3] Customer Case Study: Building an end-to-end Speech Recognition model in PyTorch with AssemblyAI. `https://www.comet.com/site/customers/assemblyai/`, 2024.

[4] Ising model. `https://en.wikipedia.org/wiki/Ising_model`, 2024. Wikipedia contributors.

[5] Nuro. `https://nuro.ai`, 2024.

[6] OpenAI. `https://openai.com`, 2024.

[7] Waymo. `https://waymo.com`, 2024.

[8] Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., Esesn, B. C. V., Awwal, A. A. S., and Asari, V. K. The history began from alexnet: A comprehensive survey on deep learning approaches, 2018.

[9] B. D. Hammel. The Ising model. `https://www.bdhammel.com/ising-model/`, 2017.

[10] Ba, L. J., and Caruana, R. Do deep nets really need to be deep?, 2014.

[11] Clark, K., Khandelwal, U., Levy, O., and Manning, C. D. What does bert look at? an analysis of bert's attention. In *BlackBoxNLP@ACL* (2019).

[12] Duda, J. Waymo now providing 24/7 curbside service at Sky Harbor. *Axios Phoenix* (August 2024).

[13] Funahashi, K.-I. On the approximate realization of continuous mappings by neural networks. *Neural Networks 2*, 3 (1989), 183–192.

[14] Hanin, B. Deep Learning Theory (ORF 543). Princeton University, 2024. Accessed: 2024-06-01.

[15] He, J., Chadha, C., Kushwaha, S., Koric, S., Abueidda, D., and Jasiuk, I. Deep energy method in topology optimization applications. *Acta Mechanica 234* (12 2022), 1–15.

[16] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

[17] House, P. AlphaGo, Lee Sedol, and the Reassuring Future of Humans and Machines. *The New Yorker* (March 2016).

[18] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization, 2017.

[19] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Computation 1*, 4 (12 1989), 541–551.

[20] Lin, H. W., and Tegmark, M. Criticality in formal languages and statistical physics, 2017.

[21] Lin, H. W., Tegmark, M., and Rolnick, D. Why does deep and cheap learning work so well? *Journal of Statistical Physics 168*, 6 (July 2017), 1223–1247.

[22] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. Microsoft coco: Common objects in context, 2015.

[23] Mahendran, A., and Vedaldi, A. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision 120* (12 2016).

[24] Rush, S., Huang, A., Subramanian, S., Sun, J., Almubarak, K., and Biderman, S. The Annotated Transformer. `http://nlp.seas.harvard.edu/annotated-transformer/`, 2022. Accessed: 2024-06-01.

[25] Tegmark, M. Connections between physics and deep learning. `https://www.youtube.com/watch?v=5MdSE-N0bxs`, 2016. MIT CBMM Summer Course 2016.

[26] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2017), NIPS'17, Curran Associates Inc., p. 6000–6010.