How Convolutional Neural Networks Defy the Curse of Dimensionality: Deep Learning Explained

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY-SA 4.0

SUBMISSION DATE / POSTED DATE

13-01-2022 / 19-01-2022

CITATION

Abend, Kenneth (2022): How Convolutional Neural Networks Defy the Curse of Dimensionality: Deep Learning Explained. TechRxiv. Preprint. <https://doi.org/10.36227/techrxiv.18316439.v1>

DOI

[10.36227/techrxiv.18316439.v1](https://doi.org/10.36227/techrxiv.18316439.v1)

How Convolutional Neural Networks Defy the Curse of Dimensionality: Deep Learning Explained

Kenneth Abend, *Life Sr. Member, IEEE*

Abstract---We all know that the number of samples needed to estimate parameters grows with the number of parameters to be estimated. In a deep neural network, there is an enormously large number of neurons and hence a far larger number of weights for the synapses between neurons. How can a neural network ever learn that many weights? It is shown here, that in a convolutional neural network, the number of independent weights can be orders of magnitude smaller than the number of neurons. The working of such a network is clearly explained. In 2021, The Franklin Institute presented its prestigious Bower Award and Prize for Achievement in Science for a theme championed by the author, “Neural Networks for Machine Learning.”

Index Terms—convolutional network, deep learning, machine learning, neural networks, pattern recognition

I. THE CONVOLUTIONAL NEURAL NETWORK

THE theory behind the accuracy of pattern recognizers has a long history¹⁻⁴. For a given number of features, the performance improves as you increase the number of learning samples (the data from which the machine learns). Performance increases, and then plateaus. The required number of samples to reach the plateau grows very rapidly (e.g., exponentially) with the number of features (the dimensionality of the pattern vector) and/or with the number of parameters or weights that need to be estimated. It takes a lot of points (sample vectors) to fill a high-dimensional space.

The problem is that to increase performance we need more features and/or weights, and hence much more data samples. Consider a layered, non-recurrent, neural network such as shown in Figure 1. For pattern classification, each output activation might represent the likelihood that the network’s input is a member of a specific class. For a more general problem, using reinforcement learning, each output activation would represent the quality of a particular action. Since more features provide better performance, the number of inputs to each layer should be as large as can be supported by the number of data samples available. This would imply a very large number of neurons and hence a much larger number of weights; an enormous number, if the network is deep.

FIG. 1. CONVOLUTIONAL NEURAL NETWORK: EACH HIDDEN UNIT APPLIES THE SAME LOCALIZED, LINEAR FILTER TO ITS INPUT. WEIGHTS ARE LOCAL AND SHARED. FEW ARE INDEPENDENT.

We know from Statistics, that a large number of independent parameters requires an enormously large number of data samples. No problem. Design a network with many weights, but few independent weights. Thus, we can have our cake and eat it. We will show that the convolutional neural network does this because the weights are not independent. But first, let me briefly remind you (in one paragraph) how a deep neural network optimizes its performance. (See the free online book by Michael Nielson: www.NeuralNetworksandDeepLearning.com for a more complete treatment)

An artificial neuron (Threshold Logic Unit, Perceptron, or ADALINE) fires, activates, or recognizes something when a weighted sum of its input features exceeds a threshold. For a neural network, we replace thresholds with a differentiable non-linearity. For pattern recognition with a squared error loss, $\frac{1}{2}(y-a)^2$, we note that the derivative of the loss is the error. If we measure the output errors and differentiate the loss with respect to the weights, the chain rule of differential calculus allows us to back-propagate the errors and hence expose the hidden layers. Thus, least mean squares (LMS)⁵ and back-propagation fit like hand and glove. Section II serves as an Appendix. It provides mathematical details and generalizes to more general cost functions and to reinforcement learning.

Now return to Figure 1. Considering only the heavy lines, it represents a simple example of the first three layers of a multi-

layer or deep neural network. Ignore the light lines. They are only here to remind you that the output layer should be fully connected so that final decisions or actions depend on the network's entire input. For example, all pixels in a sample picture to be described or all acoustic time samples of a sample sentence to be understood.

Let m be the dimensionality of the input. For example, it could be 3 times the number of pixels in a color picture; or it could be the number of acoustic time samples in a spoken sentence. Let $m_1 = m$, and let $m_2 = k_2 m$ be the number of neurons in the 2nd layer (the first hidden layer). Let k be the number of inputs to each neuron in the second layer, $k \ll m$. Each unit in a convolutional hidden layer is looking for local features by applying a localized linear filter to the input. If we start by examining a 4x4 or 4x5 greyscale local regions, k is 16 or 20. The diagram here is one dimensional so the inputs may be time samples of an acoustic speech waveform. In figure 1, $k_2 = 1$, $m = m_1 = m_2 = 4$ and $k = 3$. Since, in reality, $m \gg k$, we ignore edge effects and assume k is constant. For an arbitrary numerical example, let us assume the input is m dimensional with $m = 100,000$. If the first two layers were fully connected there would be $m_1 m_2 = k_2 m^2$ or k_2 times 1 billion weights. The use of only k local weights brings this number down to $k m_2 = k k_2 m$ (or k_2 times 2 million weights).

Now here comes the kicker. If we are looking for only one feature (a vertical edge, for example) each hidden unit in the second layer would use the same linear filter and there would be only k weights that need to be determined. Of course, we do not choose the feature; the network does. But obviously we must look for several different features. Actually, k_2 of them. With k_2 sets of k weights, the total number of distinct weights that need to be determined from the learning samples is only $k_2 k$. For simplicity, we choose $k_2 = k$. If $k = 20$, by using widely shared local weights, we have increased the number of neurons in the first two layers from $2m = 200,000$ to $(k_2 + 1)m = 2,100,000$ while decreasing the number of weights that need to be learned, from $m^2 = 10,000,000,000$ to $k^2 = 400$.

That is the savings in the first convolutional layer. The pooling layer reduces the number of neurons in the layer by a factor k' so that $m_3 = m_2 / k'$, where k' can be a multiple of k_2 . In the figure k_2 is 1 instead of k , and $k' = 2$. The most economically efficient pooling is max pooling. Simply choose the largest of k' activations in the previous layer. For example, we might select the best feature amongst the k_2 candidates. However, the feature (e.g., a vertical edge) is not needed at each and every pixel. Similarly, the phoneme is not needed at each and every acoustic time sample. Layer 3 decimates the sample rate by a factor of $k' / k_2 < k$. In our example, the reduction factor, k' , must be less than $k k_2 = k^2 = 400$. The numbers used here are extreme to make a point.

Note that even if we continue to train a deep convolutional network until all the training samples are correctly classified, we are not likely to arrive at the best possible set of first layer weights or the best set of initial features for generalizing to new data. Apparently, this does not matter too much. We arrive at an excellent set of last layer weights and a solution that can often generalize to unseen data. When at a saddle point, with many dimensions to choose from, we are almost certain to find a fast way down. We do not reach a global minimum, but with

a large deep network, there are many excellent local minima for the expected cost.

One problem is that a neural network identified a rabbit as a wolf because there was snow on the ground and the network had never seen a wolf without snow or a rabbit with snow. Like a person, the machine has to keep learning.

In a typical deep network, there can be several convolution layers between each pooling layer. The differentiable activation function that replaces the threshold is no longer "S" shaped, but is the Rectified Linear Unit (ReLU), $a = f(z) = \max(0, z)$. Its derivative (for back-propagation) is always either 0 or 1, reducing computational complexity, and it is made for order for reinforcement learning. Since we choose the highest quality action, we can ignore negative scores ($z < 0$) and even eliminate neurons that usually produce zero activation. It is interesting to note that Dr. Fukushima^{6,7} was using the ReLU long before it became popular.

In April 2021, Dr. Kunihiro Fukushima of Japan received the \$250,000 Bower Award and Prize for Achievement in Science from the Franklin Institute "for his pioneering research that applied principles of neuroscience to engineering through his invention of the first deep convolutional neural network, "Neocognitron"—a key contribution to the development of artificial intelligence."

Figure 2 summarizes (approximately, and to the best of my limited knowledge) the development of Deep Convolutional Neural Networks. Dr. Fukushima first developed the Cognitron in 1974, and then made it shift and distortion invariant with the Neocognitron^{6,7} in 1979. His Neocognitron, like the brain, has local weights built in, and he also used weight replication. He may have also created shared weights artificially, by using shifted versions of all his training samples.

	Year	Back Propagation	Local Weights	Competitive Pooling	Shared Weights	LSTM (Feedback)
Fukushima	1979		Y	Y	Y	
Werbos	1982	Y				
LeCun	1989	Y	Y	Y	Y	
Hochreiter & Schmidhuber	1991-1997					Y

FIG. 2. COMPONENTS OF DEEP CNN LEARNING

Paul Werbos proposed the backpropagation of errors to hidden layers in his Harvard Ph.D. dissertation in 1974. Unfortunately, no one took notice until he actually utilized an efficient 1970 backpropagation algorithm from Finland⁸ in a Neural Network in 1982. Backpropagation became popular, but was not used with a convolutional architecture until Alex Waible used it in a 3-layer network⁹ in 1987 and Dr. LeCun did so in a deeper CNN¹⁰ in 1989. In September 2018, Dr. Yan LeCun received the Harold Pender Award from the University of Pennsylvania and in March 2019, Drs. Yan LeCun, Yoshua Bengio, and Geoffrey Hinton jointly received the Turing Award from the Association for Computing Machinery (ACM).

Performance is enhanced by adding feedback (recursion) every few layers¹¹. Since this allows data to remain in the system longer, it was named LSTM for Long Short-Term Memory. Deep Learning, made possible and practical by back-

propagation and the convolutional architecture, starts from raw data and progresses to increasing levels of abstraction. The human mind also does this, though not necessarily going through the same steps. For example, for print reading or written language translation, the layers we may go through are: pixels → edges → strokes → characters → words → meaning.

With LSTM, word recognition can depend on not only the current letters, but also on the previous word.

In 2003, Andrew Ng combined reinforcement learning concepts with neural networks and aerodynamics to design an autonomous helicopter in his PhD Dissertation at U. CA, Berkeley, and he built one at Stanford U. in 2006. The Neural Network, Alpha Go, had sparse reinforcement rewards (only at game's end). It learned to become the world champion in both Chess and Go by playing millions of games against itself.

II. AN OVERVIEW OF DEEP LEARNING ALGORITHMS

Learning can be generalized to other cost functions, and then, by use of reinforcement learning, to more general problems. In error correction for pattern recognition, we wish to minimize the expected loss due to errors. So, we define a desired output and a scalar cost or loss for an error for a given input vector, $\mathbf{x}$. The decision, and hence the cost (given the true state), depends only on the output activations.

$$C_{\mathbf{x}} = C(\mathbf{a}^L).$$

The derivative of this function with respect to the unknown parameters is what we backpropagate and call the "error". We define the cost for errors by averaging over a set of training samples

$$C = \frac{1}{n} \sum C_{\mathbf{x}}.$$

This is the quantity to be minimized by steepest descent.

For Reinforcement Learning, we define desired actions and a reinforcement reward for a desired action given the state of nature. The action depends on the output activations, and the reward depends on the action and the state of nature, where the state may depend on the previous actions. The state of nature is not given. This is learning without a teacher, and is a compound decision problem¹². We only see the past rewards. We define Return as the accumulated reward and we attempt to maximize the expected Return. We chose the action which maximizes the quality of the action, $Q(\mathbf{a}^L)$, where Q is the average value of discounted future returns. C is replaced by $-Q$.

Fig. 3. Notation for a Non-recurrent Neural Network

Let $\mathbf{v}$ be the vector of all parameters (the weights and biases of all the neurons). Then

$$\Delta v_i = -\eta \frac{\partial C}{\partial v_i} \quad (\text{or } \eta \frac{\partial Q}{\partial v_i})$$

for gradient descent. Using the notation of figure 3, backpropagation becomes linear algebra. We measure the output errors or compute them from the output activations and backpropagate the error vector. For **each** neuron in the l^{th} layer:

$$\text{Input } (l = 1): \quad a^l = x \text{ or } f(x)$$

Feedforward:

$$\mathbf{z}^l = (\mathbf{w}^l)^t \mathbf{a}^{l-1} + \mathbf{b}^l, \quad a^l = f(\mathbf{z}^l)$$

$$\text{Error: } \delta^l \equiv \frac{\partial C}{\partial z^l}$$

Output error ($l = L$):

$$\delta^L = \frac{\partial C}{\partial z^L} = \frac{\partial C}{\partial a^L} f'(\mathbf{z}^L)$$

Back-propagate error:

$$\delta^l = [(\mathbf{w}^{l+1})^t \delta^{l+1}] f'(\mathbf{z}^l)$$

$$\text{Output } \frac{\partial C}{\partial v} : \quad \frac{\partial C}{\partial b} = \delta$$

$$(\text{bias correction} = -\eta \delta)$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \quad (\text{weight}$$

$$\text{correction} = -\eta a_{in} \delta_{out})$$

The fact that the parameter correction is a small positive constant (the learning rate, η) times the divergence of quality with respect to the parameters,

$$\Delta \mathbf{v} = \eta \nabla_{\mathbf{v}} Q$$

is intuitively satisfying. We chose the action with the highest estimated quality and then increase or decrease parameters in a way to increase the estimate of the quality of the chosen action. This works because whenever we receive a reward it increases the estimate of the quality of the decisions made to get that reward. It turns out that the problem of weights and/or activations continuing to grow until they saturate (or the problem of having too many weights driven to zero) appears to be helped by LSTM. In fact, such automatic regularization was the reason it was introduced.

REFERENCES

1. T.J. Harley, L.N. Kanal, and K. Abend, "A note on the expected error in the probability of misclassification," *Proc. IEEE*, vol. 53, no. 11, pp.1764-1765, Nov. 1965.
2. K. Abend, T. J. Harley, B. D. Fritchman, and C. Gumacos: "On optimum receivers for channels having memory" (Corresp.). *IEEE Trans. Inf. Theory*, vol. 14, no.6, pp. 819-820, Nov. 1968
3. K. Abend, T. J. Harley, Jr., B. Chandrasekaran, and G. F. Hughes, "Comments on the mean accuracy of statistical pattern recognition", *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 420-423, May 1969.
4. L. Kanal and B. Chandrasekaran, "On dimensionality and sample size in statistical pattern classification," *Pattern Recognition*, vol. 3, pp. 225-234, 1971.
5. B. Widrow, "Adaptive sampled data system – A mathematical theory of adaptation," IRE WESCOM Conv. Rec., Part 4, pp. 74-85, 1959
6. K. Fukushima, "Neocognitron: A Self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybernetics*, vol.36, no.153 pp. ,193-202, 1980.
7. K. Fukushima, "Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition," *Neural Networks*, vol. 1, pp. 119-130, 1988
8. J. Schmidhuber, "Deep learning in neural networks: An overview". *Neural Networks*, vol. 61, pp. 85-117, 2015. [More](#).

9. Alex Waibel et al, "Phoneme recognition using time-delay neural networks", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 3, pp. 328.-339 March 1989.
10. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, vol.521, pp. 436-444, May 2015.
11. S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural Computation*, vol. 9, no. 8, pp 1735-1780, 1997.
12. K. Abend, "Compound decision procedures for unknown distributions and dependent states of nature," in *Pattern Recognition*, L. Kanal. ed. Washington, D.C., USA: Thompson Book Co., 1968, pp.207-249,

Links

Neural Networks for Machine Learning Symposium brochure:
Speaker biographies and Abstracts, April 2021

<https://tinyurl.com/chjuwjum>

Neural Networks for Machine Learning Symposium
Recording, April 28, 2021

<https://www.youtube.com/watch?v=GSP1QUmPgUQ>

Franklin Institute Class of 2021 Awards
www.fi.edu/awards (search "all laureates")

Kenneth Abend (Life Sr. Member) received his B.E.E. degree (cum laude) from the City College of New York and his M.S.E. and Ph.D. degrees from the University of Pennsylvania.

He was a senior signal processing research engineer at Philco-Ford, a Principal Engineer at RCA Missile and Surface Radar Systems, Director of advanced signal processing at Gorca, Vice President and Chief Scientist at VueSonix Sensors. He was also Instructor at Penn State Great Valley Graduate Center, and Adjunct Associate Professor of Systems Engineering at the University of Pennsylvania. He is retired, living in Florida and New Jersey, and serving on the Committee on Science and the Arts (CSA) of the Franklin Institute in Philadelphia, PA.

Dr. Abend received the "Outstanding Young Engineer" award at the 50th Anniversary of CCNY's School of Engineering in 1969, was Organizer, Chairman, and plenary speaker for the Spectral Estimation session of the 1980 IEEE Communication Theory Workshop, participated in the 1989 Defense Science Board Summer Study on Noncooperative Identification, and was elected to membership in the CSA in 1999. His research has centered on digital and statistical signal processing, pattern recognition, and distributed array imaging.

Insert in greyscale