Probability Amplitude Based Neural Networks

elem6

Abstract

We propose a new type of neural networks inspired by the path integral formulation of quantum mechanics. Current models of neural networks use a linear function as the link between neurons in different layers, and the non-linearity is provided by the activation function. We show that it is possible to replace the linear link between two neurons with a non-linear complex function called probability amplitude in quantum mechanics. Empirically, fully connected probability amplitudes based neural networks generalize comparably with other types of fully connected neural networks.

1 Introduction

The capability of universal approximation [3, 4, 5, 6] is one of the key reasons that deep neural networks [1] perform so well. The commonly used connections between neurons in different layers are as follows. A given computing unit (neuron) in a hidden layer takes as input the affine transformation of the outputs of neurons in the lower layer, and outputs the value of its activation function f of the form

$$x^l_\alpha = f\left(\sum_\beta w^l_{\alpha\beta} x^{l-1}_\beta + b^l_\alpha\right), \tag{1}$$

where x^l_α is the output of α-th neuron in l-th layer. $\mathbf{w}^l$ and $\mathbf{b}^l$ are weights and biases, respectively. To distinguish neuron indices from layer indices and the imaginary unit i, we use Greek letters to denote neurons. There are several popular choices for the activation function f, such as the logistic sigmoid (σ), hyperbolic tangent function (tanh) [7], and rectified linear unit (ReLU) [8, 9].

Beside affine functions, a neural network can also use radial basis functions [10, 11] as connections between neurons in different layers. In this type of networks, each hidden node acts as a centroid with coordinate $\mathbf{c}$, and its activation function is a function of the distance (not necessarily the Euclidean metric) between the input $\mathbf{x}$ and $\mathbf{c}$, namely,

$$x^l_\alpha = \lambda_\alpha \phi\left(\frac{\|\mathbf{x}^{l-1} - \mathbf{c}_\alpha\|}{\sigma_\alpha}\right). \tag{2}$$

Usually ϕ is taken to be a Gaussian function.

Mathematically, the aforementioned networks are all universal approximators. They can be trained to construct functions that fit a training set arbitrarily well. This is true even for randomly labeled training set simply because having enough parameters, the network can memorize all the training data [12]. However, different networks have different generalizations when used as classifiers on the test set. Furthermore, when applying backpropagation to train the networks, different activation functions perform differently since backpropagation relies heavily on the properties of the gradient of the activation functions [13]. Due to the above reasons, finding new types of connections between neurons [20, 21, 22, 23, 24, 25, 26, 27, 28] is an active research area.

In this work, we propose a type of neural networks based on probability amplitude (henceforth PaNet) which is inspired by the path integral formulation [2] of quantum mechanics. In the following, we give a detailed description of how to construct the network, and show some of its applications. To limit the scope, we will only consider fully connected networks.

2 Structure of PaNet

A PaNet is similar to other neural networks. It consists of an input layer, an output layer, and one or more hidden layers. The difference lies in how neurons in different layers being connected. More generally, a PaNet is a layered directed graph. Associated with each edge $\beta \to \alpha$ of the graph, we define a complex link function

$$\Gamma^l_{\alpha\beta} = \exp\left\{ i(w^l_{\alpha\beta} x^{l-1}_\beta + b^l_{\alpha\beta}) \right\}. \tag{3}$$

As before, l is the layer index, x^{l-1}_β is the output from vertex β. $w^l_{\alpha\beta}$ and $b^l_{\alpha\beta}$ are the scale and shift factors that can be determined by a learning algorithm, which are similar to the weights and biases of a traditional neural networks. A vertex (neuron) of the graph is the basic computing unit which takes as arguments the incoming links, and computes a real or complex value according to its activation function:

$$x^l_\alpha = f\left(\sum_\beta \Gamma^l_{\alpha\beta} \right).$$

To limit the scope, we only consider the following types of activation function,

$$x^l_\alpha = f\left(\left\| \frac{1}{N} \sum_\beta \Gamma^l_{\alpha\beta} \right\|^2 \right), \tag{4}$$

where we introduced a normalization constant N. In Figure 1, a neuron on the right is shown to be linked to three other neurons on the left with links Γ_β's, where $\beta = 1, 2, 3$. Its activation is the squared modulus of the normalized sum of the links.

The complex function $\Gamma^l_{\alpha\beta}$ is called probability amplitude in quantum mechanics. It is so called because in the context of path integral formulation, the

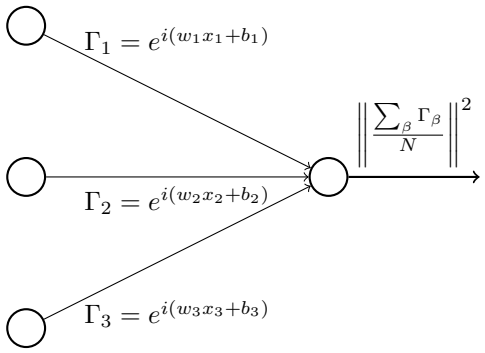


Figure 1: A simple neural network based on probability amplitude.

squared modulus $\|\Gamma^l_{\alpha\beta}\|^2$ is proportional to the probability distribution of a physical observable such as position or momentum. For example, we can imagine a particle moves through a three-layer network starting from vertex β in the first layer, the probability of finding it at vertex α in the third layer is

$$P_{\alpha\beta} \propto \left\| \sum_\gamma \Gamma^3_{\alpha\gamma} \Gamma^2_{\gamma\beta} \right\|^2 .$$

We will, however, not use this sum over paths formulation. We will stick to the activation function in Eq. 4 where computation is done layer by layer so that the standard backpropagation can be applied to train the network.

3 Properties

Classification, an important application of neural networks, is essentially a process of reducing redundant information. For example, the MNIST [14] data set consists a set of images of handwritten digit and their labels. The images have a size of 784 (28×28) pixels. The goal is to classify the test images to digits. A neural network for classifying the data set can be thought as consisting 10 distinctive functions, each is constructed to recognize one of the digits. Suppose we normalize the images so that the value of each pixel lies in $[0, 1]$, then these functions are mappings from D-dimensional unit cube $[0, 1]^D$ to $[0, 1]$, where $D = 784$. Knowing the location and intensity of a particular pixel in an image will not provide much information about what the digit is. This implies that the image contains a lot of redundant information. The redundancy comes at least from: 1) large portion of the pixels come from the uniform background of the images; 2) for a given digit, the handwritten images varies both in the number of bright pixels and the locations/intensities of these pixels. It is the local correlations among the pixels at certain length scales that determine which digit they represent. For example, instead of full connections between layers,

convolutional neural networks [15] use local receptive fields to extract features and result in much better accuracy.

Thus, to be effective, functions constructed by training a network 1) must not change for certain variations of the input so as to remove redundancy, and 2) should change when some combinations of pixels are present so as to detect correlations. We can visualize how affine function $y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$ achieves these two things by noting that, for a fixed $y = y_0$, the equation $\mathbf{w}^T(\mathbf{x} - \mathbf{x}_0) = 0$ defines a plane S in $\mathbb{R}^D$, where $\mathbf{w}^T\mathbf{x}_0 = y_0 - b$. For all $\mathbf{x} \in S$, the value y of the affine function does not change. On the other hand, if $\mathbf{x}$ grows along the direction of $\mathbf{w}$, the value y will achieve the maximum change since $\mathbf{w}$ is the gradient of the affine function. Thus the gradients are related to correlations.

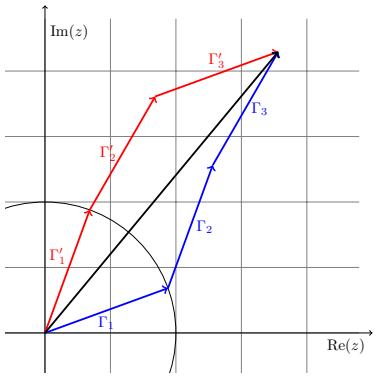


Figure 2: Sum of probability amplitudes. Two different sets of probability amplitudes may have the same sum.

Now let us look at how PaNet achieve these two things. Consider the sum of probability amplitudes

$$z = \sum_{\alpha=1}^{D} \Gamma_\alpha(x_\alpha), \tag{5}$$

where $\Gamma_\alpha(x_\alpha) = e^{i\varphi_\alpha(x_\alpha)}$ and $\varphi_\alpha(x_\alpha) = w_\alpha x_\alpha + b_\alpha$ as defined in Eq. 3 but with output neuron indices and layer indices suppressed. In the complex plane, each Γ_α is a unit vector. When all the unit vectors point to the same direction, the modulus of the sum is maximized, with $\sup \|z\| = D$. If all the vectors are evenly spaced on the unit circle, the sum reaches zero, namely, $\inf \|z\| = 0$. In between these two bounds, the more the vectors align, the larger the modulus becomes. As shown in Fig. 2, there are many arrangements for the vectors that have the same sum so that the modulus is invariant for these inputs. In the parlance of physics, these probability amplitudes can either interfere constructively or destructively. For example, the squared modulus of the sum of two probability amplitudes is

$$\|\Gamma_1 + \Gamma_2\|^2 = 2(1 + \cos(\varphi_1 - \varphi_2)).$$

When Γ_1 and Γ_2 are in phase ($\varphi_1 - \varphi_2 = 0$), the squared modulus is 2; when they are out of phase ($\varphi_1 - \varphi_2 = \pi$), it is 0.

As for the activation function in Eq. 4, the simplest function one may use is $\|z/D\|^2$. Explicitly, the activation is

$$\begin{aligned}\rho(\mathbf{x}) &= \left\| \sum_{\alpha=1}^{D} \frac{\Gamma_\alpha(x_\alpha)}{D} \right\|^2 \\ &= \frac{2}{D^2} \sum_{\alpha<\beta} \cos(w_\alpha x_\alpha + b_\alpha - w_\beta x_\beta - b_\beta) + \frac{1}{D}. \end{aligned} \tag{6}$$

Note that $\rho(\mathbf{x})$ is a sum of functions of two variables, its value depends on the pairwise phase differences of the probability amplitudes.

One interesting property of PaNet is that in general it requires at least two hidden layers to work. This is related to the problem of representing or approximating a continuous function $f : [0,1]^D \to [0,1]$ by superposition of functions with a small number of variables [16, 17, 18]. Using probability amplitudes, we can approximate the function with

$$f(\mathbf{x}) = \left\| \frac{1}{N} \sum_{\beta=1}^{N} \Gamma_\beta \left(\rho_\beta(\mathbf{x}) \right) \right\|^2, \tag{7}$$

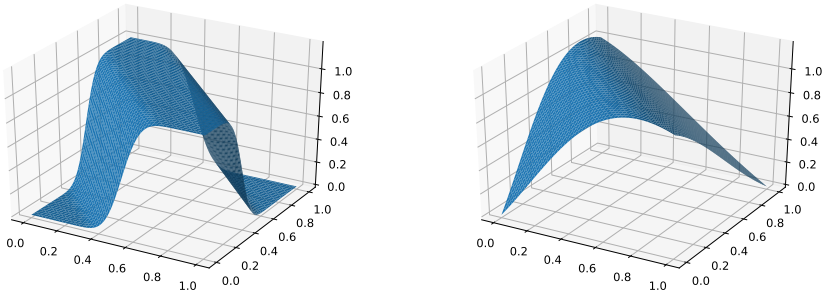
where $\rho_\beta(\mathbf{x})$ is defined in Eq. 6, and N is an integer we can choose to achieve the accuracy we need. Intuitively, as pointed out by Arnold [17], we can understand the two-layer structure by the concept of the tree of components of level sets of the function f. A level set of a function is the set of points where the function takes a fixed value. The level set may consists one or several connected components. And the level sets can be thought as organized in a tree structure. This is similar to what we have described above that a neuron must be trained to be insensitive to certain sets of inputs, and sensitive to other sets of inputs. The two-layer superposition works as follows. The inner functions map the domain of the function f to the tree of components of the level sets, the outer functions map the components to the segments that form the range of the function. It is worth stressing that affine function based neural networks work in the same way, the function can be approximated by [3]

$$f(\mathbf{x}) = \sum_{\beta=1}^{N} c_\beta \sigma \left(\mathbf{w}_\beta^T \mathbf{x} + b_\beta \right), \tag{8}$$

where the affine function $\mathbf{w}_\beta^T \mathbf{x} + b_\beta$ plays the role of $\rho_\beta(\mathbf{x})$.

4 Experiments

The simplest yet nontrivial problem is probably the XOR (exclusive-or) function, where we require $f(1,0) = f(0,1) = 1, f(0,0) = f(1,1) = 0$. We use the four



(a) (b)

Figure 3: XOR function approximated by networks trained using (a) probability amplitudes and (b) affine functions with hyperbolic tangent activation function.

data points to train both PaNet and the fully connected neural network with tanh activation function (FCN), the results are shown in Fig. 3. Without surprise both networks can satisfy the requirements quite well. Beyond these four points, they have very interesting differences. XOR function approximated by PaNet features large flat regions near these four points, and has sharper transitions between high plateaus and low plains. The function approximated by FCN has more gradual changes between different regions. Given their different behaviors, one may find one of them is more suitable for certain problems than the other, depending on the problem domain. It will be interesting to see how PaNet performs in function interpolation problems.

To see how PaNet generalizes, we used MNIST data set to test its capability. To have a better result, we introduce the function

$$\tilde{\rho}(x_1, x_2) = 2 \left\| \frac{\Gamma_1(x_1) + \Gamma_2(x_2)}{2} \right\|^2 - 1, \tag{9}$$

so that when the two vectors are in phase, $\tilde{\rho} = 1$, and when they are out of phase, $\tilde{\rho} = -1$. In general, for D-dimensional space, we can define

$$\begin{aligned} \tilde{\rho}(\mathbf{x}) &= 2 \left\| \frac{\sum_{\alpha=1}^{D} \Gamma_\alpha(x_\alpha)}{D} \right\|^2 - 1 \\ &= \frac{4}{D^2} \sum_{\alpha<\beta} \cos(w_\alpha x_\alpha + b_\alpha - w_\beta x_\beta - b_\beta) + \frac{2}{D} - 1. \end{aligned} \tag{10}$$

It can be verified numerically that the network performs better and converges faster when using $\tilde{\rho}$ as the output of the neuron than using ρ defined in Eq. 6. Note that the gradient for a given parameter is a sum of $D-1$ terms, this might help avoiding the vanishing gradient problem. Furthermore, since $\mathbf{x}$ is confined in $[0,1]^D$ and the derivatives with respect to the parameters is always finite, the gradient will not explode either.

Table 1: Results for test accuracy in percentage using different networks with different sizes for the MNIST data set.

Network	Number of Neurons in the Hidden Layer(s)								
	200	300	400	500	600	700	800	900	1000
ReLU	98.27	98.38	98.44	98.53	98.49	98.56	98.48	98.55	98.60
tanh	98.15	98.31	98.25	98.34	98.25	98.30	98.34	98.24	98.26
PaNet	98.17	98.20	98.31	98.35	98.43	98.36	98.46	98.50	98.50

As discussed previously, PaNet with two hidden layers is mathematically equivalent to affine function based networks with one hidden layer. In fact, PaNet with one hidden layer will not be able to converge on the training set. To determine its performance, we compared PaNet with two other networks. The PaNet consists two hidden layers, and with mean squared error as the cost function. The latter two networks have one hidden layer with ReLU and tanh as activation function, respectively. We also applied softmax activation function to the output layer, and used cross entropy as the cost function. For each type and size of the networks, several runs were performed. The best results for test accuracy are shown in Table 1. Within these results, PaNet is comparable with tanh network when the number of neurons in the hidden layer is small, and is comparable with ReLU network when the number of neurons in the hidden layer increases. Loosely speaking, its generalization capability is somewhat between tanh network and ReLU network.

5 Conclusion

In this paper, we proposed a new type of neural networks inspired by the path integral formulation of quantum mechanics where neurons are connected by probability amplitudes. Instead of a superposition of functions of one variable to approximate a function of multiple variables, PaNet uses essentially a superposition of functions of two variables in a pairwise fashion. From a limited first impression, one disadvantage is that it needs more parameters and layers to achieve an equivalent generalization in the case of fully connected networks. However, we would like to stress that we only considered some very narrow use cases, its full potential is still awaiting further explorations. For example, it will be interesting to adapt PaNet to convolutional neural networks or recurrent neural networks [19]. Its unique way of linking neurons may lead to novel structures in neural networks, and its mathematical structure may shed light on how neural networks generalize.

References

[1] Y. LeCun, Y. Bengio, G.E. Hinton. Deep learning. In *Nature*, 521, pages 436–444, 2015.

[2] R.P. Feynman. Space-time approach to non-relativistic quantum mechanics. In *Rev. Mod. Phys.*, vol. 20, pages 368-387, 1948.

[3] G. Cybenko. Approximations by superpositions of a sigmoidal function. In *Mathematics of Control, Signals and Systems*, vol. 2, pages 303–314, 1989.

[4] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. In *Neural Networks* **2**, pages 210–215, 1989.

[5] M. Leshno, V.Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. In *Neural Networks*, vol. 6, pages 861–867, 1993.

[6] S. Sonoda and N. Murata. Neural network with unbounded activation functions is universal approximator. In *Applied and Computational Harmonic Analysis*, vol. 43, pages 233-268, 2017.

[7] Y. LeCun, L. Bottou, G.B. Orr, and K. Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–50, 2002.

[8] V. Nair and G.E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[9] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proc. 14th International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[10] D.S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. In *Complex Systems* **2**, pages 321–355, 1988.

[11] J. Park and I.W. Sandberg. Universal approximation using radial-basis-function networks. In *Neural Computation* **3**, pages 246–257, 1991.

[12] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR* 2017.

[13] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proc. AISTATS*, vol 9, pages 249–256, 2010.

[14] Y. LeCun, C. Cortes, and C. Burges. The mnist database of handwritten digits, 1998.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, vol 86, pages 2278 – 2324, 1998.

[16] A.N. Kolmogorov. On the representation of continuous functions of many variables by superpositions of continuous functions of one variable and addition. In *Doklay Akademii Nauk USSR*, 114, pages 953 – 956, 1957.

[17] V.I. Arnold. On the representation of functions of several variables as a superposition of functions of a smaller number of variables. In *Mat. Prosveshchenie* **3**, 41 – 61, (1958); *Vladimir I. Arnold - Collected Works*, vol 1, pages 25 – 46, 2009.

[18] R. Hecht-Nielsen. Kolmogorov's mapping neural network existence theorem. In *Proceedings of the IEEE First International Conference on Neural Networks*, vol. III, pages 11–13, 1978.

[19] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. In *Nature*, 323, pages 533–536, 1986.

[20] A.L. Maas, A.Y. Hannun, and A.Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing*, 2013.

[21] K. He, X. Zhang, S. Ren, and J. Sun Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[22] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. In *arXiv preprint*, arXiv:1505.00853, 2015.

[23] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). In *ICLR*, 2016.

[24] S. Sabour, N. Frosst, and G.E. Hinton. Dynamic routing between capsules. In *NIPS*, 2017.

[25] M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.

[26] T. Minemoto, T. Isokawa, H. Nishimura, and N. Matsui. Feed forward neural network with random quaternionic neurons. In *Signal Processing*, vol. 136, pages 59–68, 2017.

[27] C.J. Gaudet, A.S. Maida. Deep quaternion networks In *arXiv preprint*, arXiv:1712.004604, 2017.

[28] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, C. J. Pal. Deep complex networks. In *ICLR*, 2018.