Improved Polynomial Neural Networks with Normalised Activations

1st Mohit Goyal[†]
Electrical and Computer Engineering Dept.
University of Illinois, Urbana Champaign
mohit@illinois.edu

2nd Rajan Goyal
Electrical Engineering Dept.
Indian Institute of Technology, Delhi
rajangoyal910@gmail.com

3rd Brejesh Lall
Electrical Engineering Dept.
Indian Institute of Technology, Delhi
brejesh@ee.iitd.ac.in

***Abstract*—Polynomials, which are widely used to study non-linear systems, have been shown to be extremely useful in analyzing neural networks (NNs). However, the existing methods for training neural networks with polynomial activation functions (PAFs), called as PNNs, are applicable for shallow networks and give a stable performance with quadratic PAFs only. This is due to the optimization issues encountered during training PNNs. We propose a working model for PAFs using a novel normalizing transformation which alleviates the problem of training PNNs with arbitrary degree. Our PAF can be directly used to train shallow PNNs in practice for degrees as high as ten. It can also be utilized to learn multivariate sparse polynomials of small degrees. We also propose a way to train deep CNNs with PAFs which achieve performance similar to deep CNNs with standard activations. Through rigorous experimentation on multiple data sets, we show that PNNs can be effectively trained in practice. This also highlights the potential of the proposed method to support the research on using polynomials to study deep learning.**

***Index Terms*—PAF, Polynomial Approximation, PNN, Activation functions**

The code is available at https://github.com/mohit1997/PAF.

I. Introduction

Deep learning has gained huge popularity in past few years. A big part of its success can be attributed to the idea of learning intermediate representations using non-linear functions generally referred to as activation functions. However, the theoretical analysis of deep neural networks is far too complex majorly because of the nature of activation functions. Polynomials, on the other hand, have widely been used for analyzing non-linear systems [1] [2]. Livni et al. [3] reviewed some of the issues with sigmoid, ReLU and threshold activations and then showed how neural networks (NNs) with polynomial activations or PNNs circumvent these problems.

PNNs have a unique property that they can be completely represented as a polynomial in input features. This property allows for quantifying their expressiveness and computational complexity [3]. Recently, Kileel et al. [4] analyzed deep PNNs from an algebraic point of view and presented upper as well as lower bounds on network width to span the entire polynomial functional space. PNNs can also be used to approximate feed forward neural networks with sigmoid activation [3]. However, most of the proposed methods favour quadratic polynomial activations [3] [5] [6] [7] and their performance is only evaluated on shallow networks.

Not surprisingly, efficient techniques to train PNNs with arbitrary degree of polynomial activations have not yet been successfully designed. This is partly because PNNs are not universal function approximators [8] [9]. Since PNNs with fixed depth have limited approximation capability, they are expected to be outperformed by DNNs with standard activations and are therefore less popular among practitioners. Secondly, polynomials are not ideally suitable for optimization with first order optimization methods. Unlike ReLU and sigmoid, their gradient and the output are unbounded and can be arbitrarily large. This results in unstable gradients, hence hindering the training of PNNs. Therefore, in this work, we aim to design PNNs that can be efficiently trained in a fashion similar to DNNs with conventional activations while achieving similar performance.

There are several ways of implementing PNNs, Livni et al. [3] used a quadratic function as activation, Fan et al. [6] proposed polynomial activations with the use of inner product between features and Adoni et al. [10] used a quadratic approximation of hyperbolic tangent activation to learn sparse polynomials. In comparison, we propose a polynomial activation using the concept of univariate function approximation. Unlike other implementations, our activation doesn't enforce a particular degree of polynomial but an upper bound on its degree allowing flexibility. Therefore, theoretically, our activation can adapt itself to any degree less than or equal to the degree specified by the user. Since a vanilla polynomial activation would suffer from gradient explosion, we mitigate this issue through normalisation which correct the mean and variance our activation. Using the properties of PNNs such as its relationship with polynomial regression, we test the suboptimality of our approach. We also do a rigorous analysis to quantify the effects of various degrees of polynomial activations. With extensive experimentation on various data sets, we show that fully connected neural networks (FCNNs) and deep convolutional neural networks (CNNs) can be successfully trained with our proposed activation.

Our main contributions can be summarised as follows:

1) We empirically show that NNs with our polynomial activation and proposed normalisation, achieve close to

[†]Work carried out when the author was a student at IIT Delhi.

optimal loss (globally minimum) on small networks with first order optimization methods. This is because, for small network depth, PNNs with our activation are immune to gradient instability (inherent in polynomial activations) for degrees as high as ten.

2) We also show that using our polynomial activation, one can achieve results similar to those obtained using conventional activations on deep CNNs. Since ReLU [11] is by far the most popular activation, we compare our results with only ReLU activation using the same network architecture in our experiments.
3) NNs with our activations are capable of learning multi-variate sparse polynomials of small degrees with significantly better generalization error than ReLU and hyperbolic tan activation.
4) Our activation can be used along with other conventional activations in a NN. We show that by replacing ReLU activation of the first hidden layer with our activation, better generalization can be obtained.

II. Model Setup

As discussed earlier, there exist several promising methods for implementing PNNs; however, most of these works are for quadratic polynomial activations and shallow networks. These methods can not simply be extrapolated to deep NNs due to optimization difficulties. As a result, determining the capability of PNNs in comparison DNNs becomes much harder. One way to quantify their capacity is to approximate activation functions which are generally used in practice using polynomials. Figure 1 shows polynomial approximations for ReLU activation on a bounded domain learned with gradient descent. We observe that, even for a non-differentiable function, a polynomial of degree four gives small approximation error. This means that replacing standard activations with polynomials of similar orders (as shown in figure 1) can guarantee comparable performance. On a broader note, using Stone Weierstrass theorem, it is possible to show that a neural network using Lipschitz continuous activation functions can be approximated with a PNN to any desired error as a function of PAF's degree.

The goal of this study is to design trainable PNNs that achieve performance similar to standard activation functions in NNs. Deriving motivation from above paragraph, we will start off with the concepts of univariate polynomial approximation to implement PNNs. Later, we will rectify the issues with the above approach by normalizing the input to activation and then showcase the potential of polynomial activations.

A. Preliminaries

The objective of univariate function approximation is to find an approximation of a function $\hat{f}(x)$ using a pre-defined basis by taking its projection on that basis. For a polynomial basis of *degree* k i.e. $\{1, x, \ldots, x^k\}$, we are interested in learning the approximation for $\hat{f}(x)$ denoted as $f(x)$ given by,

$$f(x) = a_0 + a_1 x + \ldots + a_k x^k \tag{1}$$

where $\{a_0, a_1, \ldots, a_{N-1}\}$ are learnable parameters.

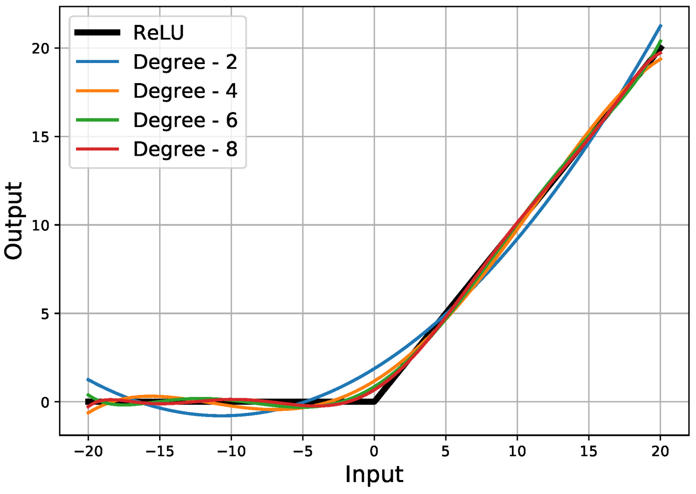

Fig. 1. Approximation of ReLU activation with various degree of polynomial.

Since the optimal activation or the true activation is not known a priori to training the network, the polynomial approximation can not be performed directly. Fortunately, we are not interested in the resulting activation but the final prediction of our NN. Therefore, the only difference between univariate function approximation and training neural networks with polynomial activation functions is that we learn the coefficients by minimizing the objective function. We call this proposed activation as Polynomial Activation Function (PAF) and we will denote NNs with polynomial activations as PNNs.

Now, we state some of the basic mathematical properties of PAFs and PNNs which form the backbone of this study and would allow for better understanding of PAFs.

Consider a vector $\mathbf{x} = [x_1, ..., x_n] \in \mathbb{R}^n$. Let $\mathcal{B}_k(\mathbf{x})$ denote the polynomial basis of degree k constructed using the elements of vector $\mathbf{x}$. Then,

$$\mathcal{B}_k(\mathbf{x}) = \left\{ \prod_{i=0}^{i=n} x_i^{\alpha_i} \;\middle|\; \sum_{i=1}^{i=n} \alpha_i = j,\ j \in \{0, 1, ..., k\} \right\}$$

Theorem 1: The cardinality $|\mathcal{B}_k(\mathbf{x})|$ of the polynomial basis $\mathcal{B}_k(\mathbf{x})$ with degree k constructed using elements of $\mathbf{x}$ is equal to $\binom{k+n}{n}$, where n is the dimensionality of $\mathbf{x}$.

Let $\mathbf{x_k}$ denote the $|\mathcal{B}_k(\mathbf{x})|$ dimensional vector containing the elements of set $\mathcal{B}_k(\mathbf{x})$. Now, we show that any PNN can be represented as a polynomial in input features.

Theorem 2: Consider a PNN with H hidden layers with input as $\mathbf{x}$ and output is a m dimensional vector $\mathbf{y}$. Suppose, the activation at the final layer is linear and all the hidden layers are activated with PAF of degree k_i, where i is the index of the hidden layer. Then, the output of the PNN can be reparametrized as

$$\mathbf{y} = \mathbf{W}\mathbf{x_k} \tag{2}$$

where, $k = \prod_{i=1}^{H} k_i$ (referred as degree of PNN), $\mathbf{W}$ is a matrix of size $m \times |\mathcal{B}_k(\mathbf{x})|$ denoting the new parameters (coefficients for the polynomial basis).

Note: Due to space constraints, proofs for the theorems are omitted and can be found in [12].

Theorem 2 shows the relationship between PNNs and polynomial regression over the original input features. It also shows that the degree of the polynomial functional space spanned by the PNN grows exponentially with its depth. The impact of PNN's width ([4]) is however more complicated, for small width the search space of PNN can be much smaller than of the above reparametrisation. For shallow PNNs with small degrees, our analysis ignore the impact of width.

For classification tasks, the output of the neural network is converted to probabilities using softmax or sigmoid activation. Hence, the above result holds only for the final layer prior to activation (i.e before softmax or sigmoid) which denote the unscaled probabilities. In conclusion, the resulting reparametrized form is equivalent to a linear classifier and linear regression on the polynomial basis $\mathcal{B}_k(\mathbf{x})$ constructed from input $\mathbf{x}$ for classification and regression tasks respectively.

The equivalence between PNN and polynomial regression, however, is not absolute especially when looking at their optimization aspect. Similar to any regression task, global optima for polynomial regression can be obtained using linear least squares regression (LSR). However, matrix inversion takes polynomial time in the number of features. Since, these features grow exponentially with the degree of polynomial (theorem 1), LSR becomes impractical. On the other hand, while PNN has to optimized through first order methods such as gradient descent which result in local optima, this is more useful in almost all practical cases because the polynomial features are implicitly learned and all the basis functions need not be stored in the memory.

B. Scaling PAF

Polynomial basis does not necessitate any restriction on the range of input x; however as we will show in experiments, PNNs with higher degree PAFs are difficult to optimize with first order methods such as Stochastic Gradient Descent (SGD) [13]. This can be attributed to the nature of gradients corresponding to polynomial basis functions. PAF $f(x)$ of degree k and the corresponding gradients can be written as follows,

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_k x^k \tag{3}$$

$$\nabla_x f(x) = a_1 + 2a_2 x + \ldots + (k)a_k x^{k-1}$$
$$\nabla_{a_i} f(x) = x^i. \tag{4}$$

The proportionality of gradient to exponents of x can lead to problems of exploding and vanishing gradients as the range of input changes. The optimization becomes more prone to this effect as the depth or the degree of PNN (defined earlier) increases. To counter this issue, we propose mean variance normalization for each basis function prior to weighted addition.

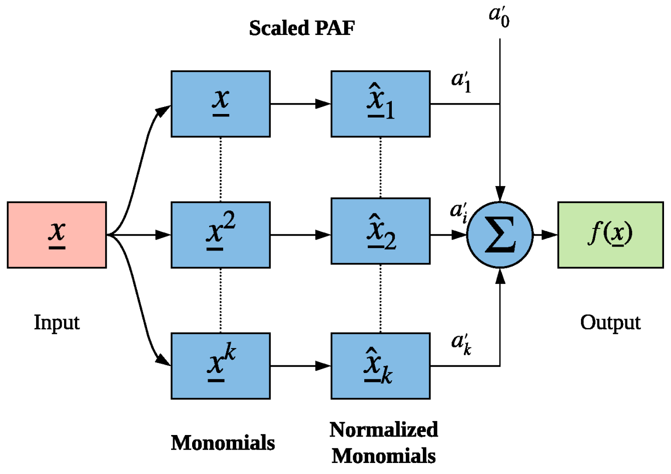

Fig. 2. Overview of scaled PAF for one input feature using polynomial activation of degree k. The monomials computed from the input vector $\underline{x}$ undergo mean variance normalisation followed by weighted combination.

Let $\underline{x}$ be a vector representing the training mini batch for one input feature. The transformed basis functions $\hat{\underline{x}}_i$ are then used in PAF as shown below:

$$\hat{\underline{x}}_i = \frac{\underline{x}^i - mean(\underline{x}^i)}{\sqrt{var(\underline{x}^i) + \epsilon}}, \quad i \in \{0, 1, 2, \ldots, k\} \tag{5}$$

$$f(\underline{x}) = a'_0 + a'_1 \hat{\underline{x}}_1 + \ldots + a'_k \hat{\underline{x}}_k \tag{6}$$

where mean and variance are computed over the training data set. The coefficients (a'_is) can serve as means for recovering original mean and variance resulting in information preservation (eq. 6). Specifically, the coefficients (a'_is) can be modified so as to satisfy the following condition,

$$f(\underline{x}) = a'_0 + \Sigma_{i=1}^{k} a'_i \hat{\underline{x}}_i = a_0 + \Sigma_{i=1}^{k} a_i \underline{x}^i \tag{7}$$

$$\text{when, } a_0 = a'_0 - \sum_{i=1}^{k} a'_i \frac{mean(\underline{x}^i)}{\sqrt{var(\underline{x}^i) + \epsilon}} \tag{8}$$

$$\text{and } a_i = \frac{a'_i}{\sqrt{var(\underline{x}^i) + \epsilon}}, \quad \forall i \in \{0, 1, 2, \ldots, k\} \tag{9}$$

Hence, this transformation corrects the inappropriate range of basis functions through normalisation which has been shown to help in faster convergence of NNs ([14]). Note that since the transformation is linear, the functional space spanned does not change and the same properties are applicable to PNNs with the above activation. The implementation details for FCNNs and CNNs are described below.

Implementation Details

In case of FCNNs, we share the activation coefficients across all features in each hidden layer. This is done so as to reduce the number of learnable parameters and prevent overfitting. The mean and variance are calculated individually for each feature across the mini batch dimension. Hence, for a hidden layer with a PAF of degree k, $k + 1$ number of additional parameters need to be trained.

In case of CNNs, $\underline{x}$ would denote a three dimensional tensor (mini batch size$\times$length$\times$width) representing one input channel rather than vector for one input feature. The mean and variance is then calculated over the entire three dimensional tensor and the activation coefficients are shared across this tensor. Also, all the channels have separate activation coefficients. Hence, the number of parameters depend on the input size. For a input tensor with C channels, PAF of degree k would introduce $C \cdot (k+1)$ number of additional trainable parameters.

Note that averaging over mini batch (3D tensor in case of CNNs) is not viable during test time. Hence, the exponentially averaged means pop_mean and variances pop_var calculated using training data statistics are calculated and stored. For every mini batch, we update pop_mean and pop_var using batch_mean and batch_mean as,

$$\text{pop\_mean} = \text{batch\_mean} * (1-\beta) + \text{pop\_mean} * (\beta)$$
$$\text{pop\_var} = \text{batch\_var} * (1-\beta) + \text{pop\_var} * (\beta)$$

where β ($\beta = 0.99$ by default) is the decay parameter. These are then later fed to eq. 5 for normalization at test time. The technique is similar to the one used in batch normalization [15]. During training, k has to be set by the user and is therefore a hyperparameter. We will provide the favourable values for this hyperparameter in experiment section.

Through out the rest of the paper, this variant of polynomial activation will be called as scaled PAF, and the original activation without this normalisation as unscaled PAF. If not specified, PAF would denote the scaled variant.

III. Experiments and Results

In this section, we will conduct empirical evaluation of neural networks using PAFs. We also provide a benchmark by training the same neural network with ReLU activation. First, we will discuss the results for specifically designed ablation studies using the key properties of PNNs. This study is done only for shallow PNNs while ignoring the impact of PNN width. Using the insights from these experiments, we then shift our focus to evaluating the performance of Convolutional Neural Networks with PAFs on standard benchmark data sets such as MNIST, Fashion MNIST and CIFAR-10.

Notation used in experiments: A PNN with degree - $k_1 \times \ldots \times k_H$ denotes a NN with H hidden layers s.t. ith hidden layers uses a PAF of degree k_i.

A. Impact of Normalisation

In this experiment we showcase the benefits provided by scaling PAF over its unscaled variant. We train two neural networks using Adam Optimizer [16], each having two fully connected (FC) hidden layers of size 128 on MNIST data set. The first hidden layer is called FC1 and the second as FC2. We experiment with multiple degrees for PAF ranging from two to eight. We train for fixed 5000 iterations with a mini-batch size of 50 with learning rate scheduling across all degrees. Table I shows the mean and standard deviations of test accuracy on ten runs. Note that we do not perform hyper parameter tuning and thus cross validation is not employed in this experiment.

TABLE I
Mean and standard deviations (on 10 runs) of test accuracy of scaled and unscaled PAF of different degrees on MNIST data set. Best results are bold faced.

Degree	Scaled PAF	Unscaled PAF
2×2	$\mathbf{98.05 \pm 0.06}$	97.49 ± 0.42
3×3	$\mathbf{97.90 \pm 0.15}$	91.12 ± 3.38
4×4	$\mathbf{97.94 \pm 0.16}$	32.82 ± 18.51
5×5	$\mathbf{97.98 \pm 0.12}$	10.00 ± 0.41
6×6	$\mathbf{97.59 \pm 0.36}$	10.33 ± 1.06
7×7	$\mathbf{97.81 \pm 0.12}$	9.96 ± 0.15
8×8	$\mathbf{97.25 \pm 0.59}$	10.15 ± 0.54
9×9	$\mathbf{96.95 \pm 0.82}$	10.23 ± 0.55
10×10	$\mathbf{96.75 \pm 1.08}$	10.21 ± 0.58

We observed that the performance of unscaled PAF declines rapidly beyond degree two. On the other hand, scaled PAF shows consistent performance upto degrees as high as ten. However, at higher degrees the variance of the test accuracy increases minimally. We also observe that as PAF's degree is increased the network becomes more sensitive to learning rate. For this experiment, we used the same learning rate schedule in all cases.

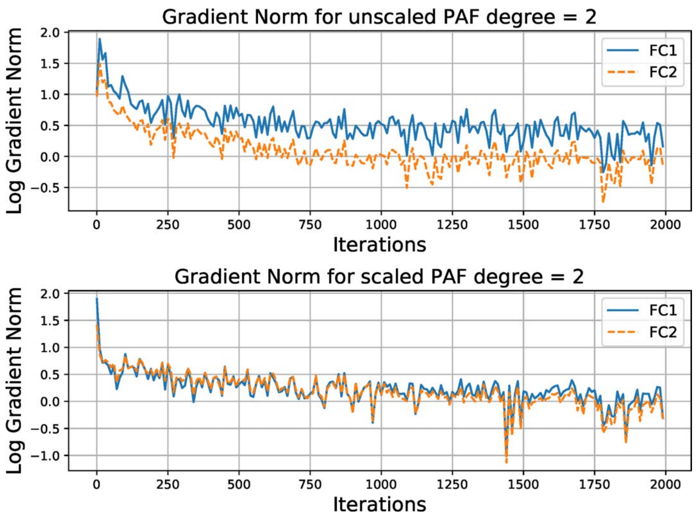

Fig. 3. Gradient Norm backpropagated to the two layers for scaled and unscaled version of PAF with degree two.

We claimed in section II-B that normalisation helps in correcting the range of monomial terms which stabilizes the gradients during training. Hence, to quantify the difference in the gradient magnitudes, we further plot the Frobenius norm of gradient backpropagated to FC1 and FC2 in first two thousand iterations for both the activations. Figures 3 and 4 show the log gradient norm with degrees two and four respectively. With degree two, the gradient norms for unscaled PAF and scaled PAF at FC2 are comparable to each other. However, at FC1, the gradients for unscaled PAF are larger than scaled PAF. This shows how the proposed normalisation takes care of the gradient explosion in deeper layers. Similarly, at degree four, the gradients for unscaled PAF are approximately 10^6 and 10^{10} times higher at FC2 and FC1 layers respectively than in the

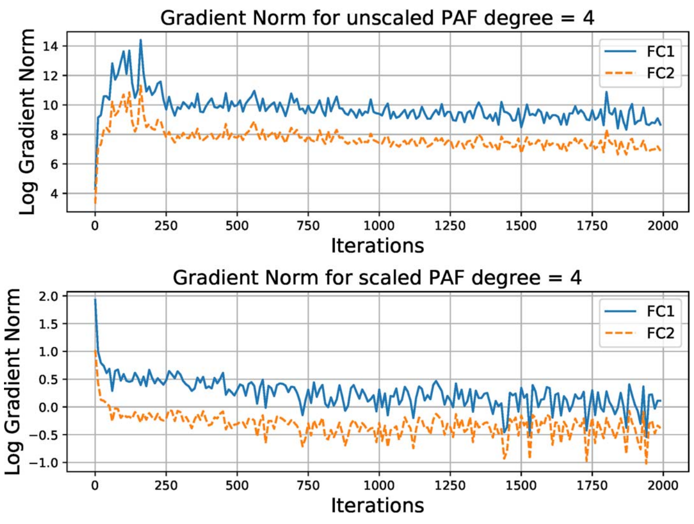

Fig. 4. Gradient Norm backpropagated to the two layers for scaled and unscaled version of PAF with degree four.

case of scaled PAF. For scaled PAF at degree four, we also observe that gradient grows as it backpropagates into shallower layers. Figure 5 shows the gradients at FC1 for various degrees of PAF, which shows that the growth is limited for a network with small depth, but it is to be expected that for deep networks such gradient explosion would be detrimental.

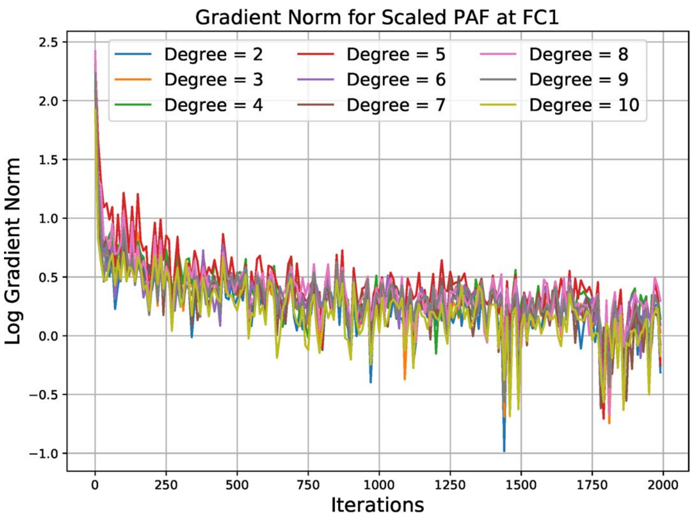

Fig. 5. Gradient Norm backpropagated to the FC1 layer for scaled PAF with various degrees.

Note: In the rest of the experiments, we apply L2 regularization penalty on PAF coefficients which showed better generalization performance. The values used for this hyperparameter are available in the training scripts provided on our github repository.

B. Optimization of PNNs and Impact of degree

Regression - Boston Housing: The objective of this experiment is two fold. (i) Since PNNs span a subset of the search space for polynomial regression on input features, we compare the performance of these two schemes using gradient descent optimization. (ii) We also compare the performance of PNN with polynomial regression for which globally optimal solution can be obtained using coordinate descent. This would indicate the suboptimality in PNN's performance.

For this regression task, we employ boston housing data set ([17]) that has thirteen input features. We evaluate several formulations with a polynomial search space of degree eight. Specifically, we test the following algorithms: (i) Standard Neural Network with two hidden layers (128 and 64 layer width) each activated with ReLU and accelerated with batch normalization *(NNRELUBN)* along with L2 regularization (ii) *(PNN)* with two hidden layers (128 and 64 layer width) activated with PAF of degree four and two respectively (iii) Linear Regression on polynomial features *(LRSGD)* with L1 regularization optimized using SGD (Adam optimizer) (iv) Lasso Linear Regression *(LLS)* on polynomial features (degree eight) optimized with coordinate descent. For a fair comparison, both NNRELUBN and PNN are trained with Adam optimizer with learning rate scheduling. Table II shows the performance of the four algorithms.

TABLE II
COMPARISION OF THE FOUR ALGORITHMS ON BOSTON HOUSING DATA SET. *NNRELUBN* WITH MODEL SIMILAR TO *PNN* PROVIDES A BASELINE FOR COMPARISON WITH OTHER METHODS. *RMSE stands for root mean squared error.*

Algorithm	Train RMSE	Test RMSE
NNRELUBN	1.32	3.78
PNN (4×2)	2.09	3.98
LLS, Degree=8, Penalty=0.01	1.59	3.06
LRSGD, Degree=8 , Penalty=0.01	22.03	22.69

The number of learn-able parameters in case of PNN ($\sim 10k$) are significantly small as compared to polynomial regression on input features ($\sim 0.3M$). Despite the fact that PNN spans only a subset of all polynomials with same degree, PNN outperforms LRSGD and moreover, gives comparable performance with respect to LLS which uses undergoes gradient free optimization. One possible reason for this can be that each local/global optima in PNN maps to more than one configuration of learnable parameters. Due to this redundancy in the network parameters, PNN tends to easily converge with SGD. This has been theoretically studied for overparameterised NNs ([18] [19]). Unlike PNN, each parameter configuration is unique in LRSGD, resulting in difficult to optimize with gradient descent.

Regression - Learning Sparse Polynomial: Now we shift our focus to the task of learning k-sparse sparse polynomials (with k monomial terms). NNs have been theoretically shown to efficiently approximate polynomials [20]. Adoni et al. ([21]) proved that irrespective of the activation function, a single (hidden) layered neural network can learn k-sparse polynomial of small degrees in finite iterations with appropriate number hidden nodes. Since PNNs are basically reparametrisation of polynomials, learning polynomials allow us to quantify the suboptimality of PNN formulation. With the degree of the true function known a priori, PNN's degree can be tuned accordingly so at bring the true polynomial into the functional space spanned by PNN. Therefore, we evaluate NNs generalization on unseen data points with different activations on the task of learning sparse polynomials.

We generate 100000 samples each from 100 gaussian distributed random variables which are input to the network. The ground truth is obtained using a randomly chosen 10-sparse polynomial of degree three or four. We employ NNs with two hidden layers (both of 512 width) for learning synthetically generated sparse polynomials. Table III shows the train and test mean squared errors (MSEs) for *PNN* (three variants with different degrees) and NNs (accelerated with batch normalisation) with ReLU and Tanh activation denoted as *NNRELUBN* and *NNTANHBN* respectively. We employ strong L1 regularization on the first layer to take into account the sparsity for both the models. In this experiment, we do not use the same activation weights/coefficients for each hidden node across a layer to avoid underfitting.

TABLE III
TEST AND TRAIN MSEs FOR NN AND PNN ON LEARNING k-SPARSE POLYNOMIAL. PNNs WITH DEGREE 2×2, 3×3 AND 4×4 HAVE BEEN EVALUATED. THE DEGREE (k) OF THE TRUE POLYNOMIAL IS MENTIONED IN THE SECOND COLUMN.

Model	k	Train MSE	Test MSE
NNTANHBN	3	0.06	2.25
	4	0.12	16.24
NNRELUBN	3	0.03	0.40
	4	0.12	12.90
PNN (2×2)	3	0.008	0.008
	4	0.012	0.011
PNN (3×3)	3	0.008	0.008
	4	0.05	0.08
PNN (4×4)	3	0.014	0.018
	4	0.07	0.10

In contrast to NNRELUBN and NNTANHBN, all three variant of PNN give approximately ten fold or even better test error. Moreover both train and test error is small which means that shallow PNNs can give almost globally optimal performance. Among the three variants, PNN (2×2) gives the smallest train and test error, followed by PNN (3×3) and PNN (4×4). Note that PNN (2×2) is sufficient to learn both the polynomials of degree three and four. However, the slight degradation in performance of PNNs with higher degrees can be attributed to two factors (i) optimization difficulties (ii) overfitting. As shown in previous experiment, gradient grows towards shallower layers of NN for higher degree PAFs. This explains larger train error for higher degree PNN. Also, PNNs with higher degrees could potentially overfit learning polynomials of smaller degrees which explains larger gap between train and test errors. Nevertheless, this experiment showcase the potential of PNNs in learning polynomials.

Classification - Two Spiral: We consider two spiral problem, a binary classification task, and compare the performance of logistic regression on polynomial features and PNNs with softmax activation at the output (denoted as PNN). Note that unlike linear regression, logistic regression doesn't have closed form solution. ([22]) provides various solvers for logistic regression but we report the performance of SGD as other solvers ([23]) didn't prove out to be beneficial for this task. Also, theorem 2 shows the multiplicative dependence of PAF's degree on PNN's degree which implies, with small increment in the number of parameters, the functional space spanned by a PNN can be significantly enlarged. Hence, we quantify improvements in the performance of PNNs on changing the PAF's degree. However, it is important to note that this incurs a computational cost which increases with PAF's degree. Note that a small NN is chosen that underfits the task so as to highlight the impact of degree. For degrees higher than four, we use gradient clipping which showed significant improvements in performance.

Table IV compares three classes of algorithms, (i) *NNRELUBN* - conventional *NN* with batch normalization and two hidden layers (20 width each) with ReLU activation , (ii) *PNN* with two hidden layers (20 width each) each with PAF activation of various degrees (2-7), and (iii) *LRSGD* - Logistic Regression on polynomial basis of degree forty nine with SGD. Both NNRELUBN and PNN are trained with Adam optimizer and learning rate scheduling was used.

TABLE IV
COMPARISION OF THE THREE DIFFERENT ALGORITHMS ON TWO SPIRAL DATA SET.

Algorithm	Train Acc (%)	Test Acc (%)
NNRELUBN	80.14	80.98
PNN (2×2)	60.84	60.12
PNN (3×3)	70.44	70.92
PNN (4×4)	85.05	86.52
PNN (5×5)	96.33	98.31
PNN (6×6)	98.04	98.88
PNN (7×7)	99.56	99.75
LRSGD (degree=49)	80.68	82.25

We observe that there is a monotonic increase in the train and test accuracy of PNN as the degree is increased. Since the only change is in PAF's degree, the increase in number of parameters from PNN ($k \times k$) to PNN ($k + 1 \times k + 1$) is just 2 parameters (one for each hidden layer). Note that PAF coefficients are shared across the hidden layer. However, the performance of LRSGD is strikingly poor as compared to PNN (7×7) given that the functional space of LRSGD is wider. This shows the effectiveness of PNNs formulation on classification tasks. Figures 6, 7 show the classification boundary learnt by PNN (7×7) and NNRELUBN respectively. We can see that due to the polynomial form of the activation function, the resulting boundary learned is smooth and therefore provides good generalization (extending both the spirals will decrease classification accuracy in the case of NNRELUBN). On the other hand, the boundary learned by NN with ReLU activation displays sharp turns and irregular decision boundary.

C. CNNs with PAFs

In this experiment, we evaluate the performance of CNNs with PAFs (denoted as *PNN*) having three or more hidden layers and also compare their performance with CNNs having

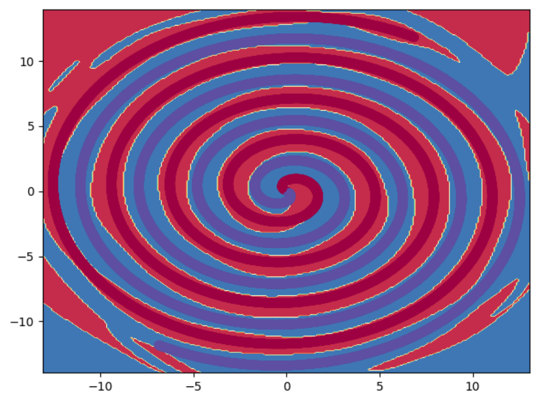

Fig. 6. PNN (7×7) classification map on two-spiral data set

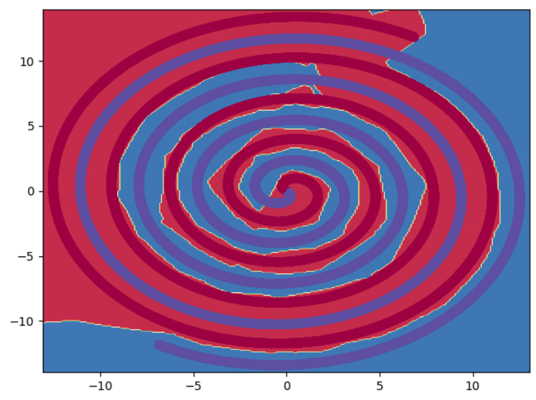

Fig. 7. NNRELUBN classification map on two-spiral data set

the same architecture, batch normalisation and ReLU activation (*NNRELUBN*). As shown in previous experiments, higher degree PAFs have huge variance due to gradient explosion in deeper layers as the gradient is back propagated. Therefore, we report the performance of CNNs with PAFs of degree two. We also compare a hybrid CNN architecture (*Hybrid*) that has the same architecture as NNRELUBN but uses PAF of degree two only at the first hidden layer instead of ReLU. In all the experiments we regularise the PAFs coefficients. Specific architecture details are available on our GitHub repository.

We consider three image classification data sets which are popularly utilised as a benchmark for performance for DNNs.

1) **MNIST**: This is standard hand digit image classification data set containing 70,000 gray scale images. We experiment withk a custom CNN having two convolutional layers with maxpooling followed by two fully connected layers, the latter one being a standard softmax layer.
2) **Fashion MNIST:** This is a data set containing 70,000 gray scale images of clothing items which have resolution similar to MNIST data set. We use a small residual network with two convolution layers and then two residual blocks followed by three fully connected layers with last one being a softmax layer.

TABLE V
TEST ERRORS ON MNIST DATA SET FOR THREE MODELS. BEST RESULTS ARE BOLD FACED. **Degree of the PAF in first hidden layer.*

Model	Degree	Test Error (%)
NNRELUBN	-	0.62
PNN	$2 \times 2 \times 2$	0.53
Hybrid	2*	**0.50**

TABLE VI
TEST ERRORS ON FASHION MNIST DATA SET FOR THREE MODELS. BEST RESULTS ARE BOLD FACED. **Degree of the PAF in first hidden layer.*

Model	Degree	Test Error (%)
NNRELUBN	-	6.44
PNN	2^8	7.03
Hybrid	2*	**6.35**

3) **CIFAR-10:** This is another image classification data set consisting of 60,000 color images split into ten classes. We use ResNet [24] architecture with 32 layers and 44 layers on this data set.

TABLE VII
TEST ERROR ON CIFAR-10 USING RESNET ARCHITECTURE FOR FOUR MODELS ALONG WITH NUMBER OF LEARNABLE PARAMETERS IN MILLIONS. BEST RESULTS ARE BOLD FACED. **Degree of the PAF in first hidden layer.*

Model	Degree	# Params	Test Error (%)
NNRELUBN (ResNet 32)	-	0.46M	7.51
NNRELUBN (ResNet 44)	-	0.66M	7.17
PNN (ResNet 32)	2^{31}	0.46M	8.50
Hybrid (ResNet 32)	2*	0.46M	**7.12**

Discussion We compare the performance of three CNN based models with same architecture but different activations on standard benchmark data sets—MNIST, FMNIST, and Cifar10. Due to parameter sharing in the proposed model for PAF, the increment in the number of learnable parameters of PNN and Hybrid models in comparison to NNRELUBN is minimal. Note that Hybrid model uses a PAF in the first hidden layers which doesn't affect the gradients in other layers. However, since PAF coefficients can be regularised, Hybrid shows slightly better performance than NNRELUBN on all three data sets. In contrast, due to larger depth of network in Fashion MNIST and CIFAR-10 experiments, PNN model suffers from gradient explosion issues. This is why the testing error is higher for PNN. Note that a higher test error can also be due to model overfitting but, since the number of parameters is almost same, we attribute this to optimization difficulties. Also, On MNIST data set, we observe that PNN outperforms NNRELUBN with a smaller test error that is comparable to Hybrid model. Therefore, on all three data sets, it is evident from the results that using our PAF, PNNs can be trained almost identically to DNNs using conventional activation functions and perform similar to them. Although, evaluating our PAF takes polynomial time which is further increased due to normalisation step, standard DNNs are much more practical for usage. Nevertheless, our approach can be utilised by researchers to experiment with PNNs resulting in better understanding of deep neural networks.

IV. Conclusion

In this work, we presented a new form of activation function which is motivated from polynomial approximation of univariate functions. The activation weights are learned during training while searching a finite degree polynomial space in input features. Using a simple normalisation step, we showed that shallow NNs with polynomial activation functions can be trained with degrees as high as ten without any form of gradient instability. We also provided an in-depth analysis of PNNs and empirically showcased their performance over linear regression and linear classification on polynomial features. Since PNNs span a polynomial function space, we showed that PNNs can be employed to efficiently learn multivariate sparse polynomials of small degrees. In comparison to other activations, NNs with PAFs provided much better generalization on test set. We also showed that CNNs using polynomial activations can perform at par with standard CNNs. This work, therefore, establishes the effectiveness of NNs with polynomial activations.

References

[1] M. Halás, "Nonlinear systems: A polynomial approach," in *Computer Aided Systems Theory - EUROCAST 2009*, R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 595–602.

[2] A. Halme and J. Orava, "Generalized polynomial operators for nonlinear systems analysis," *IEEE Transactions on Automatic Control*, vol. 17, no. 2, pp. 226–228, April 1972.

[3] R. Livni, S. Shalev-Shwartz, and O. Shamir, "On the computational efficiency of training neural networks," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 855–863. [Online]. Available: http://papers.nips.cc/paper/5267-on-the-computational-efficiency-of-training-neural-networks.pdf

[4] J. Kileel, M. Trager, and J. Bruna, "On the expressive power of deep polynomial neural networks," *CoRR*, vol. abs/1905.12207, 2019. [Online]. Available: http://arxiv.org/abs/1905.12207

[5] M. Soltani and C. Hegde, "Towards provable learning of polynomial neural networks using low-rank matrix estimation," in *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Storkey and F. Perez-Cruz, Eds., vol. 84. Playa Blanca, Lanzarote, Canary Islands: PMLR, 09–11 Apr 2018, pp. 1417–1426. [Online]. Available: http://proceedings.mlr.press/v84/soltani18a.html

[6] F. Fan, W. Cong, and G. Wang, "A new type of neurons for machine learning," *CoRR*, vol. abs/1704.08362, 2017. [Online]. Available: http://arxiv.org/abs/1704.08362

[7] M. Soltani and C. Hegde, "Fast and provable algorithms for learning two-layer polynomial neural networks," *IEEE Transactions on Signal Processing*, vol. 67, no. 13, pp. 3361–3371, July 2019.

[8] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, no. 6, pp. 861 – 867, 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608005801315

[9] F. Fan and G. Wang, "Universal approximation with quadratic deep networks," *CoRR*, vol. abs/1808.00098, 2018. [Online]. Available: http://arxiv.org/abs/1808.00098

[10] A. Andoni, R. Panigrahy, G. Valiant, and L. Zhang, "Learning polynomials with neural networks," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML14. JMLR.org, 2014, p. II1908II1916.

[11] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, pp. 947–, 06 2000. [Online]. Available: https://doi.org/10.1038/35016072

[12] M. Goyal, R. Goyal, and B. Lall, "Learning activation functions: A new paradigm of understanding neural networks," *CoRR*, vol. abs/1906.09529, 2019. [Online]. Available: http://arxiv.org/abs/1906.09529

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[14] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*. London, UK, UK: Springer-Verlag, 1998, pp. 9–50. [Online]. Available: http://dl.acm.org/citation.cfm?id=645754.668382

[15] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 448–456.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[17] D. Harrison and D. L. Rubinfeld, "Hedonic housing prices and the demand for clean air," *Journal of Environmental Economics and Management*, vol. 5, no. 1, pp. 81 – 102, 1978. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0095069678900062

[18] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The Loss Surfaces of Multilayer Networks," in *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Lebanon and S. V. N. Vishwanathan, Eds., vol. 38. San Diego, California, USA: PMLR, 09–12 May 2015, pp. 192–204.

[19] S. Arora, N. Cohen, and E. Hazan, "On the optimization of deep networks: Implicit acceleration by overparameterization," *CoRR*, vol. abs/1802.06509, 2018. [Online]. Available: http://arxiv.org/abs/1802.06509

[20] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 930–945, May 1993.

[21] A. Andoni, R. Panigrahy, G. Valiant, and L. Zhang, "Learning polynomials with neural networks," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14. JMLR.org, 2014, pp. II–1908–II–1916.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://scikit-learn.org/

[23] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," *Advances in Neural Information Processing Systems*, vol. 2, 07 2014.

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