Polynomial Regression As an Alternative to Neural Nets

Xi Cheng
Department of Computer Science
University of California, Davis
Davis, CA 95616, USA xicheng@ucdavis.edu

Bohdan Khomtchouk
Stanford University
Stanford, CA 94305, USA
bohdan@stanford.edu

Norman Matloff
University of California, Davis
Davis, CA 95616, USA
matloff@cs.ucdavis.edu

Pete Mohanty
Stanford University
Stanford, CA 94305, USA
pmohanty@stanford.edu

June 19, 2018

Abstract

Despite the success of neural networks (NNs), there is still a concern among many over their "black box" nature. Why do they work? Here we present a simple analytic argument that NNs are in fact essentially polynomial regression models. This view will have various implications for NNs, e.g. providing an explanation for why convergence problems arise in NNs, and it gives rough guidance on avoiding overfitting. In addition, we use this phenomenon to predict and confirm a multicollinearity property of NNs not previously reported in the literature. Most importantly, given this loose correspondence, one may choose to routinely use polynomial models instead of NNs, thus avoiding some major problems of the latter, such as having to set many tuning parameters and dealing with convergence issues. We present a number of empirical results; in each case, the accuracy of the polynomial approach matches or exceeds that of NN approaches. A many-featured, open-source software package, **polyreg**, is available.

1 The Mystery of NNs

Neural networks (NNs), especially in the currently popular form of many-layered *deep learning networks* (DNNs), have become many analysts' go-to method for predictive analytics. Indeed, in the popular press, the term *artificial intelligence* has become virtually synonymous with NNs.[1]

Yet there is a feeling among many in the community that NNs are "black boxes"; just what is going on inside? Various explanations have been offered, such as [Shwartz-Ziv and Tishby(2017)], but the present paper will present significant new insights.

2 Contributions of This Paper

The contribution of the present work will be as follows:[2]

(a) We will show that there is a rough correspondence between any fitted NN and a fitted ordinary parametric polynomial regression (PR) model; **in essence, NNs are a form of PR**. We refer to this loose correspondence here as NNAEPR, Neural Nets Are Essentially Polynomial Models.

(b) We **exploit NNAEPR to learn about general properties of NNs via our knowledge of the properties of PR.** This will turn out to provide new insights into aspects such as the numbers of hidden layers and numbers of units per layer, as well as how convergence problems arise. In addition we use NNAEPR to predict and confirm a multicollinearity property of NNs not previous reported in the literature.

(c) Property (a) suggests that in many applications, one might simply fit a polynomial model in the first place, bypassing NNs. This would have the advantage of **avoiding the problems of choosing tuning parameters** (the polynomial approach has just one, the degree), nonconvergence and so on.

(d) Acting on point (c), we compare NN and polynomial models on a variety of datasets, finding **in all cases that PR gave results at least as good as, and often better than, NNs**.

[1] There are many different variants of NNs, but for the purposes of this paper, we can consider them as a group.

[2] Author listing is alphabetical by surname. XC wrote the entire core code for the **polyreg** package; NM conceived of the main ideas underlying the work, developed the informal mathematical material and wrote support code; BK assembled the brain and kidney cancer data, wrote some of the support code, and provided domain expertise guidance for genetics applications; PM wrote extensive support code, and provided specialized expertise on NNs. All authors conducted data experiments.

(e) Accordingly we have developed an open source, feature-rich software package in R (a Python version is planned), **polyreg**, that enables active implementation of the above ideas [Cheng et al.(2018)]. Details are presented in Appendix A.

Point (a) is especially important. Some researchers, e.g. [Choon et al.(2008)], have conducted empirical investigations of the possible use of polynomial regression in lieu of NNs. Some theoretical connections between NNs and polynomials have been noted in the literature, e.g. [Hornik et al.(1989)]. Furthermore, some authors have constructed networks consisting of AND/OR or OR/AND polynomials as alternatives to NNs [Shin and Ghosh(1995)].

But our contribution is to show that, in essence, conventional NNs actually *are* PR models. Our focus will be on the activation function. Using an informal mathematical analysis on that function, we show *why* NNs are essentially a form of PR. In that sense, our work is somewhat like that of [Benitez et al.(1997)], where an explicit equivalence of NNs to fuzzy rule-based systems was shown.

It must be noted here that our empirical work has not yet involved specialized networks such as convolutional NNs (CNNs), recurrent NNs (RNNs) and so on. Though we intend to adapt our ideas to these frameworks, we view them as orthogonal issues. For instance, we view the convolutional "front ends" in CNNs as largely playing the role of preprocessing stages, easily adaptable to other approaches such as polynomial models. Indeed, convolutional versions of, for instance, random forests have been developed [Zhou and Feng(2017)] [Miller et al.(2017)]. Similar points hold for RNNs, say using *structural equation models* [Monecke and Leisch(2012)] with polynomial forms. Thus, though certainly a topic for future work, we view the preprocessing issue as separate from our findings that NNs are essentially PR models, with various important implications, and that PR models perform as well as NNs.

3 Notation

Consider prediction of Y from a vector X of p features. In regression applications, Y is a continuous scalar, while in the classification case it is an indicator vector, with $Y_i = 1, Y_j = 0$ for $j \neq i$ signifying class i. Prediction entails estimation of the regression function $r(t) = E(Y|X = t)$, which in the classification case is the vector of conditional class probabilities. The function $r(t)$ must be estimated, parametrically or nonparametrically, from sample data consisting of n cases/observations.

4 Polynomial Regression Models

PRs — models that are linear in parameters but polynomial in the predictors/features — are of course as old as the linear model itself. (And they extend naturally to generalized linear models.) Though they might be introduced in coursework for the case $p = 1$, multivariate polynomial models are popular in *response surface methods* [Myers et al.(2009)].

One issue of concern is *multicollinearity*, correlations among the predictors/features [Faraway(2016)]. PR models are long known to suffer from multicollinearity at high degrees [Chatterjee and Greenwood(1990)].

Indeed, in the case $p = 1$, fitting a polynomial of degree $n-1$ will be an ephemeral "perfect fit," with $R^2 = 1$. Then any variable, predictor or response, will be an exact linear combination of the others, i.e. full multicollinearity.

For this and other reasons, e.g. large fitted values at the edges of the data, many authors recommend not using polynomial models of degree higher than 2 or 3, and in fact in our empirical experiments in this paper, we have seldom found it necessary to use a higher degree.

5 The NNAEPR Principle

Universal Approximation Theorems such as [Hornik et al.(1989)] show that, under various regularity assumptions and sufficient data, NNs can approximate the regression function $r(t)$ to any desired degree of accuracy. Here we illustrate that in an informal way, though with an additional focus on the activation function.

Take the simple case $p = 2$. Denote the features by u and v. The inputs to the first hidden layer, including from the "1" node, will then be of the form $a_{00} + a_{01}u + a_{02}v$ and $a_{03} + a_{05}u + a_{05}v$.

As a toy example, take the activation function to be $a(t) = t^2$. Then outputs of that first layer will be quadratic functions of u and v. Similarly, the second layer will produce fourth-degree polynomials and so on. In that manner, we can generate polynomials which are dense in the space of regression functions.

For more realistic activation functions, simply note that they themselves can usually be approximated by polynomials. One could make a Taylor series argument, for instance, or appeal to the the famous Stone-Weierstrass Theorem [**?**], which states that any continuous function on a compact set can be approximated uniformly by polynomials. In any event, for practical purposes here, we see that most activation functions can be approximated by a polynomial. Then apply the same argument as above.

In other words, **NNs can loosely be viewed as a form of polynomial regression**, our NNAEPR Principle introduced earlier. This is a core point in this paper, as will be seen in the succeeding sections. And while other machine learning models might be approximated by polynomials, the relation of PR to NNs is actually rather direct, as seen above. Hence our focus on NNs.

6 Lurking Multicollinearity

As noted, PR models of course have a very long history and are well-understood. We can leverage this understanding and the NNAEPR Principle to learn some general properties of NNs. In this section, we will present an example, with intriguing implications.

As mentioned, PR models tend to produce multicollinearity at higher degrees. The material on NNAEPR in the preceding section, viewing NNs as kind of a polynomial regression method, suggests that NNs suffer from multicollinearity problems as well.

Indeed, the conceptual model of the preceding section would suggest that the outputs of each layer in an NN become more collinear as one moves from layer to layer. We investigated this, using the `keras` package to fit NNs and test the multicollinearity.[3]

We used the famous **mnist** dataset, included in the package, as a testbed. Our test consisted of sequential models containing linear stacks of layers, with five layers in total. This included two dropout layers. We set the number of units to be equal in each layer.

One then needs a measure of multicollinearity. A very common one is the *variance inflation factor* (VIF) [Faraway(2016)].[4] When running a linear regression analysis (linear in the coefficients, though here polynomial in the predictors/features), a VIF value is computed for each coefficient. Larger values indicate worse degrees of multicollinearity. There are no firm rules for this, but a cutoff value of 10 is often cited as cause for concern.

6.1 Experimental Results

In our case here, we calculated two measures of overall multicollinearity in a given NN layer: the proportion of coefficients with VIF larger than 10 and the average VIF.

[3] Our **polyreg** also facilitates such investigation.

[4] Various other measures of multicollinearity have been proposed, such as *generalized variance*.

First experiment: The number of units is 10 in each layer, and the model is the following.

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                7850
_________________________________________________________________
dropout_1 (Dropout)           (None, 10)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                110
_________________________________________________________________
dropout_2 (Dropout)          (None, 10)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                110
=================================================================
```

The VIF results are shown in Table 1. On average, VIF increases as one moves on to the next layer.

Second experiment: We set the number of units to 64 in the first four layers, while the last layer still has 10 outputs. The model is the following.

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                50240
_________________________________________________________________
dropout_1 (Dropout)           (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
```

The results of the VIF values of coefficients are shown in Table 2.

Third experiment: We set the number of units to 128 in the first four layers and the last layer still has 10 outputs. The model is the following.

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
dropout_1 (Dropout)           (None, 128)               0
```

Table 1: Results of first model

Layer	Percentage of VIFs that are larger than 10	Average VIF
dense_1	0	3.43303
dropout_1	0	3.43303
dense_2	0.7	14.96195
dropout_2	0.7	14.96195
dense_3	1	1.578449×10^{13}

Table 2: Results of second model

Layer	Percentage of VIFs that are larger than 10	Average VIF
dense_1	0.015625	4.360191
dropout_1	0.015625	4.360191
dense_2	0.96875	54.39576
dropout_2	0.96875	54.39576
dense_3	1	3.316384×10^{13}

```
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
```

The results of the VIF values of coefficients are shown in Table 3.

6.2 Impact

In the above experiments, the magnitude of multicollinearity increased from layer to layer. This increasing multicollinearity may correspond to the multicollinearity warning in polynomial regression. Thus, NNs and polynomial regression appear to have the same pathology, as expected under NNAEPR.

Table 3: Results of third model

Layer	Percentage of VIFs that are larger than 10	Average VIF
dense_1	0.0078125	4.3537
dropout_1	0.0078125	4.3537
dense_2	0.9921875	46.84217
dropout_2	0.9921875	46.84217
dense_3	1	5.196113×10^{13}

In other words, **NNs can suffer from a hidden multicollinearity problem**. This in turn is likely to result in NN computation convergence problems.

We thus believe it would be helpful for NN software to include layer-by-layer checks for multicollinearity. If a layer is found to output a higher degree of multicollinearity, one might consider reducing the number of units in it, or even eliminating it entirely. Applying dropout to such layers is another possible action. One related implication is that later NN layers possibly should have fewer units than the earlier ones.

It also suggests a rationale for using *regularization* in NN contexts, i.e. shrinking estimators toward 0 [Hastie et al.(2015)] [Matloff(2017)]. The first widely-used shrinkage estimator for regression, *ridge regression*, was motivated by amelioration of multicollinearity. The above discovery of multicollinearity in NNs provides at least a partial explanation for the success of regularization in many NN applications. Again due to NNAEPR, this is true for PR models as well. We intend to add a ridge regression option to **polyreg**.

Much more empirical work is needed to explore these issues.

7 PR as Effective, Convenient Alternative to NNs

7.1 Empirical Investigations

We compared PR to NNs on a variety of datasets, both in regression and classification contexts (i.e. continuous and categorical response variables, respectively). The results presented here are complete, representing every analysis conducted by the authors, i.e. not just the "good" ones.[5] However, not all tuning parameter combinations that were run are presented; only a few typical settings are shown. Generally the settings that produced extremely poor results for NNs are not displayed.

Each table displays the results of a number of settings, with the latter term meaning a given method with a given set of tuning parameters. For each setting:

- The dataset was split into training and test sets, with the number of cases for the latter being the min(10000,number of rows in full set).
- The reported result is mean absolute prediction error in the regression case and overall proportion of correct classification in the classification case.

[5]We also started an analysis of the Missed Appointments Data on Kaggle, https://www.kaggle.com/joniarroba/noshowappointments. However, we abandoned it because no model improved in simply guessing No (appointment not missed). However, PR and NNs did equally well.

No attempt was made to clean the data, other than data errors that prevented running the code. Note that this is especially an issue with the Census and NYC taxi datasets.

The reader will recognize a number of famous datasets here, many from the UC Irvine Machine Learning Repository. There are also some "new" datasets, including: a specialized Census dataset on Silicon Valley programmer and engineer wages, curated by one of the authors; data on enrollments in Massive Open Online Courses (MOOCs); data from a Crossfit competition; and data exploring the impact of genetics on brain and kidney cancers.

Abbreviations in the tables:

- PR: Polynomial regression. Degree is given, and if not the same, maximum interaction term degree. A "PCA" designation means that dimension reduction via 90%-total-variance Principal Components Analysis was performed before generating the polynomials.
- KF: NNs through the Keras API, [Chollet et al.(2015)], accessed in turn via the R-language package **kerasformula** [Mohanty(2018)]. The default configuration is two layers with 256 and 128 units (written as "layers 256,128"), and a dropout proportion of 0.4.
- DN: NNs through the R-language package **deepnet** [Rong(2014)]. The notation is similar to that of KF.

DN can be much faster (if less flexible) than KF, and thus DN was sometimes used in the larger problems, or for comparison to KF.[6] However, their performance was similar.

7.1.1 Programmers and Engineers Census Data

This is data on programmers and engineers in Silicon Valley in the 2000 Census. There are 20090 rows and 16 columns.

First, we predict wage income, a regression context. The results are shown in Table 4. We then predict occupation (six classes), shown in Table 5. Here PR substantially outperformed NNs.

7.1.2 Million Song Data

This is a very well-known dataset, listing audio characteristics of songs, along with their year of publication. The latter is the object of prediction. There are

[6] Concerning speed, KF does have a GPU version available; DN does not.

Table 4: Prg/Eng, predict income

setting	accuracy
PR, 1	25595.63
PR, 2	24930.71
PR, 3,2	24586.75
PR, 4,2	24570.04
KF, default	27691.56
KF, layers 5,5	26804.68
KF, layers 2,2,2	27394.35
KF, layers 12,12	27744.56

Table 5: Prg/Eng, predict occ.

setting	accuracy
PR, 1	0.3740667
PR, 2	0.3845197
KF, default	0.3378
KF, layers 5,5	0.3398
KF, layers 5,5; dropout 0.1	0.3399

Table 6: Million Song, predict year

setting	accuracy
PR, 1, PCA	7.7700
PR, 2, PCA	7.5758
KF, default	8.4300
KF, layers 5,5	7.9381
KF, layers 2,2	8.1719
DN, layers 2,2	7.8809
DN, layers 3,2	7.9458
DN, layers 3,3	7.8060

Table 7: Concrete, predict strength

method	correlation (pred. vs. actual)
neuralnet	0.608
kerasformula	0.546
PR, 2	**0.869**

515345 cases, with 90 predictor variables. The results are shown in Table 6. PR was somewhat ahead of NNs in this case.

Note that the PR experiments used PCA. As mentioned, a limitation of PR is that memory/time can become an issue. Remedies, to be discussed in Section 9, include dimension reduction via PCA, which was used here.

7.1.3 Concrete Strength Data

Here one is predicting compressive strength of concrete. This dataset provides some variety in our collection, in that it is much smaller, only 1030 rows. There are eight predictors.

In Table 7, we see that PR significantly outperformed both **kerasformula** and the **neuralnet** package. This is probably to be expected in a small dataset. (Mean absolute error is not reported in this case; the displayed values are correlations between predicted and actual values, the square root of R^2.)

7.1.4 Letter Recognition Data

This is another UCI dataset, consisting of images of letters, preprocessed to record 16 geometric features. There are 20000 images. In spite of our attempts

Table 8: Letter Recognition, predict letter

setting	accuracy
PR, 1	0.7285
PR, 2	**0.9030**
KF, layers 5,5	0.4484

Table 9: NYC Taxi, predict trip time

setting	accuracy
PR, 1	**580.6935**
PR, 2	591.1805
DN, layers 5,5	592.2224
DN, layers 5,5,5	623.5437
DN, layers 2,2,2	592.0192

with various combinations of tuning parameters, the performance of NNs, both KF and DN, here was poor. See Table 8.

7.1.5 New York City Taxi Data

This is a Kaggle dataset (https://www.kaggle.com/c/nyc-taxi-trip-duration), in which we predict trip time. Results are shown in Table 9. There was perhaps a slight edge to PR over NNs.

7.1.6 Forest Cover Data

In this remote sensing study, the goal is to predict the type of ground cover, among seven classes, from 54 features. There are 581,012 observations and always guessing the mode (Class 2) would yield 49% accuracy. Table 10 shows PRs and NNs both get about 70% right, with a slight edge to NNs.

However, this example also illustrates an important limitation of PR: for degree 2, our software could not accommodate the size of the polynomial matrix generated. Section 5 outlines future work to remedy this problem.

7.1.7 MOOCs Data

This dataset on Harvard/MIT MOOCs was obtained from the Harvard Dataverse Network, http://thedata.harvard.edu. Here $n = 641138,\ p = 20$.

Table 10: Forest Cover, predict grnd. cover type

setting	accuracy
PR, 1	0.6908
PR, 2	-
KF, layers 5,5	0.7163

Table 11: MOOCs, predict certification

setting	accuracy
PR, 1	0.9871
PR, 2	0.9870
KF, layers 5,5	0.9747
KF, layers 2,2	0.9730
KF, layers 8,8; dropout 0.1	0.9712

We wished to predict whether a student will complete the course and receive certification. A major challenge of this dataset, though, is the large number of missing values. For instance, 58132 of the records have no value for the **gender** variable. The only fully intact variables were **certified**, **nforum_posts** and the course and user ID columns.

For the purpose of this paper, we simply used the intact data, adding four new variables: The first was **ncert.c**, the total number of certifications for the given course. If student A is taking course B, and the latter has many certifications, we might predict A to complete the course. Similarly, we added **ncert.u**, the number of certifications by the given user, and variables for mean number of forum posts by user and course. Altogether, we predicted **certified** from **nforum_posts** and the four added variables.

The results are shown in Table 11. Note that only about 2.7% of the course enrollments ended up certified, so one hopes for an accuracy level substantially above 0.973. PR did achieve this, but NNs did not do so.

7.1.8 Crossfit Data

In this section and the next, we present more detailed analyses.

The focus is on publicly available data from recent Crossfit annual opens, amateur athletics competitions consisting of five workouts each year. For each, we fit a neural net and polynomial linear (but not additive) models. To foreshadow, our PR package, **polyreg**, fit with a first or second degree polynomial and second

Table 12: Crossfit Open, predict Rx rank

model	accuracy	range among 5 runs
KF	0.081	0.164
PR, 1	0.070	0.027
PR, 2	0.071	0.069
PR, 3	0.299	7.08
PR, 4	87.253	3994.5

degree interactions, outperforms the NNs slightly in terms of median MAE. (Though the third degree model did poorly and the fourth degree produced wild estimates symptomatic of severe collinearity.)

Using **kerasformula**, we built a dense neural network with two five-node layers (other than the outcome) with relu activation. Kernel, bias, and activity L1-L2 regularization were employed and dropout rate was set to 40% and 30%, respectively. ADAM minimized MSE with batches of 32. Four separate models were fit, corresponding to polynomial degrees 1, 2, 3, and 4, using our PR package. All four models, fit by ordinary least squares, contain two-way interactions.

For each of four datasets (Men's 2017, Men's 2018, Women's 2017, Women's 2018) we fit 10 models, representing all distinct pairs of opens that could be used as features to predict each competitor's final rank. "Rx", as in "prescription", denotes the heaviest weights and most complex movements. In this analysis, we restrict the population to those who competed in at least one round at the "Rx" level and who reported their age, height, and weight. The final sample sizes were 118,064 (Men's 2018), 41,103 (Men's 2017), 53,958 (Women's 2018), and 13,815 (Women's 2017). The outcome is rank in the overall competition; like all other variables, it is scaled 0 to 1.

The results (Table 12) suggest that a first or second degree polynomial (with two-way interactions) is best in this case in terms of median mean absolute error (low bias). The first degree model is preferable because it has lower variance. The third and fourth degree models are not admissible.

Next, to assess sample size requirements we compare KF to the best fitting PR. For each competition, we take subsamples of 1000, 2000, ..., N_{open} (using only the first two competitions as features). Figure **??** reports median out-of-sample measures of fit. PR is uniformly lower though KF nearly converges at fairly modest sample sizes. Notably, PR's performance is all but invariant–PR performs as well $N_{subsample} = 1,000$ as it does on the full sample for all four competitions.

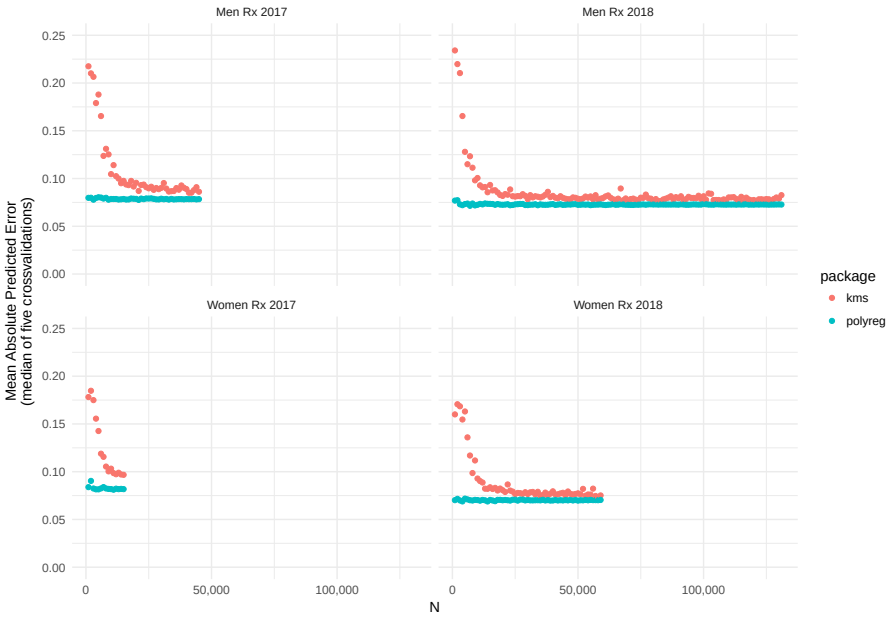

Figure 1: Predictive Accuracy by Sample Size

7.1.9 Big Data and Small Data: New Case-Studies in Cancer Genomics

We construct two cancer datasets from the NCI Genomic Data Commons (GDC) [Grossman et al.(2016)]. The first is a compendium of all known cases for the aggressive brain cancer glioblastoma multiforme (GBM), as well as lower grade glioma (LGG), with n = 129,119. The second is of kidney cancer cases, specifically of papillary renal cell carcinoma (n = 32,457). We build models that classify patients as 'alive' or 'dead' based on genetic mutation type and three variables that assess the impact of the cancer (as well as patient gender). The larger sample size notwithstanding, the brain cancer dataset is considerably more challenging since it does not contain any quantitative variables. By contrast, in addition to the impact variables, the kidney dataset contains patient age at time of diagnosis. The kidney data also includes patient ethnicity. The brain cancer data are also higher entropy (49.94% of patients are alive compared with 83.68% in the kidney cancer data). As added challenge, the working memory limitations discussed in the next section affected this analysis.

For each data set, we fit six polynomial models, which differed as to whether second-order interactions were included (in addition to first order). Models which took advantage of principal components included quadratic terms; models fit on the mostly qualitative raw data did not. We fit an NN with **deepnet** with as

Table 13: Cancer, predict vital status

model	accuracy on brain cancer data	accuracy on kidney cancer data
deepnet	0.6587	0.5387
nnet	0.6592	0.7170
PR (1, 1)	0.6525	0.8288
PR (1, 2)	0.6558	0.8265
PR (PCA, 1, 1)	0.6553	0.8271
PR (PCA, 2, 1)	0.5336	0.7589
PR (PCA, 1, 2)	0.6558	0.8270
PR (PCA, 2, 2)	0.5391	0.7840

many hidden nodes as columns in the model matrix (64 for the brain data and 40 for the kidney). We also fit an NN with **nnet** of size 10. For each design, we cross-validated, holding 20% out for testing (and we report the median of trials).

The results are encouraging for PR (Table 13). On the brain cancer data, where polynomial regression might be expected to struggle, **polyreg** performs as well out of sample as the NNs. On the kidney cancer data, **polyreg** performs noticeably better than either **deepnet** or **nnet**.

8 NNs and Overfitting

It is well-known that NNs are prone to overfitting [Chollet and Allaire(2018)], which has been the subject of much study, e.g. [**?**]. In this section, we explore the causes of this problem, especially in the context of the NNAEPR Principle. These considerations may explain why in some of the experiments reported here, PR actually outperformed NNs, rather than just matching them.

8.1 Too Many Tuning Parameters

In part, overfitting stems from the multitude of tuning parameters in typical implementations. As one tries to minimize the objective function, the larger the number of tuning parameters, the more likely that the minimizing configuration will seize upon anomalies in the training set, hence overfitting.

One popular technique to counter overfitting in neural networks is *dropout* [Srivastava et al.(2014)]. This "thins out" the network by randomly culling a certain proportion of the neurons. Note, though, that if the dropout rate is to be determined from the data, this is yet another tuning parameter, compounding

the problem.

8.2 NNAEPR and the Activation Function

Our findings here show other possible sources of overfitting. In Section 5, we showed that if the activation function is a polynomial, then NNs are exactly PR models. We then noted that almost any activation function is close to some polynomial, so that the NN fit is then essentially PR. This then implies that the higher the degree of that "nearby" polynomial, the higher the potential for overfitting.

8.3 Complex NNs As a First Resort

In addition, our results suggest that in practice, many analysts are simply attempting to fit far too large a model in the first place, with many layers, and hundreds of neurons per layer. We found that with PR, we seldom needed to go beyond degree 2, which NNAEPR would imply one should have only one or two layers, each with a small number of neurons.

This also led us to suspect that fitting large NN models often results in having most of the weights being 0 or nearly 0. We have begun to investigate this, and preliminary results — coupled with NNAEPR — suggest that configuring large networks in the initialization of NN analyses may be a poor strategy.

9 Limitations and Remedies

As mentioned, the PR method, while effective, has potential limitations in terms of memory, run time and multicollinearity. In this section, we discuss existing and future remedies.

To set the stage, let's take a closer look at the problems. As before, let n and p denote the number of cases and the number of predictors/features, respectively. Denote the degree of the polynomial by d.

First, how large will the polynomial matrix be? It will have n rows; how many columns, l_d, will there be? This can be calculated exactly, but a rough upper bound will suffice. With $d = 1$, the number of possible terms l_d is about p. What happens when we go to degree $d + 1$ from degree d? Consider one of the p variables x_i. We can form l_d new terms by multiplying each of the existing ones by x_i. So, we have $l_{d+1} \approx (p + 1)l_d$. That implies that l_d is $O(p^d)$. So the polynomial matrix can be large indeed.

The computation for the linear and generalized linear model (e.g. logistic) involves

inversion (or equivalent) of an $l_d \times l_d$ matrix. Using QR decomposition, this takes $O(n, l_d^2)$ time.

Beyond that, there is the statistical issue. The results of [Portnoy(1984)] would suggest that we should have $l_d < \sqrt{n}$. Though modern research results on the LASSO and the like are more optimistic, it is clear that we need to keep d small unless n is extremely large. This is consistent with our empirical findings here, in which we found that $d = 2$ is sufficient in most cases.

Dimension reduction via PCA remedies time, space, and multicollinearity problems and was found to work well in the case of Million Song dataset. Another possible form of dimension reduction would be to mimic the dropout "lever" in NNs. One could delete randomly-selected columns of the polynomial features matrix.[7]

Multicollinearity itself might be handled by ridge regression or the LASSO. It should be kept in mind, though, that our results indicate the multicollinearity is manifested in NNs as well, and is a possible sign of overfitting in both. See related work in [Shin and Ghosh(1995)].

Parallel computation is definitely a possibility. The software currently provides the option of parallelizing PR in the classification case. Both time and space issues might be resolved by using the Software Alchemy technique of [Matloff(2016)].

10 Conclusions and Future Work

We have presented a new way to look at NNs, as essentially a form of PR. Though some previous work had noted some other kinds of connection of NNs to polynomials, we presented a simple analytic argument involving the activation function showing the connection explicitly and in a very strong manner.

We have shown that viewing NNs as PR models reveals properties of NNs that are, to our knowledge, new to the field, and which should be useful to practitioners. For instance, our NNAEPR Principle predicted that NNs should have multicollinearity problems in later layers, which in turn suggests ways to avoid convergence problems. Among other things, this suggests some useful improvements in diagnostic features in NN software.

Most importantly, we have shown that PR should be an effective *alternative* to NNs. We performed experiments with a variety of data of various types, and in *all* cases, PR performed either similarly to, or substantially better than, NNs — without the NN troubles of trying to find good combinations of tuning parameters.

The fact that in some cases PR actually outperformed NNs reflects the fact that

[7] Our **polyreg** package does have such an option.

NNs are often overparameterized, in essence fitting a higher-degree polynomial than they should.

Much remains to be done. The problems and remedies outlined in the preceding section need to be tested and implemented; the phenomenon of multicollinearity in NNs needs thorough investigation; more experimentation with large-p datasets should be conducted; and the approach here needs to be integrated with preprocessing of images, text and so on as with, e.g., CNNs.

It is conceivable that PR may be competitive with other machine learning techniques, such as random forests, SVM and so on. We have focused on NNs here because of the direct connection to PR described in Section 5, but similar connections to other methods could be explored.

A The polyreg Package

Though we have presented the **polyreg** package [Cheng et al.(2018)] as an alternative to using NNs, it is useful in its own right.

The package provides an easier interface to perform polynomial regression for both prediction and classification, using linear regression and logistic regression respectively. It features solutions to potential problems arising with polynomial regression, such as repetition of dummy predictor variables, overfitting and multicollinearity.

A.1 Model Description

The polynomial terms in a model are generated by the **getPoly()** function. Suppose that we use a dataset containing two predictor variables. The basic linear regression model can be fitted as:

$$r(t) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Then the polynomial regression model of degree 2 for this dataset is:

$$r(t) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 x_2$$

And for degree 3:

$$r(t) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 x_2 + \beta_6 x_1^3 + \beta_7 x_2^3 + \beta_8 x_1^2 x_2 + \beta_9 x_1 x_2^2$$

A.2 Polynomial Term Generation

The **getPoly()** function generates these terms for any degree and any number of predictors p. However, if one of the predictor variables is a categorical variable, we often encode it into dummy variables. Suppose that x_1 is a categorical variable and only has two classes, which could be presented as either 0 or 1. Then we know that:

- x_1, x_1^2, x_1^3, and all powers of x_1 have the same values.
- x_1x_2, $x_1^2x_2$ and so on will all have the same values.

The **getPoly()** function generates the polynomial terms of the predictor variables in a dataset without repetition. In the small example above, the terms that are generated by **getPoly()** are:

$$x_1, x_2, x_2^2, x_1x_2, x_2^3, x_1x_2^2$$

The model fitting is handled by the **polyFit()** function, which calls **getPoly()**.

A.3 Reducing the Large Number of Terms

Clearly there is a combinatorial explosion of terms. To keep the number of terms to a manageable level, the function also provides a parameter, **maxInteractDeg**, to specify the maximum degree of dummy and nondummy predictor variable interaction terms.

Another way to reduce the number of terms is the principal components analysis (PCA) approach, specified by the argument **pcaMethod**. By default, the number of components is set so that 90% of the total variance is attained; in general, this is specified via **pcaPortion**.

Still another approach is the **dropout** parameter. Inspired by the notion of the same name in NNs, this one deletes random terms in the polynomial model.

A.4 Other Parameters

The parameter **use** specifies the model to be fitted (currently linear or logistic). In the latter case, there are further choices. If the number of classes in response variables is two, then we directly use the `glm()` function with **family = binomial(link = "logit")**.

If there are more than two classes, the original **glm()** can't generate the predicted results for these classes. We provide the parameter, **glmMethod**, to use the

all-vs-all or the *one-vs-all* method for multiclass classification [Matloff(2017)]. The multinomial logistic model may also be used.

A.5 Prediction

Using the generated model from the **polyFit()** function, we obtain a **polyFit** object, and pass in this object and new data to the generic function **predict.polyFit()** for prediction. Also, we have the **xvalPoly()** function to perform cross validation.

A.6 Comparison to NNs

To easily compare PR with the performance of neural networks, we provide functions for cross validation of a given dataset using neural networks functions from different packages.

- **xvalNnet:** This function uses the **nnet** function from **nnet** package, and provides the option to scale the data matrix.
- **xvalKf:** This function uses the **kms** function from **kerasformula** package.
- **xvalDnet:** This function uses the **nn.train** function from **deepnet** package.

References

[Benitez et al.(1997)] J. M. Benitez et al. *Are Artificial Neural Networks Black Boxes?*, volume 8. September 1997.

[Chatterjee and Greenwood(1990)] Sangit Chatterjee and Allen G. Greenwood. Note on second–order polynomial regression models. *Decision Sciences*, 21 (1):241–245, 1990.

[Cheng et al.(2018)] Xi Cheng et al. *polyreg: Polynomial Regression*, 2018. URL `https://github.com/matloff/polyreg`.

[Chollet et al.(2015)] François Chollet et al. Keras. `https://keras.io`, 2015.

[Chollet and Allaire(2018)] François Chollet and JJ Allaire. Deep learning with r. 2018.

[Choon et al.(2008)] Ong Hong Choon et al. A functional approximation comparison between neural networks and polynomial regression. *WSEAS Trans. Math.*, 7(6):353–363, June 2008. ISSN 1109-2769.

[Faraway(2016)] J.J. Faraway. *Linear Models with R, Second Edition*. Chapman & Hall/CRC Texts in Statistical Science. CRC Press, 2016. ISBN 9781439887349. URL https://books.google.com/books?id=i0DOBQAAQBAJ.

[Grossman et al.(2016)] Robert L. Grossman et al. Toward a shared vision for cancer genomic data. *New England Journal of Medicine*, 375(12):1109–1112, 2016. doi: 10.1056/NEJMp1607591. URL https://doi.org/10.1056/NEJMp1607591. PMID: 27653561.

[Hastie et al.(2015)] Trevor Hastie et al. *Statistical learning with sparsity: the lasso and generalizations*. CRC Press, 2015.

[Hornik et al.(1989)] Kurt Hornik et al. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8. URL http://www.sciencedirect.com/science/article/pii/0893608089900208.

[Matloff(2017)] N. Matloff. *Statistical Regression and Classification: From Linear Models to Machine Learning*. Chapman & Hall/CRC Texts in Statistical Science. CRC Press, 2017. ISBN 9781351645898. URL https://books.google.com/books?id=IHs2DwAAQBAJ.

[Matloff(2016)] Norman Matloff. Software alchemy: Turning complex statistical computations into embrassingly-parallel ones. *Journal of Statistical Software*, 71(1), 2016.

[Miller et al.(2017)] K. Miller et al. Forward Thinking: Building Deep Random Forests. *ArXiv e-prints*, May 2017.

[Mohanty(2018)] Pete Mohanty. *kerasformula: A High-Level R Interface for Neural Nets*, 2018. URL https://github.com/rdrr1990/kerasformula. R package version 1.0.1.

[Monecke and Leisch(2012)] Armin Monecke and Friedrich Leisch. sempls: Structural equation modeling using partial least squares. *Journal of Statistical Software, Articles*, 48(3):1–32, 2012. ISSN 1548-7660. doi: 10.18637/jss.v048.i03. URL https://www.jstatsoft.org/v048/i03.

[Myers et al.(2009)] R.H. Myers et al. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. Wiley Series in Probability and Statistics. Wiley, 2009. ISBN 9780470174463. URL https://books.google.com/books?id=89oznEFHF_MC.

[Portnoy(1984)] Stephen Portnoy. Asymptotic behavior of m-estimators of p regression parameters when p^2/n is large. *Ann. Statist.*, 12(4):1298–1309, 12 1984. doi: 10.1214/aos/1176346793. URL http://dx.doi.org/10.1214/aos/1176346793.

[Rong(2014)] Xiao Rong. *deepnet: deep learning toolkit in R*, 2014. URL `https://CRAN.R-project.org/package=deepnet`. R package version 0.2.

[Shin and Ghosh(1995)] Y. Shin and J. Ghosh. Ridge polynomial networks. *Trans. Neur. Netw.*, 6(3):610–622, May 1995.

[Shwartz-Ziv and Tishby(2017)] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *CoRR*, abs/1703.00810, 2017. URL `http://arxiv.org/abs/1703.00810`.

[Srivastava et al.(2014)] Nitish Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting. volume 15, pages 1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

[Zhou and Feng(2017)] Zhi-Hua Zhou and Ji Feng. Deep forest: Towards an alternative to deep neural networks. *CoRR*, abs/1702.08835, 2017. URL `http://arxiv.org/abs/1702.08835`.