An Exploration of the Versatility of Robust Recovery of Over-Parameterized Models

Seungkyoon Antonio Bong

A thesis presented for the degree of
Bachelor of Science in Computer Science

Department of Computer Science
Yale University, New Haven, CT 06511
United States of America

ABSTRACT

In the rich field of robust estimation, a new concept of robust recovery of corrupted models is presented in a paper by Gao and Lafferty (2020). In this paper, we explore extensions and applications of the works in that paper. First, we extend the framework of model repair to convolutional networks and find that by reformulating the convolutional network in matrix form akin to a feed-forward network, we can recover the parameters of a convolutional network with results comparable to the feed-forward case. This speaks to the generalizability of the model repair algorithm. This can have ramifications for the storage and maintenance of convolutional and feed-forward models in a standardized manner which actually may turn out to help the recoverability and maintenance of convolutional models. Additionally, we show that the results of model repair on neural networks are reproducible on colon genomics data for a variety of model types and should generalize to other data. Because model repair on over-parameterized models can nearly perfectly recover the original response variables, this result can have significant impacts on data privacy as models with even 50% of their parameters corrupted can still be exactly repaired. Finally, we prove that a simplified robust estimator without any noise, the degenerate point subset problem (c-DPS) of finding any valid fixed fraction of collinear points is NP-hard. This indicates a likely computational bottleneck for robust estimators and because of the connection to model repair, may restrict the computational tractability of the model repair problem. In each of the relevant sections, we find encouraging results and present potential future works and extensions to the research to expand the novel and growing field of model repair.

1 INTRODUCTION

Statistical models, such as regression models, are designed with underlying assumptions about the distribution and soundness of the data. When these idealized assumptions are violated, some methods, like ordinary least squares which is highly sensitive to outliers, can become compromised. This has led to a class of estimation techniques known as robust estimation which is less sensitive to departures from assumptions of the data. Traditional robust estimation attempts to deal with these outliers and corruptions to the data through methods of estimation that are immune to corruptions.

Although traditional robust estimation assumes that the data is inherently corrupted, a recent paper by Gao and Lafferty (2020) studies the case in which the data is sound but a statistical model becomes corrupted after fitting the data. This is different from the typical robust estimation problem in that instead of re-estimating the model from scratch, we can actually recover the uncorrupted version of the model directly from the corrupted model with the weight initializations, original design matrix, and some key properties. Of particular note, the paper even finds that the initial response variable is unnecessary to accurately recover the original model with high probability. As statistical models and machine learning become more widespread and powerful, we can imagine that malicious black-hat hackers may target deployed machine learning models to disrupt a company's workflow and data. Additionally with the resurgence of machine learning and as larger models with more parameters become built in the coming years, there can be faults in the storage of models that point to the importance of fault tolerant maintenance of these models.

In the paper, Gao and Lafferty (2020) find that there are two key components to accurate and high probability recovery of corrupted models. The first is that a statistical model is sufficiently over-parameterized; there must be significantly more parameters in the model than observations. Although over-parameterization is not generally desired in models as it leads to issues of interpretability and identifiability, the paper finds that it is a necessary component of successful model repair. However, it is not an unreasonable assumption in an era where the numbers of parameters in statistical models continually increase as computational ability improves even as the complexity of problems do not necessarily increase. The second requirement is that the estimator incorporates redundancy in the estimated parameters. For example, sparse estimators, even over-parameterized ones, will not generally be repairable as the corrupted parameters depend on the redundant information to recover the original parameter values.

Being a new area of robust estimation, this field is very unexplored and thus warrants further research. In this paper, we explore several different extensions to and explorations of this novel model repair framework. The first application of this thesis is to extend the model repair algorithm from standard feed-forward networks to convolutional networks. As we will see, a major roadblock in this extension depends on the invertibility on convolutional networks, which is still an active area of research, and the transformation of a convolutional network into a compatible form for robust model repair. With the transformation defined in this paper, we find that we can recover corrupted convolutional networks with similar trends observed in the feed-forward case in Gao and Lafferty (2020). The second major application is to verify the results of the model repair algorithm on real data. Although the results of the model repair algorithm in the paper were verified on toy dataset randomly generated using a seed, we would like to verify that these results are replicable on real data. The successful recovery of a model on real data can have significant impacts on privacy as we will see later. Given that the model repair problem is extremely novel, we would like to determine the computational tractability and bottleneck with these repair algorithms. The third and final major extension is thus to prove via a Turing reduction that a simpler, related problem known as the degenerate point subset problem is NP-hard. This implies that the robust regression problem is likely not solvable in polynomial-time and thus potentially signifies a computational bottleneck for the model repair algorithms presented in the paper.

2 BACKGROUND

As statistical models grow larger and more prevalent, questions about the validity and integrity of these data and models become more pertinent. While there is rich literature on model repair for corrupt data, the literature is less rich for corrupt models. Gao and Lafferty (2020) attempt to formulate and address this problem in a novel field by formulating and solving this model repair problem for a variety of models (e.g. linear models, gradient-descent estimators, random features models, and neural networks).

They formulate the general problem of model repair for a corrupted estimator as follows:

- $\hat{\theta} \in R^p$ is a model with p parameters estimated on n data points $\{x_i, y_i\}_{i=1}^n$
- The model $\hat{\theta}$ is corrupted as $\eta = \hat{\theta} + z$ where
  - $z_j \sim (1-\varepsilon)\delta_0 + \varepsilon Q_j$ and Q_j is an arbitrary distribution
  - z is independent of the design $\{x_i\}_{i=1}^n$
  - In other words, each $\hat{\theta}_j$ is corrupted by additive noise from Q_j with probability ε
- The goal is to recover $\hat{\theta}$ from η without re-estimating the model from scratch

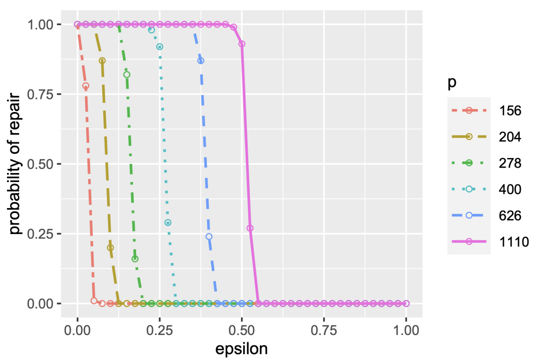
For linear models, we start with the following assumptions:

- Design matrix $X \in \mathbb{R}^{n \times p}$ and response vector $y \in \mathbb{R}^n$
- Minimization of the squared error $||y - X\theta||_2^2$
- The minimal norm solution solution: $\hat{\theta} = X^T(XX^T)^{-1}y$

Similar to above, we assume we only have a corrupted model $\eta = \hat{\theta} + z$, where $z_j \sim (1-\varepsilon)\delta_0 + \varepsilon Q_j$. It turns out that by solving the following expression with $u \in \mathbb{R}^n$:

$$\widetilde{u} = \underset{u}{\operatorname{argmin}} ||\eta - X^T u||_1$$

we can recover the original model $\hat{\theta}$ as $\widetilde{\theta} = X^T \widetilde{u}$ with high probability. A plot from the paper of the probability of exact repair as a function of the corruption probability ε is shown to the right.



For feed-forward neural networks, the paper defines and theoretically analyzes a feed-forward network with one hidden layer, but empirically demonstrates the application to two layers with the following framework:

- $f(x) = \frac{1}{\sqrt{p}} \sum_{j=1}^p \beta_j \varphi(W_j^T x)$

- φ is some activation function (ReLU and hyperbolic tangent were studied in the paper)
- Squared error loss: $L(\beta, W) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \frac{1}{\sqrt{p}}\sum_{j=1}^{p}\beta_j\varphi(W_j^T x)\right)^2$
- Trained using gradient descent algorithm or stochastic gradient descent
  - Either normally or by training the first layer, freezing it, and retraining the second layer
- The model repair is then performed layerwise similar to the linear model.

  For a corrupted neural network layer $\Theta \in \mathbb{R}^{d\times p}$ with initialization $W(0)$, we compute the following for each $j \in [p]$:

  $$\widetilde{v}_j = \underset{v_j}{\operatorname{argmin}}\,||\Theta_j - W_j(0) - X^T v_j||_1$$

  and find the recovered model as $\widetilde{W}_j = W_j(0) + X^T\widetilde{v}_j$

In general, the paper finds that to repair a corrupted model, the statistical model must (1) be over-parameterized and (2) incorporate redundancy. In fact, because of these two key properties, the response variable is not needed to recover the statistical models analyzed in this paper; the design matrix functionally encodes the response variable because of the redundancy and over-parameterization of the model.

The essence and validity of the formulation of model repair depends on estimators being represented in terms of the row space of the design matrix. This leads to a view and interpretation of model repair as a robust estimation problem. The recovery algorithms which are further analyzed and implemented in this paper thus depend on solving a median regression problem and stem from this dual formulation. In particular, it is this dual view that gives significance to the proof of NP-completeness of robust estimators in this paper and implies a potential computational bottleneck for robust estimators and the related model repair algorithms.

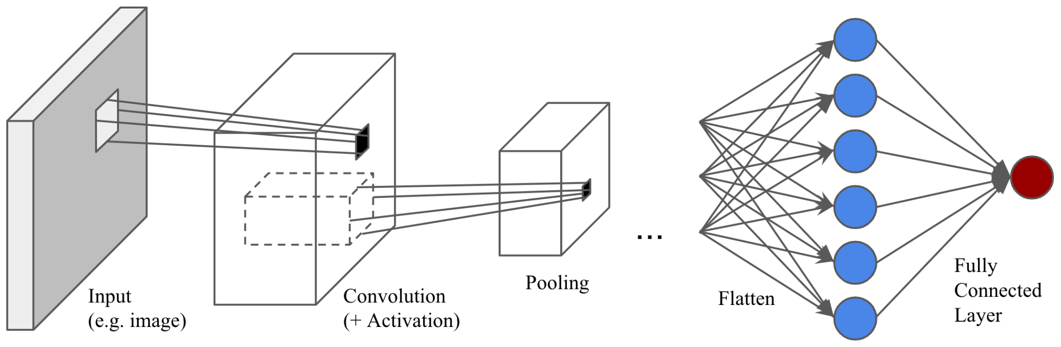
However, as we will analyze in the model repair for convolutional networks, this formulation is not directly applicable to convolutional networks which do not have a clear data design matrix and much less-so a well-defined row space of it. In particular, the design matrix for convolutional networks have additional spatial components and dimensions because there can be 2 or more spatial dimensions (e.g. grayscale images or 3D imaging data) as well as multiple channels (e.g. RGB images) and so the row-space is quite ambiguous. To fix this ill-defined row space, a key work for the convolutional model repair is to define a transformation to convert a convolutional network and its data to equivalent matrix forms that are computationally equivalent, apart from some reshaping of the outputs and matrices. With this form and appropriate transformations to the design matrix, we have better defined row and column spaces and empirically demonstrate encouraging results on the repairability of corrupted convolutional models.

3 EXTENSION TO CONVOLUTIONAL NETWORKS

Motivation

The applications to neural networks in Gao and Lafferty (2020) were restricted to feed-forward neural networks which are the simplest networks to analyze as there is rich literature on matrices (e.g. row spaces and column spaces) and feed-forward networks. However, as convolutional networks become more popular, especially in the computer vision domain, we would like to see if the same principles of model repair apply to convolutional networks. To verify this, we create a 2-layer convolutional network, train it, corrupt it, and attempt to recover it. Specifically, we will attempt to train the first layer followed by the second (after freezing the first) to parallel the experimentation of the feed-forward network. As with the feed-forward example, this approach can be interpreted as applying a second convolutional layer to the features extracted from the input by the first convolutional layer and avoids complications with a changing rowspace in $\widetilde{X}$. The work will be attempted on the MNIST dataset.

Background On Convolutional Networks and Their Invertibility



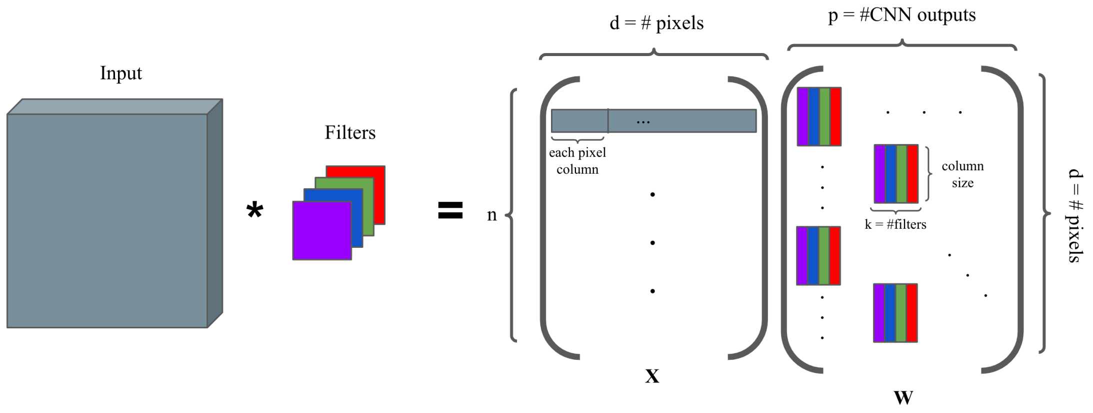
Above, we provide a visualization of a typical convolutional network. In general, convolutional networks are applied to inputs (typically images) that often have more than one spatial dimension. This is because unlike typical neural networks, convolutional filters are applied or ”convolved” on spatially neighboring data points or pixels. In this format, the convolutional network is able to better capture local relations among pixels compared to regular feed-forward networks which treat each pixel as an independent unit. An activation function such as the hyperbolic tangent or sigmoid can then be applied on top of this convolution operation. Because the number of outputs units depends on the number of filters as well as the number of convolutions that are applied across the input (which is a function of the stride, filter size, input size, and padding), the output size can grow quite large. Typically after a convolutional layer, there is a pooling layer such as average pooling or max pooling where you take the average or maximum value, respectively, of a ”pool” of neighboring points to reduce the dimensions of the feature map and the number of model parameters which can grow particularly large with convolutional networks. By applying multiple convolutional, and pooling, layers in sequence, one can extract more subtle and fine-grained features of the images. This spatial feature extraction is a key reason why convolutional networks are so highly used for image data (e.g. to extract facial features from facial images). After a series of convolutional layers, the output can be flattened, as it would typically be done for a classification or regression problem; there are other models such as autoencoders for which the output would not be flattened, but for the purposes of this paper (i.e. regression), we ignore those cases. With the flattened output, we can feed the output into a fully-connected regression layer and make the corresponding prediction.

Pivoting to the model repair problem, as Gao and Lafferty (2020) state, ”at its core, our formulation and analysis of model repair rests upon representing an estimator in terms of the row space of functions of the data design matrix.” However, as it is mentioned in the brief introduction to convolutional networks above, the data design matrix, and much less the row space of it, is quite ill-defined for a convolutional network. Because convolutional networks integrate spatial locality in pixels, it is not possible to naively flatten each image into a vector and bind them into a matrix as there is no obvious way to apply the convolutional filters to this vectorized form of the inputs.

In two separate papers, both Ma et al. (2018) and Gilbert et al. (2017) investigate the invertibility of convolutional networks. Although the applications and domains of each paper are quite different, they both use similar transformations to convert the convolutional network to a matrix form that is more interpretable and can be studied in the context of the model repair algorithm.

There are several key differences between the framework that we introduce and those in Ma et al. (2018) and Gilbert et al. (2017). For example, both analyze one-dimensional CNN architectures, but argue that the same style of transformation should extend to the 2-dimensional case. Below, we present an extension of the same principles to construct a framework that works for 2-dimensional CNNs.

Transformation of Convolutional Networks to Matrix Form



Above, we see a visualization of the transformation from the initial convolutional operation to a matrix form that can be used for the robust recovery of a convolutional network. The steps of the transforamtion are as follows. We first vectorize the input images from $\mathbb{R}^{d' \times d'}$ to $\mathbb{R}^{d}$ where $d = d' \cdot d'$, flattening the column dimension first to create the design matrix $X \in \mathbb{R}^{n \times d}$. Flattening the column dimension first is important for consistency but also to align with Keras' built-in $flatten()$ operator which flattens starting from the last dimension. Next, we construct the transformed convolutional weight matrix. To do so, we first need to define the output size of this matrix form. In particular, if the output shape of the convolutional layer is (m', m', k) where $m = m' \cdot m'$ is the number of convolutional outputs per filter and k is the number of filters, then the output has dimension $\mathbb{R}^{mk \times 1}$ and the weight matrix has shape $W \in \mathbb{R}^{d \times mk}$. Additionally, we fill the rows of W accordingly to have the outputs of multiple filters on one pixel column side-by-side, followed by blocks of columns, and finally blocks of rows. This is again so that the output vector is flattened starting from the last dimension (i.e. starting with number of filters). Although the MNIST digit data only has one channel, the same principle can be applied to inputs with multiple channels by multiplying the number of columns in X and the number of rows in W by the number of input channels and appropriately filling in the parameter values in the weight matrix W.

In CNN.R, you can see examples of convolutional networks that were trained and successfully transformed to the matrix form. To verify the results, we vectorized the input images into a design matrix X and demonstrated that the output of XW was equivalent to the output of the convolutional layer on the original images. Additionally, to demonstrate that we could both find the correct hidden output and reconstruct the original predictions by passing in the reconstructed hidden output, we compared the output with the predictions using the original convolutional model and found that we could recreate the equivalent output. However, with these approaches there are small differences (e.g. $1e-6$) indicating that the transformations are not perfect. These results were both applied in the model repair algorithm to build the convolutional matrix and to calculate $\widetilde{X}$.

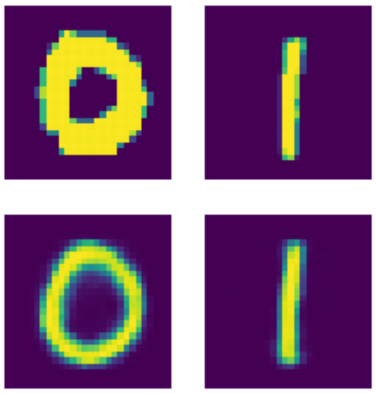
Model Repair Algorithm

Algorithm 1 Model Repair for Convolutional Network

```
  • Input: $n$ images with dimension $d' \times d'$ (one channel), initialized and trained weights $(\theta(0), \theta)$ for
    $k$ convolutional filters, initialized and trained weights $(\beta(0), \beta)$ for dense layer
1: Convert inputs to one dimension: Flatten each image by row into a 1-dimensional signal of length
   $d = d' \times d'$. Then create a design matrix $X \in \mathbb{R}^{n \times d}$ (i.e. each pixel is treated as a variable)
2: Corrupt each weight in $\beta, \theta$ with probability $\varepsilon$ to create $\eta, \Theta'$
3: Convert $\Theta', \theta(0)$ to matrix form by repeating filter weights to create $\Theta, W(0) \in \mathbb{R}^{d \times mk}$, where $m$ is
   the number of convolutional outputs per filter (visualized below except with only one channel)
   If $\theta$ were transformed in the same manner as $\theta(0)$, the original convolutional output could be modeled
   as $XW$; output would need to be reshaped to $m' \times m' \times k$ where $m = m' \times m'$
4: Run model repair for neural networks with contaminated model $(\eta, \Theta)$, design matrix $X$, and
   initializations $\beta(0), W(0)$ to recover $\widetilde{\beta}$ and $\widetilde{W}$
```

After repairing W, the optimal approach to reformat and apply W to the input prior to passing it into the pooling layer to find $\widetilde{X}$ is unclear. We explore a couple approaches in the results and conclusions sections.

Experiment Setup



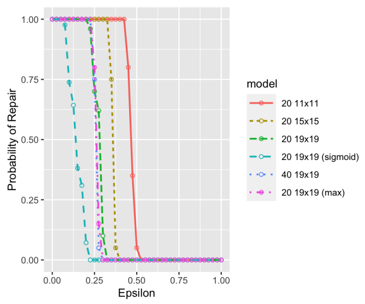
For the experiments, we use the MNIST digit data. By selecting a subset of the data (i.e. only 0 and 1), we convert the typical classification problem into a binary regression problem where we regress on the probability of an image being a 1 ($y = 1$) or not ($y = -1$ for hyperbolic tangent activation and $y = 0$ for sigmoid). This will be modeled with one convolutional layer passed through either a max or average pooling layer, and finally a dense regression layer with a hyperbolic tangent activation function to get the final prediction. To see the data collection and cleaning process, see CNN.ipynb.

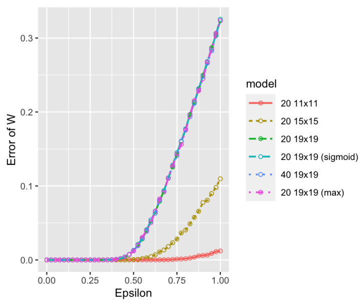
All of the models were trained in R using Keras with a TensorFlow back-end. The models can be trained through CNN.R and repaired through repair-cnn.R, but the code for the full pipeline starting from training a model through model repair can be found in cnn-full.R. Although most models are trained by retraining layerwise, we compare the results to the case in which we use regular gradient descent training in the next section. In the experiments, we vary the number of filters, size of the filters, type of pooling layer (max or average pooling), and activation function. The 6 models are as follows (note that all models have stride 1 and zero padding:

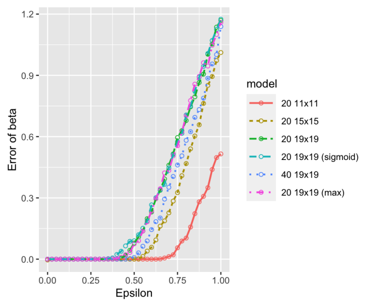
1. 20 filters of size 11×11 with hyperbolic tangent activation and average pooling
2. 20 filters of size 15×15 with hyperbolic tangent activation and average pooling
3. 20 filters of size 19×19 with hyperbolic tangent activation and average pooling
4. 20 filters of size 19×19 with sigmoid activation and average pooling
5. 20 filters of size 19×19 with hyperbolic tangent activation and max pooling
6. 40 filters of size 19×19 with hyperbolic tangent activation and average pooling

Each model was trained for at least 20 trials (with some being trained for longer). Because some of the trials took a while to run on the Yale High Performance Cluster (HPC), we trained some runs in smaller batches (in case the model repair got cut short by the time limit) and aggregated the runs afterwards; see aggregate-cnn.R for more details.

Results





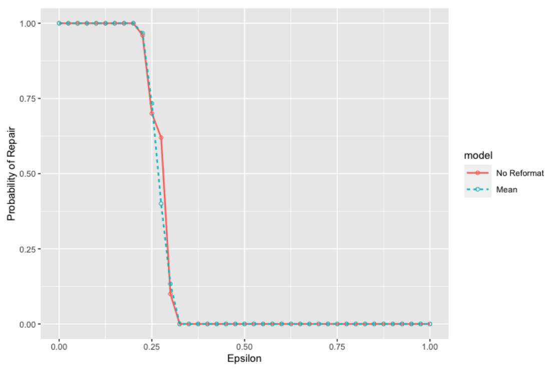


In the plot above, we notice some extremely interesting observations about the probabilities and errors of model repair. First, in the plot of probability of repair, we notice that the two models with 20 19×19 filters (one with average pooling and one max pooling) and the model with 40 19×19 filters have extremely similar probabilities of repair for varying levels of ε. This has several key implications. First, it is somewhat peculiar the two models with 20 19×19 filters have nearly identical probabilities of repair (as well as W and β errors). Because max pooling selects the pixels of highest intensity, we hypothesized that this would interfere with the redundancy that is required for successful model repair of β and restricts sparse estimators from being able to be robustly recovered with the model repair framework, but it appears this may not necessarily be the case. Another surprising result is that the model with 40 19×19 filters appears to do about as well as the models with half as many filters. As we increase the total number of parameters, we would typically expect the probability of repair to increase. We see that the same model with 40 19×19 filters has slightly lower β error and so it may be the case that there is a marginal improvement in the recoverability of the model with 40 filters over 20 that is not discernible in the plot of repair probability for 20 trials.

Additionally, we see that the probability of repair for the model with the sigmoid function is significantly less than the analogous model with hyperbolic tangent. As the sigmoid activation function was not studied in Gao and Lafferty (2020) unlike hyperbolic tangent or ReLU, we surmise that models built with the sigmoid function are generally less repairable, likely having to do with the asymmetry of the sigmoid activation compared to the hyperbolic tangent activation function. We leave proof of this for a future work.

Finally, what is potentially the most interesting trend in the plots at first glance is the fact that the model with the smallest filter size and thus the fewest parameters (20 11×11 filters) actually was the most repairable and had the smallest β and W errors, followed by the model with the next smallest filter size and next fewest parameters (20 15×15 filters). This is actually a consequence of our convolutional matrix transformation. When we construct our convolutional weight matrix W, recall that although the number of rows is fixed to the number of pixels in each image, the number of columns actually scales to the total number of convolutional outputs from the convolutional layer. For convolutional layers with stride 1, an increase in the size of the filters leads to a direct decrease in the convolutional output size. Therefore, the model with 20 11×11 filters (which actually has output size $(18, 18, 20)$ and thus $p = 18 \cdot 18 \cdot 20 = 6480$ parameters) has significantly more parameters than the model with 20 19×19 filters (output size $(10, 10, 20)$ and thus $p = 10 \cdot 10 \cdot 20 = 2000$ parameters).

Although the models above were trained layerwise, by comparing the performance to the case in which we use regular gradient descent training, we notice analogous results to those for feed-forward networks from the paper. In particular, we notice that the repairability of the models goes to zero because the column space of W is constantly changing during training which makes β difficult to exactly repair. However, the error plots look very similar indicating that approximate recoverability is an easier requirement for regular gradient descent training than exact repair.



As noted, the optimal approach to reformatting and applying W to the input after repair is unclear. Although the models above are repaired by directly calculating $\widetilde{X}$ from the recovered $\widetilde{W}$, there are many other approaches that could be taken to extract the hidden features. For example, after recovering the repaired W matrix $\widetilde{W}$, one could convert the matrix back into a convolutional operation prior to finding $\widetilde{X}$ to reflect the shared parameter constraint of CNNs. Because of the redundancy in convolutional parameters in the matrix form, this would require mapping the redundant parameters back to one unified parameter. To the left, we show a plot comparing the approach of directly applying $\widetilde{W}$ with finding the mean of all corresponding parameters in $\widetilde{W}$ to repair the filters and note that the results are quite comparable. However, because of the lossy transformation discussed in the convolutional transformation to matrix form (and back), we are wary of regularly transforming back and forth between these two forms; a priority in extending this work would thus be to find a less lossy form of this transformation. Other approaches (e.g. with median of all corresponding parameters in $\widetilde{W}$) are left to future works.

Conclusion

As the work above demonstrates, the results of model repair appear to generally extend to the case of convolutional networks for a variety of model architectures. However in an relatively unexplored field of model repair, the work in this report on convolutional networks touches the surface of possibilities of convolutional model repair. Unlike feed-forward networks, convolutional networks generally have a lot more hyperparameters that can be tuned and modifications that can be made. For example, even in the work in this paper. we experimented with filter size, number of filters, activation function, and pooling type, but there are many other parameters such as pool size, stride, padding, etc that could be modified to determine exact conditions under which model repair is possible with convolutional networks.

Because the matrix form of the convolutional network represents an equivalent operation to the regular convolutional operation, the form of model repair in matrix form presented in the paper and code is valid and empirically works. However, there are modifications to the corruption and repair algorithm which could be modified for convolutional networks. For example, although we are currently corrupting the convolutional filters prior to building the matrix W, we could attempt to corrupt W after transforming to matrix form. Because this matrix form of convolutional networks incorporates explicit redundancy in the repeated parameters, we actually find that this increased redundancy improves repairability, as seen because the model with 20 11×11 filters is actually more repairable than one of 20 19×19 filters which technically has more initial parameters. This could actually encourage the transformation and storage of convolutional networks in matrix form. The results suggest that this could help the robustness and maintenance of convolutional networks and could also lead to standardization of model storage for both convolutional and feed-forward networks.

As noted in the discussion of the convolutional transformation to matrix form, our current transformation of the convolutional filters is lossy; after transforming, there is a non-negligible difference in the hidden output and prediction output (on the order of $1e-6$). This difference may have to do with the interface between the TensorFlow back-end and R which may be unable to store numbers with enough precision to accurately reflect the transformation. Therefore, an extension of this work would be to find a less lossy transformation and representation of the convolutional operation to get exact model repair.

4 MODEL REPAIR WITH REAL DATA

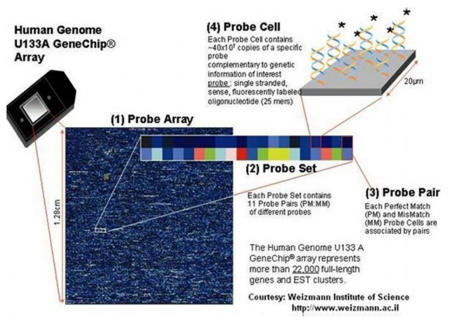
Motivation

In the paper, the datasets are primarily toy datasets of known, generated distributions. It would thus be worthwhile to use a real, larger dataset and ensure that the results of the paper are truly robust and can be applied in real domains. In particular, the work in this paper has implications for privacy because it suggests that the uncorrupted model and thus even the original response variable can be recovered from a corrupted model. Therefore, robustness on real data is especially important to test to be able to best respond and protect sensitive user data.

There are additional potential implications of this application. For example, because encryption can be considered a reversible form of corruption, we could imagine that there are complications with insufficient encryption; with a particularly vulnerable form of encryption, there may be a risk of data or model leaks. In fact, as Gao and Lafferty (2020) finds, a corrupted model is recoverable even as $\varepsilon \to 1$ which means that if a hacker were able to decrypt or access a fraction of a redundant and over-parameterized model's parameters, it would still be able to recover the original model with high probability and thus recover sensitive data (e.g. the response variable). With genomics data, this could be a patient's risk of cancer or probability of death, which has significant ramifications for privacy.

Colon Data

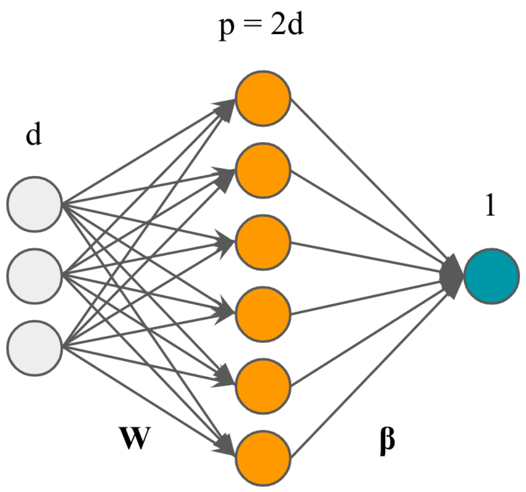
The data used for this model is from a paper from the Weizmann Institute which attempted to cluster tumor and normal colon tissues by probing oligonucleotide arrays (Alon et al., 1999). The data set consists of 40 tumor tissue and 22 normal colon tissue samples with a total of 6500 gene expressions measured across the Affymetrix oligonucleotide array of each tissue. The public version of this data only includes 2000 of the genes with the highest minimal intensity across the 62 different samples. Therefore, we ultimately end up with a full design matrix $X \in \mathbb{R}^{62 \times 2000}$. Additionally, the response variable in this case is whether the original


Human Genome U133A GeneChip® Array
(4) Probe Cell
Each Probe Cell contains ~40x10⁷ copies of a specific probe complementary to genetic information of interest
probe: single stranded, sense, fluorescently labeled oligonucleotide (25 mers)
20μm
(1) Probe Array
1.28cm
(2) Probe Set
Each Probe Set contains 11 Probe Pairs (PM:MM) of different probes
(3) Probe Pair
Each Perfect Match (PM) and MisMatch (MM) Probe Cells are associated by pairs
The Human Genome U133 A GeneChip® array represents more than 22,000 full-length genes and EST clusters.
Courtesy: Weizmann Institute of Science http://www.weizmann.ac.il

tissue sample was from a healthy tissue ($y_i = 1$) or a colon sample ($y_i = 2$) which built the full response vector of $y \in \mathbb{R}^{62\times 1}$.

The full data can be found in colon_data.csv.

Model Architecture



The model architecture used for the experiments is a 2-layer feed-forward network with final output dimension of 1 trained to predict the response variable (i.e. cancer or normal tissue). To observe the effect for varying numbers of parameters in the model, we try varying input dimensions $d \in [100, 150, 250, 500, 1000]$ with hidden dimension $p = 2 \cdot d$ for a fixed value of d. In other words, the two weight matrices are $W \in \mathbb{R}^{d\times p}$ and $\beta \in \mathbb{R}^{p\times 1}$. To vary the input dimension, we select a random subsample of the original design matrix as $\bar{X} \in \mathbb{R}^{62xd}$. Additionally, we include a hyperbolic tangent activation function φ prior to the hidden layer to verify the related results from Gao and Lafferty (2020). Therefore the model output can be modeled:

$$f(X) = \varphi(XW)\beta$$

Each weight in the matrices $W(0)$ and $\beta(0)$ is initialized via a random normal distribution with a seed for reproducibility and to demonstrate that with just the seeds, model repair can be implemented using just the seeds. Additionally, the model is trained using gradient descent with mean squared error loss. Specifically, the results in the paper are demonstrated using the layerwise approach of model repair in which both layers are simultaneously trained through one pass. Then, on the second pass, the weights in W are frozen and the weights in β are initialized to 0 and independently trained. This approach was demonstrated to allow for exact repair in the original paper and extends to applications on real data.

Experiment Setup

To train the models, we first train several models of different sizes: $d \in [100, 150, 250, 500, 1000]$ with $p = 2 \cdot d$; the corresponding number d of the original variables are randomly subsetted using a reproducible seed. With decreasing learning rates to reflect the increasing size and numbers of parameters in each model, each model is then trained for 100000 steps for both steps of the layerwise model training. The models, along with the corresponding weight initializations and design matrices are then saved to be used in the model repair algorithm. The code and plots for the models trained with both 20000 and 100000 steps can be found in Colon.ipynb.

Then to perform the model repair, we import the trained and initialized weight matrices into R, corrupting ε fraction of the model parameters for various values of ε and for a variable number of trials. Because of lack of computational resources, some of the model repair trials on the larger models were performed in smaller sized batches and aggregated ex post facto; see aggregate-colon.R for further details.

Then, to actually repair the model, we use the *rq* method of the *quantreg* module which calculates median regression for $\tau = 0.5$ to repair each column $j \in [p]$ of the matrix W:

$$\widetilde{v}_j = \underset{v_j}{\operatorname{argmin}} ||\Theta_j - W_j(0) - X^T v_j||_1$$

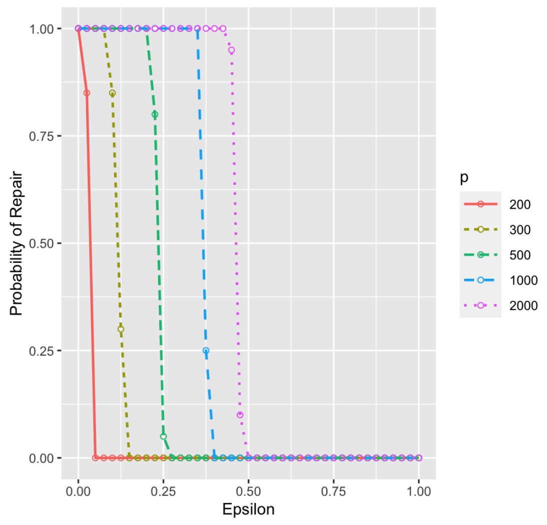
to find $\widetilde{W}_j = W_j(0) + X^T \widetilde{v}_j$ followed by recovering the output layer:

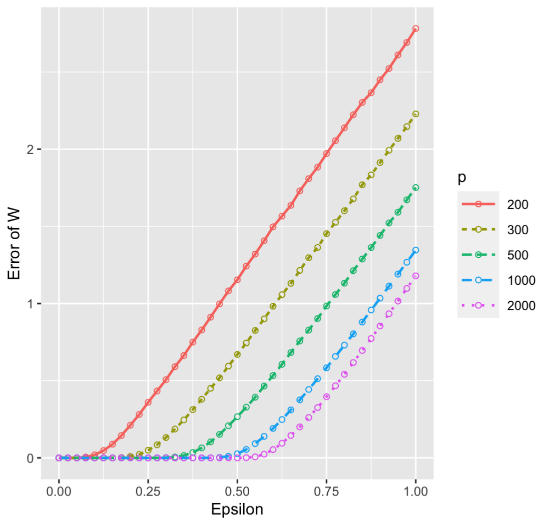
$$\widetilde{u} = \underset{u}{\operatorname{argmin}} ||\eta - \beta(0) - \varphi(\widetilde{W}^T X^T)u||_1$$

to find $\widetilde{\beta} = \beta(0) + \varphi(\widetilde{W}^T X^T)\widetilde{u}$

To see the code run for these simulations, see repair-colon.R

Results





In the plot above, we see the results for model repair on the colon data from Alon et al. (1999). As we might have expected, we notice that the probability of repair increases as we increase the number of parameters in the neural network. In particular, even for values of ε around 0.5 (i.e. half of the model parameters are corrupted), the model is exactly repairable with $p = 2000$. Note that this bound of $\varepsilon = 0.5$ is not strict and the probability of repair appeared to be high for $p = 4000$ with $\varepsilon > 0.5$; the plot for $p = 4000$ is omitted from the paper due to an insufficient number of trials and parallel trends to lower values of p. Additionally, looking at the plot on the right, we notice more similarities between the paper's results on the toy dataset and the colon dataset. Specifically, we notice that for any value of ε, the error decreases as we increase the number of model parameters. This is another demonstration of the results from Gao and Lafferty (2020) that over-parameterization is a key component in determining the repairability of a model.

The plot for the error of the β layer follows the same pattern as W for a model that is repairable; specifically, as long as the error of W is not extremely large for a fixed value of ε, we see the same trend in the error of β. However, when W is unrepairable for a fixed value of ε and the W error becomes too large, we get some errors when attempting to repair β such as "Error in rq.fit.fnb(x, y, tau = tau, ...) : ... singular design". Even though β is already unrepairable in these cases and thus the repairability of W is of less concern, we did further research out of curiosity. We found that the error may be due to some collinearities in the initial covariates and/or tissue samples with nearly identical gene expressions. In an attempt to fix this, we add a slight amount of jitter to the design matrix that would leave the variables mostly unaffected but correct the design matrix singularity. While this helped the repairability of some of the less corrupted models, this did not fully fix the issues with the singularities. Taking a step back and re-analyzing our design matrix, the leading theory is that because genomics data is known for being sparse and because it was initially meant for clustering, there are likely collinearities in the gene expressions, which become observations in the quantile regression model where we transpose the inputted design matrix X. In fact, it has been noted that quantile regression generally has issues with singularities when the data is binned, as it likely is with clustering data to differentiate between normal and colon tissues. Thus for significant amounts of corruption, we would be more likely to run into these issues of singularities. To bypass this issue, a solution could be to run a separate dimensionality reduction method such as principal component analysis or factor analysis to decrease collinearity and ensure that each of the variables are linearly independent from one another and train the neural network on that data. However, as noted initially, this issue is likely not of great concern as it only affects models that are overly corrupted and already beyond repair (i.e. for values of ε that are well above the values for which probability of repair is 0 in the left plot).

For the models that were trained with one-pass and not layerwise (i.e. training the first layer, then initializing to 0 and training the second layer after freezing the first layer), the probability of exact repair without retraining in a layerwise approach is zero for models of all sizes (and thus uninteresting to plot) which is analogous to the results found in Gao and Lafferty (2020). Interestingly, we notice nearly identical trends in the plots of the errors of W and β to the model trained layerwise with no discernible difference between the plots. This indicates that while the parameters are nearly recoverable with the naive one-pass approach to training, exact repair must be done layerwise, which corroborates the results from the paper. This is because when the two layers are trained jointly, the column space of $\widetilde{X} = \varphi(XW)$ is constantly changing and updating, and thus makes the second layer impossible to repair exactly. However, because of the neural tangent kernel, the trained parameters are still able to be roughly recovered which is why there is no discernible difference between the plots of errors for the retrained and naively trained model repairs.

Additionally, the plots and results for stepsize 20000 compared to 100000 are nearly identical and thus omitted to avoid redundancy. This implies that regardless of how under- or over-fit a model is, the model repair is still applicable and can recover a corrupted model with high probability. However, the amount of overfitting does have significant ramifications for the recoverability of the response variables on which the data are trained. Therefore, by recognizing that a model can be recovered essentially regardless of the amount of training, the risk of data privacy leaks is very pertinent for all models and especially so for overfit and over-parameterized models that can precisely predict sensitive information about a consumer.

Conclusion

By verifying the results of Gao and Lafferty (2020) on real data, we have demonstrated that the simulations and results of model repair can have substantial consequences on real data. For example, one application of the work could be that when deploying or storing a machine learning model, a company should carefully distributively store the model across multiple databases. By partitioning it across multiple sources, if there is failure or corruption on one component of the stored model, the results of this paper imply that the original model can still be recovered without retraining from scratch using the uncorrupted elements of the over-parameterized and redundant model even as the fraction of corrupted parameters $\varepsilon \to 1$.

Conversely, if we are not careful with data and models that are trained on sensitive user data, we may end up insufficiently encrypting a model and thus leaking people's data. Because encryption can be thought of as a reversible form of corruption, a poor encryption algorithm could theoretically be exploited by black-hat hacker who recovers or obtains a subset of the uncorrupted parameters and is then able to repair the corrupted ones. Similarly, if we want to corrupt a model before disposing of a model, we need to ensure that the corruption is sufficient such that the model is not in a recoverable state. For example with sensitive data like genomics, we want to be extra careful that we do not leave a model in a vulnerable state that is assumed to be corrupted and allow a malicious hacker to use the weakly corrupted data to recover the original model or the original response variables. As we saw, with sufficiently over-parameterized models, the threshold for a model's corruption can be 50% and even approach 100% and still be recoverable.

5 NP-COMPLETENESS OF ROBUST ESTIMATORS

Motivation

An interesting observation from the paper is that the model repair problem for a corrupted linear model can be re-formulated as a robust regression problem. If we consider η as the corrupted response vector and $A = X^T \in \mathbb{R}^{p \times n}$ as the uncorrupted design matrix, we can solve the analogous robust regression problem.

As a robust regression problem, we can search broader related literature to determine the lower-bound on the run-time complexity. If we make a simplifying assumption that all $(1-\varepsilon)n$ uncorrupted data points exist on the same hyperplane (i.e. no noise), this is akin to the degenerate point subset problem (c-DPS) where $c = \varepsilon$. A publication from 2006 proves that c-DPS is NP-hard via a reduction from the NP-hard vertex cover problem (Bernholt, 2006). While this proof relies on the portion of points not on the hyperplane (i.e. corruption probability) $c = \varepsilon \leq 0.5$, we would like to see if the work could extend to the case of $c > 0.5$.

Introduction

In the typical robust linear regression problem, we model the corruption as:

$$Y_i = X_i^T \beta^* + W_i + Z_i, \quad we = 1, \ldots, n$$

where W_i represents noise and the corruption $Z_i \sim (1-\varepsilon)\delta_0 + \varepsilon Q_i$ where Q_i is an arbitrary corruption distribution and ε represents the probability of corruption.

If we were to make a simplifying assumption that $W_i = 0$ (i.e. no noise), the output would be:

$$Y_i = X_i^T \beta^* + Z_i, \quad i = 1, \cdots, n$$

which is equivalent to finding a subset of $\lceil (1-\varepsilon)\cdot n \rceil$ co-linear points. This is exactly equivalent to the degenerate point subset problem (c-DPS), defined below, with $c = \varepsilon$.

Definition 5.1 (Degenerate Point Subset Problem (c-DPS))**.** Given n points in d dimensions and a fixed parameter c where $0 < c < 1$, are there parameters $\beta_1, \ldots, \beta_d$ such that $\lceil (1-c)\cdot n \rceil$ points are on the hyperplane defined by $y = \sum_{i=1,\ldots,d-1} \beta_i x_i + \beta_d$?

Remark. This problem is slightly different from the one defined in Bernholt's paper because we extend the restriction of c from $(0, 0.5]$ to $(0, 1)$.

Additionally, we define the well-studied vertex cover problem below:

Definition 5.2 (Vertex Cover Problem)**.** Given a graph $G = (V, E)$ and a fixed parameter k, can we find a $V' \subseteq V$ with $|V'| = k$ such that each edge in E includes at least one vertex in V'? In other words, $v_1 \in V' \vee v_2 \in V' \; \forall \; e = (v_1, v_2) \in E$.

NP-Completeness of c-DPS

Theorem 5.1. *The c-DPS problem is NP-complete for all fixed values of c where* $0 < c < 1$.

Proof. To prove this statement, we start by reducing the vertex cover problem to the c-DPS problem. In other words, we use the c-DPS problem as a subroutine to answer the vertex cover problem. Therefore, we start by defining this Turing Reduction.

Construction of Reduction

c-DPS returns yes if there exists a hyperplane of h points among the $n = h + c$ total:

$$y = \sum_{i=1,\ldots,d-1} \beta_i x_i + \beta_d$$

Via a reduction from vertex cover, we can prove that the c-DPS problem is NP-hard for any c where $0 < c < 1$. Given a graph G and parameter k for the vertex cover, we construct a new set of points for c-DPS as follows:

- v_i and v_i^*:

  For each vertex i in G, we construct two new points v_i and v_i^* (where $x_i = 1$) which are used to determine if vertex i is in the vertex cover. i is included in the vertex cover if v_i is in the hyperplane (i.e. $\beta_i = 2$) and not included if v_i^* is in the hyperplane (i.e. $\beta_i = 1$). We will later prove that $\beta_i = 1$ and $\beta_i = 2$ are the only two meaningful values of β_i.

- e_{ij} and e_{ij}^*:

  For each edge (i, j) in G, we construct two new points e_{ij} and e_{ij}^* which are used to determine if the edge is covered in the vertex cover. If only one of the vertices i, j is in the vertex cover, then e_{ij} is on the hyperplane; if both are in the vertex cover, then e_{ij}^* is on the hyperplane. As we mentioned above and see in the coordinate table below, this is because $\beta_i = 2$ if and only if i is in the vertex cover and 1 otherwise. If neither of the vertices is in the vertex cover, neither e_{ij} nor e_{ij}^* will be on the hyperplane.

- K and K^*

  The point K is used to ensure that $\sum_{i=1,\ldots,|V|} \beta_i = |V| + k$ (because $x_i = 1$, $1 \leq i \leq |V|$). Therefore, K is included only if exactly k vertices belong to the vertex cover (i.e. $|\{\beta_i = 2 | i = 1, \ldots |V|\}| = k$). K^* is non-essential for the vertex cover but adds symmetry for the $c = 0.5$ base case.

- $K_1,\ldots,K_\eta$

  These points are new and non-essential for the vertex cover problem, but allow us to extend the reduction of c-DPS to $c > 0.5$. Because they have the same coordinates as K with negative y values, we see that these points cannot be included in any hyperplane found by c-DPS of size h.

- p and p^*

  The point p is used to ensure that $\beta_d = 0$ (because $x_i = 0$, $1 \le i \le d-1$). p^* is non-essential for the vertex cover but adds symmetry for the $c = 0.5$ base case.

- $p_1,\ldots,p_\zeta$

  These points are non-essential for the vertex cover, but allow us to extend the NP-completeness of c-DPS from the base reduction of $c = 0.5$ to all $0 < c \le 0.5$.

The exact values of the coordinates for each of the points described above are defined below:

		$x_1,\ldots\ldots,x_i,\ldots\ldots,x_j,\ldots,x_{\lvert V\rvert}$	,	$x_{\lvert V\rvert+1},\ldots,x_{\lvert V\rvert+l},\ldots,x_{d-1}$		y
v_i	$=($	$0,\ldots,0,1,0,\ldots\ldots\ldots\ldots,0$	,	$0,\ldots\ldots\ldots\ldots\ldots,0$	$)$	1
v_i^*	$=($	$0,\ldots,0,1,0,\ldots\ldots\ldots\ldots,0$	,	$0,\ldots\ldots\ldots\ldots\ldots,0$	$)$	2
e_{ij}	$=($	$0,\ldots,0,1,0,\ldots,0,1,0,\ldots,0$	,	$0,\ldots\ldots\ldots\ldots\ldots,0$	$)$	3
e_{ij}^*	$=($	$0,\ldots,0,1,0,\ldots,0,1,0,\ldots,0$	,	$0,\ldots\ldots\ldots\ldots\ldots,0$	$)$	4
K	$=($	$1,\ldots\ldots\ldots\ldots\ldots\ldots,1$	,	$0,\ldots\ldots\ldots\ldots\ldots,0$	$)$	$\lvert V\rvert+k$
K^*	$=($	$1,\ldots\ldots\ldots\ldots\ldots\ldots,1$	,	$0,\ldots\ldots\ldots\ldots\ldots,0$	$)$	$\lvert V\rvert+k-1$
K_m	$=($	$1,\ldots\ldots\ldots\ldots\ldots\ldots,1$	,	$0,\ldots\ldots\ldots\ldots\ldots,0$	$)$	$-m$
p	$=($	$0,\ldots\ldots\ldots\ldots\ldots\ldots,0$	,	$0,\ldots\ldots\ldots\ldots\ldots,0$	$)$	0
p^*	$=($	$0,\ldots\ldots\ldots\ldots\ldots\ldots,0$	,	$0,\ldots\ldots\ldots\ldots\ldots,0$	$)$	1
p_l	$=($	$0,\ldots\ldots\ldots\ldots\ldots\ldots,0$	,	$0,\ldots\ldots,0,1,0,\ldots\ldots,0$	$)$	1

Additionally note the following dimensions which will be useful in our proof of the correctness of the reduction:

$$\zeta = \begin{cases} 2\cdot\lceil\frac{1-2c}{2c}\cdot(|V|+|E|+2)\rceil, & \text{if } c < 0.5 \\ 0, & \text{otherwise} \end{cases}$$

$$\eta = \begin{cases} \lceil\frac{2c-1}{1-c}\cdot(|V|+|E|+2)\rceil, & \text{if } c > 0.5 \\ 0, & \text{otherwise} \end{cases}$$

$$d = 1+|V|+\zeta$$

$$n = 2\cdot(|V|+|E|+2)+\zeta+\eta$$

$$h = (|V|+|E|+2)+\zeta$$

Proof of Reduction

Lemma 5.2. *If there is a vertex cover of size k, there exists a hyperplane with h points.*

Proof. We start with a vertex cover V' of size k. Then, we can find a hyperplane with h points by setting $\beta_i = 2$ if vertex $i \in V'$ and $\beta_i = 1$ otherwise. Additionally, we set $\beta_{|V|+1},\ldots,\beta_{d-1} = 1$ and $\beta_d = 0$.

By construction, we know that either v_i or v_i^* is found on the hyperplane (because $\beta_i \in \{1,2\}$) which is a total of $|V|$ points in the hyperplane. Additionally, because V' is a vertex cover, we know that for any edge $e = (i,j)$, either one of the vertices i,j is included in the vertex cover (i.e. $(\beta_i,\beta_j) \in \{(2,1),(1,2)\}$ and e_{ij} is included in the hyperplane) or both i,j are included in the vertex cover (i.e. $\beta_i = \beta_2 = 2$ and e_{ij}^* is included in the hyperplane). This is an additional $|E|$ points in the hyperplane. Next, because there is a vertex cover of size k, we know that exactly k of the original vertices are included in the vertex cover and thus there are k values of $\beta_i = 2$ and $|V|-k$ values of $\beta_i = 1$ for $i \in [1,|V|]$ and so K must be on the hyperplane ($k\cdot 2+(|V|-k)\cdot 1 = |V|+k$). Because we choose $\beta_d = 0$, p must be trivially included in the hyperplane. Finally, because we set $\beta_{|V|+1},\ldots,\beta_{d-1} = 1$, all ζ points p_l are found on the hyperplane. Therefore, we find $|V|+|E|+2+\zeta = h$ points on the hyperplane. $\square$

Lemma 5.3. *If there is no vertex cover of size k (i.e. all vertex covers are larger than k), then all hyperplanes contain less than h points.*

Proof. Because we cannot have vertical hyperplanes with our current hyperplane representation, we know that no two points in each of the following sets of points can be on the same hyperplane because they have the same coordinate values with differing y values:

- v_i and v_i^*
- e_{ij} and e_{ij}^*
- p and p^*
- K, K^*, and $K_1, \dots K_\eta$

Including the points $p_1, \dots p_\zeta$, we can have at most $|V|+|E|+2+\zeta = h$ points on the same hyperplane. Therefore, it is sufficient to demonstrate that without a vertex cover V' of size $|V'| \le k$, at least one of these sets of points will not be included and thus the hyperplane contains less than h points.

First, we assume that parameter values are "correct" (i.e. follow the distribution of expected values): $\beta_i \in \{1,2\}$ for $i = 1, \dots, |V|$ and $\beta_l = 1$ for $l = |V|+1, \dots, d-1$ and $\beta_d = 0$. With that in mind, we consider any arbitrary subset of the original vertex cover problem's vertex set $V' \subseteq V$ and set $\beta_i = 2 \iff v_i \in V'$ (otherwise $\beta_i = 1$).

Considering $|V'| \le k$, we know that based on the assumption in the lemma, there exists no vertex cover of size $\le k$. Therefore, there must exist some $(i,j) \in E$ that is not covered by the vertex cover and thus either $v_i \notin V'$ or $v_j \notin V'$; otherwise V' would be a valid vertex cover of size $\le k$. Thus, $\beta_i = \beta_j = 1$ which means that neither e_{ij} nor e_{ij}^* can be on the hyperplane. As described above, this means that we cannot have a total of h points on the hyperplane; the hyperplane would have at most $|V| + |E \backslash \{(i,j)\}| + 2 + \zeta = h-1$ points. Alternatively if we consider subsets of size larger than k i.e. $|V'| > k$, then $\sum_{i=1,\dots|V|} \beta_i > |V| + k$ and thus none of K, $K*$, or $K_1, \dots K_\eta$ can be included on the hyperplane.

Second, we assume that coefficient values are not necessarily "correct" (i.e. they can have any value). We first recognize that β_d must be either 0 or 1 or else neither p or p^* would be included in the hyperplane. If we consider $\beta_d = 1$, we realize that $\beta_i \in [0,1]$ for all $i = 1, \dots, |V|$ so that one of v_i or v_i^* is on the hyperplane, but that implies $\sum_{i=1,\dots,|V|} \beta_i \le |V|$ which means neither K or K^* cannot be located on the hyperplane; $\sum_{i=1,\dots,|V|} \beta_i \ge 0$ so it would also be impossible for $K_1, \dots K_\eta$ to be included on the hyperplane. Thus, with $\beta_d = 0$, we consider the possibility of $\beta_i \notin [1,2]$ for some $i \in [1, |V|]$ but realize that this would mean neither v_i nor v_i^* is not located on the hyperplane: $\sum_{j=1,\dots,d-1} x_j \cdot \beta_j = \beta_i \notin [1,2] \implies v_i, v_i^*$ not located on the hyperplane. □

We also note that it is trivial to check that a candidate solution to c-DPS is valid by iterating through all points i and checking that $y = \sum_{i=1,\dots,d-1} \beta_i x_i + \beta_d = y_i$. Therefore, we have completed proving the NP-completeness of c-DPS. □

***Initial Restriction to** $c \le 1/2$*

In the original framework, we construct two points for each original vertex and two points for each original edge from the vertex cover graph. Additionally we construct four additional points K, K^*, p, p^* to ensure that the number of vertices included in the vertex cover is equal to k and that $\beta_d = 0$, respectively (again K^*, p^* are non-essential but included for symmetry). With these sets of points, we can therefore prove the NP-hardness of c-DPS with $c = \frac{|V|+|E|+2}{2(|V|+|E|+2)} = 0.5$. By including ζ additional points $p_1, \dots, p_\zeta$ points, we extend our argument from the base case of $c = 0.5$. By including ζ additional points, this is equivalent to the c-DPS problem with:

$$\begin{aligned} c = 1 - \frac{h}{n} &= 1 - \frac{|V|+|E|+2+\zeta}{2\cdot(|V|+|E|+2)+\zeta} \\ &= 1 - \frac{(|V|+|E|+2+\zeta/2)+\zeta/2}{2\cdot(|V|+|E|+2)+\zeta} \\ &= \frac{1}{2} - \frac{\zeta/2}{2\cdot(|V|+|E|+2)+\zeta} \le 0.5 \end{aligned}$$

Therefore by adding sufficiently many points $p_1, \dots, p_\zeta$, the argument applies to $c \to 0$ as $\zeta \to \infty$.

Extension to $c > 1/2$

Similar to how we add ζ points $p_1, \ldots, p_\zeta$ which are trivially on the hyperplane, we introduce η points which cannot be on the hyperplane. Because vertical hyperplanes are not possible with the current hyperplane representation, only one of the η points $K_1, \ldots, K_\eta$ that have the same coordinates as K and K^* with negative y values could be included on a hyperplane. Additionally because all of the values of $\beta_i = 1$ or 2 for $1 \leq i \leq |V|$ as we prove above, no K_m for $m = 1, \ldots \eta$ could be included in a solution hyperplane of size h. With these points, h is unchanged but the total number of points n has now increased to $n = 2 \cdot (|V| + |E| + 2) + \zeta + \eta$. By construction, if $\eta > 0$ then $c > 0.5$ and $\zeta = 0$ and thus:

$$\begin{aligned} c = 1 - \frac{h}{n} &= 1 - \frac{|V| + |E| + 2}{2 \cdot (|V| + |E| + 2) + \eta} \\ &= 1 - \frac{(|V| + |E| + 2 + \eta/2) - \eta/2}{2 \cdot (|V| + |E| + 2) + \eta} \\ &= \frac{1}{2} + \frac{\eta/2}{2 \cdot (|V| + |E| + 2) + \eta} \geq 0.5 \end{aligned}$$

Therefore by adding sufficiently many points $K_1, \ldots, K_\eta$, the argument applies to $c \to 0$ as $\eta \to \infty$.

Application

We have now proven the NP-completeness of c-DPS. If we were to consider the original robust regression problem with noise, this problem must also be NP-complete. It is relatively trivial to note that the C-DPS problem could be reduced to the robust regression by setting the noise to 0 and directly applying the robust regression to the c-DPS problem. As it was discussed by Gao and Lafferty (2020), the corrupted model recovery algorithm can be framed as a robust regression problem with an uncorrupted design matrix. The result in this paper is thus quite significant because it indicates that there is a computational bottleneck in the model repair of corrupted linear models. This has further implications for neural network model recovery which, at the simplest level, can be used as a subroutine to solve the linear model and thus is likely also NP-hard. Especially given that the model repair of a model requires over-parameterization, this may make model repair for extremely large models computationally infeasible. A future work could be to connect this computational bottleneck to other estimators and get lower-bounds on the complexity of these other problems. Another possible application of the result of the NP-completeness of robust regression could be to time the recovery of models of varying sizes and determine how large the model needs to be for model repair to become computationally impractical or infeasible.

ACKNOWLEDGEMENTS

I would like to thank Professor Lafferty for his guidance throughout my research and project suggestions.

REFERENCES

Alon, U., Barkai, N., Notterman, D. A., Gish, K., Ybarra, S., Mack, D., and Levine, A. J. (1999). Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences*, 96(12):6745–6750.

Bernholt, T. (2006). Robust Estimators are Hard to Compute. Technical Reports 2005,52, Technische Universität Dortmund, Sonderforschungsbereich 475: Komplexitätsreduktion in multivariaten Datenstrukturen.

Gao, C. and Lafferty, J. (2020). Model repair: Robust recovery of over-parameterized statistical models.

Gilbert, A. C., Zhang, Y., Lee, K., Zhang, Y., and Lee, H. (2017). Towards understanding the invertibility of convolutional neural networks.

Ma, F., Ayaz, U., and Karaman, S. (2018). Invertibility of convolutional generative networks from partial measurements. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.