EXTRACTING INFORMATION FROM DEEP LEARNING MODELS FOR COMPUTATIONAL BIOLOGY

Tianxiang Gao

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2017

Approved by:

Vladimir Jojic

Jeff Dangl

Leonard McMillan

Marc Niethammer

Mohit Bansal

ABSTRACT

Tianxiang Gao: Extracting information from deep learning models for computational biology
(Under the direction of Vladimir Jojic and Jeff Dangl)

The advances in deep learning technologies in this decade are providing powerful tools for many machine learning tasks. Deep learning models, in contrast to traditional linear models, can learn nonlinear functions and high-order features, which enable exceptional performance. In the field of computational biology, the rapid growth of data scale and complexity increases the demand for building powerful deep learning based tools. Despite the success of using the deep learning methods, understanding of the reasons for the effectiveness and interpretation of models remain elusive.

This dissertation aims to provide several different approaches to extract information from deep models. This information could be used to address the problems of model complexity and model interpretability.

The amount of data needed to train a model depends on the complexity of the model. The cost of generating data in biology is typically large. Hence, collecting the data on the scale comparable to other deep learning application areas, such as computer vision and speech understanding, is prohibitively expensive and datasets are, consequently, small. Training models of high complexity on small datasets can result in overfitting – model tries to over-explain the observed data and has a bad prediction accuracy on unobserved data. The number of parameters in the model is often regarded as the complexity of the model. However, deep learning models usually have thousands to millions of parameters, and they are still capable of yielding meaningful results and avoiding over-fitting even on modest datasets. To explain this phenomenon, I proposed a method to estimate the degrees of freedom – a proper estimate of the complexity – in deep learning models. My results show that the actual complexity of a deep learning model is much smaller than its number

of parameters. Using this measure of complexity, I propose a new model selection score which obviates the need for cross-validation.

Another concern for deep learning models is the ability to extract comprehensible knowledge from the model. In linear models, a coefficient corresponding to an input variable represents that variables influence on the prediction. However, in a deep neural network, the relationship between input and output is much more complex. In biological and medical applications, lack of interpretability prevents deep neural networks from being a source of new scientific knowledge. To address this problem, I provide 1) a framework to select hypotheses about perturbations that lead to the largest phenotypic change, and 2) a novel auto-encoder with guided-training that selects a representation of a biological system informative of a target phenotype. Computational biology application case studies were provided to illustrate the success of both methods.

ACKNOWLEDGEMENTS

I would like to thank my advisors, Vladimir Jojic and Jeff Dangl, for their input and guidance.

I would also like to thank my committee members, Leonard McMillan, Marc Niethammer, and Mohit Bansal, for their feedback and advice.

Additionally, I would like to thank my labmates, as their company and discussion made my time more fruitful and enjoyable: Yeu-Chern Harn, Gabriel Castrillo, Sur Herrera Paredes, Omri Finkel, Isai Salas Gonzlez, David Furman, Petter Brodin, Terry Law, Sarah Grant, Ke Wang, Yi Hong.

I would like to thank my parents, who encouraged me to explore, create, tinker, and learn.

Finally, I would like to thank my wife, Shan, and my beloved daughters, Crystal and Hermione, for their continued support and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DoF	Degrees of Freedom
AIC	Akaike Information Criterions
BIC	Bayesian Information Criterions
DNN	Deep Neural Network
SdA	Stacked De-noising Auto Encoder
QIP	Quadratic Integer Programming
SynComs	Microbiome Synthetic Communities
Pi	Inorganic orthophosphate
NN	Neural Network
LM	Linear Model
INT	Linear Model with Interaction Features
GAE	Guided-Autoencoder
PCA	Principal Component Analysis
AE	Autoencoder
MSE	Mean Square Error

CHAPTER 1: INTRODUCTION

Extracting knowledge from natural observed data has a long history, which can be traced back when human's measured the sunlight angle and related it to the year cycle. People have long been using manual analysis to identify the essential truth behind the raw data.

Nowadays, with the advances in modern technologies, millions, billions and trillions of data samples can be generated every day. Modern machine learning technology enables us to learn useful and complex information and knowledge from these massive data, which could never be done before.

A machine learning model can be regarded as a function that captures the relationships among different variables. It can be either supervised learning, where some target variables are predicted from a given set of input variables, like LASSO (Tibshirani, 1996); or unsupervised learning, which learns the relationship among a set of variables, like Principle Component Analysis (Jolliffe, 2002). Generally, these methods learn coefficients for mapping input variables. These coefficients indicate the influence of the inputs to the output. By analyzing these coefficients, the relationship between input and output can be easily understood.

However, most relationships in the world are not purely linearly related. Hence, many nonlinear methods were developed to capture those complex relationships, like Random Forests (Breiman, 2001), which aggregate the predictions of multiple learners to generate the output, and Deep learning (LeCun et al., 2015), which is a class of methods based on multiple layers of artificial neural networks.

Despite the success of deep learning in many tasks, there are still several concerns with this method that prevents people from further utilizing its power. One of the concerns is over-fitting. In machine learning, over-fitting is a problem where a model tries to over-explain the observed

data and makes bad predictions on unobserved data. It usually happens when the number of parameters in the model is much larger than the number of training samples, where the model can utilize all the parameters to memorize the exact value of all the samples. In order to avoid over-fitting, regularization methods like LASSO (Tibshirani, 1996), LARS (Efron et al., 2004) were used for linear models. For the purpose of deep learning models regularization, denoising-auto-encoders(Vincent et al., 2008), and drop-out (Srivastava et al., 2014) can be used to prevent over-fitting.

In order to compare the performance of different models, people used cross-validation (Golub et al., 1979), where data is split into different folds and a model is trained on all but one of the folds and validated on the rest. Akaike Information Criterion (Akaike et al., 1973), and Bayesian Information Criterion (Schwarz et al., 1978) were also used, where a compensation term related to the number of parameters in the model was added to the training loss. However, deep learning models usually contain thousands to millions of parameters and they can still yield meaningful results and avoid overfitting even on modest datasets. The number of parameters in the model could be an overestimate of the complexity of the deep learning models. Hence, it is important that a proper measurement of the complexity is used to make sure the model is well-trained.

Another concern with regard to deep learning models is the model's interpretability. In many applications, like biology, medicine, or any safety-related tasks like auto-piloting, prediction accuracy is not the most important factor. People need to understand the mechanism to trust that the model is working as assumed. On the other hand, people need guidance and conclusions from the model for further actions and experiments. For nonlinear methods like deep learning, the relationship between input and output can be very complex. Unlike linear models, the influence of a specific input factor can be very different when the context of other input factors are changing. These issues limit the use of deep learning.

There are many challenges with deep learning for biology data. First, it is still very expensive to obtain data samples in the field of biology compared to other fields like social media images and texts. With a limited amount of data, it is important that the model can be properly trained without

over-fitting. On the other hand, it is important to efficiently design the experiments to be carried out, as the cost of conducting experiments in biology can be very expensive. Also, the information and knowledge extracted from deep learning models should be robust and interesting in order to be practically tested. There are many undesired interferences in biological data during the collection, like batch-effects, measurement noise. It is important that the undesired effects are treated properly in order to train a meaningful model.

In this dissertation, my works will be focused on exploring the capability of extracting valuable information from deep learning models for computational biology data.

1.1 Thesis Statement

The information in deep models can be extracted using several different approaches – the complexity in deep models can be properly measured by its degrees of freedom; the hypotheses in deep models can be extracted using a hypotheses generation framework; the informative representations in deep models can be learned by guided-autoencoders, and this information could help us better understand and trust the deep models for computational biology.

1.2 Outline of Contributions

This work provides several novel methods for extracting information from deep learning models. All the works are published or in the submission process. These contributions include:

Degrees of freedom in deep neural networks This work focuses on measuring the complexity of deep neural networks from their degrees of freedom – the sum of the sensitivity of the output labels during the training process. Degrees of freedom can be used as a proper complexity measurement. The methods were applied to both a synthetic and two image classification tasks. The degrees of freedom can be used as a better measurement than the number of parameters when evaluating the complexity of the model. With proper regularization, deep

neural networks could have complexity much smaller than their parameters. This work is described in Gao and Jojic (2016), and detailed in Chapter 3.

Selecting hypotheses from deep neural networks I propose a framework to extract testable hypotheses from a deep neural network. To identify potentially testable hypotheses, the perturbation level and expected change associated with the hypothesis can be measured. Valuable hypotheses are the ones with small perturbation level in the input and large expected change in response. An economical hypotheses selection algorithm is introduced to efficiently select hypotheses with total cost constraints. Such a framework enables us to generate testable hypotheses from deep neural networks. The proposed hypotheses selection framework has been applied to a microbial synthetic community design case study with biological validation. This work is currently under submission Paredes et al. (2017), and detailed in Chapter 4.

Learning informative representations using guided-auto-encoders I propose a guided-auto-encoder model to learn informative representations from the data with regard to a target variable of interest. The learned representation can be used to reconstruct the data with good accuracy, and predict the target label accurately. This method can be used to provide an informative summarization of the data. Depending on the requirement, representations with different levels of informativeness and compression can be learned by tuning the guided-ratio factor. The method was applied to an immunology data set to construct a health-related immune measurement metric. This work is currently under submission Furman et al. (2017), and detailed in Chapter 5.

CHAPTER 2: BACKGROUND AND RELATED WORK

Comparing to naïve linear models and handcrafted features, deep models can be used to automatically extract nonlinear relationships and learn complicated concepts. However, it is hard to measure the complexity of the model and extract useful information from the model in comparison to shallow models. In this chapter, I will review the background of deep learning, complexity estimation and model selection methods, and recent advances in model interpretation for deep learnings.

2.1 Deep Learning

Deep learning can be regarded as a class of machine learning algorithm with neural networks with multiple layers of nonlinear transformations. It was inspired by the neuron architecture in the brain. Neural networks – the most basic architecture in deep learning – have been studied for many decades (Rumelhart et al., 1988; Hinton, 1990). But the neural networks were not able to outperform shallow models like Support vector machines (Boser et al., 1992) until 2006. Breakthroughs in 2006 enabled deep neural networks to be trained efficiently and to outperform shallow models (Hinton and Salakhutdinov, 2006). This allows deep learning based methods to beat the records in many fields like computer vision (Krizhevsky et al., 2012; Farabet et al., 2013), speech recognition (Hinton et al., 2012). A review of the deep learning method and application can be found (LeCun et al., 2015).

There are many deep-learning methods for computational biology application (Min et al., 2016; Angermueller et al., 2016). For example, DeepBind (Alipanahi et al., 2015) use deep convolution network to achieve the benchmark for predicting sequence specificities for DNA-and RNA-binding proteins. DeepSEA (Zhou and Troyanskaya, 2015) also applied deep convolutional neural networks

to predict effects of noncoding variants. Gene expression inference using Deep learning has also been used in (Chen et al., 2015).

2.2 Basic Neural Network

The neural network is the most basic structure in deep learning. In this section, I will introduce the definitions and basic settings of the neural networks.

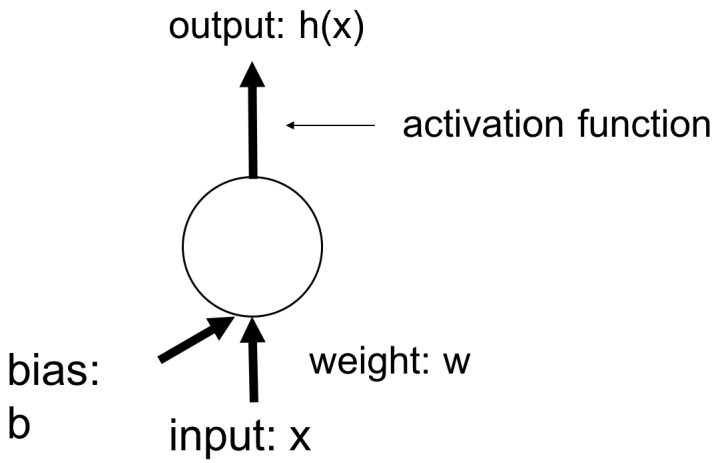

Figure 2.1: An example of a single neuron

The basic unit in a neural network is a neuron, as shown in Figure 2.1. Define the input as a p dimensional vector $\mathbf{x} \in \mathbb{R}^p$, the output of the neuron is:

$$h(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x} + b),$$

where $\mathbf{w} \in \mathbb{R}^p$ is the weight vector, b is a scalar bias term, $g(\cdot)$ is a non-linear activation function. There are three major activation functions: 1) sigmoid function

$$g(x) = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

2) hyperbolic tangent function:

$$g(x) = \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

and 3) rectified linear unit:

$$g(x) = \text{ReLU}(x) = \max(x, 0).$$

The sigmoid unit has an output between $(0, 1)$, hence it is often used to model the output as a probability. The hyperbolic tangent function has an output range between $(-1, 1)$ with a slope of 1 at the origin, which makes it "locally linear" when the input is very small. Comparing to the sigmoid and hyperbolic tangent functions, the rectified linear unit does not suffer from the gradient vanishing problem when the input becomes too small or too large. A visualization of the three functions is shown in Figure 2.2.

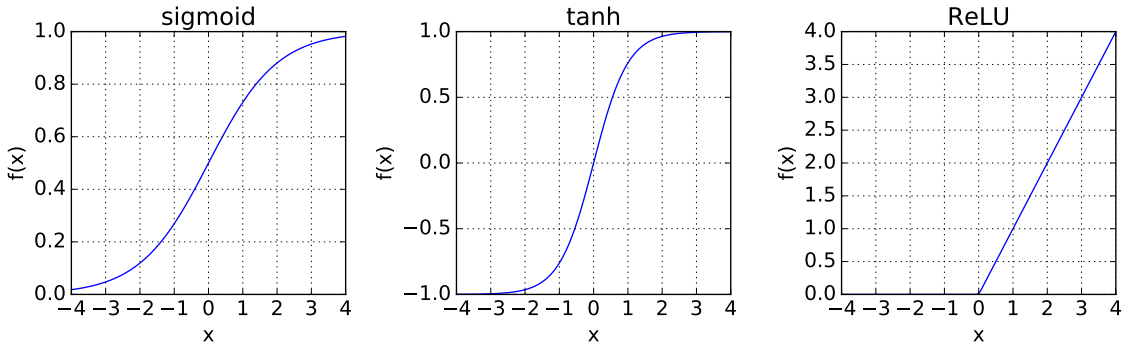

Figure 2.2: Three common non-linear activation functions.

Multiple neurons can be stacked together to form a neural network hidden layer, as shown in Figure 2.3. Hence, the output of a hidden layer of m neurons can be written as:

$$\mathbf{h}(\mathbf{x}) = g(\mathbf{W}\mathbf{x} + \mathbf{b}),$$

where $\mathbf{W} \in \mathbb{R}^{m \times p}$ is the weight matrix and $\mathbf{b} \in \mathbb{R}^m$ is the bias vector.

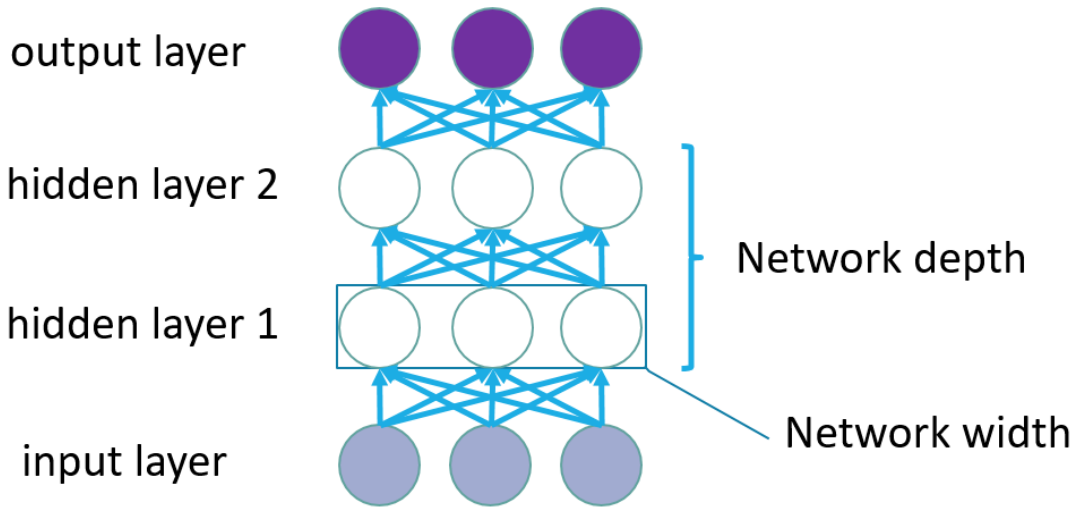

Figure 2.3: An example two-layer neural network. The number of hidden layers is defined as the depth of the network. The number of nodes in the hidden layer is defined as the width of the network, if the hidden layers have the same size.

Multiple layers of neurons can be stacked on top of each other to form a deep neural network. Let $\mathbf{h}_l(\mathbf{x})$ be the output of lth layer:

$$\mathbf{h}_l(\mathbf{x}) = g(\mathbf{W}_l \mathbf{h}_{l-1} + \mathbf{b}_l),$$

where $\mathbf{W}_l$ and $\mathbf{b}_l$ are the weight matrix and bias vector of lth layer. For convenience, I define $\mathbf{h}_0(\mathbf{x}) = \mathbf{x}$. The depth of a deep neural network is the number of hidden layers in the network. In general, the output of the neural network with depth L can be written as:

$$f_{NN}(\mathbf{x}) = \mathbf{h}_L(\mathbf{x}).$$

Given a training dataset $\{(\mathbf{x}_i, y_i)\}$, in order to train a neural network, one can use stochastic gradient descend to optimize the loss on a training dataset:

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \sum_i \mathcal{L}(f(\mathbf{x}_i, \boldsymbol{\theta}), y_i) + R(\boldsymbol{\theta}),$$

where $\boldsymbol{\theta}$ is the set of all parameters in the network. $\mathcal{L}$ is the loss term and R is the regularization term. Depending on the requirement of the task, different loss functions and regularizations can be used.

2.3 Complexity Estimation and Model Selection

Model selection is one of the key tasks in machine learning, as the performance of a method on training data is an optimistic estimate of its general performance. Efron (2004) provided an estimate of optimism, the difference of error on test and training data, and related it to a measure of a model's complexity deemed the effective degrees of freedom. This result reflects Occam's razor since models with higher degrees of freedom tends to have higher optimism. Degrees of freedom, defined as parameter counts, have been frequently used in model selection. However, even in linear models, the number of parameters is not a good indicator of model's complexity. Straightforward examples of this behavior are models fit using sparsity penalties. In that context, degrees of freedom is related to the number of non-zero parameters instead of total parameter count.

Ye (1998) introduced the concept of Generalized Degrees of freedom (GDF) for complex modeling procedures with Gaussian distributed outputs. GDF is defined based on the sensitivity of the fitted values to the perturbations in observed values. Efron (2004) provided a framework for estimating degrees of freedom for modeling procedures with output in exponential family distribution.

In order to estimate degrees of freedom in deep neural networks for classification problems, where the outputs can be regarded as a categorical distribution, Efron's results can be extended to the context of multinomial logistic regression. Similar to Ye's GDF, the computation of the degrees of freedom involves assessing network's changes in output as a result of perturbation of the training data. The more sensitive the network's output to the perturbation, the more degrees of freedom it has. A recent work utilizes Ye's GDF to estimate the degrees of freedom in ensemble regression (Reeve and Brown, 2017).

The prior work on the model complexity is rich, and I briefly review some key contributions. Bayesian Information Criterion (BIC) (Schwarz et al., 1978) and Akaike Information Criterions (AIC) (Akaike, 1974) are most commonly used techniques for model selection. Both aim to construct an estimate of the test log-likelihood by correcting the training set log likelihood with terms dependent on the number of parameters in the model in order to produce a score that is a less biased estimate of test log-likelihood. The weighting of the parameter count is different, BIC depends on the sample size, and AIC uses a constant. BIC applied to the family of models that contain the true model is consistent with the limit of the data. AIC, with some mild constraints, guarantees the selection of a model with least square error, among models that do not include the true model.

Crucial to the practical application of these methods is the correct count of parameters. Bayesian model selection elegantly avoids the need to specify the complexity of the network by evaluating evidence, a marginal probability of the data given the model. This approach marginalizes over all of the parameters, making models of different parameterizations comparable. The size of the parameter space directly impacts the evidence through this integration, as the prior on parameters gets spread thinly across high dimensional spaces. Unfortunately, the cost of computing such integrals is often prohibitive, but the models selected using these techniques have been shown to be very competitive. (MacKay, 2003; Neal, 1996; Guyon et al., 2004).

Kolmogorov-Chaitin complexity (Kolmogorov, 1965) describes dataset complexity in terms of a program that recapitulates the data. Generation of task-specific neural networks using algorithmically simple programs was explored by Schmidhuber (1997). Networks whose parameters could not be captured by a simple program were avoided. A related method of Minimal Description Length reflects the desire for compact representation of the data. Its application (Hinton and Zemel, 1994) shows how the trade-off between the data and parameter compression can lead to an objective for training auto-encoders.

Degrees of freedom of linear model fits with Lasso-type penalties have been analyzed, e.g. Lasso (Zou et al., 2007), Fused Lasso (Tibshirani et al., 2005) and Group Lasso (Vaiter et al., 2012).

The number of predictors and the number of degrees of freedom greatly differ due to the imposed sparsity and weight tying. Recent results on degrees of freedom for non-continuous procedures such as best subset regression and forward stagewise regression (Janson et al., 2015) highlight challenges in determining the complexity of these procedures as the estimators can be discontinuous. Research on Stein's Unbiased Risk Estimate (SURE) has yielded model selection techniques (Stein, 1981) as well as algorithms for their estimation (Ye, 1998; Ramani et al., 2008). Generalization of SURE to exponential families has been proposed by Eldar (2009). However, its focus is on estimating parameter risk instead of prediction error. In linear models, the two neatly coincide. But this does not carry over to logistic regression and more broadly sigmoidal neural networks.

2.4 Interpretability in Deep Learning

A common problem in most deep learning based methods is the interpretation. For example, how can we extract useful hypothesss or knowledge from the learned deep neural networks? For convolutional neural networks, displaying the learned motif filters (Alipanahi et al., 2015) was used. However, it is hard to compare and capture the contribution from different motif filters. For feedforward neural networks, Li *et al*. (Li et al., 2015) proposed a regularization layer between input nodes to the network to perform feature selection on input features.In a network with many output nodes, it is possible that each input node is responsible for an output node, which makes it hard and unreasonable to exclude any input features in the final model. Tan *et al*. (Tan et al., 2014, 2016) proposed an unsupervised method using a one layer denoising autoencoders to learn higher order representations from gene expression data. They used the learned weights between the links to distill the relationship between higher-order representations and raw features. Generalizing their approach to models with more than one layers can be potentially hard. Visualization techniques are also proposed (Chen et al., 2015) to visualize learned knowledge in deep neural networks by highlighting significant weights in the neural network.

One problem in interpreting neural networks is the re-parameterization problem. Basically, the nodes in each hidden layer are interchangeable. The weights and meaning of a hidden node are

usually determined by random seeds used in initialization. Therefore, interpreting from a single node is useless. Rippel *et al*. (Rippel et al., 2014) proposed a nested-dropout method to learn ordered representations for auto-encoders to avoid this problem.

There are many works on model interpretation with gradient-based methods. Simonyan *et al*. (Simonyan et al., 2013) visualized the absolute gradient of the output in a network w.r.t. each input pixel in order to interpret which part of the image can influence the output most. Wang *et al*. (Wang et al., 2016) introduced the Extended Data Jacobian Matrix to analyze neural networks. A variant version of a gradient-based approach – guided backpropagation – takes account of gradients with positive errors (Springenberg et al., 2014).

There are many very recent works aimed at fairly describeing the contribution of the change in the output to different input factors. DeepLift (Shrikumar et al., 2017) use a back-propagation on the change in each layer to obtain a better contribution interpretation. SHAP (Lundberg and Lee, 2017) provides a unified framework for interpreting models based on Shapely values, which is a solution concept from cooperative game theory.

CHAPTER 3: IDENTIFYING COMPLEXITY IN DEEP NEURAL NETWORKS

3.1 Introduction

Understanding the complexity of deep learning models is important. It provides us evidence if the model is properly trained without over-fitting. In this work, I will show that the complexity in Deep neural networks can be better measured using the sum of the sensitivity of the output to the perturbation in training labels rather than the number of parameters. This work has been published as a conference paper in The Conference on Uncertainty in Artificial Intelligence (UAI) 2016 (Gao and Jojic, 2016).

3.2 Degrees of Freedom for Categorical Distribution

In this section, I derive the definition of degrees of freedom for a categorical distribution from the optimism according to (Efron, 2004). Then, I introduce an efficient Monte-Carlo sampling based method (Ramani et al., 2008) to estimate degrees of freedom.

3.2.1 Definitions

We focus on models aimed at multi-class classification task. The data is assumed to be composed of features $\mathbf{X} \in \mathbb{R}^{n \times p}$, where n is the number of observations and p is the number of features, and output labels $\mathbf{y}$ range over k categories. I will denote the categorical distribution with $\mathcal{C}(\cdot)$. Categorical distribution over k categories can be parameterized using a vector of non-negative values with a sum of 1. Sample label y_i are regarded as realization of categorical random variables for a specific parameter vector $\boldsymbol{\mu}_i$. Hence $y_i \sim \mathcal{C}(\boldsymbol{\mu}_i)$, where $\boldsymbol{\mu}_i = [\mu_{i1}, \mu_{i2}, \ldots, \mu_{ik}]$ is the **true probability** of sample y_i being in each class. $\mu_{ic} \in [0, 1]$ and $\sum_{c=1}^{k} \mu_{ic} = 1$. Members of the

exponential family follow the form:

$$f(\mathbf{p}_i|\boldsymbol{\mu}_i) = r(\mathbf{p}_i)\exp\{\boldsymbol{\theta}(\boldsymbol{\mu}_i)^T\mathbf{p}_i - A(\boldsymbol{\mu}_i)\}$$

where $\mathbf{p}_i$ is the vector of sufficient statistics for sample i, $\boldsymbol{\theta}(\boldsymbol{\mu}_i)$ is the vector of natural parameters, $r(\mathbf{p}_i)$ is the base measure, and $A(\boldsymbol{\mu}_i)$ is the log-partition function.

For a categorical distribution with parameter $\boldsymbol{\mu}_i$, we have $\mathbf{p}_i = h(y_i) = [\delta(y_i - 1), \ldots, \delta(y_i - k - 1)]^T$, where $\delta(\cdot)$ is the Kronecker delta function, $\delta(a) = 1$ if $a = 0$, $\delta(a) = 0$ if $a \neq 0$. In other words, $\mathbf{p}_i$ is a vector of the **observations** of sample i being in each class. Base measure is $r(\mathbf{p}_i) = 1$; natural parameters are $\theta_c(\boldsymbol{\mu}_i) = \ln \mu_{ic} - \ln(1 - \sum_{l=1}^{k-1} \mu_{il})$, and the log partition function is $A(\boldsymbol{\mu}_i) = \ln(1 + \sum_{c=1}^{k-1} e^{\theta_c(\boldsymbol{\mu}_i)})$. Note that both $\boldsymbol{\mu}_i$ and $\mathbf{p}_i$ are of dimension $k - 1$. Let $\mathbf{P} = [\mathbf{p}_1, \ldots, \mathbf{p}_n]^T$ be the matrix of observations for all sample labels $y_1, \ldots, y_n$.

3.2.2 Optimism in Models with Categorical Distribution

Optimism is the difference between expected test log deviance error and training log deviance error for a model fitting procedure. It is related to the complexity of the model and degrees of freedom is derived from optimism. If the optimism for a modeling procedure can be estimated, we can use it for model selection. (Efron, 2004) provides the derivations of expected optimism for the single parameter exponential family. We follow Efron's approach to derive the definition of degrees of freedom for modeling procedure with output in categorical distribution form.

Given sample input $\mathbf{x}_i$, the output label is assumed to follow a categorical distribution $y_i \sim \mathcal{C}(\boldsymbol{\mu}_i)$. Let $\mathcal{L}(\cdot)$ be the training procedure that fits the estimated probability for the training input labels. Let $\hat{\boldsymbol{\mu}}_i = \mathcal{L}(\mathbf{p}_i)$ be the **estimated probability** for sample i from observations $\mathbf{p}_i$. The log deviance error for $\hat{\boldsymbol{\mu}}_i$ and $\mathbf{p}_i$ is:

$$\begin{aligned} \text{err}_i &= -2\log f(\mathbf{p}_i|\hat{\boldsymbol{\mu}}_i^T) \\ &= -2[\boldsymbol{\theta}(\hat{\boldsymbol{\mu}}_i)^T\mathbf{p}_i - A(\hat{\boldsymbol{\mu}}_i)] \end{aligned}$$

Suppose another sample y_i^0 is drawn from the same distribution as y_i, $y_i^0 \sim \mathcal{C}(\mu_i)$. Let $\mathbf{q}_i = h(y_i^0)$ be the vector of its observations. The expected log deviance error of $\mathbf{q}_i$ using $\hat{\boldsymbol{\mu}}_i$ is:

$$\begin{aligned} \text{Err}_i &= \underset{y_i^0}{\text{E}}\{-2\log f(\mathbf{q}_i|\hat{\boldsymbol{\mu}}_i)\} \\ &= -2[\boldsymbol{\theta}(\hat{\boldsymbol{\mu}}_i)^T\boldsymbol{\mu}_i + A(\hat{\boldsymbol{\mu}}_i)] \end{aligned}$$

The definition of optimism is:

$$\begin{aligned} O_i &= \text{Err}_i - \text{err}_i \\ &= 2\boldsymbol{\theta}(\hat{\boldsymbol{\mu}}_i)^T(\mathbf{p}_i - \boldsymbol{\mu}_i) \end{aligned}$$

Hence, optimism is the difference between log deviance error on the training set and expected log deviance error with respect to the true distribution.

The expected optimism over $y_i \sim \mathcal{C}(\boldsymbol{\mu}_i)$ for the estimated probability $\hat{\boldsymbol{\mu}}_i$ and true probability $\boldsymbol{\mu}_i$ is:

$$\Omega_i = 2\underset{y_i}{\text{E}}\{\boldsymbol{\theta}(\hat{\boldsymbol{\mu}}_i)^T(\mathbf{p}_i - \boldsymbol{\mu}_i).\}$$

As we do not know the true probability $\boldsymbol{\mu}_i$, we cannot compute the expected optimism. However, an approximate measurement using first order Taylor series expansion can be applied. We can approximate $\boldsymbol{\theta}(\hat{\boldsymbol{\mu}}_i)$ by taking the Taylor series expansion at $\mathbf{p}_i = \boldsymbol{\mu}_i$ to obtain:

$$\boldsymbol{\theta}(\hat{\boldsymbol{\mu}}_i) \approx \boldsymbol{\theta}(\mathcal{L}(\boldsymbol{\mu}_i)) + \mathbf{D}^{(i)}(\mathbf{p}_i - \boldsymbol{\mu}_i).$$

$\mathbf{D}^{(i)}$ is the first derivative matrix where each entry $D_{jc}^{(i)} = \frac{\partial\theta_j(\mathcal{L}(\mathbf{v}))}{\partial v_c}|_{\mathbf{v}=\boldsymbol{\mu}_i}$.

Therefore, the approximate expected optimism can be estimated:

$$\begin{aligned}\tilde{\Omega}_i &= 2\underset{y_i}{\mathrm{E}}\{[\boldsymbol{\theta}(\mathcal{L}(\boldsymbol{\mu}_i)) + \mathbf{D}^{(i)}(\mathbf{p}_i - \boldsymbol{\mu}_i)]^T(\mathbf{p}_i - \boldsymbol{\mu}_i)\} \\ &= 2\underset{y_i}{\mathrm{E}}\{\sum_{j=1}^{k-1}\sum_{l=1}^{k-1}(p_{ij} - \mu_{ij})(p_{il} - \mu_{il})D_{jl}^{(i)}\} \\ &= 2\sum_{j=1}^{k-1}\sum_{l=1}^{k-1}\mathrm{cov}(p_{ij}, p_{il})\frac{\partial\theta_j(\mathcal{L}(\mathbf{v}))}{\partial v_l}|_{\mathbf{v}=\boldsymbol{\mu}_i}\end{aligned}$$

The expected optimism can be estimated by assuming $p_i \sim \mathcal{C}(\hat{\boldsymbol{\mu}}_i)$, so:

$$\hat{\Omega}_i = 2\sum_{j=1}^{k-1}\sum_{l=1}^{k-1}\mathrm{cov}(p_{ij}, p_{il})\frac{\partial\theta_j(\mathcal{L}(\mathbf{v}))}{\partial v_l}|_{\mathbf{v}=\hat{\boldsymbol{\mu}}_i}. \tag{3.1}$$

For a categorical distribution, $\mathrm{cov}(p_{ij}, p_{il}) = -\hat{\mu}_{ij}\hat{\mu}_{il}$, if $i \neq j$. $\mathrm{var}(p_{ij}) = \hat{\mu}_{ij}(1 - \hat{\mu}_{ij})$. Therefore, Equation (3.1) can be reduced to:

$$\hat{\Omega}_i = 2\sum_{j=1}^{k-1}\frac{\partial\mathcal{L}_j(\mathbf{v})}{\partial v_j}|_{\mathbf{v}=\hat{\boldsymbol{\mu}}_i}. \tag{3.2}$$

Equation (3.2) for $k = 2$ is exactly the result for the Bernoulli distribution derived in (Efron, 2004). Efron also showed that Eqn (3.2) gives the correct degrees of freedom for maximum likelihood estimation (Efron, 1975). In a p-parameter curved exponential family, we have:

$$\sum_{i=1}^{n}\frac{\partial\mathcal{L}(\mathbf{v})}{\partial v_i}|_{\mathbf{v}=\hat{\boldsymbol{\mu}}_i} = p.$$

Here, I define the degrees of freedom for a classification model estimator $\hat{\boldsymbol{\mu}}_i = \mathcal{L}_i(\mathbf{P})$ on all the data samples $\mathbf{P}$ to be:

$$\mathrm{df} = \sum_{i=1}^{n}\sum_{c=1}^{k-1}\frac{\partial\mathcal{L}_{ic}(\mathbf{P})}{\partial p_{ic}}, \tag{3.3}$$

where i indicates the sample index and c indicates the category index. This definition tells that the degrees of freedom is the sum of each sample's sensitivity of its estimated probability to the perturbations in its observation for all categories.

3.2.3 Degrees of Freedom for Model Selection

As degrees of freedom is related to the expected optimism, degrees of freedom can be used for model selection. According to Equation (3.2) and (3.3), the relationship between expected test and training log deviance errors is:

$$\sum_{i=1}^{n} \mathrm{E}_{y_i}\{\mathrm{Err}_i\} = \sum_{i=1}^{n} \mathrm{E}_{y_i}\{\mathrm{err}_i\} + 2\mathrm{df}. \tag{3.4}$$

Equation (3.4) is very similar to Akaike Information Criterions (AIC) (Akaike, 1974):

$$\mathrm{AIC} = \sum_{i=1}^{n} \mathrm{err}_i + 2k, \tag{3.5}$$

where k is the number of parameters. I refer to 2df in Equation (3.4) and $2k$ in Equation (3.5) as "complexity correction" for training log deviance error. In simple linear regression models, $\mathrm{df} = k$, and the complexity corrections are the same. However, in complex models such as deep neural networks, simply counting number of parameters can result in an overestimate of the expected test log deviance error. Therefore, I introduce DoFAIC for model selection:

$$\mathrm{DoFAIC} = \sum_{i=1}^{n} \mathrm{err}_i + 2\mathrm{df}. \tag{3.6}$$

DoFAIC uses degrees of freedom instead of the number of parameters for complexity correction. Hence, the basic assumption is that DoFAIC can produce a better criterion for model selection than Naïve AIC.

3.2.4 Monte-Carlo Estimation for Degrees of Freedom

For most practical estimators of the model's predictions with respect to the data derivatives, $\frac{\partial \mathcal{L}_{ic}(\mathbf{P})}{\partial p_{ic}}$ are not available in closed form. For example, fitting multinomial logistic regression using stochastic gradient descent with adaptive learning rates requires a fairly sophisticated derivation which accounts for changes in step-sizes as a result of data perturbation. For deep neural networks, this difficulty grows due to the use of back-propagation. In this paper, I used a sampling-based method to efficiently estimate

Monte-Carlo Estimation A theoretical result for a stochastic estimate of the degrees of freedom of nonlinear estimators has been proposed by Ramani et al. (2008). I restate the key result from that paper here.

Theorem 3.1. *Let* b *be a zero mean i.i.d. random vector (that is independent of y) with unit variance and bounded higher moments. Then*

$$\sum_i \frac{\partial f(\mathbf{y})}{\partial y_i} = \lim_{\epsilon \to 0} \underset{\mathbf{b}}{E} \left[\mathbf{b}^T \left(\frac{f(\mathbf{y} + \epsilon \mathbf{b}) - f(\mathbf{y})}{\epsilon} \right) \right]$$

provided that f *admits a well-defined second-order Taylor expansion.*

The prediction in a neural net via forward pass is a smooth function of the observations of training labels. I will abbreviate "differentiable with respect to observations" as d.w.r.t.o. Sigmoid and soft-max are smooth functions of their inputs. The cross-entropy loss is a multivariate function that depends on data and weights, and all of its partial derivatives exist. For simplicity, assume that the network is trained using gradient descent. Each update of the network's parameters is a linear combination of previous weights and a gradient of the loss. Assuming that the initial weights d.w.r.t.o. and loss is smooth then the update yields weights that are d.w.r.t.o. Random initialization and pre-training both yield initializations that are independent of observations, hence the partial derivatives of the initial weights with respect to observations are 0. By induction, gradient descent, at any iteration, yields weights that are d.w.r.t.o. Forward pass through sigmoidal network yields

estimated probabilities which are smooth with respect to observations. Thus, the Taylor expansion required by the above theorem exists.

Using this theorem, the derivative of a function $\frac{\partial f(x)}{\partial x}$ can be estimated by perturbing the inputs. I applied a modified version of the method (Ramani et al., 2008) for categorical distribution. Applying random perturbation to the observations can be used to estimate the degrees of freedom:

$$\begin{aligned} \text{df} &= \sum_{i=1}^{n} \sum_{c=1}^{k-1} \frac{\partial \mathcal{L}_{ic}(\mathbf{P})}{\partial p_{ic}} \\ &= \lim_{\epsilon \to 0} \left\{ \mathrm{E}_{\mathbf{B}} \left[\sum_i \sum_c b_{ic} \left(\frac{\mathcal{L}_{ic}(\mathbf{P}+\epsilon\mathbf{B}) - \mathcal{L}_{ic}(\mathbf{P})}{\epsilon} \right) \right] \right\}, \end{aligned}$$

where $\mathbf{B}$ is a zero-mean i.i.d. random matrix with unit variance and bounded higher order moments. Therefore, df can be approximated with T independent samplings of $\mathbf{B}^{(t)}$:

$$df \approx \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{n} \sum_{c=1}^{k-1} b_{ic}^{(t)} \left(\frac{\mathcal{L}_{ic}(\mathbf{P} + \epsilon \mathbf{B}^{(t)}) - \mathcal{L}_{ic}(\mathbf{P})}{\epsilon} \right), \tag{3.7}$$

where ϵ is a small value. In our experiments, I choose $\epsilon = 10^{-5}$. To better estimate the sensitivity, we can use the average of multiple runs as the final estimation. The algorithm for estimating degrees of freedom is summarized in Algorithm 1.

Algorithm 1 Monte Carlo algorithm for computing degrees of freedom of a multi-class classifier

Input: training data $\mathbf{X} \in \mathbf{R}^{p \times N}$, $\mathbf{y} \in \{1, 2, .., k\}^N$

1: Compute observations matrix $\mathbf{P} = h(\mathbf{y})$
2: Train model on $\mathbf{X}$ and $\mathbf{P}$
3: Compute estimated probabilities for each sample $\mathcal{L}(\mathbf{P})$
4: Sample entries of $\mathbf{B}^{(t)} \in \mathbf{R}^{p \times k}$ from zero-mean, unit variance normal distribution
5: Train model on $\mathbf{X}$ and $\mathbf{P}^{(t)} = \mathbf{P} + \epsilon \mathbf{B}^{(t)}$
6: Using trained model compute estimated probabilities for each sample $\mathcal{L}(\mathbf{P}^{(t)})$;
7: Repeat 4-6 for T times;
8: Calculate df from Equation (3.7)

Note that training on original and perturbed observations matrices can be performed in parallel. Finally, I also derived analytical derivatives for stochastic gradient descent learning which yields the same degrees of freedom as the algorithm presented above. However, this method requires

maintenance of partial derivatives of each parameter with respect to each sample's observations. Such storage requirements make this method impractical for real world applications.

Variance reduction For deep neural networks, training takes a considerable amount of time. In order to estimate degrees of freedom in a reasonable computational time, we used a variance reduction technique – common random numbers – during Monte-Carlo sampling. When comparing the degrees of freedom on a specific data, fixed $\mathbf{P}$, for several different fitting procedures, we used the same perturbation matrix $\mathbf{B}$ for all the models. I used the same random seed for all models throughout the training. For example, in deep neural network training, we use the same random seed to initialize weights and bias; during pre-training with denoising-autoencoders, I use the same random seed for drop-out and input corruptions. For stochastic gradient descent methods, I use the same mini-batches splittings during training. In the experiment, I found that degrees of freedom can be estimated well enough using just one perturbed copy of the data when using these variance reduction techniques.

3.2.5 Degrees of Freedom in Multinomial Logistic Regression

In order to validate the above algorithm in a setting with known degrees of freedom, I perform an empirical analysis of the degrees of freedom in different multinomial logistic regression models.

An i.i.d. zero mean unit variance random design matrix $\mathbf{X}$ with $n = 100$ samples and $p = 20$ features is generated. Each sample is presented as $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{ip}]$. With $k = 4$ classes, I generated a random weight matrix $\mathbf{W} \in \mathbb{R}^{p\times k}$, where each entry $w_{ic} \sim \mathcal{N}(0, 1)$. I generate each label from $y_i = \text{argmax}_j\, \mu_{ij}$, where $\boldsymbol{\mu}_i = e^{\mathbf{x}_i \mathbf{W}}$.

5 models were fitted using multinomial logistic regression. In the ith model, we only use the first $2i$ features in $\mathbf{X}$ to fit. Therefore, the ith model only contains $2(i + 1)(k - 1)$ parameters and the degrees of freedom are equal to the number of parameters. We perform 5 Monte-Carlo degrees of freedom estimates for each model.

I plot degrees of freedom in Figure 3.1(a). In each plot, the blue line is the mean of the five Monte-Carlo estimates. Error bar represents the standard error. Degrees of freedom are very close

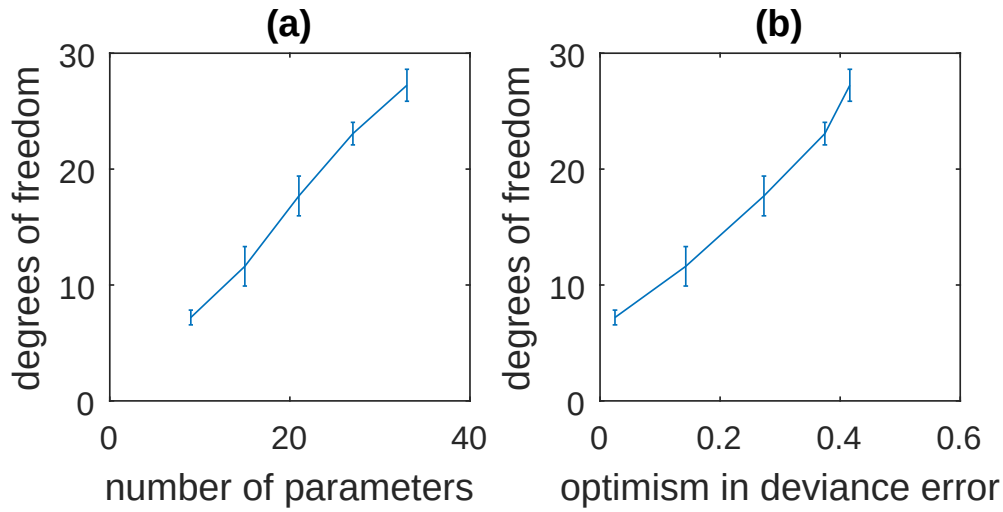

Figure 3.1: **(a)** Comparison between degrees of freedom estimates in multinomial logsitic regression and the true number of parameters used in the model. **(b)** Comparison between degrees of freedom estimates in multinomial logsitic regression and the optimism in log deviance error.

to the number of parameters we used in the model. The standard error for Monte-Carlo estimate is small.

I also randomly generated 1000 samples for testing. Optimism is calculated by the difference between average testing log deviance error and training log deviance error. The degrees of freedom and optimisms for all 5 models are plotted in Figure 3.1(b). It shows that the optimism has a linear relationship with degrees of freedom, as expected.

3.2.6 Degrees of Freedom of an XOR Network

I generated a small synthetic example using exclusive-or (XOR) operator, where XOR$(a, b) = 0$ if $a = b$, and XOR$(a, b) = 1$ if $a \neq b$. Given an input $x_1, x_2 \in \{0, 1\}$, the output $y = XOR(x_1, x_2)$, a model of XOR operator is expected to be learned. In general, building a neural network with two hidden nodes as shown in Figure 3.2 and weights in Table 3.1 is enough to learn a perfect XOR classifier.

A network that trained properly should have weight matrix with form in Table 3.1. If x contains no noise, F, a multiplier, can be infinitely large to achieve perfect estimation. Therefore, we set y to be 0.9 instead of 1.

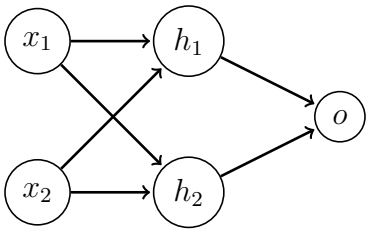

Figure 3.2: A Neural Network with 2 Hidden Nodes

Table 3.1: An XOR Network

x_1	0	1	0	1
x_2	0	0	1	1
$h_1 = \sigma(F(-0.5 + x_1 - x_2))$	0	1	0	0
$h_2 = \sigma(F(-0.5 - x_1 + x_2))$	0	0	1	0
$y = \sigma(K(-0.5 + h_1 + h_2))$	0	1	1	0

I train networks with different structures on XOR data using back-propagation and estimate their degrees of freedom using Monte-Carlo method. Even though there are 9 parameters in the network, I found that the degrees of freedom for all learned models are 4. Note that the symmetry in weights of the inputs to the two hidden nodes, eliminates degrees of freedom, as does implicit tying of the weights of inputs to the output node. To give an intuition why this tying occurs, note that the predominantly correctly labeled data drives the network to keep the weights close to each other. Hence, a small perturbation in the labels can affect multiple weights simultaneously but does not disturb their balance. This observation encourages us to investigate deeper models.

3.3 Degrees of Freedom in Deep Neural Networks

In this section, I investigate degrees of freedom in deep neural network models. From the XOR example, we know that the degrees of freedom in a network is not equal the number of parameters in the model. The structure of the network and different regularization techniques will impact degrees of freedom.

3.3.1 Terminologies and Settings

In the following experiments, I explore deep networks trained to solve larger classification problems. Each of the networks takes a real-valued vector $\mathbf{x}_i \in \mathbb{R}^{p\times 1}$ as input and outputs the probability $\hat{\boldsymbol{\mu}}_i$ for this sample being in one of k categories. Sigmoid activation functions are used for all the hidden nodes and a soft-max layer is used in the last layer. The number of hidden layers is called "**depth**" of the network. I only consider networks with an equal number of units in each hidden layer, and we call this number "**width**" of the network. Next, I investigate degrees of freedom in networks with different width and depth.

Stacked-Auto-Encoder (SdA) pre-training I used SdA (Vincent et al., 2010) to pre-train the neural network with the input dataset, as unsupervised pre-training helps the network to achieve a better generalization from the training data on supervised tasks (Erhan et al., 2010). In the denoising autoencoder, **corruption** is used in layer-wised pre-training. The corruption is introduced by zeroing out input to the auto-encoder with a certain probability. The chosen probability of corruption is called **corruption rate**. **Dropout** (Srivastava et al., 2014) is also used during the pre-training of SdA, where the output of hidden units are randomly zeroed with probability, which is called **dropout rate**. Assume that increasing in corruption rate or dropout rate will reduce degrees of freedom as they provide more regularization to the network.

Weight-decay I used a weight decay penalty on the sum of the squares of all the weights in the network during both pre-training and fine-tuning stage. Adding this penalty prevents the network from over-fitting. I refer to the multiplier associated with the sum of squares as **weight decay rate**. The degrees of freedom is expected to drop with increasing weight decay rate.

Implementation All the codes are based on Theano (Bastien et al., 2012; Bergstra et al., 2010) and I ran experiments on a cluster of machines with NVIDIA Tesla compute cards.

3.3.2 Data Sets

I prepared a synthetic dataset and two real datasets MNIST and CIFAR-10 to estimate degrees of freedom.

Synthetic I built a synthetic dataset from a randomly generated network with 30 input nodes, 2 hidden layers with 30 hidden nodes in each, and 4 output nodes. I generated $n = 5000$ random zero-mean unit variance inputs with 30 dimensions. Each layer was fully connected to the previous layer, and I generated weights $w \sim \mathcal{N}(0, 5)$. I used the sigmoid activation function for each hidden layer and a soft-max on the final output. The output sample labels y are then sampled according to the probabilities from the soft-max layer. To get the optimism, I also generated another 5000 samples for test.

MNIST [1] (LeCun et al., 1998) is a benchmark dataset that contains handwritten digit images. Each sample is a 28×28 image from 10 classes. 50000 samples were used for training.

CIFAR-10 [2] (Krizhevsky and Hinton, 2009) is a dataset that contains 32×32 tiny color images from 10 classes. Each sample has 3072 features. 50000 samples were used for training.

3.3.3 Degrees of Freedom and the Structure of the Network

To investigate the degrees of freedom for networks with different structures, I estimated the degrees of freedom for networks with width $[10, 20, \ldots, 100]$ and depth with 1,2,3 and 4, where all the hidden layers have equal widths. I used SdA to pre-train with 0.1 dropout rate and 0.1 corruption rate. I use weight decay penalty $1e - 5$ for both pre-training and fine-tuning. The estimated degrees of freedom are shown in Figure 3.3. The lines represent the degrees of freedom estimate from 1 Monte-Carlo run, and the color of each indicates the depth of the models.

From the results, I found that networks with more width have more degrees of freedom. This is reasonable as increasing width leads to more independence between parameters. However, the

[1] `http://yann.lecun.com/expdb/mnist/`

[2] `https://www.cs.toronto.edu/~kriz/cifar.html`

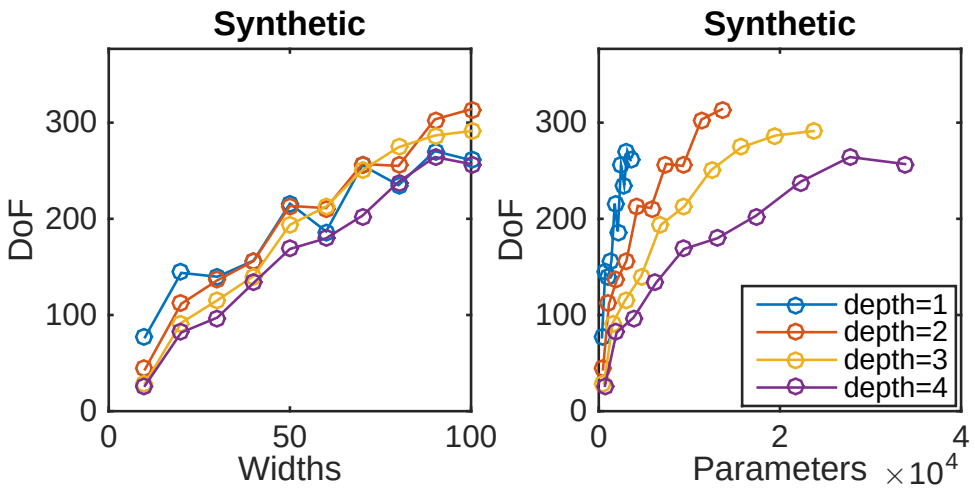

Figure 3.3: Degrees of freedom estimates for different models trained on synthetic data. Left: degrees of freedom vs network width. Right: degrees of freedom vs number of parameters in the network, which is linearly related to the network depth and quadratically related to the number of width.

degrees of freedom in deep networks is generally much less than its the number of parameters. The ratio of the parameters to degrees of freedom is on the order of 10^2. Loosely, one degree of freedom is acquired for 100 parameters. Among the models with the same number of parameters, deeper networks have less degrees of freedom. This observation indicates that the depth of the network has regularization on the complexity.

To further validate our assumption that deeper networks have less degrees of freedom, I also estimated degrees of freedom on the MNIST and CIFAR-10 datasets. We tested networks with width $[100, 300, 500, 700]$, all other settings are the same as in the above synthetic experiment. The results are shown in Figure 3.4. The lines represent the degrees of freedom estimates from a single Monte-Carlo sample and the color of each indicates the depth of that model.

The same conclusions hold for MNIST and CIFAR-10 as for synthetic data. The only difference is increasing depth results in more degrees of freedom than models trained with synthetic data. This could be attributed to the differences of input data size and complexity between the real datasets, MNIST and CIFAR-10, and the much simpler synthetic datasets.

3.3.4 Degrees of Freedom and Regularization Techniques

When training a deep neural network, many practical methods can be used for regularization. I investigate how the different techniques affect the degrees of freedom in the model.

I train networks using the same settings as in Section 3.3.3. In this experiment, I separately trained networks with different settings of penalty rates: corruption rate, dropout rate, and weight decay rate. One rate is changed at a time while keeping rest fixed.

For all three datasets, I trained networks using corruption rates and dropout rates from $[0, 0.1, 0.2, \ldots, 0.9]$, and weight decay rates from 10^{-6} to 10^{-3}. For each setting of regularization parameters, I trained a 3 layer network $[30, 30, 30]$ for synthetic data and $[300, 300, 300]$ on MNIST and CIFAR-10 data. I used one Monte Carlo sample to estimate degrees of freedom in each model. The result is shown in Figure 3.5.

Neither corruption rate nor dropout rate affected degrees of freedom drastically for synthetic data. This is because the input of the synthetic data is generated randomly. Hence, pre-training cannot learn higher-level features for synthetic data. For MNIST and CIFAR-10, I found that both corruption rate and dropout rate have an impact on degrees of freedom . In CIFAR-10, the regularization effect is much larger. These results suggest that the regularization strength from dropout and corruption can be data-specific.

Weight decay penalty has a very strong effect on the degrees of freedom for all three datasets. Further, the weight decay exhibited a highly nonlinear impact on the degrees of freedom, in dramatic contrast to its effect in ridge regression.[3]

3.3.5 Model Selection using Degrees of Freedom

To validate that DoFAIC is a useful criterion for model selection, I compare it against model selection based on error estimates using cross-validation. For brevity, I refer to the cross-validation estimate of error as the cross-validation error. A 5-fold cross-validation experiment is conducted

[3] Ridge regression degrees of freedom scale with $\frac{1}{1+\lambda}$ which is nonlinear but a much tamer multiplier than in neural networks

for Synthetic, MNIST and CIFAR data on models with different network structures learned in Section 3.3.3. I calculated DoFAICs for all the models we trained using Equation (3.6) with the estimated degrees of freedom. I also calculated Naïve AIC using Equation (3.5) with the number of parameters in the network. I compared these estimates against cross-validation errors. The result is shown in Figure 3.6. Each circle in the plot represents a model with a specific structure. The x-axis is the mean cross-validation log deviance error across 5 folds.

Further, I calculate the Spearman rank correlation between cross-validation log deviance errors and DoFAIC/Naïve AIC estimates for each dataset. The result is shown in Table 3.2.

Table 3.2: Spearman Rank Correlation between Cross-validation error and DoFAIC/Naïve AIC

Dataset	DoFAIC ρ	Naïve AIC ρ
Synthetic	0.9865	-0.6711
MNIST	0.9853	-0.9471
CIFAR-10	0.9941	-0.7824

DoFAIC is very consistent with cross-validation error. Naïve AIC, on the other hand, exhibits a negative correlation with cross-validation error due to highly non-linear behavior. This is because Naïve AIC overestimates the complexity of the model by using the large number of parameters in the network. The actual complexity in deeper and larger networks are much less than the number of parameters.

For all three datasets, both DoFAIC and cross-validation chose the same model. This indicates that DoFAIC can be used for model selection. Note that k-fold cross-validation, which needs at most k rounds of training, while DoFAIC only requires at most 2 rounds of training. This makes DoFAIC an efficient model selection criterion.

3.4 Discussion

In this chapter, I investigated the degrees of freedom for classification models and presented an efficient method to estimate their degrees of freedom. I showed that for simple classification models, degrees of freedom is equal to the number of parameters in the model. In deep networks, the

degrees of freedom is generally much less than the number of parameters in the model, and deeper networks tend to have fewer degrees of freedom. I also theoretically and empirically showed we can use DoFAIC as an efficient criterion for model selection, which has a comparable performance to cross-validation.

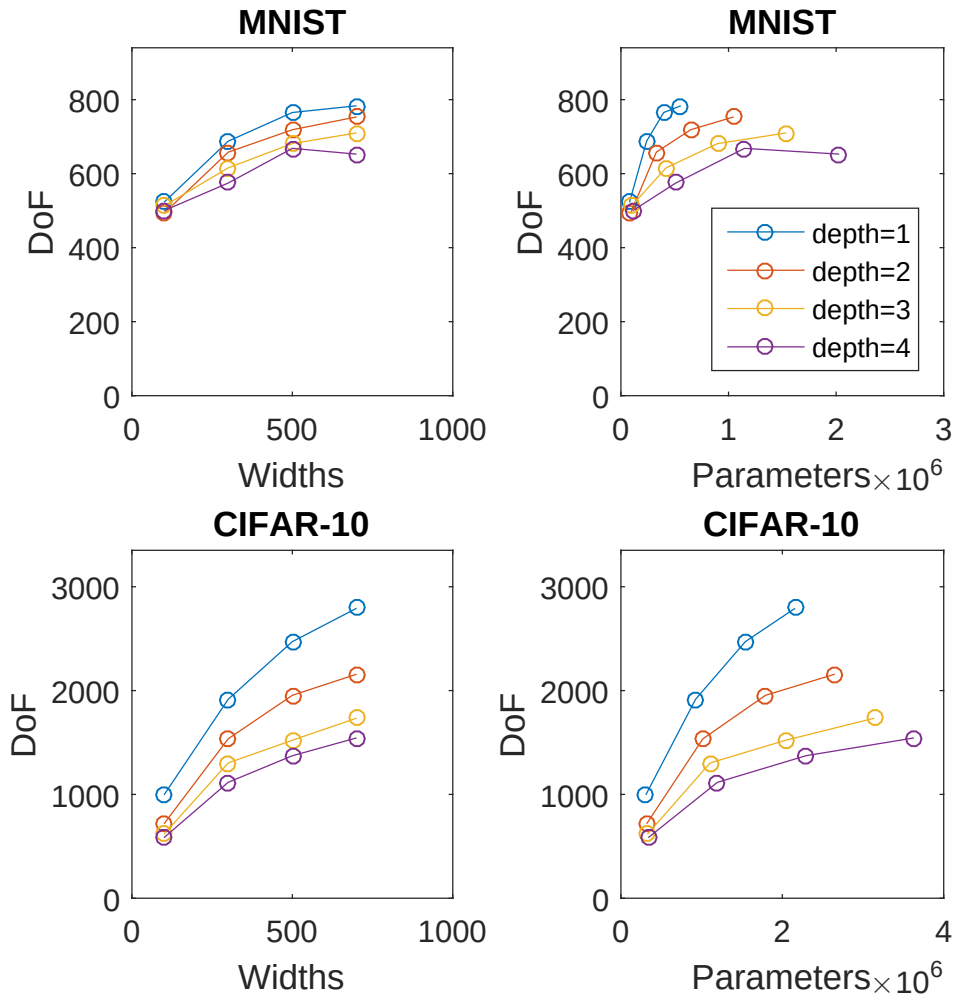

Figure 3.4: Degrees of freedom estimates for different models trained on MNIST and CIFAR-10. Left: degrees of freedom vs network width. Right: degrees of freedom vs the number of parameters in the network, which is linearly related to the network depth and quadratically related to the number of widths.

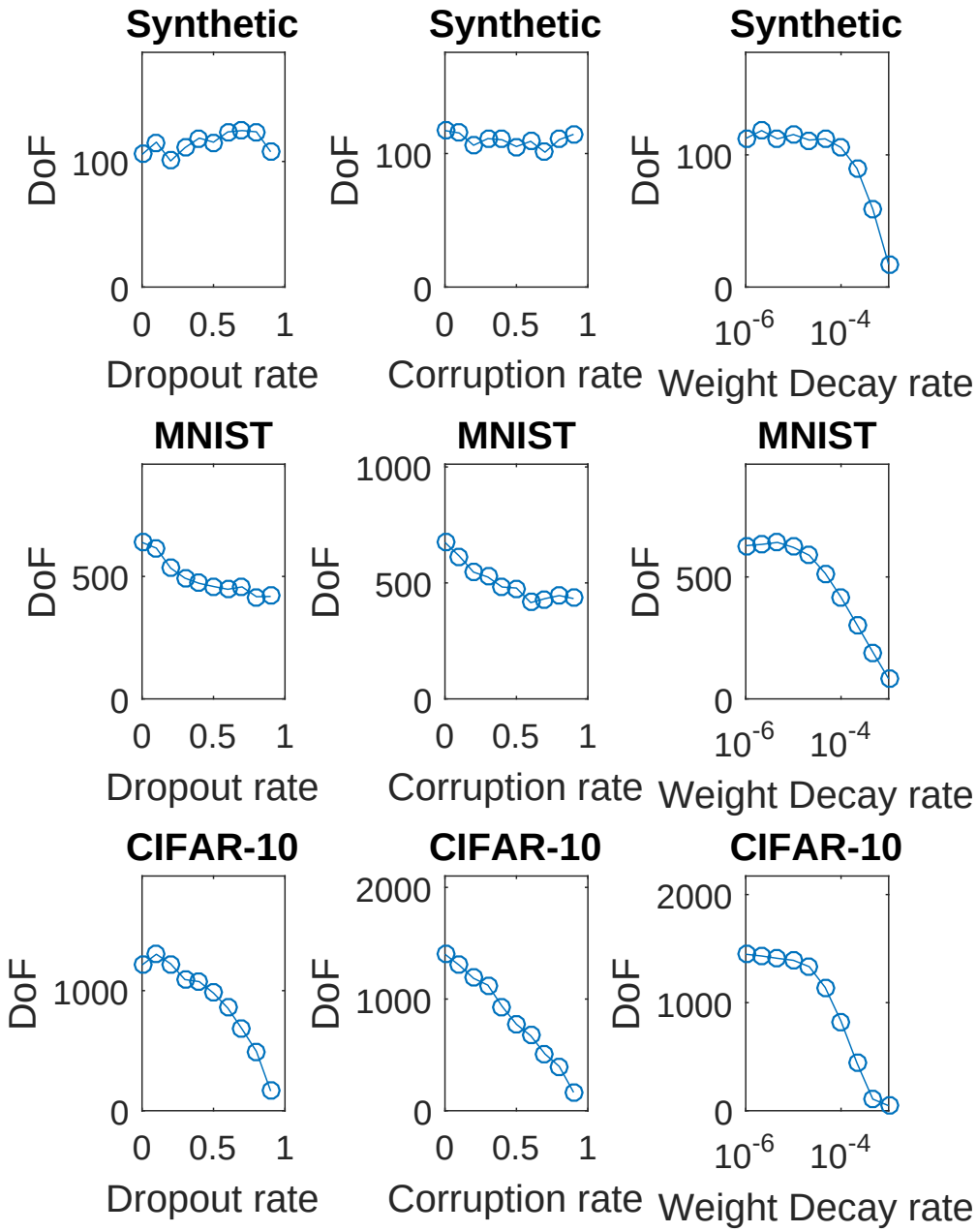

Figure 3.5: Degrees of freedom estimates for models trained on Synthetic data, MNIST and CIFAR-10 under different regularizations. The lines represent the degrees of freedom estimate.

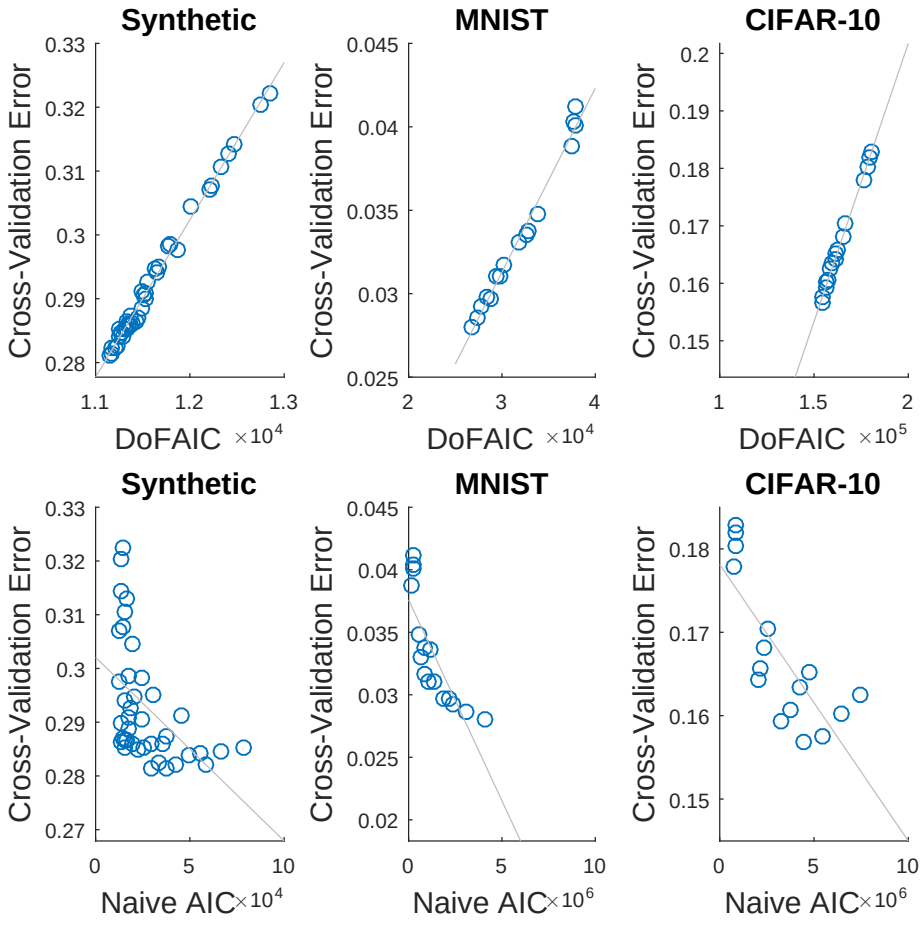

Figure 3.6: Comparison between DoFAIC (first row) / Naïve AIC (second row) and 5-fold cross-validation.

CHAPTER 4: SELECTING HYPOTHESES FROM DEEP NEURAL NETWORKS

4.1 Introduction

Deep learning methods have been successfully applied to many different tasks. However, despite providing a high-accuracy model, more knowledge and guidance were expected to be summarized from the well-trained deep models. However, unlike simple models, the information from nonlinear models is hard to be distilled.

In this work, I focus on the task of generating hypothesis from nonlinear models – deep neural networks in this case. In general, a model is trained on some training data and a prediction function is obtained as $f(\mathbf{x})$, where $\mathbf{x}$ is the input vector, and $f(\mathbf{x})$ is the prediction on the target output. In classification problems, $f(\mathbf{x})$ is the predicted probability; in regression problems, $f(\mathbf{x})$ is the predicted target value.

In general, the prediction function $f(\mathbf{x})$ is obtained in order to make accurate predictions. In some cases, people care more about how to utilize the trained model to obtain reasonable, conductible hypotheses. The hypothesis, in this case, is a pair of similar input vectors that is believed to induce a significant difference in the output. It can also be considered as a context when introducing specific small perturbations that can lead to substantial changes in the output.

In this work, I provided a framework for hypothesis generation. This framework includes 3 aspects: 1) A formal definition of hypothesis and quantified properties of interests. 2) Methods to obtain hypotheses of interests from a nonlinear model. 3) An economical procedure of selecting hypotheses for further validation. The framework was applied to both a toy image classification example and a plant-microbial design case study. The result shows that the framework is useful in generating hypotheses and provide guidance for validation experiments.

4.2 Method

In this section, I am going to introduce the definition of the hypothesis and the procedure for selecting hypotheses from a trained model and constraints.

4.2.1 Definitions

First, a *treatment* is defined as a pair of input vectors $T = (\mathbf{x}_r, \mathbf{x}_p)$, where $\mathbf{x}_r$ is the reference state and $\mathbf{x}_p$ is the perturbed state. $\Delta\mathbf{x} = \mathbf{x}_p - \mathbf{x}_r$ is defined as *perturbation*. A significant difference between the outcomes of these two states is expected. Given a trained model with predictive function $f(\mathbf{x})$, the *expected change* of a hypothesis can be defined as $D(h, f)$. In a classification problem, where $f(\mathbf{x})$ is the predicted probability of $\mathbf{x}$ being in the target class, D is defined as the log-fold change in the probability:

$$D(h, f) = \log f(\mathbf{x}_p) - \log f(\mathbf{x}_r)$$

In the regression problem, where $f(\mathbf{x})$ is the predicted value of the target output, D can be regarded as the squared difference:

$$D(h, f) = (f(\mathbf{x}_r) - f(\mathbf{x}_p))^2$$

When conducting the validation experiment, besides identifying states that can induce most significant changes, perturbing as few as factors as possible could make the treatment interesting and efficient. Therefore, *perturbation level* is introduced for a hypothesis as $\delta(h)$. Depending on the problem, different forms of perturbation level can be used. For example, the perturbation level can be defined as the L1 distance between perturbed state and referenced state: $\delta(h) = |\mathbf{x}_r - \mathbf{x}_p|$.

4.2.2 Generating Hypothesis

Each hypothesis h corresponds to a expected change $D(h, f)$ and perturbation level $\delta(h)$. Ideally, preferred hypotheses are the ones with small perturbation level and large expected change.

Let $H(\tau, K)$ be the *preferred hypotheses set*. All the hypotheses in the set should meet the following constraints:

$$D(h, f) \geq \tau, \quad \forall h \in H(\tau, K)$$
$$\delta(h) \leq K, \quad \forall h \in H(\tau, K).$$

If the trained model $f(\mathbf{x})$ is a linear function, the preferred hypotheses set can be easily obtained, as all the factors in the state are independent and their contributions to the output are linear. But if the trained model $f(\mathbf{x})$ is a nonlinear function, finding the exactly preferred hypotheses set can be very hard. However, if the hypotheses space is finite and small, all satisfied hypotheses can be identified by enumerating all possible pairs of states.

When state space is continuous and large, identifying the exactly preferred hypotheses set can be extremely inefficient. Given a referenced state $\mathbf{x}_r$, instead of finding all possible perturbed state that can induce desired expected changes in the outcome, hypotheses can be generated by finding one perturbed state that can induce the most changes. Therefore, the task can be written as a constrained optimization problem:

$$\max_{\mathbf{x}_p} \quad D(\mathbf{x}_r, \mathbf{x}_p, f)$$
$$\text{s.t.} \quad \delta(\mathbf{x}_r, \mathbf{x}_p) \leq K$$

There are many works from the model interpretation field aim to solve these kinds of problems, including perturbation-based approaches (Zeiler and Fergus, 2014; Zhou and Troyanskaya, 2015; Zintgraf et al., 2017), gradient-based approaches (Simonyan et al., 2013; Springenberg et al., 2014)

and novel approaches like Deep-Lift(Shrikumar et al., 2017). In this work, I am using perturbation based method for hypothesis generation.

Algorithm 2 Filter hypotheses from a candidate hypothesis set

Input: Candidate hypothesis set $C = \{h_1, h_2, \ldots, h_n\}$, perturbation threshold τ^p, expected change threshold τ^c, a trained model f.
Output: Hypothesis set H

$H \leftarrow \emptyset$
for each candidate hypothesis $h \in C$ **do**
    Calculate the expected change $D(h, f)$
    Calculate the perturbation level $\delta(h)$
    if $D(h, f) > \tau^c$ **and** $\delta(h) < \tau^p$ **then**
        $H \leftarrow H \cup h$

In general, to get a list of hypotheses of interest, one can enumerate through referenced states of interests (usually the training samples), calculate the corresponding best perturbation level and expected change, and pick any hypothesis that satisfied the desired constraints as the preferred hypotheses set. I provided an algorithm for selecting hypotheses when a candidate set of hypotheses is of interests in Algorithm 2.

4.2.3 An Economical Procedure of Hypothesis Selection

Validating a hypothesis comes at a cost. In many biological experiments, exhaustively validating all possible hypotheses is inefficient and can be extremely expensive. Therefore, an economical way of selecting hypothesis is important. Given a limited experimental resource, one might choose to greedily select the hypothesis from those with the most expected change. In some cases, it might not be the most efficient way of selecting hypotheses. For example, a single perturbed state might serve 2 referenced states, and by choosing that state might allow us to test more hypotheses. On the other hand, each state might come with a different cost. This requires us to utilize a better way of choosing hypotheses.

In this section, I am going to introduce different procedures to select hypotheses under a total cost constraint and maximize the total benefit.

The choice of hypothesis can be considered as a graph problem. A state $\mathbf{x}_s$ can be regarded as a vertex s in the graph. A hypothesis h can be regarded as an edge in the graph that connects two vertices r, p (referenced state and perturbed state). Considering a graph G of edges (hypotheses) as H and vertices (states) as S, each edge $h = (s, t) \in H$ has a benefit $b_{s,t} = v(h)$ as the expected change. For each vertex s, it associates with a binary variable z_s. $z_s = 1$ indicates the vertex is selected, while $z_s = 0$ indicates the vertex is not selected. Selecting one vertex corresponding to conducting a validation experiment with the state $\mathbf{x}_s$. Hence, a cost w_s associates with each vertex s.

The goal is to find a subset of vertices with total costs less than K, and the edges that covered by the vertices (where both ends are selected) contain the most benefit.

A simple idea is to use the greedy algorithm. The idea is to select hypotheses based on their benefits from high to low as long as the total cost is less than K, as shown in Algorithm 3.

Algorithm 3 Greedy algorithm for hypothesis selection

Input: Hypothesis set $H = \{h_1, h_2, \ldots, h_n\}$, corresponding benefit $B = \{b_1, b_2, \ldots, b_n\}$, cost for each state $\mathbf{w}$, total cost constraint K
Output: Selected state set Z

```
Z ← ∅
Sort H and B based on the value B from high to low
for each hypothesis h ∈ H do
    u, v ← the states of h
    if total cost of Z ∪ {u, v} is less than K then
        Z ← Z ∪ {u, v}
```

The greedy algorithm apparently would not provide the optimal solution. In order to better state the problem, it can be written as an quadratic constraint optimization problem:

$$\begin{aligned} \max_{\mathbf{z}} \quad & \textstyle\sum_{s,t \in S} z_s z_t b_{(s,t)} \\ \text{s.t.} \quad & z_s \in \{0, 1\}, \forall s \in S \\ & \textstyle\sum_{s \in S} w_s z_s \leq K \end{aligned} \tag{4.1}$$

Hence, the problem becomes a quadratic integer programming (QIP) problem (Bertsekas, 1999). In this work, the CPLEX quadratic integer programming solver (ILOG, 2014) is used. The algorithm takes the input of a state cost vector $\mathbf{w}$ and a hypothesis benefit matrix B and returns a binary choice vector $\mathbf{z}$. The choice vector $\mathbf{z}$ indicates which states to be validated. The procedure is described in Algorithm 4.

Algorithm 4 Quadratic integer programming algorithm for hypothesis selection

Input: Hypothesis set $H = \{h_1, h_2, \ldots, h_n\}$, corresponding benefit $B = \{b_1, b_2, \ldots, b_n\}$, cost for each state $\mathbf{w}$, total cost constraint K.

Output: Selected state set Z

1: $\mathbf{z} \leftarrow$ optimal solution of objective 4.1

2: $Z \leftarrow \{i | z_i == 1, \forall i\}$

If the cost of conducting an experiment is same for all states, the cost can be set to $w_s = 1$ for all states. The total benefit provides a higher opportunity that more hypotheses are validated.

4.2.4 Summary

Now I introduced the definition of hypothesis, the properties of a preferred hypothesis set, the procedure of generating hypothesis from a trained model, and when cost is under consideration, how to smartly choose the hypotheses. This framework can be applied to different tasks as need with flexible constraints and economical considerations.

In the next section, I am going to present both a toy example and a biological case study using the hypothesis generation framework. In the first toy example, the framework is used to help identify hypotheses about masked perturbations on a digit-classification task. In the second one, the hypotheses generation framework is used to help design microbial communities that improve plant nutrition.

4.3 Toy Example: Hypothesis Generation for Digit-classification image

In this toy example, I will show how to generate hypotheses using the framework from a convolutional neural network trained on images to predict digits. This example is aimed to provide a guide for using the hypothesis generation framework.

4.3.1 Task Description

In this example, hypotheses generation is applied to a digit image classification scenario on the MNIST dataset (LeCun et al., 1998). To simplify the task, I aimed to generate hypothesis with small perturbations that can change images with digit "0" to digit "6". Assume that there is a cost associated with the validation of each image, as in the real world setting, this could be done by recruiting people to label images. A candidate image set (has not been seen by the model) is provided as the state space for hypotheses. Valuable hypotheses are generated under different cost constraints.

4.3.2 Hypothesis Generation

First, a convolutional neural network was trained using Keras (Chollet et al., 2015) on 60,000 MNIST images. The model reached an accuracy of 98.7%. A candidate state space with 50 "0"-digit images and 50 "6"-digit images is provided. All pair-combinations of 100 images were enumerated to generate the candidate hypotheses set, which results in 9900 candidate hypotheses. The expected changes in the log predicted probabilities of class "6" was calculated. The perturbation level was calculated using the mean absolute difference between images.

The hypotheses were selected using Algorithm 2 with perturbation level smaller than $\tau^p = 0.15$ and expected change greater than $\tau^c = 5$ as the preferred hypotheses set. 502 hypotheses and 88 candidate images were selected after filtering. An overview of the generated hypotheses and threshold is shown in Figure 4.1. Each point in the graph represents a candidate hypotheses, with the x-axis as the perturbation level and y-axis as the expected change. Red points are hypotheses that

indeed results in a change from digit “0” to digit “6”, and blue points do not. The vertical/horizontal dashed line represents the threshold used for perturbation level/expected change. Two sample selected hypotheses are shown in Figure 4.2. These examples have small perturbation level and large expected change. Each row represents a hypothesis. From left to right, referenced image, perturbed image, the difference between two images. The hypothesis in the first row has perturbation level 0.072 and expected change 6.87. The hypothesis in the second row has perturbation level 0.085 and expected change 8.71.

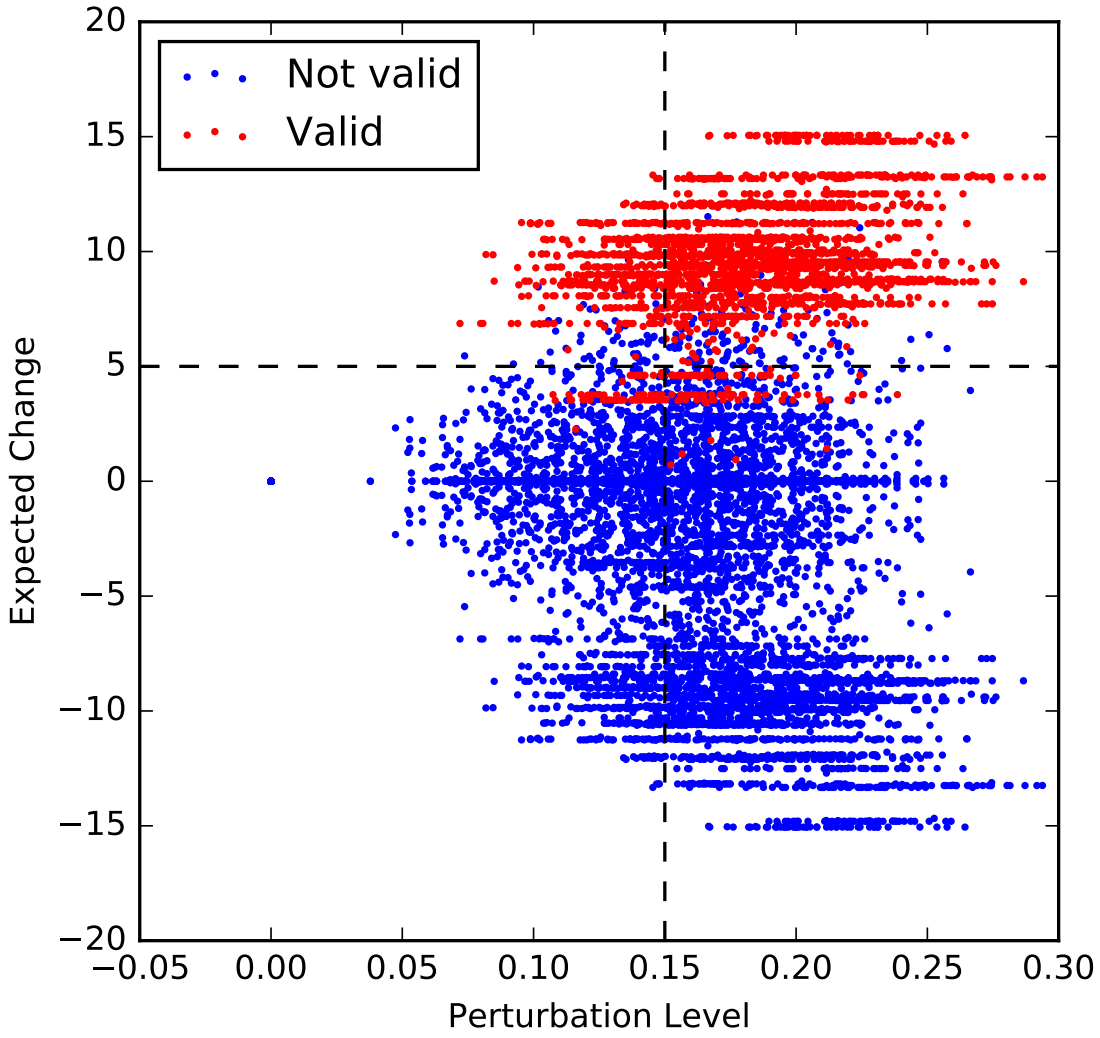

Figure 4.1: Overview of hypotheses space and thresholds on digit “0” to digit “6”.

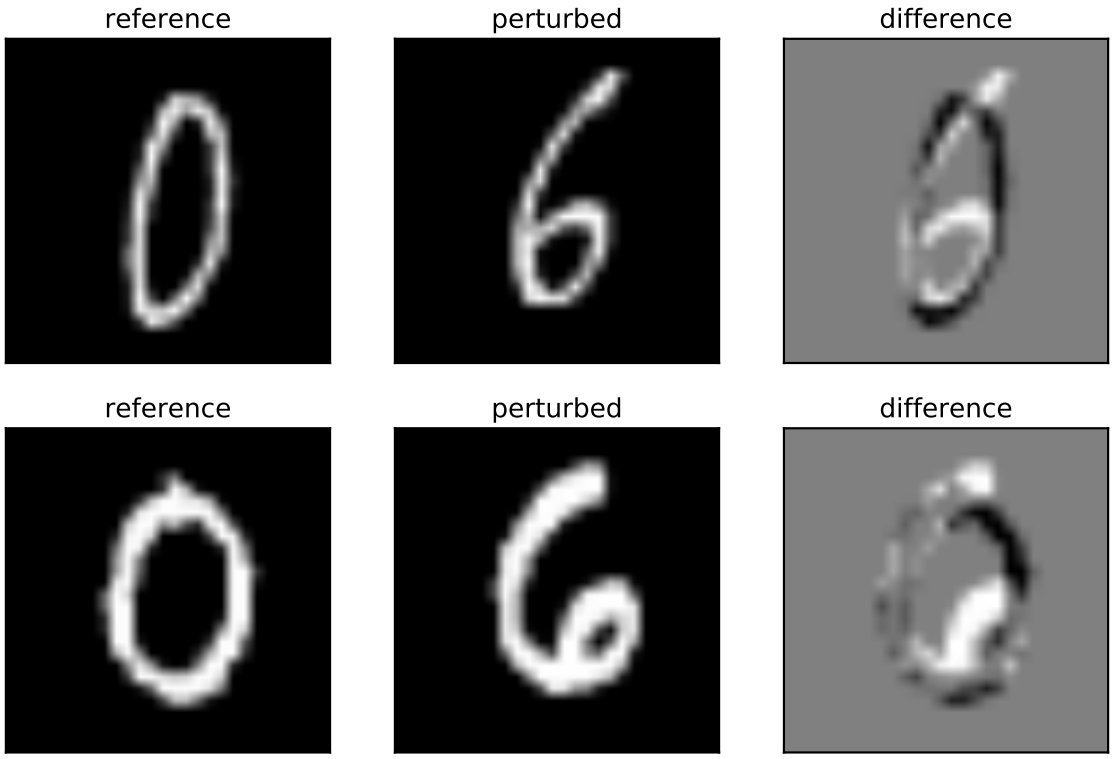

Figure 4.2: Examples of selected hypotheses.

4.3.3 Hypothesis Selection

If the cost of validating the label of an image is considered, it is important to choose hypotheses to validate optimally. In this work, I used both the greedy method described in Algorithm 3 and the quadratic integer programming (QIP) method described in Algorithm 4 to select hypotheses with the cost constraint.

Different cost constraint K are chosen from $[10, 30, 50, 70, 90]$ for the algorithms. For each running, I calculate 1) total benefit obtained from both methods, 2) valid hypotheses count from the selected hypotheses, 3) total hypotheses generated from both methods. The result is shown in Figure 4.3. QIP method outperformed the greedy method and generated more valid hypotheses under different cost constraint.

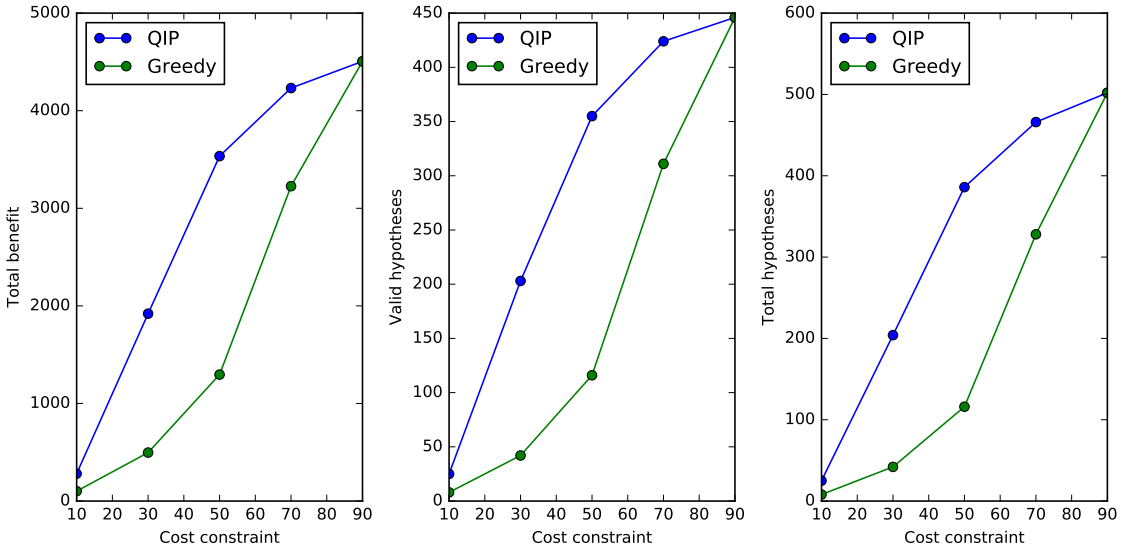

Figure 4.3: Hypothesis selection for digit classification using greedy and QIP algorithm.

In this toy example, I demonstrated the procedure of generating hypotheses from the MNIST dataset. This framework can be used to generate reasonable hypotheses. It can also account for generating most efficient hypotheses under a cost budget.

4.4 Case Study: Hypothesis Generation for Microbial Community Design

In this work, I provided a case study for hypothesis generation for identifying the microbial synthetic community designs that will bring significant improvement in nutrition concentration in the plant. This work is in collaboration with Dangl's lab. The work has been submitted to PLoS. (Paredes et al., 2017).

4.4.1 Background

In ecosystem function and plant development, microbial communities are a key important factor (Seedorf et al., 2014; Faith et al., 2014; Hacquard et al., 2015; Bai et al., 2015). One goal in microbiology is to design microbiome synthetic communities (SynComs) with desired functions and properties, for example, increasing the nutrient absorbability in the plant. In this work, I focus on design of SynComs that can increase the absorption of a class of macronutrient for the plant,

inorganic orthophosphate (Pi-content), which is limited in the soil. An example of the plant growth under the environment of different Pi-content levels is shown in Figure 4.4.

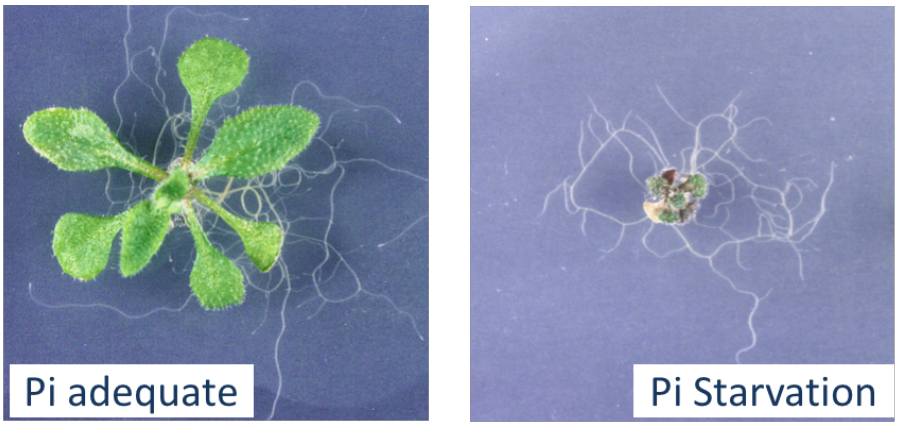

Figure 4.4: Plant growth under different levels of inorganic orthophosphate (Pi).

It is a huge cost to conduct and measure Pi-content for all possible SynComs exhaustively. Instead, machine learning models can be trained on existing data. These models can be used to generate hypothesis about the Pi-content change of untested designs.

In this work, I first modeled the data using a multilayer feed forward neural network. Next, I used the model to generate a set of hypotheses using the framework. These generated hypotheses were finally validated from biology experiments. This case study provides a full procedure for generating hypotheses and validation. A significant amount of hypotheses are validated in the end, which provides a list of applicable treatments guidance for building beneficial microbial synthetic community.

4.4.2 Modeling Pi-content using Neural Network

As Pi-content measurement of a SynCom has different mean under different phosphate conditions, the model is expected to capture these interactions. These interactions come from both environmental effects and microbiome synthetic communities, as shown in Figure 4.5.

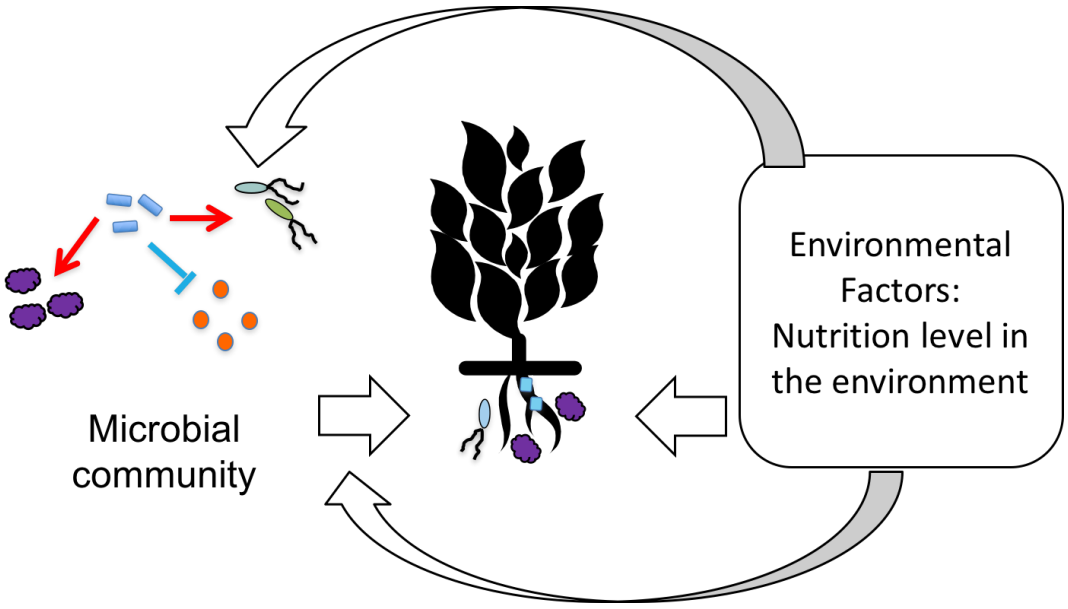

Figure 4.5: Pi-content in the plant can be influenced by the interaction between environmental factors and microbial communities.

Deep learning methods have been successfully applied to many problems and outperformed many simple linear methods (LeCun et al., 2015; Angermueller et al., 2016; Min et al., 2016). Multilayer Feedforward neural network (NN), a typical framework for deep neural network structure family, is a prediction model where input data are combined and transformed non-linearly through multiple layers of hidden nodes. NN can capture high-order interactions between input features through intermediate layers, as they are universal approximators (Hornik et al., 1989).

In this work, I trained NN models on an existing data set – primary experiment. It contains the Pi-content measurements from 14 pair-combinations from 9 different non-overlapping bacteria groups (P1,P2,P3, I1,I2,I3,N1,N2,N3) under 4 different environment conditions (combination of a starvation/non-starvation pre-treatment of the plant, and a rich/poor nutrient condition in the experiment soil). A model takes the design information as the input, and output the predicted Pi-content measurement. Table 4.1 summarized the input vector.

Table 4.1: Summarization of the input design information

Feature Name	Replicate b	Pre-treatment p	Phosphate level q	Synthetic community bacterial block indicator S
$x = 0$ or 1	$b = 1$ or 2	$p = -Pi$ or $+Pi$	$q = 30\mu M$ or $100\mu M$	bacterial block is absent/present in this design

Several NN models are trained on the data from the primary experiment, and the best model has 3 hidden layers and 200 hidden nodes in each layer, as shown in Figure 4.6.

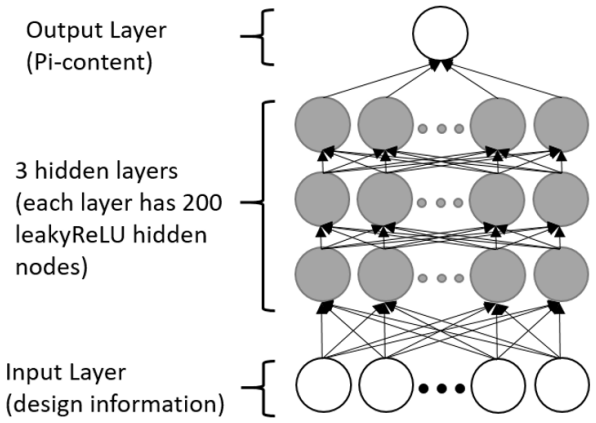

Figure 4.6: Schematic representation of the neural network defined and applied for predictions.

A leave-SynCom-out cross-validation experiment is performed. Each model is trained on all but one synthetic community and evaluated on that held out the synthetic community. The result is shown in Figure 4.7. Each dot in the plot represents the mean Pi content prediction error on the held out synthetic community. The result shows that NN has lower prediction error on held-out SynCom samples comparing to simple linear model (LM) and a linear model with manually constructed interaction features (INT).

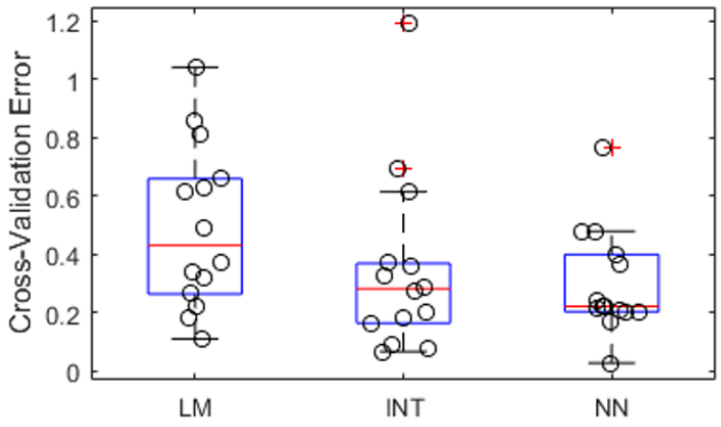

Figure 4.7: Cross-validation error from the three types of models tested for their ability to predict shoot Pi content.

To visualize the learned model of NN, I investigated the ”sensitivity” of the network, which is the change in the predicted output when perturbing a particular input feature (Figure 4.8). Each dot represents the sensitivity under a specific context. This approach is similar to derivatives analysis of the network in many computer vision tasks (Simonyan et al., 2013; Wang et al., 2016). Sensitivity in LM is constant for a feature across different context. In INT and NN, sensitivity is different in the different context. NN can capture more complex context-dependent sensitivity than INT. From the sensitivity analysis, I found that changing phosphate-level from 30μM to 100μM will always increase the Pi-content. However, when switching pre-treatment from -Pi to +Pi, the change in Pi-content is not always positive.

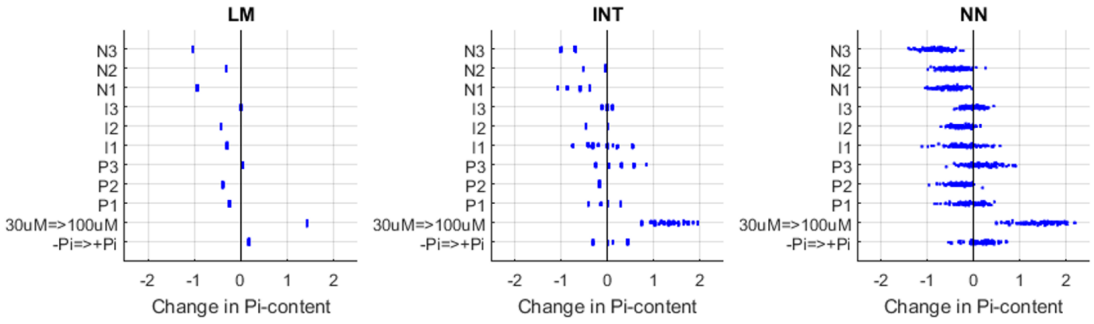

Figure 4.8: Sensitivity of Pi accumulation with respect to each biological variable for each type of model.

4.4.3 Hypothesis Generation and Validation

In this section, I use the trained model to generate hypothesis about possible treatments in untested SynComs. Enhancing the plant nutrition concentration by increasing environment nutrition level seems obvious and not interesting in this case. Hence, I focus on generating hypothesis that improves nutrition concentration by switching microbial blocks while keeping the environment nutrition level fixed.

The possible perturbation level allowed in this study is microbial block swapping. Let S_r and S_p be the referenced and perturbed SynCom, where $|S_r| = |S_p| = 2$ and $|S_r \cap S_p| = 1$. This means that swapping a microbial block in the referenced SynCom to another different microbial block. All possible swaps of bacterial blocks under a certain phosphate condition (pre-treatment p phosphate-level q) are considered as the candidate hypotheses set that satisfies the perturbation level constraint.

To calculate the expected change, the hypothesis that can induce a significant difference in the output are considered. Hence, not only the mean difference but also the variance is needed to be modeled. In this work, the worst-case variance estimate is used. The largest residual variance (the difference between observed value and predicted value) related to a bacterial block is transferred from the training data to all related conditions. Hence, the $p - value$ of all possible hypotheses can be calculated using predicted mean from the trained NN model and variance from the training data.

A specific starvation phosphate condition of interest $-Pi, 30\mu M$ is chosen and hypotheses that at least one state is not tested are generated. In this case, as there are not many valid states, all 25 hypotheses are selected and validation experiment for 20 SynComs was conducted. The same procedures as the primary experiment have been used to gather Pi-content measurements in the plant under phosphate condition $-Pi, 30\mu M$.

The validation result of the chosen hypotheses is shown in Figure 4.9.These block replacements involved 20 different synthetic communities. Each box represents selected replacements in a particular constant background noted at the top. Each arrow represents a replacement of the bacterial block on the left with the block on the right. Asterisk indicates the blocks that lead to

maximal plant Pi accumulation in the validation experiment. There was a significant correlation (C= 0.42, p-value = 0.0375) between predicted and observed shoot Pi content changes caused by the bacterial block replacements (Figure 4.10). Strikingly, 23 out of 25 bacterial block replacements increased shoot Pi content (p-value = 0.004; 1000 permutation tests with synthetic community labels randomly permuted). Moreover, the improvement in shoot Pi content was statistically significant in 16 out of 25 bacterial block replacements (p-value = 0.032; 1000 permutation tests with synthetic community labels randomly permuted). Only 1 out of 25 bacterial block replacements significantly decreased Pi content.

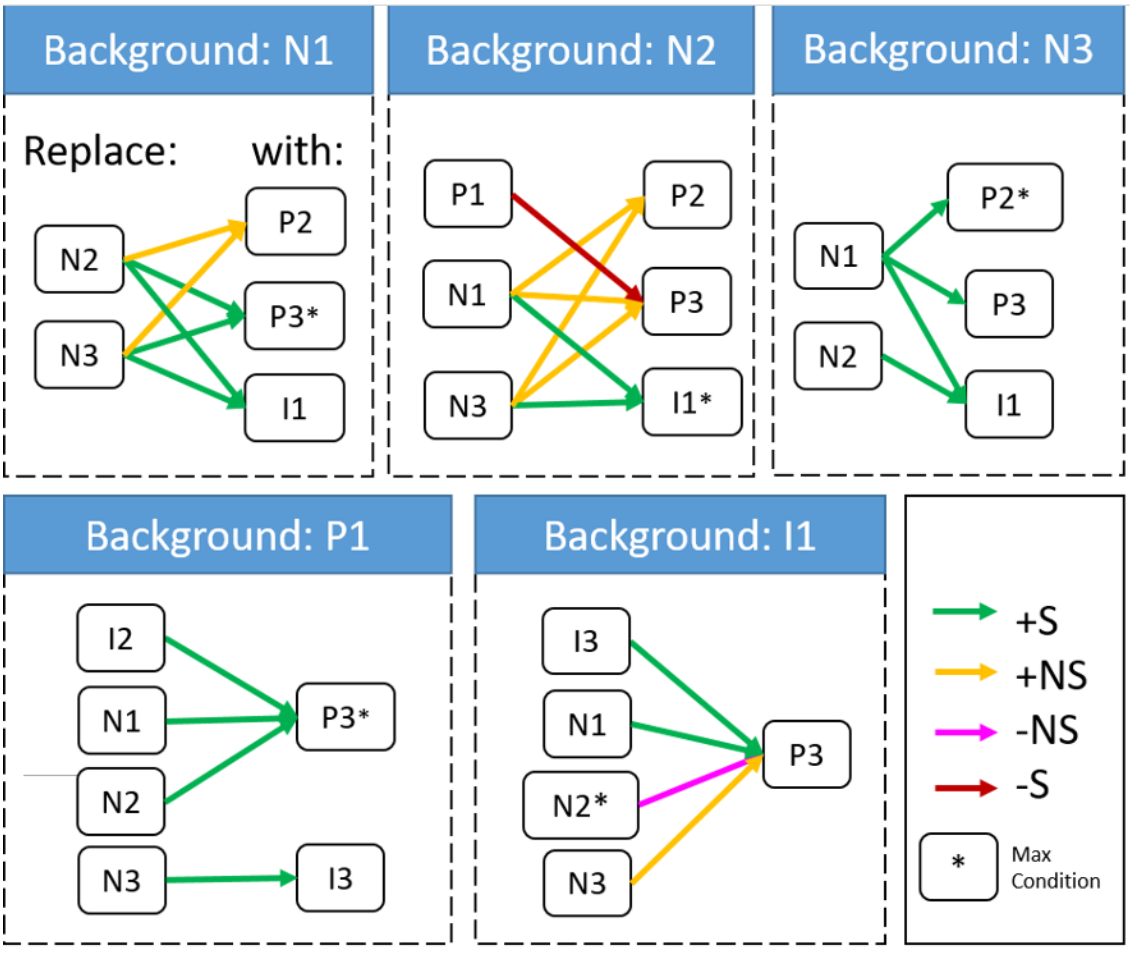

Figure 4.9: The most significant 25 block replacements with a positive effect on the shoot Pi concentration predicted by the neural network.

As the selection space is small, all possible combinations from the hypothesis space are tested. However, the greedy method and QIP are still tested on the validated experiment result to select

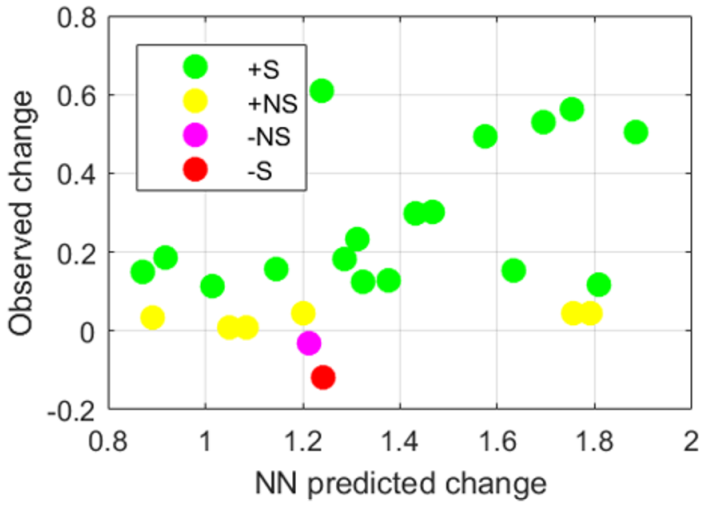

Figure 4.10: The shoot Pi accumulation change predicted by the neural network (x-axis) and the change observed experimentally are significantly correlated (Spearman's correlation co-efficient 0.42, p-value = 0.0375).

hypothesis with cost constraints. The result is shown in Figure 4.11. As expected, using QIP could provide an equal or better hypothesis selection compared to the greedy method.

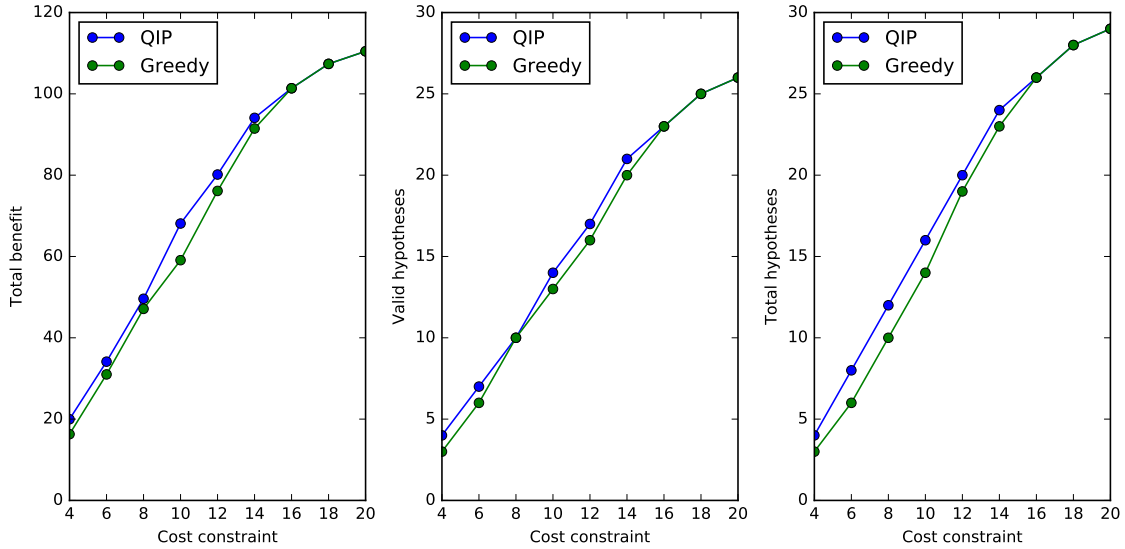

Figure 4.11: Hypothesis selection for synthetic microbial community design using greedy and QIP algorithm.

4.5 Summary

In this chapter, I provided a hypothesis generation framework for nonlinear methods. To select reasonable hypothesis for validation, perturbation level and expected change of the hypothesis should be used to filter hypotheses with good quality. I provide two algorithms for selecting hypotheses when the experimental budget is limited. Two case studies including an image digit classification and a microbial synthetic community design were provided. The experimental results showed that the method is useful in extracting hypothesis.

CHAPTER 5: LEARNING INFORMATIVE REPRESENTATIONS USING GUIDED AUTOENCODERS

5.1 Introduction

Nowadays, the data often contain a large number of dimensions. In order to summarize and visualize the data, methods that can extract a compressed representation of the data are expected. The compressed representation could contain an essential information and characteristics of the original data. Hence, analysis of these compressed representations could help us to efficiently gather valuable information and knowledge.

To define the concept formally, I introduce some basic concepts and definitions about compressed representation. Given the real value input data vector $\mathbf{x}$ with p dimensions as the *original data*, an *encoder* function is expected to transfer the original data to the *compressed representation* $\mathbf{c}$ of m dimensions:

$$\text{enc}(\mathbf{x}) = \mathbf{c}.$$

m is defined as *code length* of the encoder, as it represents how many codes are need to store the data. In order to compress the original data, the code length should be smaller than the original data dimension: $m < p$.

The original data should be reconstructible from the compressed representation $\mathbf{c}$. Therefore, a *decoder* function corresponds to the encoder function should transfer the compressed representation $\mathbf{c}$ to the *reconstructed data* $\hat{\mathbf{x}}$:

$$\text{dec}(\mathbf{c}) = \hat{\mathbf{x}}.$$

Let $f(\mathbf{x})$ be the combination function of the encoder and decoder functions $f(\mathbf{x}) = dec(enc(\mathbf{x}))$. To evaluate the quality of the compressed representation functions, a loss function can be calculated.

In this case, the mean-square-error is used:

$$\mathcal{L}(f, \mathbf{x}) = ||f(\mathbf{x}) - \mathbf{x}||_2^2$$

In general, a set of encoder/decoder functions can be trained by optimizing the following objective:

$$\underset{f}{\text{minimize}} \quad \mathcal{L}(f, \mathbf{x}) + R(f) \tag{5.1}$$

where $R(f)$ is a regularization term on the encoder/decoder to avoid overfitting.

Principal component analysis (PCA) was used a lot in many statistical applications. It learns a compact representation, which is obtained from a linear transformation of the original data (Jolliffe, 1986). Its encoder and decoder function can be written as:

$$\begin{aligned} \text{enc}(\mathbf{x}) &= \mathbf{W}\mathbf{x} \\ \text{dec}(\mathbf{c}) &= \mathbf{W}^T\mathbf{c}, \end{aligned}$$

where $\mathbf{W}$ is a matrix of m by p. The learned compressed representations – principle components – $\mathbf{c}$ are ranked by their contribution to the total variance of the data.

Besides linear methods like PCA, deep structures like autoencoders represent another family of compressed representation learning. An autoencoder performs a nonlinear transformation of the data through neurons and activation functions through multiple layers (Bengio et al., 2007). With the ability to learn nonlinear transformation, autoencoders can model more compressed representations comparing to linear methods.

It is important to use a model that can learn compressed representations. However, sometimes it is more important that this compressed information carries characteristics of interests. This is somehow important for nonlinear transformations, as many very different representations might have similar reconstruction errors. In this situation, one usually wants a representation that is the most informative of a specific target of interest.

Hence, under a fixed code length, a representation can be constructed with two focuses: (1) the learned compressed representation can be recovered to the original data as accurate as possible (reconstruction loss); (2) the learned compact representation should be as informative of the desired target as possible (prediction loss). To achieve this goal, I proposed a novel structure – guided-auto-encoder – that is able to learn a compressed representation contains both the information from original data and information about the target. I will show that by balancing the ratio factor in the learning, representations with different informative levels can be obtained.

In the following sections, I will first introduce the autoencoder and guided-autoencoder. Next, I will show a case study of using guided-autoencoder to obtain an immunology scores from the immunology data.

5.2 Methods

5.2.1 Multi-layer Autoencoder

Autoencoders with encoder and decoder functions are considered as multiple-layer feedforward neural networks. The encoder and decoder have the same number of layers. Hence, the depth of an autoencoder is defined as the number of layers in encoder/decoder.

For convenience, the input layer is defined as $h_0(\mathbf{x}) = \mathbf{x}$, and the output of lth hidden layer is defined as $h_l(\mathbf{x})$. The number of nodes in layer l is m_l. The input into the lth layer of the network is defined as:

$$a_l(\mathbf{x}) = h_{l-1}(\mathbf{x})^T \mathbf{W}_l + \boldsymbol{\beta}_l,$$

where $\mathbf{W}_l$ is a real value weight matrix of m_{l-1} by m_l and $\boldsymbol{\beta}_l$ is a vector of length m_{l-1}. The output of lth hidden layer is:

$$h_l(\mathbf{x}) = \tanh(a_l(\mathbf{x})),$$

where tanh is the hyperbolic tangent function:

$$\tanh(x) = \frac{1 - e^(-2x)}{1 + e^(-2x)}.$$

For an autoencoder of depth L, the output of the Lth layer $h_L(\mathbf{x})$ is defined as the coding layer. Hence, the encoder function can be written as:

$$\text{enc}(\mathbf{x}) = h_L(\mathbf{x}).$$

The decoder function consists of layers from $L+1$ to $2L-1$. In this work, the weight matrices $\mathbf{W}_l$ are not tied between encoder and decoder layers.

To model the data with real values, a linear output layer is on top of the last decoding layer:

$$f_{AE}(\mathbf{x}) = h_{2L-1}(\mathbf{x})^T \mathbf{W}_{2L-1} + \boldsymbol{\beta}_{2L},$$

where $f_{AE}(\mathbf{x})$ is the overall reconstruction function.

An example guided-auto-encoder is showed in Figure **??**.

Now the autoencoder with multiple layers of non-linear transformation is introduced. To train this autoencoder, we can minimize the reconstruction loss by optimizing the following objective:

$$\underset{\theta}{\text{minimize}} \quad ||f_{AE}(\mathbf{x}, \theta) - \mathbf{x}||_2^2 + \lambda ||\theta||_2^2, \tag{5.2}$$

where θ represents all the parameters used in the autoencoder, and λ is the weight decay penalty for regularization term. To optimize the objective (5.2), the stochastic optimization method ADAM (Kingma and Ba, 2014) is used.

5.2.2 Guided-Autoencoder

A guided-autoencoder aims to model both the reconstruction and prediction of a target label. Given the input $\mathbf{x}$, a target label y and an auto-encoder f_{AE}, the guided-autoencoder builds a linear predictive function on the coding layer:

$$f_G(\mathbf{x}) = h_L(\mathbf{x})^T \mathbf{w}_G + \beta_G,$$

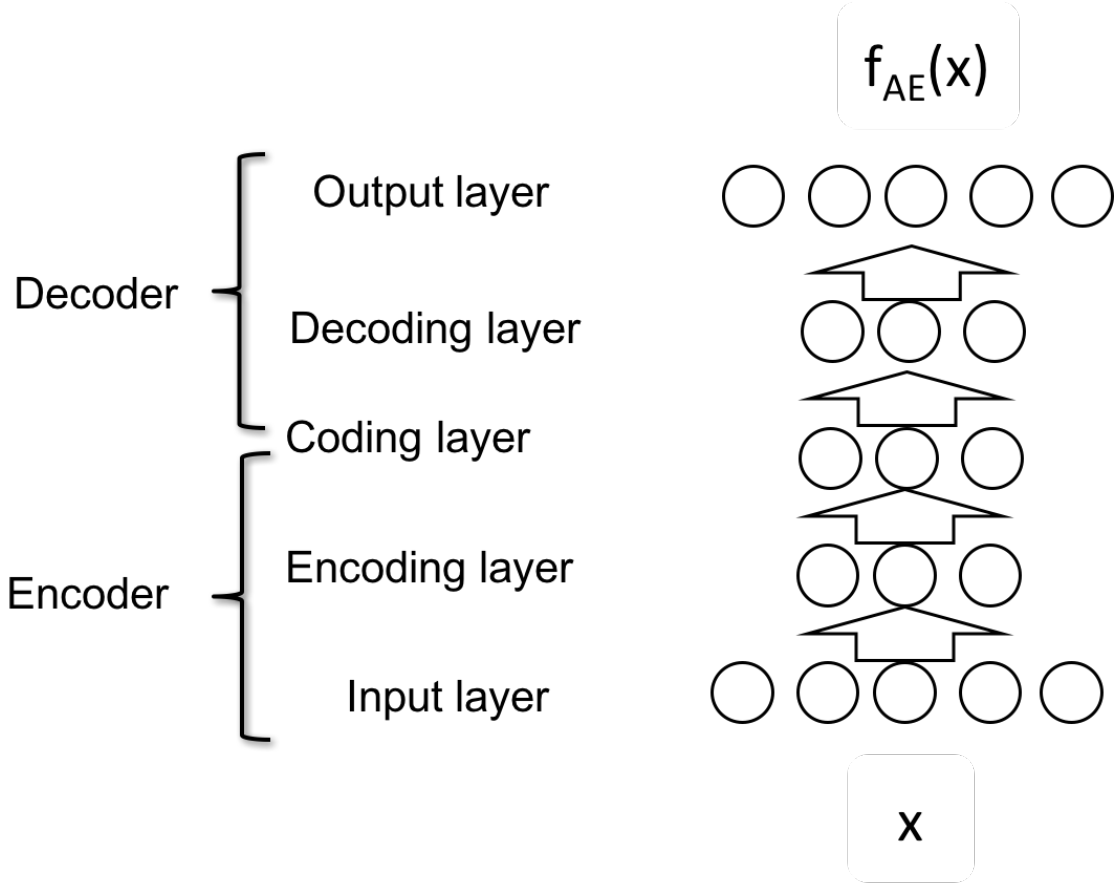

Figure 5.1: An example auto-encoder.

f_G contains the parameters $\mathbf{w}_G$ and β_G as the function and parameters for the guided layer. Let θ be the set of all parameters, the training objective becomes:

$$\underset{\theta}{\text{minimize}} \quad \alpha||f_G(\mathbf{x}) - y||_2^2 + (1-\alpha)||f_{AE}(\mathbf{x}) - \mathbf{x}||_2^2 + \lambda||\theta||_2^2 \tag{5.3}$$

where $\alpha \in [0, 1]$ is the guided ratio. If $\alpha = 0$, the objective (5.3) is equivalent to the objective of autoencoder (5.2). If $\alpha = 1$, the optimization has the nothing to do with the parameters in the decoder, and the objective is equivalent to training the encoder function as a feed forward neural network. The essential factor in guided-autoencoder is choosing a proper guided ratio α during training as needed. In fact, later I will show that by choosing different guided ratio α, one can generate representations of different levels of balancing between reconstruction and prediction power.

An example guided-auto-encoder is showed in Figure 5.2.

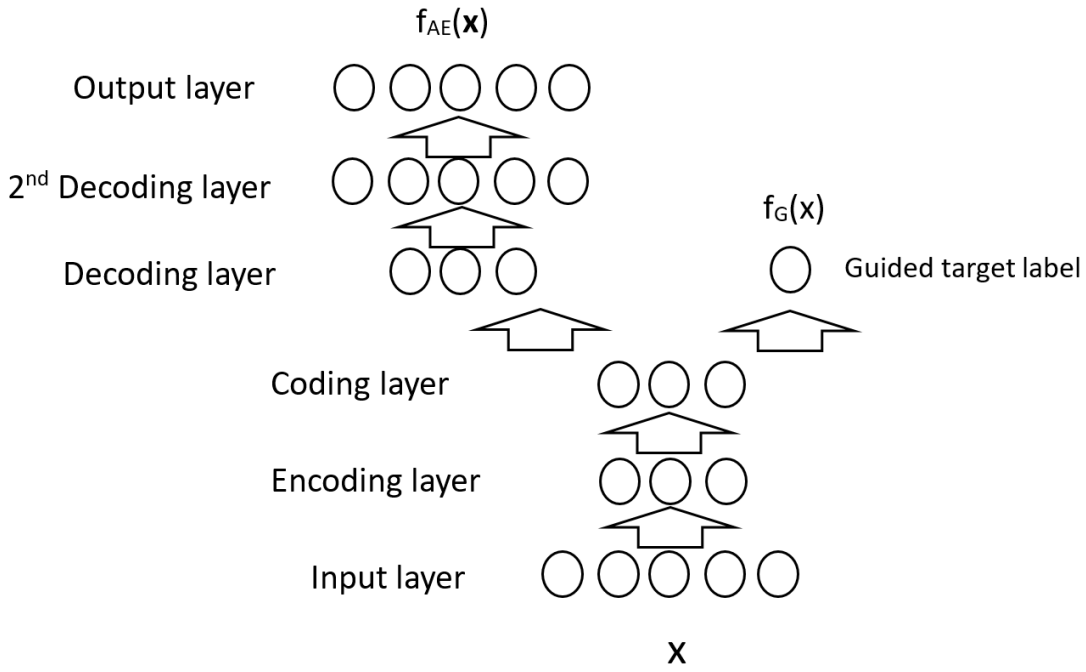

Figure 5.2: An example guided-auto-encoder with depth 2 and width 3.

5.3 Learning Informative Representations from Immunology Data

In this work, I am going to provide a case study on learning informative representations using guided-autoencoder from immunology data. The result indicates that informative health-related codes can be extracted using guided-autoencoder.

5.3.1 Background

Low-grade chronic has been associated with most diseases associated with aging, including cardiovascular disease, cancer, neurodegenerative disorders and many others, but the mechanisms underlying this type of inflammation are poorly understood. Recent technological developments have made it possible to extensively monitor biological material for a relatively low cost, enabling systems-wide characterization of immune systems.

In this work, I set out to study the human blood immunome of 1,000 subjects age 9-96. 50 different inflammatory proteins levels were measured. A compressed representation of the inflammatory proteins measurements could be learned to summarize the patient of different types. The relationship between inflammatory protein levels and the chronological age of the subject is also of interest. Hence, an "inflammatory score" built on composite objectives – informative of age and reconstructible of inflammatory protein levels – could be learned from guided-autoencoders.

5.3.2 Data preparation and model training

The inflammatory protein level data contains $n = 1000$ samples and $p = 50$ features. To use the data for analysis, the data were log-transformed. As the data is a combination of two different cohorts, a linear batch-correction on the transformed data is performed to remove batch-effects. Finally, each processed feature is centralized with mean zero and standardized with unit standard deviation. The target label of interests – chronological age – is used here. To make a comparable comparison, ages are standardized among all subjects.

PCA and autoencoders were trained on the data to obtain compressed representations. ElasticNet(Zou and Hastie, 2005) was used to regress the compressed representations to the age. A 5-fold-cross-validation for each model of code length from 1 to 10 is performed. To make a comparison, a linear model is trained as the baseline for prediction error. The prediction and reconstruction loss for PCA and autoencoder is shown in Figure 5.3. From the result, we can see that autoencoder is able to get smaller reconstruction loss compared to PCA under each code length. But neither methods can extract compressed representations that have smaller or comparable prediction error than the baseline linear model.

To learn informative representations that are predictive of the chronological age of the subjects, guided-autoencoder can be used to include the prediction loss in the objective during training. In order to compare different models, total loss – the sum of prediction loss and reconstruction loss – was used as a model evaluation metric. In each fold, the best hyper-parameters (depth, weight decay

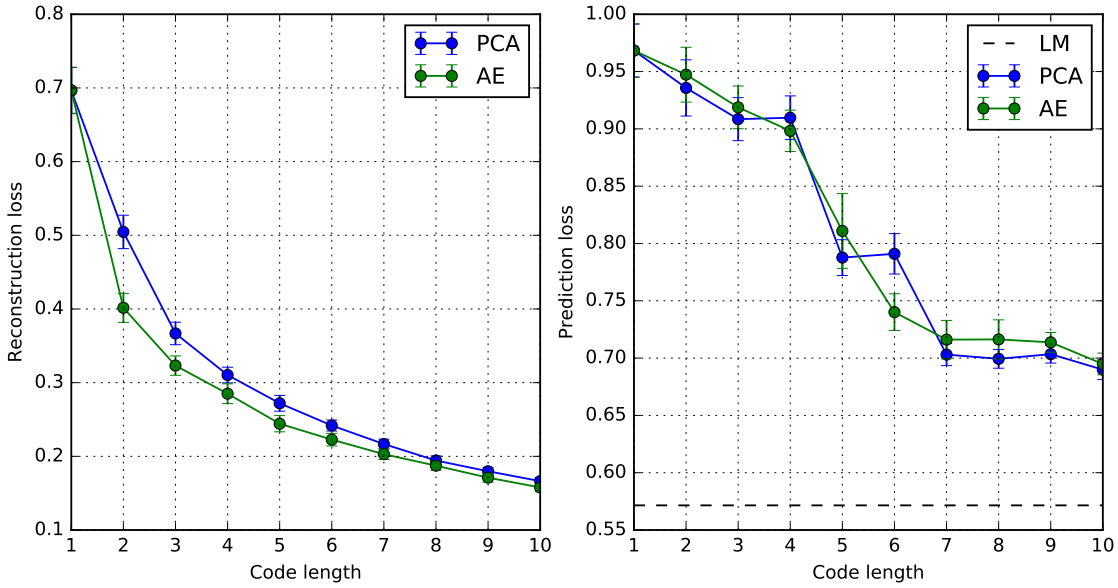

Figure 5.3: The reconstruction loss and prediction loss for PCA and autoencoder with different code length.

and guided ratio) were chosen from a 3-fold-cross-validation. The result is shown in Figure 5.4. Guided-autoencoder is able to achieve the lowest total loss of each code length.

To better understand the effectiveness of guided-autoencoders. The prediction loss and reconstruction loss of guided-autoencoders trained with different guided-ratios with code length 3 are shown in Figure 5.5. Note that the top left most GAE is equivalent to autoencoder, and the bottom right GAE is equivalent to a neural network. A properly trained GAE can achieve a good balance between reconstruction loss and prediction loss. By tuning the guided-ratio in guided-autoencoder, the solutions form a path representing the trade-offs between prediction power and reconstruction power.

When guided-ratio is zero, guided-autoencoder is equivalent to autoencoder. In this case, the prediction loss of autoencoder is similar to PCA. By introducing more and more guided-ratio, compressed representations with less prediction loss and more reconstruction loss can be obtained. Guided-autoencoders could achieve both smaller prediction loss than the linear model and smaller reconstruction loss than PCA, which indicates those compressed representations are both very informative of the age and contains essential information about the original data.

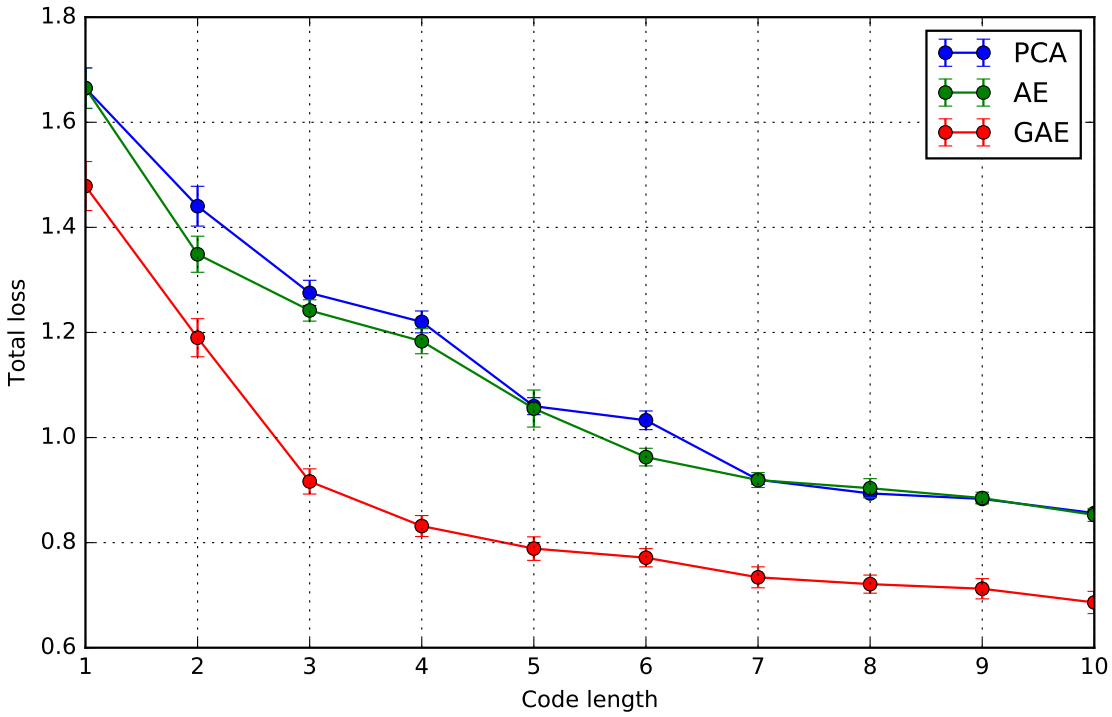

Figure 5.4: The total loss including prediction and reconstruction loss for PCA, autoencoder and guided-autoencoder with different code length.

5.3.3 Inflammatory age extraction

In order to provide a marker summarization of the subject's immune system state, a novel immunology measurement concept – *inflammatory age* – is invented. This novel measurement should be relevant to both the inflammatory protein level and the chronological age of the subject. In this work, the inflammatory age is defined as the output of a nonlinear function of inflammatory protein levels. This real value measurement should both be informative of the chronological age and contain the essential information about the inflammatory protein levels.

A guided-autoencoder is trained as the nonlinear transformation function for inflammatory age. As the total loss will always decrease when more code length is allowed. Hence, the best code length of the guided-autoencoder needs to be identified. In this work, the best code length is chosen as the one with non-significant improvement in the total loss when introducing one more code under a 5-fold cross-validation. The best code length is 5, in this case, as shown in Figure 5.6 .

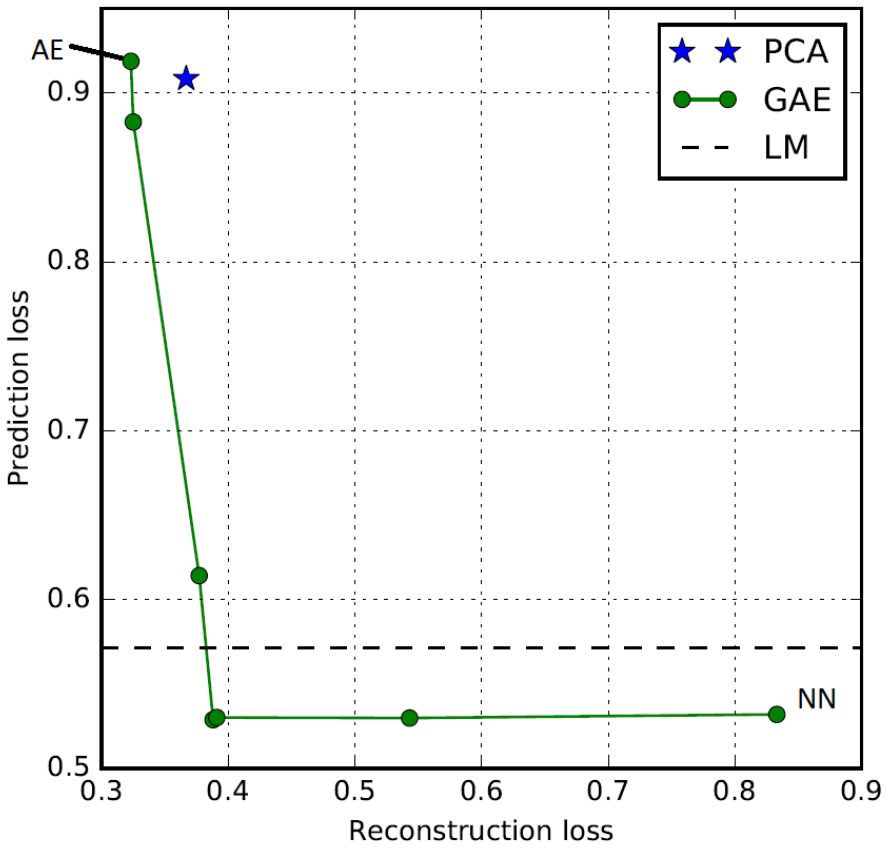

Figure 5.5: Test reconstruction loss and prediction loss for PCA and guided-autoencoders trained different guided-ratio with code length 3.

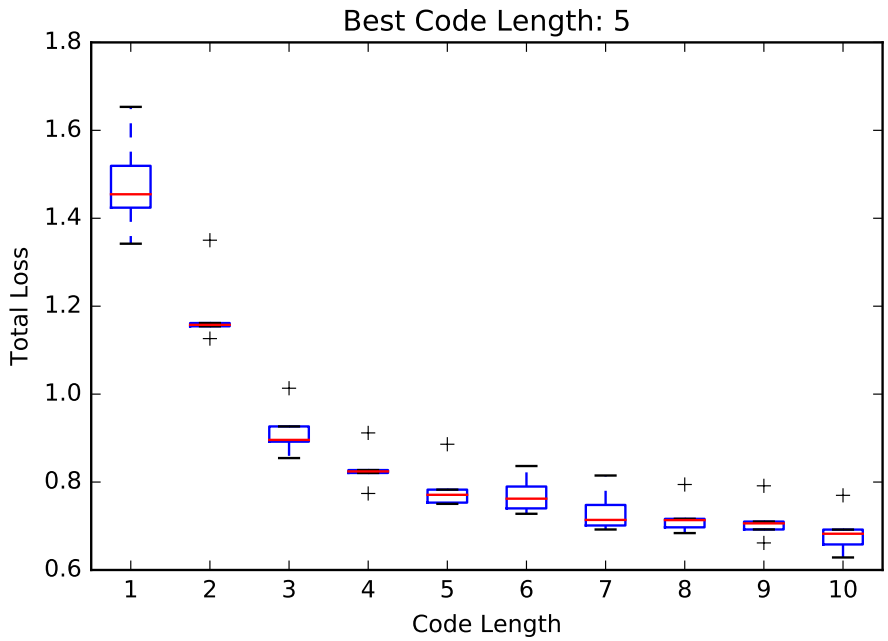

Figure 5.6: The 5-fold cross-validation total loss for guided-autoencoders with different code length. The best code length should have non-significant improvement in total loss when adding more code. In this case, the best code length is 5.

After obtaining the best code length as 5, a 5-fold-cross-validation is used to select the best hyper-parameter setting (depth = 2, guided-ratio = 0.2, weight-decay penalty = 0.001) on all guided-autoencoders with code length 5. The best hyper-parameter setting was finally used to train the guided-autoencoder on the whole dataset. The age predictive function from the final guided-autoencoder was used as the inflammatory age predictor.

The learned compressed representation colored by chronological age is shown in Figure 5.7. X-axis and y-axis represent the first and second dimension of the compressed representation (PCA for guided-autoencoders). The color indicates the chronological age of the subject (red for young and yellow for old). One can visually tell the separation between old and young subjects from the guided-autoencoders; while in PCA, subjects with different ages are clogged together.

5.4 Summarization

In this chapter, I provided a novel structure – guided-autoencoder – to learn informative compressed representations. Traditional autoencoder can be used to generate compressed representations,

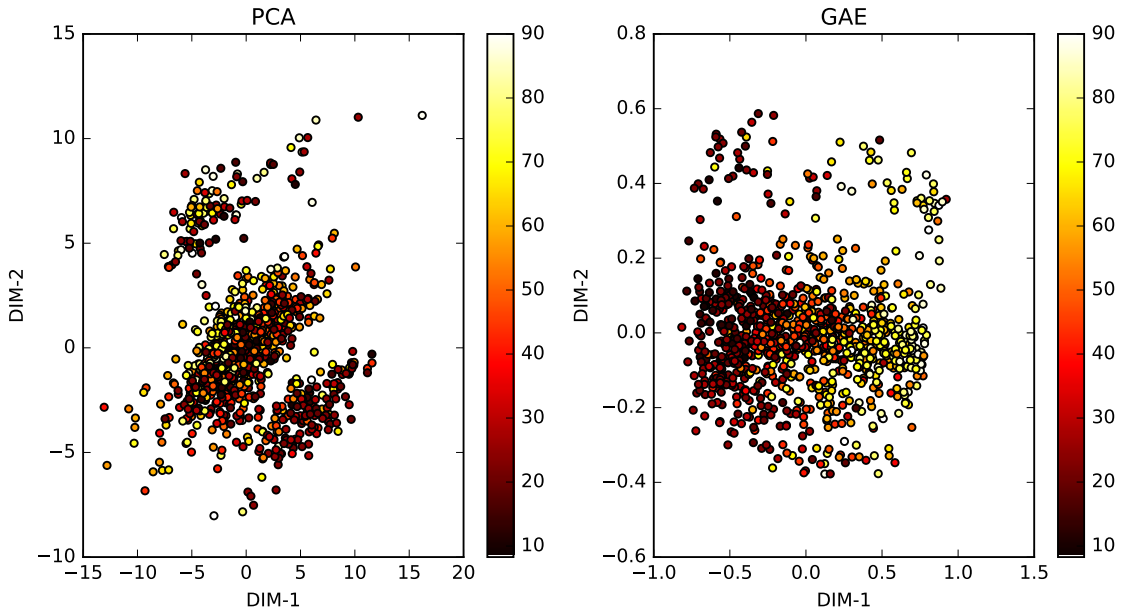

Figure 5.7: Visualization of compressed representation PCA and guided-autoencoders.

but without guided-training, the compressed representations could be not predictive of the information of interest. With the guided-autoencoder, compressed representations with small reconstruction loss and prediction loss can be obtained. In this case, the compressed representation contains both the information about the input data and the information of the target label of interest. By using different guided-ratio during the training, representations with different levels of reconstruction and prediction power can be obtained. Finally, the guided-autoencoder is used to extract an immunology state marker – inflammatory score. The score is a summary of the subject's immunology state and related to the chronological age.

CHAPTER 6: DISCUSSION

In this dissertation, I introduced 3 methods extracting information from deep learning models. Chapter 3 introduced a method for estimating degrees of freedom in deep learning models. In deep models, the number of parameters can be very large and can be an overestimate of the complexity of the model. Degrees of freedom is a more proper metric for measuring the complexity comparing to the number of parameters. A Monte-Carlo based method can be used to efficiently estimate the degrees of freedom in the model. The degrees of freedom estimated on an image classification model shows that with reasonable regularization techniques, deep learning models with many parameters can still be trained properly without overfitting issue. This provides strong evidence for applying deep learning methods on computational biology data with not many samples.

Chapter 4 introduced a general hypotheses generation framework for deep learning models. We can calculate the perturbation level and expected change from the hypotheses to evaluate their values. The testable and valuable hypotheses should have small perturbation level and large expected change. When designing the validation experiment on a set of candidate hypotheses, a cost-efficient algorithm is provided to select hypotheses that maximize the expected benefit under a total cost constraint. The proposed framework is used to generate hypotheses on a microbial synthetic community design task. The results indicate that the proposed framework can be used to efficiently generate hypotheses from deep learning models.

In Chapter 5, an informative representation learning method based on autoencoders is introduced. When learning a compressed representation of the data of high dimensionality, reconstruction loss is optimized under the constraint of code length used. Due to the re-parameterization issue, different models with similar reconstruction error can be trained, but some models can contain more informative representations comparing to others. Guided-autoencoder aims to minimize the

reconstruction loss and prediction loss on a specific guided target at the same time. By choosing different guided training ratio, representations of different focus can be generated. The guided-autoencoder method is applied to an immunology data to extract compressed representations that are informative of the chronological age of the subjects.

In this thesis, I showed that different kinds of information – complexity, hypothesis, informative representations – can be extracted from deep learning models. For a long-term goal, many more model interpretation related tasks can be developed to facilitate the information extraction of the deep model. The information can also be used as a feedback monitor during the training process to ensure the model is properly trained. The guided-autoencoder can also be used for generative adversarial networks (Goodfellow et al., 2014) to learn informative generative codes. Rules discovery – a specific perturbation pattern that is common holds in many contexts – for deep learning models is also of interest to many fields. I hope that the methods I introduced in this thesis can be used as a founding basis for the better intelligent system, and the information could be used for validation and reassurance that the deep learning is working properly.

BIBLIOGRAPHY

Akaike, H. (1974). A new look at the statistical model identification. *Automatic Control, IEEE Transactions on*, 19(6):716–723.

Akaike, H., Petrov, B., and Csaki, F. (1973). Information theory and an extension of the maximum likelihood principle.

Alipanahi, B., Delong, A., Weirauch, M. T., and Frey, B. J. (2015). Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*.

Angermueller, C., Pärnamaa, T., Parts, L., and Stegle, O. (2016). Deep learning for computational biology. *Molecular Systems Biology*, 12(7):878.

Bai, Y., Müller, D. B., Srinivas, G., Garrido-Oter, R., Potthoff, E., Rott, M., Dombrowski, N., Münch, P. C., Spaepen, S., Remus-Emsermann, M., et al. (2015). Functional overlap of the arabidopsis leaf and root microbiota. *Nature*.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N., and Bengio, Y. (2012). Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

Bertsekas, D. P. (1999). *Nonlinear programming*. Athena scientific Belmont.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Chen, Y., Li, Y., Narayan, R., Subramanian, A., and Xie, X. (2015). Gene expression inference with deep learning. *bioRxiv*, page 034421.

Chollet, F. et al. (2015). Keras. `https://github.com/fchollet/keras`.

Efron, B. (1975). Defining the curvature of a statistical problem (with applications to second order efficiency). *The Annals of Statistics*, pages 1189–1242.

Efron, B. (2004). The estimation of prediction error. *Journal of the American Statistical Association*, 99(467).

Efron, B., Hastie, T., Johnstone, I., Tibshirani, R., et al. (2004). Least angle regression. *The Annals of statistics*, 32(2):407–499.

Eldar, Y. C. (2009). Generalized SURE for exponential families: Applications to regularization. *Signal Processing, IEEE Transactions on*, 57(2):471–481.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660.

Faith, J. J., Ahern, P. P., Ridaura, V. K., Cheng, J., and Gordon, J. I. (2014). Identifying gut microbe–host phenotype relationships using combinatorial communities in gnotobiotic mice. *Science translational medicine*, 6(220):220ra11–220ra11.

Farabet, C., Couprie, C., Najman, L., and LeCun, Y. (2013). Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929.

Furman, D., Gao, T., Hastie, T., Sayed, N., Tibshirani, R., Rosenberg-Hasson, Y., Dekker, C., Haddad, F., Maecker, H., Jojic, V., Wu, J., Montoya, J., and Davis, M. (2017). Inflammation, cardiovascular disease and mortality risk : Results from the 1000 immunomes project. *Cell (to be submitted)*.

Gao, T. and Jojic, V. (2016). Degrees of freedom in deep neural networks. *arXiv preprint arXiv:1603.09260*.

Golub, G. H., Heath, M., and Wahba, G. (1979). Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Guyon, I., Hur, A. B., Gunn, S., and Dror, G. (2004). Result analysis of the nips 2003 feature selection challenge. In *Advances in Neural Information Processing Systems 17*, pages 545–552. MIT Press.

Hacquard, S., Garrido-Oter, R., González, A., Spaepen, S., Ackermann, G., Lebeis, S., McHardy, A. C., Dangl, J. L., Knight, R., Ley, R., et al. (2015). Microbiota and host nutrition across plant and animal kingdoms. *Cell host & microbe*, 17(5):603–616.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97.

Hinton, G. E. (1990). Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence*, 46(1-2):47–75.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length and helmholtz free energy. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 3–10. Morgan-Kaufmann.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

ILOG, I. (2014). Cplex optimization studio. *URL: http://www-01. ibm. com/software/commerce/optimization/cplex-optimizer*.

Janson, L., Fithian, W., and Hastie, T. J. (2015). Effective degrees of freedom: a flawed metaphor. *Biometrika*, page asv019.

Jolliffe, I. (2002). *Principal component analysis*. Wiley Online Library.

Jolliffe, I. T. (1986). Principal component analysis and factor analysis. In *Principal component analysis*, pages 115–128. Springer.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kolmogorov, A. N. (1965). Three approaches to the quantitative definition of information. *Problems of information transmission*, 1(1):1–7.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Cortes, C., and Burges, C. J. (1998). The MNIST database of handwritten digits.

Li, Y., Chen, C.-Y., and Wasserman, W. W. (2015). Deep feature selection: Theory and application to identify enhancers and promoters. In *Research in Computational Molecular Biology*, pages 205–217. Springer.

Lundberg, S. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. *arXiv preprint arXiv:1705.07874*.

MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

Min, S., Lee, B., and Yoon, S. (2016). Deep learning in bioinformatics. *arXiv preprint arXiv:1603.06430*.

Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Paredes, S. H., Gao, T., Law, T. F., Finkel, O. M., Mucyn, T., José, P., Teixeira, P. L., González, I. S., Feltcher, M. E., Powers, M. J., Shank, E. A., Jones, C. D., Jojic, V., Dangl, J. L., and Castrillo, G. (2017). A simplified framework for dissecting complex host-microbiota interactions. *PLoS biology (in submission)*.

Ramani, S., Blu, T., and Unser, M. (2008). Monte-carlo sure: A black-box optimization of regularization parameters for general denoising algorithms. *Image Processing, IEEE Transactions on*, 17(9):1540–1554.

Reeve, H. W. and Brown, G. (2017). Degrees of freedom in regression ensembles.

Rippel, O., Gelbart, M. A., and Adams, R. P. (2014). Learning ordered representations with nested dropout. *arXiv preprint arXiv:1402.0915*.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

Schmidhuber, J. (1997). Discovering neural nets with low kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873.

Schwarz, G. et al. (1978). Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464.

Seedorf, H., Griffin, N. W., Ridaura, V. K., Reyes, A., Cheng, J., Rey, F. E., Smith, M. I., Simon, G. M., Scheffrahn, R. H., Woebken, D., et al. (2014). Bacteria from diverse habitats colonize and compete in the mouse gut. *Cell*, 159(2):253–266.

Shrikumar, A., Greenside, P., and Kundaje, A. (2017). Learning important features through propagating activation differences. *arXiv preprint arXiv:1704.02685*.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Stein, C. M. (1981). Estimation of the mean of a multivariate normal distribution. *The annals of Statistics*, pages 1135–1151.

Tan, J., Hammond, J. H., Hogan, D. A., and Greene, C. S. (2016). Adage-based integration of publicly available pseudomonas aeruginosa gene expression data with denoising autoencoders illuminates microbe-host interactions. *mSystems*, 1(1):e00025–15.

Tan, J., Ung, M., Cheng, C., and Greene, C. S. (2014). Unsupervised feature construction and knowledge extraction from genome-wide assays of breast cancer with denoising autoencoders. In *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, volume 20, pages 132–143. NIH Public Access.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

Tibshirani, R., Saunders, M., Rosset, S., Zhu, J., and Knight, K. (2005). Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society Series B*, 67(1):91–108.

Vaiter, S., Deledalle, C., Peyré, G., Fadili, J. M., and Dossal, C. (2012). The Degrees of Freedom of the Group Lasso. In *International Conference on Machine Learning Workshop (ICML)*, Edinburgh, United Kingdom.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408.

Wang, S., Mohamed, A.-r., Caruana, R., Bilmes, J., Plilipose, M., Richardson, M., Geras, K., Urban, G., and Aslan, O. (2016). Analysis of deep neural networks with the extended data jacobian matrix. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 718–726.

Ye, J. (1998). On measuring and correcting the effects of data mining and model selection. *Journal of the American Statistical Association*, 93(441):120–131.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer.

Zhou, J. and Troyanskaya, O. G. (2015). Predicting effects of noncoding variants with deep learning-based sequence model. *Nature methods*, 12(10):931–934.

Zintgraf, L. M., Cohen, T. S., Adel, T., and Welling, M. (2017). Visualizing deep neural network decisions: Prediction difference analysis. *arXiv preprint arXiv:1702.04595*.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320.

Zou, H., Hastie, T., Tibshirani, R., et al. (2007). On the "degrees of freedom" of the lasso. *The Annals of Statistics*, 35(5):2173–2192.