NORTHWESTERN UNIVERSITY

Topics in Deep Learning Classification

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Industrial Engineering and Management Sciences

By

Jaehoon Koo

EVANSTON, ILLINOIS

June 2020

ABSTRACT

Topics in Deep Learning Classification

Jaehoon Koo

In machine learning, classification that assigns a label to a sample is a fundamental problem and serves a building block for various applications of artificial intelligence such as speech recognition, sentimental analysis, and image recognition. During the last years, deep learning rejuvenates artificial intelligence; in particular, it leads to tremendous progress in classification tasks. In this study, we develop enhanced deep learning methodologies for supervised classification. We also explore training schemes and implementations of the models using high-end computing machines. Furthermore, we study an interesting variant of the classification problem, called inverse classification that explores interpretability of classification models. This dissertation consists of three chapters, 1) Improved Classification Methods Based on Deep Belief Networks (DBN), 2) Combined Convolutional and Recurrent Neural Networks for Hierarchical Classification of Images, and 3) A New Framework for Inverse Classification Using Mixed Integer Programming.

In the first chapter, we explore how to incorporate unsupervised learning methods in supervised classification. Generative models are commonly used to initialize classifiers

before fine-tuning. Typically, this requires solving separate unsupervised and supervised learning problems. In this work, we focus on DBN, which is a widely used unsupervised model. We develop several supervised models incorporating DBN in order to improve the two-phase learning strategy. The improvements over two-phase are consistent.

In the second chapter, we focus on hierarchical classification of images. Object classes have known hierarchical relations, and classifiers exploiting these relations can perform better. To incorporate this perspective, we develop a combined model for classification that extracts hierarchical representations of images by a convolutional neural network and learns a tree of label paths to predict a final label of images by a recurrent neural network. The proposed model leads to image classification that captures the hierarchical characteristics of the classes.

In the third chapter, we shift our attention to studying interpretability of classification models rather than improving classification accuracy. We study an inverse classification problem that is a machine learning task designed to identify small changes needed in input features of an instance to adjust its associated prediction as desired. To solve this problem, we formulate a constrained mixed integer programming problem and design an associated algorithm based on Lagrangian and subgradient methods.

Acknowledgements

First and foremost, I would like to sincerely thank my advisor Professor Diego Klabjan for his limitless support and guidance during my years at Northwestern. I thank him for giving me opportunities to be exposed to many exciting challenges and helping me achieve my goal. I am very thankful to have such a great advisor.

I also want to thank my other committee members: Professor Aggelos Katsaggelos, Professor Edward Malthouse, and Dr. Jean Utke for being on my dissertation committee and providing constructive feedback.

I want to thank all of my folks, friends, and colleagues at Northwestern for cheering me up throughout the good times and hard times. I am also grateful to my friends Taehoon Han, Kyungsu Park, and Sangheum Hwang for always encouraging me to pursue my dreams.

Most importantly, I want to thank my family. I especially would like to thank my mom and dad who always embraced and believed in me, and encouraged me throughout my studies. Also, I would like to thank my sister Sunyoung and my adorable nephew Hojun. I send my love and gratitude to my family.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Improved Classification Based on Deep Belief Networks

1.1. Introduction

A Restricted Boltzmann machine (RBM), an energy-based model to define an input distribution, is widely used to extract latent features before classification. Such an approach combines unsupervised learning for feature modeling and supervised learning for classification. Two training steps are needed. The first step, called pre-training, is to model features used for classification. This can be done by training a RBM that captures the distribution of input. The second step, called fine-tuning, is to train a separate classifier based on the features from the first step (Larochelle et al., 2012). This two-phase training approach for classification can be also used for deep networks. Deep belief networks (DBN) are built with stacked RBMs, and trained in a layer-wise manner (Hinton and Salakhutdinov, 2006). Two-phase training based on a deep network consists of DBN and a classifier on top of it.

The two-phase training strategy has three possible problems. 1) It requires two training processes; one for training RBMs and one for training a classifier. 2) It is not guaranteed that the modeled features in the first step are useful in the classification phase since they are obtained independently of the classification task. 3) It can be difficult to decide which classifier to use. Therefore, there is a need for a method that can conduct feature modeling and classification concurrently (Larochelle et al., 2012).

To resolve these problems, recent papers suggest transforming RBMs to a model that can deal with both unsupervised and supervised learning. Since a RBM can calculate the joint and conditional probabilities, the suggested prior models combine a generative and discriminative RBM. Consequently, this hybrid discriminative RBM is trained concurrently for both objectives by summing the two contributions (Larochelle and Bengio, 2008; Larochelle et al., 2012). In a similar way, a self-contained RBM for classification is developed by applying the free-energy function based approximation to RBM, which is used for a supervised learning method, reinforcement learning (Elfwing et al., 2015). However, these approaches are limited to transforming RBM that is a shallow network.

In this study, we develop alternative models to solve a classification problem based on DBN. Viewing the two-phase training as two separate optimization problems, we apply optimization modeling techniques in developing our models. Our first approach is to design new objective functions. We design an expected loss function based on $p(h|x)$ built by DBN and the loss function of the classifier. Second, we introduce constraints that bound the DBN weights in the feed-forward phase. The constraints ensure that extracted features are good representations of the input during model training. Third, we apply bilevel programming to the two-phase training method. The bilevel model has a loss function of the classifier in its objective function but it constrains the DBN values to the optimal to phase-1. This model searches possible optimal solutions for the classification objective only where DBN objective solutions are optimal.

Our main contributions are several classification models combining DBN and a loss function in a coherent way. In the computational study we verify that the suggested models perform better than the two-phase method.

1.2. Literature Review

The two-phase training strategy is applied to many classification tasks on different types of data. Two-phase training with RBM and support vector machine (SVM) is explored in classification tasks on images, documents, and network intrusion data (Xing et al., 2005; Norouzi et al., 2009; Salama et al., 2011; Dahl et al., 2012). Replacing SVM with logistic regression is explored in Mccallum et al. (2006); Cho et al. (2011). Gehler et al. (2006) use the 1-nearest neighborhood classifier with RBM to solve a document classification task. Hinton and Salakhutdinov (2006) suggest a DBN consisting of stacked RBMs that is trained in a layer-wise manner. A two-phase method using DBNs and deep neural networks is used to solve various classification problems such as image and text recognition (Hinton and Salakhutdinov, 2006; Bengio and Lamblin, 2007; Sarikaya et al., 2014). Recently, this approach is applied to motor imagery classification in the area of brain–computer interface (Lu et al., 2017), biomedical research, classification of Cytochrome P450 1A2 inhibitors and non-inhibitors (Yu et al., 2017), web spam classification that detects web pages deliberately created to manipulate search rankings (Li et al., 2018), and human emotion recognition that classifies physiological signals such as "happy," "relaxed," "disgust," "sad," and "neutral" (Hassan et al., 2019). All these papers rely on two distinct phases, while our models assume a holistic view of both aspects.

Many studies are conducted to improve the problems of two-phase training. Most of the research is focused on transforming RBMs so that the modified model can achieve generative and discriminative objectives at the same time. Schmah et al. (2009) propose a discriminative RBM method, and subsequently classification is done in the manner of a Bayes classifier. However, this method cannot capture the relationship between the classes since the RBM for each class is trained separately. Larochelle and Bengio (2008); Larochelle et al. (2012) propose a self-contained discriminative RBM framework where the objective function consists of the generative learning objective $p(x, y)$, and the discriminative learning objective, $p(y|x)$. Both distributions are derived from RBM. Similarly, a self-contained discriminative RBM method for classification is proposed (Elfwing et al., 2015). The free-energy function based approximation is applied in the development of this method, which is initially suggested for reinforcement learning. This prior paper relies on the RBM conditional probability, while we handle general loss functions. Our models also hinge on completely different principles.

1.3. Background

Restricted Boltzmann Machines. RBM is an energy-based probabilistic model, which is a restricted version of Boltzmann machines (BM) that is a log-linear Markov Random Field. It has visible nodes x corresponding to input and hidden nodes h matching the latent features. The joint distribution of the visible nodes $x \in \mathbb{R}^J$ and hidden variable $h \in \mathbb{R}^I$ is defined as

$$p(x, h) = \frac{1}{Z} e^{-E(x,h)},\ E(x, h) = -hWx - ch - bx$$

where $W \in \mathbb{R}^{I \times J}$, $b \in \mathbb{R}^J$, and $c \in \mathbb{R}^I$ are the model parameters, and Z is the partition function. Since units in a layer are independent in RBM, we have the following form of conditional distributions:

$$p(h|x) = \prod_{i=1}^{I} p(h_i|x),\ p(x|h) = \prod_{j=1}^{J} p(x_j|h).$$

For binary units where $x \in \{0,1\}^J$ and $h \in \{0,1\}^I$, we can write $p(h_i = 1|x) = \sigma(c_i + W_i x)$ and $p(x_j = 1|h) = \sigma(b_j + W_j x)$ where $\sigma()$ is the sigmoid function. In this manner RBM with binary units is an unsupervised neural network with a sigmoid activation function. The model calibration of RBM can be done by minimizing negative log-likelihood through gradient descent. RBM takes advantage of having the above conditional probabilities which enable to obtain model samples easier through a Gibbs sampling method. Contrastive divergence (CD) makes Gibbs sampling even simpler: 1) start a Markov chain with training samples, and 2) stop to obtain samples after k steps. It is shown that CD with a few steps performs effectively (Hinton, 2002; Bengio, 2009).

Deep Belief Networks. DBN is a generative graphical model consisting of stacked RBMs. Based on its deep structure DBN can capture a hierarchical representation of input data. Hinton et al. (2006) introduced DBN with a training algorithm that greedily trains one layer at a time. Given visible unit x and ℓ hidden layers the joint distribution is defined as (Hinton et al., 2006; Bengio, 2009)

$$p(x, h^1, \cdots, h^\ell) = p(h^{\ell-1}, h^\ell) \left(\prod_{k=1}^{\ell-2} p(h^k | h^{k+1}) \right) p(x|h^1).$$

Since each layer of DBN is constructed as RBM, training each layer of DBN is the same as training a RBM.

Classification is conducted by initializing a network through DBN training (Hinton et al., 2006; Bengio and Lamblin, 2007). A two-phase training can be done sequentially by: 1) pre-training, unsupervised learning of stacked RBM in a layer-wise manner, and 2) fine-tuning, supervised learning with a classifier. Each phase requires solving an optimization problem. Given training dataset $D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(|D|)}, y^{(|D|)})\}$ with input x and label y, the pre-training phase solves the following optimization problem at each layer k

$$\min_{\theta_k} \quad \frac{1}{|D|}\sum_{i=1}^{|D|}\left[-log\, p(x_k^{(i)};\theta_k)\right]$$

where $\theta_k = (W_k, b_k, c_k)$ is the RBM model parameter that denotes weights, visible bias, and hidden bias in the energy function, and $x_k^{(i)}$ is visible input to layer k corresponding to input $x^{(i)}$. Note that in layer-wise updating manner we need to solve ℓ of the problems from the bottom to the top hidden layer. For the fine-tuning phase we solve the following optimization problem

$$\min_{\phi} \quad \frac{1}{|D|}\sum_{i=1}^{|D|}\left[\mathcal{L}(\phi; y^{(i)}, h(x^{(i)}))\right] \tag{1.1}$$

where $\mathcal{L}()$ is a loss function, h denotes the final hidden features at layer ℓ, and ϕ denotes the parameters of the classifier. Here for simplicity we write $h(x^{(i)}) = h(x_\ell^{(i)})$. When combining DBN and a feed-forward neural networks (FFN) with sigmoid activation, all the weights and hidden bias parameters among input and hidden layers are shared for both training phases. Therefore, in this case we initialize FFN by training DBN.

1.4. Proposed Models

We model an expected loss function for classification. Considering classification of two phase method is conducted on hidden space, the probability distribution of the hidden variables obtained by DBN is used in the proposed models. The two-phase method provides information about modeling parameters after each phase is trained. Constraints based on the information are suggested to prevent the model parameters from deviating far from good representation of input. Optimal solution set for unsupervised objective of the two-phase method is good candidate solutions for the second phase. Bilevel model has the set to find optimal solutions for the phase-2 objective so that it conducts the two-phase training in one-shot. We call our models combined models.

DBN Fitting Plus Loss Model. We start with a naive model of summing pre-training and fine-tuning objectives. This model conducts the two-phase training strategy simultaneously; however, we need to add one more hyperparameter ρ to balance the impact of both objectives. The model (DBN+Loss) is defined as

$$\min_{\theta_L, \theta_{DBN}} \mathbb{E}_{\mathbf{y},\mathbf{x}}[\mathcal{L}(\theta_L; \mathbf{y}, h(\mathbf{x}))] + \rho\, \mathbb{E}_{\mathbf{x}}[-\, log\, p(\mathbf{x}; \theta_{DBN})]$$

and empirically based on training samples D,

$$\min_{\theta_L, \theta_{DBN}} \frac{1}{|D|} \sum_{i=1}^{|D|} \left[\mathcal{L}(\theta_L; y^{(i)}, h(x^{(i)})) - \rho\, log\, p(x^{(i)}; \theta_{DBN})\right] \tag{1.2}$$

where θ_L, θ_{DBN} are the underlying parameters. Note that $\theta_L = \phi$ from (1.1) and $\theta_{DBN} = (\theta_k)_{k=1}$. This model has already been proposed if the classification loss function is based on the RBM conditional distribution (Larochelle and Bengio, 2008; Larochelle et al., 2012).

Expected Loss Model with DBN Boxing. We first design an expected loss model based on conditional distribution $p(h|x)$ obtained by DBN. This model conducts classification on the hidden space. Since it minimizes the expected loss, it should be more robust and thus it should yield better accuracy on data not observed. The mathematical model that minimizes the expected loss function is defined as

$$\min_{\theta_L, \theta_{DBN}} \quad \mathbb{E}_{\mathbf{y},\mathbf{h}|\mathbf{x}}[\mathcal{L}(\theta_L; \mathbf{y}, h(\theta_{DBN}; \mathbf{x}))]$$

and empirically based on training samples D,

$$\min_{\theta_L,\theta_{DBN}} \frac{1}{|D|}\sum_{i=1}^{|D|}\left[\sum_h p(h|x^{(i)})\mathcal{L}(\theta_L;y^{(i)},h(\theta_{DBN};x^{(i)}))\right].$$

With notation $h(\theta_{DBN};x^{(i)}) = h(x^{(i)})$ we explicitly show the dependency of h on θ_{DBN}. We modify the expected loss model by introducing a constraint that sets bounds on DBN related parameters with respect to their optimal values. This model has two benefits. First, the model keeps a good representation of input by constraining parameters fitted in the unsupervised manner. Also, the constraint regularizes the model parameters by preventing them from blowing up while being updated. Given training samples D the mathematical form of the model (EL-DBN) reads

$$\begin{aligned}&\min_{\theta_L,\theta_{DBN}} \frac{1}{|D|}\sum_{i=1}^{|D|}\left[\sum_h p(h|x^{(i)})\mathcal{L}(\theta_L;y^{(i)},h(\theta_{DBN};x^{(i)}))\right]\\ &\text{s.t.}\quad |\theta_{DBN}-\theta^*_{DBN}|\le\delta\end{aligned}$$

where θ^*_{DBN} are the optimal DBN parameters and δ is a hyperparameter. This model needs a pre-training phase to obtain the DBN fitted parameters.

Expected Loss Model with DBN Classification Boxing. Similar to the DBN boxing model, this expected loss model has a constraint that the DBN parameters are bounded by their optimal values at the end of both phases. This model regularizes parameters with those that are fitted in both the unsupervised and supervised manner. Therefore, it can achieve better accuracy even though we need an additional training to the two-phase trainings. Given training samples D the model (EL-DBNOPT) reads

$$\begin{aligned} &\min_{\theta_L,\theta_{DBN}} \frac{1}{|D|}\sum_{i=1}^{|D|}\left[\sum_h p(h|x^{(i)})\mathcal{L}(\theta_L; y^{(i)}, h(\theta_{DBN}; x^{(i)}))\right] \\ &\text{s.t.} \quad |\theta_{DBN} - \theta^*_{DBNOPT}| \leq \delta \end{aligned} \tag{1.3}$$

where θ^*_{DBNOPT} are the optimal values of DBN parameters after two-phase training and δ is a hyperparameter.

Feed-forward Network with DBN Boxing. We also propose a model based on boxing constraints where FFN is constrained by DBN output. The mathematical model (FFN-DBN) based on training samples D is

$$\begin{aligned} &\min_{\theta_L,\theta_{DBN}} \quad \frac{1}{|D|}\sum_{i=1}^{|D|}\left[\mathcal{L}(\theta_L; y^{(i)}, h(\theta_{DBN}; x^{(i)}))\right] \\ &\text{s.t.} \quad |\theta_{DBN} - \theta^*_{DBN}| \leq \delta. \end{aligned} \tag{1.4}$$

Feed-forward Network with DBN Classification Boxing. Given training samples D this model (FFN-DBNOPT), which is a mixture of (1.3) and (1.4), reads

$$\begin{aligned} &\min_{\theta_L,\theta_{DBN}} \quad \frac{1}{|D|}\sum_{i=1}^{|D|}\left[\mathcal{L}(\theta_L; y^{(i)}, h(\theta_{DBN}; x^{(i)}))\right] \\ &\text{s.t.} \quad |\theta_{DBN} - \theta^*_{DBNOPT}| \leq \delta. \end{aligned}$$

Bilevel Model. We also apply bilevel programming to the two-phase training method. This model searches optimal solutions to minimize the loss function of the classifier only where DBN objective solutions are optimal. Possible candidates for optimal solutions of the first level objective function are optimal solutions of the second level objective function.

This model (BL) reads

$$\begin{aligned} \min_{\theta_L, \theta^*_{DBN}} \quad & \mathbb{E}_{\mathbf{y},\mathbf{x}}[\mathcal{L}(\theta_L; \mathbf{y}, h(\theta^*_{DBN}; \mathbf{x}))] \\ \text{s.t.} \quad & \theta^*_{DBN} = \arg\min_{\theta_{DBN}} \ \mathbb{E}_{\mathbf{x}}[-log\, p(\mathbf{x}; \theta_{DBN})] \end{aligned}$$

and empirically based on training samples,

$$\begin{aligned} \min_{\theta_L, \theta^*_{DBN}} \quad & \frac{1}{|D|} \sum_{i=1}^{|D|} \left[\mathcal{L}(\theta_L; y^{(i)}, h(\theta^*_{DBN}; x^{(i)})) \right] \\ \text{s.t.} \quad & \theta^*_{DBN} = \arg\min_{\theta_{DBN}} \ \frac{1}{|D|} \sum_{i=1}^{|D|} \left[-log\, p(x^{(i)}; \theta_{DBN}) \right]. \end{aligned}$$

One of the solution approaches to bilevel programming is to apply Karush–Kuhn–Tucker (KKT) conditions to the lower level problem. After applying KKT to the lower level, we obtain

$$\begin{aligned} \min_{\theta_L, \theta^*_{DBN}} \quad & \mathbb{E}_{\mathbf{y},\mathbf{x}}[\mathcal{L}(\theta_L; \mathbf{y}, h(\theta^*_{DBN}; \mathbf{x}))] \\ \text{s.t.} \quad & \nabla_{\theta_{DBN}} \mathbb{E}_{\mathbf{x}}[-log\, p(\mathbf{x}; \theta_{DBN})|_{\theta^*_{DBN}}] = 0. \end{aligned}$$

Furthermore, we transform this constrained problem to an unconstrained problem with a quadratic penalty function:

$$\min_{\theta_L, \theta^*_{DBN}} \ \mathbb{E}_{\mathbf{y},\mathbf{x}}[\mathcal{L}(\theta_L; \mathbf{y}, h(\theta^*_{DBN}; \mathbf{x}))] + \frac{\mu}{2} ||\nabla_{\theta_{DBN}} \mathbb{E}_{\mathbf{x}}[-log\, p(\mathbf{x}; \theta_{DBN})]|_{\theta^*_{DBN}}||^2 \tag{1.5}$$

where μ is a hyperparameter. The gradient of the objective function is derived in the appendix.

1.5. Computational Study

To evaluate the proposed models classification tasks on three datasets are conducted: the KDD'99 network intrusion dataset (NI)[1], the isolated letter speech recognition dataset (ISOLET) [2], a collection of newswire articles (Reuters)[3], and the MNIST hand-written images [4]. The experimental results of the proposed models on these datasets are compared to the results of the two-phase method.

In FFNs, we use the sigmoid function in the hidden layers and the softmax function in the output layer, and negative log-likelihood is used as the loss function. We select the hyperparameters based on the settings used in Wang and Klabjan (2017), which are fine-tuned. We first implement the two-phase method with DBNs of 1, 2, 3 and 4 hidden layers to find the best configuration for each dataset, and then apply the best configuration to the proposed models.

Implementations are done in Theano using GeForce GTX TITAN X. We use the mini-batch gradient descent method to solve the optimization problems for each model. To calculate the gradients of each objective function of the models Theano's built-in functions, 'theano.tensor.grad,' is used. We denote the two-phase approach as 2-Phase.

1.5.1. Network Intrusion

The classification task on NI is to distinguish between normal and bad connections given the related network connection information. The preprocessed dataset consists of 41 input features and 5 classes, and 4,898,431 examples for training and 311,029 examples for testing.

[1]kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[2]archive.ics.uci.edu/ml/datasets/ISOLET
[3]archive.ics.uci.edu/ml/datasets/reuters-21578+text+categorization+collection
[4]yann.lecun.com/exdb/mnist/

The experiments are conducted on 20%, 30%, and 40% subsets of the whole training set, which are obtained by stratified random sampling.We use the following hyperparameters. Each layer has 15 hidden units and is trained for 100 epochs with learning rate 0.01 during pre-training, and the whole network is trained for 500 epochs with learning rate 0.1 during fine-tuning. The mini-batch size is 1,000, and ρ in the DBN+Loss and μ in the BL model are diminishing during epochs.

On NI the best structure of 2-Phase is 41-15-15-5 for all three datasets, and so we compare it to the proposed models with the same sized networks. We compute the means of the classification errors and their standard deviations for each model averaged over 5 random runs. In each table, we stress in bold the best three models with ties broken by standard deviation. Table 1.1 shows the experimental results of the proposed models with the same network as the best 2-Phase. BL performs the best in all datasets, achieving the lowest mean classification error without the pre-training step. The difference in the classification error between our best model, BL, and 2-Phase is statistically significant as the p-values are 0.03, 0.01, and 0.03 for 20%, 30%, and 40% datasets, respectively. This shows that the model being trained concurrently for unsupervised and supervised purpose can achieve better accuracy than the two-phase method. Furthermore, both EL-DBNOPT and FFN-DBNOPT yield similar to, or lower mean error rates than 2-Phase in all of the three subsets.

1.5.2. ISOLET

The classification on ISOLET is to predict which letter-name is spoken among the 26 English alphabets given 617 input features of the related signal processing information.

Table 1.1. Classification errors with respect to the best DBN structure on NI

	20% dataset		30% dataset		40% dataset	
	Mean	Sd.	Mean	Sd.	Mean	Sd.
2-Phase	8.14%	0.12%	8.18%	0.12%	8.06%	0.02%
DBN+Loss	**8.07%**	**0.06%**	**8.13%**	**0.09%**	**8.05%**	**0.05%**
EL-DBN	8.30%	0.09%	8.27%	0.07%	8.29%	0.14%
EL-DBNOPT	8.14%	0.14%	8.15%	0.15%	8.08%	0.10%
FFN-DBN	8.17%	0.09%	8.20%	0.08%	8.07%	0.11%
FFN-DBNOPT	**8.07%**	**0.12%**	**8.12%**	**0.11%**	**7.95%**	**0.11%**
BL	**7.93%**	**0.09%**	**7.90%**	**0.11%**	**7.89%**	**0.10%**

The dataset consists of 5,600 for training, 638 for validation, and 1,559 samples for testing. We use the following hyperparameters. Each layer has 1,000 hidden units and is trained for 100 epochs with learning rate 0.005 during pre-training, and the whole network is trained for 300 epochs with learning rate 0.1 during fine-tuning. The mini-batch size is 20, and ρ in the DBN+Loss and μ in the BL model are diminishing during epochs.

In this experiment the shallow network performs better than the deep network; 617-1000-26 is the best structure for 2-Phase. One possible reason for this is that the training set does not include many samples. EL models perform well on this dataset. EL-DBNOPT achieves the best mean classification error, tied with FFN-DBNOPT. With the same training effort, EL-DBN achieves a lower mean classification error and smaller standard deviation than the two-phase method, 2-Phase. Considering a relatively small sample size of ISOLET, EL shows that it yields better accuracy on unseen data as it minimizes the expected loss, i.e., it generalizes better. In this data set, p-value is 0.07 for the difference in the classification error between our best model, FFN-DBNOPT, and 2-Phase.

Table 1.2. Classification errors with respect to the best DBN structure for ISOLET.

	Mean	Sd.
2-Phase	3.94%	0.22%
DBN+Loss	4.38%	0.20%
EL-DBN	**3.91%**	**0.18%**
EL-DBNOPT	**3.75%**	**0.14%**
FFN-DBN	3.94%	0.19%
FFN-DBNOPT	**3.75%**	**0.13%**
BL	4.43%	0.18%

1.5.3. Reuters

Reuters is a public dataset of newswire articles, used to predict 52 news categories given 2,000 input features of the most common words. The dataset consists of 6,532 samples for training and validation, and 2,568 samples for testing. We use the following hyperparameters. Each layer has 500 hidden units and is trained for 100 epochs with learning rate 0.1 during pre-training, and the whole network is trained for 500 epochs with learning rate 0.1 during fine-tuning. The mini-batch size is 50, and ρ in the DBN+Loss and μ in the BL model are decreased during epochs.

On this dataset, 2000-500-52 is the best structure for 2-Phase; the shallow network performs better than the deep network. As we pointed out in ISOLET, a small training set is one possible reason for this. As Table 1.3 shows, FFN-DBNOPT achieves the best mean classification error. It is statistically significant as p-value is 0.01 for the difference in the classification error between our best model, FFN-DBNOPT, and 2-Phase. We find that our combined models, BL and DBN+Loss, obtain better test accuracy than 2-Phase.

Table 1.3. Classification errors with respect to the best DBN structure for Reuters.

	Mean	Sd.
2-Phase	10.09%	0.16%
DBN+Loss	**9.77%**	**0.15%**
EL-DBN	15.83%	0.12%
EL-DBNOPT	**9.65%**	**0.18%**
FFN-DBN	10.09%	0.13%
FFN-DBNOPT	**9.60%**	**0.25%**
BL	9.79%	0.18%

1.5.4. MNIST

The task on the MNIST is to classify ten digits from 0 to 9 given by 28×28 pixel hand-written images. The dataset is divided in 60,000 samples for training and validation, and 10,000 samples for testing. We use the following hyperparameters. Each layer has 1,000 hidden units and is trained for 100 epochs with learning rate 0.01 during pre-training, and the whole network is trained for 300 epochs with learning rate 0.1 during fine-tuning. The mini-batch size is 10, and ρ in the DBN+Loss and μ in the BL model are diminishing during epochs. Note that DBN+Loss and BL do not require pre-training.

2-Phase with three-hidden layers of size, 784-1000-1000-1000-10, is the best. In Table 1.4, the best mean test error rate is achieved by FFN-DBNOPT, 1.32%. Furthermore, the models with the DBN classification constraints, EL-DBNOPT and FFN-DBNOPT, perform similar to, or better than the two-phase method. This shows that DBN classification boxing constraints regularize the model parameters by keeping a good representation of input.

Table 1.4. Classification errors with respect to the best DBN structure for the MNIST.

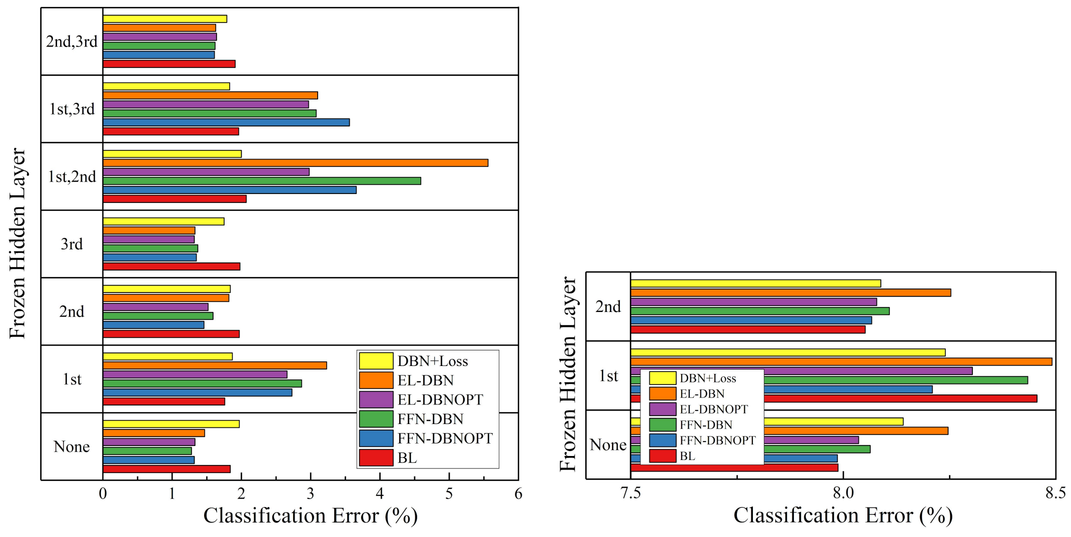
	Mean	Sd.
2-Phase	**1.33%**	**0.03%**
DBN+Loss	1.84%	0.14%
EL-DBN	1.46%	0.05%
EL-DBNOPT	**1.33%**	**0.04%**
FFN-DBN	1.34%	0.04%
FFN-DBNOPT	**1.32%**	**0.03%**
BL	1.85%	0.07%

1.5.5. Ablation Study

We conduct an ablation study in order to understand various aspects of our models. First, we study which part of our models is the most influential in classification accuracy. Second, we study how the size of training data affects the performance of our models.

1.5.5.1. Hybrid 2-Phase and Combined Model. In this ablation study, we examine which part of our networks contributes the most to classification accuracy if some parts of the network are trained based on 2-Phase and the other part by a combined model. We conduct ablation experiments by freezing layer-wise. For a network with three hidden layers, one case is to freeze a hidden layer with weights from 2-Phase while the rest is trained by a combined model; and the other case is to freeze any two hidden layers by 2-Phase weights and the rest is trained by a combined model. In this study, we use MNIST and NI (40% training set) since their best model structure has more than two hidden layers.

Figure 1.1 shows classification errors on the test set for MNIST and NI. In both datasets, we find that freezing the top hidden layer obtains higher classification accuracy than freezing the lower layers. We conclude that lower hidden layers in our networks contribute to performance more than higher layers. In addition, based on the MNIST experiment, we find that freezing two hidden layers yields worse results than freezing one layer. This further affects that the combined model is beneficial. In this setting, we also observe that the lowest hidden layer contributes the most to classification accuracy since freezing the top two hidden layers returns a higher accuracy than freezing the other two layers.



(a) MNIST (b) NI

Figure 1.1. Experimental results for hybrid 2-Phase and combined model

1.5.5.2. Impact of Samples. We also conduct an ablation study to understand if our models are affected by the size of training samples. We select the NI dataset as it has the largest set of training samples, and conduct the ablation study on our best model, BL, for this dataset. In order to carry out a meaningful ablation study with respect to the effect of samples on BL, we formulate a model by combining BL and 2-Phase and training on all samples, but one portion of samples is subject to BL while the remaining samples use

2-Phase. Formally, given training dataset $D = S \cup \bar{S}$, our ablation model reads

$$\min \quad \frac{1}{|S|}\sum_{i\in|S|}\mathcal{J}^i_{\text{BL}} + \frac{1}{|\bar{S}|}\sum_{i\in|\bar{S}|}\mathcal{L}^i_{\text{2-Phase}}$$

where $\mathcal{J}_{\text{BL}}$ denotes the loss function of BL as defined in (1.5), and $\mathcal{L}_{\text{2-Phase}}$ denotes the loss function of 2-Phase. Model weights are shared by both models. We use the same setting of hyperparameters as in Section 1.5.1. We conduct experiments on different sizes of S and $\bar{S}$, and each experiment uses five random runs to create each S by sampling from training set. Figure 1.2 shows test accuracy on different sizes of S. Note that $\frac{|S|}{|D|} = 0$ corresponds to pure 2-Phase while $\frac{|S|}{|D|} = 1$ means using solely BL on all samples. We observe that as we increase the size of S, the classification error on test decreases. In addition, we find that test error drops sharply once the BL is actually introduced (ratio = 0.2). We conclude that the impact of BL is very pronounced. Even if BL is used only a small fraction of samples (while the remaining samples are treated by 2-Phase), it improves the performance significantly.

From both studies, we conclude that using a combined model only a subset of a network or samples has a significant benefit.

1.6. Conclusion

DBN+Loss performs better than two-phase training 2-Phase in two instance. Aggregating two unsupervised and supervised objectives is effective. Second, the models with DBN boxing, EL-DBN and FFN-DBN, do not perform better than 2-Phase in almost all datasets. Regularizing the model parameters with unsupervised learning is not so effective in solving a supervised learning problem. Third, the models with DBN classification

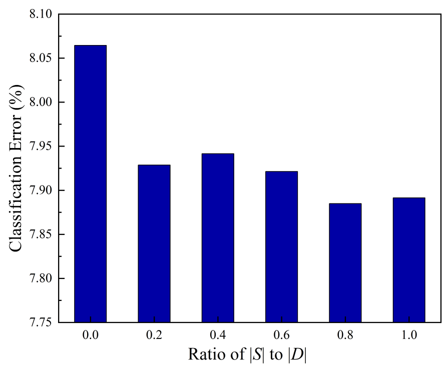


Figure 1.2. The impact of samples on NI

boxing, EL-DBNOPT and FFN-DBNOPT, perform better than 2-Phase in almost all of the experiments. FFN-DBNOPT is consistently one of the best three performers in all instances. This shows that classification accuracy can be improved by regularizing the model parameters with the values trained for unsupervised and supervised purpose. One drawback of this approach is that one more training phase to the two-phase approach is necessary. Last, BL shows that one-step training can achieve a better performance than two-phase training.

CHAPTER 2

Combined Convolutional and Recurrent Neural Networks for Hierarchical Classification of Images

2.1. Introduction

In computer vision, allocating labels to images is a fundamental problem, and it serves as a building block for various image recognition tasks such as image localization, object detection, and scene parsing (Hu et al., 2016). Over the past years, deep learning methods have made tremendous progress in these classification tasks. Especially, many approaches based on convolutional neural networks (CNNs) made significant advances in large-scale image classification (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; He et al., 2016; Hu et al., 2018; Woo et al., 2018). It is common to assume that separability of object categories is pronounced (Yan et al., 2015), and a multi-class or binary classifier is selected to label images (Hu et al., 2016).

Object categories in some settings are related to each other by means of a taxonomy. This phenomenon is typically present in datasets with a large number of categories (Yan et al., 2015). The categories of images can be represented by a tree based on two types of hierarchies: 1) Has-A hierarchy is present when each parent node physically contains some parts of each child node, and 2) Is-A hierarchy is exhibited when a parent node semantically contains child nodes, i.e. a child object is a type of the parent object. Models that can exploit details of objects lead to better classification performance. In some object

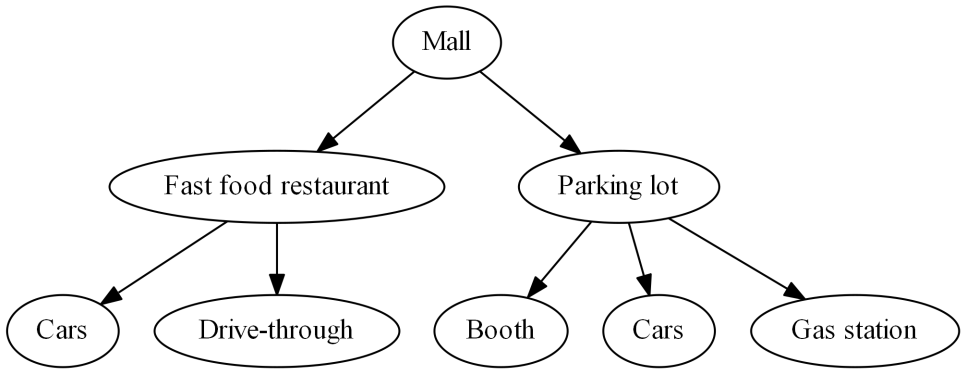


Figure 2.1. An example of a class tree

classification tasks, objects contain detailed objects, and reliably classifying high level objects lead to classifying detailed objects correctly. In such settings, we can build a Has-A hierarchical tree of categories, and models that can capture hierarchical relationships are required. Consider an investment or commercial real estate firm relying on satellite images of malls to, for example, gauge investments. We have objects of 'Booth,' 'Cars,' and 'Gas station' contained in images of parking lots. These categories have Has-A relationships, and a hierarchical tree of classes can be built as shown in Figure 2.1. To classify an image of 'Booth,' we need a model to find a path of 'Mall'-'Parking lot'-'Booth.' In this work, we consider the classification problem where we are given a tree of classes, and for an image we need to assign 'a path' in the tree.

We propose hierarchical classification models for images, named *deep hierarchical neural networks*, that extract hierarchical representations of images from a CNN and, by using a recurrent neural network (RNN), find a label path in the hierarchical class tree to predict labels of an image. Recent studies reveal that CNN features learn hierarchical representations of images at different layers representing an image ranging from detailed, part-level, to abstract, object-level (Yosinski et al., 2015). Part-level representations are typically captured at lower layers of the CNN, and object-level representations are learned at higher layers. Because of this insight, it is conceivable to associate the different level feature maps with the different depth layers in the hierarchical label tree. High level features should be able to classify top layers in the tree while low level features focusing on details are suitable to predict classes in the bottom layers of the tree. It is natural to view a path in the tree as a sequence and then to model it via an RNN. For these reasons, we combine an RNN or sequence-to-sequence network (S2S) to classify a target sequence with a CNN. As a result we predict target paths rather than a single label. The proposed networks consist of three parts: 1) a CNN takes a raw image as input, and produces convolutional features at each layer, 2) the features at different layers of the CNN are converted to a vector of fixed dimension, and 3) an RNN or S2S takes the converted CNN features as input, and outputs predictions at each level of the label tree. Figure 2.2 presents the structure of the proposed models.

To facilitate training of our compound model, we apply an alternating training scheme between the CNN and RNN sub models. Under this scheme, we alternate updating one while keeping the other frozen in the beginning of training and then unfreeze the entire network in the final phase of training. Such a scheme is needed because each sub model pursues different learning purposes in that the CNN learns representations of images and the RNN learns sequential behaviors of the classes. Alternating prevents both learning tasks from diverging in the early stage of training, consequently leading to better classification performance. In addition, different methods such as a linear, convolutional, and pooling operation are used to coerce the varying dimensions of the CNN features to the fixed dimension vector the RNN takes as input. The pooling retains much of the spatial information of the trained CNN features and avoids additional trainable parameters.

In our study, we use a real world, proprietary dataset of images from the insurance industry and a public dataset, Open Images (Krasin et al., 2017). Categories of both datasets have mainly Has-A relationships. We compare our models to state-of-the-art CNNs, and find that our models perform better. We conclude that our models can learn a hierarchical tree with both fixed- and variable-length target paths.

The main contributions of this work are as follows.

(1) We suggest a new structure of deep neural networks for hierarchical classification of images. Our models extract features from different CNN layers, and feed them to an RNN or S2S to learn a hierarchical path of categories. Our models can learn both fixed- and variable-length target paths; CNN-RNN are for fixed and CNN-S2S are for variable path lengths.

(2) We apply residual learning to the RNN part in order to facilitate training of our compound model and improve generalization of the model.

The rest of this chapter is organized as follows. In Section 2.2, the related literature is discussed. Section 2.3 describes the proposed models, and Section 2.4 provides a computational study including experimental details and analysis of the experimental results. Conclusions are given in Section 3.6.

2.2. Related Work

Hierarchical structures have been studied for image recognition by using standard computer vision (Tousch et al., 2012). Related literature is categorized based on how a

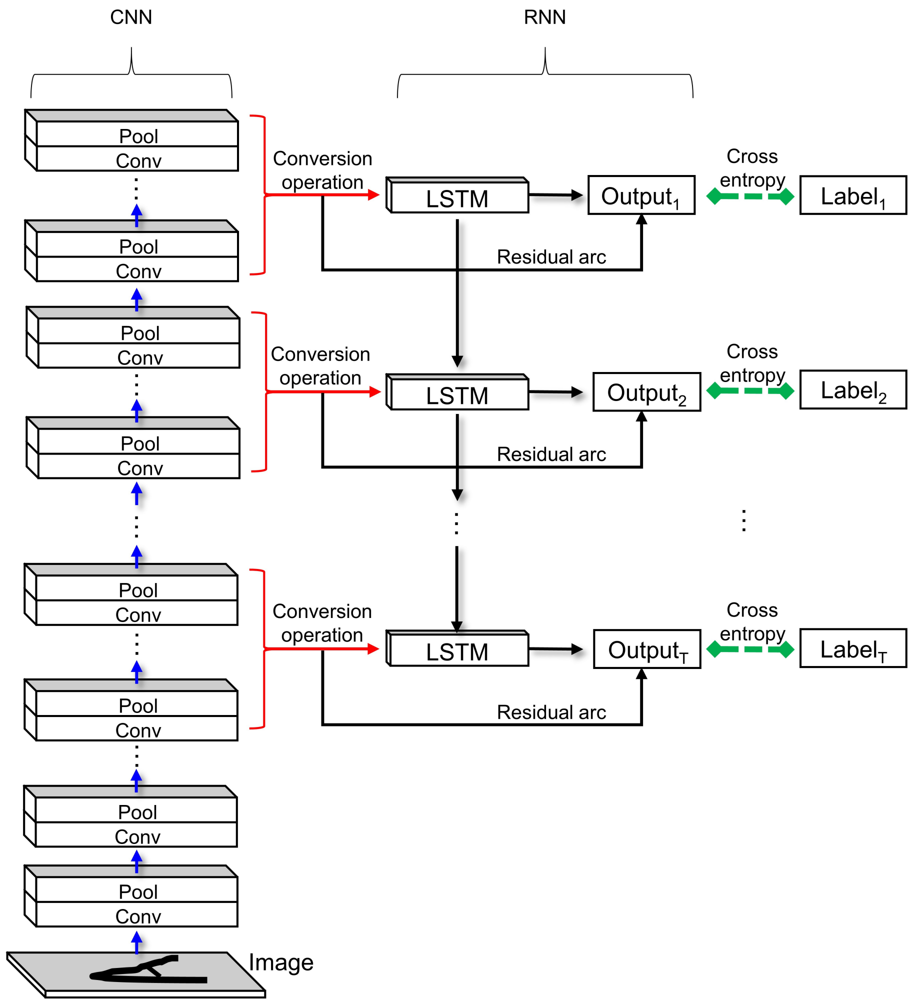


Figure 2.2. The model

hierarchy is constructed (Yan et al., 2015); a hierarchy is predefined in Marszalek and Schmid (2007); Deng et al. (2012); Verma et al. (2012); Jia et al. (2013), and it is trained by top-down and bottom-up methods in Marszalek and Schmid (2008); Sivic et al. (2008); Li et al. (2010); Deng et al. (2011); Salakhutdinov et al. (2011); Bannour and Hudelot (2012); Liu et al. (2013).

In the past, researchers adapted CNNs to hierarchical classification. Srivastava and Salakhutdinov (2013) introduce CNNs to hierarchical classification. Their proposed method improves the performance of minority classes over standard CNN by incorporating priors imposed by a tree structure of the classes. Xiao et al. (2014) suggest CNN based hierarchical networks; each branch model predicts a super-class, and leaf models return final predictions. Yan et al. (2015) suggest a hierarchical deep neural net that embeds CNNs into a two-level hierarchy of easy and difficult classes where the hierarchy is built automatically. The model uses coarse category classifiers for easy classes, and fine category classifiers for difficult classes. Schwing and Urtasun (2015) propose a method for hierarchical semantic segmentation. They combine a Markov random field model that is used for segmentation with a CNN to extract image representations. All these works rely on using CNNs in their models to obtain a better feature learner for images while we approach the problem from the perspective of improving prediction of target label paths by combining an RNN or S2S with CNNs.

Approaches combining CNNs and RNNs have been studied to solve different image classification tasks such as scene parsing, object detection, image captioning, etc. Such CNN-RNN frameworks use the final feature map from the CNN and use it as an input to RNN (possibly combined with other features such as caption). Such a network takes advantages of the CNN's representational feature learning over images and the RNN's high performance in capturing sequential information. Deng et al. (2016) propose an RNN that trains a graph structure for recognition of group activities. Stewart et al. (2016) propose a model for object detection. The proposed model combines a CNN that encodes an image into features with an LSTM that decodes the encoded information into a set of people detections. In Liang and Hu (2015), a CNN-RNN model for object recognition is suggested by incorporating recurrent connections into each convolutional layer. It improves capturing of context information, which is important for object recognition. Wang et al. (2016) propose a CNN-RNN framework for multi-label image classification. The proposed model produces class probabilities by concatenating CNN features and outputs of an RNN that takes a label vector as input. Shi et al. (2015) propose a convolutional Long Short-Term Memory (ConvLSTM) in which convolutional operations are embedded in every LSTM layer. They show that ConvLSTM captures spatiotemporal correlations. Guo et al. (2018) suggest several models to classify coarse- and fine-level categories of a semantic hierarchy; one of their models combines CNN and RNN so that top CNN features are input to RNN. Our approach is different from these methods since we exploit CNN features at each layer rather than only at the top layer. ConvLSTM overlays an RNN to each layer however its purpose is completely different; it does not focus on hierarchical classes but rather on sequences of images.

Recent papers suggest methods that consider CNN features from different layers, not only from the top layer, for hierarchical classification. Zhu and Bain (2017) suggest methods that take features at different middle layers of a CNN for coarse classes, and those at the top CNN layer for fine classes. Their network does not correlate the extracted CNN features to the final prediction; the extracted CNN features are trained independently without considering them as a sequence. Wehrmann et al. (2018) also propose a method considering features from middle layers of deep neural networks. They introduced an RNN to fit a hierarchical tree by inputting the extracted features of a feed forward network. Their network feeds raw inputs from each layer to RNN (which is possible if all layers have the same number of neurons; not the case in CNNs) and its performance on image recognition tasks based on CNN is not studied. They also do not introduce the notion of residual arcs which we find to be of great importance and they do not consider a S2S setting which is required if paths in the tree are of different length.

2.3. Proposed Models

In this section, we describe the proposed models that predict target paths in a hierarchical class tree. Our models extract hierarchical features from a CNN taking an image as input, and feed the extracted features to an RNN if tree paths have the same lengths. The RNN part is replaced by S2S if tree paths have variable lengths. For this reason, we present two models, CNN-RNN and CNN-S2S.

2.3.1. Fixed Path Length Tree Model: CNN-RNN

We propose a hierarchical fixed path length (FPL) classification model to fit a class tree that has target paths with a fixed-length. To this end, we are given a rooted tree R where each node corresponds to a class. We assume that each leaf node is of the same depth $T+1$. The root node corresponds to an artificial class. A training sample consists of (x, y) where x is an image and y is a path from the root node to a leaf in R. By our assumption on R, every y has the sample number T of labels. In this model, an RNN is combined with a CNN. Our model starts with a CNN, a feature learner, that extracts hierarchical features representing part-level and object-level of images. A CNN is used since CNN features learn spatial representations of images through its local-connectivity of the networks, i.e. the features are learned locally; and the extracted features at different layers have hierarchical relations (Zeiler and Fergus, 2014; Yosinski et al., 2015). In order to feed features from different CNN layers to the RNN, a conversion process is required. The dimensions of CNN features at each layer are different for combinations of convolution and pooling layers. However, RNN input dimensions at each step should be the same. To solve this we introduce a process that converts variable dimension CNN features to a fixed dimension vector. As we view a path in the tree as a sequence, we model it via an RNN (Graves et al., 2009). Taking converted CNN features as input, the RNN is trained to produce predictions of the target path y in the class tree.

Formally, a network of L convolution-pooling layers is defined as

$$a_l = f\left(a_{l-1}; \phi_l\right) \quad \text{for } l = 1, 2, \ldots, L$$

where a_0 is input image x, ϕ_l are model parameters at layer l, and f is a convolution-pooling function. Note that $a_l \in \mathbb{R}^{D_l \times W_l \times H_l}$; where D_l, W_l, and, H_l denote depth, width, and height at the l^{th}-layer. The general form of the conversion operation of CNN features, a_s, fed to the RNN is defined as

$$u_t = \frac{1}{|S_t|} \sum_{s \in S_t} g(a_s; \alpha_s) \quad \text{for } t = 1, 2, \ldots, T$$

where $S_t \subseteq \{1, \ldots, L\}$ is a subset of the CNN layers at each step t of the RNN such that the subsets are "increasing;" i.e. for every $1 \leq n < T$ we have if $i \in S_n, j \in S_{n+1}$, then $i < j$. Also, g is a function of converting CNN outputs into RNN inputs, i.e. $g : \mathbb{R}^{D_s \times W_s \times H_s} \times \mathbb{R}^{\nu_s} \rightarrow \mathbb{R}^p$ where p is a dimension of the RNN input at each step and $\alpha_s \in \mathbb{R}^{\nu_s}$ are possible trainable model parameters. Conversion methods are discussed in Section 2.3.3. The RNN takes converted fixed dimension CNN features $u^t \in \mathbb{R}^p$ as inputs, and predicts labels for each layer of the class tree. The RNN is governed by

$$\begin{aligned} h_t &= r_h\left(u_t, h_{t-1}; \theta_h\right), \text{ and} \\ o_t &= r_o\left(h_t; \theta_o\right) \quad \text{for } t = 1, 2, \ldots, T \end{aligned} \tag{2.1}$$

where h_0 is an initial hidden state, r_h and r_o are the state transition and output functions, and θ_h and θ_o are trainable parameters.

The loss function reads

$$\sum_{t=1}^{T} w_t \cdot CE(y_t, \text{softmax}(o_t))$$

where w_t represents weight for level t in R and CE denotes the cross entropy. The aim is for o_t to predict a node in R at level t. By definition of RNN all o_t have to have the same dimension. However, the number of classes at each level in R varies. For this reason, we have $o_t \in \mathbb{R}^N$ with $N+1$ being the total number of classes (nodes) in R. Label vector y_t is then the one-hot encoding with respect to an N-dimensional vector. In inference we employ beam search to find the most likely predicted path in R.

We improve the FPL model by applying residual learning to the RNN part of the model. Residual learning for deep networks is introduced by He et al. (2016). We connect the residual arc between input u_t (converted CNN features) and output o_t of the RNN in order to prevent the original CNN features from losing much information while the RNN is trained. Outputs of the RNN with residual learning are calculated by

$$o_t^{\text{residual}} = u_t + z\left(o_t; \xi\right) \quad \text{for } t = 1, 2, \ldots, T$$

where z is a linear mapping function to align dimensions of u_t and o_t with ξ being trainable parameters. In the loss function o_t is then replaced by o_t^{residual}.

2.3.2. General Tree Model: CNN-S2S

In this section, we propose CNN-S2S to fit a general class tree that has target paths with variable lengths from the root to the final class node. In this model, the assumption of a fixed-length of class path in the FPL model is relaxed, and the RNN part of the FPL model is replaced by an S2S. Traditional RNNs are limited to solving problems where input and target sequences are of the same length. S2S introduces an encoder to transform the input sequence to a fixed-dimension representation, and a decoder to process this fixed-length representation to a variable-length sequence (Cho et al., 2014). We propose a model combining a CNN with an S2S that can deal with target label paths of variable lengths. Let now R be a general rooted tree. Label path y is any path from the root to a node (not necessarily a leaf) in R. We denote by $|y|$ the number of nodes in y. Similar to the FPL model, the general tree model feeds fixed dimension CNN features to S2S, and predicts labels as a vector with the same length as the number of classes in y.

Given the converted CNN features u_t as defined in Section 2.3.1 and the encoder presented by (2.1), the decoder reads

$$\bar{h}_t = \bar{r}_h\,(\bar{o}_t, \bar{h}_{t-1}; \bar{\theta}_h),$$

$$\bar{o}_t = \bar{r}_o\,(\bar{h}_t; \bar{\theta}_o)$$

for $t = 1, 2, \ldots, |y|$, and $\bar{h}_0 = h_T$. The loss function defined in the FPL model is used in the general tree model, and beam search is again used in inference.

2.3.3. Conversion Operation

In our models, the conversion operation is the operation of connecting the CNN and the RNN. This operation is needed in order to feed extracted CNN features that have varying dimensions to the RNN that requires the one fixed input dimension. We design conversion operations not only to align related dimensionality, but also to retain much information of the learned CNN representations.

We first describe a linear conversion that converts CNN features directly through the n-mode product of a tensor. This conversion is a series of linear transformations to modify dimensionality of the CNN features that requires to train additional model weights. This operator was previously proposed in Zhu and Bain (2017); Wehrmann et al. (2018). The linear conversion, $g(a_s; \alpha_s)$, is defined as

$$\begin{aligned} &g(a_s; \alpha_s) = \operatorname{vec}(a_s \times_1 U_s^1 \times_2 U_s^2 \times_3 U_s^3) \\ &s \in S_t, t = 1, 2, \ldots, T \end{aligned}$$

where $U_s^1 \in \mathbb{R}^{m \times D_s}$, $U_s^2 \in \mathbb{R}^{n \times W_s}$, and $U_s^3 \in \mathbb{R}^{v \times H_s}$ are trainable parameter matrices at CNN layer $s \in S_t$ ($\alpha_s = (U_s^1, U_s^2, U_s^3)$). Here $\times_k$ is the k-mode product of a tensor by a matrix and vec is a flattening operation. In this conversion, a desired RNN or S2S input dimension, p, is determined by $m \cdot n \cdot v = p$.

We also propose conversion methods by using convolutional and pooling operations. Convolutional conversion retains spatial information of the CNN features and aligns related dimensions efficiently. However, we have additional trainable model weights similar to the linear conversion. The convolutional conversion, $g(a_s; \alpha_s)$, is defined as

$$g(a_s; \alpha_s) = \mathrm{vec}(\mathrm{Conv}(a_s; \alpha_s))\ s \in S_t, t = 1, 2, \ldots, T$$

where α_s are trainable parameters for convolution operation Conv. Note that the filter size, stride, and depth of Conv have to be selected in such a way that the resulting vector is in $\mathbb{R}^p$. The details are provided in the appendix.

The pooling conversion does not require to train additional model weights, at the same time it keeps spatial information of the original CNN features. The pooling conversion, $g(a_s; \alpha_s)$, is defined as

$$\begin{aligned} & g(a_s; \alpha_s) = \mathrm{vec}(\mathrm{Pool}(a_s) \times_1 U_s^1 \times_2 U_s^2 \times_3 U_s^3) \\ & s \in S_t, t = 1, 2, \ldots, T \end{aligned}$$

where Pool is the pooling operation. The details are provided in the appendix.

2.4. Computational Study

In this section we present two cases: one based on a proprietary dataset and the other one based on a public dataset. The models have been implemented using Tensorflow. A single GPU card has been used in every run. In the experiments, we compare our models to state-of-the-art CNNs: CNN architectures by Visual Geometry Group (VGG) (Simonyan and Zisserman, 2014), Residual neural networks (Res) (He et al., 2016), Squeeze-and-Excitation networks (SE) (Hu et al., 2018), and Convolutional Block Attention Module (CBAM) (Woo et al., 2018). For the RNN and S2S parts of our networks a bidirectional RNN with LSTM cells is applied.

2.4.1. Real World Data

We conduct experiments on a real world proprietary dataset containing approximately 180,000 images. We select validation and test sets with approximately 36,000 images each. The classes have mainly Has-A hierarchical relationships with 18 classes and the tree of depth four in the FPL tree, and 15 classes (nodes in the tree) and the tree of maximal depth four in the general tree. The general tree setting has target paths with lengths ranging from two to four.

Preprocessing of raw data and hyperparameters are determined based on Simonyan and Zisserman (2014); He et al. (2016). Original images are resized with its shorter side sampled in [256, 512] and then cropped to 224×224. Hyperparameters are set as follows: batch size is set to 32, input dimensions of the RNN converted from CNN features range from 512 to 4,096, the dimensionality of the RNN hidden states range from 512 to 1,024, the FPL model has three RNN layers, and the general tree model has one S2S layer. The CNN part is initialized with the weights trained on ImageNet. Orthogonal random initialization is adapted for the RNN and S2S weights. Due to low memory requirements, we use the pooling conversion for the real dataset since it it the only option on this dataset. The conversion operations are composed later on the public dataset where it is established that pooling is best.

We apply an alternating training scheme for the CNN and RNN parts; i.e. we update one while keeping the other frozen and flip in the beginning of training, and then in a later phase unfreeze the entire network. Alternating prevents divergence during training of the CNN and the RNN part as each has a distinct purpose; the CNN learns hierarchical features of images, and the RNN learns hierarchical trees of categories. In addition, the

Table 2.1. Test accuracies of the FPL model on real data

Method	Accuracy (%)	
	Path	Node
VGG-16	64.0	71.3
VGG-RNN	59.9	77.3
VGG-RNN-Alt	61.7	78.1
VGG-RNN-Alt-Resi	**65.3**	**80.5**
Res-50	62.1	68.8
Res-RNN	63.4	78.9
Res-RNN-Alt	62.4	77.0
Res-RNN-Alt-Resi	**64.0**	**78.4**

quality of weight initializations is uneven between the CNN and the RNN as the CNN starts with high-quality pretrained weights from a large-scale dataset, ImageNet, while the RNN starts with random weights. For this reason, in our training we unfreeze the RNN first in the alternating scheme. These settings are applied to VGG-16 and Res-50.

To evaluate performance of our models, two metrics are compared. For path accuracy we count a prediction correct if the entire predicted path matches all of the labels in the ground truth while for node accuracy we count how many nodes in the target path are correct in the predicted path. Node accuracy captures how accurately predictions fit the ground truth at different levels. Remark 1: Because as simple CNN classifiers, VGG-16 and Res-50, can predict only the final node of a path and for compatibility of the metric we imply that if the final node is correctly predicted, all its predecessors are correctly predicted as well.

In Tables 2.1 and 2.2 accuracies on the test set are presented. CNN models are denoted by Res-50 and VGG-16. Our FPL and general tree models are denoted by (CNN structure)-(RNN structure)-(alternating training scheme)-(residual learning). Training takes around six days for 30 epochs.

Table 2.2. Test accuracies of the general tree model on real data

Method	Accuracy (%)	
	Path	Node
VGG-16	72.8	87.2
VGG-S2S	**73.7**	**88.1**
VGG-S2S-Alt	72.3	87.2
Res-50	71.5	86.4
Res-S2S	**74.1**	**88.0**
Res-S2S-Alt	68.3	85.0

FPL model: Table 2.1 presents the experimental results of the FPL model. Residual variants with alternating training perform best on both Res and VGG. This shows that both CNN and RNN of our model successfully play their specific roles; the CNN learns hierarchical features of images, and the RNN correctly predicts target paths. Furthermore, residual arcs and alternating training help improving test accuracy. Our models with Res perform better than Res-50 on both path and node accuracies. However, for VGG our non-residual variants VGG-RNN and VGG-RNN-Alt perform worse on path accuracy than VGG-16 even though our models show higher node accuracy. This can be interpreted as our models solving more difficult problems than CNN regarding path accuracy (see Remark 1). Explicitly predicting all nodes along a path in a tree is more difficult than the path correctness implicitly assumed as soon as only the final node is correctly predicted.

General tree model: Table 2.2 shows experimental results of the general tree model. Our model without alternating training performs better on both path and node accuracies than CNNs. This proves that our models successfully extract hierarchical features of images and learns a label path with variable lengths of target paths. However, alternating training in the experiments did not help to improve performance of our models. This can be interpreted as S2S with orthogonal initialization being good enough to avoid diverging in training.

2.4.2. Public Data: Open Images

Open Images V4 is a public dataset of 9 million images annotated with image-level labels, object bounding boxes and visual relationships (Krasin et al., 2017). We use a subset of the original dataset for hierarchical classification. The subset contains approximately 950,000 images with 2.4 million labels for training and 36,000 images with 127,000 labels for test; it is a multi-label dataset. Since the labels reside in different levels of the original class hierarchy, we build a class tree of depth four by concatenating subtrees of the original hierarchy. There are 30 classes in the tree with the classes having mainly Has-A relationships. Figure 2.3 presents a subtree of the tree with the full tree presented in the appendix. We follow the same preprocessing steps of raw images and model architectures as those used in Section 2.4.1.

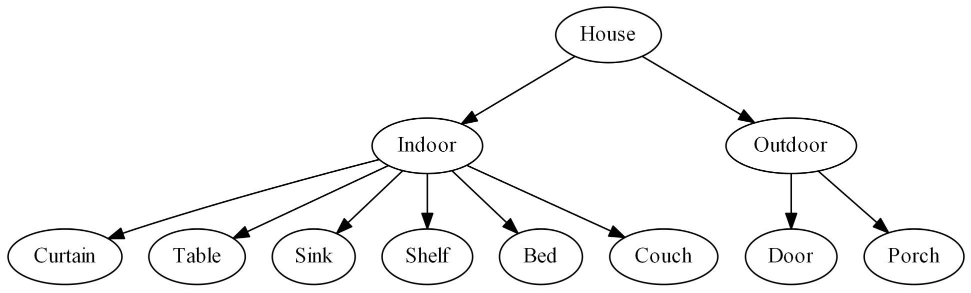


Figure 2.3. A part of the tree of Open Images

To evaluate multi-label classification models, scores computed by precision and recall are typically considered such as the F1 score and area under precision-recall curve (Vens et al., 2008; Bi and Kwok, 2011; Zhang and Zhou, 2014). In this study, we select the F1 score as our evaluation metric that computes the harmonic average of the precision and recall.

To solve the multi-label classification problem which in our case corresponds to multiple paths in the tree for a single sample, we select the sigmoid function at the output layer of CNNs and our models rather than the softmax function. The predictions are selected as those with the logit value above a threshold. A path is selected if all logits of the nodes in the path are above the threshold. The threshold is selected so as to maximize the F1 score on the validation dataset. It is then used on the test dataset. In the same way as in the real data experiments, we calculate the path and node F1 scores. To provide more reliable results, we compute the means of the test F1 scores and their standard deviations for each model averaged over 3 random runs. Each training takes around four days for 15 epochs.

Table 2.3 presents the test F1 scores of the general tree model (FPL does not apply here since the paths have different lengths). VGG-16, Res-50, CBAM-Res-50, and SE-Res-50 are CNN models. Our general tree models are denoted by (CNN structure)-(RNN structure)-(alternating training scheme)-(conversion methods). For our models with Linear, Conv, and Pool, we use 256 and 512 for input and hidden state dimensions of S2S, respectively. Under this setting all three conversion operations can be executed without a memory problem. For our models with Pool, we also use larger dimensions of 2048 and 1024 for input and hidden state of S2S, respectively, since the pooling conversion has lower memory requirements than others. We stress in bold the best three models in the table. We find that SE-Res-50 performs best among all CNN models. All of our models based on different CNN architectures such as VGG, Res, SE, and CBAM perform better on path and node F1 scores than standard CNN models. The largest improvement over CNN models is made

Table 2.3. Test F1 scores of general tree on Open Images

Method	Mean F1 score and Sd. (%)	
	Path	Node
VGG-16	68.62 (0.09)	72.29 (0.09)
VGG-S2S-Linear	**71.49 (0.10)**	**74.84 (0.12)**
VGG-S2S-Alt-Linear	70.76 (0.06)	73.97 (0.15)
VGG-S2S-Conv	68.61 (0.41)	71.62 (0.51)
VGG-S2S-Alt-Conv	70.70 (0.14)	73.95 (0.06)
VGG-S2S-Pool	71.39 (0.16)	74.76 (0.18)
VGG-S2S-Alt-Pool	**71.49 (0.10)**	**74.84 (0.07)**
VGG-S2S-Pool-$\mathcal{L}$	**71.93 (0.09)**	**75.29 (0.07)**
VGG-S2S-Alt-Pool-$\mathcal{L}$	71.26 (0.13)	74.58 (0.17)
Res-50	71.15 (0.04)	74.47 (0.07)
Res-S2S-Linear*	**72.14 (0.14)**	**75.46 (0.16)**
Res-S2S-Alt-Linear	70.78 (0.09)	73.80 (0.14)
Res-S2S-Conv	70.42 (0.39)	73.55 (0.44)
Res-S2S-Alt-Conv	70.75 (0.15)	73.98 (0.18)
Res-S2S-Pool	**71.98 (0.06)**	**75.40 (0.07)**
Res-S2S-Alt-Pool	71.70 (0.06)	74.97 (0.09)
Res-S2S-Pool-$\mathcal{L}^*$	**72.05 (0.06)**	**75.43 (0.03)**
Res-S2S-Alt-Pool-$\mathcal{L}$	71.66 (0.04)	74.89 (0.05)
SE-Res-50	71.22 (0.10)	74.49 (0.08)
SE-Res-S2S-Linear*	**72.05 (0.04)**	**75.33 (0.08)**
SE-Res-S2S-Alt-Linear	69.52 (0.11)	72.45 (0.14)
SE-Res-S2S-Conv	70.49 (0.15)	73.65 (0.14)
SE-Res-S2S-Alt-Conv	70.35 (0.09)	73.51 (0.06)
SE-Res-S2S-Pool	**71.79 (0.15)**	**75.12 (0.16)**
SE-Res-S2S-Alt-Pool	71.42 (0.05)	74.73 (0.04)
SE-Res-S2S-Pool-$\mathcal{L}$	**71.75 (0.09)**	**75.07 (0.04)**
SE-Res-S2S-Alt-Pool-$\mathcal{L}$	71.07 (0.13)	74.27 (0.18)
CBAM-Res-50	71.17 (0.09)	74.42 (0.04)
CBAM-Res-S2S-Linear	**71.71 (0.19)**	**74.98 (0.19)**
CBAM-Res-S2S-Alt-Linear	66.44 (0.43)	68.84 (0.53)
CBAM-Res-S2S-Conv	70.24 (0.07)	73.43 (0.05)
CBAM-Res-S2S-Alt-Conv	72.88 (0.09)	69.82 (0.07)
CBAM-Res-S2S-Pool	**71.53 (0.12)**	**74.82 (0.10)**
CBAM-Res-S2S-Alt-Pool	71.29 (0.18)	74.55 (0.11)
CBAM-Res-S2S-Pool-$\mathcal{L}$	**71.57 (0.07)**	**74.89 (0.07)**
CBAM-Res-S2S-Alt-Pool-$\mathcal{L}$	71.32 (0.05)	74.55 (0.01)

$\mathcal{L}$ denotes models with larger S2S than others.

$*$ denotes top three performers in the table.

by our model VGG-S2S-Pool-$\mathcal{L}$, and the highest F1 score is achieved by Res-S2S-Linear. This shows that our models fit the multi-label class tree better by extracting CNN features from images and predicting paths with variable lengths by S2S. Alternating training does not work for S2S in many cases even though our model with alternating is one of the best three performers in VGG. As pointed out in real data experiments, the effect of alternating training can be overshadowed by other factors such as weight initialization. In almost all cases, pooling conversion performs better than other conversion methods. This is because pooling at conversion keeps the spatial information of the trained CNN features. However, convolutional conversion works only in VGG when it is combined with alternating training. As this requires additional trainable weights, it makes our models harder to train and consumes additional memory. We find that its usage may be limited to smaller networks. We also find that models with larger S2S vector sizes perform better than smaller models when alternating training is not used. Our models feed CNN features to S2S, and so the input to S2S is a representation of images. Since larger S2S models can extract more information from images from trained CNN features than smaller models, the larger models are expected to achieve better results. This perspective is not true for alternating training even though larger models without alternating perform the best among models with pooling conversion.

2.5. Conclusion

In this work, we develop a new structure of deep neural networks for hierarchical classification of images. Combining CNN as a feature learner with RNN or S2S as a sequence classifier, the proposed models can predict a target path in a hierarchical tree of classes. By means of capturing hierarchical representations, the proposed models take features from different CNN layers, and feed them to RNN or S2S. Depending on the class tree structure two models are suggested; the FPL model (CNN-RNN) and the general tree model (CNN-S2S) for a fixed- and variable-length target paths. To expedite training and improve generalization of the model, we also suggest a CNN-RNN variation that adds residual arcs to the RNN part. To examine the performance of our models, we conduct experiments on a proprietary and a public dataset of images. Experimental results show that our models perform better than state-of-the-art CNNs, VGG, Res, SE, and CBAM. For CNN-RNN, our models with residual arcs perform best in predicting fixed length paths for both CNN networks. For CNN-S2S, our models without alternating training perform the best in almost all cases. For this reason we recommend not to use alternating training in the S2S case. Considering GPU memory is limited in practice, we recommend to use pooling conversion as it performs well and has low memory requirements for training.

CHAPTER 3

A New Framework for Inverse Classification Using Mixed Integer Programming

3.1. Introduction

Classification is a building block for solving various machine learning tasks such as customer segmentation, sentimental analysis, and image recognition. Numerous state-of-the-art classification models such as deep neural networks are developed to achieve high classification accuracy (Aggarwal et al., 2010; Lash et al., 2017a). In this chapter, we study an interesting variant of a classification problem, called inverse classification that studies interpretability of classification models rather than improving classification accuracy. Given a trained classifier, inverse classification models identify minimal changes of input features of an instance so that the instance is predicted as a desired class that is different from its original label (Laugel et al., 2018). It is first introduced as a topic of sensitivity analysis (Mannino and Koushik, 2000) and then augmented as an interpretability approach (Barbella et al., 2009). Viewing inverse classification as a utility-based data mining problem Lash et al. (2017a) argue that it is a subtopic of strategic learning (Boylu et al., 2010). Inverse classification is also related to a counterfactual explanation in interpretable machine learning. A counterfactual explanation reveals how an instance should be perturbed to change its original prediction significantly. By crafting counterfactual instances we can interpret how a classifier computes individual predictions (Wachter et al., 2018; Molnar, 2019). Laugel et al. (2018) and Lowd and Meck Lowd and Meek (2005) point out that inverse classification is related to adversarial learning (Tygar, 2011) that aims to attack a classifier by applying small perturbations to samples to modify their initial predictions. Inverse classification and counterfactual explanation study focus on interpretability of classification models. Meanwhile, adversarial learning mainly focuses on robustness of associated models. For example, developing a defensive system against adversary attacks is a study of interest in adversarial learning. Perturbing samples so that they are predicted as a desired label is a goal to be achieved in all of these areas.

We present a typical setting considered in inverse classification, counterfactual explanations, and adversarial learning. Assume that we have a classifier $f(x) : x \in X \rightarrow y \in Y$ with input x and output y. Our goal is to generate an adversarial sample $\hat{x}$ that is the same form as a given sample x, and the adversarial sample is to be predicted as a desired class that is different from the original label. Especially in adversarial learning, there are two types of adversarial examples. A non-targeted adversarial example $\hat{x}$ is generated by adding small perturbation to x so that $\hat{x}$ is classified as any class that is not the original ground truth. A targeted adversarial sample fools a classifier so that it produces a desired label as $f(\hat{x}) = \bar{y}$ where $\bar{y}$ is the desired class determined by an adversary. A L_p norm of perturbation between adversarial samples and given samples is usually used as a loss function where p can be $0, 1, 2, \infty$ (Dong et al., 2018). In some cases, a set of budget constraints for the perturbation is introduced, and subsequently, not all of candidate instances can be successfully perturbed as desired. Here, we should optimally spend a budget on instances so that as many instances as possible can be successfully perturbed within the budget. Existing inverse classification and adversarial attack frameworks that minimize cost of the perturbation do not consider this perspective effectively.

In this chapter, we develop a new framework that can be applied to inverse classification and counterfactual explanations as well as adversarial learning. We assume to have a budget constraint on perturbation in input features of samples, and the input features are continuous. In order to obtain the maximum number of successfully perturbed samples within a budget, we define an objective function that maximizes the number of instances to be perturbed, which is different from the existing formulation. For this, a binary variable to decide which sample to be perturbed is introduced. In addition, we include a set of constraints that probability of a desired class by a classifier is higher than the remaining classes by a margin, which guarantees to predict the desired class as well as some confidence in adversarial attacks. Therefore, the proposed model is a constrained mixed integer problem. We extend our deterministic mixed integer programming to stochastic programming by assuming some of decision variables follow Bernoulli or categorical distributions. Consequently, we have chance constraints for budget and prediction confidence.

In this study, we use a real world proprietary dataset from the insurance industry and a public dataset on health clinic, MIMIC (Goldberger et al., 2000; Johnson et al., 2016). We design our solving algorithms based on a gradient method that updates solutions in a better direction. We first reformulate our constrained problem as unconstrained using Lagrangian method. Afterward, we design algorithms based on the projected subgradient method. Subgradients are used to update variables iteratively in non-differentiable functions of our model, and projection is used to deal with bounds on variables such as Lagrangian multipliers. For our chance constraint model, we apply the Gumbel trick to calculate associated gradients, while avoiding computing them over expectations. We compare the performance of our algorithms by evaluation metrics such as the number of feasible samples and the budget spent per sample.

The contributions of this work are as follows.

(1) We introduce a new framework to solve inverse classification using mixed integer programming. Our framework is designed to achieve the maximal number of successfully perturbed samples within a budget. As far as we know, this framework has not been applied to inverse classification in the existing literature.
(2) We design customized algorithms based on gradient methods to solve our problems.
(3) In the computational study, our approach performs well on different budget scenarios, and we verify the scalability of our algorithms.

The rest of this chapter is organized as follows. In Section 3.2, the related work is discussed. Section 3.3 describes the proposed models, and algorithms to solve them are presented in 3.5. Section 3.5 provides a computational study including experimental details and analysis of the experimental results. Conclusions are given in Section 3.6.

3.2. Related Work

In inverse classification and counterfactual explanation study, literature is categorized based on the following perspectives; frameworks of formulation such as unconstrained or constrained problems, and algorithmic mechanisms (Lash et al., 2017a,b). A formulation framework is related to feasibility and implementablity of perturbed samples, which categorizes literature as either an unconstrained (Aggarwal et al., 2010; Yang et al., 2012) or a constrained problem (Mannino and Koushik, 2000; Barbella et al., 2009; Chi et al., 2012; Lash et al., 2017a,b; Wachter et al., 2018). Since an unconstrained formulation does not consider practical constraints such as a budget, it tends to produce unrealistic perturbation of input features of a sample as for example 'change your purchase history.' A constrained formulation provides realistic perturbation, however, we should design a sophisticated algorithm that deals with constraints. There are three factors to be considered in the framework; a) changeable or unchangeable features to be perturbed, e.g. an unchangeable feature could be product purchase history, b) how difficult to change features such as a feature-specific cost, and c) a limit to the amount of perturbations over all instances, that is, a budget (Lash et al., 2017a,b). In Barbella et al. (2009), their recommended perturbation is moderate; however, they do not consider a) which features are changeable, b) feature specific costs, and c) budget constraints. Mannino and Koushik (2000) consider b), but do not consider a) and c). Lash et al. (2017a) propose a general framework that considers a), b), and c); however, a prediction confidence constraint is not included. Based on algorithms to solve an inverse classification problem, literature is categorized into two groups; greedy (Mannino and Koushik, 2000; Aggarwal et al., 2010; Chi et al., 2012; Yang et al., 2012; Lash et al., 2017b) and non-greedy (Barbella et al., 2009; Lash et al., 2017a). Greedy methods produce adversarial samples relatively fast but tend to be unrealistic in the real world since they do not necessarily achieve optimal solutions. Non-greedy methods tend to focus on more moderate objectives so that obtained adversarial samples are more realistic. In Mannino and Koushik (2000); Aggarwal et al. (2010); Chi et al. (2012); Yang et al. (2012); Lash et al. (2017b), heuristic methods that do not use gradients such as local search, hill climbing, and genetic algorithm are used.

In Lash et al. (2017a), the projected gradient method is adopted as a solving algorithm. In Barbella et al. (2009), a non-linear solver package, CONOPT3 [1], is used to solve a constrained problem. Our work is different from the aforementioned papers since none of the existing methods consider maximizing the number of perturbed samples in their formulation. In addition, we verify our approach on state-of-the-art models, deep neural networks, as our classifiers. Our framework considers budget and predictive confidence constraints.

In adversarial learning, recent research mostly focuses on generating adversarial samples to attack deep learning models since its purpose is to study robustness of the state-of-the-art classifiers. Most adversarial attacks are targeted against deep nets as they are currently the winning models in many classification tasks. Several attack algorithms have been developed in order to generate adversarial images. Szegedy et al. (2013) propose the following optimization problem and solve it using boxing constrained L-BFGS,

$$\begin{aligned} \min_{\hat{x}} \quad & c||x - \hat{x}||_2^2 + J(\hat{x}) \\ \text{s.t.} \quad & \hat{x} \in [0,1] \end{aligned}$$

where J is a loss function of classification toward a desired label such as cross-entropy and Kullback–Leibler divergence. Goodfellow et al. (2015) propose a method called Fast Gradient Sign method (FGSM) using sign of gradients, and L_∞ is used as a distance metric for perturbation. It is not guaranteed to produce optimal solutions, but quick to obtain close adversarial examples. Given an input image x FGSM reads

$$\hat{x} = x - \epsilon \, \text{sign}(\nabla J(x))$$

[1] http://www.conopt.com/

where ϵ is selected to be small. Kurakin et al. (2017) propose an improved FGSM, iterative gradient sign method (I-FGSM), by taking multiple smaller steps α toward gradient sign rather than one step of size ϵ, and its output is clipped by the same size ϵ. It produces superior results to FGSM by updating $\hat{x}$ on each iteration t as

$$\hat{x}_t = \hat{x}_{t-1} - \text{clip}_{\epsilon}(\alpha \, \text{sign}(\nabla J(\hat{x}_{t-1}))$$

where $\hat{x}_0 = 0$. Recently, Papernot et al. (2016) propose a greedy algorithm to generate adversarial examples using gradients to compute a Saliency map, called Jacobian-based Saliency Map Attack (JSMA). However, these threat model algorithms do not consider a budget constraint that is critical and practical in inverse classification even though they have a bound on a pixel domain. In addition, our framework is designed to achieve the maximal number of successfully perturbed instances within a budget.

3.3. Proposed Models

In this section, we present our constrained optimization problem to solve inverse classification. Our formulation is designed to generate the maximal number of adversarial examples that are classified as a desired class. In addition, we include a set of budget constraints on perturbation of input features to make our formulation practical. We first introduce notation and our baseline model. Afterward, we present a variation of the baseline model, a chance constraint model by assuming decision variables follow a probability distribution.

We denote a given input, $\mathbf{x} \in \mathbb{R}^p$ and a perturbed input, $\hat{\mathbf{x}} \in \mathbb{R}^p$. We assume that all features are continuous. Let $f(\mathbf{x}) : \mathbb{R}^p \mapsto [0, 1]^k$ be a function associated with a classification model, and it computes the score of $\mathbf{x}$ being in a class such as probability of k classes. We optimize over samples in $\hat{\mathbf{x}}_j$ given a new desired label vector $\bar{\mathbf{y}}_j \in [0,1]^k$ with $j = 1, \ldots, |\mathcal{S}|$ where $\mathcal{S}$ is a set of instances, $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{|\mathcal{S}|}\}$. We denote a perturbed input feature matrix $\hat{\mathbf{X}}$. Furthermore, we introduce binary variables, $\mathbf{z} \in \{0,1\}^{|\mathcal{S}|}$, to decide which instance to be perturbed. In our formulation, we maximize the binary variables to obtain successfully perturbed instances as many as possible. We introduce a budget constraint on perturbation of input features, which is computed by a distance between given and perturbed input features. We assume that a budget is assigned to each feature, and the distance is calculated by the Euclidean norm. In addition, we introduce a constraint, called prediction confidence constraint to signify a margin for prediction reflecting the uncertainty in the score function. Formally, our max samples model (MS) is formulated as

$$\begin{aligned} \max_{\mathbf{z}, \hat{\mathbf{X}}} \quad & \sum_{j=1}^{|\mathcal{S}|} z_j \\ \text{s.t.} \quad & g_i(\mathbf{z}, \hat{\mathbf{x}}) \leq 0, i = 1, \ldots, p, \\ & h_j(\mathbf{z}, \hat{\mathbf{x}}) \leq 0, j = 1, \ldots, |\mathcal{S}| \end{aligned} \tag{3.1}$$

where g_i and h_j is a nonlinear function associated with budget and prediction confidence constraints, respectively. For budget constraints on perturbation of input features, they are explicitly written as

$$g_i(\mathbf{z}, \hat{\mathbf{x}}) = \sum_{j=1}^{|S|} z_j \, ||\hat{x}_{ij} - x_{ij}||_2 - B_i, i = 1, \ldots, p \tag{3.2}$$

where $B_i \in \mathbb{R}_+$ is a given budget for feature i. Prediction confidence constraints are explicitly expressed as

$$f(\hat{\mathbf{x}}_j)_u + \delta \leq f(\hat{\mathbf{x}}_j)_{y_j} \; j = 1, \ldots, |\mathcal{S}|, u = 1, \ldots, k, u \neq y_j \tag{3.3}$$

where $y_j = \underset{u}{\operatorname{argmax}} \, [\bar{\mathbf{y}}_j]_u$ is a desired class of $\mathbf{x}_j$ and $\delta > 0$ is a given margin. In MS, we rewrite (3.3) by multiplying binary variables so that we consider the constraints only on selected samples as follows.

$$h_j(\mathbf{z}, \hat{\mathbf{x}}) = z_j \max\{0, \max_{\substack{u=1,\ldots,k \\ u \neq y_j}} f(\hat{\mathbf{x}}_j)_u - f(\hat{\mathbf{x}}_j)_{y_j} + \delta\}, j = 1, \ldots, |\mathcal{S}|. \tag{3.4}$$

We further improve MS by assuming that the binary variables have a probability distribution. We assume that our variables follow Bernouli or Categorical distributions considering dependency among instances. First, we present a Bernouli case where there is no relationship among perturbed instances. Let $z_j \sim \text{Ber}(\pi_j), j = 1, \ldots, |\mathcal{S}|$, that is,

$$z_j = \begin{cases} 1 & \text{with probability } \pi_j \\ 0 & \text{with probability } 1 - \pi_j. \end{cases} \tag{3.5}$$

Transforming MS (3.1), we propose a Bernoulli chance max samples model (BCMS),

$$\begin{aligned} \max_{\Pi, \hat{\mathbf{x}}} \quad & \mathbb{E}_{\mathbf{z}} \left[\sum_{j=1}^{|\mathcal{S}|} z_j \right] \\ \text{s.t.} \quad & \Pr\big(g_i(\mathbf{z}, \hat{\mathbf{x}}) \leq 0\big) \geq 1 - \epsilon, i = 1, \ldots, p, \\ & \Pr\big(h_j(\mathbf{z}, \hat{\mathbf{x}}) \leq 0\big) \geq 1 - \epsilon, j = 1, \ldots, |\mathcal{S}| \end{aligned} \tag{3.6}$$

where g_i and h_j is a nonlinear function associated with budget and prediction confidence constraints, respectively, which is defined as the same as in MS. We can explicitly rewrite BCMS (3.6) as the same as in expectation,

$$\begin{aligned} &\max_{\mathit{\Pi},\hat{\mathbf{x}}} \quad \sum_{j=1}^{|\mathcal{S}|} \pi_j \\ &\text{s.t.} \quad \Pr\big(\sum_{j=1}^{|S|} z_j \,||\hat{x}_{ij} - x_{ij}||_2 - B_i \le 0\big) \ge 1-\epsilon, i = 1,\dots,p \\ &\qquad\ \ \pi_j\, h(\hat{\mathbf{x}}_j) \le 0, j = 1,\dots,|\mathcal{S}|. \end{aligned} \tag{3.7}$$

Our chance max samples model with Categorical distribution (CCMS) considers dependency among instances to be perturbed. CCMS has the same formulation as defined in (3.6), but we use the following binary variables to determine which instance to perturb,

$$\bar{z}_\xi \in \mathbb{R}^{|\mathcal{S}|} \sim \mathrm{Cat}(\mathit{\Pi}), \mathit{\Pi} = (\pi_1,\dots,\pi_{|\mathcal{S}|}), \text{ and } \mathbf{z} \in \mathbb{R}^{|\mathcal{S}|} = \min(\mathbf{1}, \sum_{\xi=1}^{|\mathcal{S}|} \bar{z}_\xi). \tag{3.8}$$

We present a model based on the existing framework (Szegedy et al., 2013; Molnar, 2019) of generating adversarial samples that is designed to solve inverse classification with minimal cost of perturbation. This model optimizes over samples in $\hat{\mathbf{x}}_j \in D$ to minimize a loss function $l_j = KL\,(\bar{\mathbf{y}}_j||f(\hat{\mathbf{x}}_j)) + a\,||\hat{\mathbf{x}}_j - \mathbf{x}_j||_2$, $a \in [0,\infty)$ where KL denotes

Kullback-Leibler divergence. The model (KL) reads

$$\begin{aligned}
&\min_{\hat{\mathbf{x}}} \quad \sum_{j=1}^{|\mathcal{S}|} l_j\left(\hat{\mathbf{x}}\right) \\
&\text{s.t.} \quad g_i(\hat{\mathbf{x}}) \leq 0, i = 1, \ldots, p, \\
&\qquad\; h_{ju}(\hat{\mathbf{x}}) \leq 0, j = 1, \ldots, |\mathcal{S}|, u = 1, \ldots, k, u \neq y_j
\end{aligned} \tag{3.9}$$

where g and h is a nonlinear function associated with budget (3.2) and prediction confidence constraints (3.3) without a binary variable $\mathbf{z}$. This model has a smaller number of variables to optimize than our models; however, it does not consider which instance to be perturbed. Therefore, it is not necessary to achieve the maximal number of the successfully perturbed instances since budgets are not optimally spent.

3.4. Algorithms

In this section, we describe the proposed algorithms to solve our models. We design our algorithms based on Lagrangian and subgradient methods. We first reformulate our constrained problem to an unconstrained problem by multiplying Lagrangian multipliers to constraints, and adding them to our loss function such that we derive Lagrangian function for each model. Afterward, we develop algorithms to solve the Lagrangian functions based on the projected subgradient method. We use the subgradient method since our models contain non-differentiable functions, and projection is used to keep Lagrangian multipliers positive during their update. For chance max samples models, we apply the Gumbel trick (Maddison et al., 2014) to use approximated gradients in updating binary variables rather than to compute exact gradients of the variables in the expectation in order to alleviate enormous computing cost. Since our algorithms do not necessarily find optimal solutions, we discuss how to evaluate the performance of our algorithms on perturbed instances.

Algorithm for Max Samples Model. We first define Lagrangian function for our baseline model MS (3.1) as

$$\begin{aligned} L(\mathbf{z}, \hat{\mathbf{X}}, \mathit{\Lambda}, M) &= \sum_{j=1}^{|\mathcal{S}|} z_j - \sum_{i=1}^{p} \lambda_i g_i - \sum_{j=1}^{|\mathcal{S}|} \mu_j h_j \\ &= \sum_{j=1}^{|\mathcal{S}|} z_j - \sum_{i=1}^{p} \lambda_i \big(\sum_{j=1}^{|S|} z_j \, ||\hat{x}_{ij} - x_{ij}||_2 - B_i \big) - \sum_{j=1}^{|\mathcal{S}|} \mu_j z_j h_j(\hat{\mathbf{x}}_j) \\ &= \sum_{j=1}^{|\mathcal{S}|} c_j z_j + \sum_{i=1}^{p} \lambda_i B_i \end{aligned}$$

where $c_j = 1 - \sum_{i=1}^{p} \lambda_i \, ||\hat{x}_{ij} - x_{ij}||_2 - \mu_j h_j(\hat{\mathbf{x}}_j)$, and λ and μ are Lagrangian multipliers. We propose Algorithm 1 to solve $\min_{\mathit{\Lambda},M} \max_{\mathbf{z},\hat{\mathbf{X}}} L(\mathbf{z}, \hat{\mathbf{X}}, \mathit{\Lambda}, M)$. The algorithm consists of two main loops to solve our min max problem; the inner loop updates input features and binary variables to maximize L, and the outer loop updates Lagrangian multipliers to minimize L. In the algorithm, we initialize all Z with one as we desire to achieve as many successfully perturbed samples as possible. Meanwhile, we add a line to break the inner loop when all Z are zero, which is a case of no updates on variables. Note that $\nabla_{\lambda_i} L = -\sum_{j=1}^{|S|} z_j \, ||\hat{x}_{ij} - x_{ij}||_2 + B_i$, and $\nabla_{\mu_j} L = -z_j h_j(\hat{\mathbf{x}}_j)$. In addition, line 7-13 in Algorithm 1 is derived by solving

$$\max_{\mathbf{z}} \quad \sum_{j=1}^{|\mathcal{S}|} c_j z_j$$

where c_j is a constant in the algorithm.

Algorithm 1 MS

```
1: Initialize $\mathit{\Lambda}, M, \alpha, \beta$.
2: while until convergence do
3:     $z_j \leftarrow 1, j = 1, \ldots, |\mathcal{S}|$
4:     while until convergence do
5:         Break if $\sum_j z_j = 0$
6:         $\hat{\mathbf{X}}^* \leftarrow \operatorname{argmax}_{\hat{\mathbf{X}}} L(\mathbf{z}, \hat{\mathbf{X}}, \mathit{\Lambda}, M)$
7:         for j=1,...,$|\mathcal{S}|$ do
8:             if $c_j \geq 0$ then
9:                 $z_j \leftarrow 1$
10:            else
11:                $z_j \leftarrow 0$
12:            end if
13:        end for
14:    end while
15:    $\lambda_i \leftarrow \left(\lambda_i - \alpha \nabla_{\lambda_i} L\right)_+, i = 1, \ldots, p$
16:    $\mu_j \leftarrow \left(\mu_j - \beta \nabla_{\mu_j} L\right)_+, j = 1, \ldots, |\mathcal{S}|$
17: end while
```

Algorithm for Bernoulli and Categorical Chance Max Samples Model. We define Lagrangian function for BCMS (3.7) as

(3.10)

$$L(\mathit{\Pi}, \hat{\mathbf{X}}, \mathit{\Lambda}, M) = \sum_{j=1}^{|\mathcal{S}|} \pi_j \big(1 - \mu_j h_j(\hat{\mathbf{x}}_j)\big) + \sum_{i=1}^{p} \lambda_i \Big[\Pr\big(\sum_{j=1}^{|S|} z_j \, ||\hat{x}_{ij} - x_{ij}||_2 - B_i \leq 0\big) - (1 - \epsilon)\Big].$$

We solve $\min_{\mathit{\Lambda}, M} \max_{\mathit{\Pi}, \hat{\mathbf{X}}} L$ in which we have to compute gradients of $\mathbb{E}_{\mathbf{z} \sim \mathrm{Ber}(\mathit{\Pi})}$ with respect to $\mathit{\Pi}$. To relieve burden of computing exact gradients, we apply the Gumbel trick (Maddison et al., 2014) to use their approximation. Let the exact probability be $\Pr_i = \mathbb{E}_{\mathbf{z}} \mathcal{X}[\sum_{j=1}^{|S|} z_j \, ||\hat{x}_{ij} - x_{ij}||_2 - B_i \leq 0]$. We first approximate $\mathcal{X} \approx \frac{1}{1 + \exp(-\mathbf{k} \frac{\mathbf{x} - \tau}{1 - \tau})}$ where $\mathbf{k}$ and τ are hyperparameters. Then, we have $\mathbb{E}_{z_1, \ldots, z_{|S|}} \big[\frac{1}{1 + \exp(-\mathbf{k} \frac{\mathbf{x} - \tau}{1 - \tau})}\big]$ where $\mathbf{x} = \sum_{j=1}^{|S|} z_j \, ||\hat{x}_{ij} - x_{ij}||_2 - B_i$. Applying the

Gumbel trick, the expectation term is approximately computed as

$$\mathrm{Pr}_i \approx \frac{1}{N}\sum_{n=1}^{N} \mathrm{P}_n^i = \frac{1}{N}\sum_{n=1}^{N} \frac{1}{1+\exp(-\mathbf{k}\frac{\mathbf{x}_n^i-\tau}{1-\tau})} \tag{3.11}$$

where $\mathbf{x}_n^i = \sum_{j=1}^{|S|} v_{nj}||\hat{x}_{ij}-x_{ij}||_2-B_i$, $v_{nj}{=}z_j(\pi_j, g_1^n, g_2^n) = \frac{\exp((\log\pi_j+g_1^n)/\omega)}{\exp((\log\pi_j+g_1^n)/\omega)+\exp((\log(1-\pi_j)+g_2^n)/\omega)}$, and $g_1^n \sim \mathcal{G}, g_2^n \sim \mathcal{G}$; the Gumbel distribution is denoted by $\mathcal{G}$. We can rewrite Eq. (3.10) as

$$L \approx \sum_{j=1}^{|\mathcal{S}|} \pi_j\big(1-\mu_j h_j(\hat{\mathbf{x}}_j)\big) + \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{p} \lambda_i \mathrm{P}_n^i - (1-\epsilon)\sum_{i=1}^{p}\lambda_i. \tag{3.12}$$

We propose Algorithm 2 to solve $\min\limits_{\mathit{\Lambda},M}\max\limits_{\mathit{\Pi},\hat{\mathbf{X}}} L$. This algorithm has two main loops to maximize L with respect to $\mathit{\Pi}$ and $\hat{\mathbf{X}}$, and to minimize L with respect to Lagrangian multipliers. Especially, a part of generating Gumbel's samples is added to the inner loop so that approximated gradients are used to update variables afterwards. Note that $\ddot{L} = \sum_{j=1}^{|\mathcal{S}|} \pi_j\big(1-\mu_j h_j(\hat{\mathbf{x}}_j)\big)$ in the algorithm.

For CCMS, we define Lagrangian function based on (3.6), which is written as

$$\begin{aligned} L(\mathit{\Pi}, \hat{\mathbf{X}}, \mathit{\Lambda}, M) = \mathbb{E}_{\mathbf{z}}\Bigg[&\sum_{j=1}^{|\mathcal{S}|} z_j\big(1-\mu_j h_j(\hat{\mathbf{x}}_j)\big) \\ &+ \sum_{i=1}^{p} \lambda_i\Big\{\mathrm{Pr}\big(\sum_{j=1}^{|S|} z_j\ ||\hat{x}_{ij}-x_{ij}||_2 - B_i \le 0\big) - (1-\epsilon)\Big\}\Bigg] \end{aligned}$$

where z_j is j^{th} element of $\mathbf{z}$ as defined in (3.8). Based on Gumbel's approach we approximate $\mathbf{z}$ by

$$[\bar{z}_\xi]_j = \frac{\exp((\log\pi_j + g_{j\xi})/\omega)}{\sum_{k=1}^{|\mathcal{S}|}\exp((\log\pi_k + g_{k\xi})/\omega)} \text{ where } g_{1\xi}, \ldots, g_{|\mathcal{S}|\xi} \sim \mathcal{G}, \xi = 1, \ldots, |\mathcal{S}|.$$

Algorithm 2 BCMS

```
1: Initialize $\mathit{\Pi}, \mathit{\Lambda}, M, \alpha, \beta, \gamma, \eta$.
2: while until convergence do
3:     while until convergence do
4:         for $n = 1, \dots, N$ do
5:             for $j = 1, \dots, |\mathcal{S}|$ do
6:                 $g_1^n \sim \mathcal{G}_j, g_2^n \sim \mathcal{G}_j$
7:                 $v_{nj} \leftarrow z_j(\pi_j, g_1^n, g_2^n)$
8:             end for
9:             $\dot{L}_n \leftarrow \sum_{i=1}^{p} \lambda_i \mathrm{P}_n^i$
10:        end for
11:        $\nabla_{\mathit{\Pi}} L \leftarrow \frac{1}{N} \sum_{n=1}^{N} \nabla_{\mathit{\Pi}} \dot{L}_n + \nabla_{\mathit{\Pi}} \ddot{L}$
12:        $\mathit{\Pi} \leftarrow \min\{1, \big(\mathit{\Pi} + \alpha \nabla_{\mathit{\Pi}} L\big)_+\}$
13:        $\hat{\mathbf{X}} \leftarrow \hat{\mathbf{X}} + \beta \nabla_{\hat{\mathbf{X}}} L$
14:    end while
15:    $\lambda_i \leftarrow \big(\lambda_i - \gamma \nabla_{\lambda_i} L\big)_+, i = 1, \dots, p$
16:    $\mu_j \leftarrow \big(\mu_j - \eta \nabla_{\mu_j} L\big)_+, j = 1, \dots, |\mathcal{S}|$
17: end while
```

Approximated Lagrangian function for CCMS is written as follows.

$$
\begin{aligned}
L \approx \mathbb{E}_{\mathbf{G}\sim\mathcal{G}}\Big[& \sum_{j=1}^{|\mathcal{S}|} z_j(\mathit{\Pi}, \mathbf{G})\big(1 - \mu_j h_j(\hat{\mathbf{x}}_j)\big) \\
& + \sum_{i=1}^{p} \lambda_i \big\{ \Pr\big(\sum_{j=1}^{|\mathcal{S}|} z_j(\mathit{\Pi}, \mathbf{G}) \, ||\hat{x}_{ij} - x_{ij}||_2 - B_i \leq 0 \big) - (1-\epsilon) \big\} \Big] \\
= \frac{1}{N} & \sum_{n=1}^{N} \sum_{j=1}^{|\mathcal{S}|} v_{nj}\big(1 - \mu_j h_j\big) + \frac{1}{N} \sum_{n=1}^{N} \sum_{i}^{p} \lambda_i \mathrm{P}_n^i - (1-\epsilon) \sum_{i=1}^{p} \lambda_i \\
= \frac{1}{N} & \sum_{n=1}^{N} \tilde{L}_n + \frac{1}{N} \sum_{n=1}^{N} \dot{L}_n - (1-\epsilon) \sum_{i=1}^{p} \lambda_i
\end{aligned}
\tag{3.13}
$$

where $\mathbf{G} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ is a Gumbel matrix, and P_n^i is the same as in (3.11), but changing the binary variables for Categorical case such that

$$[\mathbf{z}]_j = z_j \approx v_{nj} = z_j(\mathit{\Pi}, \mathbf{G}^n) = \min(1, \sum_{\xi=1}^{|\mathcal{S}|} \frac{\exp((\log\pi_j + g_{j\xi}^n)/\omega)}{\sum_{k=1}^{|\mathcal{S}|}\exp((\log\pi_k + g_{k\xi}^n)/\omega)}).$$

We propose Algorithm 3 to solve $\min_{\mathit{\Lambda},M} \max_{\mathit{\Pi},\hat{\mathbf{x}}} L$, which is the same structure as the algorithm for BCMS (Algorithm 2). A part of simulating Gumbel's samples is edited for Categorical distribution (line 4-13).

Algorithm 3 CCMS

```
1: Initialize $\mathit{\Pi}, \mathit{\Lambda}, M, \alpha, \beta, \gamma, \eta$.
2: while until convergence do
3:     while until convergence do
4:         for $n = 1, \ldots, N$ do
5:             for $j = 1, \ldots, |\mathcal{S}|$ do
6:                 for $\xi = 1, \ldots, |\mathcal{S}|$ do
7:                     $[\mathbf{G}^n]_{j\xi} \leftarrow g_{j\xi}^n \sim \mathcal{G}$
8:                 end for
9:             end for
10:            $v_{nj} \leftarrow z_j(\mathit{\Pi}, \mathbf{G}^n), j = 1, \ldots, |\mathcal{S}|$
11:            $\tilde{L}_n \leftarrow \sum_{j=1}^{|\mathcal{S}|} v_{nj}\big(1 - \mu_j h_j\big)$
12:            $\dot{L}_n \leftarrow \sum_{i=1}^{p} \lambda_i \mathrm{P}_n^i$
13:        end for
14:        $\nabla_{\mathit{\Pi}} L \leftarrow \frac{1}{N}\sum_{n=1}^{N}(\nabla_{\mathit{\Pi}}\tilde{L}_n + \nabla_{\mathit{\Pi}}\dot{L}_n)$
15:        $\mathit{\Pi} \leftarrow \big(\mathit{\Pi} + \alpha\,\nabla_{\mathit{\Pi}} L\big)_+$
16:        $\pi_j \leftarrow \frac{\pi_j}{\sum_i \pi_i}, j = 1, \ldots, |\mathcal{S}|$
17:        $\hat{\mathbf{X}} \leftarrow \hat{\mathbf{X}} + \beta\,\nabla_{\hat{\mathbf{X}}} L$
18:    end while
19:    $\lambda_i \leftarrow \big(\lambda_i - \gamma\,\nabla_{\lambda_i} L\big)_+, i = 1, \ldots, p$
20:    $\mu_j \leftarrow \big(\mu_j - \eta\,\nabla_{\mu_j} L\big)_+, j = 1, \ldots, |\mathcal{S}|$
21: end while
```

Algorithm for KL. We present Lagrangian function for KL (3.9) as follows.

$$L(\hat{\mathbf{X}}, \varLambda, M) = \sum_{j=1}^{|\mathcal{S}|} l_j\,(\hat{\mathbf{x}}_j, \bar{\mathbf{y}}_j) + \sum_{i=1}^{p} \lambda_i\; g_i(\hat{\mathbf{x}}) + \sum_{\substack{u=1,\\ u\neq y_j}}^{k} \sum_{j=1}^{|\mathcal{S}|} \mu_{ju}\; h_{ju}(\hat{\mathbf{x}}) \tag{3.14}$$

where λ and μ are Lagrangian multipliers. Algorithm 4 is designed to solve $\max\limits_{\varLambda,M} \min\limits_{\hat{\mathbf{X}}} L$. Similar to Algorithm 1, it consists of two loops; the inner loop updates input features to minimize L, and the outer loop updates Lagrangian multipliers to maximize L.

Algorithm 4 KL

```
1: Initialize $\varLambda, M, \alpha, \delta$
2: while until convergence do
3:     $\hat{\mathbf{X}} \leftarrow \operatorname*{argmin}_{\hat{\mathbf{X}}} L(\hat{\mathbf{X}}, \varLambda, M)$
4:     $\lambda_i \leftarrow \left(\lambda_i + \alpha\, \nabla_{\lambda_i} L\right)_+, i = 1, \ldots, p$
5:     $\mu_{ju} \leftarrow \left(\mu_{ju} + \beta\, \nabla_{\mu_{ju}} L\right)_+, j = 1, \ldots, |\mathcal{S}|, u = 1, \ldots, k, u \neq y_j$
6: end while
```

Algorithm for Budget Allocation. In this paragraph, we present an extension of KL (3.9) and its related algorithm. Suppose that we have R groups of instances to be perturbed separately since they have different characteristics, and a given budget $\mathbf{B} = (B_1, B_2, \ldots, B_p) \in \mathbb{R}^p_+$ is to be assigned for all instances. In this setting, we have to solve a budget allocation problem. Each group $\mathcal{S}^r$ has the same settings as defined in the KL (3.9), and is disjoint, $\mathcal{S} = \bigcup_{r=1}^{R} \mathcal{S}^r$. Each problem is defined as $L^r = \max\limits_{\varLambda^r, M^r} \min\limits_{\hat{\mathbf{x}}^r} L(\hat{\mathbf{x}}^r, \varLambda^r, M^r)$ for $r = 1, \ldots, R$, and the budget constraint term in L^r is written as $\sum_{i=1}^{p} \lambda_i^r\; \{\sum_{j=1}^{|S^r|} ||\hat{x}_{ij}^r - x_{ij}^r||_2 - b_i^r\}$ where the budget is assigned by a size of instances in each group such that $b_i^r = \frac{|S^r|}{\sum_{\bar{r}=1}^{R} |S^{\bar{r}}|} B_i$. Once each group of perturbed instances is obtained by running Algorithm 4, we allocate budget to L^r problems, $r = 1, \ldots, R$.

Then, we solve a following budget allocation problem,

$$
\begin{aligned}
\min_{\mathbf{b}_1,\ldots,\mathbf{b}_R} \quad & \sum_{r=1}^{R} L^r(\mathbf{b}^r) \\
\text{s.t.} \quad & \sum_{r=1}^{R} \mathbf{b}^r = \mathbf{B}
\end{aligned} \tag{3.15}
$$

where $L^r(\mathbf{b}^r) = \max_{\Lambda^r, M^r} \min_{\hat{\mathbf{x}}^r} L(\hat{\mathbf{X}}^r, \Lambda^r, M^r | \mathbf{b}^r)$, and $\mathbf{b}^r = (b_1^r, b_2^r, \ldots, b_p^r) \in \mathbb{R}_+^p$. Algorithm 5 is proposed to solve the model (3.15). Note that in the algorithm $\nabla \mathcal{L} = \nabla_{b_i^r}(\sum_{\bar{r}=1}^{R} L_{\bar{r}}) = -\lambda_i^r$

Algorithm 5 Budget allocation (BA)

```
1: Initialize $b_i^r = \frac{|S^r|}{\sum_{\bar{r}=1}^{R} |S^{\bar{r}}|} B_i, r = 1, \ldots, R, i = 1, \ldots, p$
2: while until convergence do
3:     $b_i^r \leftarrow b_i^r - \alpha \nabla \mathcal{L}, r = 1, \ldots, R, i = 1, \ldots, p$
4:     $b_i^r \leftarrow \frac{b_i^r}{\sum_{\bar{r}=1}^{R} b_i^{\bar{r}}} B_i, r = 1, \ldots, R, i = 1, \ldots, p$
5: end while
```

and λ_i^r is Lagrangian multiplier with respect to a budget constraint for feature i in L^r; therefore, we run Algorithm 4 for each $\mathbf{b}^r$ update to compute $\nabla_{\mathbf{b}^r} \mathcal{L}$.

Evaluation. We describe how to evaluate our algorithms. The performance of algorithms is accessed based on how many samples are successfully perturbed within a given budget. We consider only feasible instances as they are regarded as a practical success in inverse classification. Since Lagrangian and subgradient methods do not necessarily guarantee the optimality and feasibility of solutions, we conduct the following steps to obtain a final solution set.

(1) Run proposed algorithms to obtain 'good' and possibly infeasible solutions.

(2) Find a subset of instances, $\tilde{\mathcal{S}} \subseteq \mathcal{S}$, satisfying prediction confidence constraints. That is, all of the instances in $\tilde{\mathcal{S}}$ are classified as desired.

(3) Solve the following problem over $\tilde{\mathcal{S}}$ to find a set of feasible instances.

$$\begin{aligned} \max_{z_1,\ldots,z_{|\tilde{\mathcal{S}}|}} \quad & \sum_{s=1}^{|\tilde{\mathcal{S}}|} z_s \\ \text{s.t.} \quad & \sum_{s=1}^{|\tilde{\mathcal{S}}|} a_{is}\, z_s \leq B_i, i = 1, \ldots, p \\ & z_s \in \{0, 1\}, s = 1, \ldots, |\tilde{\mathcal{S}}| \end{aligned} \tag{3.16}$$

where $a_{is} = ||\hat{x}_{is} - x_{is}||_2$, and B_i is a given budget.

(4) Obtain a final 'good' and feasible set, $\hat{\mathcal{S}} \subseteq \tilde{\mathcal{S}} \subseteq \mathcal{S}$.

3.5. Computational Study

In this section, we conduct a computational study on two datasets; a proprietary dataset and a public dataset. We experiment with different budget scenarios and scalability of the number of samples. Model implementations for all the experiments are done in Python using Tesla V100 GPU and Intel Xeon CPU E5-2697 v4 @ 2.30Hz for the real dataset; and Titan XP GPU and Intel Xeon Silver 4112 CPU @ 2.60GHz for the public dataset.

3.5.1. Real World Data

We conduct experiments on a real world proprietary dataset that contains sequential input features for true and false predictions, which is introduced in Stec et al. (2013). The dataset consists of five different types of features. They are three types of sequential features; sparse and dense features based on their frequency, and delta features that are related to the time between specified events in the sequence; and two types of static features. Besides, there are five different classes; one true target and four different false classes. More details are described in Stec et al. (2013). For a classifier to attack, we use a recurrent network that can deal with the five types of features, called the Sparse Time LSTM (STLSTM) (Stec et al., 2013). We use a trained STLSTM that achieves accuracy of around 70%.

3.5.1.1. Budget Experiments. We perturb 300 examples selected from test set and they are grouped into 15 different cases by input sequence lengths and original labels. That is, $|\mathcal{S}| = 300$, and $R = 15$. The 300 samples are originally labeled as one of false classes and correctly predicted by the trained classifier. Our purpose is to perturb them so that they are predicted as a true target. We perturb 19 dense features that can directly affect class predictions. To decide the sizes of budgets, we first run our algorithms with unlimited budgets to measure how much of the budget is needed for successful perturbation. Then, as well as practical consideration from data source experts we determine small, middle, and large sizes of budgets by the amounts that are proportional to total spent with the unlimited budgets. We use $\delta = 0.1$, and initialize Lagrangian multipliers with adding additive white Gaussian noise.

Figure 3.1 shows results of budget experiments. Note that the algorithm for budget allocation is denoted by BA, and the other algorithms are denoted by their model name. The final feasible solution set, $\hat{\mathcal{S}}$, is obtained based on the evaluation method in 3.4. We find that algorithms with Gumbel's method for BCMS and CCMS perform better than other algorithms. They achieve a larger size of successfully perturbed examples than other algorithms, and also they achieve lower spent per sample. This is because the objective of max samples models is to maximize the number of successfully perturbed samples. In addition, we observe that the larger budget achieves the larger size of successfully perturbed

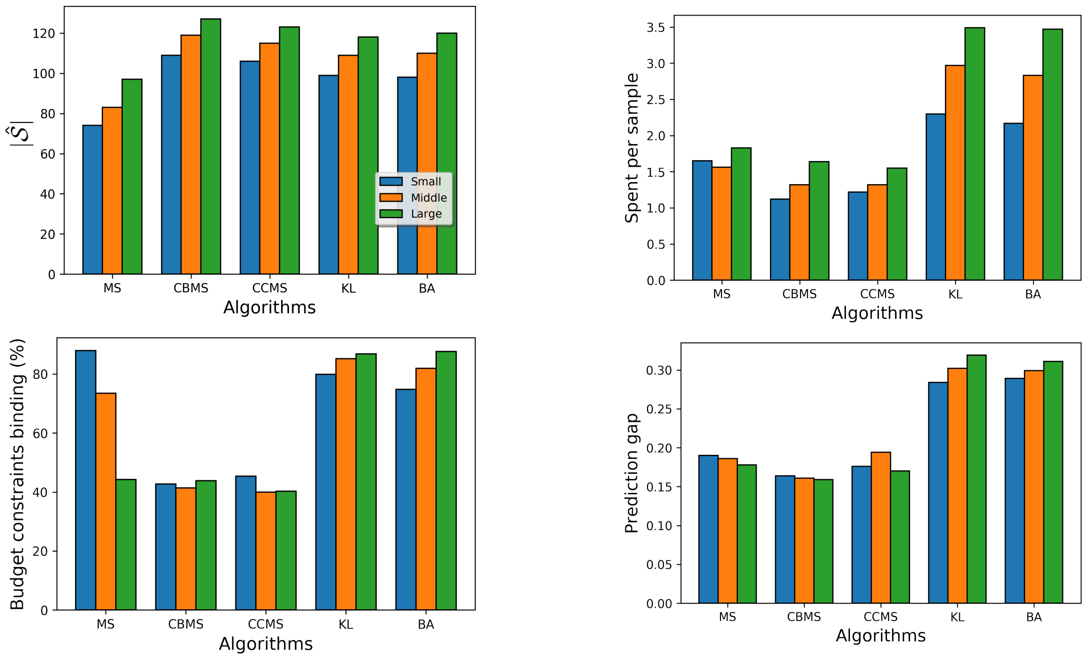


Figure 3.1. Real data: Budget experiments

samples for all algorithms, which is expected. Comparing KL to BA, improvement by BA is negligible. We also analyze budget and prediction confidence constraints. For budget constraints binding, we compute how much of the budget is spent for each budget constraint, and calculate the mean of them. In addition, a prediction gap is computed by measuring the gap between the top and the second top predictions. We find that budget constraint bindings of BCMS and CCMS are lower than other algorithms, and their prediction gaps are smaller than the others. We reason this as BCMS and CCMS spend budgets large enough to guarantee a certain level of confidence; it is larger than δ in their predictions, but not more than necessary. On the other hand, KL and BA have a large prediction gap that shows high confidence in their predictions; however, it might be more than necessary. This is why their spent per sample is relatively large.

3.5.1.2. Scalability Experiments. We conduct scalability analysis of our algorithms. We use three different sizes of samples $|\mathcal{S}| = 300, 600$, and 900, and samples in each set are grouped into 15 cases based on their input sequence lengths and labels. In addition, they have inclusive relationships such that $\mathcal{S}_{300} \subset \mathcal{S}_{600} \subset \mathcal{S}_{900}$. In this context, we have two strategies of initializing samples to be perturbed. First, we initialize input features of samples in a large set with ones previously obtained from a subset, and the rest of samples that are not in the subset, but in the large set is initialized by random, namely subset initialization. For example, we run an algorithm on $\mathcal{S}_{300}$, and then run the algorithm on $\mathcal{S}_{600}$. When we run it on $\mathcal{S}_{600}$, we initialize samples of $\mathcal{S}_{300}$ in $\mathcal{S}_{600}$ with obtained from the run on $\mathcal{S}_{300}$, and samples of $\mathcal{S}_{600} \setminus \mathcal{S}_{300}$ by random. The other strategy is to initialize all samples by random. Similar to budget experiments, all samples are originally labeled as one of false classes and correctly predicted by the trained classifier. We perturb 19 dense features so that they are predicted as a true target. We use the middle size of budget and the other hyperparameters as the same as those used in budget experiments 3.5.1.1.

Figure 3.2 shows results of scalability experiments. The final feasible solution set, $\hat{\mathcal{S}}$, is obtained based on the evaluation method in 3.4. Note that a run on $\mathcal{S}_{300}$ with subset initialization is denoted by 300-Sub, and one with random initialization is denoted by 300-Ran in the figure. Similar to the budget experiments, algorithms with Gumbel's method for BCMS and CCMS perform better than other algorithms. The number of successfully perturbed examples by them is larger than other algorithms, and also spent per sample is smaller. In terms of two initialization strategies, both cases show similar results; we conclude effectiveness of two initialization strategies is negligible in this instance. For analysis of budget and prediction confidence constraints, we find similar results to the

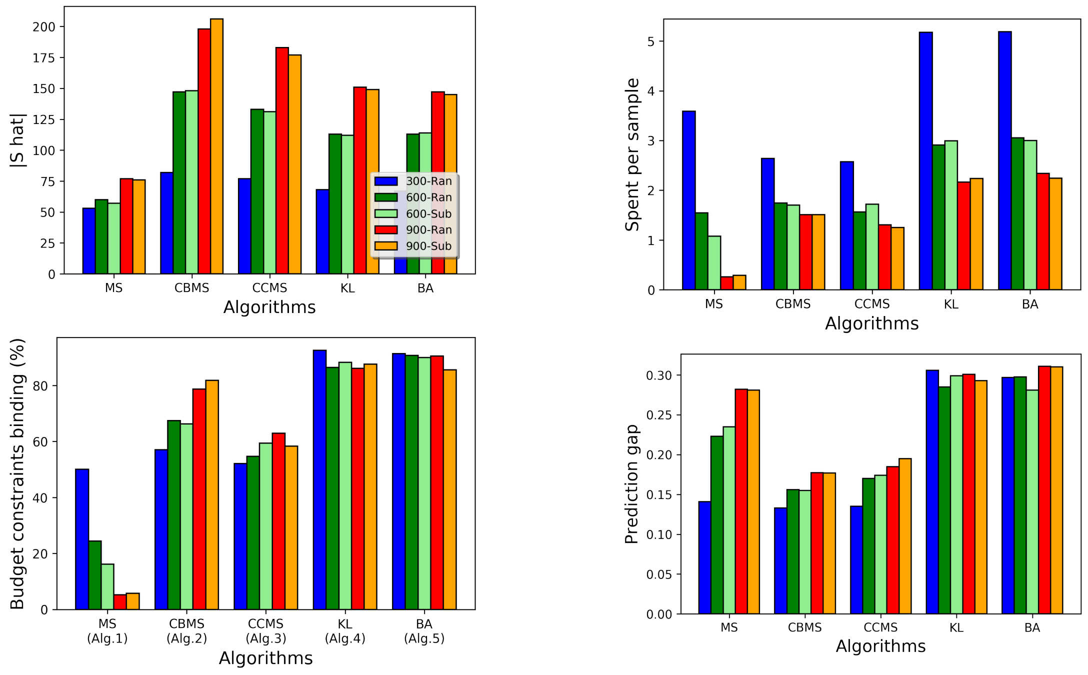


Figure 3.2. Real data: Scalability experiments

budget experiments. Budget constraints bindings for BCMS and CCMS are lower than KL and BA, and their prediction gaps are smaller than the others. The aforementioned analysis applies to all of different sizes of samples. We conclude that our algorithms can scale efficiently.

3.5.2. Public Data: MIMIC

MIMIC is a public dataset that describes clinical information of patients admitted to an Intensive Care Unit at the Beth Israel Deaconess Medical Center in Boston, Massachusetts from 2001 to 2012. It contains 58,576 data points for patients admissions. Baseline characteristics and in-hospital mortality outcome measures are as follows; the median age of adult patients is 65.86 years, 56.76% of patients are male, in-hospital mortality is around 10.49%, and the median length of a hospital stay is 7.08 days (Goldberger et al., 2000; Johnson et al., 2016). In this study, we use 13 input features for 30-day mortality predictions, which is used in Luo et al. (2018). The 13 features are sequential and continuous, and cover Chloride, Potassium, Bicarb, Sodium, Hematocrit, Hemoglobin, MCV, Platelets, WBC count, RDW, BUN, Creatinine, and Glucose. Detailed information on the features is described in Luo et al. (2018). Since MIMIC is a time series data and has missing values, we use a recurrent network based on Gated Recurrent Unit, called GRU-D, that is widely used for coping with multivariate time series with missing values (Che et al., 2018). In our experiment, we use a trained GRU-D that achieves AUC of around 0.78. We conduct only a budget experiment for this dataset since it is a small dataset for a scalability experiment.

We perturb 75 examples selected from test set and they are grouped into 5 different cases. That is, $|\mathcal{S}| = 75$, and $R = 5$. The 75 samples are originally labeled as dead and correctly predicted by the trained classifier. Our purpose is to perturb the samples so that they are predicted as alive. To decide the sizes of budgets, we first run our algorithm with unlimited budget constraints to measure how much perturbation is needed. Then, we determine small, middle, and large sizes of budgets by around 40%, 60%, and 80% of total spent with unlimited budgets. We use $\delta = 0.1$, and initialize Lagrangian multipliers with adding additive white Gaussian noise.

Figure 3.3 shows results of budget experiments. The final feasible set, $\hat{\mathcal{S}}$, is obtained based on the evaluation method in 3.4. Similar to results of the real data experiment, algorithms for max samples models, MS, BCMS and CCMS perform better than KL and BA. They obtain a larger size of successfully perturbed examples than the other

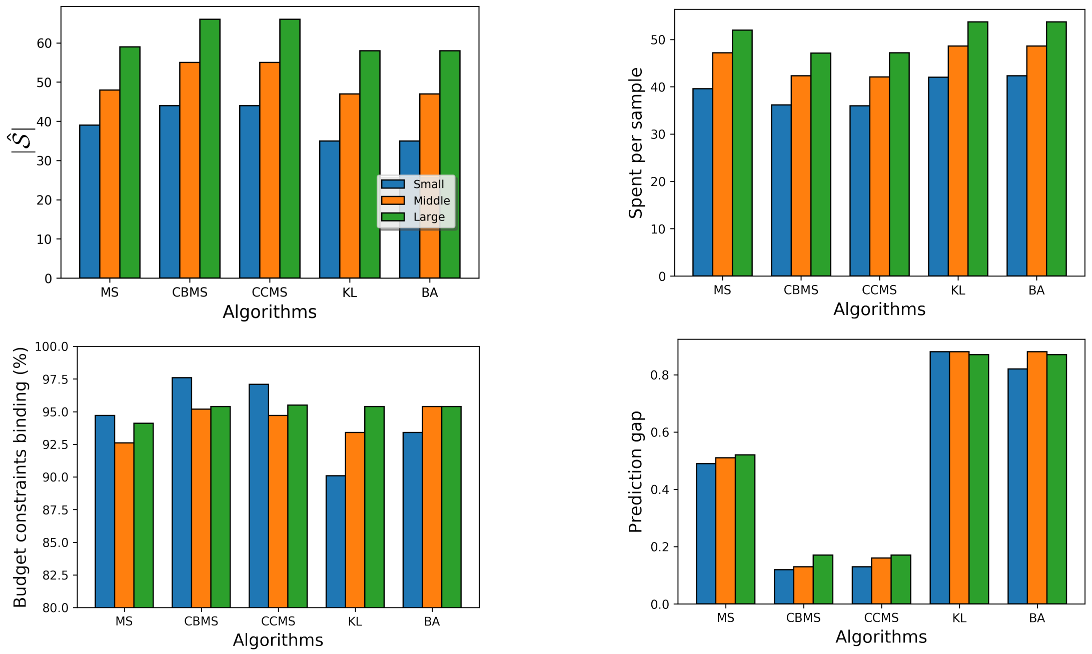


Figure 3.3. MIMIC: Budget experiments

algorithms, and also they achieve smaller spent per sample. This is because max samples models are formulated to maximize the number of successfully perturbed samples. Both KL and BA show similar performance in terms of the number of successfully perturbed examples, which means improvement by BA is negligible. Based on both real and public data experiments, we recommend using KL since BA requires a more computational effort than KL. In addition, we find that budget constraints of all algorithms are almost bindings. That is, most of the budgets are used. On the other hand, KL and BA have a larger prediction gap than MS, CBMS, and CCMS. KL and BA obtain high confidence in their predictions; however, it can be more than it needs to be since δ is 0.1.

3.6. Conclusion

In this study, a new framework based on mixed integer programming for inverse classification is proposed. We formulate a constrained optimization problem that maximizes the number of successfully perturbed samples with budget and prediction confidence constraints. In addition, we formulate a stochastic problem with chance constraints by extending the deterministic mixed integer problem. To solve our constrained problems, algorithms based on Lagrangian and subgradient methods are also developed. Based on an extensive computational study, we find that our algorithms perform greatly in various budget settings and achieve scalability of data.

References

Aggarwal, C. C., Chen, C., and Han, J. (2010). The inverse classification problem. *Journal of Computer Science and Technology*, 18(1):458–468.

Bannour, H. and Hudelot, C. (2012). Hierarchical image annotation using semantic hierarchies. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, pages 2431–2434.

Barbella, D., Benzaid, S., Christensen, J., Jackson, B., Qin, X. V., and Musicant, D. (2009). Understanding support vector machine classifications via a recommender system-like approach. In *the International Conference on Data Mining*, pages 305–311.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127.

Bengio, Y. and Lamblin, P. (2007). Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems (NIPS) 19*, volume 20, pages 153–160. MIT Press.

Bi, W. and Kwok, J. T. (2011). Multi-label classification on tree- and DAG-structured hierarchies. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 17–24.

Boylu, F., Aytug, H., and Koehler, G. J. (2010). Induction over strategic agents. *Information Systems Research*, 21(1):170–189.

Che, Z., Purushotham, S., Cho, K., Sontag, D., and Liu, Y. (2018). Recurrent neural networks for multivariate time series with missing values. *Scientific Reports*, 8(6085):645–653.

Chi, C.-L., Street, W. N., and Ward, M. M. (2012). Individualized patient-centered lifestyle recommendations: An expert system for communicating patient specific cardiovascular risk information and prioritizing lifestyle options. *Journal of Biomedical Informatics*, 45:1164–1174.

Cho, K., Bart van Merrienboer, and Caglar Gulcehre, a. D. B., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

Cho, K., Ilin, A., and Raiko, T. (2011). Improved learning algorithms for restricted Boltzmann machines. In *Artificial Neural Networks and Machine Learning (ICANN)*, volume 6791. Springer, Berlin, Heidelberg.

Dahl, G. E., Adams, R. P., and Larochelle, H. (2012). Training restricted Boltzmann machines on word observations. In *International Conference on Machine Learning (ICML) 29*, volume 29, pages 679–686, Edinburgh, Scotland, UK.

Deng, J., Krause, J., Berg, A. C., and Fei-Fei, L. (2012). Hedging your bets: Optimizing accuracy-specificity trade-offs in large scale visual recognition. In *Proceedings of the 2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3450–3457.

Deng, J., Satheesh, S., Berg, A. C., and Li, F. (2011). Fast and balanced: Efficient label tree learning for large scale object recognition. In *Proceedings of the 24th Conference on Neural Information Processing Systems (NIPS)*, pages 567–575.

Deng, Z., Vahdat, A., Hu, H., and Mori, G. (2016). Structure inference machines: Recurrent neural networks for analyzing relations in group activity recognition. In *Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4772–4781.

Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., and Li, J. (2018). Boosting adversarial attacks with momentum. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–12, Salt Lake City, USA.

Elfwing, S., Uchibe, E., and Doya, K. (2015). Expected energy-based restricted Boltzmann machine for classification. *Neural Networks*, 64:29–38.

Fischer, A. and Igel, C. (2012). An introduction to restricted Boltzmann machines. *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, 7441:14–36.

Gehler, P. V., Holub, A. D., and MaxWelling (2006). The rate adapting Poisson (RAP) model for information retrieval and object recognition. In *International Conference on Machine Learning (ICML) 23*, volume 23, pages 337–344, Pittsburgh, PA, USA.

Goldberger, A. L., Amaral, L. A. N., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C.-K., and Stanley, H. E. (2000). Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):2155–220.

Goodfellow, I., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*.

Graves, A., Liwicki, M., Bertolami, S. F. R., Bunke, H., and Schmidhuber, J. (2009). A novel connectionist system for improved unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 855–868.

Guo, Y., Liu, Y., Bakker, E. M., Guo, Y., and Lew, M. S. (2018). CNN-RNN: A large-scale hierarchical image classification framework. *Multimedia Tools and Applications*, pages 10251–10271.

Hassan, M. M., Alam, M. G. R., Uddin, M. Z., Shamsul Huda, A. A., and Fortino, G. (2019). Human emotion recognition using deep belief network architecture. *Information Fusion*, 51:10–18.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and*

Pattern Recognition (CVPR), pages 770–778.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800.

Hinton, G. E., Osindero, S., and Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–54.

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

Hu, H., Zhou, G. T., Deng, Z., Liao, Z., and Mori, G. (2016). Learning structured inference neural networks with label relations. In *Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2960–2968.

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-Excitation networks. In *Proceedings of the 2018 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7132–7141.

Jia, Y., Abbott, J., Austerweil, J. L., Griffiths, T. L., and Darrell, T. (2013). Visual concept learning: Combining machine vision and Bayesian generalization on concept hierarchies. In *Proceedings of the 26th Conference on Neural Information Processing Systems (NIPS)*, pages 1842–1850.

Johnson, A. E. W., Pollard, T. J., Shen, L., Lehman, L., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L. A., and Mark, R. G. (2016). Mimic-iii, a freely accessible critical care database. *Scientific Data*.

Krasin, I., Duerig, T., Alldrin, N., Ferrari, V., Abu-El-Haija, S., Kuznetsova, A., Rom, H., Uijlings, J., Popov, S., Kamali, S., Malloci, M., Pont-Tuset, J., Veit, A., Belongie, S., Gomes, V., Gupta, A., Sun, C., Chechik, G., Cai, D., Feng, Z., Narayanan, D., and Murphy, K. (2017). OpenImages: A public dataset for large-scale multi-label and multi-class image classification.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th Conference on Neural Information Processing Systems (NIPS)*, pages 1097–1105.

Kurakin, A., Goodfellow, I., and Bengio, S. (2017). Adversarial examples in the physical world. *Computing Research Repository (CoRR)*, abs/1607.02533v4.

Larochelle, H. and Bengio, Y. (2008). Classification using discriminative restricted Boltzmann machines. In *International Conference on Machine Learning (ICML) 25*, pages 536–543, Helsinki, Finland.

Larochelle, H., Mandel, M., Pascanu, R., and Bengio, Y. (2012). Learning algorithms for the classification restricted Boltzmann machine. *Journal of Machine Learning Research*, 13:643–669.

Lash, M. T., Lin, Q., Street, W. N., and Robinson, J. G. (2017a). A budget-constrained inverse classification framework for smooth classifiers. In *IEEE International Conference on Data Mining Workshops*, pages 1184–1193, New Orleans, LA, USA.

Lash, M. T., Lin, Q., Street, W. N., Robinson, J. G., and Ohlmann, J. (2017b). Generalized inverse classification. In *the 2017 SIAM International Conference on Data Mining (SDM)*, pages 162–170.

Laugel, T., Lesot, M.-J., Marsala, C., Renard, X., and Detyniecki, M. (2018). Comparison-based inverse classification for interpretability in machine learning. In Medina, J., Ojeda-Aciego, M., Verdegay, J. L., Pelta, D. A., Cabrera, I. P., Bouchon-Meunier, B., and Yager, R. R., editors, *Information Processing and Management of Uncertainty in Knowledge-Based Systems. Theory and Foundations*, pages 100–111, Cham. Springer International Publishing.

Li, L. J., Wang, C., Lim, Y., Blei, D. M., and Fei-Fei, L. (2010). Building and using a semantivisual image hierarchy. In *Proceedings of the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3336–3343.

Li, Y., Nie, X., and Huang, R. (2018). Web spam classification method based on deep belief networks. *Expert Systems With Applications*, 96(1):261–270.

Liang, M. and Hu, X. (2015). Recurrent convolutional neural network for object recognition. In *Proceedings of the 2015 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3367–3375.

Liu, B., Sadeghi, F., Tappen, M., Shamir, O., and Liu, C. (2013). Probabilistic label trees for efficient large scale image classification. In *Proceedings of the 2013 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 843–850.

Lowd, D. and Meek, C. (2005). Adversarial learning. In *KDD '05 Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 641–647, Chicago, Illinois, USA.

Lu, N., Li, T., Ren, X., and Miao, H. (2017). A deep learning scheme for motor imagery classification based on restricted Boltzmann machines. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25:566–576.

Luo, Y., Szolovits, P., Dighe, A. S., and Baron, J. M. (2018). 3D-MICE: Integration of cross-sectional and longitudinal imputation for multi-analyte longitudinal clinical data. *Journal of the American Medical Informatics Association*, 25(6):645–653.

Maddison, C. J., Tarlow, D., and Minka, T. (2014). A* sampling. In *Proceedings of the 27th Conference on Neural Information Processing Systems (NIPS)*, pages 3086–3094.

Mannino, M. V. and Koushik, M. V. (2000). The cost-minimizing inverse classification problem: a genetic algorithm approach. *Decision Support Systems*, 29:283–300.

Marszalek, M. and Schmid, C. (2007). Semantic hierarchies for visual object recognition. In *Proceedings of the 2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–7.

Marszalek, M. and Schmid, C. (2008). Constructing category hierarchies for visual recognition. In *Proceedings of the 2008 European Conference on Computer Vision (ECCV)*, pages 479–491.

Mccallum, A., Pal, C., Druck, G., and Wang, X. (2006). Multi-conditional learning : generative / discriminative training for clustering and classification. In *National Conference on Artificial Intelligence (AAAI)*, volume 21, pages 433–439.

Molnar, C. (2019). *Interpretable machine learning: A guide for making black box models explainable.* Creative Commons License.

Norouzi, M., Ranjbar, M., and Mori, G. (2009). Stacks of convolutional restricted Boltzmann machines for shift-invariant feature learning. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR)*, pages 2735–2742, Miami, FL, USA.

Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. (2016). The limitations of deep learning in adversarial settings. page 372–387.

Salakhutdinov, R., Torralba, A., and Tenenbaum, J. (2011). Learning to share visual appearance for multiclass object detection. In *Proceedings of the 2011 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1481–1488.

Salama, M. A., Eid, H. F., Ramadan, R. A., Darwish, A., and Hassanien, A. E. (2011). Hybrid intelligent intrusion detection scheme. *Advances in Intelligent and Soft Computing*, pages 293–303.

Sarikaya, R., Hinton, G. E., and Deoras, A. (2014). Application of deep belief networks for natural language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(4):778–784.

Schmah, T., Hinton, G. E., Zemel, R. S., Small, S. L., and Strother, S. (2009). Generative versus discriminative training of RBMs for classification of fMRI images. In *Advances in Neural Information Processing Systems (NIPS) 21*, volume 21, pages 1409–1416. Curran Associates, Inc.

Schwing, A. G. and Urtasun, R. (2015). Fully connected deep structured networks. *Computing Research Repository (CoRR)*, abs/1503.02351.

Shi, X., Chen, Z., Wang, H., Yeung, D. Y., WaikinWong, and chun Woo, W. (2015). Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Proceedings of the 28th Conference on Neural Information Processing Systems (NIPS)*, pages 802–810.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *Computing Research Repository (CoRR)*, abs/1409.1556.

Sivic, J., Russell, B. C., Zisserman, A., Freeman, W. T., and Efros, A. A. (2008). Unsupervised discovery of visual object class hierarchies. In *Proceedings of the 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8.

Srivastava, N. and Salakhutdinov, R. (2013). Discriminative transfer learning with tree-based priors. In *Proceedings of the 26th Conference on Neural Information Processing Systems (NIPS)*, pages 2094–2102.

Stec, A., Klabjan, D., and Utke, J. (2013). Unified recurrent neural network for many feature types. *ArXiv e-prints*, abs/1809.08717.

Stewart, R., Andriluka, M., and Ng, A. Y. (2016). End-to-end people detection in crowded scenes. In *Proceedings of the 2016 IEEE Computer Society Conference on Computer*

Vision and Pattern Recognition (CVPR), pages 1063–6919.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *Computing Research Repository (CoRR)*, abs/1312.6199.

Tousch, A. M., Herbin, S., and Audibert, J. Y. (2012). Semantic hierarchies for image annotation: A survey. *Pattern Recognition*, pages 333–345.

Tygar, J. D. (2011). Adversarial machine learning. *IEEE Internet Computing*, 15(5):4–6.

Vens, C., Struyf, J., Schietgat, L., Džeroski, S., and Blockeel, H. (2008). Decision trees for hierarchical multilabel classification. *Machine Learning*, pages 185–214.

Verma, N., Mahajan, D., Sellamanickam, S., and Nair, V. (2012). Learning hierarchical similarity metrics. In *Proceedings of the 2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2280–2287.

Wachter, S., Mittelstadt, B., and Russell, C. (2018). Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harvard Journal of Law and Technology*, 31(2):842–887.

Wang, B. and Klabjan, D. (2017). Regularization for unsupervised deep neural nets. In *National Conference on Artificial Intelligence (AAAI)*, volume 31, pages 2681–2687.

Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C., and Xu, W. (2016). CNN-RNN: A unified framework for multi-label image classification. In *Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2285–2294.

Wehrmann, J., Cerri, R., and Barros, R. C. (2018). Hierarchical multi-label classification networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 5075–5084.

Woo, S., Park, J., Lee, J.-Y., and Kweon, I. S. (2018). CBAM: Convolutional block attention module. In *Proceedings of the 2018 European Conference on Computer Vision (ECCV)*, pages 1–17.

Xiao, T., Zhang, J., Yang, K., Peng, Y., and Zhang, Z. (2014). Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 177–186.

Xing, E. P., Yan, R., and Hauptmann, A. G. (2005). Mining associated text and images with dual-wing Harmoniums. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 21, pages 633–641, Edinburgh, Scotland.

Yan, Z., Zhang, H., Piramuthu, R., Jagadeesh, V., DeCoste, D., Di, W., and Yu, Y. (2015). HD-CNN: Hierarchical deep convolutional neural networks for large scale visual recognition. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2740–2748.

Yang, C., Street, W. N., and Robinson, J. G. (2012). 10-year cvd risk prediction and minimization via inverse classification. In *the 2nd ACM SIGHIT symposium on International health informatics*, pages 603–610, Miami, Florida, USA.

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., and Lipson, H. (2015). Understanding neural networks through deep visualization. *In the Deep Learning Workshop, 31st International Conference on Machine Learning (ICML).*

Yu, L., Shi, X., and Shengwei, T. (2017). Classification of Cytochrome P450 1A2 of inhibitors and noninhibitors based on deep belief network. *International Journal of Computational Intelligence and Applications*, 16:1750002.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Proceedings of the 2014 European Conference on Computer Vision (ECCV)*, pages 818–833.

Zhang, M. L. and Zhou, Z. H. (2014). A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, pages 1819–1836.

Zhu, X. and Bain, M. (2017). B-CNN: Branch convolutional neural network for hierarchical classification. *Computing Research Repository (CoRR)*, abs/1709.09890.

APPENDIX A

Appendix: Chapter 1

A.1. Approximation of DBN Probability in the Proposed Models

DBN defines the joint distribution of the visible unit x and the ℓ hidden layers, $h^1, h^2, \cdots, h^\ell$ as

$$p(x, h^1, \cdots, h^\ell) = p(h^{\ell-1}, h^\ell) \left(\prod_{k=0}^{\ell-2} p(h^k | h^{k+1}) \right)$$

with $h^0 = x$.

DBN Fitting Plus Loss Model. From (1.2), $p(x)$ in the second term of the objective function is approximated as

$$p(x; \theta_{DBN}) = \sum_{h^1, h^2, \cdots, h^\ell} p(x, h^1, \cdots, h^\ell) \approx \sum_{h^1} p(x, h^1).$$

Expected Loss Models. $p(h|x)$ in the objective function is approximated as

$$\begin{aligned}
p(h^\ell|x) &\approx p(h^\ell|x, h^1, \cdots, h^\ell) \\
&= \frac{p(h^\ell, h^{\ell-1}, \cdots, h^1, x)}{p(h^{\ell-1}, h^{\ell-2}, \cdots, h^1, x)} \\
&= \frac{p(h^{\ell-1}, h^\ell)\left(\prod_{k=0}^{\ell-2} p(h^k|h^{k+1})\right)}{p(h^{\ell-2}, h^{\ell-1})\left(\prod_{k=0}^{\ell-3} p(h^k|h^{k+1})\right)} \\
&= \frac{p(h^{\ell-1}, h^\ell)p(h^{\ell-2}|h^{\ell-1})}{p(h^{\ell-2}, h^{\ell-1})} \\
&= \frac{p(h^{\ell-1}, h^\ell)p(h^{\ell-2}, h^{\ell-1})}{p(h^{\ell-2}, h^{\ell-1})p(h^{\ell-1})} \\
&= p(h^\ell|h^{\ell-1}).
\end{aligned}$$

Bilevel Model. From (1.5), $\nabla_{\theta_{DBN}} log\, p(x)$ in the objective function is approximated for $i = 0, 1, \cdots, \ell$ as

$$\begin{aligned}
[\nabla_{\theta_{DBN}} log\, p(x)]_i &= \frac{\partial\, log\, p(x)}{\partial\, \theta^i_{DBN}} \\
&= \frac{\partial\, log\, \left(\sum_{h^1,h^2,\cdots,h^\ell} p(x, h^1, h^2, \cdots, h^\ell)\right)}{\partial\, \theta^i_{DBN}} \qquad \text{(A.1)} \\
&\approx \frac{\partial\, log\, (\sum_{h^{i+1}} p(h^i, h^{i+1}))}{\partial\, \theta^i_{DBN}}
\end{aligned}$$

where $\theta_{DBN} = (\theta^0_{DBN}, \theta^2_{DBN}, \cdots, \theta^i_{DBN}, \cdots \theta^\ell_{DBN})$. The gradient of this approximated quantity is then the Hessian matrix of the underlying RBM.

A.2. Derivation of the Gradient of the Bilevel Model

We write the approximated $||\nabla_{\theta_{DBN}} - log\, p(x)||^2$ at the layer i as

$$||[\nabla_{\theta_{DBN}} - log\, p(x)]_i||^2 \approx ||\frac{\partial\ - log\ (\sum_{h^{i+1}} p(h^i, h^{i+1}))}{\partial\ \theta^i_{DBN}}||^2$$

$$= \Bigg[\left(\frac{\partial - log\, p(h^i)}{\partial\theta^i_{11}}\right)^2 + \left(\frac{\partial - log\, p(h^i)}{\partial\theta^i_{12}}\right)^2 + \cdots + \left(\frac{\partial - log\, p(h^i)}{\partial\theta^i_{nm}}\right)^2 \Bigg]$$

where m and n denote dimensions of h^i and h^{i+1} and θ^i_{pq} denotes the p^{th} and q^{th} component of the θ^i_{DBN}. The gradient of the approximated $||\nabla_{\theta_{DBN}} - log\, p(x)||^2$ at the layer i is

$$\frac{\partial}{\theta^i_{pq}} \left(\sum_{p,q} \left(\frac{\partial - log\, p(h^i)}{\partial\theta^i_{pq}} \right)^2 \right)$$

$$= 2\Bigg[\left(\frac{\partial - log\, p(h^i)}{\partial\theta^i_{11}}\right) \left(\frac{\partial^2 - log\, p(h^i)}{\partial\theta^i_{11}\theta^i_{pq}}\right) + \left(\frac{\partial - log\, p(h^i)}{\partial\theta^i_{12}}\right) \left(\frac{\partial^2 - log\, p(h^i)}{\partial\theta^i_{12}\partial\theta^i_{pq}}\right) + \cdots + \left(\frac{\partial - log\, p(h^i)}{\partial\theta^i_{pq}}\right) \left(\frac{\partial^2 - log\, p(h^i)}{\partial\theta^i_{pq}\partial\theta^i_{pq}}\right) + \cdots + \left(\frac{\partial - log\, p(h^i)}{\partial\theta^i_{nm}}\right) \left(\frac{\partial^2 - log\, p(h^i)}{\partial\theta^i_{nm}\theta^i_{pq}}\right) \Bigg]$$

for $p = 1, ...n, q = 1, ...m$. This shows that the gradient of the approximated $||\nabla_{\theta_{DBN}} - log\, p(x)||^2$ in (1.5) is then the Hessian matrix times the gradient of the underlying RBM. The stochastic gradient of $-log\, p(x)$ of RBM with binary input x and hidden unit h with respect to $\theta_{DBN} w_{pq}$ is

$$\frac{\partial RBM}{\partial w_{pq}} = p(h_p = 1|x)x_q - \sum_x p(x)p(h_p = 1|x)x_q$$

where RBM denotes $-log\, p(x)$ (Fischer and Igel, 2012). We derive the Hessian matrix with respect to w_{pq} as

$$\begin{aligned}
&\frac{\partial^2 RBM}{\partial w_{pq}^2} \\
&= \frac{\partial}{w_{pq}}[p(h_p = 1|x)x_q)] - \sum_x \frac{\partial}{w_{pq}}[p(x)p(h_p = 1|x)x_q)] \\
&= \sigma(\widetilde{net_p})(1 - \sigma(\widetilde{net_p}))x_q^2 - \sum_x [\frac{\partial p(x)}{\partial w_{pq}} p(h_p = 1|x)x_q \\
&\quad + p(x)\sigma(\widetilde{net_p})(1 - \sigma(\widetilde{net_p}))x_q^2], \\
&\frac{\partial^2 RBM}{\partial w_{pk} \partial w_{pq}} \\
&= \frac{\partial}{w_{pk}}[p(h_p = 1|x)x_q)] - \frac{\partial}{w_{pk}}[\sum_x p(x)p(h_p = 1|x)x_q)] \\
&= \sigma(\widetilde{net_p})(1 - \sigma(\widetilde{net_p}))x_q x_k - \sum_x [\frac{\partial p(x)}{\partial w_{pk}} p(h_p = 1|x)x_q \\
&\quad + p(x)\sigma(\widetilde{net_p})(1 - \sigma(\widetilde{net_p}))x_q x_k],
\end{aligned}$$

$$\begin{aligned}
&\frac{\partial^2 RBM}{\partial w_{kq}\partial w_{pq}}\\
&= \frac{\partial}{w_{kq}}[p(h_p = 1|x)x_q)] - \frac{\partial}{w_{kq}}[\sum_x p(x)p(h_p = 1|x)x_q]\\
&= -\sum_x[\frac{\partial p(x)}{\partial w_{kq}}p(h_p = 1|x)x_q + p(x)\frac{\partial}{\partial w_{kq}}[p(h_p = 1|x)x_q]],\\
&\frac{\partial^2 RBM}{\partial w_{kp}\partial w_{pq}} = -\sum_x[\frac{\partial p(x)}{\partial w_{kp}}p(h_p = 1|x)x_q + p(x)]
\end{aligned}$$

where $\sigma()$ is the sigmoid function, $\widetilde{net_p}$ is $\sum_q w_{pq}x_q + c_p$, and c_p is the hidden bias. Based on what we derive above we can calculate the gradient of approximated $||[\nabla_{\theta_{DBN}} - log\, p(x)]_i||^2$.

APPENDIX B

Appendix: Chapter 2

B.1. Deciding Dimensions for Convolutional Conversion

Let F_{conv_s}, K_{conv_s}, G_{conv_s}, and Z_{conv_s} denote the filter size, number of filters, stride, and zero padding for convolution Conv for $s \in S_t$, respectively. Also, let D_{conv_s}, W_{conv_s}, and H_{conv_s} denote the depth, width, and height of output after the convolution operation. In this conversion, the desired RNN or S2S input dimension, p, is determined by $D_{conv_s} \cdot W_{conv_s} \cdot H_{conv_s} = p$. The formulas for convolution lead to $D_{conv_s} = K_{conv_s}; W_{conv_s} = \frac{W_s - F_{conv_s} + 2Z_{conv_s}}{G_{conv_s}} + 1$; and $H_{conv_s} = \frac{H_s - F_{conv_s} + 2Z_{conv_s}}{G_{conv_s}} + 1$. We assume that H_s and W_s are the same, i.e. the feature map is shaped in a square.

1) $W_s = H_s = F_{conv_s}$: We have

$$\begin{aligned} p &= D_{conv_s} \cdot W_{conv_s} \cdot H_{conv_s} \\ &= K_{conv_s} \cdot \left(\frac{W_s - F_{conv_s} + 2Z_{conv_s}}{G_{conv_s}} + 1\right) \\ &\quad \cdot \left(\frac{H_s - F_{conv_s} + 2Z_{conv_s}}{G_{conv_s}} + 1\right), \end{aligned}$$

and thus $K_{conv_s} = p, Z_{conv_s} = 0$, and $G_{conv_s} \in \mathbb{Z}^+$.

2) $W_s = H_s > F_{conv_s}$, $F_{conv_s} = G_{conv_s}$: We have

$$\begin{aligned}
p &= D_{conv_s} \cdot W_{conv_s} \cdot H_{conv_s} \\
&= K_{conv_s} \cdot (\frac{W_s - F_{conv_s} + 2Z_{conv_s}}{G_{conv_s}} + 1) \\
&\quad \cdot (\frac{H_s - F_{conv_s} + 2Z_{conv_s}}{G_{conv_s}} + 1) \\
&= K_{conv_s} \cdot (\frac{W_s - F_{conv_s} + 2Z_{conv_s}}{F_{conv_s}} + 1)^2 \\
&= K_{conv_s} \cdot (\frac{W_s + 2Z_{conv_s}}{F_{conv_s}})^2.
\end{aligned}$$

This leads to $F_{conv_s} \in [1, W_s), Z_{conv_s}$ such that $(\frac{W_s+2Z_{conv_s}}{F_{conv_s}})^2 \in \mathbb{Z}^+$, and K_{conv_s} such that $\frac{p}{(\frac{W_s+2Z_{conv_s}}{F_{conv_s}})^2} \in \mathbb{Z}^+$.

3) $W_s = H_s > F_{conv_s}, F_{conv_s} \neq G_{conv_s}$, $G_{conv_s} = 1$: We have

$$\begin{aligned}
p &= D_{conv_s} \cdot W_{conv_s} \cdot H_{conv_s} \\
&= K_{conv_s} \cdot (\frac{W_s - F_{conv_s} + 2Z_{conv_s}}{G_{conv_s}} + 1) \\
&\quad \cdot (\frac{H_s - F_{conv_s} + 2Z_{conv_s}}{G_{conv_s}} + 1) \\
&= K_{conv_s} \cdot (\frac{W_s - F_{conv_s} + 2Z_{conv_s}}{G_{conv_s}} + 1)^2 \\
&= K_{conv_s} \cdot (W_s - F_{conv_s} + 2Z_{conv_s} + 1)^2,
\end{aligned}$$

and thus $F_{conv_s} \in [1, W_s), Z_{conv_s} = 0$, and K_{conv_s} such that $\frac{p}{(W_s - F_{conv_s} + 1)^2} \in \mathbb{Z}^+$.

4) $W_s = H_s > F_{conv_s}, F_{conv_s} \neq G_{conv_s}$, $G_{conv_s} \in (1, F_{conv_s})$: We have

$$\begin{aligned}
p &= D_{conv_s} \cdot W_{conv_s} \cdot H_{conv_s} \\
&= K_{conv_s} \cdot (\frac{W_s - F_{conv_s} + 2Z_{conv_s}}{G_{conv_s}} + 1) \\
&\quad \cdot (\frac{H_s - F_{conv_s} + 2Z_{conv_s}}{G_{conv_s}} + 1) \\
&= K_{conv_s} \cdot (\frac{W_s - F_{conv_s} + 2Z_{conv_s}}{G_{conv_s}} + 1)^2,
\end{aligned}$$

implying that $F_{conv_s} \in [1, W_s), G_{conv_s} \in (1, F_{conv_s}), Z_{conv_s}$ such that $(\frac{W_s - F_{conv_s} + 2Z_{conv_s}}{G_{conv_s}} + 1)^2 \in \mathbb{Z}^+$, and K_{conv_s} such that $\frac{p}{(\frac{W_s - F_{conv_s} + 2Z_{conv_s}}{G_{conv_s}} + 1)^2} \in \mathbb{Z}^+$.

B.2. Deciding Dimensions for Pooling Conversation

A desired RNN input dimension, p, is determined by satisfying $D_{pool_s} \cdot W_{pool_s} \cdot H_{pool_s} = p$. The pooling operation leads to $D_{pool_s} = D_s$; $W_{pool_s} = \frac{W_s - F_{pool_s}}{G_{pool_s}} + 1$; and $H_{pool_s} = \frac{H_s - F_{pool_s}}{G_{pool_s}} + 1$. We assume that H_s and W_s are the same, i.e. the feature map is shaped in a square, and F_{pool_s} and G_{pool_s} are the same, i.e. disjoint pooling.

1) $H_s = W_s$, $F_{pool_s} = G_{pool_s}$: We have

$$\begin{aligned}
p &= D_{pool_s} \cdot W_{pool_s} \cdot H_{pool_s} \\
&= D_s \cdot (\frac{W_s - F_{pool_s}}{G_{pool_s}} + 1) \cdot (\frac{H_s - F_{pool_s}}{G_{pool_s}} + 1) \\
&= D_s \cdot (\frac{b_s - a_{pool_s}}{a_{pool_s}} + 1)^2 \\
&= D_s \cdot (\frac{b_s}{a_{pool_s}})^2,
\end{aligned}$$

and thus $a_{pool_s} = b_s \cdot \sqrt{\frac{D_s}{p}} = F_{pool_s} = G_{pool_s}$.

2) $H_s = W_s$, $F_{pool_s} \neq G_{pool_s}$: We have

$$\begin{aligned} p &= D_{pool_s} \cdot W_{pool_s} \cdot H_{pool_s} \\ &= D_s \cdot (\frac{W_s - F_{pool_s}}{G_{pool_s}} + 1) \cdot (\frac{H_s - F_{pool_s}}{G_{pool_s}} + 1) \\ &= D_s \cdot (\frac{W_s - F_{pool_s}}{G_{pool_s}} + 1)^2, \end{aligned}$$

leading to $F_{conv_s} \in [1, W_s)$, and $G_{conv_s} \in (1, F_{conv_s})$.

B.3. Description of the General Tree of Open Images

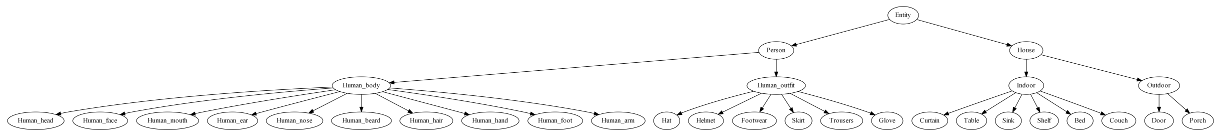


Figure B.1. The full general tree of Open Images

Open Images provides a semantic hierarchy that consists of 600 object classes (Krasin et al., 2017). We first take subtrees of classes that have mainly Has-A relationships such as 'Human body'-'Human foot' and 'Human outfit'-'Hat,' and then concatenate the subtrees to create the final class tree. The tree is of depth four with 30 class nodes. Figure B.1 presents the whole tree.