RISK-CALIBRATED EVIDENTIAL CLASSIFIERS

A Thesis

by

Maryam Saleki

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Computer Science

Özyeğin University
January 2020

RISK-CALIBRATED EVIDENTIAL CLASSIFIERS

Approved by:

Assistant Professor Reyhan Aydoğan Advisor
Department of Computer Science
Özyeğin University

Associate Professor Murat Şensoy
Department of Computer Science
Özyeğin University

Assistant Professor Boray Tek
Department of Computer Engineering
Işık University

Date Approved: 17 January 2020

To my family

ABSTRACT

In some applications, intelligent agents rely on classifiers in order to make their decisions and accuracy of their predictions may play a significant role in performing their tasks successfully. Although deep neural networks perform very well in many classification tasks, they may sometimes fail in their predictions and the cost of all misclassification errors are usually considered as the same, which is not true in practice. For instance, classifying a pedestrian in a given image as a cyclist may cost significantly different from classifying it as a car for a self-driving car application. The costs of errors can be asymmetric, vary from agent-to-agent, and depend on context. Accordingly, this thesis proposes a novel approach for uncertainty quantification and risk-awareness in deep neural networks for classification.

Our main intuition is that the predictive uncertainty can be quantified in a principled way; hence, classifiers can associate high uncertainty with their predictions when these predictions are more likely to be wrong. Furthermore, they incorporate the notion of misclassification risk during training, which allows them to avoid making wrong predictions leading to higher losses. To achieve this, the proposed risk-calibrated classifiers quantify the uncertainty in predictions based on the mean and variance of the Dirichlet distribution, and increase the uncertainty value for the predictions, which are more likely to be wrong. Furthermore, the model increases the uncertainty for the classifications, which are more risky.

To validate the performance of our approach, we conducted experiments on a variety of well-known data sets. The results show that the proposed risk-calibrated classifiers associate high uncertainty with their misclassification. Furthermore, the risk minimization objective of our loss function allows neural networks to make less risky decisions for classification.

ÖZETÇE

Bazı uygulamalarda, akıllı etmenler kararlarını vermek için sınıflandırma algoritmalarına güvenir ve tahminlerinin doğruluğu görevlerini başarıyla yerine getirmede önemli bir rol oynayabilir. Derin sinir ağları birçok sınıflandırma görevinde çok iyi performans gösterse de, bazen tahminlerinde başarısız olabilirler. Genellikle, tüm yanlış sınıflandırma hatalarının maliyeti aynı kabul edilir; fakat bu pratikte doğru değildir. Örneğin, kendi kendini süren bir araba uygulaması için, görüntüdeki yayayı bisikletli olarak tahmin edilmesi ile araba olarak tahmin edilmesinin maliyeti önemli ölçüde farklı olabilir. Hataların maliyeti asimetrik olabilir, etmenden etmene değişebilir ve içeriğe bağlıdır. Bu tez, derin sinir ağlarında sınıflandırma için belirsizlik ölçümü ve risk farkındalığı için yeni bir yaklaşım önermektedir.

Ana sezgimiz, öngörücü belirsizliğin ilkeli bir şekilde ölçülebilmesidir; bu nedenle, bu tahminlerin yanlış olma olasılığı daha yüksek olduğunda sınıflandırıcılar yüksek belirsizliği tahminleriyle ilişkilendirebilirler. Ayrıca, eğitim sırasında yanlış sınıflandırma riski göz önünde bulundurlar; bu da daha yüksek kayıplara yol açan yanlış tahminler yapmaktan kaçınmalarını sağlar. Bunu başarmak için, önerilen risk kalibrasyonlu sınıflandırıcılar, Dirichlet dağılımının ortalamasına ve varyansına bağlı olarak tahminlerdeki belirsizliği ölçmekte ve yanlış olma olasılığı daha yüksek olan tahminler için belirsizlik değerini arttırmaktadır. Ayrıca, model daha riskli olan sınıflandırmalar için belirsizliği arttırmaktadır.

Yaklaşımımızın performansını doğrulamak için, iyi bilinen çeşitli veri setleri üzerinde deneyler yaptık. Sonuçlar, önerilen risk-kalibre edilmiş sınıflandırıcıların, yüksek belirsizliği yanlış sınıflandırmalarıyla ilişkilendirdiklerini göstermektedir. Ayrıca, kayıp fonksiyonumuzun risk minimizasyon hedefi, sinir ağlarının sınıflandırma için daha az riskli kararlar vermesini sağlar.

ACKNOWLEDGEMENTS

Completion of this master dissertation was possible with the support of several people. I would like to thank all of them.

Foremost, I would like to express my gratitude to my thesis advisor Assist.Prof. Reyhan Aydoğan for guiding and supporting me during three years of my Master's studies. She convincingly guided and encouraged me to be professional and do the right thing even when the road became tough. I admire her work ethics and discipline.

I would also like to pay my special regards to Assoc.Prof.Murat şensoy for his continuous support, motivational advice and immense knowledge. He has been a tremendous mentor for me. Without his persistent help, the goal of this thesis would not have been realized.

I would also like to thank my committee member Assist.Prof. Boray Tek for spending time reading this thesis and providing useful feedback regarding the content.

I would like to say a heartfelt thank you to my family and all my friends for always believing in me and encouraging me to follow my dreams and helping me in whatever way they could during this challenging period.

I want to thank Artificial Intelligence laboratory members because of their warm friendship. They were all kind; and it was fun to work together in the laboratory.

This research is supported by the Newton Fund and the Scientific and Technological Research Council of Turkey (TUBITAK) under grant number 116E918.

This work has significantly benefited from the discussions and collaboration with a number of great researchers including Dr Simon Julier from the University College London (UCL) and Dr John Reid from Blue Prism AI Labs. I would like to thank them for their support and contribution to this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Nomenclature

$\boldsymbol{f}_\theta(\cdot)$	Output of logits layer
α	Dirichlet distribution parameter
β	Beta distribution
$\boldsymbol{\gamma_\theta}(\boldsymbol{x})$	Redistribution of uniform prior counts
$\boldsymbol{f'_\theta}(\boldsymbol{x})$	Input of the logits layer
$\boldsymbol{g}_\theta(\cdot)$	Neural network parameterised by θ
$\boldsymbol{p}$	Pignistic probability
$\boldsymbol{\mathcal{L}}$	Base loss
μ	Mean of Gaussian distribution
μ_{MLE}	Maximum likelihood estimation for Gaussian distribution mean
$\odot$	Element-wise product
σ	Variance of Gaussian distribution
σ_{MLE}	Maximum likelihood estimation for Gaussian distribution variance
a	Neural network activation function
b	Neural network bias
c	Evidence vector for dirichlet distribution
D	Kullback-Leibler Divergence distance
E	Probability event

H	Cross entropy loss
H	Probability hypothesis
KL	Kullback-Leibler Divergence
MAP	Maximum A Posterior
MLE	Maximum Likelihood Estimation
$P(A, B)$	Joint probability of A and B
$P(A \mid B)$	Conditional probability of A given B
$p(x)$	Probability of x
PDF	Probability Density Function
PMF	Probability Mass Function
w	Weight
E	Expected cross entropy loss
C	Cost matrix
R	Risk matrix
var(π_k)	Variance of dirichlet distribution
π_k	Mean of dirichlet distribution

CHAPTER I

INTRODUCTION

Deep learning has drawn a lot of attention in recent years because of its overwhelming success and flexibility in various learning tasks like classification and regression. Deep learning models exceeded human performance in some tasks such as image recognition, video games, voice generation and medical diagnoses. However, unlike humans, they do not reason about the consequences of their possible mistakes. These models are mostly based on loss functions that do not take into account the distribution of probability mass over the wrong categories. For instance, the most widely used loss metric in classification in deep learning is the cross entropy, which is formalized as the negative log likelihood of the predicted probability for the true category. It only rewards the true category and completely ignores how the remaining probability mass is distributed over the wrong categories and the associated risk of choosing these categories. This means that a classifier trained with this loss may be completely ignorant of the agent's risk of making wrong classification decisions, where wrong decision may pay the high cost for these models. For example, on 7th May 2016, a car operating with automated vehicle control systems crashed with a truck near Williston, Florida, USA. Unfortunately, the car driver died due to the severe injury. The car manufacturer reported that the car's vision system classified the white side of the truck as the sky.

In order to incorporate the cost of misclassification into the training of neural network classifiers, one can use cost-sensitive learning [1], that aims to minimize the cost of misclassification errors by avoiding predictions placing high probabilities for high-risk categories, in addition to increasing the classification accuracy. Cost-sensitive learning is not able to quantifying the uncertainity in their predictions. This means that an autonomous agent using these classifiers cannot estimate if it can depend on their predictions for making a decision or taking action. Hence, the main advantage of these classifiers for the agent would be limited to decreasing the cost of classification errors due to their tendency to predict less risky categories. Few different methods have been proposed for quantifying the uncertainty in deep neural network classifiers in recent years. [2] proposed evidential deep neural network classifiers for uncertainty quantification. However, none of the models considered the risk of classification errors on their predictions. In this works, we propose the risk-calibrated deep classifiers by reformulating the evidential deep learning within the Bayesian learning framework by placing the prior distribution on the generated evidence using misclassification risk. Our proposed model trained using two main objectives in mind: (i) their predictive uncertainty can be quantified in a principled way; hence, they can associate high uncertainty with their predictions when these predictions are more likely to be wrong, (ii) they incorporate the notion of misclassification risk during training, which allows them to avoid making wrong predictions leading to higher losses. To achieve this, our risk-calibrated classifiers do two things. First, it quantifies the uncertainty in its predictions. It refuses evidences that lead to high wrong predictions, and second, it tries to make rational decisions under the cases that contain uncertainty.

The remaining of the thesis is organized as follows. In Chapter 2, we briefly describe the main concepts related to deep neural networks. In Chapter 3, we introduce the research problem. In Chapter 4, we describe the evidential deep learning and uncertainty quantification in detail and propose a novel approach for learning how to make decisions with uncertain predictions by extending the evidential deep learning. In Chapter 5, we evaluate our approach concerning other methods using well-known data sets. In Chapter 6, we discuss the related work and the proposed approach.

CHAPTER II

BACKGROUND

Machine learning is one of the fundamental areas of the Artificial Intelligence. It aims to explain the observed data, find out the patterns/regularities in the data and accordingly build models to achieve various tasks [3]. It mainly benefits from the theory of statistics. There are a variety of machine learning applications. While we can use machine learning to guess whether it is risky to give credits for a given customer (classification) or to predict the rent of a particular house based on its features (regression), we can find out the similar customers based on their shopping behaviour (clustering). The former is an example of supervised learning and the latter example is unsupervised learning.

In supervised learning, the training data is labelled using discrete or continuous values. If the labels are discrete, the learning problem is called classification, where the labels are usually mutually exclusive categories. When the labels are real-valued, the learning is called regression. In this thesis, we focus on classification problems and more specifically the classifiers based on deep neural networks. In this chapter, we first provide required background for deep neural networks (Section 2.1) and then present the fundamental theories of probability (Section 2.2).

2.1 *Deep Neural Networks*

The powerhouse of human intelligence is the brain. Human brain is composed of 100 billions of neurons, which are interacting with one another through dense connections [4]. Artificial neural networks is based on the idea of simulating how human brain works to solve complicated computational problems [5]. Each neuron in the human brain consists of different parts such as dendrites, axons, and synapses, and interact with other neurons by passing messages through biochemical and electrical signals (see Figure 1). The artificial neural networks, as similar to human brain, aim to learn patterns in sensory data (e.g., images) to solve tasks such as classification. The computation in neural networks is based on matrix operations and learning is performed usually by means of gradient descent based methods. In neural networks (see Figure 1), each neuron is parametrized by a *weight* vector $\boldsymbol{w}$ and a scalar b, which is called bias. Given an input vector $\boldsymbol{x}$, the output of the neuron for this input is calculated as Equation (1) where the output y could have any range between $-\infty$ and $+\infty$, which is feed into a non-linear function, called an *activation* function.

$$y = \boldsymbol{w}^T \boldsymbol{x} + b. \tag{1}$$

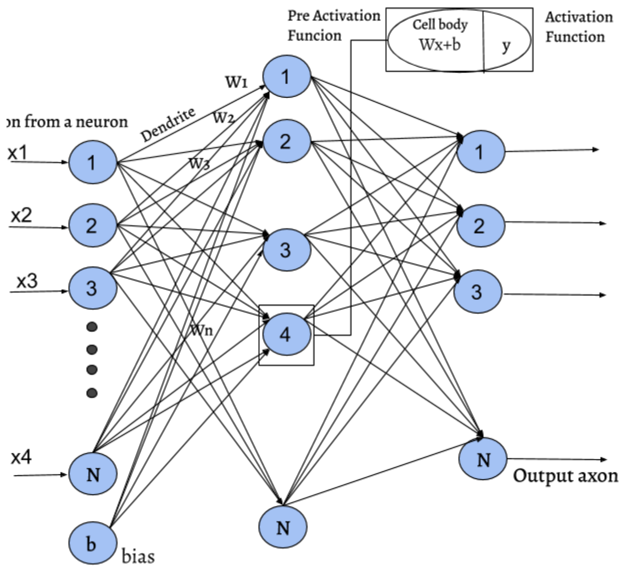


Figure 1: An Example Neural Network.

When a neural network has multiple hidden layers as shown in Figure 2, it is called deep neural networks. Figure 2 shows a deep neural network architecture with three hidden layers between the input and output layers. As the number of hidden layers increases, a more complicated function can be learnt by the model. In this thesis, we use a special type of deep neural network called *convolutional neural networks*, which is widely used on computer vision applications [6].

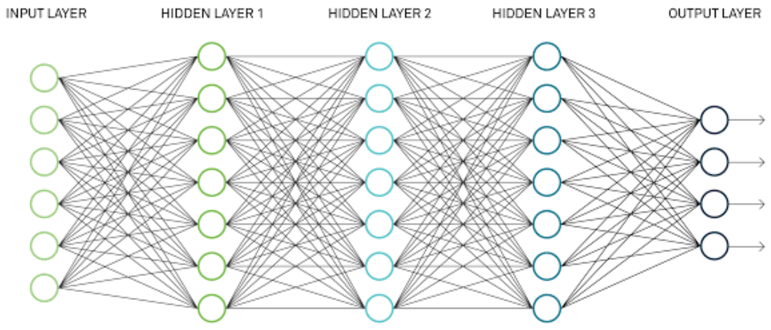


Figure 2: Deep neural network with three hidden layers.

In convolutional neural networks as shown in Figure 3, the hidden layers consist of convolutional layers, pooling layers and also fully connected layers as explained follows.

- **Convolutional layers:** They are used to extract features to be used in the following layer by using filters (i.e., kernels). These filters are $m \times m$ square matrices and contain numeric values called weights. The value of each pixel is multiplied by this matrix; which results in a new matrix called "feature map".

- **Pooling layers:** Pooling layers reduce the size of input image by discarding the useless information while preserving the important ones. For instance, the special types of pooling which is called Max pooling, transforms the given feature map of the image into several windows with certain size (e.g., 2×2 matrix) by sliding it continuously from the left to the right through the entire image and maps each window to a single value by choosing the largest value in the window.

- **Fully connected layers:** The feature map constructed by previous layers flattens into a vector and given as an input to the fully connected layers. Based on the classification model (e.g., binary or multi-class), these layers output the values for each class label.

Similar to the standard neural networks, in convolutional neural networks we also use activation function in order to increase the non-linearity of the model. According to Figure 3 , we can use activation functions before pooling layers as well as after the pooling layers.

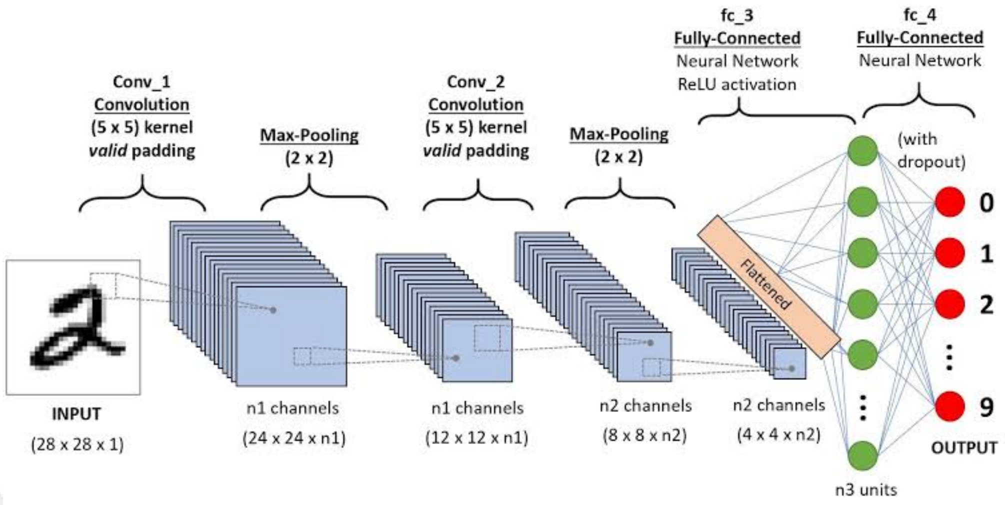


Figure 3: Convolutional Neural Network Architecture.[1]

2.1.1 Activation Functions

The activation functions in neural networks have the same responsibility as axons in the human brain; it plays a significant role in the process of learning in neural networks. The neural network without activation function, no matter how many layers embedded between the input and output layer, acts as a single layer perceptron, and its output will be a linear transformation of the input, which is not enough for a network to learn complex nonlinear functions. There are a variety of activation functions defined for neural networks in the literature as follows:

- **Step function**: It is a binary activation function, which is used in two-class classification problems (e.g., when the outcome of a classifier is yes or no). The output of this function depends on a threshold value. If the input value exceeds the threshold, then the function outputs 1; otherwise, it outputs 0. One of the drawbacks of this function is that its gradient is always zero, which made it useless for back-propagation [8]. Since neural networks tune their parameters such as weight and biases using a

[1]Figure is taken from [7]

gradient; therefore, step function would not be the right choice.

- **Sigmoid Function**: It is a non-linear activation function that is used for binary classification. Its output is in the range of $[0, 1]$. Sigmoid activation function formulated as Equation (2). According to this formula, negative numbers approaches to 0 and positive numbers approaches to 1 as their magnitude increases. Derivative of sigmoid function can be calculated easily. That's why it is commonly used in neural networks. Since the value of the probability is always between zero and one, sigmoid activation function is suitable for predicting the probability of target classes.

  $$\text{sigmoid}\left(x\right) = \frac{1}{1 + e^{-x}} \tag{2}$$

  One of the drawbacks of sigmoid activation function is that the large change in the input values has insignificant effect on it's output which means the large positive and large negative inputs will have derivative near to zero. As the number of layers with sigmoid activation function increase, we will have many small derivatives that multiplied with each other which decrease the gradient. In this situation the parameters of network such as weights and biases do not update properly which decrease the accuracy of the learning algorithm.

- **Tanh Function**:

  One of the drawbacks of sigmoid activation function is that it causes neural network to get stuck during the training phase. This happens when large negative inputs pass to the sigmoid, and sigmoid outputs the values that are very close to the zero. It causes to produce the same output. In tanh activation function, the output is zero centered as shown in Figure 4 (c) and it maps inputs to the values between -1 and +1. Tanh activation function specified in Equation (3).

  $$\tanh\left(x\right) = \frac{2}{1 + e^{-2x}} - 1 \tag{3}$$

- **ReLU Function**: It stands for rectified linear unit and easy to implement. That is, if the value of x is positive it outputs the value of x itself; otherwise, it outputs zero as shown in Equation (4). One of the advantages of Relu activation function is that it does not activate all the neurons. Since Relu has gradient 1 for the outputs greater than 0 and 0 otherwise, then it's derivative bounded by the range (0,1) and it can solves vanishing gradient problem [8]. The range of this function is $[0, inf)$

$$\text{relu}\left(x\right) = max\left(0, x\right) \tag{4}$$

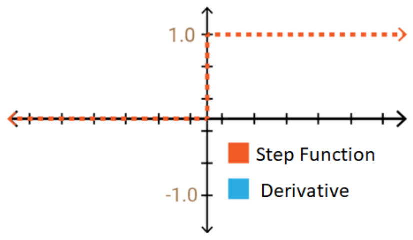
- **Leaky Relu Function**: This function is also a piece-wise linear function of the input x. Here, the negative values of the input is not mapped to zero; instead, they are multiplied by a small positive constant, α as denoted in Equation (5). By adding alpha term to the equation, it solves the vanishing gradient problem of ReLU function for negative input values.

$$LReLU\left(x\right) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases} \tag{5}$$

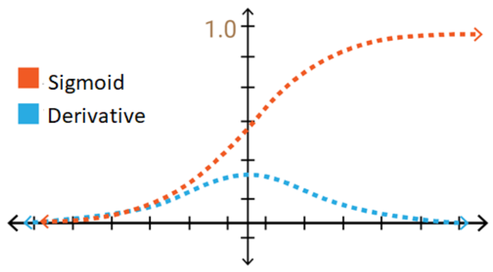
- **Softmax Function**: This function is the generalization of sigmoid function. While the sigmoid is used for binary classification, the softmax is suitable for multi-class classification problems. It converts logits to the probabilities over a number of categories. The softmax function can defined as Equation (6).

$$f\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=0}^{k} e^{x_j}} \quad i = 0, 1, 2, ..., k \tag{6}$$

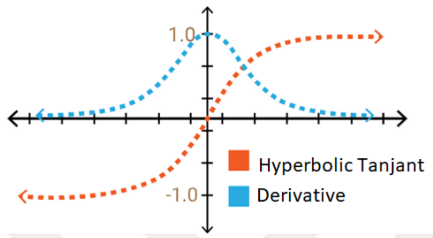
Figure 4 (See url [9]) demonstrates the different activation functions and their derivative which are described above.



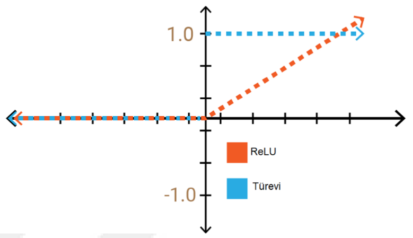
(a) $Step$



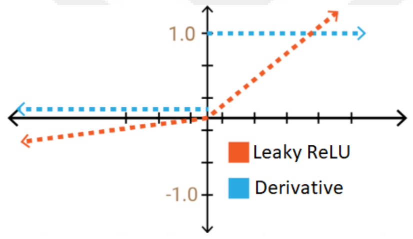
(b) $Sigmoid$



(c) $Tanh$



(d) $Relu$



(d) $LeakyRelu$

Figure 4: Neural Network Activation Functions.[2]

2.1.2 Loss Functions

The neural networks are trained using gradient decent algorithm or its variants to find the optimal values for the network parameters. In order to evaluate how well the neural network trained, we need a performance measurement. In classification problems, we use loss functions (i.e., cost functions) to measure classification errors. The objective is typically to minimize the loss function in order to find the parameters of neural network that best fits the data. There are different loss functions for binary and multi-class classification algorithms. In this thesis we focus on cross entropy and KL divergence losses.

[2]The Figure is taken from [9]

- **Cross Entropy**: Given inputs, the model tries to make a prediction near to the distribution of the target classes. In other words, the cross entropy loss function measures to what extents the probability distribution of the predicted values is similar to the probability distribution of the target classes. Equation (7) shows how to calculate the cross entropy where $p(x)$ denotes the true distribution of target classes and $q(x)$ denotes the predicted distribution. The true distribution $p(x)$ is always 1 for true classes and 0 for wrong classes.

$$H\left(p,q\right) = -\sum_{x} p\left(x\right) \log q(x) \tag{7}$$

  It is worth noting that this formula only cares about the probability of the prediction belongs to the true class. To illustrate this, consider the image classifier with outcome probabilities for the following three classes: *cat*, *dog* and *horse* such as [0.8,0.8,0.3]. Their corresponding true distribution is [0,1,0] denoting that the given instance is a dog. Since the cross entropy loss only regards true class distribution, the probabilities of cat and horse is ignored.

- **Kullback-Leibler (KL) Divergence**: It is a metric for measuring the distances of two probability distributions and computed as Equation (8).

$$D\left(p|q\right) = \sum_{x} p(x) log \frac{p(x)}{q(x)} = \underbrace{\sum_{x} p\left(x\right) \log p(x)}_{\text{negative entropy}} - \underbrace{\sum_{x} p\left(x\right) \log q(x)}_{\text{cross entropy}} \tag{8}$$

From the point of information theory, KL divergence represents the difference between the cross entropy and the entropy. It can be interpreted as the number of extra bits required to encode information if the actual distribution of the data (p) is different from the prediction distribution (q).

2.1.3 Training Neural Networks

The modern artificial neural networks for supervised learning problems are trained usually using gradient based methods and the *backpropagation* algorithm [10]. In this process, all the data samples feed to the network in batches (or mini-batches) and the value of network parameters such as weights and biases are adjusted based on the gradient of the loss function. The goal of training in neural network is to find the parameters that decrease the difference between predicted and actual labels, and thus minimize the loss. Training the neural networks is divided into two phases:

- **Forward Propagation**: In forward propagation, once all training samples in the batch are feed to the network and the output of the network is computed based on the existing network parameters (i.e., weights and biases). In each forward pass, the outputs of individual layers in the network are also stored for later use. Lastly, the output of the network is used to calculate the loss function. Then, the back propagation is performed for adjusting the network parameters using the loss function.

- **Back Propagation**: In this phase, the derivative of the loss with respect to the parameters of each layer in the network is calculated using the chain rule of derivatives. While doing so, the stored forward pass values are also used for efficient calculation of the gradient values with respect to each network layer. Once the gradients are calculated, variant of the gradient descent algorithm is used to update network parameters to minimize the loss. Equation (9) shows how to calculate the new value of each weight. In this formulate, η is a learning rate that indicates the speed of updating the network parameters using gradient and $\pounds\,(W)$ is the lost function of the parameters.

$$w_{new} = w_{old} - \eta . \bigtriangledown_w \pounds\,(W) \tag{9}$$

2.2 *Theory of Probability*

The probabilities are used to represent the uncertainty in the outcomes of stochastic events. The probability theory provides a powerful framework for building machine learning models and interpreting their predictions. In this section, we cover some basic concepts which are related to our approach.

2.2.1 Random Variables

Random variables play significant role in the theory of probability. They act like a function that maps the outcome of a probabilistic event (e.g., having head or tail when flipping a coin) to the real values. A random variable can be discrete or continuous. The outcome of the flipping a coin is a discrete random variable X, which can be represented as integers 0 (head) and 1 (tail). The probabilities for this random variable X is represented as $P(X = 0)$ and $P(X = 1)$ such that $P(X = 0) + P(X = 1) = 1$. As a short-hand notation, we usually use $P(A)$ instead of $P(X = 1)$, where A represents the random variable $X = 1$. Similarly, $P(X = 0)$ is referred to as $P(\neg A) = 1 - P(A)$.

2.2.2 Conditional Probability

The conditional probability of an event A given B is referred to as $P(A|B)$ and represents the probability that the event A will occur given the information that the event B has already occurred. For example, suppose the probability of raining in a usual day is $P(rain) = 0.4$. Now, suppose you picked the random day and you are given extra information that the weather is cloudy, so our belief about the probability of having a rainy day is maintained as $P(rain \mid cloudy)$, which is called conditional probability. The value of this probability is greater than the probability of raining without knowing that it is cloudy (i.e., the prior probability of raining).

For two independent variables A and B, their conditional probability are equal to their prior probabilities, i.e., $P(A|B) = P(A)$ and $P(B|A) = P(B)$. The probability of two event to occur at the same time is called joint probability and represented as $P(A, B)$. We have the following relationships between probabilities according to Equations (10) and (11) respectively.

$$P\left(A \mid B\right) = \frac{P\left(A, B\right)}{P\left(B\right)} \tag{10}$$

$$P\left(A, B\right) = P\left(A\right) P\left(B|A\right) \tag{11}$$

2.2.3 Bayes Theorem

Bayes theorem is known as a tool for computing the conditional probability [11]. Given hypothesis H and event E, the Bayes theorem gives the information about the relation between the probability of hypothesis H before and after getting the evidence E. The Bayes theorem for hypothesis H and event E is shown in Equation (12).

$$P\left(H|E\right) = \frac{P\left(E|\,H\right)P\left(H\right)}{P\left(E\right)} \tag{12}$$

Let us assume that H and E represents taking an umbrella and weather situation, respectively. Then, $P(H\big|E)$ can be calculated using the Bayes theorem in Equation (12), where $P(H)$ represented the prior, which is equal to the probability of taking an umbrella regardless of an information about the weather; $P(H\big|E)$ represents the posterior, which is the probability of hypothesis H after observing event E; $P(E\big|H)$ is called likelihood and represents the conditional probability of the event E given H; and $P(E)$ is the marginal probability of E.

2.2.4 Probability Distributions

A probability distribution is a function that maps all the possible outcomes for a random variable to their likelihood values. The probability distribution for discrete random variables are called probability mass function (PMF) and the probability distribution of continues random variables are called probability density function (PDF).

2.2.5 Maximum Likelihood Estimation and Maximum A Posteriori

Maximum likelihood estimation is used to find the parameters of the model that maximizes the likelihood of data. In order to compute the maximum likelihood estimation, first we have to define a probability distribution for the modeled data therefore, we assume our data generated by Gaussian distribution which is formulated as Equation (13). Gaussian distribution specifies by two parameters mean μ and variance σ. The probability density of observing a data x that follows Gaussian distribution is given by Equation

$$P\left(x;\mu,\sigma\right)=\frac{1}{\sigma\sqrt{2\pi}}exp\left(-\frac{(x-\mu)^2)}{2\sigma^2}\right) \tag{13}$$

In order to computing the maximum likelihood estimation we need to find the good estimate for both the parameters of Gaussian distribution according to Equation (14)

$$\begin{aligned}\mu_{MLE}&=\arg\max_{\mu} N\left(x\mid\mu,\sigma\right)\\ \sigma_{MLE}&=\arg\max_{\sigma} N\left(x\mid\mu,\sigma\right)\end{aligned} \tag{14}$$

Since solving the above equations is difficult, we use log likelihood function as our evaluation. For a data set X we assume $\theta=[\mu,\sigma]$. Let us assume that the data has n samples and is represented as $\boldsymbol{X}=\{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n\}$. Then, the model parameters θ are computed using MLE according to Equation (15). In MLE, we do not take into account the prior probability of model parameters.

$$\begin{aligned}\theta_{MLE}&=\arg\max_{\theta}\left[\log\Big(P(\boldsymbol{X}\mid\theta)\Big)\right]\\ &=\arg\max_{\theta}\left[\log\Big(\prod_{i=1}^{n}P(\boldsymbol{x}_i\mid\theta)\Big)\right]\\ &=\arg\max_{\theta}\left[\sum_{i=1}^{n}\log P(\boldsymbol{x}_i\mid\theta)\right]\end{aligned} \tag{15}$$

On the other hand, this information is important and should be taken into account in some situations. We can maximize the posterior probability of model parameters by taking into account both the likelihood and the prior for the model parameters. This techniques is called maximum a posterior (MAP) and defined formally as Equation (16)

$$\begin{aligned}\theta_{MAP}&=\arg\max_{\theta}\left[\log\Big(P(X\mid\theta)P(\theta)\Big)\right]\\ &=\arg\max_{\theta}\left[\log\Big(P(\theta)\prod_{i=1}^{n}P(\boldsymbol{x}_i\mid\theta)\Big)\right]\\ &=\arg\max_{\theta}\left[\log P(\theta)+\sum_{i=1}^{n}\log P(\boldsymbol{x}_i\mid\theta)\right]\end{aligned} \tag{16}$$

2.2.6 Dirichlet Distribution

The Dirichlet distribution is the conjugate prior of the categorical and multinomial distributions. It is a probability density function (pdf) for possible values of the probability mass function (pmf), $\boldsymbol{\pi} = [\pi_1, \ldots, \pi_K]$ over K categories. It is characterized by following parameters: $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_K]$ and is calculated as specified in Equation (17) where $\mathcal{S}_K$ is the K-dimensional unit simplex and $B(\boldsymbol{\alpha})$ is the K-dimensional multinomial beta function [12].

$$\text{Dirichlet}(\boldsymbol{\pi}|\boldsymbol{\alpha}) = \begin{cases} \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{K} \pi_i^{\alpha_i - 1} & \text{for } \boldsymbol{\pi} \in \mathcal{S}_K, \\ 0 & \text{otherwise,} \end{cases} \tag{17}$$

A Dirichlet distribution can be used to model the probability density of categorical distributions, which can be interpreted as probability distributions for assigning a sample to one of K categories as in the classification problems. Figure 5 demonstrates Dirichlet distributions over three categories. In this case, each Dirichlet distribution has three parameters ($K = 3$), i.e., one parameter for each category. When all parameters are one (i.e., $\boldsymbol{\alpha} = [1, 1, 1]$), the Dirichlet distribution is uniform, which means that all categorical distributions over these three categories are equally likely.

The parameters of a Dirichlet distribution are considered as real-valued pseudocounts. The parameters of the uniform Dirichlet distribution is usually taken as the prior counts $\boldsymbol{\beta}$ to which observations or evidences for the training data is added. The resulting parameters (pseudocounts) define the updated (posterior) Dirichlet distribution. In a classification problem, for each input $\boldsymbol{x}$ (e.g., an image), we want to predict a latent evidence vector $\boldsymbol{c}$, which is used to calculate the Dirichlet distribution with parameters $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_K]$, where $\alpha_i = c_i + \beta_i$. This distribution represents the probability density of all possible categorical distributions $\boldsymbol{\pi} = [\pi_1, \ldots, \pi_K]$ over the predefined K categories for the sample. Let $[1, 4, 14]$ be the evidence (e.g., observations) to be added to the prior counts $\boldsymbol{\beta} = [1, 1, 1]$, then the posterior Dirichlet distribution will have the parameters $\boldsymbol{\alpha} = [2, 5, 15]$, which indicates that categorical distributions placing more mass on the third category are more likely than others, as shown in Figure 5(b). Similarly, if the evidence vector is $[9, 9, 9]$, the resulting Dirichlet distribution parameters become $\boldsymbol{\alpha} = [10, 10, 10]$, which indicates that the categorical distributions placing similar amount of mass on all categories become more likely, as shown in 5(c).

According to the Figure 5 (a), alpha with small value α=[1,1,1] forms the symmetrical distribution. Dirichlet distribution with all parameters 1 also called uniform distribution, which means it assigns equal masses to the all samples. As the values in alpha vector increase symmetrically see Figure 5(b) Dirichlet puts its mass on the center of the simplex.

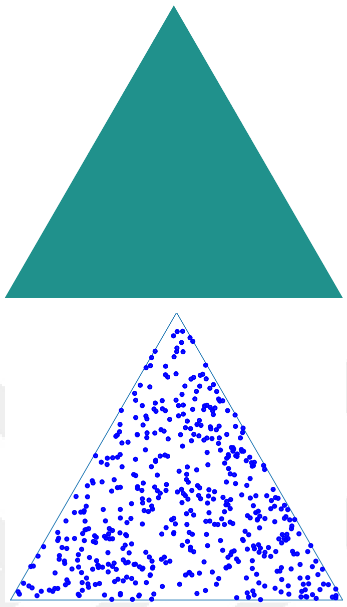
For alpha parameter with asymmetric values, Dirichlet assigns high mass to the highest parameter. According to Figure 5(c), as the value of third element is more than others Dirichlet puts it mass on top right side of simplex.

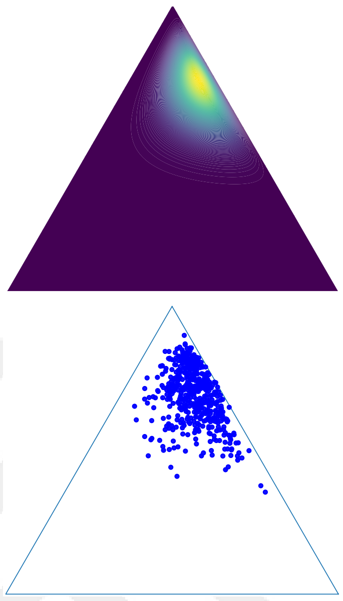
The mean and variance of the Dirichlet distribution for π_k, i.e., the probability of the category k, are computed as Equation (18)

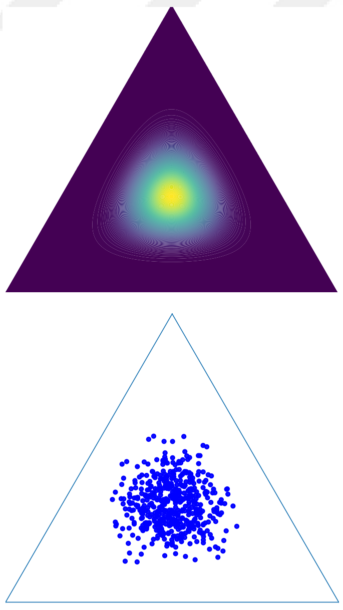
$$\hat{\pi}_k = \frac{\alpha_k}{\sum_{i=1}^{K} \alpha_i}$$
$$Var(\pi_k) = \frac{\alpha_k(\alpha_0 - \alpha_k)}{\alpha_0^2(\alpha_0 + 1)} \quad \text{where} \quad \alpha_0 = \sum_{i=1}^{K} \alpha_i. \tag{18}$$

The mean of the posterior Dirichlet distribution for a sample can be used as the predictive categorical distribution for classification tasks and its variance can be used to quantify the uncertainty of the prediction. For example, the predictive categorical distribution for the Dirichlet distributions with parameters $\boldsymbol{\alpha} = [1, 1, 1]$ and $\boldsymbol{\alpha} = [10, 10, 10]$ are both $[1/3, 1/3, 1/3]$, which is the uniform categorical distribution and has the maximum entropy. On the other hand, the variance of the latter is much smaller than that of the former. The uniform Dirichlet distribution with parameters $\boldsymbol{\alpha} = [1, 1, 1]$ represents the case that we are totally uncertain about the classification of the input while the predictive uncertainty decreases as the sum of the pseudocounts increases.

In Subjective Logic [13], given a Dirichlet distribution over possible categories, the uncertainty of predictive distribution over these categories are calculated as $u = K/\sum_{i=1}^{K} \alpha_i$.


(a) $\boldsymbol{\alpha} = [1, 1, 1]$


(b) $\boldsymbol{\alpha} = [2, 5, 15]$


(c) $\boldsymbol{\alpha} = [10, 10, 10]$

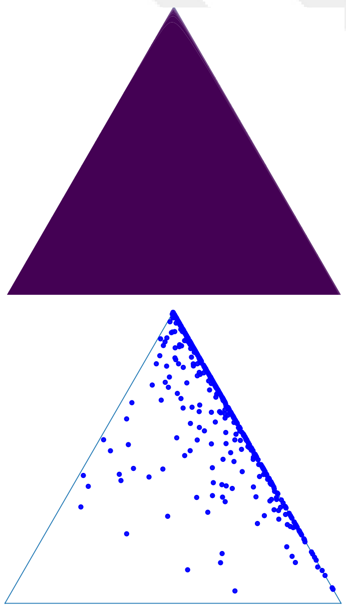

(d) $\boldsymbol{\alpha} = [0.1, 0.9, 2]$

Figure 5: At the top, density plots (blue = low, red = high) for the Dirichlet distributions over the probability simplex in $\mathbb{R}^3$ for various values of the $\boldsymbol{\alpha}$ parameters and, at the bottom, 500 categorical distributions sampled from each of these Dirichlet distributions.

Similar to the variance of a Dirichlet distribution, this uncertainty metric for a Dirichlet distribution is inversely proportional to the sum of Dirichlet parameters, i.e., α_i. On the other hand, one nice property of this uncertainty metric is its ease of interpretation, since it takes values between zero and one. For $\boldsymbol{\alpha} = [1, 1, 1]$, the uncertainty is calculated as 1.0; however, it decreases to 0.1 for $\boldsymbol{\alpha} = [10, 10, 10]$. Hence, while the corresponding Dirichlet distributions have the same mean (i.e., the uniform categorical distribution), they have significantly different level of uncertainty, which is also evident in Figure 5 (a) and (c).

Having Dirichlet distributions as an output of a classifier instead of a single softmax output, we can exploit uncertainty of the predictive categorical distribution to avoid making possibly wrong decisions based on vague predictions.

CHAPTER III

RISK-CALIBRATED CLASSIFIERS

In this section first, we discuss the problem statement and the classification model we used. Then we describe our approach in detail using concepts from the probability theory and neural networks.

3.1 Problem Statement

In this thesis, all matrices and vectors are represented in bold face such as $\boldsymbol{R}$ and $\boldsymbol{x}$, where the k^{th} row of the matrix $\boldsymbol{R}$ and k^{th} element of the vector $\boldsymbol{x}$ are given by $\boldsymbol{R}_k$ and x_k, respectively.

We consider a setting where an agent has a task of making a classification decision for a given sample, e.g., An image of an object, by assigning it into one of K disjoint categories that is known as a multi-class classification problem. For this purpose, it trains a neural network in a supervised way using pairs $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \in D$, where $\boldsymbol{x}$ and $\boldsymbol{y}$ represent a training sample and one-hot encoding[1] of its true category, respectively. The agent's classification decision for a given sample may involve some cost if it is wrong. This cost is referred to as *risk* in the rest of the thesis. The risk of misclassification may be task-specific and agent-specific. It is encoded compactly into an asymmetric non-negative square risk matrix, which is referred to as $\boldsymbol{R} \in [0, \infty)^{K \times K}$. The k^{th} row of this matrix is the vector $\boldsymbol{R}_k$, whose each element R_{ki} indicates the cost of classifying a sample from the category k to the category i. The risk of correct classification is zero, i.e., $R_{kk} = 0$ for any k. The agent wants to minimize the overall risk of its classification decisions by properly learning how to classify samples given its risk matrix.

[1]One-hot encoding of the category k is a vector $\boldsymbol{y}$ s.t. $y_k = 1$ and $y_i = 0$ for all $i \neq k$.

In neural networks, the *softmax* function is frequently used to compute a predictive categorical distribution over possible categories for an input sample. Since Dirichlet distribution is prior for categorical distribution, it can be used as a distribution over all possible softmax outputs for the classification of a given sample. This allows us to represent uncertainty of predictions for the classification of a sample through the variance of the corresponding Dirichlet distribution as motivated by Example 1.

Example 1*:* For the sake of simplicity for explaining the relation between the Dirichlet distributions and labels of samples in a data set, let us formalize the process of labeling a sample into one of three categories (red, green, and blue) as drawing balls with these colors from an urn. If we know exact numbers of balls with each color (n_r, n_g, and n_b) or their ratios, we can precisely compute the categorical distribution $\boldsymbol{\pi} = [\pi_r, \pi_g, \pi_b]$ for assigning the sample into one of these categories, i.e., $\pi_r = n_r/(n_r + n_g + n_b)$. However, if we do not know it, we cannot know exactly what $\boldsymbol{\pi}$ is, but we can still model its density $p(\boldsymbol{\pi})$ as a Dirichlet distribution. When we do not know any thing about n_r, n_g, and n_b, then any $\boldsymbol{\pi}$ is equally likely. In this case, $p(\boldsymbol{\pi})$ is the uniform Dirichlet distribution, which has high variance (high uncertainty about $\boldsymbol{\pi}$); however, when we exactly know these numbers or their ratio, $p(\boldsymbol{\pi})$ is a Dirichlet distribution with zero variance, i.e., it places all probability mass on a single $\boldsymbol{\pi}$ value, so we have zero uncertainty about $\boldsymbol{\pi}$.

3.2 Learning to Predict Pseudocounts

In this work, we consider classification tasks with fixed K categories and predict a Dirichlet distribution for each sample. The predicted Dirichlet distribution represents both the predictive categorical distribution and its uncertainty in a principled way.

We formally describe our approach using a generative model, which is demonstrated as a plate diagram in Figure 6. This model indicates that each sample is generated by drawing $\boldsymbol{c}$ from a latent prior distribution parametrized by $\boldsymbol{\nu}$. We call $\boldsymbol{c}$ as an evidence vector, where each element $0 < c_i < \infty$ represents evidence for the corresponding category i. Let us consider a binary classification task with two categories: *cat* and *dog*. Then, the evidence for the cat and dog correspond to how much the sample will look like a cat and dog, respectively. If the evidence is much higher for the cat category, the sample would look like a cat rather than a dog. On the other hand, if the total evidence is zero, the generated sample looks like neither a cat nor a dog. The label of the generated sample, i.e., $y \in \{1, \ldots, K\}$, is drawn from a latent categorical distribution $\boldsymbol{\pi} = [\pi_1, \ldots, \pi_K]$, which is defined by a Dirichlet distribution parametrized by $\boldsymbol{\beta}$ and $\boldsymbol{c}$, where $\boldsymbol{\beta}$ represents the prior counts and is updated with $\boldsymbol{c}$ to have parameters of the Dirichlet distribution for $\boldsymbol{\pi}$. Let us note that, π_k represents the probability that the sample has label k.

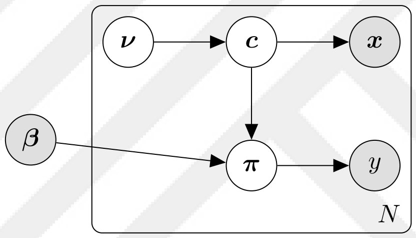


Figure 6: A graphical representation of a model for generative evidential networks using plate notation.

The prior distribution of the evidence can be formalized as Equation (19) unnormalized exponential distribution [14].

$$p(\boldsymbol{c}|\boldsymbol{\nu}) \propto exp\Big(-\sum_i \nu_i c_i\Big), \text{ where } \forall i \;\; \nu_i > 0. \tag{19}$$

Given the evidence vector $\boldsymbol{c}$ and the prior counts $\boldsymbol{\beta} = [1, \ldots, 1]$, the Dirichlet distribution for the categorical distribution $\boldsymbol{\pi}$ computes as Equation (20)

$$p(\boldsymbol{\pi}|\boldsymbol{c}, \boldsymbol{\beta}) = \text{Dirichlet}(\boldsymbol{\pi}|[c_1 + 1, \ldots, c_K + 1]). \tag{20}$$

The cross-entropy is frequently used to train deep classifiers, whose output is an estimation of the categorical distribution $\boldsymbol{\pi}$. In this thesis, we consider a classifier predicting a distribution for $\boldsymbol{\pi}$, i.e., $p(\boldsymbol{\pi}|\boldsymbol{c}, \boldsymbol{\beta})$, instead of its point estimate using the softmax function. However, given the predicted Dirichlet distribution, we can calculate the expected *cross-entropy* for category k as Equation (21)

$$\begin{aligned}\mathbb{E}_{\boldsymbol{\pi}\sim p(\boldsymbol{\pi}|\boldsymbol{c},\boldsymbol{\beta})}[-log(p(y=k|\boldsymbol{\pi}))] &= \int -log(\pi_k)p(\boldsymbol{\pi}|\boldsymbol{c},\boldsymbol{\beta})d\boldsymbol{\pi} \\ &= \psi(K+\sum_{i=1}^{K} c_i) - \psi(1+c_k),\end{aligned} \tag{21}$$

where ψ is the digamma function. In this work, we aim to predict a Dirichlet distribution for each sample $\boldsymbol{x}$ using a neural network estimating the evidence for the sample. The predicted distribution is used as $p(\boldsymbol{\pi}|\boldsymbol{c},\boldsymbol{\beta})$ and written as Equation (22)

$$q_\theta(\boldsymbol{\pi}|\boldsymbol{x}) = \text{Dirichlet}(\boldsymbol{\pi}|\boldsymbol{g}_\theta(\boldsymbol{x}) + \mathbf{1}) \tag{22}$$

where $\mathbf{1} = [1,\ldots,1]$ is a vector for prior counts $\boldsymbol{\beta}$ and $\boldsymbol{g}_\theta(\cdot)$ is a neural network parameterised by θ, which takes a sample $\boldsymbol{x}$ as input and returns an evidence vector $\boldsymbol{c} = [c_1,\ldots,c_K]$, where $c_i = g_{\theta i}(\boldsymbol{x})$.

Let $\boldsymbol{f}_\theta(\cdot)$ be the output of the penultimate layer (i.e., logits layer) of any neural network for classification with arbitrary architecture. Instead of applying softmax activation function to $\boldsymbol{f}_\theta(\boldsymbol{x})$ for predicting a single categorical distribution for the sample $\boldsymbol{x}$, we propose to apply another activation function $a(\cdot)$ to calculate $\boldsymbol{g}_\theta(\boldsymbol{x}) = a(\boldsymbol{f}_\theta(\boldsymbol{x}))$, which will then be interpreted as evidence to update the prior counts. For any input $-\infty < z < \infty$, this activation function should take values $0 < a(z) < \infty$. We used the *exponential* function, i.e., $a(z) = e^z$; however, others such as the *softplus* function $log(e^z + 1)$ can also be used. For any input sample $\boldsymbol{x}$, having total predicted evidence very close zero implies that the resulting distribution $q_\theta(\boldsymbol{\pi}|\boldsymbol{x})$ is very similar to the uniform Dirichlet distribution, so we have a very *uncertain* predictive categorical distribution for the sample.

We can use the maximum likelihood estimation (MLE) to train the neural network to predict an evidence vector for each sample $\boldsymbol{x}$. For this purpose, we select a base loss function $\mathcal{L}(\boldsymbol{x}, y|\boldsymbol{\pi})$ conditioned on the latent categorical distribution; then, we calculate the overall loss by aggregating individual sample losses and integrating out the latent $\boldsymbol{\pi}$ values using the predicted Dirichlet distribution. Let D_k represent an empirical distribution for the samples from the category k in the training data. If the *cross-entropy* loss is used as the base loss, we have the following loss function:

$$\mathcal{L}(\theta) = \sum_{k=1}^{K} \Big(\sum_{\boldsymbol{x} \in D_k} \Big[\psi(K + \sum_{i=1}^{K} g_{\theta i}(\boldsymbol{x})) - \psi(1 + g_{\theta k}(\boldsymbol{x})) \Big] \Big) \tag{23}$$

During training, to encourage the neural network to keep the evidence for the correct category as much as possible while minimizing the evidence for the wrong categories, we use the Kullback–Leibler (KL) divergence[2] between the predicted Dirichlet distribution and a desired Dirichlet distribution. For a sample $\boldsymbol{x}$ and its one-hot label $\boldsymbol{y}$, this regularization term is written as $\mathrm{KL}[p(\boldsymbol{\pi}|\boldsymbol{g_\theta}(\boldsymbol{x}) + \mathbf{1}) \| p(\boldsymbol{\pi}|\boldsymbol{g_\theta}(\boldsymbol{x}) \odot \boldsymbol{y} + \mathbf{1})]$, where $\odot$ represents element-wise product, and $\boldsymbol{g_\theta}(\boldsymbol{x}) \odot \boldsymbol{y} + \mathbf{1}$ is the parameters of the desired distribution. The desired Dirichlet distribution agrees with the predicted Dirichlet distribution for the correct category, but it places zero evidence in all other categories.

3.3 Decision Making Under Uncertainty

In decision theory, the notion of pignistic probabilities is introduced to represent the probability that a rational agent will assign to an option when required to make a decision. The pignistic probabilities $\boldsymbol{p} = [p_1, \ldots, p_K]$ are mathematically equivalent to the Shapley value in game theory [15] and inherently incorporate the decision maker's uncertainty for choosing one of K options (i.e., the uncertainty related to $\boldsymbol{\pi}$) and the incurred cost of choosing each one. Hence, while calculating the expected risk of choosing one category as the label of a sample, the pignistic probabilities ($\boldsymbol{p}$) replace the categorical probabilities ($\boldsymbol{\pi}$) to account for the risk of misclassification. In the settings where there is no cost for misclassification or each misclassification has the same cost, $\boldsymbol{p}$ should be equal to $\boldsymbol{\pi}$. However, in our setting, we have an explicit risk matrix, which may lead some divergence between these two probabilities.

We also model the pignistic probabilities for the classification decision for the sample $\boldsymbol{x}$

[2] $\mathrm{KL}[p(\boldsymbol{\pi}|\boldsymbol{\alpha}) \| p(\boldsymbol{\pi}|\boldsymbol{\beta})] = \sum_{i=1}^{K} \Big[(\alpha_i - \beta_i) \Big(\psi(\alpha_i) - \psi(\alpha_0) \Big) \Big] - log\big(\frac{B(\boldsymbol{\alpha})}{B(\boldsymbol{\beta})}\big)$

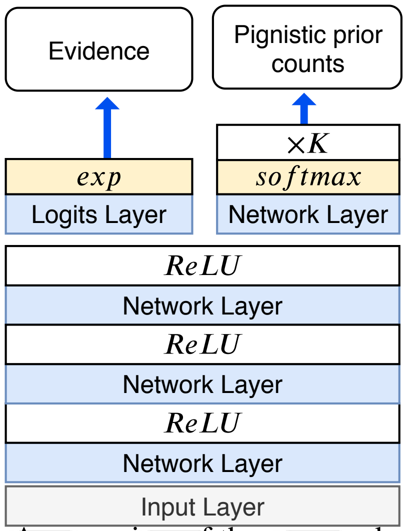


Figure 7: An overview of the proposed approach.

as a Dirichlet distribution according to Equation (24)

$$q_\theta(\boldsymbol{p}|\boldsymbol{x}) = \text{Dirichlet}(\boldsymbol{p}|\boldsymbol{g}_\theta(\boldsymbol{x}) + \boldsymbol{\gamma}_\theta(\boldsymbol{x})) \tag{24}$$

where $\boldsymbol{\gamma}_\theta(\boldsymbol{x})$ is basically the redistribution of uniform prior counts over categories and calculated as $\boldsymbol{\gamma}_\theta(\boldsymbol{x}) = K\text{softmax}(\boldsymbol{W}\boldsymbol{f}'_\theta(\boldsymbol{x}) + \boldsymbol{b})$, where $\boldsymbol{W}$ and $\boldsymbol{b}$ are the additional weight and bias variables, respectively, to calculate $\boldsymbol{\gamma}_\theta(\boldsymbol{x})$ based on $\boldsymbol{f}'_\theta(\boldsymbol{x})$, which represents the input of the *logits layer* of the neural network. Here, $\boldsymbol{\gamma}_\theta(\boldsymbol{x})$ serves as prior counts for the pignistic probabilities for the sample. We avoid setting it to some fixed constant manually for all samples, since the risk of misclassification may change from sample to sample (e.g., based on its true category). By doing so, we aim neural network to learn the less risky categories and increase its prior count through the softmax output. Let us note that we still have $\sum_i \gamma_{\theta i}(\boldsymbol{x}) = K$. Hence, the total prior counts for the pignistic probabilities $\boldsymbol{p}$ is same as that of the categorical probabilities $\boldsymbol{\pi}$. This allows us to keep the evidence predicted by the network for the categorical probabilities (i.e., $\boldsymbol{g}_\theta(\boldsymbol{x})$) to be commensurate with the prior counts for the pignistic probabilities. Figure 7 presents an overview of our approach.

Figure 5 (d) shows the resulting Dirichlet distribution after redistributing the counts in the uniform Dirichlet distribution in Figure 5 (a), using $[0.033, 0.3, 0.667]$ as the softmax output to calculate $\boldsymbol{\gamma}_\theta(\boldsymbol{x})$. The resulting Dirichlet distribution generates almost no probability mass for the first category while placing more probability mass on the third category. This is a desired prior for the pignistic probabilities of a sample if the misclassification cost for the sample is inversely proportional to the redistributed counts.

Given its pignistic probabilities, the average risk of misclassifying a sample from the category k can be calculated as $risk(\boldsymbol{x}) = \sum_{i=1}^{K} R_{ki} p_i$. We can integrate out the pignistic probabilities using their Dirichlet distribution parametrized by $\boldsymbol{\alpha}$ with $\alpha_i = g_{\theta i}(\boldsymbol{x}) + \gamma_{\theta i}(\boldsymbol{x})$, and calculate the expected risk as Equation (25)

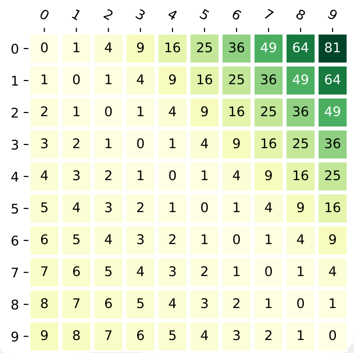
$$\begin{aligned}\mathbb{E}_{q_\theta(\boldsymbol{p}|\boldsymbol{x})}[risk(\boldsymbol{x})] &= \int \Big(\sum_{i=1}^{K} R_{ki} p_i\Big) \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{K} p_i^{\alpha_i - 1} d\boldsymbol{p} \\ &= \sum_{i=1}^{K} R_{ki} \hat{p}_i = \frac{\sum_{i=1}^{K} R_{ki} g_{\theta i}(\boldsymbol{x})}{K + \sum_{j=1}^{K} g_{\theta j}(\boldsymbol{x})} + \frac{\sum_{i=1}^{K} R_{ki} \gamma_{\theta i}(\boldsymbol{x})}{K + \sum_{j=1}^{K} g_{\theta j}(\boldsymbol{x})}\end{aligned} \tag{25}$$

Hence, to minimize the risk of misclassification and increase the uncertainty for the misclassified samples, we combine the loss with the expected risk of misclassification as Equation (26)

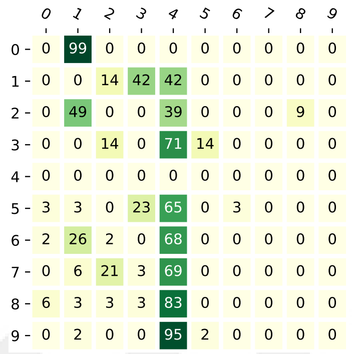
$$\begin{aligned}\mathcal{L}_{total}(\theta) = \mathcal{L}(\theta) + \sum_{\langle \boldsymbol{x}, \boldsymbol{y} \rangle \in D} \Big(&\mathrm{KL}[p(\boldsymbol{\pi} | \boldsymbol{g_\theta}(\boldsymbol{x}) + \mathbf{1}) \| p(\boldsymbol{\pi} | \boldsymbol{g_\theta}(\boldsymbol{x}) \odot \boldsymbol{y} + \mathbf{1})] \\ &+ \sum_{i=1}^{K} \Big[\underbrace{\kappa(R_{ki} + \epsilon) g_{\theta i}(\boldsymbol{x})}_{(1)} + \underbrace{\kappa R_{ki} \gamma_{\theta i}(\boldsymbol{x})}_{(2)}\Big]\Big)\end{aligned} \tag{26}$$

where ϵ is a very small positive number (e.g., $1\mathrm{e}{-6}$) and $\kappa \in [0, 1]$ is the weight of the expected risk in the loss. In this loss, we neglect the denominator of the expected risk to avoid neural network to produce high evidence for less risky categories when it cannot predict the correct category. Also, by excluding the denominator in the loss and including ϵ, the term (1) in Equation(26) acts as the negative logarithm of the prior for the evidence in Equation (19), where the prior distribution for the evidence has the parameters $\nu_i = \kappa(R_{ki} + \epsilon)$. During training, it regularizes the evidence generated for each sample and helps avoiding fictitious evidence for wrong categories. This is equivalent to learning evidence using maximum a posteriori (MAP), instead of maximum likelihood estimation. Furthermore, the term (2) in the loss serves as an objective to optimize prior counts for the pignistic probabilities, which influence the classification decisions only when the predictive uncertainty is high.

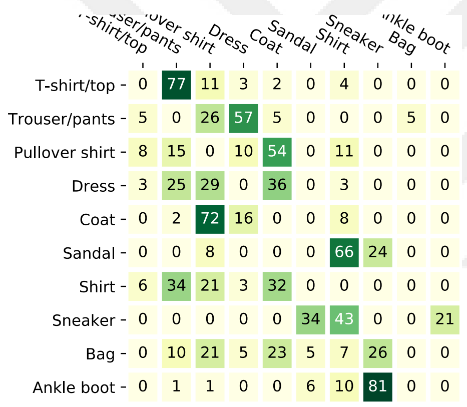
Let us note that the evidence regularization in this way is separate from the L_1 or L_2 regularization methods, which are used to regularize network parameters, not directly the output of the network. Through the redistributed prior counts of the pignistic probabilities, the loss function reduces the expected risk for misclassification when $\boldsymbol{\pi}$ is highly uncertain. If the categorical probabilities were used to calculate the risk of misclassification, the expected risk in the loss would encourage more evidence for the wrong categories with lower risk to reduce the expected risk when the correct category cannot be predicted correctly. However, the formulation of pignistic probabilities encourages zero evidence for the wrong categories if the redistributed prior counts reduce the expected risk sufficiently.



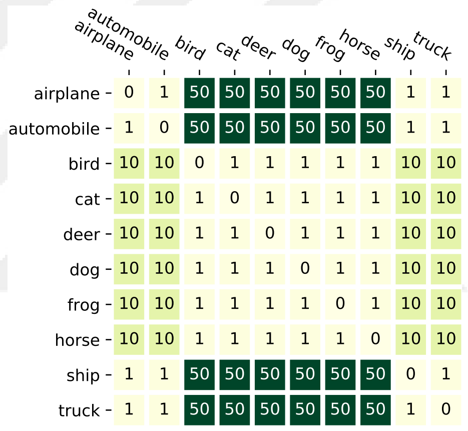
(a) MNIST risk mat.



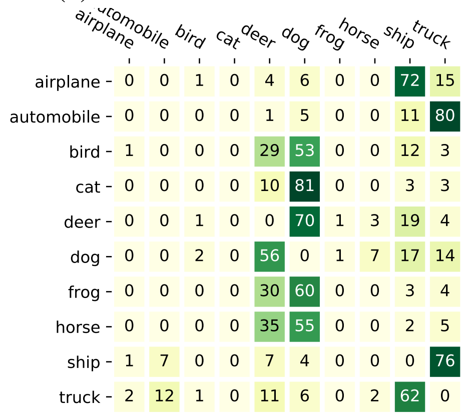
(b) MNIST misclass.



(c) FashionMNIST misclass



(d) CIFAR10 risk mat



(e) CIFAR10 misclas.

Figure 8: The risk matrices in the experiments and the percentage of misclassification for each category for the proposed approach.

CHAPTER IV

EVALUATION

In this section, we evaluate how agents may use our approach (i) to minimizing their cost when making classification decisions based on the predictions of deep neural networks and (ii) to determine if these predictions are wrong based on their uncertainty. For this purpose, we use the LeNet5 neural network architecture [16] with three well-known datasets: MNIST[1], FashionMNIST[2], and CIFAR10[3]. Figure 9 shows some sample images from the MNIST dataset. These images indicates that some digits may be very confusing and it is very likely that a classifier may classify a handwritten digit incorrectly.

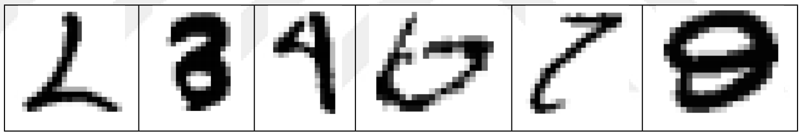


Figure 9: Some sample images from the MNIST data set, which are labelled as 2, 3, 4, 6, 7, and 9, respectively.

In our evaluations, we choose LeNet5 as a neural network architecture, because it has recently been used to evaluate uncertainty quantification in deep neural networks [2, 17] and it is a simple convolutional neural network architecture, which cannot achieve the state-of-the-art performance in terms of accuracy. It fits very well to our purpose, since we do not aim to achieve high classification accuracy, but we desire to equip even simple neural networks with the ability of estimating their uncertainty when making mistakes and choosing less costly categories when they are uncertain. The configuration of the network architecture used for MNIST and FashionMNIST is presented in Table 1. For CIFAR10, we used the same architecture; however, we use 192 filters for Conv_1 and Conv_2, also used

[1]In MNIST, samples are 28×28 images of handwritten digits between $0 - 9$.
[2]FashionMNIST contains MNIST-like samples representing fashion products.
[3]In CIFAR10, samples are low-resolution images of 10 animal and vehicle categories.

Layer	**Filters/Neurons**	**Patch Size**	**Stride**	**Activation**
Conv$_1$	20	5×5	1	ReLU
Max Pool	-	2×2	2	-
Conv$_2$	50	5×5	1	ReLU
Max Pool	-	2×2	2	-
FC$_1$	500	-	-	ReLU
FC$_2$	10	-	-	-

Table 1: LeNet5 architecture for MNIST and FashionMNIST.

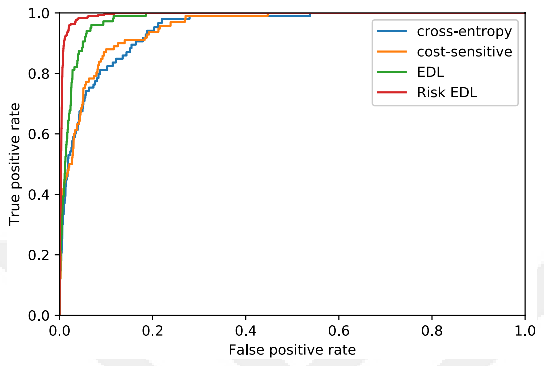
1000 neurons in FC$_1$ as described in [17]. We used L2 regularization with coefficient 0.005 in the fully-connected layers.

We compare our approach with three different approaches: the *standard* learning, which uses the *cross-entropy* loss; *cost-sensitive* learning [18], which incorporates the risk matrix into the standard loss to minimize the cost of misclassifications; and, evidential deep learning (EDL) [2], which uses the regularized loss in Equation (23). We name our model as *risk-calibrated evidential deep learning*, which is referred to as *Risk EDL*. It uses the expectation of the predicted Dirichlet distribution for pignistic probabilities (i.e., $q_\theta(\boldsymbol{p}|\boldsymbol{x})$) as the predictive distribution for classification. All these models uses the same LeNet5 architecture, but with different loss functions.

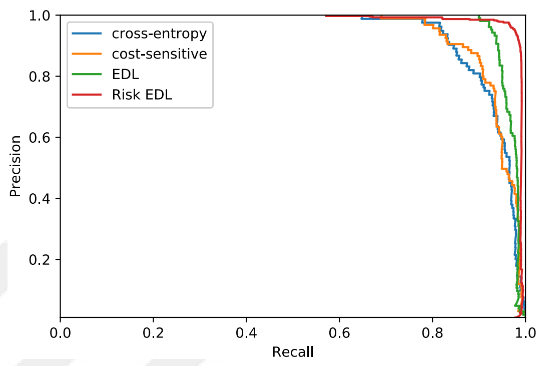
In our experiments, we used two different matrices for $\boldsymbol{R}$ as shown in Figure 8 (a) and (d), and set $\kappa = 1/max(\boldsymbol{R})$. For MNIST and FashionMNIST, we used the category indices to create the risk matrix. Let i refers to the true category index of a sample and j refers to the index of the predicted category for the sample. We set $R[i,j] = (i-j)^2$ if $j > i$; otherwise, it is set to $i - j$.

For MNIST, this risk matrix implies that overestimating the value of the digit in an image incurs much higher cost than under estimating it. For CIFAR10, we divided the categories into two groups: *animals* and *vehicles*. In this case, we set $R[i,j] = 1$ if i and j are from the same group. If the true index i is animal, but the predicted index j is not, we set $R[i,j] = 10$; otherwise, we set it to 50 as shown in Figure 8 (d).

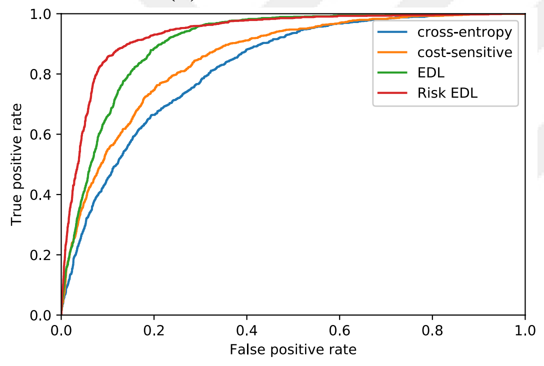
MNIST and FashionMNIST datasets contain 60 and 10 thousands training and test



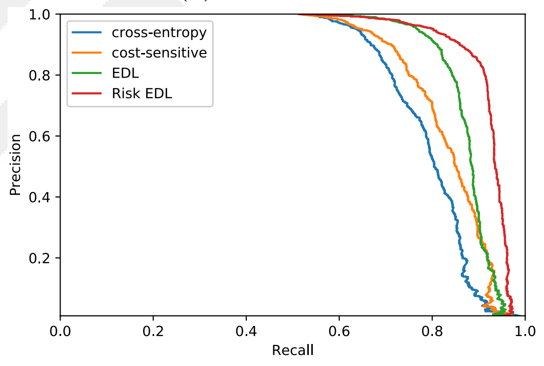
(a) MNIST ROC



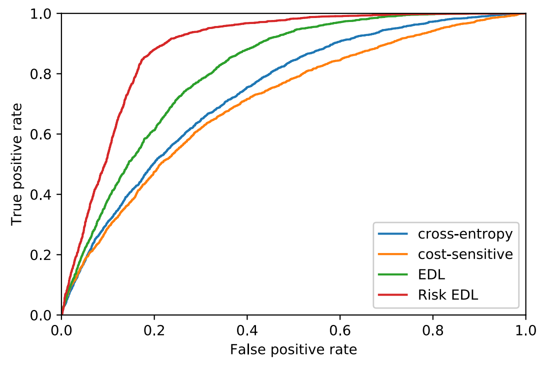
(b) MNIST PR



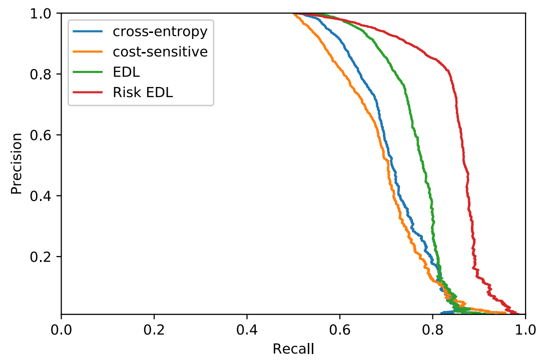
(c) FashionMNIST ROC



(d) FashionMNIST PR



(e) CIFAR10 ROC



(f) CIFAR10 PR

Figure 10: ROC and PR curves for misclassification detection.

Model	**MNIST**	**Fashion**	**CIFAR10**
Cross-entropy	11.16	8.42	5.8
Cost-sensitive	5.9	6.1	4.7
EDL	11.0	10.04	6.1
Risk EDL	3.44	3.48	3.54

Table 2: Prediction cost

Model	**MNIST**	**Fashion**	**CIFAR10**
Cross-entropy	99.0	90.0	73.0
Cost-sensitive	99.1	90.0	73.1
EDL	98.8	89.4	73.6
Risk EDL	99.3	91.3	75.0

Table 3: Test accuracy

samples, respectively, while the number of training and test samples are 50 and 10 thousands for CIFAR10 dataset. We train the models for 50 epochs and evaluated the trained models using the test samples. Table 2 and 3 present our results for the prediction cost and accuracy on the test samples. Our results indicates that cost-sensitive training achieves lower misclassification cost than standard or evidential deep learning. This is reasonable, since these methods do not consider the risk matrix during training. Our model achieves significantly lower misclassification cost than the cost-sensitive training, while also enhancing the accuracy of the model on the test samples in all datasets. That is, when compared with cost-sensitive training, our approach reduces the cost of misclassification by 42%, 43%, and 25%, for MNIST, FashionMNNIST, and CIFAR10, respectively. This is a very significant improvement.

However, an important problem that should be addressed is the detection of misclassified samples. If an agent can detect misleading classification predictions of a classifier (instead of blindly following these predictions), it can actively reduce the cost of misclassification by avoiding making classification decisions when possible. In the literature on uncertainty quantification for deep classifiers [17], the entropy[4] of a prediction is used as a proxy for its

[4]The entropy of a predicted categorical distribution $\boldsymbol{p}$ is defined as $\sum_{i=1}^{K} -p_i \log p_i$.

Model	MNIST	Fashion	CIFAR10
Cross-entropy	0.938	0.79	0.71
Cost-sensitive	0.942	0.83	0.69
EDL	0.967	0.87	0.76
Risk EDL	0.992	0.92	0.85

Table 4: Misclassification percentages for area under the PR curve

Model	MNIST	Fashion	CIFAR10
Cross-entropy	0.945	0.82	0.71
Cost-sensitive	0.945	0.85	0.71
EDL	0.979	0.90	0.81
Risk EDL	0.996	0.94	0.85

Table 5: Misclassification percentages for area under the ROC curve

uncertainty. We also evaluate different approaches using the entropy of their predictions and measure how well we can separate the misclassified and correctly classified samples using the entropy. Ideally, when it is unable to correctly classify a sample, a model should say *I do not know* by providing a very uncertain prediction (i.e., a prediction with maximum entropy). We used the area under the Precision-Recall (PR) and Receiver Operating Characteristic (ROC) curves[5] to measure the successful detection of misclassified samples using the entropy of predictions for these samples. Figure 10 shows PR and ROC curves for misclassification detection. The area under curve (AUC) for PR and ROC curves are presented in Table 4 and 5; they indicate that our approach allows much better misclassification detection and separation of misclassified samples from others for all datasets.

Figure 8 (b) and (c) demonstrate misclassification matrices of our approach for the MNIST and FashionMNIST datasets. The rows of this matrix represent the true category indices and columns represent indices of the predicted categories. The value at index $[i, j]$ represents what percentage of the misclassified samples from category i are predicted as belong to category j. Similarly, Figure 8 (e) represents the misclassification matrix for CIFAR10. These matrices indicate how our approach avoid classifying samples to high-risk

[5]ROC curves are not suitable for analysing imbalanced data. That is why we used oversampling to balance the data before computing the ROC curves.

Model	MNIST	Fashion	CIFAR10
Cross-entropy	0.975	0.77	0.76
Cost-sensitive	0.965	0.8	0.76
EDL	0.970	0.91	0.98
Risk EDL	0.993	0.97	0.98

Table 6: Area under the PR curve for out-of-distribution samples

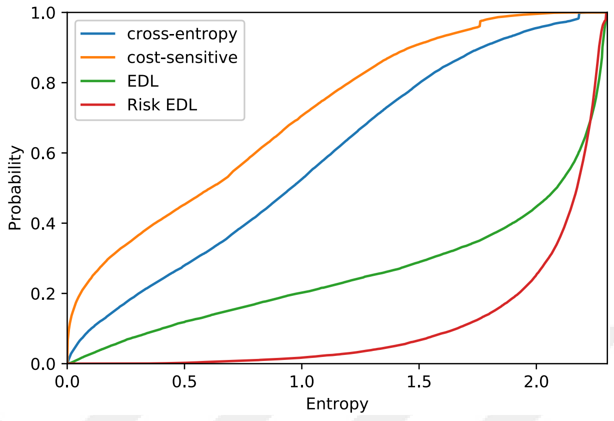
Model	MNIST	Fashion	CIFAR10
Cross-entropy	0.972	0.78	0.78
Cost-sensitive	0.961	0.81 / 0.81	0.78
EDL	0.97	0.89	0.99
Risk EDL	0.993	0.96	0.99

Table 7: Area under the ROC curve for out-of-distribution samples

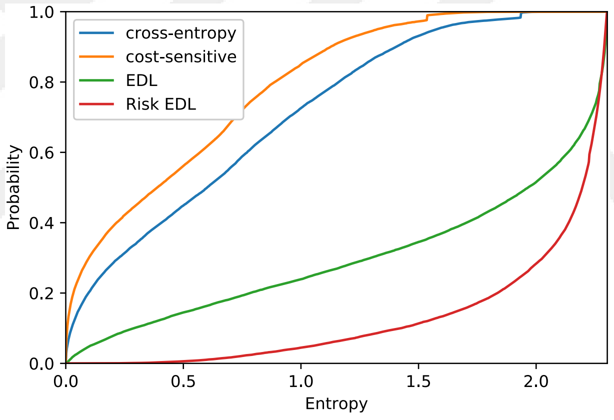
categories and explain lower cost of misclassification for our approach.

Recent studies indicate that deep classifiers are very vulnerable to out-of-distribution (OoD) samples [19]. In other words, the deep classifiers are usually very confident when they classify samples that are not from their training set distribution. Ideally, classifiers should give a very uncertain prediction in such settings. If the entropy is taken as a measure of uncertainty, their predictions should be a uniform probability distribution over possible categories, which indicates highest predictive uncertainty. To evaluate our approach for the predictive uncertainty for OoD samples, we use samples from the notMNIST and CIFAR100 datasets. The notMNIST dataset [17] contains images of letters from A to J on various typefaces. It is used to evaluate classifiers trained with MNIST and FashionMNIST datasets. CIFAR100 contains images similar to images from CIFAR10, but from 100 different categories. We used it to evaluate classifiers trained with CIFAR10 dataset. Figure 11 shows the empirical entropy CDF of the predictions for the OoD samples for each dataset. An ideal classifier should give the near maximum entropy predictions for OoD samples, so the entropy CDF curve should be very close to the bottom right corner. The figure indicates that our approach is closer to the ideal; the entropy of its predictions for OoD samples are much more closer to the maximum entropy than others.

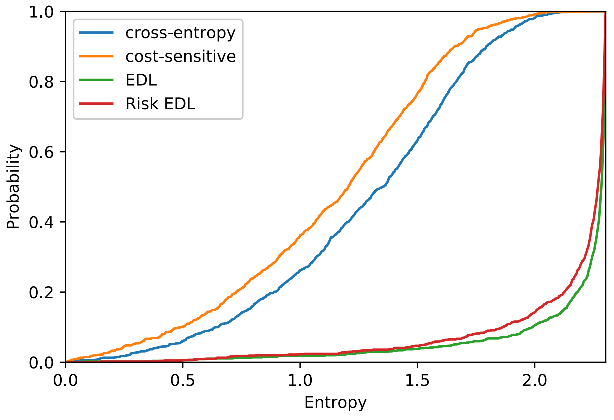
We also compare the uncertainty of in- and out-of-distribution samples to see how easy to



(a) MNIST



(b) FashionMNIST

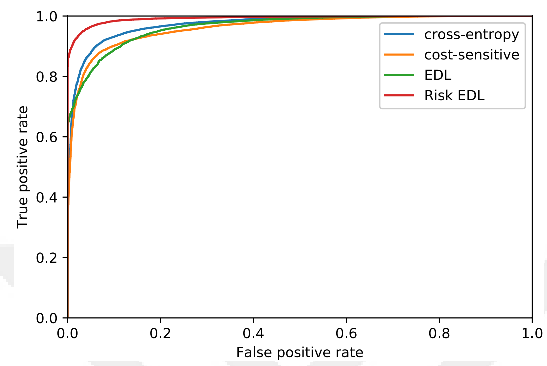


(c) CIFAR10

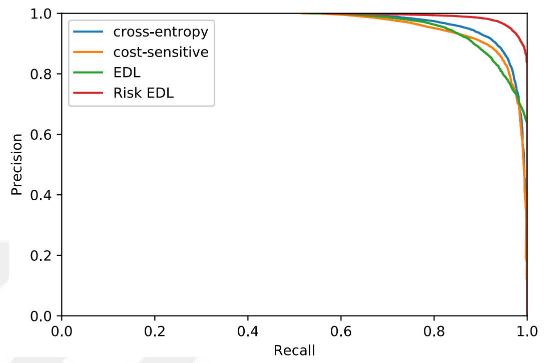
Figure 11: Empirical entropy CDFs of the predictions for the out-of-distributions samples.

separate these two using the predictive uncertainty. For this purpose, we use predictions for the correctly classified samples from the training distribution, i.e., in-distribution samples, and predictions for the OoD samples. We calculate their entropy and use the PR and ROC curves to see how easy to separate these two using only the entropy of predictions. Our results are shown in Figure 12, which clearly indicates that our approach allows much better separation between in-distribution and out-of-distribution samples using the predictive uncertainty. We also summarized our results also in Table 6 and 7 for all datasets.

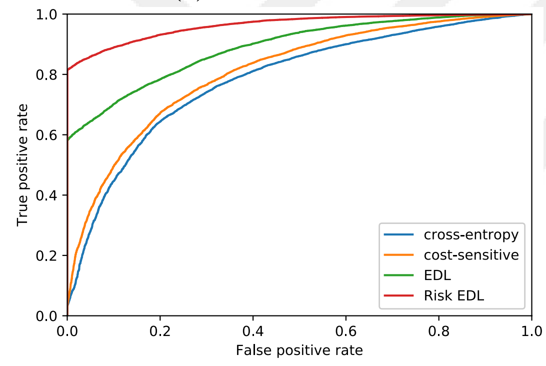
In this section, we carefully and comprehensively evaluated our approach and showed that (i) it allows an agent to minimize the cost of misclassification by learning and exploiting pignistic probability distributions, and (ii) it allows an agent to detect misclassified and out-of-distribution samples through predictive uncertainty.



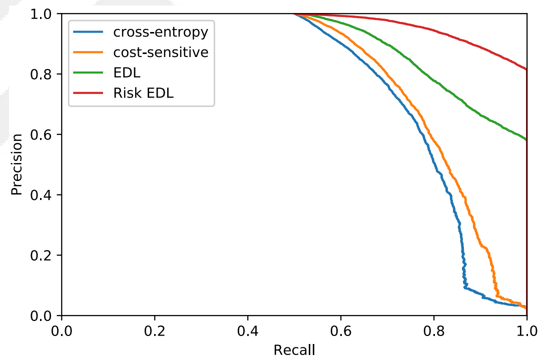
(a) MNIST ROC



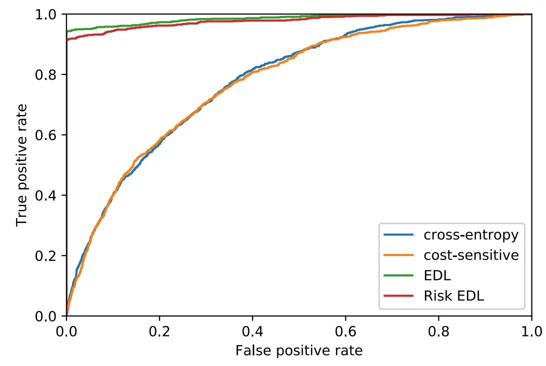
(b) MNIST PR



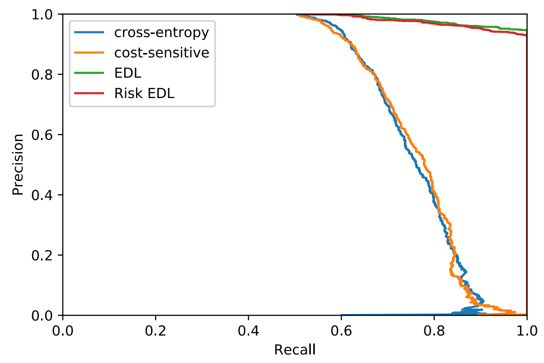
(c) FashionMNIST ROC



(d) FashionMNIST PR



(e) CIFAR10 ROC



(f) CIFAR10 PR

Figure 12: ROC and PR curves for out-of-distribution detection.

CHAPTER V

RELATED WORK

In this chapter, we comprehensively discuss related works in cost-sensitive learning and applied techniques in two different categories, such as (i) altering the distribution of training data, (ii) manipulating the learning algorithm.

Deep neural networks, in general, are becoming a fundamental component in intelligent systems. They are deployed to many fields, such as face and speech recognition, medical diagnosis and autonomous systems (e.g.,self-driving vehicles). Although these models can exceed human performances in some tasks, they can make mistakes in their predictions. The main problem with these models is that they consider the same cost for all the prediction errors where in real-life issues the cost of misclassifications are not same, and sometimes leads to irreversible consequences. For example, in medical diagnosis, if a patient with a cancer disease mistakenly is classified as a healthy person; thus, required treatment is delayed because of the wrong diagnosis, the patient may lose his life. Rather than standard classifiers that assign equal importance to all types of errors, cost-sensitive learning treats distinct classification errors differently.

Cost-sensitive learning algorithms incorporate with a cost matrix [1]. The cost matrix C is a square matrix here each element $C[i, j]$ of this matrix represents a cost of classifying a sample as class i, where actually it belongs to class j. The diagonal of this matrix is always zero, which means the sample is correctly classified. Cost matrix for a binary classification (i.e., having two class labels) is defined in Table 8. The rows in the cost matrix represent the predicted labels while the columns represent the actual labels in the classification task. Since the cost of the wrong prediction should be greater than the cost of the true prediction, it is expected to have $C[1, 0]$ >$C[0, 0]$ and $C[0, 1]$ >$C[1, 1]$.

	actual negative	actual positive
predict negative	C[0,0]	C[0,1]
predict positive	C[1,0]	C[1,1]

Table 8: A cost matrix representation for a binary classification.

The goal of the classifier is to make the predictions that minimizing the total expected cost calculated according to Equation (27). where $P\left(j \mid \boldsymbol{x}\right)$ denotes the probability of data instance x belongs to j, $C[i,j]$ denotes the cost of classifying a data instance x to class i and n is the number of classes. By aiming to minimize this cost, the classifier is inclined to make the wrong predictions with the lowest cost (less risky) when it is not possible to make the correct predictions.

$$L(\boldsymbol{x}, i) = \sum_{j=1}^{n} P\left(j \mid \boldsymbol{x}\right) C[i,j] \tag{27}$$

Cost sensitivity can also be applied to the normal error-based classifiers in two different ways: (i) such as altering the class distribution of training data [20, 21, 22, 23, 24, 18, 25, 26] (ii) altering the learning algorithm directly [27, 28, 29, 30, 31, 32, 33]. In following sections, we give brief explanation of different cost-sensitive methods that used in literature for both categories.

5.1 Cost-sensitive Learning by Altering the Distribution of Training Data

One of the approaches in cost-sensitive learning is changing the class distribution of the training data by applying methods such as meta-learning [34], re-balancing [1], re-weighting and re-sampling [20, 35, 36, 37, 38], expanding data space, and gradient boosting [35]. We briefly review proposed cost-sensitive methods separately in the following sections.

5.1.1 Re-sampling

In the real-world applications, most of the classifiers face with the *class imbalanced problem* where the class distribution is not balanced in the given dataset. It can influence the overall accuracy significantly. For instance, if the data set includes inadequate number of instances for a particular class label, a classifier may fail in predicting that class correctly. In order to increase the accuracy of the classifiers in such situations, changing the distribution of the data for generating more balanced distribution is required. One of the simple ways is using methods such as sampling and weighting.

Cheng *et al.* [39] proposed two approaches called Balanced Random Forest (BRF) and Weighted Random Forest (WRF) to deal with highly-skewed class distribution. The first method under-samples the majority class and over-sample the minority class in order to provide balanced distribution. The second method uses the idea of cost-sensitive learning, it over-samples the examples belong to the high cost classes and under-samples the examples belong to the low cost classes. Weighted random forest decreases the overall cost by improving the performance of minority class. Zadronzy *et al.* [20] point out that standard sampling techniques do not have any effect in dealing with class-imbalanced problem. They suggest cost-proportion reject sampling method. This method is able to draw samples from the given data distribution independently. Weiss *et al.*[26] tested the performance of the sampling methods and observed that for the class-imbalanced problems, incorporating the cost information into the learning algorithm directly, performs better than methods such as under-sampling and over-sampling. Charles [1] advocates that cost-sensitive classifiers should learn from balanced data set and be able to compute optimal decision based on probability estimation given by classifier. To achieve this idea, he changed the proportion of negative and positive examples using re-balancing technique.

Domingos [34] stated that over-sampling may increase the training time while under-sampling may decrease the number of data samples. It is claimed that these models are suitable for binary classification but would not work in multi-class cases, so they proposed a new technique called meta-cost. According to this technique, if we have information about the probability that the sample x belongs to any particular class then the Bayes optimal prediction is the class i that minimizes the conditional risk according to Equation (28).

$$R(\boldsymbol{i}|\boldsymbol{x}) = p(j|x)c(i,j) \tag{28}$$

Meta cost re-labels training data samples with their estimated minimum cost according to Equation (28). It builds a new training set by duplicating the training examples automatically and learns the classifier for each one to calculate the class probabilities as a fraction of votes that come from each classifier.

5.1.2 Weighting

One of the other techniques that used for balancing the data samples distribution is weighting, which is widely applied to the decision trees and Bayesian classifiers. In this technique [1, 20], different weights are assigned to training samples proportional to their misclassification costs. In iterative weighting [35], this process is repeated iteratively and weights are updated until they converge.

5.1.3 Re-labeling and Thresholding

Re-labeling and thresholding are known as the non-sampling techniques for changing the distribution of the training examples. Re-labeling technique is divided into the two categories such as re-labeling training data samples and re-labeling test data-samples. In this approach, the classes of instances are re-labeled by using minimum expected cost. Thresholding technique refers to the process of choosing the probability as a threshold and uses this threshold in future predictions.

Zhou *et al.* [25] use thresholding technique on their work. The best probability estimation of the training samples is considered as a unit threshold and overall misclassification is computed as a function of that threshold. The local minima of overall misclassification cost and threshold curve represent the lowest misclassification cost.

5.1.4 Adversarial Perturbations

Zhang [37] implemented the classifier that is robust against the adversarial perturbation by considering the harm of all adversarial transformation of each class. Adversarial perturbation is known as manipulating the pixels of training image samples in a way that the model considers it as a different example. In this approach, they encoded the results of each perturbation transformation into a cost matrix, where each element in the cost matrix represents the cost of the adversary that caused the class i misclassified as class j by perturbation. The value of the cost matrix depends on the adversarial transformation, where if the adversarial transformation is crucial, the value of the cost matrix becomes one; otherwise, it becomes zero. The generated cost matrix combines with the objective function in the training phase. In order to detect sample robustness, they consider the *l* norm bound for each example, where if there is no adversarial perturbation around the example in a specific radius, then that example said to be certified robust. They defined robust cost as the ratio of the sum of all adversarial examples to the total number of examples. Zhao [40] proposed a method called Cost-sensitive feature selection using L_1 and L_2 norm. L_1 norm, which is known as the least absolute error minimizing the sum of the absolute differences between the actual and predicted value. Respectively L_2 norm refers to the least square, and it minimizes the sum of squares of differences between actual and predicted value. L_1 and L_2 norms known as the regularizer that add to loss function in order to avoid over-fitting in machine learning models. Zhao defined the joint $L_{2,1}$ norm minimization of the loss function with misclassification cost. Misclassification cost for false negative and true positive denoted by C_{FN} and C_{FP} respectively and calculated as given Equation (29) where C represents the number of features, α represents the ratio of the number of minority classes to the number of samples, and β denotes the ratio of the number of majority classes to the number of samples.

$$\begin{gathered} C_{FN} = (1+\alpha) \times \mu \times |C| \\ C_{FP} = \beta \times \mu \times |C| \end{gathered} \tag{29}$$

5.2 Cost-sensitive Learning by Modifying the Learning Algorithm

Cost-sensitive learning is also achieved by modifiying the learning algorithm such as decision trees, neural networks and support vector machines. In the following subsections, we mentioned some of the related work that fits in this category. [27, 29, 28].

5.2.1 Decision Tree Algorithms

Decision tree algorithms can be modified to support cost-sensitive learning by using the following methods such as : (i) Embedding cost information into decision tree during construction, (ii) Embedding cost information into decision tree after constructions, (iii) Boosting technique , and bagging technique.

Constructing the decision tree involves two phases such as growing and pruning. One of the traditional ways of applying costs-sensitivity to decision tree algorithms is embedding the cost information into the growing phase of trees using a cost-sensitive splitting criteria. Splitting refers deciding the test attributes (i.e., branching nodes). Sahin [32] presented a novel cost-sensitive decision tree algorithm that minimizes the sum of misclassification costs while picking the splitting feature at each non-terminal node. Saidi [41] compared different techniques such as binary decision tree (CART), boosting and bagging techniques with each other. Their results indicated that boosting ensemble technique achieved better trade-off between misclassification cost and total accuracy. In the boosting ensemble, a group of classifier algorithms are combined. The performance of combined classifiers has significant improvements in comparison with base classifiers.

Lomax [42] introduced a new version of decision trees called non-linear decision tree algorithm that takes the misclassification cost into account. They believed that non-linear decision nodes are more suitable for the basis of cost-sensitive induction than other parallel nodes. Decision tree pruning also is one of the techniques used to minimize the misclassification cost through improved probability estimation whereas Elkan [1] stated that it is better to do no pruning when using the decision tree to predict probabilities.

5.2.2 K-nearest Neighbor Algorithm

K-Nearest-Neighbor algorithm is a memory based learning method where the class of a new example is determined based on its similarity to the existing training examples.

Zhang [40] introduces two cost-sensitive extensions of K-nearest algorithm namely

Direct-CS-KNN and *Distance-CS-KNN*. In Distance-CS-KNN algorithm, after selecting all neighbors in the training set, the class probabilities are computed as the ratio of class neighbor i, to the number of neighbors k, which is usually constant integer in the range of $(1, 12)$. In Direct-CS-KNN algorithm, they compute the distance weights between the test and train samples using a function called distance-weighted voting.

They injected cost-sensitivity by combining distance-weighted voting function with the k-nearest neighbor algorithm. In their approach, the cost of choosing a positive nearest neighbor Cp is computed as $FP \times Wn$ and the cost of choosing the negative nearest neighbor Cn computes as $FN \times Wp$. Therefor, if $Cp > Cn$, then the test sample is classified as negative and the probability of this prediction computes according to the following relation $\frac{Cp}{(Cp+Cn)}$. Otherwise, the test data sample is classified as positive, and it's probability is computed as $\frac{Cn}{(Cp+Cn)}$.

5.2.3 Support Vector Machine Algorithms

Support vector machine is a supervised learning algorithm used in classification task. Given labeled training data samples, the model outputs the hyperplane, which can categorize and classify new examples. Support vector machines have proven to be impressive in real-world applications.

The most recent research in cost-sensitive support vector machine is related to the Iranmehr [43]. They introduced a new method in which, instead of directly manipulating the algorithm, they extended the support vector loss (hinge loss), which minimizes the Bayes risk considered as a metric for measuring the performance of the binary classifier. The proposed loss function can enforce cost sensitivity to the both separable and non-separable training data.

Brefeld *et al.* [44] introduced a cost-sensitive support vector machines that consider Bayes rule for example-dependent costs. Since the Bayes rule only depends on the differences between the real and predicted classification cost, they assigned zero cost for the correct classifications. Masandi [45] proposed a new cost-sensitive support vector machine by extending the support vector loss to the cost-sensitive loss, which can minimize the risk and probability. Their approach is associated with the Bayes Rule and this association guarantees implementing the Bayes-optimal cost-sensitive classification boundary.

5.2.4 Neural Network Algorithm

Studies in cost-sensitive learning with neural networks divided into different categories such as (i) modifying probability estimation of network outputs, (ii) altering outputs of the network directly, (iii) modifying the learning rate and (iv) replacing error minimization function.

The first technique states to leave the training phase unchanged and only manipulates the probabilities of test samples by considering the expected cost of misclassification. In the second technique, the actual output of the network is manipulated to give a higher misclassification cost. The third technique changes the back-propagation by assigning a higher rate to the examples with high cost (high misclassification cost), which means high cost examples will have higher weighs. In the fourth technique, rather than minimizing the square error in a classification problem, the gradient descent algorithm reduces the misclassification cost [46].

Kukar and Kononenko [46] for the first time in 1998 introduced the first work on cost-sensitive artificial neural networks. Their approach enables the neural network to decrease the misclassification cost by back-propagation. Since standard back-propagation algorithms try to decrease the classification error rather than minimizing cost of errors, they are not suitable for cost-sensitive learning. One of the approaches for modifying back-propagation is to change the error function by embedding cost factor $C[i, j]$ to it. The main problem with the back-propagation algorithm is that it causes over-fitting of the training data samples. Over-fitting happens typically when the model learns the details and noises of the training data-samples, which decreased the model accuracy. For avoiding this problem, there are techniques such as early stopping weight decay and dropout.

Kukar and Kononeko introduced a new model regularizing the weights using Bayesian inference, which is similar to the Back-propagation. Rather than traditional Back-propagation that considers the constant value for the weights in neural networks, the Bayes by back-prob method considers the distribution over the weights, and weights are computed using maximum likelihood estimation. Weights are regularized by defining the prior distribution and computing maximum posterior.

Chung *et al.* [47] implemented the cost-sensitive deep neural network. They proposed the loss function that incorporates the cost information into both training and pre-training stage. They used the stacked denoising autoencoder for initializing the parameters of the network in the pre-training stage. The stacked autoencoder is an extended version of the standard autoencoder. It is considered as a multi-layer neural network in which each layer consists of an auto-encoder. So several auto-encoders form a stack, and the input of each stack autoencoder is the output of the last layer. Using denoising auto-encoder in the pre-training phase means first to train the autoencoder and compute the loss. Then the decoder part of the auto-encoder is removed and new auto-encoder embedded; therefore, the input of the new encoder is the latent representation of the previous one. And this process repeats until the algorithm converges. In unsupervised learning, pre-training is a way to initialize the weights of the neural network, so using stacked denoising autoencoder in the pre-training phase helps initialize the weights with strong properties rather than initializing them randomly.

5.2.5 Bayesian Deep Learning

Bayesian deep learning has emerged as a field combining deep neural networks with Bayesian probability theory, which provides a principled way of modelling uncertainty of machine learning models by employing prior distribution on their parameters and inferring the posterior distribution for these parameters using approximations such as Variational

Bayes [48, 49]. Then, the posterior predictive distribution is approximated with sampling methods, which brings a significant computational overhead and leads to noise in predictive uncertainty estimates. In these models, predictive uncertainty is modelled by taking samples from the posterior distributions of model parameters and using the sampled parameters to create a distribution of predictions for each input of the network. However, modelling uncertainty of network parameters may not necessarily lead to good estimates of the predictive uncertainty of neural networks [50].

5.2.6 Evidential Deep Learning

Sensoy *et al.* [2] proposed the evidential deep learning (EDL) in order to quantify uncertainty in classification task. They proposed a neural network classifier which is able to say "I don't know" when it is uncertain about its prediction. In order to achieve it, they used softmax as the output of the classifier. Since softmax converts the logits to the probabilities between zero and one, they interpreted them as the parameters of a categorical distribution and replaced them with parameters of Dirichlet distribution. In order to compute the uncertainty, they assigned belief mass to the whole frame. The belief mass b_k for class label k computed using the evidence for the class labels as follows $b_k = e_k/S$. The prior distribution for the class with zero belief masses is corresponded to the uniform Dirichlet distribution. Any opinion corresponded to the uniform Dirichlet distribution does not contain any information and implies total uncertainty. In this case, the belief masses for classification become $b = [0, \ldots, 0]$, since the uniform Dirichlet distribution does not contain any evidence. Sensoy *et al.* proposed three different loss functions for predicting Dirichlet distributions for classification tasks, i.e., the expected mean square error, the expected cross-entropy, and the expected negative log-likelihood, where the expectations are calculated with respect to the predicted Dirichlet distributions. In this work, we extended the expected cross-entropy loss of EDL with the risk of misclassification.

5.2.7 Cost-sensitive Learning and Imbalanced Data

Class imbalance problem refers to the unequal proportion of class data, where some classes have abundant data that make them an over-represented majority, and some others have rare data, which makes them an under-represented minority. Imbalanced classes lead classifiers to be unable to predict proper boundaries for distinguishing between minor and major classes.

Khan [18] proposed the cost-sensitive deep neural network classifier with the ability to learn robust feature representations for the majority and minority classes. To achieve this, they modified the outputs of the neural network (logit layer) using the cost matrix. Their proposed cost matrix is an all-ones matrix rather than a $1 - I$ matrix with all positive values in a range of $(0, 1]$. The diagonal of the proposed matrix represented the utility for correct predictions. They used cost-sensitive mean error shown in Equation (30).

$$E\left(\theta, \xi\right) = \frac{1}{M} \sum_{i=1}^{M} l\left(d^{(i)}, y_{\theta,\xi}^{(i)}\right) \tag{30}$$

In this formula, $d^{(i)}$ denotes the desired output, and y denotes predicted output. Their proposed loss parametrizes the predicted outputs by θ (theta is the neural network parameter) and proposed cost matrix ξ — both the network parameters and cost matrix optimized by gradient descent and back-propagation. Their learning model's objective is to find the optimum values for network parameters and cost matrix that gives the minimum possible cost E. They defined the loss function as shown in Equation (30). They modified cross-entropy loss $l(d, y)$ by incorporating the class-dependent cost (ξ) in predicted values y_n that is related to the output o_n through the softmax function according to Equation (31). Modified cross-entropy loss is cost calibrated and guess aversive.

$$\begin{aligned} l(d, y) &= -\sum_{n} (d_n log y_n) \\ y_n &= \frac{\xi_{p,n} exp(o_n)}{\sum_k \xi_{p,k} exp(o_k)} \end{aligned} \tag{31}$$

Table 9 shows the comparison matrix of the related work.

	Cost-sensitive Learning				
	Learning Method	Modifying the Distribution	Modifying Learning Algorithm	Cost Sensitivity	Risk Quantification
Elkan[1]	Binary classifier	Re-balancing Growing Pruning	-	Cost matrix	-
Domingo[34]	Binary-multiclass classifier	Re-labeling Over-sampling	-	Cost matrix	-
Kukar[46]	Multi-class Neural nets	-	Back-propagation modification	Cost matrix	-
Zhou[28]	Binary-multiclass classifier	Over-sampling Under-sampling Thresholding	-	Cost matrix	-
Zadrozny[20]	Binary classifier	Sampling Weighting	-	-	-
Abe[35]	Multi class classifier	Data space expansion Iterative weighting	-	Cost matrix	-
Chung[47]	Multi-class Deep neural net	-	Loss function modification	Cost matrix	-
Tu[29]	Multi-class support vector regression	-	One-side support vector Loss	Cost matrix	-
Zadronzy[24]	Decision-tree Naive Bayes classifier	Smoothing	-	Cost matrix	-
Zheng[23]	Support vector classifier	-	Support-vector loss modification	Cost matrix	-
Zhou[22]	Multi-class classifier	re-scaling re-weighting	-	Cost matrix	-
Chen[39]	Random-forest classifier	balancing weighting	-	Cost matrix	-
Zhang[37]	Multi-class Convolutional Neural net	Adversarial Perturbation	-	Cost matrix	-
Weiss[21]	Multi-class classifier	Sampling	-	Cost matrix	-
Khan[18]	Multi-class Deep neural net classifier	-	Cost-sensitive Loss	Cost matrix	-
Sheng[25]	Binary classification	Thresholding	-	Cost matrix	-
Mccarthy[26]	-Multi-class - classifier	Sampling	-	Cost matrix	-
Jing[30]	Binary Bayesian classifier	Instance weighting	-	Cost matrix	-
Chai[31]	Naive Bayes classifier	Naive bayes	-	-	-
Risk-aware Evidential Deep-learning	Multi-class classifier	-	Expected Cross-Entropy	Cost matrix	Risk matrix

Table 9: Comparison matrix for related work on cost-sensitive learning

CHAPTER VI

CONCLUSION

As a result of striking success of deep learning in recent years, deep classifiers are now an indispensable part of autonomous systems. However, these black-box models may be very confident when their predictions are wrong and lead agents to make mistakes in their decisions. Furthermore, standard training of deep models neglects that different mistakes involve in different level of risk for the agents depending them. In this thesis, we significantly extend the evidential deep learning to address this problem.

Our approach allows agents to incorporate their own risk for classification mistakes into the training of evidential classifiers [2]. We borrow ideas from decision theory and integrated the notion of pignistic probabilities into neural networks in a novel way using the predictive uncertainty. Our usage of pignistic probabilities in making classification decisions are similar to the usage of pignistic probabilities in Dempster–Shafer theory (DST) of evidence [51] and Subjective Logic [13]. By incorporating the notion of risk and pignistic probabilities into the loss function, our approach improves the uncertainty quantification of evidential networks and, at the same time, significantly minimizes the cost of misclassification for autonomous agents. It is worth noting that we learn these probabilities using gradient descent as a part of the training of deep neural networks. We believe that the link between the pignistic probabilities and making classification decisions using deep neural networks should be explored further in future.

Our experimental results showed that we do not only decrease the overall cost of incorrect predictions, but also increase the uncertainty of wrong predictions so that these predictions can easily be discriminated from the correct predictions. Hence, our approach allows a more accurate detection of classification errors and creates an opportunity for avoiding the undesirable consequences of them. In other words, our approach can quantify predictive uncertainty,which makes it possible to associate high uncertainty with predictions that are more likely to be wrong and incorporate risk in training phase which makes it possible to avoid making wrong predictions.

We would like to extend our approach in future to enhance robustness of neural networks against adversarial perturbations, which are engineered to result in the maximum damage through misclassification of samples into high-risk categories.

Bibliography

[1] C. Elkan, "The foundations of cost-sensitive learning," in *International joint conference on artificial intelligence*, vol. 17, pp. 973–978, Lawrence Erlbaum Associates Ltd, 2001.

[2] M. Sensoy, L. Kaplan, and M. Kandemir, "Evidential deep learning to quantify classification uncertainty," in *Advances in Neural Information Processing Systems*, pp. 3179–3189, 2018.

[3] E. Alpaydin, "Introduction to machine learning/ethem alpaydin," 2014.

[4] R. E. Cytowic and D. M. Eagleman, *Wednesday is indigo blue: Discovering the brain of synesthesia*. MIT Press, 2011.

[5] M. H. Hassoun *et al.*, *Fundamentals of artificial neural networks*. MIT press, 1995.

[6] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," in *NIPS*, 2012.

[7] "ConvolutionalNeuralNet architecture." https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Accessed: 2020-01-17.

[8] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018.

[9] "Activation functions. architecture." https://towardsdatascience.com/comparison-of-activation-functions-for-deep-neural-networks-706ac4284c8a. Accessed: 2020-01-17.

[10] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT Press, 2016.

[11] J. Joyce, "Bayes' theorem," 2003.

[12] S. Kotz, N. Balakrishnan, and N. Johnson, *Continuous Multivariate Distributions*, vol. 1. New York: Wiley, 2000.

[13] A. Josang, *Subjective Logic: A Formalism for Reasoning Under Uncertainty*. Springer, 2016.

[14] S. Lefkimmiatis, P. Maragos, and G. Papandreou, "Bayesian inference on multiscale models for poisson intensity estimation: Applications to photon-limited image denoising," *IEEE Transactions on Image Processing*, vol. 18, no. 8, pp. 1724–1741, 2009.

[15] D. Dubois, H. Prade, and P. Smets, “A definition of subjective possibility,” *International Journal of Approximate Reasoning*, vol. 48, no. 2, pp. 352–364, 2008.

[16] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, “Gradient-based learning applied to document recognition,” *In the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[17] C. Louizos and M. Welling, “Multiplicative normalizing flows for variational bayesian neural networks,” in *ICML*, 2017.

[18] S. H. Khan, M. Hayat, M. Bennamoun, F. A. Sohel, and R. Togneri, “Cost-sensitive learning of deep feature representations from imbalanced data,” *IEEE transactions on neural networks and learning systems*, vol. 29, no. 8, pp. 3573–3587, 2017.

[19] E. Hüllermeier and W. Waegeman, “Aleatoric and epistemic uncertainty in machine learning: A tutorial introduction,” *arXiv preprint arXiv:1910.09457*, 2019.

[20] B. Zadrozny, J. Langford, and N. Abe, “Cost-sensitive learning by cost-proportionate example weighting.,” in *ICDM*, vol. 3, p. 435, 2003.

[21] G. M. Weiss, K. McCarthy, and B. Zabar, “Cost-sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs?,” *Dmin*, vol. 7, no. 35-41, p. 24, 2007.

[22] Z.-H. Zhou and X.-Y. Liu, “On multi-class cost-sensitive learning,” *Computational Intelligence*, vol. 26, no. 3, pp. 232–257, 2010.

[23] E.-h. Zheng, P. Li, and Z.-h. Song, “Cost sensitive support vector machines,” *Control and decision*, vol. 21, no. 4, p. 473, 2006.

[24] B. Zadrozny and C. Elkan, “Learning and making decisions when costs and probabilities are both unknown,” in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 204–213, ACM, 2001.

[25] V. S. Sheng and C. X. Ling, “Thresholding for making classifiers cost-sensitive,” in *AAAI*, pp. 476–481, 2006.

[26] K. McCarthy, B. Zabar, and G. Weiss, “Does cost-sensitive learning beat sampling for classifying rare classes?,” in *Proceedings of the 1st international workshop on Utility-based data mining*, pp. 69–77, ACM, 2005.

[27] M.-Z. Tang, C.-H. Yang, W.-H. Gui, and Y.-F. Xie, “Cost-sensitive probabilistic neural network with its application in fault diagnosis,” *Control and Decision*, vol. 25, no. 7, pp. 1074–1078, 2010.

[28] Z.-H. Zhou and X.-Y. Liu, “Training cost-sensitive neural networks with methods addressing the class imbalance problem,” *IEEE Transactions on knowledge and data engineering*, vol. 18, no. 1, pp. 63–77, 2005.

[29] H.-H. Tu and H.-T. Lin, “One-sided support vector regression for multiclass cost-sensitive classification.,” vol. 2, no. 4, p. 5, 2010.

[30] L. Jiang, C. Li, and S. Wang, "Cost-sensitive bayesian network classifiers," *Pattern Recognition Letters*, vol. 45, pp. 211–216, 2014.

[31] X. Chai, L. Deng, Q. Yang, and C. X. Ling, "Test-cost sensitive naive bayes classification," in *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pp. 51–58, IEEE, 2004.

[32] Y. Sahin, S. Bulkan, and E. Duman, "A cost-sensitive decision tree approach for fraud detection," *Expert Systems with Applications*, vol. 40, no. 15, pp. 5916–5923, 2013.

[33] B. Krawczyk, M. Woźniak, and G. Schaefer, "Cost-sensitive decision tree ensembles for effective imbalanced classification," *Applied Soft Computing*, vol. 14, pp. 554–562, 2014.

[34] P. Domingos, "Metacost: A general method for making classifiers cost-sensitive," in *KDD*, vol. 99, pp. 155–164, 1999.

[35] N. Abe, B. Zadrozny, and J. Langford, "An iterative method for multi-class cost-sensitive learning," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 3–11, ACM, 2004.

[36] X.-Y. Liu and Z.-H. Zhou, "The influence of class imbalance on cost-sensitive learning: An empirical study," in *Sixth International Conference on Data Mining (ICDM'06)*, pp. 970–974, IEEE, 2006.

[37] X. Zhang and D. Evans, "Cost-sensitive robustness against adversarial examples," *arXiv preprint arXiv:1810.09225*, 2018.

[38] H.-T. Lin, "A simple cost-sensitive multiclass classification algorithm using one-versus-one comparisons," *National Taiwan University, Tech. Rep*, 2010.

[39] C. Chen, A. Liaw, and L. Breiman, "Using random forest to learn imbalanced data. 2004," *University of California, Berkeley*, 2015.

[40] S. Zhang, "Cost-sensitive knn classification," *Neurocomputing*, pp. 112–131, 2019.

[41] M. Saidi, M. E. H. Daho, N. Settouti, and M. E. A. Bechar, "Comparaison of ensemble cost sensitive algorithms: Application to credit scoring prediction.," in *ICAASE*, pp. 56–61, 2018.

[42] S. Lomax and S. Vadera, "A survey of cost-sensitive decision tree induction algorithms," *ACM Computing Surveys (CSUR)*, vol. 45, no. 2, p. 16, 2013.

[43] A. Iranmehr, H. Masnadi-Shirazi, and N. Vasconcelos, "Cost-sensitive support vector machines," *Neurocomputing*, vol. 343, pp. 50–64, 2019.

[44] U. Brefeld, P. Geibel, and F. Wysotzki, "Support vector machines with example dependent costs," in *European Conference on Machine Learning*, pp. 23–34, Springer, 2003.

[45] H. Masnadi-Shirazi, N. Vasconcelos, and A. Iranmehr, “Cost-sensitive support vector machines,” *arXiv preprint arXiv:1212.0975*, 2012.

[46] M. Kukar, I. Kononenko, *et al.*, “Cost-sensitive learning with neural networks.,” in *ECAI*, pp. 445–449, 1998.

[47] Y.-A. Chung, H.-T. Lin, and S.-W. Yang, “Cost-aware pre-training for multiclass cost-sensitive deep learning,” *arXiv preprint arXiv:1511.09337*, 2015.

[48] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wiestra, “Weight uncertainty in neural networks,” in *ICML*, pp. 1613–1622, 2015.

[49] Y. Gal and Z. Ghahramani, “Dropout as a Bayesian approximation: Representing model uncertainty in deep learning,” in *ICML*, pp. 1050–1059, 2016.

[50] D. Hafner, D. Tran, A. Irpan, T. Lillicrap, and J. Davidson, “Reliable uncertainty estimates in deep neural networks using noise contrastive priors,” *arXiv preprint arXiv:1807.09289*, 2018.

[51] A. Dempster, “A generalization of Bayesian inference,” in *Classic works of the Dempster-Shafer theory of belief functions*, pp. 73–104, Springer, 2008.