Learning Smooth Conditional Class Probability Functions With Deep Neural Networks: From Statistical Theory to Practice

H. C. (Bram) Otten (s2330326)

Supervisors: Prof. Dr. A. J. Schmidt-Hieber
J. M. Bos

MASTER THESIS

Defended on October 25, 2021

Universiteit Leiden
The Netherlands

STATISTICS & DATA SCIENCE

Preface

I would like to thank my supervisors, Prof. Dr. Johannes Schmidt-Hieber and PhD candidate Thijs Bos; as well as Марија Трајаноска. You and I have had to do a bit too much for this project.

I was explicitly looking for a hard thesis topic a year ago. I wanted to struggle with the fundamentals of statistical/machine learning because I otherwise never would, and because I thought it was cool.

Of course, I got my struggle (which is to say that *we* got *our* struggle). But some difficulties were unexpected. I started this project confident about my writing skills, but my unclarity caused delays. And with the pace I thought I had, this project would have been done before the summer. Somewhat figuratively speaking, the report indeed shrank to less than half the size it had then. But some now-removed mathematics and especially interpretation was just unnecessary, confusing, or wrong.

My complaints only emphasize that I have actually learned *more* than just what statistical theory for hip machine learning methods is about. I can already tell I will appreciate the whole experience even more as it descends further into the past. I hope and think the people I thanked before will be and are fine with the experience now that it is past, too.

Contents

Abstract

We provide a simulation study complementing the theoretical results of Bos and Schmidt-Hieber (2021) for supervised classification using deep neural networks. Their *main risk bound* suggests a faster truncated Küllback-Leibler divergence risk convergence rate with smoother conditional class probability functions and when fewer conditional class probabilities are near zero; as well as that convergence rate is fast when the functions have a high degree of smoothness even if many probabilities are near zero. The proportion of small conditional class probabilities can be measured by *small value bound* index α. We calculate α for an illustrative selection of settings with conditional class probability functions that have an arbitrarily high Hölder smoothness index β. We estimate the Küllback-Leibler divergence risk convergence rate in these settings by evaluating networks trained on simulated datasets of various sizes. We find slower convergence rates than suggested by the main risk bound. However, in line with expectations, α has no consistent effect on convergence rate when combined with arbitrarily high β.

Keywords: *simulation study, ReLU networks, softmax, small value bound, Hölder smoothness, Küllback-Leibler divergence.*

Single Page Summary (With Overview of Contributions)

Deep neural networks (DNNs) have recently been applied successfully to supervised classification tasks such as image and speech recognition. The theoretical understanding of DNNs has improved over the years, but this work is far from complete. For example, consider the batch normalization method due to Ioffe and Szegedy (2015). The method was proposed as a fix for the *internal covariate shift* problem and became widely used. Then Santurkar et al. (2019) found batch normalization to work for completely different reasons.

A supervised classification task is generally approached as the approximation of a discrete conditional probability distribution. This enables one to use a wide range of loss functions with characteristics that are desirable for DNN training. A commonly used loss function is the negative logarithm of the likelihood. A commonly used measure for the difference between two distributions more generally is the Küllback-Leibler (KL) divergence.

In our simulation study, we examine whether the expected KL divergence loss (or *risk)* of adequately trained DNNs changes along with properties of the dataset as expected after the theoretical work by Bos and Schmidt-Hieber (2021). Their derived *main risk bound* suggests a faster risk *convergence rate* with smoother conditional class probability functions and when fewer conditional class probabilities are near zero. The main risk bound suggests a fast convergence rate when the functions have a high degree of smoothness, even if many probabilities are near zero.

We perform a simulation study, so that we can generate multiple datasets and know the true conditional class probabilities. We define the distribution from which we sample input, as well as the conditional probability function used for sampling corresponding output classes from a categorical distribution. We use the true conditional class probabilities only during evaluation; the DNNs train on class labels. There are no particularly unrealistic or strict requirements on the input distribution and conditional class probability function; we look at an illustrative selection of distributions and *arbitrarily smooth* conditional class probability functions. We experiment with multiple training set sizes and a large test set per setting to uncover estimated risk convergence rates.

The theoretical results hold for DNNs with a feedforward architecture, that make use of rectified linear unit and softmax activation functions. The theoretical work prescribes rates at which the networks' width, depth, and number of nonzero parameters should grow with the size of the training set. The main risk bound assumes low negative log-likelihood training loss, but prescribes no optimization method; we use a variant of stochastic gradient descent.

As figure 1 exemplifies, our DNNs are able to approximate simple conditional class probability functions well from a visual perspective. We found slower estimated KL divergence risk convergence rates than expected. However, in line with theory, the proportion of small conditional class probabilities did *not* affect convergence rate when the conditional class probability function is arbitrarily smooth.

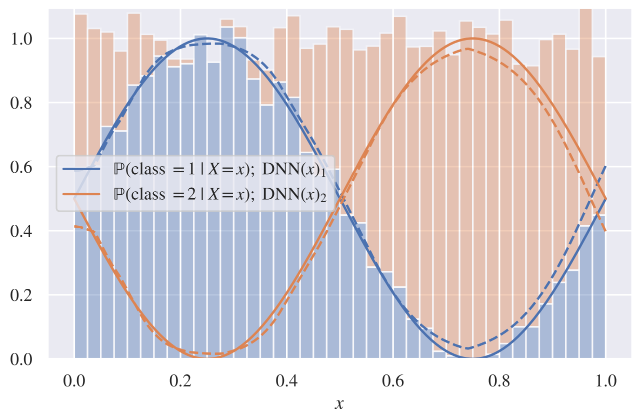


Figure 1: DNN performance when the input distribution is $X \sim \text{uniform}([0,1])$, and the conditional class probability functions consist of squashed sines: $\mathbb{P}(\text{class} = 1 \mid X = x) = (1 + \sin(2\pi x))/2$ and $\mathbb{P}(\text{class} = 2 \mid X = x) = (1 - \sin(2\pi x))/2$. The class distribution of the 8192-sized training set is shown in the histogram. The full lines display the true conditional class probabilities, while the dashed lines display the trained DNN's predicted probabilities on a large test set.

1 Introduction

1.1 Deep Neural Networks for Classification

We are not impressed when computers outperform humans on well-defined simple tasks such as calculating, remembering, or consistently being logical. Many challenging tasks we use computers for now involve learning patterns from noisy data sampled from an unknown distribution. Deep neural networks (DNNs) have recently been applied successfully to a wide range of such tasks. For instance, DNNs that involve convolutional filters obtain state-of-the-art supervised image classification performance since Krizhevsky, Sutskever, and Hinton (2012).

The focus of this work is on supervised classification with deep *feedforward* neural networks. In this setting, there is access to a training dataset $\mathcal{D}_n := \{(\boldsymbol{X}_j, \boldsymbol{Y}_j) : j = 1, \ldots, n\}$ with elements independent *of* but distributed *as* the pair of random variables $(\boldsymbol{X}, \boldsymbol{Y})$. The n samples in the training set consist of input distributed as $\boldsymbol{X} \in \mathbb{R}^d$, and an indication of one of K corresponding class membership labels in a vector $\boldsymbol{Y} \in \mathcal{B}^K := \{\boldsymbol{v} \in \{0,1\}^K : \sum_{k=1}^K v_k = 1\}$. We denote the conditional probability of a sample of input $\boldsymbol{x}$ being labeled class $k \in \{1, \ldots, K\}$ given the distribution of $(\boldsymbol{X}, \boldsymbol{Y})$ by $p_k(\boldsymbol{x}) := \mathbb{P}_{Y|X}(Y_k = 1 \mid \boldsymbol{X} = \boldsymbol{x})$. The label that determines the $\boldsymbol{Y} = \boldsymbol{y}$ associated with $\boldsymbol{X} = \boldsymbol{x}$ is drawn from a categorical distribution with probability vector $\boldsymbol{p}(\boldsymbol{x}) := (p_1(\boldsymbol{x}), \ldots, p_K(\boldsymbol{x}))$. Example 1 describes what a supervised image classification dataset could consist of.

Example 1

In a supervised image classification task, datasets consist of samples from the images' distribution, along with corresponding labels. For instance, $\boldsymbol{X}$ can represent the distribution of grayscale intensities in the $d = 1024 \times 1536$ pixels of mammography images. The associated $\boldsymbol{Y}$ can then represent whether there are early signs of cancer. For instance, $\boldsymbol{Y} = (1, 0)$ indicates no early signs, and $\boldsymbol{Y} = (0, 1)$ indicates there are early signs. In this cancer detection context, mammography experts generally provide the labels manually.

We consider the two images in Figure 1 samples from image distribution $\boldsymbol{X}$. We can refer to the respective samples as $\boldsymbol{X} = \boldsymbol{x}_{\mathrm{A}}$ and $\boldsymbol{X} = \boldsymbol{x}_{\mathrm{B}}$. Since both samples contain early signs of cancer, the corresponding label vectors $\boldsymbol{Y} = \boldsymbol{y}_{\mathrm{A}}$ and $\boldsymbol{Y} = \boldsymbol{y}_{\mathrm{B}}$ are $(0, 1)$.

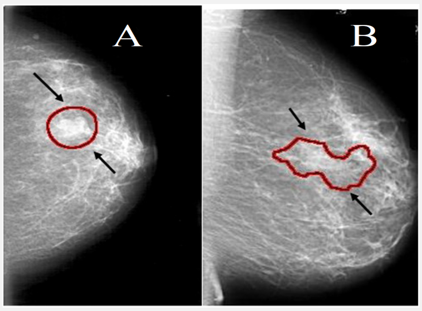


Figure 1: Mammography images with *masses* (A, left) or *microcalcifications* (B, right), which can both be early signs of cancer (Ragab et al. 2019).

Deep neural networks learn to approximate the conditional class probability function $\boldsymbol{p}$. A DNN $\widehat{\boldsymbol{p}}$ should learn parameters that minimize the *risk* $\mathbb{E}_{(X,Y)} L(\widehat{\boldsymbol{p}}(\boldsymbol{X}), \boldsymbol{Y})$, where L is a chosen loss function. Loss functions measure error; risk is the expected amount of loss. Calculating the risk requires knowledge of the distribution of $(\boldsymbol{X}, \boldsymbol{Y})$. Since this knowledge is not available, the DNN's learning process uses the loss over a training set, $(1/n) \sum_{j=1}^n L(\widehat{\boldsymbol{p}}(\boldsymbol{X}_j), \boldsymbol{Y}_j)$, as a proxy. A DNN trains by iteratively changing its parameters to reduce training loss using (stochastic) gradient descent. The *generalization error* of a DNN is measured over samples from $(\boldsymbol{X}, \boldsymbol{Y})$ that did not occur in the training set. Details follow later in this work.

A widely used loss function for training DNNs for supervised classification is the negative log-likelihood (NLL, also known as *cross-entropy*). In this case, the training loss is

$$\mathrm{NLL}(\widehat{\boldsymbol{p}} \mid \mathcal{D}_n) = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} Y_k^i \log \widehat{p}_k(\boldsymbol{X}_i),$$

where log denotes the natural logarithm (other logarithm bases will be mentioned in subscript).

Since $\widehat{\boldsymbol{p}}(\boldsymbol{X}_j)$ is compared with $\boldsymbol{Y}_j$, it may seem logical to let DNNs predict a class label rather than a discrete probability distribution per input. The predicted class label can in fact be extracted from a discrete probability distribution: it is the most probable class. The training loss associated with the label prediction approach could be the proportion of misclassifications over a $\mathcal{D}_n$,

$$1 - \frac{1}{n}\sum_{j=1}^{n}\left[\mathbb{1}_{\{k\,:\,y_k^j=1\}}(\operatorname{argmax}\widehat{\boldsymbol{p}}(\boldsymbol{X}_j))\right], \tag{1}$$

where the indicator function $\mathbb{1}_A(x)$ is 1 if $x \in A$ and 0 if $x \notin A$.

1.2 Statistical Theory for DNNs

The statistical theory of deep neural networks has been of interest to researchers for decades, but the current wave of DNN popularity has followed good performance *in practice*. A detailed history of the theory and application of neural networks is out of the scope of this work. The recent impressive practical results seem to stem mostly from the availability of larger datasets and the computational feasibility of training larger DNNs (Goodfellow, Bengio, and Courville 2016, §1.2).

That DNNs' popularity seems to follow practical rather than theoretical results can be illustrated with a recent addition to DNN training algorithms, *batch normalization*. Ioffe and Szegedy (2015) proposed the method as a fix for the *internal covariate shift* problem.[1] Batch normalization became widely used, in part because it was applied by an image classification DNN that obtained impressive benchmark results (He et al. 2015a). Later, Santurkar et al. (2019) investigated how batch normalization really works. The investigation found that batch normalization can indeed improve a DNN's performance, but does so for different reasons. Batch normalization can speed up and stabilize the training process, but does not necessarily reduce covariate shift. The history of batch normalization is an example of:

- how a useful DNN variation can be identified by its impressive practical performance;
- how a DNN variation can become widely used if it obtains good practical results;
- but also that there is, generally speaking, a lack of theoretical understanding of DNNs. It is often unclear what influences DNN performance, as well as how and why this influence occurs.

Theoretical results about deep neural networks do not necessarily reflect real-world performance. For instance, Hornik, Stinchcombe, and White (1989) showed that neural networks with at least one hidden layer and sufficiently many neurons can approximate practically any function with any amount of accuracy. However, there is no training algorithm that is guaranteed to find this approximation. There is a need for *relevant* theoretical results, which can be obtained with known training algorithms and which are applicable in realistic settings.

The development of relevant theoretical results has become easier since deep neural networks composed of rectified linear units (ReLUs) have become widely used. The wide use of the ReLU activation function within DNNs follows promising (practical) results by Nair and Hinton (2010) and Glorot, Bordes, and Bengio (2011). The definition of the ReLU activation function is $\sigma(x) := \max\{0, x\}$. Its derivative is thus 0 or 1 ($\sigma'(0)$ can be defined as either). It is easier to develop theoretical results for DNNs that use ReLU activation functions, than for DNNs that use previously common activation functions with more complicated derivatives. Although the zero-derivative may cause issues during the gradient-based training process of DNNs – prompting proposals for and comparisons of slight variations of ReLUs in, e.g., He et al. (2015b) and Ramachandran, Zoph, and Le (2017) – the standard ReLU activation function remains popular in both practice and research.

Theoretical results for deep neural networks can come in the form of bounds on risk. The *convergence rate* of risk bounds – convergence as the training set size increases – can depend on a wide range of quantities. Schmidt-Hieber (2020) gives an example of a potentially applicable theoretical result that involves the discussed concepts. The paper finds fast mean squared error risk convergence rates for sparse feedforward DNNs that consist of ReLUs and have small parameter values in a nonparametric regression setting.

Kim, Ohn, and Kim (2019) study risk convergence rates of DNNs that use the ReLU activation function as well, but examine the supervised classification setting. The paper investigates the misclassification rate risk ((1) defines the misclassification rate *loss function*). Fast rates occur when conditional class

[1] The technical details are not important, but the internal covariate shift refers to the distribution of all of a network's layers' in- and output changing throughout the iterative learning process. This is potentially problematic because the parameter magnitudes are adapted to the distributions of the previous rather than current iteration (batch).

probabilities are close to 0 or 1, or if, e.g., the *margin condition* is met. The margin condition limits the proportion of conditional class probabilities near the decision boundary. The decision boundary is the *hypersurface* beyond which one class becomes more likely than another. In case there are two classes, the decision boundary is $\{\boldsymbol{x}\colon p_1(\boldsymbol{x}) = p_2(\boldsymbol{x}) = 1/2\}$ – for any $\boldsymbol{x}$ not on the decision boundary, either $p_1(\boldsymbol{x}) > p_2(\boldsymbol{x})$ or vice versa.

1.3 This Work

We perform a simulation study complementing the theoretical result of Bos and Schmidt-Hieber (2021) for supervised classification using deep neural networks. Their main result states that two properties of the conditional class probability function determine the convergence rate of the truncated Küllback-Leibler (KL) divergence risk. The KL divergence risk is defined by

$$\mathbb{E}_{(\boldsymbol{X},\boldsymbol{Y})} \operatorname{KL}\left(\widehat{\boldsymbol{p}}(\boldsymbol{X}), \boldsymbol{p}(\boldsymbol{X})\right) \coloneqq \mathbb{E}_{(\boldsymbol{X},\boldsymbol{Y})}\left[\sum_{k=1}^{K} p_k(\boldsymbol{X}) \log\left(\frac{p_k(\boldsymbol{X})}{\widehat{p}_k(\boldsymbol{X})}\right)\right], \tag{2}$$

and is a commonly used measure for the difference between two probability distributions. In our setting, the two distributions are discrete, and the reference distribution is that of the true conditional class probabilities, $\boldsymbol{p}(\boldsymbol{X})$. The KL divergence risk of two identical distributions is zero.

For our simulation study, we consider an illustrative but simple selection of settings, each characterized by a distribution of input ($\boldsymbol{X}$) and an *arbitrarily smooth* conditional class probability function ($\boldsymbol{p}$). We calculate relevant quantities for the theoretical result per setting, and later run simulations with multiple training set sizes for an estimated risk convergence rate. We compare the estimated risk convergence rate to the rate suggested by the theoretical result. We study input distributions and conditional class probability functions with domains $[0,1]^d$. This simplifies calculations but results can be generalized, since values in any d-dimensional compact set can be mapped to values in $[0,1]^d$.

The theoretical result only holds for DNNs with a specific architecture. This architecture is completely feedforward and uses the ReLU activation function in the input and all hidden layers. The later-introduced softmax activation function is used in the networks' final layer. The networks' parameters are constrained to $[-1,1]$. Finally, the theory prescribes rates at which the networks' width, depth, and number of nonzero parameters should grow with the size of the training set. The theory assumes low negative log-likelihood training loss, but does not prescribe a training method. We use a variant of stochastic gradient descent.

Our scope is limited to examining whether the theoretical convergence rate of Bos and Schmidt-Hieber (2021) can be found using simulations. Since the result only holds for specific DNNs, we do not compare with other supervised classification methods. Similarly, since the result depends on quantities that can only be known in simulation settings, we do not attempt to find its effect in real-world settings. We do justify choices not prescribed by the theory, such as for hyperparameter values. We also highlight where and explain why we deviate from theory.

If our practical results are in line with expectations, they indicate relevance of the theoretical result. The expected result would imply that other factors do not diminish the theoretical influence of the proportion of small conditional class probabilities and smoothness of conditional class probability functions in simulation settings.

There is potential for practical relevance, because small conditional class probabilities are common. For instance, in image classification tasks with many classes, most classes have low conditional probability. This is necessarily true if K is large: the probability that an image that displays a car in fact displays a cat when there are hundreds of labels that occur equally often is practically zero. Additionally, the minimization of negative log-likelihood using ReLU and softmax networks is practically *the* standard approach for supervised classification with DNNs.

Our work is a rare combination of practice and statistical theory about deep neural networks. The discrete probability distribution approximation approach to supervised classification is common, but simulation studies using the assumed statistical model are not. If we find that DNNs can indeed learn the relationship between $\boldsymbol{X}$ and $\boldsymbol{Y}$ in our model – where $\boldsymbol{Y}$ is sampled from a categorical distribution with probability function $\boldsymbol{p}(\boldsymbol{X})$ – we lend credence to the practice of interpreting a softmax DNN's output as probabilities rather than as class-ranking.

We also provide an implementation of a reproducible experimental setup for examining the discrete probability distribution approximation capabilities of any supervised classification algorithm. Our methodology can be adapted for examination of other theoretical supervised classification results, or to generate datasets of different settings. The Python implementation, available on `https://github.com/bramotten/DNN-Classification-Theory-In-Practice`, makes use of the NumPy, Hyperopt, TensorFlow, and Keras packages.

The outline of the rest of this work is as follows:

- Section 2.1 formally introduces the deep neural networks we study.
- Section 2.2 describes the small value bound, a measure for the proportion of small conditional class probabilities.
- Section 2.3 describes Hölder smoothness.
- Section 2.4 describes the main (truncated KL divergence) risk bound of Bos and Schmidt-Hieber (2021) – this is the theoretical result we aim to complement.
- Section 3 describes (novel) tools for simplifying calculation of the small value bound.
- Section 4.1 introduces the specific settings we consider in our simulation study, and features the calculation of quantities that are relevant for the theoretical result.
- Section 4.2 explains how we sample from uncommon distributions.
- Section 4.3 describes the DNN implementation details, and demonstrates that the implemented DNNs can indeed approximate the true conditional class probability functions in the considered simulation settings.
- Section 4.4 explains how we search for DNN hyperparameter values.
- Section 4.5 describes the experimental setup used in our simulation study.
- Section 4.6 describes how we fit formulas of the theoretical convergence rate form to the estimated risk convergence rate we obtain from our experiments.
- Section 4.7 explains the reproducibility of the whole simulation study.
- Section 5 describes our results.
- Section 6 summarizes this work.
- Section 7 contains suggestions for future work.

2 Statistical Theory for DNNs

2.1 Description of DNNs for Classification

2.1.1 ReLU and Softmax Architecture

We detail the deep neural network architecture of Bos and Schmidt-Hieber (2021) first. We use this DNN architecture in our simulation study as well. The remainder of Section 2.1 describes training DNNs with (stochastic) gradient descent more generally. We sometimes introduce multiple approaches with the same goal, such as multiple weight initialization strategies. In these cases, the approach taken in our simulation study is specified later.

We specify the number of hidden layers of a DNN with $L \in \mathbb{N}$, and the width of the layers with $\boldsymbol{m} = (m_0, \ldots, m_{L+1}) \in \mathbb{N}^{L+2}$. The widths of the non-hidden input and output layers correspond to the dimensionality of the input and output of the network respectively, so $m_0 = d$ (the dimensionality of an input sample) and $m_{L+1} = K$ (the number of output classes). The depth of a DNN is $L + 2$.

The DNNs we consider use the activation function of a rectified linear unit (ReLU) on all neurons except those in the output layer. This activation function is defined as $\sigma(x) := \max\{0, x\}$. We use the following notation to rectify and shift r non-activated neuron values with bias vector $\boldsymbol{v} \in \mathbb{R}^r$ at once:

$$\sigma_{\boldsymbol{v}} \begin{pmatrix} x_1 \\ \vdots \\ x_r \end{pmatrix} = \begin{pmatrix} \sigma(x_1 + v_1) \\ \vdots \\ \sigma(x_r + v_r) \end{pmatrix}.$$

We use the softmax activation function on the neurons in the final layer. This function converts the K non-activated neuron values into probabilities:

$$\Phi \begin{pmatrix} x_1 \\ \vdots \\ x_K \end{pmatrix} = \frac{1}{\sum_{j=1}^{K} e^{x_j}} \begin{pmatrix} e^{x_1} \\ \vdots \\ e^{x_K} \end{pmatrix},$$

so that $\Phi(\boldsymbol{x})_k \in [0, 1]$ and $\sum_{k=1}^{K} \Phi(\boldsymbol{x})_k = 1$. The output of our DNNs is thus a discrete probability distribution (of a discrete random variable that can take on K values).

A feedforward DNN with our architecture is a function $\widehat{\boldsymbol{p}} \colon [0, 1]^d \longrightarrow [0, 1]^K$ defined by

$$\widehat{\boldsymbol{p}}(\boldsymbol{x}) = \Phi \left(\boldsymbol{W}^L \sigma_{\boldsymbol{v}^{L-1}} \cdots \boldsymbol{W}^2 \sigma_{\boldsymbol{v}^1} \left(\boldsymbol{W}^1 \sigma_{\boldsymbol{v}^0} \left(\boldsymbol{W}^0 \boldsymbol{x} \right) \right) \cdots \right), \tag{3}$$

where $\boldsymbol{W}^j$ is the $m_{j+1} \times m_j$ matrix of real numbers representing weights and $\boldsymbol{v}^j \in \mathbb{R}^{m_j}$ contains the biases.[2] There is a weight associated with every input of every neuron, and a bias associated with every neuron. These $\sum_{j=0}^{L} m_{j+1} \left(m_j + 1 \right)$ *parameters* are learned by the network during the training process.

To make (3) more concrete, the values of the m_1 neurons in the first hidden layer are:

$$\begin{aligned} \sigma_{\boldsymbol{v}^0}(\boldsymbol{W}^0 \boldsymbol{x}) &= \sigma_{\boldsymbol{v}^0} \left(\begin{pmatrix} w^0_{1,1} & \cdots & w^0_{1,d} \\ \vdots & \ddots & \vdots \\ w^0_{m_1,1} & \cdots & w^0_{m_1,d} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} \right) \\ &= \sigma_{\boldsymbol{v}^0} \begin{pmatrix} \sum_{i=1}^{d} w^0_{1,i}\, x_i \\ \vdots \\ \sum_{i=1}^{d} w^0_{m_1,i}\, x_i \end{pmatrix} = \begin{pmatrix} \sigma \left(v^0_1 + \sum_{i=1}^{d} w^0_{1,i}\, x_i \right) \\ \vdots \\ \sigma \left(v^0_{m_1} + \sum_{i=1}^{d} w^0_{m_1,i}\, x_i \right) \end{pmatrix}. \end{aligned}$$

We also define a subset of networks with at most s non-zero parameters – i.e., a certain degree of sparsity – and no parameters with absolute value greater than 1:

$$\mathcal{F}(L, \boldsymbol{m}, s) := \left\{ \widehat{\boldsymbol{p}} \text{ as in (3)} \colon \sum_{j=0}^{L} \|\boldsymbol{W}^j\|_0 + \sum_{j=0}^{L} \|\boldsymbol{v}^j\|_0 \leq s,\ \max_j \|\boldsymbol{W}^j\|_\infty \leq 1,\ \max_j \|\boldsymbol{v}^j\|_\infty \leq 1 \right\}. \tag{4}$$

Definition (4) contains two *norms*. These are the norms we may use for a vector $\boldsymbol{x} \in \mathbb{R}^d$:

[2] The domain of $\widehat{\boldsymbol{p}}$ is identical to that of $\boldsymbol{p}$, which we assume to be $[0, 1]^d$.

- The number of nonzero entries $\|\boldsymbol{x}\|_0 := \sum_{j=1}^{d}(x_j)^0 = \sum_{j=1}^{d} \mathbb{1}_{\{x:\, x\neq 0\}}(x_j)$.
- The absolute sum of all entries $\|\boldsymbol{x}\|_1 := \sum_{j=1}^{d} |x_j|$.
- The maximum entry $\|\boldsymbol{x}\|_\infty := \max_j |x_j|$. For application on a function $f\colon D \to \mathbb{R}$, we define $\|f\|_\infty := \sup_{x\in D} |f(x)|$.

2.1.2 Loss Functions and Estimating Risk

Loss functions quantify the *cost* of the difference between the true conditional class probability function $\boldsymbol{p}$ and the trained deep neural network's approximation, $\widehat{\boldsymbol{p}}$. The expected value of a loss function is the *risk* function. We introduce the loss and (estimated) risk functions we use in this section.

We first describe two types of non-training datasets: the *validation* set and the *test* set. These datasets should be independent *of* but distributed *as* the training set. They can be created by splitting an initial dataset $\{(\boldsymbol{X}_j, \boldsymbol{Y}_j)\colon j = 1, \ldots, n + l + m\}$ into three parts that serve the following purposes:

- The training set $\mathcal{D}_n$ is used in the optimization part of the training process, to learn DNN parameters (weights and biases).
- The validation set $\mathcal{V}_l$ is used to reduce the generalization error of the training process. The validation loss indicates generalization error, since the validation set's samples are not *directly* used in training. The validation set is used for tuning DNN hyperparameters and monitoring training progress.
- The test set $\mathcal{T}_m$ is only used for evaluating a DNN that is trained using the definite hyperparameter settings. The performance of a DNN on the test set gives a more honest indication of generalization error than its performance on the validation set, because the DNN's (hyper)parameters are indirectly chosen for minimal validation loss.

Our primary loss function is the negative log-likelihood or (categorical) cross-entropy loss. It is widely used for training DNNs in supervised classification tasks. The NLL *training loss* is

$$\mathrm{NLL}(\widehat{\boldsymbol{p}} \mid \mathcal{D}_n) = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} Y_k^i \log \widehat{p}_k(\boldsymbol{X}_i), \tag{5}$$

while $\mathrm{NLL}(\widehat{\boldsymbol{p}} \mid \mathcal{V}_l) = (1/l)\sum_{i=1}^{l}\sum_{k=1}^{K} Y_k^i \log \widehat{p}_k(\boldsymbol{X}_i)$ is the *validation loss*. The expected training imperfection of a specific $\widehat{\boldsymbol{p}}$ compared to the hypothetical DNN in class $\mathcal{F}$ that obtains the lowest possible training loss is

$$\Delta_n(\widehat{\boldsymbol{p}} \mid \mathcal{D}_n) = \mathbb{E}_{\mathcal{D}_n}\left[-\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{Y}_i \log \widehat{\boldsymbol{p}}(\boldsymbol{X}_i) - \min_{\boldsymbol{q}\in\mathcal{F}} -\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{Y}_i \log \boldsymbol{q}(\boldsymbol{X}_i)\right].$$

The first risk function we introduce is the B-truncated Küllback-Leibler divergence risk

$$R_B(\widehat{\boldsymbol{p}}(\boldsymbol{X}), \boldsymbol{p}(\boldsymbol{X})) = \mathbb{E}\left[\mathrm{KL}_B(\widehat{\boldsymbol{p}}(\boldsymbol{X}), \boldsymbol{p}(\boldsymbol{X}))\right] = \mathbb{E}_{\boldsymbol{X}}\left[\boldsymbol{p}(\boldsymbol{X}) \max\left\{B, \log\left(\frac{\boldsymbol{p}(\boldsymbol{X})}{\widehat{\boldsymbol{p}}(\boldsymbol{X})}\right)\right\}\right]. \tag{6}$$

If B is chosen reasonably large, the B-truncated KL divergence risk is only different from the regular KL divergence risk, defined in (2), when $\widehat{\boldsymbol{p}}(\boldsymbol{X}) \ll \boldsymbol{p}(\boldsymbol{X})$. For instance, with $B = 2$, $\widehat{\boldsymbol{p}}(\boldsymbol{X})$ must be at least $e^2 \approx 7.39$ times larger than $\boldsymbol{p}(\boldsymbol{X})$ to reach the threshold where $B > \log(\boldsymbol{p}(\boldsymbol{X})/\widehat{\boldsymbol{p}}(\boldsymbol{X}))$. Truncation is useful for theoretical reasons, but later we *estimate* the (B-truncated) KL divergence risk with

$$\mathrm{KL}(\widehat{\boldsymbol{p}} \mid \mathcal{T}_m, \boldsymbol{p}) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} p_k(\boldsymbol{X}_i) \log\left(\frac{p_k(\boldsymbol{X}_i)}{\widehat{p}_k(\boldsymbol{X}_i)}\right). \tag{7}$$

Note that this estimate of risk depends on $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_m$ from the test set as well as the true conditional class probability function $\boldsymbol{p}$.

We also consider the estimated mean squared error (MSE) risk

$$\mathrm{MSE}(\widehat{\boldsymbol{p}} \mid \mathcal{T}_m, \boldsymbol{p}) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} \left(p_k(\boldsymbol{X}_i) - \widehat{p}_k(\boldsymbol{X}_i)\right)^2. \tag{8}$$

The proportion of small conditional class probabilities has a natural effect on the KL divergence risk, but not on the MSE risk. For example, let the true conditional class probability be 0.1 away from the estimated probabilities $(0.11, 0.89)$ in a binary classification setting. For the KL divergence, the (0.01, 0.99) true probability results in $0.01\log(0.01/0.11) + 0.99\log(0.99/0.89) \approx 0.082$. The (0.21, 0.79) true probability results in $0.21\log(0.21/0.11) + 0.79\log(0.79/0.89) \approx 0.042$. There is no effect on the MSE: $(0.21 - 0.11)^2 + (0.79 - 0.89)^2 = 0.02 = (0.01 - 0.11)^2 + (0.99 - 0.89)^2$.

2.1.3 Optimization

The parameters (weights and biases) of a DNN are optimized for minimal training loss using an iterative (stochastic) gradient descent-based algorithm during a training (or learning) process.

The parameters must first be initialized, usually with zeros as biases and small values as weights. Initialization plays a surprisingly important role in the performance of trained DNNs (Goodfellow, Bengio, and Courville 2016, beginning of §8.4). The initialization strategy should involve randomness in order to differentiate neurons from each other. This differentiation is stronger when initial values are sampled from a broad distribution. However, the amount of differentiation must not be too large, so that the parameter updates per training iteration have a similar effect on all neurons.

Glorot and Bengio (2010) suggest sampling all weights in the j^{th} layer from a uniform $\left(\left[-\sqrt{6/(m_j + m_{j+1})},\ \sqrt{6/(m_j + m_{j+1})}\right]\right)$ distribution. The goal of this layer width-dependent distribution is equal variance of activated neuron values and update gradients within layers. This equal variance is never achieved because of the nonlinearity of activation functions, but Glorot initialization works reasonably well in practice. He et al. (2015b) derive a weight initialization strategy for equal variance within layers as well, but they keep in mind the nonlinearity of, in particular, the ReLU activation function. They arrive at a normal distribution with mean zero and standard deviation $\sqrt{2/m_j}$. He et al. (2015b) found their strategy to outperform Glorot initialization on benchmark datasets, but both strategies are widely used.

After parameter initialization, a number of iterations of the following loop are performed:

1. Compute the loss of the DNN with current parameters.
2. Compute the gradient of the loss function with respect to the DNN's parameters.
3. Change parameters by a small amount, in order to reduce training loss.

The gradient of step 2 decides the direction in which parameters are changed in step 3. The amount of change is decided by a (stochastic) gradient descent-based optimization algorithm and its *learning rate* hyperparameter, η. Classic optimization algorithms base the parameter update on only the gradient and η, but variants like Kingma and Ba (2017)'s Adam optimizer take into account *momentum* from previous iterations.

The forward and backward passes in steps 1 and 2 are usually performed with batches instead of all samples of the training set. This is primarily done for a computational reason – training is quicker with many parameter update iterations based on a subset of data, rather than with fewer, more accurate parameter updates based on all data (Goodfellow, Bengio, and Courville 2016, §8.1.3). When using batches, we say we are using the *stochastic* version of gradient descent or one of its variants in step 3. When all the data has been used for parameter updates once, one *epoch* has taken place.

2.1.4 Regularization

The ultimate objective of supervised classification is not low training loss but low generalization error. *Regularization* methods are intended to decrease generalization error, through reducing *overfitting* on the training set.

If a DNN overfits, it has learned to classify the samples in the training set well, but classifies other samples from an identical distribution poorly. Overfitting can be diagnosed by a large discrepancy between training loss and validation or test loss. A DNN that *underfits* has not learned to approximate the general relationship either, but obtains high training loss as well.

The first regularization approach we discuss is early stopping. The simplest early stopping strategy is to limit the number of epochs, instead of letting training continue until training loss no longer decreases. Another early stopping strategy makes use of the validation loss, which can be calculated after every epoch. This second early stopping strategy halts training when validation loss no longer improves enough per epoch. The idea behind the strategy is that a network starts to overfit if its validation loss does not decrease anymore, regardless of training loss.

A more statistical approach to regularization is to add a parameter norm penalty, that discourages large parameter values, to the loss function. For instance, the negative log-likelihood training loss with L_1 regularization penalty $\lambda > 0$ on all weights (but *not* biases) is

$$\mathrm{NLL}_\lambda(\widehat{\boldsymbol{p}} \mid \mathcal{D}_n) = -\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{Y}_i \log \widehat{\boldsymbol{p}}(\boldsymbol{X}_i) + \lambda \sum_{j=1}^{m_{L+1}} \|\boldsymbol{W}^j\|_1. \tag{9}$$

The regularization penalty λ must not be too large or small. A value of λ that is too small leads to ineffective L_1 regularization and does not reduce overfitting; a value of λ that is too large leads to underfitting. With L_1 regularization, a DNN's parameters are encouraged to be small and often zero, as the *sparse* DNNs in $\mathfrak{F}$ should be.[3]

2.2 Small Value Bound

The small value bound (SVB) index $\alpha \geq 0$ is a measure for the proportion of small conditional class probabilities. A function class $\mathcal{H}$ is α-small value bounded if for all $\boldsymbol{p} = (p_1, \ldots, p_K) \in \mathcal{H}$ there is a $C > 0$ such that

$$\mathbb{P}_{\boldsymbol{X}}(p_k(\boldsymbol{X}) \leq t) \leq Ct^{\alpha} \quad \text{for all } t \in (0,1] \text{ and } k \in \{1, \ldots, K\}. \tag{10}$$

In our context, the expression $\mathbb{P}_{\boldsymbol{X}}(p_k(\boldsymbol{X}) \leq t)$ represents the proportion of conditional probabilities of class k that are smaller than or equal to t under the distribution of $\boldsymbol{X}$. This means $\mathbb{P}_{\boldsymbol{X}}(p_k(\boldsymbol{X}) \leq t)$ is the same as $\mathbb{E}_{\boldsymbol{X}}[\mathbb{1}_{\{\boldsymbol{x}:\, p_k(\boldsymbol{x}) \leq t\}}(\boldsymbol{X})]$.

With a tighter small value bound, a higher proportion of conditional class probabilities is away from zero. A *larger* index α implies a *tighter* SVB, because a *larger* α implies a *smaller* t^{α}. Figure 2 displays t^{α} for a few values of α.

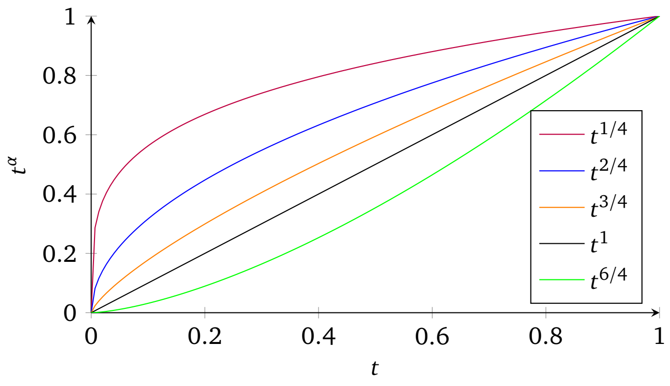


Figure 2: The influence of a few values of α on t^{α}.

Example 2 demonstrates how the α-SVB index can be calculated. More on the calculation of the α-SVB index follows later in this work.

Example 2

This example demonstrates the calculation of small value bound index α in a simple $X \sim \text{uniform}([0,1])$ and $\boldsymbol{p}(x) = (x^5,\ 1 - x^5)$ setting. Figure 3 displays this setting. Filling in definition (10) with p_1 and p_2 shows that $\boldsymbol{p}$ is small value bounded with index $\alpha = 1/5$ and $C = 1$:

$$\begin{aligned}
\mathbb{P}_X(p_1(X) \leq t) &= \mathbb{P}_X(X^5 \leq t) = \mathbb{P}_X(X \leq t^{1/5}) = t^{1/5}; \\
\mathbb{P}_X(p_2(X) \leq t) &= \mathbb{P}_X(1 - X^5 \leq t) = \mathbb{P}_X(X^5 > 1 - t) = 1 - \mathbb{P}_X(X^5 \leq 1 - t) \\
&= 1 - \mathbb{P}_X(X \leq (1-t)^{1/5}) = 1 - (1-t)^{1/5} \leq t^{1/5}.
\end{aligned}$$

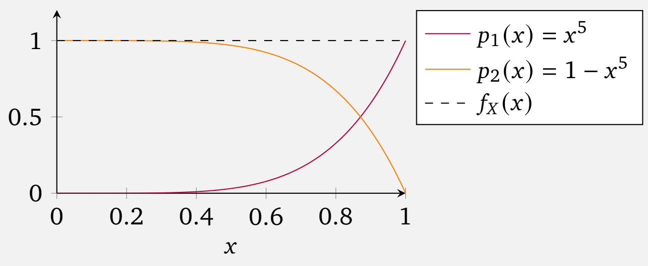


Figure 3: The conditional class probability function $\boldsymbol{p}(x) = (x^5,\ 1 - x^5)$ and probability density function f_X of $X \sim \text{uniform}([0,1])$.

[3] Although to get closer to DNNs in $\mathfrak{F}$, the biases would have to be regularized too. The negative log-likelihood with L_1 regularization penalty $\lambda > 0$ on all weights as well as biases is $-\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{Y}_i \log \widehat{\boldsymbol{p}}(\boldsymbol{X}_i) + \lambda \sum_{j=1}^{m_{l+1}} \left(\left\|W^j\right\|_1 + \left\|v^j\right\|_1 \right)$.

2.3 Hölder Smoothness

Before introducing Hölder smoothness, we clarify the remaining standard notation:

- We refer to the n^{th} derivative of a function f with respect to its input as $f^{(n)}$, whereas non-parenthesized superscripts are used for indexing or raising numbers to powers.
- We define floor $\lfloor x \rfloor := \max\{m \in \mathbb{Z} \colon m < x\}$ and ceiling $\lceil x \rceil := \min\{m \in \mathbb{Z} \colon m \geq x\}$. For instance, $\lfloor 3 \rfloor = \lfloor 2.4 \rfloor = 2$ and $\lceil 3 \rceil = \lceil 2.4 \rceil = 3$.

Smoothness intuitively means that the outcome of a function does not change drastically when input is changed subtly. Smoothness and differentiability are closely related.

The ball of functions with domain $D \subset \mathbb{R}^m$ that have Hölder smoothness index $\beta > 0$ with radius $Q \in \mathbb{R}$ is defined as

$$C^{\beta}(D, Q) = \left\{ g : D \to \mathbb{R} : \sum_{\gamma : \|\gamma\|_1 < \beta} \|\partial^{\gamma} g\|_{\infty} + \sum_{\gamma : \|\gamma\|_1 = \lfloor \beta \rfloor} \sup_{x, y \in D, x \neq y} \frac{|\partial^{\gamma} g(\boldsymbol{x}) - \partial^{\gamma} g(\boldsymbol{y})|}{\|\boldsymbol{x} - \boldsymbol{y}\|_{\infty}^{\beta - \lfloor \beta \rfloor}} \leq Q \right\}$$

where $\partial^{\gamma} = \partial^{\gamma_1} \ldots \partial^{\gamma_m}$, with $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_m) \in \mathbb{N}^m$. Functions in $C^{\beta}([0,1]^d, Q)$ are called β-smooth because the partial derivatives of orders up to β are bounded. This idea is made more concrete in the one-dimensional case in Example 3.

The ball of β-Hölder smooth conditional class probability functions with radius Q is

$$\mathcal{G}(\beta, Q) := \left\{ \boldsymbol{p} = (p_1(\boldsymbol{x}), \ \ldots, \ p_K(\boldsymbol{x})) : [0,1]^d \to [0,1]^K, p_k \in C^{\beta}\left([0,1]^d, Q\right) \right\}.$$

All functions in $\mathcal{G}(\beta, Q)$ that also satisfy the small value bound condition with index α for a given input distribution $\boldsymbol{X}$ belong to $\mathcal{G}_{\alpha}(\beta, Q)$.

Example 3

This example demonstrates Hölder smoothness index β and ball $C^{\beta}([0,1], Q)$ calculation for the simple function $g(x) = x$.

For one-dimensional functions with domain $[0,1]$ and $\beta > 0$, the ball of β-Hölder functions with radius $Q > 0$ is

$$C^{\beta}([0,1], Q) = \left\{ g \colon [0,1] \to \mathbb{R} \colon \sum_{j=0}^{\lfloor \beta \rfloor} \left\| g^{(j)} \right\|_{\infty} + \sup_{\substack{x,y \in [0,1] \\ x \neq y}} \frac{\left| g^{\lfloor \beta \rfloor}(x) - g^{\lfloor \beta \rfloor}(y) \right|}{|x - y|^{\beta - \lfloor \beta \rfloor}} \leq Q \right\}.$$

Filling in the terms that occur in the definition of $C^{\beta}([0,1], Q)$ gives

$$\sum_{j=0}^{\lfloor \beta \rfloor} \left\| g^{(j)} \right\|_{\infty} \leq \|g^{(0)}(x)\|_{\infty} + \|g^{(1)}(x)\|_{\infty} = \|x\|_{\infty} + \|1\|_{\infty} = 2;$$

$$\sup_{\substack{x,y \in [0,1] \\ x \neq y}} \frac{\left| g^{\lfloor \beta \rfloor}(x) - g^{\lfloor \beta \rfloor}(y) \right|}{|x - y|^{\beta - \lfloor \beta \rfloor}} = \begin{cases} \sup \frac{|x-y|}{|x-y|^{\beta}} \leq 1, & \lfloor \beta \rfloor = 0 \\ 0, & \lfloor \beta \rfloor \geq 0 \end{cases}.$$

$$\text{Therefore, } g \in \begin{cases} C^{\beta}([0,1], 3), & \beta \leq 1 \\ C^{\beta}([0,1], 2), & \beta > 1 \end{cases}.$$

Thus, an arbitrarily high Hölder smoothness index β condition holds for $g(x) = x$.

2.4 Main Risk Bound

We describe the theoretical result we complement with our simulation study in this section. It is proven as theorem 3.3 in Bos and Schmidt-Hieber (2021). We first define *rate*

$$\phi_n = \begin{cases} K^{\frac{(1+\alpha)\beta+(3+\alpha)d}{(1+\alpha)\beta+d}} n^{-\frac{(1+\alpha)\beta}{\beta(1+\alpha)+d}}, & \text{if } \alpha \in [0,1] \\ K^{\frac{2\beta+4d}{2\beta+d}} n^{-\frac{2\beta}{2\beta+d}}, & \text{if } \alpha > 1 \end{cases}, \tag{11}$$

where α is the small value bound index and β is the Hölder smoothness index. We elaborate on the definition of ϕ_n after stating the main risk bound.

Theorem 2.1 (main risk bound)**.** *Consider our classification model with $\boldsymbol{p} \in \mathcal{G}_\alpha(\beta, Q), \alpha \in [0,1]$, and $n > 1$. Let $\widehat{\boldsymbol{p}}$ be a deep neural network in $\mathcal{F}(L, \boldsymbol{m}, s)$ satisfying*

1. *$A(d,\beta)\log_2(n) \leq L \lesssim n\phi_n$,*
2. *$\min_{i=1,\cdots,L} m_i \gtrsim n\phi_n$,*
3. *$s \asymp n\phi_n \log n$*

for a suitable constant $A(d,\beta)$. If n is sufficiently large, then there exist constants C', C'' only depending on small value bound index α and constant C, Hölder smoothness index β, and input dimensionality d such that whenever training imperfection $\Delta_n(\widehat{\boldsymbol{p}} \mid \mathcal{D}_n) \leq C''B\phi_n L(\log n)^2$, then

$$R_B(\widehat{\boldsymbol{p}}(\boldsymbol{X}), \boldsymbol{p}(\boldsymbol{X})) \leq C'B\phi_n L(\log n)^2.$$

The main risk bound only holds for deep and wide DNNs that train on a "sufficiently large" dataset. Theoretical results are commonly derived to understand asymptotic behavior, rather than to minimize constants. It is therefore not surprising that the conditions under which the main risk bound holds seem strict. Theoretical results that are derived to understand asymptotic behavior typically hold under looser conditions in practice. However, a consequence of the involved constants is that only the convergence rate within a setting – rather than the absolute risk values between settings – can be studied.

The non-constant factor of theorem 2.1's bounds on the B-truncated Küllback-Leibler divergence risk and training imperfection is $\phi_n(\log n)^2$. The main risk bound thus implies that the convergence rate of the truncated KL divergence risk gets faster:

- as the small value bound index α gets closer to 1, but that no more is gained for $\alpha \geq 1$;
- as the Hölder smoothness index β increases.

Since β values much greater than 1 still have an effect on ϕ_n, the convergence rate can be dominated by β. Figure 4 illustrates the asymptotic behavior of $\phi_n(\log n)^2$ for a few α and β when $d = 1$.

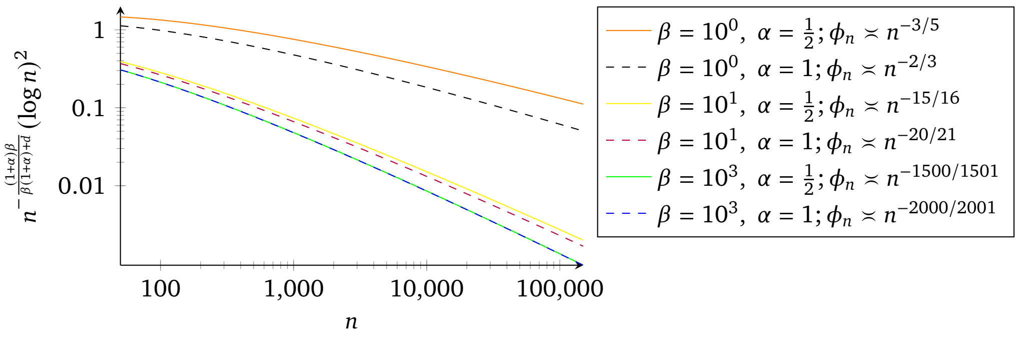


Figure 4: The asymptotic behavior of $\phi_n(\log n)^2$ for a few α and β when $d = 1$.

3 Simplifying SVB Calculations

The combination of a conditional class probability function $\boldsymbol{p}$ and input distribution $\boldsymbol{X}$ determines the small value bound index α that is satisfied per setting. We can simplify the calculation of the α-SVB index in many cases. Lemma 3.1 proves that if an α-SVB index condition is satisfied under a uniform distribution, it is satisfied under any continuous distribution with bounded density function.

We first note that satisfying an α-SVB condition implies satisfying an α'-SVB for $\alpha' \leq \alpha$. Since $t \in (0,1]$, we have for $\varepsilon > 0$ that $t^{\alpha+\varepsilon} \leq t^{\alpha}$. A larger SVB index $(\alpha + \varepsilon)$ thus indicates a tighter small value bound than an index α does.

Next, if $p(\boldsymbol{X})$ is never (arbitrarily close to) zero on the support of $\boldsymbol{X}$, we are in the $\alpha \geq 1$ case. In this case, there is a fixed $B_0 \in (0,1]$ such that $\mathbb{P}_{\boldsymbol{X}}(p(\boldsymbol{X}) \leq B_0) = 0$. Setting $C = B_0^{-\alpha}$ means $Ct^{\alpha} \geq 1$ for any $\alpha \geq 0$. Thus, (10) holds for any $\alpha \geq 0$.

The last basic point we make is: if there is a fixed $\tau \in (0,1)$ such that $\mathbb{P}_{\boldsymbol{X}}(p(\boldsymbol{X}) \leq t) \leq Ct^{\alpha}$ for all $t \in (0,\tau]$, the α-SVB condition is satisfied. This is because $C' = \max\{C, 1/\tau^{\alpha}\}$ makes $\mathbb{P}_{\boldsymbol{X}}(p(\boldsymbol{X}) \leq t) \leq C't^{\alpha}$ hold for $t > \tau$ as well. This property is useful when the behavior of $\mathbb{P}_{\boldsymbol{X}}(p(\boldsymbol{X}) \leq t)$ is different for small t than larger t – we only have to study the behavior for small t to determine the α-SVB index.

Lemma 3.1. *Let $\boldsymbol{X}$ be a continuous random variable with domain $[0,1]^d$ and probability density function $f_{\boldsymbol{X}}$. Let $\boldsymbol{U} \sim \text{uniform}([0,1]^d)$, $\boldsymbol{p}\colon [0,1]^d \to [0,1]$, and $t \in (0,1]$. If $f_{\boldsymbol{X}}$ is bounded away from zero and bounded from above – which is to say $\inf_{\boldsymbol{x}\in[0,1]^d} f_{\boldsymbol{X}}(\boldsymbol{x}) > 0$ and $\sup_{\boldsymbol{x}\in[0,1]^d} f_{\boldsymbol{X}}(\boldsymbol{x}) < \infty$ respectively – then $\mathbb{P}_{\boldsymbol{X}}(\boldsymbol{p}(\boldsymbol{X})) \asymp \mathbb{P}_{\boldsymbol{U}}(\boldsymbol{p}(\boldsymbol{U}))$.*

Proof. Since $f_{\boldsymbol{X}}$ is bounded away from zero,

$$\begin{aligned}\mathbb{P}_{\boldsymbol{X}}(\boldsymbol{p}(\boldsymbol{X}) \leq t) &= \mathbb{E}_{\boldsymbol{X}}[\mathbb{1}_{\{\boldsymbol{z}\colon\, p(\boldsymbol{z})\leq t\}}(\boldsymbol{X})] = \int \mathbb{1}_{\{\boldsymbol{x}\colon\, p(\boldsymbol{x})\leq t\}}\, f_{\boldsymbol{X}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \\ &\in \left[\inf_{\boldsymbol{x}} f_{\boldsymbol{X}}(\boldsymbol{x}) \int \mathbb{1}_{\{\boldsymbol{z}\colon\, p(\boldsymbol{z})\leq t\}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x},\ \sup_{\boldsymbol{x}} f_{\boldsymbol{X}}(\boldsymbol{x}) \int \mathbb{1}_{\{\boldsymbol{z}\colon\, p(\boldsymbol{z})\leq t\}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}\right].\end{aligned}$$

Furthermore, we know that

$$\mathbb{P}_{\boldsymbol{U}}(\boldsymbol{p}(\boldsymbol{u}) \leq t) = \mathbb{E}_{\boldsymbol{U}}[\mathbb{1}_{\{\boldsymbol{z}\colon\, p(\boldsymbol{z})\leq t\}}(\boldsymbol{U})] = \int \mathbb{1}_{\{\boldsymbol{z}\colon\, p(\boldsymbol{z})\leq t\}}(\boldsymbol{u})\, f_{\boldsymbol{U}}(\boldsymbol{u})\,\mathrm{d}\boldsymbol{u} = \int \mathbb{1}_{\{\boldsymbol{z}\colon\, p(\boldsymbol{z})\leq t\}}(\boldsymbol{u})\,\mathrm{d}\boldsymbol{u}.$$

Therefore

$$\inf_{\boldsymbol{x}} f_{\boldsymbol{X}}(\boldsymbol{x})\ \mathbb{P}_{\boldsymbol{U}}(\boldsymbol{p}(\boldsymbol{U}) \leq t) \leq \mathbb{P}_{\boldsymbol{X}}(\boldsymbol{p}(\boldsymbol{X}) \leq t) \leq \sup_{\boldsymbol{x}} f_{\boldsymbol{X}}(\boldsymbol{x})\ \mathbb{P}_{\boldsymbol{U}}(\boldsymbol{p}(\boldsymbol{U}) \leq t),$$

and thus $\mathbb{P}_{\boldsymbol{X}}(\boldsymbol{p}(\boldsymbol{X})) \asymp \mathbb{P}_{\boldsymbol{U}}(\boldsymbol{p}(\boldsymbol{U}))$, because $\inf_{\boldsymbol{x}} f_{\boldsymbol{X}}(\boldsymbol{x}) > 0$ and $\sup_{\boldsymbol{x}} f_{\boldsymbol{X}}(\boldsymbol{x}) < \infty$. □

Remark. Let $0 < \tau < 1$. There is a possibly tighter bound for $\mathbb{P}_{\boldsymbol{X}}(\boldsymbol{p}(\boldsymbol{X}) \leq t)$ around $\mathbb{P}_{\boldsymbol{U}}(\boldsymbol{p}(\boldsymbol{U}) \leq t)$ if $t \in (0,\tau]$:

$$\mathbb{P}_{\boldsymbol{X}}(\boldsymbol{p}(\boldsymbol{X}) \leq t) \in \left[\inf_{\boldsymbol{x}\colon\, p(\boldsymbol{x})\leq\tau} f_{\boldsymbol{X}}(\boldsymbol{x})\ \mathbb{P}_{\boldsymbol{U}}(\boldsymbol{p}(\boldsymbol{U}) \leq t),\ \sup_{\boldsymbol{x}\colon\, p(\boldsymbol{x})\leq\tau} f_{\boldsymbol{X}}(\boldsymbol{x})\ \mathbb{P}_{\boldsymbol{U}}(\boldsymbol{p}(\boldsymbol{U}) \leq t)\right].$$

△

Corollary 3.1.1 ($f_{\boldsymbol{X}}$ only bounded from above)**.** *Consider the case of the lemma, but with only $\sup_{\boldsymbol{x}\in[0,1]^d} f_{\boldsymbol{X}}(\boldsymbol{x}) < \infty$ and not $\inf_{\boldsymbol{x}\in[0,1]^d} f_{\boldsymbol{X}}(\boldsymbol{x}) > 0$. Then the $\alpha_{\boldsymbol{X}}$ in $\mathbb{P}_{\boldsymbol{X}}(\boldsymbol{p}(\boldsymbol{X}) \leq t) \lesssim t^{\alpha_{\boldsymbol{X}}}$ is at least the $\alpha_{\boldsymbol{U}}$ in $\mathbb{P}_{\boldsymbol{U}}(\boldsymbol{p}(\boldsymbol{U}) \leq t) \lesssim t^{\alpha_{\boldsymbol{U}}}$.*

Proof. We know $\mathbb{P}_{\boldsymbol{X}}(\boldsymbol{p}(\boldsymbol{X}) \leq t) \leq \sup_{\boldsymbol{x}\in[0,1]^d} f_{\boldsymbol{X}}(\boldsymbol{x})\ \mathbb{P}_{\boldsymbol{U}}(\boldsymbol{U}) \leq t)$, and therefore $t^{\alpha_{\boldsymbol{X}}} \lesssim t^{\alpha_{\boldsymbol{U}}}$. Since $t \in (0,1]$, that means $\alpha_{\boldsymbol{X}} \geq \alpha_{\boldsymbol{U}}$. □

Remark ($f_{\boldsymbol{X}}$ only bounded from below). Similarly, we have $\alpha_{\boldsymbol{U}} \geq \alpha_{\boldsymbol{X}}$ if we assume $\inf_{\boldsymbol{x}\in[0,1]^d} f_{\boldsymbol{X}}(\boldsymbol{x}) > 0$ but not $\sup_{\boldsymbol{x}\in[0,1]^d} f_{\boldsymbol{X}}(\boldsymbol{x}) < \infty$. △

4 Simulation Methodology

4.1 Scenarios

4.1.1 Binary Classification With One-Dimensional Input

In this section, we introduce the exact settings for our simulation study. We call these settings *scenarios*. Scenarios are characterized by a combination of a distribution of input $\boldsymbol{X}$ and a conditional class probability function $\boldsymbol{p}$. We first consider binary classification with one-dimensional input scenarios, and later multiclass classification and multidimensional input scenarios.

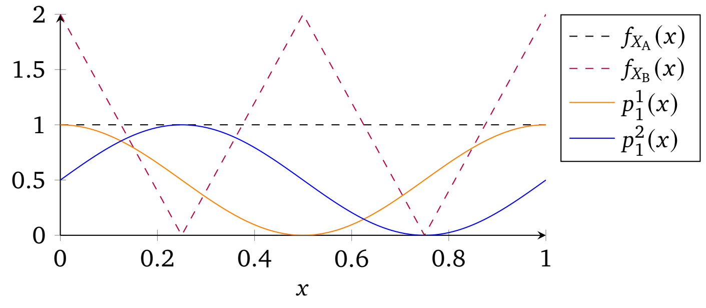


Figure 5: The probability density functions of X_A and X_B, as well as p_1^1 and p_1^2.

The scenarios with one-dimensional input ($d = 1$) have one of the following input distributions:

A. The well-known $X_A \sim \text{uniform}([0, 1])$. This random variable has probability density function $f_{X_A}(x) = 1$ if $x \in [0, 1]$ and $f_{X_A}(x) = 0$ if $x \notin [0, 1]$.

B. A random variable $X_B\colon [0, 1] \to [0, 1]$ whose probability density function resembles two pyramids with maxima $f_{X_B}(0) = f_{X_B}(1/2) = f_{X_B}(1) = 2$ and minima $f_{X_B}(1/4) = f_{X_B}(3/4) = 0$. The cumulative distribution function of X_B is defined in (12) later.

The binary classification ($K = 2$) scenarios have one of the following conditional class probability functions:

1. Squashed cosines with domain and range $[0, 1]$ and period 1:

$$\boldsymbol{p}_1(x) = \left(\frac{1 + \cos(2\pi x)}{2}, \frac{1 - \cos(2\pi x)}{2} \right).$$

2. Squashed sines with domain and range $[0, 1]$ and period 1:

$$\boldsymbol{p}_2(x) = \left(\frac{1 + \sin(2\pi x)}{2}, \frac{1 - \sin(2\pi x)}{2} \right).$$

Figure 5 displays the densities of X_A and X_B, as well as p_1^1 and p_1^2. It is convenient to denote the first and second component of $\boldsymbol{p}_j$ by p_1^j and p_2^j respectively. For example, the function $p_1^2\colon [0, 1]^d \to [0, 1]$ is defined by $p_1^2(x) = (1 + \sin(2\pi x))/2$. The subscript "$_1$" in $p_1^2(x)$ indicates that we are referring to the probability of class $k = 1$ given input x; the superscript "^{2}" indicates that we are considering the conditional class probability function of squashed sines, $\boldsymbol{p}_2$.

We refer to individual scenarios by a combination of the subscripted letter of the input distribution and the subscripted number of the conditional class probability function. For example, scenario B2 refers to the combination of X_B and $\boldsymbol{p}_2$.

We calculate the relevant quantities for the main risk bound before running any simulations. One of the quantities that occurs in the definition of convergence rate-determining ϕ_n – (11) – is the α-SVB index. We now calculate which α-SVB condition holds per scenario.

Lemma 4.1. *In scenarios A1 and A2, the* $1/2$*-SVB condition holds.*

Proof. For the $1/2$-SVB condition to hold, we must have $\mathbb{P}_{X_A}(p_k^j(X_A) \le t) \lesssim t^{1/2} = \sqrt{t}$ for all $k, j \in \{1, 2\}$ and $t \in (0, 1]$.

We examine p_1^1 of scenario A1 first. Its small outcomes occur around the root $p_1^1(1/2) = 0$. The third-order Taylor series expansion around $1/2$ is $T(x) = \pi^2(x - 1/2)^2$. The remainder is $R(x) =$

$(\pi^4/3)\cos(2\pi\xi)(x-1/2)^4$ for some ξ between $1/2$ and x. Because $x \in [0,1]$ and therefore $(x-1/2)^2 \geq (x-1/2)^4$, there is a $C' > 0$ such that $C'(x-1/2)^2 \leq T(x) - \|R(x)\|_\infty \leq p_1^1(x)$. Thus,

$$\mathbb{P}_{X_\mathrm{A}}\left(p_1^1(X_\mathrm{A}) \leq t\right) \leq \mathbb{P}_{X_\mathrm{A}}\left(C'\left(X_\mathrm{A} - \frac{1}{2}\right)^2 \leq t\right) = \mathbb{P}_{X_\mathrm{A}}\left(-\sqrt{\frac{t}{C'}} + \frac{1}{2} \leq X_\mathrm{A} \leq \sqrt{\frac{t}{C'}} + \frac{1}{2}\right).$$

This last term is equal to $\sqrt{t/C'} + 1/2 - \left(-\sqrt{t/C'} + 1/2\right) = 2\sqrt{t/C'}$ for t such that $\sqrt{t/C'} + 1/2 \leq 1$, i.e., when $t \leq C'/4$. If we fix any $\tau \in (0, C'/4]$, then $\mathbb{P}_{X_\mathrm{A}}\left(p_1^1(X_\mathrm{A}) \leq t\right) \leq C\sqrt{t}$ for *all* $t \in (0,1]$ with $C = \max\left\{2/\sqrt{C'}, 1/\sqrt{\tau}\right\}$. We can see that the general method applied last works in this case because

$$C = 2/\sqrt{C'} \implies C\sqrt{t} = 2\sqrt{t/C'} \geq 1 \text{ for all } t \geq C'/4. \text{ For instance, } 2\sqrt{(C'/4)/C'} = 1.$$

We examine p_2^1 next, to finish scenario A1. There are two roots, $r_1 = 0$ and $r_2 = 1$; the third order Taylor series expansion is for both $j \in \{1,2\}$ again $\pi^2(x-r_j)^2$, with remainder $(\pi^4/3)\cos(2\pi\xi(x-r_j)^4$ for some ξ between r_j and x. The same reasoning as for p_1^1 applies: there is a $C' > 0$ such that $\mathbb{P}_{X_\mathrm{A}}\left(p_2^1(X_\mathrm{A}) \leq t\right) \leq \mathbb{P}_{X_\mathrm{A}}\left(C'(X_\mathrm{A} - r_j)^2 \leq t\right)$ for some $t > 0$, and thus $\mathbb{P}_{X_\mathrm{A}}\left(p_2^1(X_\mathrm{A}) \leq t\right) \lesssim \sqrt{t}$ for all $t \in (0,1]$.

We examine scenario A2 now, where the roots are $p_1^2(3/4) = 0$ and $p_2^2(1/4) = 0$. For $j \in \{1,2\}$, the third-order Taylor series expansions of $p_j^2(x)$ around $r_1 = 3/4$ and $r_2 = 1/4$ are again $\pi^2(x-r_j)^2$ with remainders $-\pi^4/3\sin(2\pi\xi)(x-r_j)^4$ for some ξ between r_j and x. The same reasoning as for p_1^1 applies for both $j \in \{1,2\}$: there is a $C_j' > 0$ such that $\mathbb{P}_{X_\mathrm{A}}(p_j^2(X_\mathrm{A}) \leq t) \leq \mathbb{P}_{X_\mathrm{A}}(C_j'(X_\mathrm{A} - r_j)^2$ for some $t > 0$, and thus $\mathbb{P}_{X_\mathrm{A}}\left(p_j^2(X_\mathrm{A}) \leq t\right) \lesssim \sqrt{t}$ for all $t \in (0,1]$. □

Lemma 4.2. *In scenario B1, the* $1/2$*-SVB condition holds; in B2, the* 1*-SVB condition holds.*

Proof. For the respective α-SVB conditions to hold, we must have $\mathbb{P}_{X_\mathrm{B}}(p_k^1(X_\mathrm{B}) \leq t) \lesssim t^{1/2}$ and $\mathbb{P}_{X_\mathrm{B}}(p_k^2(X_\mathrm{B}) \leq t) \lesssim t$ for all $k \in \{1,2\}$ and $t \in (0,1]$.

The input distribution X_B has the cumulative distribution function

$$F_{X_\mathrm{B}}(x) = \begin{cases} 0 & x < 0 \\ -4x^2 + 2x & 0 \leq x \leq 1/4 \\ 4x^2 - 2x + 0.5 & 1/4 < x \leq 1/2 \\ -4x^2 + 6x - 1.5 & 1/2 < x \leq 3/4 \\ 4x^2 - 6x + 3 & 3/4 < x \leq 1 \\ 1 & x > 1 \end{cases}. \tag{12}$$

We examine scenario B1 first. The only root of p_1^1 is $1/2$ (because $p_1^1(x) = 0 \implies x = 0$). We make use of a lower bound $C'(X_\mathrm{B} - 1/2)^2 \leq p_1^1(X_\mathrm{B})$ for a $C' > 0$ as in Lemma 4.1. Using the two cases of F_{X_B} that suffice for $1/4 < \sqrt{t/C'} + 1/2 \leq 3/4 \implies t \leq C'/16$, we have:

$$\begin{aligned}\mathbb{P}_{X_\mathrm{B}}\left(p_1^1(X_\mathrm{B}) \leq t\right) \leq \mathbb{P}_{X_\mathrm{B}}\left(C'\left(X_\mathrm{B} - \frac{1}{2}\right)^2 \leq t\right) &= \mathbb{P}_{X_\mathrm{B}}\left(-\sqrt{\frac{t}{C'}} + \frac{1}{2} \leq X_\mathrm{B} \leq \sqrt{\frac{t}{C'}} + \frac{1}{2}\right) \\ &= -4\left(\sqrt{\frac{t}{C'}} + \frac{1}{2}\right)^2 + 6\left(\sqrt{\frac{t}{C'}} + \frac{1}{2}\right) - 1.5 \\ &\quad - \left(4\left(-\sqrt{\frac{t}{C'}} + \frac{1}{2}\right)^2 - 2\left(-\sqrt{\frac{t}{C'}} + \frac{1}{2}\right) + 0.5\right) \\ &= \frac{4\sqrt{t}}{\sqrt{C'}} - \frac{8t}{C'},\end{aligned}$$

which is less than $\sqrt{16/C'}\sqrt{t}$, for example. Thus, $\mathbb{P}_{X_\mathrm{B}}\left(p_1^1(X_\mathrm{B}) \leq t\right) \lesssim \sqrt{t}$ for all $t \in (0,1]$.

The first root of p_2^1 we consider is 0. With a new $C' > 0$ and the two cases of F_{X_B} that suffice for $t \leq C'/16$, we have:

$$\mathbb{P}_{X_B}\left(p_2^1(X_B) \leq t\right) \leq \mathbb{P}_{X_B}(C'(X_B - 0)^2 \leq t) = -4\left(\sqrt{\frac{t}{C'}} + 0\right)^2 + 2\left(\sqrt{\frac{t}{C'}} + 0\right) - 0 = \frac{2\sqrt{t}}{\sqrt{C'}} + \frac{4t}{C'}$$

For the second root, 0, and with a new $C' > 0$:

$$\mathbb{P}_{X_B}\left(p_2^1(X_B) \leq t\right) \leq \mathbb{P}_{X_B}(C'(X_B - 1)^2 \leq t) = 1 - \left(4\left(\sqrt{\frac{t}{C'}} + 1\right)^2 - 6\left(\sqrt{\frac{t}{C'}} + 1\right) + 3\right) = \frac{2\sqrt{t}}{\sqrt{C'}} - \frac{4t}{C'}.$$

Thus, $\mathbb{P}_{X_B}\left(p_2^1(X_B) \leq t\right) \lesssim \sqrt{t}$ for all $t \in (0, 1]$.

We examine scenario B2 now. With the lower bound and the two F_{X_B} cases around the root 3/4 of p_1^2 that suffice for $t \leq C'/16$, we have for a $C' > 0$:

$$\begin{aligned}\mathbb{P}_{X_B}\left(p_1^2(X_B) \leq t\right) \leq \mathbb{P}_{X_B}\left(C'\left(X_B - \frac{3}{4}\right)^2 \leq t\right) &= 4\left(\sqrt{\frac{t}{C'}} + \frac{3}{4}\right)^2 - 6\left(\sqrt{\frac{t}{C'}} + \frac{3}{4}\right) + 3 \\ &\quad - \left(-4\left(-\sqrt{\frac{t}{C'}} + \frac{3}{4}\right)^2 + 6\left(-\sqrt{\frac{t}{C'}} + \frac{3}{4}\right) - 1.5\right) \\ &= \frac{8t}{C'}.\end{aligned}$$

And thus, $\mathbb{P}_{X_B}\left(p_1^2(X_B) \leq t\right) \lesssim t$ for all $t \in (0, 1]$.

Finally, we examine p_2^2 with root 1/4. There is a $C' > 0$ such that:

$$\begin{aligned}\mathbb{P}_{X_B}\left(p_2^2(X_B) \leq t\right) \leq \mathbb{P}_{X_B}\left(C'\left(X_B - \frac{1}{4}\right)^2 \leq t\right) &= 4\left(\sqrt{\frac{t}{C'}} + \frac{1}{4}\right)^2 - 2\left(\sqrt{\frac{t}{C'}} + \frac{1}{4}\right) + \frac{1}{2} \\ &\quad - \left(-4\left(-\sqrt{\frac{t}{C'}} + \frac{1}{4}\right)^2 + 2\left(-\sqrt{\frac{t}{C'}} + \frac{1}{4}\right)\right) \\ &= \frac{8t}{C'}.\end{aligned}$$

And thus, $\mathbb{P}_{X_B}\left(p_2^2(X_B) \leq t\right) \leq \mathbb{P}_{X_B}(C'(X_B - 1/4)^2 \leq t) \lesssim t$. □

In scenarios A1, A2, and B1, the 1/2-SVB condition holds and results in

$$\phi_n \asymp n^{-\frac{(1+\alpha)\beta}{\beta(1+\alpha)+d}} = n^{-\frac{(1+1/2)\beta}{\beta(1+1/2)+1}} = n^{-3\beta/(3\beta+2)}.$$

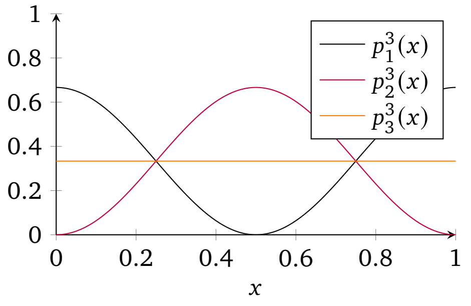
In scenario B2, the 1-SVB condition holds and results in a faster convergence rate, $\phi_n \asymp n^{-2\beta/(2\beta+1)}$. The quantity other than SVB index α that occurs in ϕ_n is Hölder smoothness index β. Since $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ both have arbitrarily large Hölder smoothness index β, the convergence rate is n^{-1} in all binary classification with one-dimensional input scenarios.

4.1.2 Multiclass Classification and Multidimensional Input

Our multiclass classification ($K = 2$) scenarios have one of the following conditional class probability functions:

$$\begin{aligned}\boldsymbol{p}_3(x) &= \left(\frac{1 + \cos(2\pi x)}{3},\ \frac{1 - \cos(2\pi x)}{3},\ \frac{1}{3}\right), \\ \boldsymbol{p}_4(x) &= \left(\frac{2x}{3},\ \frac{2(1 - x)}{3},\ \frac{1}{3}\right).\end{aligned}$$

Figure 6 displays $\boldsymbol{p}_3$ and $\boldsymbol{p}_4$. We consider $\boldsymbol{p}_3$ and $\boldsymbol{p}_4$ in combination with input distribution $X_A \sim$ uniform([0, 1]) and refer to the resulting scenarios as A3 and A4.



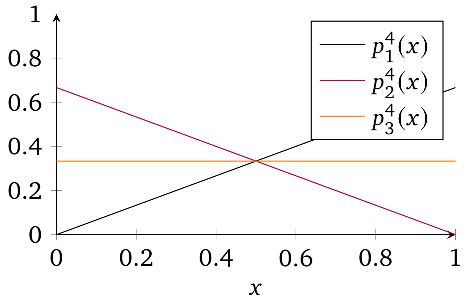


Figure 6: The conditional class probability functions $\boldsymbol{p}_3$ (left) and $\boldsymbol{p}_4$ (right).

Lemma 4.3. *In scenario A3, the* $1/2$*-SVB condition holds; in A4, the* 1*-SVB condition holds.*

Proof. For the respective α-SVB conditions to hold, we must have $\mathbb{P}_{X_A}(p_k^3(X_A) \le t) \lesssim t^{1/2}$ and $\mathbb{P}_{X_A}(p_k^3(X_A) \le t) \lesssim t$ for all $k \in \{1,2\}$ and $t \in (0,1]$.

We consider the A3 scenario first. Note that $p_1^1(x) = (3/2)p_1^3(x)$ and $p_2^1(x) = (3/2)p_2^3(x)$. We can therefore make use of the findings of Lemma 4.1, albeit with different constants: $\mathbb{P}_{X_A}(p_1^3(X_A) \le t) \lesssim \sqrt{t}$ and $\mathbb{P}_{X_A}(p_2^3(X_A) \le t) \lesssim \sqrt{t}$. Since p_3^3 is constant, $\mathbb{P}_{X_A}(p_3^3(X_A) = 1/3 \le t) \lesssim \sqrt{t}$ for all $t \in (0,1]$ as well.

In the A4 scenario, we have for all $t \in (0,1]$ the relatively simple:

- $\mathbb{P}_{X_A}\left(p_1^4(X_A) = 2X_A/3 \le t\right) = (3/2)t \lesssim t$;
- $\mathbb{P}_{X_A}\left(p_2^4(X_A) = 2(1-X_A)/3 \le t\right) = \mathbb{P}_{X_A}(X_A \ge 1 - 3t/2) = 1 - (1 - 3t/2) = (3/2)t \lesssim t$;
- $\mathbb{P}_{X_A}\left(p_3^4(X_A) = 1/3 \le t\right) \lesssim t$. □

Finally, we introduce scenarios in which there is multidimensional input ($d = 2$). These scenarios use the input distribution

$$\boldsymbol{X}_C \sim \text{uniform}([0,1]^2)$$

and have one of the following conditional class probability functions:

$$\boldsymbol{p}_5(\boldsymbol{x}) = \left(\left(\frac{x_1+x_2}{2}\right)^2,\ 1-\left(\frac{x_1+x_2}{2}\right)^2\right),$$
$$\boldsymbol{p}_6(\boldsymbol{x}) = \left(\left(\frac{x_1+x_2}{2}\right)^4,\ 1-\left(\frac{x_1+x_2}{2}\right)^4\right).$$

Lemma 4.4. *In scenario C5, the* 1*-SVB condition holds; in C6, the* $1/2$*-SVB condition holds.*

Proof. For the respective α-SVB conditions to hold, we must have $\mathbb{P}_{\boldsymbol{X}_C}(p_k^5(\boldsymbol{X}_C) \le t) \lesssim t$ and $\mathbb{P}_{\boldsymbol{X}_C}(p_k^6(\boldsymbol{X}_C) \le t) \lesssim t^{1/2}$ for all $k \in \{1,2\}$ and $t \in (0,1]$.

Let $Z = X_1 + X_2$ where $X_1, X_2 \sim \text{uniform}([0,1])$, so that Z represents the sum of the individual random variables in the random vector $\boldsymbol{X}_C$. We use a convolution to define the cumulative distribution function

$$F_Z(z) = \int_0^1 F_{X_1}(z-x)\ f_{X_1}(x_1)\ \mathrm{d}x = \begin{cases} 0, & z < 0 \\ z^2/2, & 0 < z \le 1 \\ -z^2/2 + 2z - 1, & 1 < z \le 2 \\ 1, & z > 2 \end{cases}.$$

We examine p_1^5 first. The expression $\mathbb{P}_{\boldsymbol{X}_C}(p_1^5(\boldsymbol{X}_C) \le t)$ can be rewritten as $\mathbb{P}_Z((Z/2)^2 \le t) = \mathbb{P}_Z(-2\sqrt{t} \le Z \le 2\sqrt{t})$. This means $\mathbb{P}_{\boldsymbol{X}_C}(p_1^5(\boldsymbol{X}_C) \le t) = (2\sqrt{t})^2/2 - 0 = 2t$ for $t \le 1/4$ and thus $\mathbb{P}_{\boldsymbol{X}_C}(p_1^5(\boldsymbol{X}_C) \lesssim t$ for all $t \in (0,1]$.

We examine p_1^6 next. Similarly to just above, $\mathbb{P}_{\boldsymbol{X}_C}(p_1^6(\boldsymbol{X}_C) \le t) = \mathbb{P}_Z(-2t^{1/4} \le Z \le 2t^{1/4}) = (2t^{1/4})^2/2 - 0 = 2\sqrt{t}$ for $t \le 1/16$. Thus, $\mathbb{P}_{\boldsymbol{X}_C}(p_1^6(\boldsymbol{X}_C) \le t) \lesssim \sqrt{t}$ for all $t \in (0,1]$.

We now examine p_2^5. The expression $\mathbb{P}_{\boldsymbol{X}_C}(p_2^5(\boldsymbol{X}_C) \le t)$ is equivalent to

$$\mathbb{P}_Z(1-(Z/2)^2 \le t) = \mathbb{P}_Z\left(Z \ge 2\sqrt{1-t}\right) = 1 - \mathbb{P}_Z\left(Z \le 2\sqrt{1-t}\right),$$

which for $t \in (0, 1/4)$ is

$$1 - \left(-\frac{\left(2\sqrt{1-t}\right)^2}{2} + 2\left(2\sqrt{1-t}\right) - 1 \right) = 2 - 4\sqrt{1-t} + 2(t-1).$$

The second-order Taylor series expansion of $2 - 4\sqrt{1-t} + 2(t-1)$ around 0 is $T(t) = t^2/2$ with remainder $R(t) = (t^3/3!)3/\left(2(1-\xi)^{5/2}\right) \leq t^3/2$ for some $\xi \leq t$. Hence, there is a $C' > 0$ such that $C't^2 \geq T(t) + \|R(t)\|_\infty \geq 2 - 4\sqrt{1-t} + 2(t-1)$. Thus, $\mathbb{P}_{X_C}(p_2^5(X_C) \leq t) \leq 2 - 4\sqrt{1-t} + 2(t-1) \leq C't^2$ for $t \in (0, 1/4)$, and $\mathbb{P}_{X_C}(p_2^5(X_C) \leq t) \lesssim t^2 \leq t$ for all $t \in (0, 1]$.

Finally, we examine p_2^6. Similarly to above, $\mathbb{P}_{X_C}(p_2^6(X_C) \leq t) = 1 - \mathbb{P}_Z\left(Z \leq 2(1-t)^{1/4}\right)$, which is $2 - 4(1-t)^{1/4} + 2(t-1)^{1/2}$ for $t \in (0, 1/16)$. The second-order Taylor series expansion of $2 - 4(1-t)^{1/4} + 2(t-1)^{1/2}$ around 0 is $T(t) = t^2/8$ with remainder $R(t) = (t^3/3!)((21 - 12(1-\xi)^{1/4})/(16(1-\xi)^{11/4})) \leq t^3/8$ for some $\xi \leq t \leq 1/16$. Hence, there is a $C' > 0$ such that $C't^2 \geq T(t) + \|R(t)\|_\infty \geq 2 - 4(1-t)^{1/4}2(t-1)^{1/2}$. Thus, $\mathbb{P}_{X_C}(p_2^5(X_C) \leq t) \leq 2 - 4(1-t)^{1/4} + 2(t-1)^{1/2} \leq C't^2$ for $t \in (0, 1/16)$, and $\mathbb{P}_{X_C}(p_2^6(X_C) \leq t) \lesssim t^2 \leq \sqrt{t}$ for all $t \in (0, 1]$. □

The Hölder smoothness index β of $\boldsymbol{p}_3, \boldsymbol{p}_4, \boldsymbol{p}_5$, and $\boldsymbol{p}_6$ is arbitrarily large. Based on the main risk bound, we thus have KL divergence risk convergence rate $\phi_n \asymp n^{-1}$ in scenarios A3, A4, C5, and C6.

4.2 Data Generation

The distribution of the datasets we generate follows from the distribution of input, $\boldsymbol{X}$, and the conditional class probability function, $\boldsymbol{p}$, of each scenario. We sample the label that determines $\boldsymbol{Y}_j$ from a categorical distribution with probability vector $\boldsymbol{p}(\boldsymbol{X}_j)$, where $p_k(\boldsymbol{x}) = \mathbb{P}_{\boldsymbol{Y}|\boldsymbol{X}}(Y_k = 1 \mid \boldsymbol{X} = \boldsymbol{x})$.

To sample from X_B – an uncommon distribution with pyramidal density – we use rejection sampling. The purpose of rejection sampling is to uniformly sample from the region under the curve of the target probability density function f_{X_B}. Rejection sampling uses candidate samples from a scaled version of a *proposal* distribution for which there are implemented sampling methods, such as the uniform.

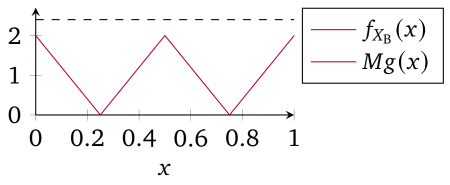


Figure 7: Rejection sampling illustration, with $f_{X_B}(x)$ the density of X_B, $g(x)$ the density of a uniform$([0, 1])$ RV, and $M = 2.4$.

Rejection sampling works as follows. Let the proposal density $g(x)$ be the density of a uniform$([0, 1])$ RV. We require $f_{X_B}(x) < Mg(x)$ and choose M. Since $f_{X_B}(x) \in [0, 2]$ and $g(x) \in [0, 1]$, we can choose $M = 2.4$ (or any other $M > 2$). Figure 7 illustrates the idea of getting f_{X_B} entirely under the curve of Mg. After these preliminary choices, rejection sampling works as follows:

1. Sample $x_j \sim$ [the random variable with density g].
2. Sample $u_j \sim$ uniform$([0, 1])$.
3. If $u_j < f_{X_B}(x_j)/(Mg(x_j))$, accept x_j as sample from X_B. Else, reject x_j, and return to step 1. This means, for instance, that $x_j = 0.5$ is accepted with probability $2/2.4 \approx 0.83$ and that $x_j = 0.3$ is accepted with probability $0.4/2.4 \approx 0.17$.

The rejection sampling method extends to any number of dimensions and works with all target densities. The downside of the method is that the expected proportion of proposed x_j that is accepted is $1/M$, which can be much smaller than 1. The resulting extra operations are inconsequential for us, since sampling is a trivial part of our total computational cost.

4.3 Our ReLU and Softmax DNNs

4.3.1 Implementation

For the implementation of deep neural networks, we use the Keras and TensorFlow packages in Python. In particular, our DNNs are Keras `models.Sequential` models, consisting of `layers.Dense` layers. In `Dense` layers, every neuron has a bias, and every connection has a weight. Neurons in the first (non-hidden) layer receive the network's input. Neurons in every other layer receive input from all neurons in the preceding layer. This corresponds to our DNN definition, (3). Our `Dense` layers:

1. use the rectified linear unit (ReLU) activation function;
2. have initial bias values zero, and weights sampled from He et al. (2015a)'s normal distribution with mean 0 and standard deviation $\sqrt{2/m_j}$ for neurons in the j^{th} layer;
3. have L_1 regularization on weights.

This corresponds to `Dense` layer arguments: 1. `activation = "relu"`; 2. `kernel_initializer = "he_normal"`; 3. `kernel_regularizer = regularizers.l1(l1)`, where variable `l1` represents the L_1 regularization penalty λ.

The resulting DNNs *can* be members of the $\mathcal{F}(L, \boldsymbol{m}, s)$ class, but biases are not regularized and parameters are not constrained to $[-1, 1]$. The definition of $\mathcal{F}$, (4), suggests both. Both additions can be made to the `Dense` layers with the arguments `bias_regularizer = regularizers.l1(l1)`, `kernel_constraint = constraints.max_norm(1.0)`, and `bias_constraint = constraints.max_norm(1.0)`. We have chosen to leave out bias regularization and parameter magnitude constraints because DNNs with said additions perform significantly worse in informal experiments.

Our first `Dense` layer has an `input_shape` m_0, and consists of m_1 neurons. Then follow $L-2$ of our (hidden) `Dense` layers with $m_2, \ldots, m_{L-1}$ neurons, respectively. The final `Dense` layer has m_L neurons, but a few different properties from the other layers. The activation function in this final layer is the softmax, and the initial weight values are sampled from a Glorot uniform. This corresponds to `Dense` layer arguments `activation = "softmax"` and `kernel_initializer = "glorot_uniform"`.

We use Kingma and Ba (2017)'s Adam optimizer to minimize the negative log-likelihood training loss (we define the latter in (5); it is named `categorical_crossentropy` in Keras). Adam hyperparameters other than learning rate η are left on `optimizers.Adam`'s defaults. The maximum number of training epochs is 500. Batch size is 128. Training stops early when (NLL) validation loss has failed to improve by more than 0.005 over 50 epochs. Early stopping usually occurs long before 500 epochs have occurred.[4] The weights and biases that obtain the best *validation loss* are chosen as final DNN parameters.

4.3.2 Demonstration

In this section, we demonstrate that DNNs are able to approximate the true conditional class probabilities in our simulation settings. Figures 8 and 9 demonstrate DNN performance in the B2 and A3 scenarios (using hyperparameter values that are specified later). The visualizations and estimated risks throughout this section are intended to make the later numerical reporting more interpretable.

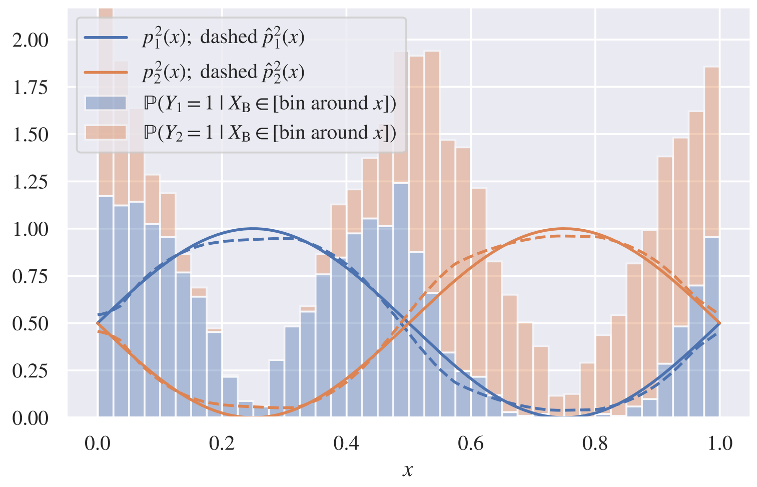


Figure 8: DNN test set performance in scenario B2. The class distribution of the 4096-sized training sets is shown in the histogram. The full lines display true $p_k(x_{(1)}), \ldots, p_k(x_{(m)})$, while the dashed lines display learned $\widehat{p}_k(x_{(1)}), \ldots, \widehat{p}_k(x_{(m)})$. The corresponding estimated risks are $\text{KL}(\widehat{\boldsymbol{p}} \mid \mathfrak{T}_m, \boldsymbol{p}) \approx 0.00472$ and $\text{MSE}(\widehat{\boldsymbol{p}} \mid \mathfrak{T}_m, \boldsymbol{p}) \approx 0.00143$. (Training seed is 0, validation seed is 1, and network seed is 2.)

[4] The specific 0.005, 50, and 500 values are conservative compared to what works well in informal experiments. With "conservative", we mean that the required minimum loss decrease (for continuing training rather than early stopping) is relatively low, and both *patience* and the maximum number of epochs are chosen relatively high.

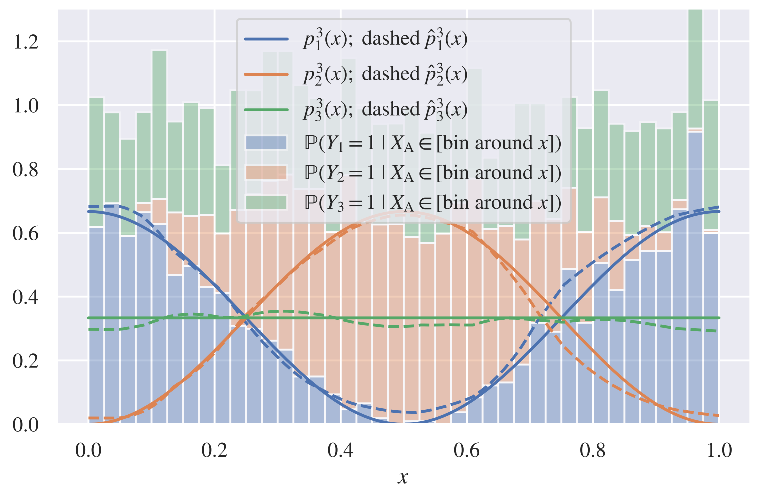


Figure 9: Same as Figure 8, but for scenario A3. The corresponding estimated risks are $\mathrm{KL}(\widehat{\boldsymbol{p}} \mid \mathfrak{T}_m, \boldsymbol{p}) \approx 0.00862$ and $\mathrm{MSE}(\widehat{\boldsymbol{p}} \mid \mathfrak{T}_m, \boldsymbol{p}) \approx 0.000838$.

Figure 8 also displays minor overfitting on an imperfectly representative training set. Around $x \approx 0.55$, the training set happens to contain a lower proportion of samples labeled class 1 than the true $p_1^2(x)$. The DNN therefore learns a too small $\widehat{p}_1(x)$ for $x \approx 0.55$. This can be alleviated with a larger L_1 regularization penalty, but that would bias $\widehat{p}_1(x)$ and $\widehat{p}_2(x)$ towards $1/2$ for all x.

We consider it a failure to train if the DNN learns a constant class probability. This behavior is relatively easy to diagnose: the training loss is comparable to predicting exactly $(1/2, 1/2)$ given any input.[5] Figure 10 displays such extreme underfitting, resulting from a too large L_1 regularization penalty.

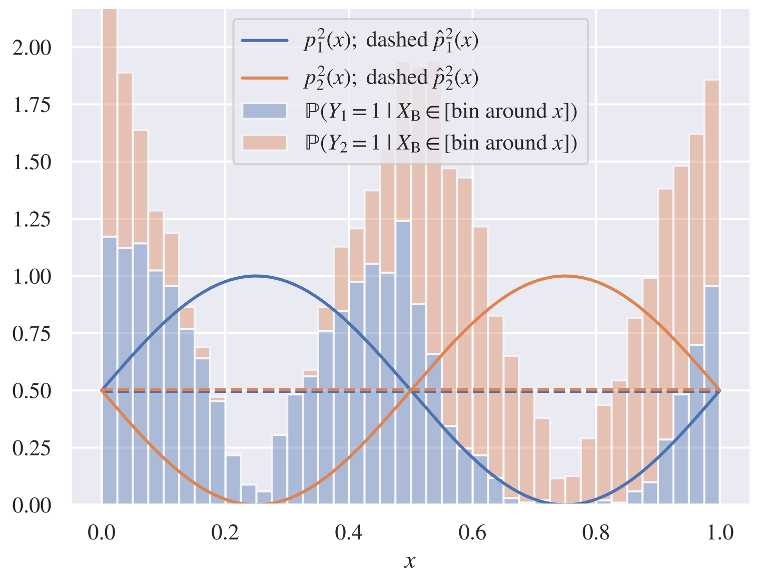


Figure 10: Same as Figure 8 but with extreme underfitting, because of too much L_1 regularization ($\lambda = 0.05$). The corresponding estimated risks are $\mathrm{KL}(\widehat{\boldsymbol{p}} \mid \mathfrak{T}_m, \boldsymbol{p}) \approx 0.171$ and $\mathrm{MSE}(\widehat{\boldsymbol{p}} \mid \mathfrak{T}_m, \boldsymbol{p}) \approx 0.0743$.

Figure 11 displays the negative log-likelihood loss on training and validation set per epoch of training the DNNs of Figures 8 and 9. As expected and desirable, a decrease in training loss tends to coincide with a decrease in validation loss. Figure 11 also demonstrates early stopping: the training processes stop after far fewer than the maximal 500 epochs, namely when validation loss stops improving.

[5] The $(1/2, 1/2)$ prediction given any input only indicates failed training in the case of binary classification with equally likely classes. The more general indication of this failure is predicting (proportion of samples in class 1 in the training set, ..., proportion of samples in class K in the training set) given any input.

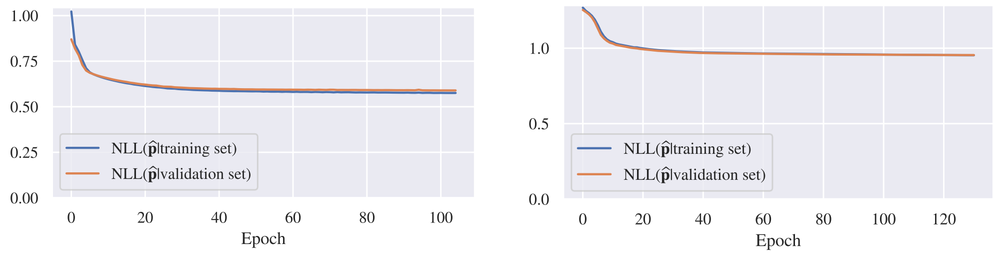


Figure 11: Negative log-likelihood training and validation loss per training epoch for the DNNs of Figure 8 (left) and 9 (right).

4.4 Hyperparameter Search

To find suitable choices for hyperparameters, we use the `hyperopt` package. We aim to find values for the L_1 regularization penalty λ and the Adam optimizer's learning rate η directly, and values for the number of hidden layers, L, and number of neurons per layer, $\boldsymbol{m}$, indirectly. The latter two are indirect because the main risk bound suggests their values should scale with training set size n. We use `hyperopt` to search for an L^* and m^* that relate to $L, \boldsymbol{m}$, and n as follows:

$$L = \lceil L^* \ \log_{1.04}(n) - 0.5 \rceil, \tag{13}$$

$$m_1 = \ldots = m_L = n^{m^*}. \tag{14}$$

The manner of scaling described in (13) and (14) is subjectively chosen as balance between the theorem's conditions and common DNN architectures. The logarithm base 1.04 in (13) is chosen for a smaller amount of scaling with n than the main risk bound's squared natural logarithm. This choice is based on performance in informal experiments. To give an idea of how the hidden layer sizes of a DNN scale with training set size n when $L^* = 0.02$ and $m^* = 0.35$:

- $n = \quad 4096 \implies \{m_1, \ldots, m_{10}\} = \{18, 18, 18, 18, 18, 18, 18, 18, 18, 18\}$;
- $n = \quad 16384 \implies \{m_1, \ldots, m_{12}\} = \{30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30\}$;
- $n = 131072 \implies \{m_1, \ldots, m_{15}\} = \{62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62\}$.

We attempt to find good values for the influential regularization penalty λ and learning rate η directly. The regularization penalty should be set to a balanced value that results in neither over- nor underfitting. The learning rate should also be balanced, for gradient updates that are neither too small nor too large for stable optimization during training. Another consideration for avoiding a large regularization penalty and small learning rate is that these generally increase the significant computational cost of training. (The regularization penalty has a major impact on the duration of training because of our reliance on early stopping.)

We use the hyperparameter search approach described in Bergstra, Yamins, and Cox (2013). The method is based on *Bayesian optimization*. It should be an improvement over manual or random hyperparameter search in terms of both empirical results and reproducibility. The hyperparameter search approach involves a manually specified hyperparameter search space, an objective function, a hyperparameter optimization algorithm, and a history of previously obtained objective scores.

The details of the hyperparameter search approach are out of our scope, but a brief description follows. We use the tree-structured Parzen estimator hyperparameter optimization algorithm, suggested in Bergstra et al. (2011). This algorithm chooses the current iteration's hyperparameters from the manually specified search space, based on the hyperparameter search's history thus far. The algorithm intends to choose informative hyperparameters, balancing exploration and exploitation. A DNN using these hyperparameters is trained on newly generated training and validation sets. We vary the size of the training sets; the size of the validation set is always 10^4. The objective score of an iteration is the estimated KL divergence risk of the trained DNN on a 10^5-sized test set.

We let `hyperopt` perform a hyperparameter search for relatively few iterations, but still expect to find better hyperparameters in this manner than with random guessing. Whether optimal values for (hyper)parameters of deep neural networks exist is itself an interesting research question.

We start with the following initial hyperparameter ranges (or distributions), based on commonly recommended starting values and informal experiments:

- Learning rate $\eta \sim \text{uniform}(0.0001, 0.001)$;
- Regularization penalty $\lambda \sim \text{uniform}(0.0001, 0.001)$;

- DNN depth-related $L^* \sim \text{uniform}(0.02, 0.08)$;
- DNN width-related $m^* \sim \text{uniform}(0.2, 0.5)$.

We run thirty `hyperopt` iterations with training set sizes $n = 8192$ and $n = 65536$ for scenarios B2, A3, and C5. These simulation scenarios are chosen because they seem to be the hardest of the scenarios per combination of input dimensionality and number of output classes. Hyperparameters that work in the hardest scenarios should work in easier scenarios as well.

Figures 17-22 in Appendix B display the estimated KL divergence risks of the thirty `hyperopt` iterations per scenario and training set size. The range of estimated risks differs significantly between the scenarios. The few very high losses result from a combination of underfitting *and* an unrepresentative class imbalance in the training and validation sets.

While the estimated risks depend on the scenario, they are robust to different hyperparameter values. This implies that most values in the initial hyperparameter ranges are sensible choices. Nevertheless, the specific hyperparameters we choose are:

- Learning rate $\eta = 0.0005$.
- Regularization penalty $\lambda = 0.0002$. Given our initial range, we pick a small value, because smaller values lead to better estimated risks in all `hyperopt` settings. With $\lambda = 0.0002$, we sometimes observe minor visual evidence of overfitting, but never the large discrepancy between training and validation/test set loss that is characteristic of overfitting.
- DNN depth-related $L^* = 0.05$ and width-related $m^* = 0.35$. The effect of L^* and m^* is not very clear, so we choose relatively small values (given our ranges). Lower values of L^* and m^* could slightly decrease computational cost. The chosen L^* and m^* result in the DNN sizes exemplified above: e.g., $n = 4096 \implies \{m_1, \ldots, m_{10}\} = \{18, \ldots, 18\}$ and $n = 131072 \implies \{m_1, \ldots, m_{15}\} = \{62, \ldots, 62\}$.

4.5 Experimental Setup

We describe the experimental setup used in our simulation study now. For each of the simulation scenarios introduced in Section 4.1, we first generate a test set of size $m = 10^5$.

The computational bulk of our experimental setup comprises forty *iterations* of:

1. Generating a training set of size n. The sampling seed is the iteration number.
2. Generating a validation set of size 10^4. The seed is 1 + [the iteration number].
3. Training *two* DNNs with the same architecture, hyperparameters, and training and validation sets. These DNNs have as seeds $n + 2\times$[the iteration number]+[0 for the first DNN; 1 for the second DNN]. We train two networks per training set because of the variability of DNN training. We want to evaluate a DNN that trained successfully in the next step, and the best network of two is more likely to have trained successfully. Informal experiments suggest training two DNNs is a good compromise between computational cost and probability of successful training.
4. Evaluating the DNN with the lowest validation loss on the test set as well, without retraining. In this step, we obtain our estimated risks, $(\text{KL}(\widehat{\boldsymbol{p}} \mid \mathfrak{T}_m, \boldsymbol{p})$ and $\text{MSE}(\widehat{\boldsymbol{p}} \mid \mathfrak{T}_m, \boldsymbol{p}))$.

To find differences in (estimated) risk convergence *rate* among the different scenarios, we perform the forty iterations described above with various training set sizes per scenario. The training set sizes we consider are $n \in \{4096,\ 8192,\ 16384,\ 32768,\ 65536,\ 131072\}$.

Instead of training two DNNs per training set, we could also use the training or validation loss to determine if training has been successful enough. In the following box, we propose an alternative experimental setup that does the former. We do not report experimental results obtained with this setup, but do provide its implementation. The setup takes the training imperfection condition of the main risk bound seriously. The idea is to retry the training process with new initialization seeds until a training loss close to the previous best is obtained. This means the amount of trained networks can turn out to be very large. In contrast, the actually used setup gives every scenario, training set size, and iteration approximately the same amount of computation time (and always two trained DNNs).

Alternative experimental set-up

The only difference between the alternative and actually used setup is the third of the steps that are performed for forty iterations per scenario and training set size. The alternative version of step 3 is:

3. Training *a variable number of* DNNs with the same architecture, hyperparameters, and training and validation sets. The number of trained DNNs varies because we wish to evaluate networks that have trained relatively successfully in the next step. To select such DNNs, we retrain with new network seeds until we find a DNN that has training loss no more than a chosen factor times the minimum training loss of any DNN for this scenario and training set size. This may be the first DNN we train in an iteration. However, in the first two iterations, we try a minimum of five networks to create an initial database of training losses to compare with. We judge training losses obtained in earlier iterations again if the minimal training loss decreases in a later iteration. The result is that *all* training losses are eventually at most the chosen factor times the minimal training loss.

The alternative setup involves retraining until training loss is as desired. It is also possible to do this with real-world datasets, for instance when cross-validating. We only assume a training loss of a certain order is possible, which is reasonable since a DNN with identical architecture trained on an already assumed to be identically distributed dataset did obtain such loss.

4.6 Determining Risk Convergence Rates

The purpose of our simulation study is to compare the convergence rates suggested by the main risk bound with the estimated risk convergence rates obtained in our experiments. We determine the estimated risk convergence rates by fitting lines of the form $\theta_1 n^{\theta_2}(\log n)^2$ to the estimated risks. The fit's θ_1 captures constants of the main risk bound, and is of no particular interest. The main risk bound's convergence rate, ϕ_n, is represented by n^{θ_2}.

To fit $\theta_1 n^{\theta_2}(\log n)^2$ to the estimated risks, we use a generalized simulated annealing optimization method by Xiang et al. (1997). We minimize

$$\sum_n \frac{1}{n} \frac{\left(s_n(\widehat{\boldsymbol{p}} \mid \mathcal{T}_m, \boldsymbol{p}) - \theta_1 n^{\theta_2}(\log n)^2\right)^2}{s_n(\widehat{\boldsymbol{p}} \mid \mathcal{T}_m, \boldsymbol{p})},$$

requiring $\theta_2 \in [-1, 0]$. We always set s_n to the first quartile of the forty estimated risks. We choose the first quartile rather than the minimum to ensure robustness. We do not choose the mean, median, or all values because we want to limit the influence of DNNs that failed to train.

4.7 Reproducibility

The Python code, models, and results can all be found on `https://github.com/bramotten/DNN-Classification-Theory-In-Practice`. All Python 3.7.10 packages (and versions) are listed in `requirements.txt`. All required code is in the `main.ipynb` Jupyter notebook.

We use and store seeds underlying every random process, so that experimental results are as reproducible as possible. The seeds that have to do with dataset generation are set as `numpy.random.seeds`.

The training of deep neural networks is not made reproducible straightforwardly. There is a `tensorflow.random.set_seed` function, but it is not sufficient. We use the fixes provided by the `tensorflow-determinism` package. With this package installed, it should be enough to set `os.environ["TF_DETERMINISTIC_OPS"] = "1"`. However, this turns out to *not* be enough. The seed and package do have an impact on the sampling of network parameters and eventual fit, but some randomness persists within the training process.

The entire experimental stage of this simulation study is automated after defining a list of hyperparameter (range)s, scenarios, and training set sizes. The experiments are interruptible because the files with results are periodically updated. The hyperparameter search is also interruptible, through periodic saves of a `hyperopt.Trials` object.

The Python dictionaries with experimental results are stored in `pickle` files in the `test_losses` folder. This folder also contains all Keras `model` files that contain, e.g, DNN parameters and the history of the training and validation loss, grouped in folders named according to the scenario, training set size, training seed, and network seed. The `hyperopt` folder contains the `hyperopt.Trials` objects per scenario and training set size.

5 Simulation Results

Figure 12 displays the forty estimated Küllback-Leibler divergence and mean squared error risks per training set size for all binary classification with one-dimensional input scenarios. The results for scenarios A1 and A2 are identical, so only the results for scenarios A2, B1, and B2 are shown.[6] Figure 13 displays the estimated risks for the multiclass classification scenarios A3 and A4 and the multidimensional input scenarios C5 and C6.

Because of the inherent variability of deep neural network training, the estimated risks are not tightly and symmetrically grouped around their mean. The distributions of the estimated risks are positively skewed. Some evaluated DNNs have failed to train successfully, but no estimated risks are as bad as the worst we have seen in the hyperparameter search. Furthermore, many networks obtain approximately the lowest estimated risk per setting, especially at larger training set sizes. Especially considering this last observation, the DNNs perform more consistently than expected.

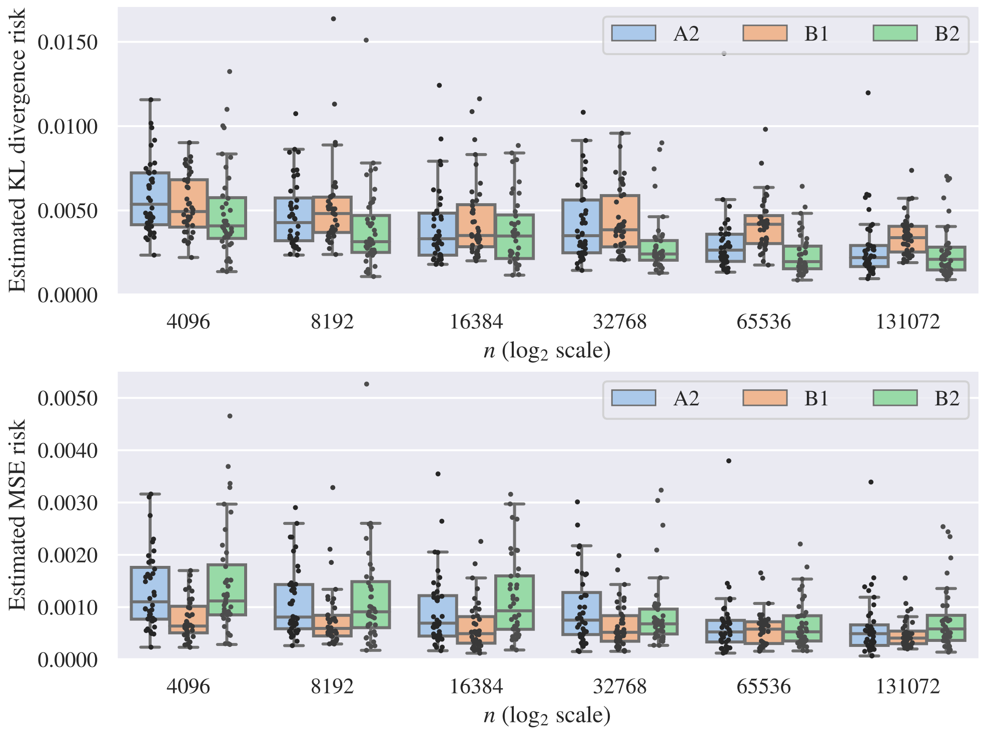


Figure 12: Box- and scatter plots of forty estimated risks per training set size n for the binary classification with one-dimensional input scenarios. The estimated risks are obtained using the experimental setup described in Section 4.5. The definitions of the estimated risks, $\mathrm{KL}(\widehat{\boldsymbol{p}} \mid \mathfrak{T}_m, \boldsymbol{p})$ and $\mathrm{MSE}(\widehat{\boldsymbol{p}} \mid \mathfrak{T}_m, \boldsymbol{p})$, are given in (7) and (8) respectively.

[6] The conditional class probability functions $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ are shifted compared to each other, but that is inconsequential in combination with the uniform input distribution X_{A}.

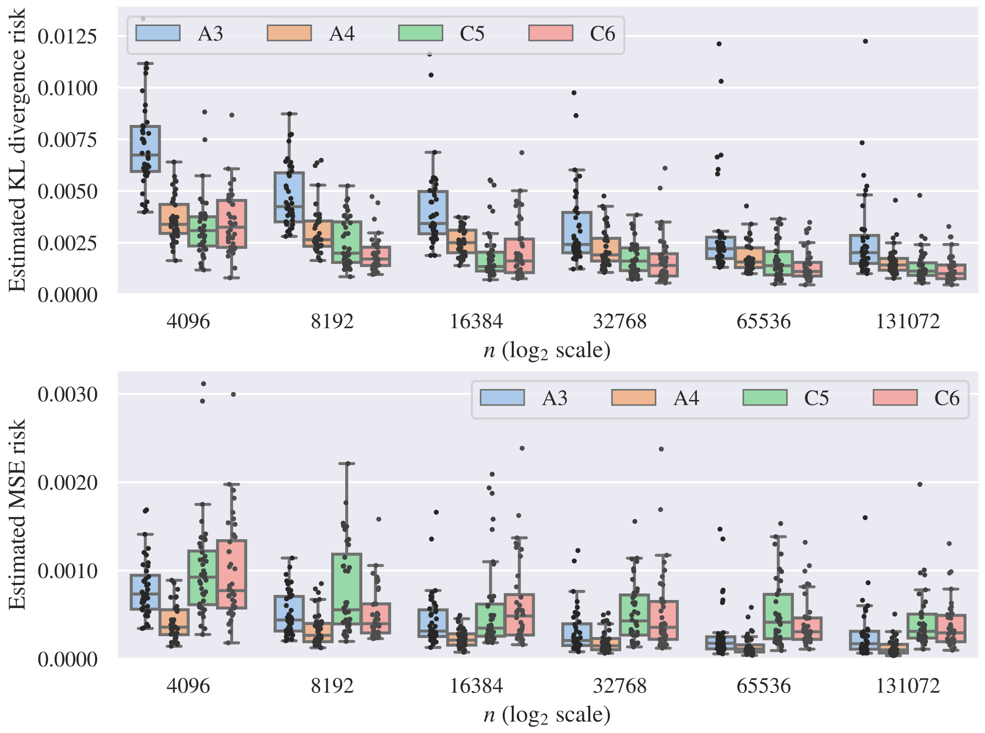


Figure 13: Same as Figure 12, but for the multiclass and multidimensional input scenarios.

Figures 14 and 15 display the first quartiles of the estimated KL divergence and MSE risks, with corresponding fits of the form $\theta_1 n^{\theta_2}(\log n)^2$. The main risk bound suggests a KL divergence risk convergence rate $\phi_n(\log n)^2$, where $\phi_n \asymp n^{-1}$ in all scenarios. Thus, it suggests $\theta_2 = -1$ in all scenarios. The observed convergence rates are considerably slower.

As suggested by the main risk bound, the α-SVB index has no consistent effect on the convergence rate in our scenarios. The convergence rate is relatively slow in the B1 scenario and relatively fast in the A3 scenario. The 1/2-SVB condition holds in both the slowly-converging scenario B1 and the quickly-converging scenario A3. In the other scenarios, which have similar convergence rates to each other, the α-SVB condition with either $\alpha = 1/2$ or $\alpha = 1$ holds.

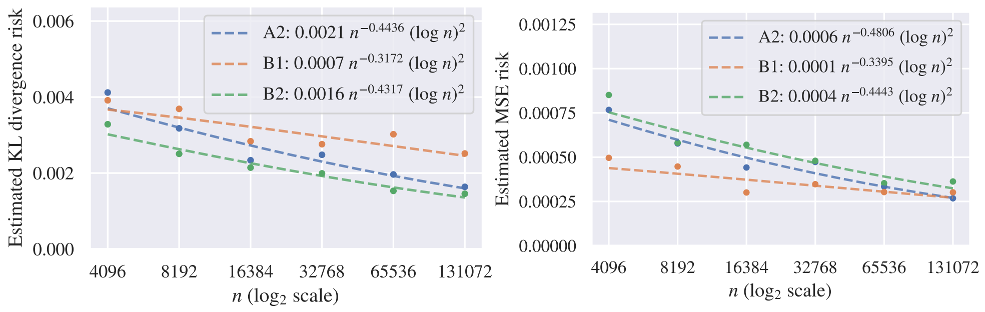


Figure 14: Scatter plots of the first quartiles of the experimentally obtained estimated risks, and fits of the form $\theta_1 n^{\theta_2}(\log n)^2$ as described in Section 4.6. The binary classification with one-dimensional input scenarios and corresponding θ_1 and θ_2 are denoted in the legend.

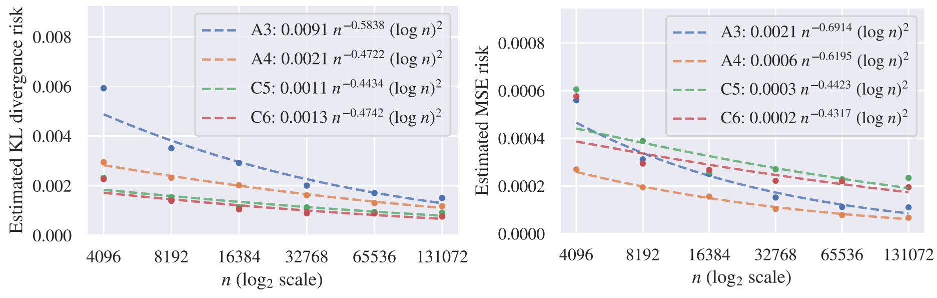


Figure 15: Same as Figure 14, but for the multiclass and multidimensional input scenarios.

6 Discussion

We used simulations to examine the main risk bound of Bos and Schmidt-Hieber (2021). The main risk bound suggests a truncated Küllback-Leibler divergence risk convergence rate of n^{-1} in all considered simulation scenarios. We found considerably slower rates.

We experimented with both binary and multiclass classification, as well as with both one- and multidimensional input settings. We reported manual calculations of the different α-small value bound index conditions per setting, and proposed a lemma and techniques that simplify such calculations. Because of the arbitrarily smooth conditional class probability functions in the proposed settings, the main risk bound suggests identical KL divergence risk convergence rates for the settings in which different α-SVB conditions hold.

We indeed found that the SVB index α did not have an effect on the estimated convergence rates in such settings. This finding supports one of the main risk bound's implications – that DNNs can obtain relatively fast KL divergence risk convergence rates even if a large proportion of true conditional class probabilities is small, as long as the conditional class probability functions are sufficiently smooth.

There is an inherent difficulty involved in examining theoretical results for DNNs using a simulation study. The results of DNN training are highly variable, so that performing many experiments is desirable. But DNN training – and thus performing extra experiments – is computationally expensive.

The source code for this simulation study is freely available. The implementation can be used to generate datasets after specification of any input density and conditional class probability function. The implementation can be used to train individual DNNs, find good hyperparameters, or perform our whole experimental setup with any combination of dataset distributions. The progress of computationally expensive operations is saved periodically, so that interruption and resumption are possible. In the case of low-dimensional input distributions, visualizations such as Figure 8 can display the datasets and the DNN's approximation of the conditional class probability functions for the purpose of monitoring experiments.

7 Open Questions

This work is exploratory and leaves many questions unanswered. Our first suggestions for future work are:

1. To study additional scenarios that satisfy only a *small* Hölder smoothness index β. In this case, the main risk bound suggests an observable effect of the small value bound index α on the truncated Küllback-Leibler divergence risk convergence rate. Our simulation study only involves scenarios in which the effect of the α-SVB index on the convergence rate is diminished because of the arbitrarily large Hölder smoothness index β.

2. To study additional scenarios that only satisfy an SVB index $\alpha < 1/2$ while following suggestion 1. Such scenarios lead to a more pronounced effect of α on the convergence rate than the lowest $\alpha = 1/2$ occuring in our scenarios.

3. To study additional scenarios that satisfy an SVB index $\alpha > 1$. Such scenarios target an interesting aspect of the main risk bound, namely that the convergence rate improves with larger α-SVB index, but only up to $\alpha = 1$.

4. To study additional scenarios that have higher-dimensional input ($d > 2$) and more than the three output classes ($K > 3$) that we considered. Such scenarios are interesting because high-dimensional input settings especially are common in DNN applications.

The next suggestion for future work arises from not just this work, but the whole practice of approaching classification as discrete probability distribution approximation. It would be useful to have publicly available datasets involving *empirical* conditional class probabilities. Example 1 describes a medical setting in which these empirical probabilities can have a sensible meaning. The empirical probability of label k given an image could represent the proportion of questioned experts who thinks the image should be labeled k. Datasets with empirical conditional class probabilities could be created in this medical setting, but not in many other supervised classification settings.

Datasets with empirical conditional class probabilities can reduce the gap between theory and practice in multiple ways. First, it will become clear whether the conditions of many theoretical classification results are often satisfied in practice. These results have conditions that can only be checked using currently unknown conditional class probability functions. Second, theoretical results that depend on conditional class probabilities can be examined using these special datasets, rather than only using simulations. The main risk bound of Bos and Schmidt-Hieber (2021) is an example of a theoretical result that could benefit from datasets with empirical conditional class probabilities. It requires an SVB index α and Hölder smoothness index β – these are now unknown for real-world datasets, but could be approximated using empirical conditional class probabilities. Appendix A describes a methodology for computing the α-SVB index using a conditional class probability function.

Future work could also investigate how well the SVB and Hölder smoothness indices can be approximated *without* knowing the true – or at least empirical – conditional class probabilities.

References

Bergstra, J., D. Yamins, and D. D. Cox (2013). “Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures.” *Proceedings of the 30th International Conference on Machine Learning - Volume 28*. URL: http://proceedings.mlr.press/v28/bergstra13.pdf.

Bergstra, J., R. Bardenet, Y. Bengio, and B. Kégl (2011). “Algorithms for Hyper-Parameter Optimization.” 25th Annual Conference on Neural Information Processing Systems (NIPS 2011). URL: https://hal.inria.fr/hal-00642998.

Bos, T. and J. Schmidt-Hieber (2021). “Convergence rates of deep ReLU networks for multiclass classification.” arXiv: 2108.00969.

Glorot, X. and Y. Bengio (2010). “Understanding the difficulty of training deep feedforward neural networks.” *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. URL: http://proceedings.mlr.press/v9/glorot10a.html.

Glorot, X., A. Bordes, and Y. Bengio (2011). “Deep Sparse Rectifier Neural Networks.” *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. URL: http://proceedings.mlr.press/v15/glorot11a.html.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep learning*. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press. 775 pages. ISBN: 978-0-262-03561-3. URL: https://www.deeplearningbook.org/.

He, K., X. Zhang, S. Ren, and J. Sun (2015a). “Deep Residual Learning for Image Recognition.” arXiv: 1512.03385.

He, K., X. Zhang, S. Ren, and J. Sun (2015b). “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.” *2015 IEEE International Conference on Computer Vision (ICCV)*. DOI: 10.1109/ICCV.2015.123.

Hornik, K., M. Stinchcombe, and H. White (1989). “Multilayer feedforward networks are universal approximators.” *Neural Networks* 2.5, pages 359–366. ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8.

Ioffe, S. and C. Szegedy (2015). “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” arXiv: 1502.03167.

Kim, Y., I. Ohn, and D. Kim (2019). “Fast convergence rates of deep neural networks for classification.” arXiv: 1812.03599.

Kingma, D. P. and J. Ba (2017). “Adam: A Method for Stochastic Optimization.” arXiv: 1412.6980.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). “ImageNet Classification with Deep Convolutional Neural Networks.” *Advances in Neural Information Processing Systems* 25, pages 1097–1105. URL: https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

Nair, V. and G. E. Hinton (2010). “Rectified Linear Units Improve Restricted Boltzmann Machines.” URL: https://www.cs.toronto.edu/~hinton/absps/reluICML.pdf.

Ragab, D. A., M. Sharkas, S. Marshall, and J. Ren (2019). “Breast cancer detection using deep convolutional neural networks and support vector machines.” *PeerJ* 7. Publisher: PeerJ Inc., e6201. ISSN: 2167-8359. DOI: 10.7717/peerj.6201.

Ramachandran, P., B. Zoph, and Q. V. Le (2017). “Searching for Activation Functions.” arXiv: 1710.05941.

Santurkar, S., D. Tsipras, A. Ilyas, and A. Madry (2019). “How Does Batch Normalization Help Optimization?” arXiv: 1805.11604.

Schmidt-Hieber, J. (2020). “Nonparametric regression using deep neural networks with ReLU activation function.” *Annals of Statistics* 48.4, pages 1875–1897. ISSN: 0090-5364. DOI: 10.1214/19-AOS1875. arXiv: 1708.06633.

Xiang, Y., D. Y. Sun, W. Fan, and X. G. Gong (1997). “Generalized simulated annealing algorithm and its application to the Thomson model.” *Physics Letters A* 233.3, pages 216–220. ISSN: 0375-9601. DOI: 10.1016/S0375-9601(97)00474-X.

Appendix A Empirical SVB Index Calculation

We measure the proportion of small conditional class probabilities using the small value bound, indexed by α. To reduce the need for manual calculation, we propose an empirical estimate of α using $\boldsymbol{p}$ and n samples of $\boldsymbol{X}$. The method works as follows:

1. Generate a grid of small positive values, for instance $\boldsymbol{\tau} = (0.0001, 0.0002, \ldots, 0.05)$.
2. Calculate $f_k(n, \tau) = (1/n) \sum_{j=1}^{n} \mathbb{1}_{\{z\colon z \le \tau\}}(p_k(\boldsymbol{X}_j))$ for each $k \le K$ and $\tau \in \boldsymbol{\tau}$.
3. Fit $\theta_1 \boldsymbol{\tau}^{\theta_2}$ for each k to minimize $(1/n) \sum_{j=1}^{n} \sum_{\tau \in \boldsymbol{\tau}} \left(f_k(n, \tau) - \theta_1 n^{\theta_2}\right)^2$, for instance using generalized simulated annealing as in Section 4.6.
4. The class k whose corresponding fit θ_2 is smallest is the class that determines the α-SVB index.

Figure 16 displays the results of this methodology for some of the simulation scenarios we consider. The results are in agreement with the manually calculated α.

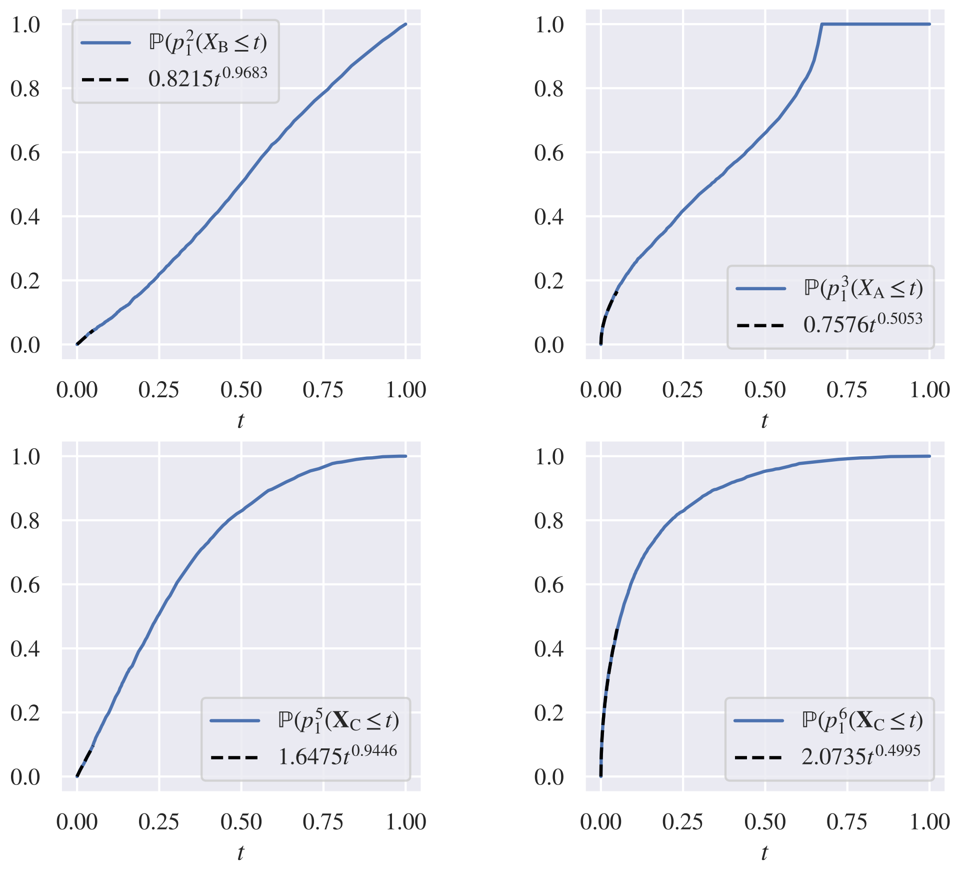


Figure 16: The empirical α-SVB based on a dataset of 4096 samples. The scenarios are indirectly indicated in the legend (clockwise from top-left: B2, A3, C5, C6). The displayed p_k is the one with the smallest empirical α in the scenario. The dashed line is a fit of $\theta_1 \boldsymbol{\tau}^{\theta_2}$ form up to $t = 0.05$.

However, regular supervised classification datasets consist of $(\boldsymbol{X}_1, \boldsymbol{Y}_1), \ldots, (\boldsymbol{X}_n, \boldsymbol{Y}_n)$ – knowledge of $\boldsymbol{p}$ is missing. Without $\boldsymbol{p}$, step 2 is impossible. Replacing step 2 with $(1/n) \sum_{j=1}^{n} \mathbb{1}_{\{y\colon y \le \tau\}}(Y_k^j)$ does not work since it gives the proportion of Y_k that is zero with any $\tau > 0$. That proportion is not informative because it is equal to the proportion of samples that does *not* belong to class k.

Appendix B Hyperparameter Search

Figures 17-22 are explained in Section 4.4.

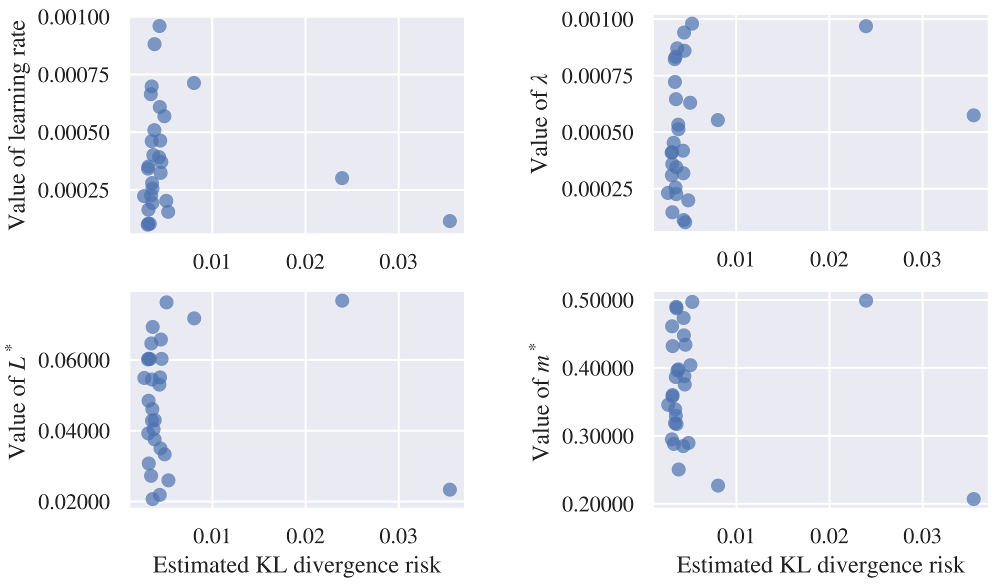


Figure 17: Thirty `hyperopt` iterations in scenario B2 with $n = 8192$.

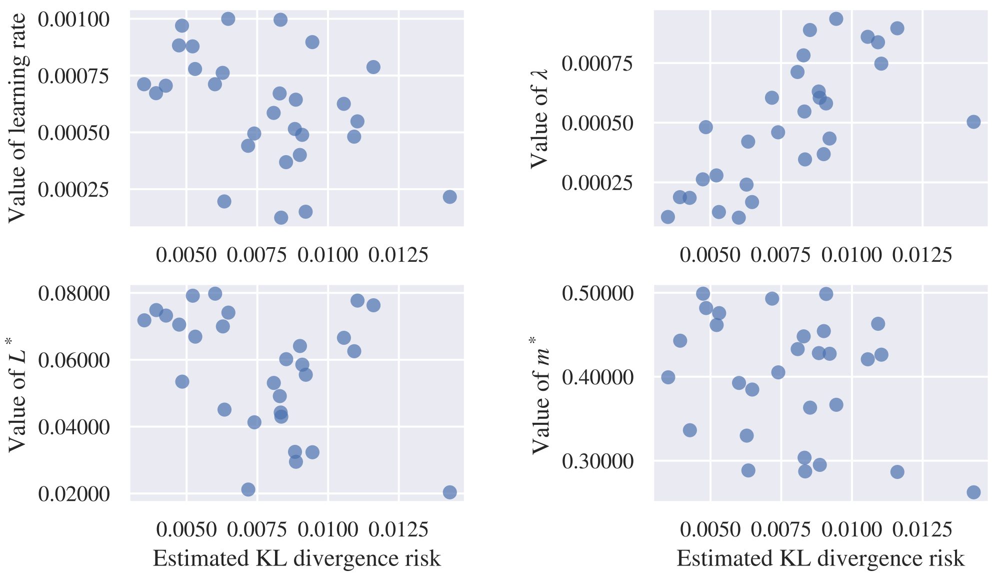


Figure 18: Thirty `hyperopt` iterations in scenario A3 with $n = 8192$.

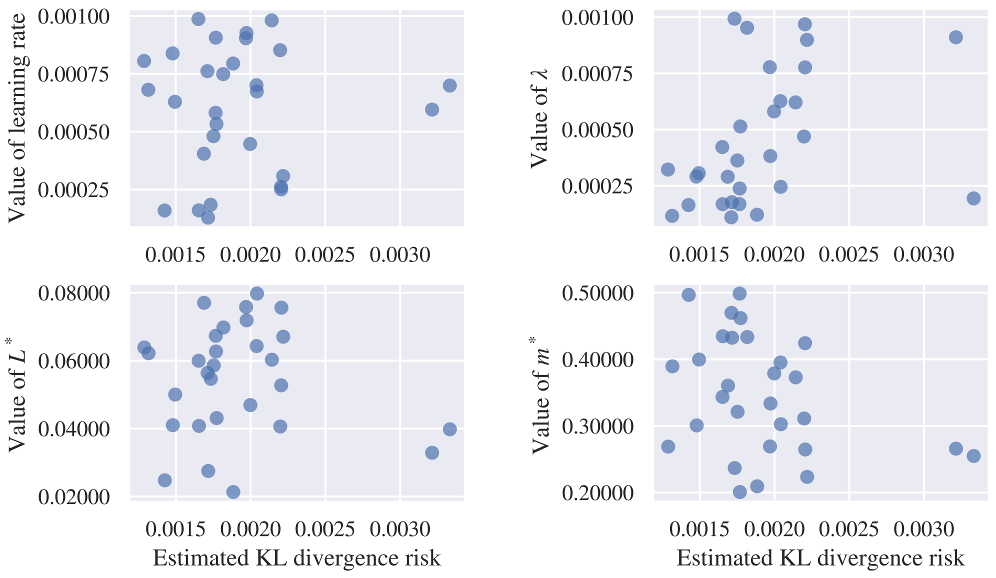


Figure 19: Thirty `hyperopt` iterations in scenario C5 with $n = 8192$.

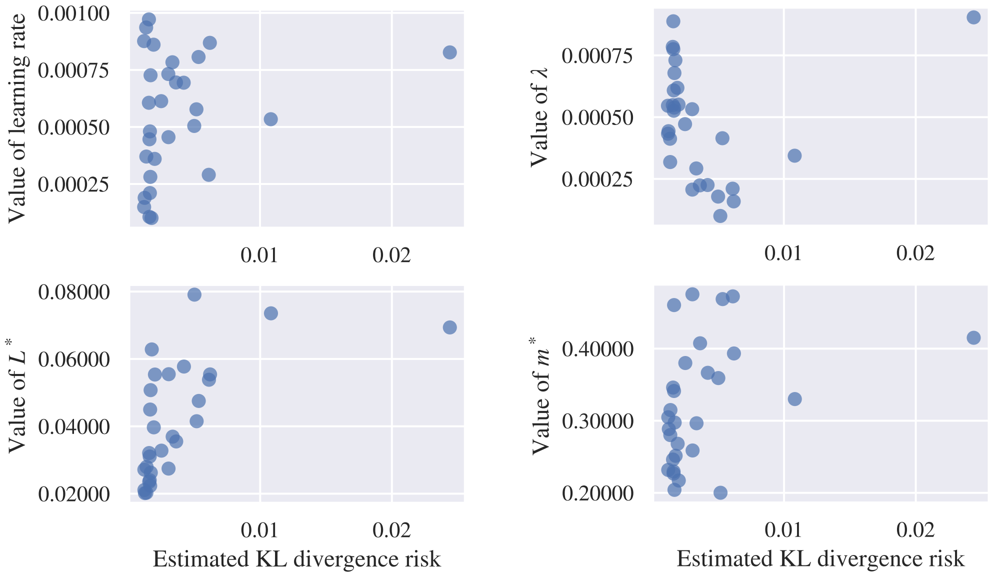


Figure 20: Thirty `hyperopt` iterations in scenario B2 with $n = 65536$.

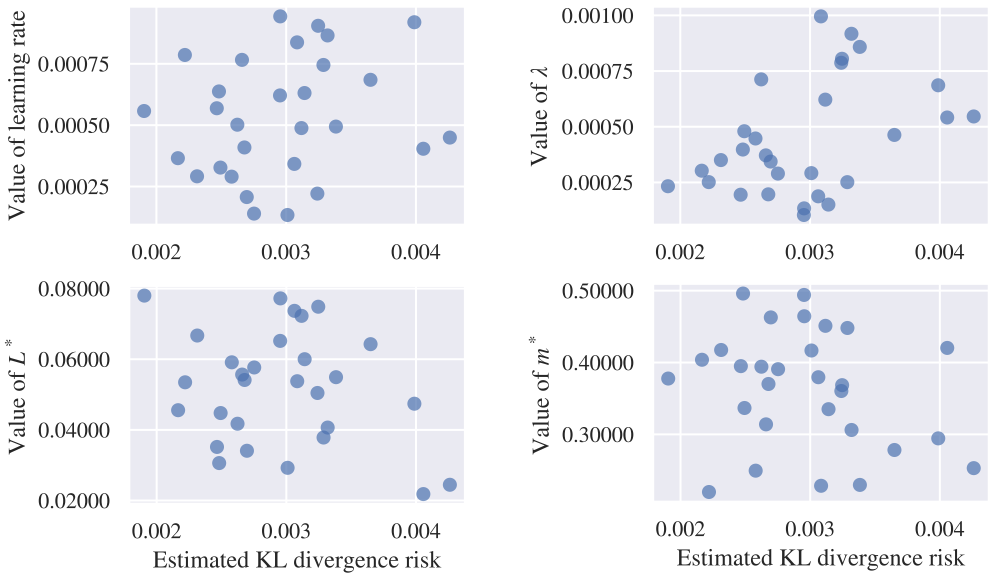


Figure 21: Thirty `hyperopt` iterations in scenario A3 with $n = 65536$.

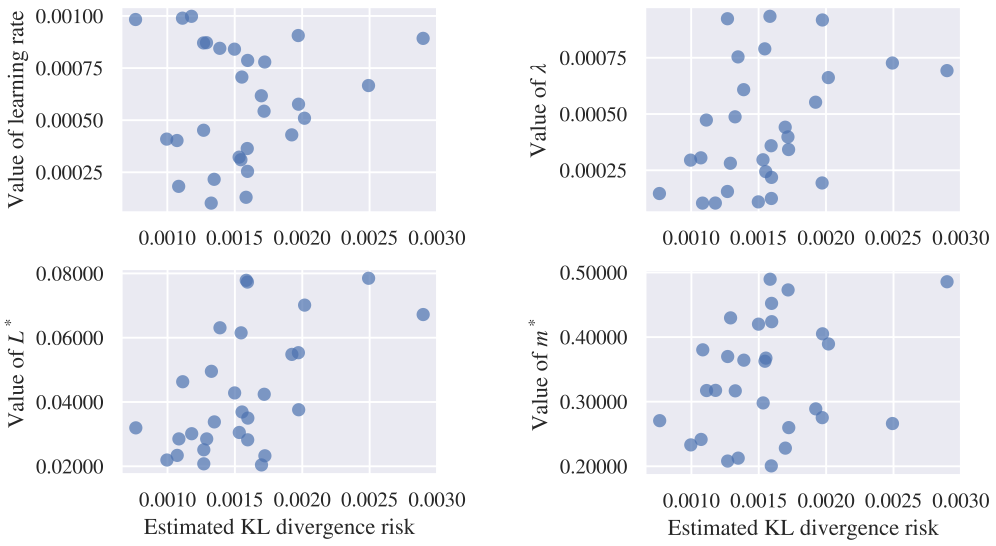


Figure 22: Thirty `hyperopt` iterations in scenario C5 with $n = 65536$.