UMBC

**DISSERTATION APPROVAL SHEET**

Title of Dissertation: Uncertainty for Malware Detection and Cyber Defense

Name of Candidate: Andre Nguyen

Doctor of Philosophy, 2021

Graduate Program: Computer Science

Dissertation and Abstract Approved:

Charles Nicholas

Professor

Computer Science

11/23/2021 | 4:09:03 PM EST

Edward Raff

Edward Raff

Professor

Computer Science

11/23/2021 | 11:14:00 PM EST

NOTE: *The Approval Sheet with the original signature must accompany the thesis or dissertation. No terminal punctuation is to be used.

# ABSTRACT

Title of dissertation: UNCERTAINTY FOR MALWARE DETECTION AND CYBER DEFENSE

André Tai Nguyen, Doctor of Philosophy, 2021

Dissertation directed by: Professor Charles Nicholas
Professor Edward Raff
Department of Computer Science
and Electrical Engineering

As organizations in government and industry increasingly rely on digitized data and networked computer systems, they face a growing risk of exposure to cyber attacks. As computer networks grow in size, so do the challenges cybersecurity professionals face in securing them. With more connected devices, more users, and more complex systems, adversarial attack opportunities increase exponentially.

Recently, the collection and release of malware datasets has allowed for the development of machine learning (ML) approaches to detect malware. Existing ML based approaches to malware detection have not yet leveraged uncertainty in a systematic manner. Cybersecurity intrinsically requires operating under uncertain conditions, so ignoring uncertainty is undesirable.

In this thesis, we explore different ways uncertainty estimation can benefit cyber defense. In particular, we demonstrate how taking into account uncertainty can be especially beneficial for highly constrained and quickly evolving malware detection use cases, laying the groundwork for the increased adoption of uncertainty

aware ML in the cybersecurity community. Leveraging uncertainty, we improve malware detection rates under extreme false positive rate constraints, improve out of distribution data detection approaches, and significantly reduce the amount of compute time needed to take advantage of the benefits of dynamic analysis. Along the way, we also illustrate why previous evaluation metrics can be misleading and demonstrate that executable file capabilities can be accurately predicted from raw byte sequences.

# UNCERTAINTY FOR MALWARE DETECTION AND CYBER DEFENSE

by

André Tai Nguyen

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2021

Advisory Committee:
Professor Charles Nicholas, Chair/Advisor
Professor Edward Raff, Co-Advisor
Professor Mauricio Santillana
Professor Frank Ferraro
Professor Tim Oates

# Acknowledgments

During the last three years, my work has benefited considerably from discussions with my friends and colleagues at the Laboratory for Physical Sciences, at UMBC, and at Booz Allen Hamilton: Fred Lu, Luke Richards, Richard Zak, Jim Holt, Lucia Jesus-Santana, Drew Farris, Derek Everett, Rob Brandon, Gary Lopez Munoz, Mehdi Rezaee, Kasra Darvish, Youssouf Kebe, and many others. In particular, I would like to Fred Lu for his contributions to the research that led to chapter 4.

I especially thank my advisors, Edward Raff and Charles Nicholas, as well as Frank Ferraro, Cynthia Matuszek, and Tim Oates, for their support, criticism, and advice.

I would like to thank the Laboratory for Physical Sciences, in particular Mark McLean, and Booz Allen Hamilton for supporting my research.

I am grateful to Mauricio Santillana for introducing me to research and to machine learning when I was a freshman in college, Finale Doshi-Velez for introducing me to Bayesian machine learning, and Eli Weinstein for his tips on how to read efficiently.

Finally, I thank all my family and friends for their support, and Christ for all His gifts.

# Table of Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| AV | Anti-Virus |
| BDL | Bayesian Deep Learning |
| FPR | False Positive Rate |
| LGBM | Light Gradient Boosting Machine |
| MCMC | Markov Chain Monte Carlo |
| MI | Mutual Information |
| ML | Machine Learning |
| NN | Neural Network |
| OOD | Out Of Distribution |
| PE | Portable-Executable |
| TPR | True Positive Rate |
| VI | Variational Inference |
| VM | Virtual Machine |

# Chapter 1

## Introduction and Related Work

Organizations in government and industry increasingly rely on information technology, digitized data, and networked computer assets. Consequently, they face a growing risk of exposure to cyber attacks (i.e., malicious attempts at stealing, altering, or destroying information technology systems and data). Yet, as computer networks grow in size, so do the challenges cybersecurity professionals face in securing them. With more connected devices, more users, and more complex systems, adversarial attack opportunities increase exponentially. In 2018, on average, large organizations took almost 200 days to identify a cyber breach and almost 70 days to contain those breaches once they were identified [144].

## 1.1 The Malware Threat

Malicious software is one of the most common methods adversaries use to exploit computer networks. Malware files are created with malicious intent to cause an effect not desired by a computer system owner. Generally, malware-based attacks use manipulated software to intentionally cause damage or access data. A single successful malware attack can result in millions of dollars in damages, with recent annual financial impact measured to be in the hundreds of billions of dollars [9, 8, 87]. Each day, adversaries design new and increasingly complex malware systems,

challenging security professionals to deploy robust and effective counter measures.

As a result, malware detectors have become a critical component of a cyber-security strategy. Traditionally, anti-virus systems used static and signature-driven systems (i.e., systems that looked for specific software known to be malicious) to detect malware [193]. More recently, dynamic software analysis has become increasingly popular, where evaluated software is run in a secure environment to directly observe whether or not it behaves maliciously. Egele et al. [53] provide a survey on challenges, features, techniques, and tools for dynamic malware analysis.

While powerful, anti-virus and dynamic analysis tools have limitations. In particular, these methods can be time consuming when the data volume and velocity are high [157], with certain methods such as dynamic analysis having particularly high computational cost.

## 1.2 Machine Learning Methods for Malware

The collection and release of malware datasets has allowed for the development of machine learning approaches to detect malware. The machine learning based automation of static and dynamic analysis enables a faster investigation of more files and allows human analysts to focus on "hard" samples.

Well-known examples of malware datasets are the EMBER datasets from Anderson and Roth [6] and the Drebin dataset from Arp et al. [12] and Spreitzenbarth et al. [173]. The EMBER2017 dataset is an open source dataset of over a million portable executable files (PE files) scanned by VirusTotal in or before 2017 [6]. The

dataset includes metadata, features derived from the PE files, and a benchmark model trained on those derived features. While the raw PE files are not available as part of EMBER, they can be downloaded via VirusTotal. A second EMBER2018 dataset was later released with a million PE files scanned in or before 2018. The EMBER2018 dataset was designed to be more challenging for machine learning algorithms to classify than the EMBER2017 dataset. The older Drebin dataset of Android malware contains 5560 applications from 179 different malware families collected between August 2010 and October 2012 [12, 173]. Newer is the 2020 "industry scale" Sophos AI SOREL-20M dataset, consisting of 20 million files [74].

Machine learning can be applied to the malware domain in many ways. Binary classification algorithms can be used to produce a decision about whether or not a file is malicious. Multi-class classification algorithms can be used to sort malware by family or type. Unsupervised learning can be used to find similar groups of files and summarize malware. On the offensive side, reinforcement learning can be used to design new malware that evades anti-virus systems [7]. Raff and Nicholas [147] and similarly Gibert, Mateu, and Planes [69] provide modern surveys on machine learning applied to malware classification.

The application of machine learning to malware poses many challenges not often seen in other applications. For example, the rapid evolution of malware techniques leads to fast concept drift. This is a problem even within malware families. For example, the Zeus banking Trojan's source code was leaked in 2011, leading to a plethora of mutations and variants [128]. Today, Zeus is the most prevalent banking Trojan in the wild and is a weapon of choice for cyber criminals targeting banks

and small businesses.

The majority of the existing research in machine learning applied to malware detection has focused on the automation of static malware analysis, where a file is analyzed without being run. Raff et al. [151] introduce MalConv, a convolutional neural network for malware detection that operates on the raw byte sequences of files. The authors note that malware classification from byte sequences can be considered as one of the longest sequence time series classification problems in practice, and as a result of the uniqueness of the problem, much conventional deep learning wisdom was found to not apply in the context of malware classification on raw bytes. Raff et al. [149] develop a new approach to temporal max pooling that improves MalConv's memory and computational costs. They also introduce a new global channel gating design that results in an attention mechanism capable of learning feature interactions across millions of time steps. Yan, Qi, and Rao [199] and Krčál et al. [101] are also examples of deep learning based static analysis.

While promising, these deep learning based static analysis approaches have been shown to be susceptible to adversarial attacks. Adversarial attacks on machine learning systems have goals that are somewhat similar to malware. These adversarial attacks seek to design data that result in a machine learning system behaving in a way that the system's creator did not expect or intend [71, 22]. Demetrio et al. [44] suggest through the use of model interpretation techniques that MalConv does not seem to learn to use meaningful features but rather information from the header which can be easily manipulated. They use this to develop an adversarial attack against MalConv that uses file header manipulation. Kolosnjaji et al. [100]

and similarly Kreuk et al. [102] demonstrate adversarial attacks against MalConv that modifies bytes at the end of malware samples. Papernot [139] provide a good discussion of adversarial attacks on malware classifiers and provide experimental results on the Drebin dataset.

The development of adversarial attacks against machine learning systems used for malware detection has led to an iteration of improving defences and attacks, similar to the cycle seen in cyber defense and offense. Fleshman et al. [56] introduce a defense against adversarial attacks that uses non-negative weight constraints in the context of binary classification tasks with asymmetric costs. In the malware domain, the attackers objective is usually to make malware seem benign, not to make benignware seem malicious. Their defense is shown to remedy attacks such as that from Kreuk et al. [102] and Kolosnjaji et al. [100].

The use of machine learning for malware detection and classification has also led to research in new evaluation methods for these algorithms. Fleshman et al. [57] introduce a new method to test and compare machine learning based static malware classifiers and signature based anti-virus tools, by measuring changes in performance in the face of adversarial modifications. Their results show that pure machine learning based systems can be more robust than traditional anti-virus products at detecting evasive malware, though may be slower to adapt to significantly novel attacks.

We note that machine learning is also useful for cyber defense outside of just malware related tasks. Lin et al. [108] provide a survey of deep learning for software vulnerability detection for example.

## 1.3 What's Missing?

The modeling of uncertainty for decision making is notably missing from the current machine learning for malware literature. Understanding when a machine learning model is uncertain about its prediction is critical in high risk applications such as cyber defense, where a successful attack can have broad implications on national security and even deadly consequences.[1] When an automated malware detection algorithm is uncertain about a sample, the uncertainty estimate could be used to flag the sample for analysis by a more computationally expensive algorithm or for review by a human. Furthermore, the uncertainty could be used to adjust prediction thresholds to achieve a desired outcome such as low false positives and could be used to guide human analysis towards an "interesting" property of the sample, narrowing the search space for the human analyst.

As we will explore in the next section, the Bayesian framework allows for the principled modeling of uncertainty in machine learning.

While there exist some methods in the malware analysis research literature that are labeled as "Bayesian," the majority of these do not fit the definition of Bayesian with respect to the modeling of uncertainty. Instead, these methods simply apply Bayes' Theorem to point estimates as is done in the majority of implementations of Naive Bayes and of Bayesian Belief Networks [203, 109, 206]. The use of Bayes' Theorem is not sufficient to make a machine learning algorithm statistically Bayesian. If a posterior distribution is not computed and a maximum likelihood es-

[1] https://www.nytimes.com/2020/09/18/world/europe/cyber-attack-germany-ransomeware-death.html

timate or a maximum a posteriori estimate is computed instead, then the approach is not Bayesian [132].

A notable exception is the Bayesian work of Backes and Nauman [16] which tackles uncertainty in machine learning for Android malware using probabilistic programming and the Drebin dataset. The work clearly motivates the use and need for Bayesian machine learning in malware analysis. The use of Bayesian models is shown to improve accuracy and was also used to detect real malware samples that were incorrectly labeled as false positives. The authors suggest that there has been no Bayesian machine learning applied to malware previous to their paper. A Bayesian logistic regression model is used, and coverage metrics are introduced for Uncertain Positives and Uncertain Negatives. The paper also introduces two methods for augmenting predictions using uncertainty.

Tangential but still of interest, Sartea, Farinelli, and Murari [162] model *active* dynamic malware analysis as a Bayesian Game (in Game Theory, a game where players have incomplete information about each other) between an analyst agent and a malware agent. In particular, *families* of malware are mapped to *types* of Bayesian games. The analyst takes actions to perform dynamic analysis under uncertainty over the game type/malware family. This approach integrates uncertainty directly into the task of dynamic analysis and minimizes the number of actions (so by extension time and computation) needed to perform accurate malware classification.

Atapour-Abarghouei, Bonner, and McGough [14] introduce a one shot learning approach for classifying ransomware post-infection using screenshots of the ransomware splash screen. Model uncertainty is estimated using dropout-based

Bayesian inference approximations. This uncertainty allows for the detection out of distribution data at test time, mapping to new ransomware or unrelated images. We note that this approach is image-based rather than file-based and cannot be used to prevent an attack.

## 1.4 Goals, Chapters, and Publications

The growing reliance on networked computer systems and digitized data has increased the exposure of organizations in government and industry to cyber attacks. However, the recent collection and release of malware datasets has allowed for the development of machine learning based malware detection algorithms. While cybersecurity intrinsically requires operating under uncertain conditions, current ML based approaches to malware detection have not yet leveraged uncertainty in a systematic manner. In this thesis, we explore different ways uncertainty estimation can benefit cyber defense. In particular, we demonstrate how taking into account uncertainty can be especially beneficial for highly constrained and quickly evolving malware detection use cases, laying the groundwork for the increased adoption of uncertainty aware ML in the cybersecurity community.

This thesis represents the amalgamation of three publications:

- chapter 3: Andre T. Nguyen, Edward Raff, Charles Nicholas, James Holt. "Leveraging Uncertainty for Improved Static Malware Detection Under Extreme False Positive Constraints." Proceedings of the 1st International Workshop on Adaptive Cyber Defense at IJCAI 2021. [134]

- chapter 4: Andre T. Nguyen, Fred Lu, Gary Lopez Munoz, Edward Raff, Charles Nicholas, James Holt. "Out of Distribution Data Detection Using Dropout Bayesian Neural Networks." Under review.

- chapter 5: Andre T. Nguyen, Richard Zak, Fred Lu, Robert Brandon, Gary Lopez Munoz, Edward Raff, Charles Nicholas, James Holt. "Minimizing compute costs: When should we run more expensive malware analysis?" In preparation.

In addition to these chapters, chapter 2 provides an introduction to the quantification of uncertainty in machine learning, and chapter 6 summarizes conclusions and potential future research directions.

# Chapter 2

# The Quantification of Uncertainty

The Bayesian framework allows for the principled modeling of uncertainty in machine learning and decision making [20]. Within this framework, probabilities represent degrees of belief as opposed to the frequentist interpretation of probabilities as long run frequencies [110]. Bayesian inference uses Bayes' Theorem to update beliefs (that are represented in the form of probability distributions) when new data is observed. Reverend Thomas Bayes introduced Bayes' Theorem in the special case where the prior is uniform [18]. Pierre-Simon Laplace later introduced Bayes' Theorem in its general form [106]. Bayesian inference in its modern form was developed by Sir Harold Jeffreys [90], though not without controversy [84, 1]. Martin, Frazier, and Robert [123] provide a thorough review of Bayesian computation in parametric settings through a historical lens, and Schoot et al. [164] provide a primer on Bayesian statistics.

## 2.1 Bayesian Machine Learning

In the context of machine learning, a Bayesian update takes the following form where $\theta$ represents model parameters, $D$ represents the data, and $M$ represents the model class:

$$P(\theta|D, M) = \frac{P(D|\theta, M)P(\theta|M)}{P(D|M)}$$

$P(\theta|D, M)$ is the posterior belief about the model parameters given the data, $P(D|\theta, M)$ is the likelihood of the data given the model parameters, $P(\theta|M)$ is the prior belief about model parameters, and $P(D|M)$ is the marginal likelihood or evidence. These are related by Bayes' rule.

For prediction, the posterior predictive can be computed as follows where $D$ is the training data and $D^*$ is the test data: $P(D^*|D) = \int P(D^*|D, \theta)P(\theta|D)d\theta$. When data points are conditionally independent given model parameters then $P(D^*|D, \theta) = P(D^*|\theta)$, and we can write:

$$P(D^*|D) = \int P(D^*|\theta)P(\theta|D)d\theta$$

The posterior predictive is an example of Bayesian model averaging [80], a posterior weighted average of $P(D^*|\theta)$.

As a result of the No Free Lunch Theorem [192], which states that there is no single best model class that is optimal for all tasks, model classes often need to be compared. Bayesian model selection takes a similar approach to that of Bayesian model parameter selection. We can compute the posterior over model classes as follows:

$$P(M|D) = \frac{P(D|M)P(M)}{P(D)}$$

Assuming a uniform prior over model classes, the maximum a posteriori (MAP) estimate, the mode of the posterior distribution, is the model class that maximizes the marginal likelihood $P(D|M) = \int P(D|\theta, M)P(\theta|M)d\theta$. MacKay [118] discusses

model comparison in the context of Bayesian neural networks.

Bayesian inference is usually intractable due to the integrals involved, unless the prior distribution is conjugate to the likelihood distribution. Unfortunately, conjugate priors exist for only exponential family distributions [132] and so can't be directly applied to complex models like Bayesian deep neural networks.

## 2.2 Bayesian Inference For Deep Learning

As exact Bayesian inference cannot be done for Bayesian deep learning models, approximate inference methods need to be used.

A straightforward approach is to use a Laplace approximation to model the posterior over neural network weights as a Gaussian [115, 43]. Ritter, Botev, and Barber [156] construct a Kronecker factored Laplace approximation to the posterior. Kristiadi, Hein, and Hennig [103] add uncertainty units to a pre-trained network that are trained post-hoc using an uncertainty-aware loss to improve uncertainty under a Laplace approximation.

Proper Markov Chain Monte Carlo (MCMC) methods will always perfectly sample from the posterior given enough time. The MCMC method of choice is Hamiltonian Monte Carlo (HMC) as described in Neal [133] under the name "hybrid Monte Carlo." Betancourt [21] provides an in-depth introduction to HMC. While HMC is notoriously difficult to tune, methods such as the No-U-Turn Sampler of Hoffman and Gelman [81] attempt to automate the tuning. While considered the gold standard, HMC isn't scalable because the method requires gradient computa-

tions that use the entire dataset. Recently, new methods for scaling HMC have been introduced such as Stochastic Gradient HMC [35] which according to the evaluation of Yao et al. [201] performs well in terms of capturing uncertainty. Ma, Chen, and Fox [114] provide a complete framework for constructing MCMC samplers, including stochastic gradient MCMC samplers. Cobb et al. [40] introduce (among other things) a Python package, `hamiltorch`, that simplifies the use of HMC with PyTorch [140]. Welling and Teh [184] introduce a method that uses simpler Langevin dynamics to allow for a stochastic gradient based sampling of the posterior.

There has been interesting recent work around the development of Bayesian coresets, which scale sampling-based inference methods by subsampling and weighting data to produce a high fidelity summary of the entire dataset [86, 30, 31, 29].

As MCMC is hard to scale and tune in practice, variational inference, which converts the integration problem into an optimization problem where the posterior is approximated using a simpler variational distribution, is often used instead [91, 23]. In particular, the exact posterior $p(\cdot|D)$ is approximated by a variational approximation $q(\cdot)$ by minimizing the Kullback-Leibler divergence:

$$q^* = \arg\min_{q \in Q} \mathrm{KL}(q(\cdot)||p(\cdot|D))$$

$$q^* = \arg\min_{q \in Q} \int q(\theta) \log \frac{q(\theta)}{p(\theta|D)} d\theta$$

$$q^* = \arg\min_{q \in Q} \int q(\theta) \log \frac{q(\theta)p(D)}{p(\theta, D)} d\theta$$

$$q^* = \arg\min_{q \in Q} \log p(D) - \int q(\theta) \log \frac{p(\theta, D)}{q(\theta)} d\theta$$

The $\log p(D)$ term does not depend on $q$, so it can be dropped from the optimization, and the subtracted term that remains is called the evidence lower bound (ELBO).[1] So the optimization becomes

$$q^* = \arg\max_{q \in Q} \text{ELBO}(q)$$

The scaling of Bayesian methods to sizable datasets has been largely made possible by stochastic variational inference [82]. Black box automatic stochastic variational inference methods [104, 153, 177, 49] that only require the specification of the model log-likelihood have improved the usability and adoption of Bayesian methods and has sped up model iteration.

Variational inference for neural networks was first introduced in the early 1990's [78]. Graves [72] revived interest in variational inference for neural networks by introducing a stochastic variational method for inference in neural networks, and Blundell et al. [24] improves on this method.

Gal and Ghahramani [62] and Gal [60] introduce an easy to implement approach to variational inference in Bayesian neural networks. In particular, they show that a neural network with dropout [79, 174], a technique commonly used to reduce overfitting in neural networks by randomly dropping units during training, applied before every weight layer, is equivalent to an approximation of a deep Gaussian process [42]. They also show that training with dropout effectively performs variational inference for the deep Gaussian process model. Leaving dropout

[1] ELBO is a lower bound since the KL divergence is always positive, and as a result $\log p(D) \geq$ ELBO.

on at test time allows for sampling from the posterior distribution. Gal, Hron, and Kendall [63] improve uncertainty calibration in these models by automatically tuning the dropout probabilities using gradient methods. Smith and Gal [169] show that training ensembles of these models with different initializations can be beneficial when the true posterior is concentrated in many local modes of similar likelihood.

The connections between Bayesian neural networks and Gaussian processes [154] have long been studied. Neal [133] and Williams [188] discuss the connections between infinitely wide Bayesian neural networks and Gaussian processes. Hybrid combinations of deep neural networks and Gaussian processes such as the Stochastic Variational Deep Kernel Learning model of Wilson et al. [190] use deep neural networks to learn the kernel of a Gaussian process. Amersfoort et al. [2] improve on Stochastic Variational Deep Kernel Learning by using a bi-Lipschitz constraint to encourage the feature extractor to approximately preserve distances. Bradshaw, Matthews, and Ghahramani [25] explore similar models, showing evidence that they are more robust to adversarial examples. Garnelo et al. [67] introduce the neural process which like the Gaussian process defines a distribution over functions and like neural networks is computationally efficient. Rudner et al. [160] establish a connection between Gaussian processes with deep kernels and neural processes. Liu et al. [111] develop an approach to make deep residual neural networks input distance aware (with respect to the training data manifold) by replacing the final layer of the network with a Gaussian process approximation and by employing spectral normalization to preserve distances from the input to the output of the network.

Variational inference methods that leverage the optimization techniques used

in traditional deep learning have also been developed. Maclaurin, Duvenaud, and Adams [119] show how stochastic gradient descent with early stopping can be used for variational inference. In a similar manner, Maddox et al. [120] build off of stochastic weight averaging [88] to fit a distribution using the stochastic gradient descent trajectory. Wilson and Izmailov [189] extend this technique using an ensembling approach to sample around multiple posterior modes. Khan et al. [96] modify the Adam optimizer [98] to perform variational inference. Another Bayesian neural network inference approach that is related [77] to variational inference is the expectation propagation [126] based probabilistic back propagation of Hernández-Lobato and Adams [76].

The quality of uncertainty estimates depends on how well the posterior distribution of the Bayesian neural network is approximated. Yao et al. [201] and Vadera et al. [179] provide comparisons of uncertainty quality for various BDL inference procedures. Variational inference is known to underestimate uncertainty [117]. A related problem, which is highlighted by Giordano, Broderick, and Jordan [70] and Turner and Sahani [178], is that variational inference will not capture all of the true posterior modes as it approximates the posterior with a simpler distribution. This can be problematic for many applications. For example, this underestimation of uncertainty can lead to there being test data that has low uncertainty while being far away from the observed training data, and as a result, out of distribution data such as new classes and adversarial examples cannot be detected [169].

We note that while most work in Bayesian deep learning captures uncertainty in model weights, Antorán, Allingham, and Hernández-Lobato [10] performs

probabilistic reasoning over network depth, extended in Antorán, Allingham, and Hernández-Lobato [11] to perform neural architecture search.

Deep learning does not necessarily need to be Bayesian to provide uncertainty estimates. Lakshminarayanan, Pritzel, and Blundell [105] propose an alternative to Bayesian deep learning that trains an ensemble of randomly initialized models. These deep ensembles are shown to produce competitive uncertainty estimates both in domain [13] and out of distribution [138], due to their ability to explore different modes in function space [58]. Training ensembles can be done efficiently [85, 185] and extended to ensemble over hyperparameters [186].

Wilson and Izmailov [189] show that deep ensembles are not a competing approach to Bayesian deep learning but in fact are an approach for approximate Bayesian marginalization. Wilson and Izmailov [189] use the Bayesian interpretation of deep ensembles to derive MultiSWAG, an improvement on deep ensembles. Pearce et al. [141] also show that modified ensembling leads to Bayesian approximate inference.

Hooker [83] proposes a frequentist testing method for diagnosing extrapolation, a task similar to the estimation of epistemic uncertainty. Munson and Kegelmeyer [131] similarly explore the detection of extrapolation risk. Schulam and Saria [165] proposes an approximation to the bootstrap [52] that is shown to produce predictive distributions competitive to those produced by Bayesian methods.

## 2.3 Uncertainty Metrics

Two kinds of uncertainty can be distinguished [60]. Aleatoric uncertainty is caused by inherent noise and stochasticity in the data. More training data will not help to reduce this kind of uncertainty. Epistemic uncertainty, on the other hand, is caused by a lack of similar training data. In regions lacking training data, different model parameter settings that produce diverse or potentially conflicting predictions can be of comparable likelihood under the posterior.

For classification tasks where epistemic and aleatoric uncertainty don't need to be differentiated, uncertainty can be measured using the predictive distribution entropy:

$$H[P(y|x, D)] = -\sum_{y \in C} P(y|x, D) \log P(y|x, D)$$

Aleatoric uncertainty can be measured using expected entropy:

$$\mathbb{E}_{P(\theta|D)} H[P(y|x, \theta)]$$

Mutual information can be used to measure epistemic uncertainty:

$$I(\theta, y|D, x) = H[P(y|x, D)] - \mathbb{E}_{P(\theta|D)} H[P(y|x, \theta)]$$

Monte Carlo estimates obtained by sampling from the posterior can be used to approximate the terms of these equations for our Bayesian models [169]. In particular, $P(y|x, D) \approx \frac{1}{T} \sum_{i=1}^{T} P(y|x, \theta_i)$ and $\mathbb{E}_{P(\theta|D)} H[P(y|x, \theta)] \approx \frac{1}{T} \sum_{i=1}^{T} H[P(y|x, \theta_i)]$

where the $\theta_i$ are samples from the posterior over models and $T$ is the number of samples. Other methods that measure disagreement between samples, such as the estimated variance of the samples, can be used to estimate epistemic uncertainty as well.

Depeweg et al. [45] demonstrate a decomposition of uncertainty into epistemic and aleatoric components in the context of Bayesian neural networks with latent variables [46] for efficient active learning in the presence of complex noise and for risk sensitive reinforcement learning. Kendall and Gal [94] discuss neural network uncertainty in the context of computer vision, and Filos et al. [55] evaluate the robustness of Bayesian deep learning in the context of diabetic retinopathy diagnosis.

Epistemic uncertainty can be viewed as measuring how far prediction time data is from the manifold induced by the training data. In other words, mutual information can be used to measure distance from the training data distribution. This is useful for detecting out of distribution data as well as uncovering adversarial attacks [169]. Xiao, Gomez, and Gal [195] use epistemic uncertainty to detect out of distribution language data. Gal and Smith [65] develop theoretical results that show that with the right model architecture and good uncertainty estimation, Bayesian neural networks can be immune to adversarial attacks.

We have discussed the threat posed by malware, the current state of machine learning based malware detection, and the tools of uncertainty. In the remaining chapters, we develop and explore new methods for leveraging uncertainty to improve machine learning applied to malware detection. In particular, in the next chapter, we develop a new, uncertainty-based approach to thresholding a model's decision

under extreme false positive rate constraints.

# Chapter 3

# Leveraging Uncertainty for Improved Static Malware Detection Under Extreme False Positive Constraints

The detection of malware is a critical task for the protection of computing environments. This task often requires extremely low false positive rates (FPR) of 0.01% or even lower, for which modern machine learning has no readily available tools. We introduce the first broad investigation of the use of uncertainty for malware detection across multiple datasets, models, and feature types. We show how ensembling and Bayesian treatments of machine learning methods for static malware detection allow for improved identification of model errors, uncovering of new malware families, and predictive performance under extreme false positive constraints. We additionally demonstrate how previous works have used an evaluation protocol that can lead to misleading results.

## 3.1 Introduction

Classifying a new file as benign (safe to run) or malicious (not safe, termed "malware") is a current and growing issue. Malware already causes billions in damages [8, 87], and with healthcare systems increasingly targeted has directly led to deaths [51]. For years most anti-virus (AV) vendors have been seeing at least 2 million malicious new files per month [172], and benign files on a network tend to

outnumber malicious files at a ratio of 80:1 [107]. This creates a common need for malware detection systems to operate with extremely low false positive rates. If false positives are too frequent, then analysts, IT, and support staff have to spend too much work on non-threats while simultaneously interrupting normal workflow. Even with this focus, Computer Incident Response Teams (CIRT) are often dealing with over 50% false positive rates and cite it as the main issue with current tooling [135].

The natural goal for AV style deployments of a malware detector is to maximize the true positive rate (TPR) for some maximally acceptable false positive rate (FPR). Production deployments are often concerned with FPRs of 0.1% at most, and preferably $\leq$0.001%. The issue of low FPR has been recognized since the very first research on machine learning based malware detection [95], yet surprisingly little work has been done to study how to maximize TPR@FPR. We present the first work addressing this gap by applying ideas from ensembling and Bayesian uncertainty estimation to a variety of common malware detection methods in use today on the two largest public corpora. In doing so we develop a number of contributions and previously unrealized insights:

1. All prior malware detection work we are aware of have evaluated TPR@FPR incorrectly or not specified their approach. The common error is to measure the TPR at the desired FPR on the test set, but this presupposes knowledge of the exact threshold to achieve the desired FPR. By instead estimating the threshold on a validation set, we show prior results have often misidentified

their true TPR rates.

2. While the benefits of ensembling have long been known, it is often presumed that significant model diversity is required to obtain meaningful benefit. We show even moderately diverse or Bayesian approaches can significantly improve the TPR, especially for the low-FPR regimes needed for malware detection.

3. By using a Bayesian approach to estimate the epistemic and aleatoric uncertainty of a model on a given sample, we develop a new approach to thresholding a model's decision that can improve TPR and better approximate the desired FPR on unseen data.

4. Malware detection deployment requires detecting novel malware families, an intrinsically out-of-distribution task. We show how epistemic and aleatoric uncertainty relates to errors and novel malware families, allowing for faster detection of new threats.

The rest of the work reported on in this chapter is organized as follows. First we will review the related research to our work in section 3.2. Next we will detail the data, algorithms, and metrics used in all of our experiments in section 3.3. We present extensive experimental results organized by our major contributions in section 3.4, which show that prior TPR estimates could be off by $\geq 35\%$ relative error, that ensembles of limited diversity can raise TPR rates by $\geq 11\%$. Then we leverage uncertainty estimates to show a statistically significant improvement over the naive approach of thresholding models for TPR/FPR trade-offs, and that

our uncertainty estimates is useful to malware analysts in identifying miss-predicted files. Our conclusions are presented in section 3.5.

## 3.2 Related Work

The need for low FP rates has been paramount since the inception of machine learning malware detection research by Kephart et al. [95]. Much of the history in this domain is focused on "signature" like tasks, where the goal was to recognize the set of already known malware, smaller than the total population of malware. This led to works that used training and testing data for evaluation [68, 59]. This approach is not meaningful for determining TPR at any FPR due to over-fitting, and is not tenable due to the now large and growing population of malware with more sophisticated obfuscation techniques. There is no agreed upon threshold for exactly how low FPs should be, with most published work ranging from 0.1% down to $\leq$0.00002% [152, 171, 143, 5, 142, 99, 163, 150]. All of these works *evaluate their false positive rates on the test set, selecting the threshold from the test set that gives them the desired FPR*, and then report the associated TPR. This is an understandable but incorrect approach, because the threshold is selected explicitly from the test set, when our goal is to test the ability of the model to achieve an FPR on unseen data. As far as we are aware, our work is the first in the malware detection space to identify this and propose selecting the threshold from a validation set, and then evaluate the precision of the FPR estimate in conjunction with the obtained TPR.

It is also worth noting that these cited prior works attempt to minimize FPR primarily by feature selection, engineering, or ML process pipeline choices that they believe will result in a more accurate model or be biased towards low FPR. Our approach is instead model agnostic, and seeks to better understand the nature of selecting thresholds to achieve TPR@FPR targets and improving it with uncertainty estimates. The only other work we are aware of that has this goal is from the related domain of spam detection by Yih, Goodman, and Hulten [204] who propose two dataset re-weighting approaches, but also determine their success using the test set. Because they focus on data re-weighting their approach is orthogonal to our own, and both could be used simultaneously. The closest other work we are aware is [50] that develop differentiable approximations to AUC and Precision at a minimum Recall, but their approach does not apply to our situation because we instead need a maximum FP rate and specific points on the AUC curve. Further, our need for very low FPR is problematic for their setting as a mini-batch of data will be insufficient for estimating low FPR.

A number of prior works have investigated diverse ensembles of different kinds of algorithms to improve malware detection accuracy [202, 113, 92, 125, 97], following the common wisdom that averaging over diverse and uncorrelated predictors improves accuracy [191, 26, 89]. As far as we are aware, we are the first to study the performance of small ensembles of low-diversity (i.e, different runs of the same algorithm) and identify their especially large impact on TPR when needing extremely low FPR. This is important for malware detection as a diverse ensemble often necessitates algorithms that are too slow for deployment, and high compute throughput

is critical to practical utility in this domain.

Building upon the use of ensembles, the modeling of uncertainty for decision making is notably missing from the current machine learning for malware literature. An exception is the approach of Backes and Nauman [16] which introduces new classes for uncertain decisions in the context of a simple Bayesian logistic regression model applied to the Drebin Android malware dataset [12, 173]. Understanding when a machine learning model is uncertain about its prediction is critical in high risk applications such as malware detection. When an automated malware detection algorithm is uncertain about a sample, the uncertainty estimate could be used to flag the sample for analysis by a more computationally expensive algorithm or for review by a human. Our work is the first we are aware of to study how modeling uncertainty can be used to improve TPR@FPR scores, and to aid analysts by showing new samples with high uncertainty are more likely to be novel malware families.

## 3.3 Methods

We provide details about the data and machine learning models used in our experiments. The majority of the existing research in machine learning applied to malware detection has focused on the automation of static malware analysis, where a file is analyzed without being run. We will also focus on the static analysis problem.

### 3.3.1 Data

Due to the need to estimate low FPR rates, we use the two largest available corpora for malware detection. These are the EMBER2018 and Sophos AI SOREL-20M datasets. We note that both of these datasets focus on low FPR evaluation, but make the same error in evaluation. Our first results in section 3.3 will show the relative magnitudes of the errors.

We use the EMBER2018 dataset which consists of portable executable files (PE files) scanned by VirusTotal in or before 2018 [6]. The dataset contains 600,000 labeled training samples and 200,000 labeled testing samples, with an equal number of malicious and benign samples in both sets. The malicious samples are also labeled by malware family using AVClass [166]. All of the testing samples were first observed after all of the training samples. EMBER2018 includes vectorized features for each sample encoding general file information, header information, imported functions, exported functions, section information, byte histograms, byte-entropy histograms, and string information [6]. While the 1.1TB of raw PE files are not available as part of EMBER2018, they can be downloaded via VirusTotal. We note that the EMBER2018 dataset was designed to be more challenging for machine learning algorithms to classify than the original EMBER2017 dataset.

We also use the recent Sophos AI SOREL-20M dataset, consisting of 20 million files [74]. The much larger number of data points in the Sophos dataset is advertised as "industry scale" and allows for the exploration of FPR constraints much smaller than allowed by EMBER2018. In particular, the test set size for Sophos consists of

1,360,622 malicious samples and 2,834,441 benign samples. As part of the Sophos dataset release, two baseline models are provided, including a feed-forward neural network (FFNN) and a LightGBM (LGBM) gradient-boosed decision tree model. Five versions of each of the models are pre-trained using different random seeds on the Sophos data using the same featurization as EMBER2018.

### 3.3.2 Models

#### 3.3.2.1 EMBER2018

We apply three models to the EMBER2018 dataset that each rely on different types of features. The first model we apply is a Bayesian deep learning model based on the MalConv model of Raff et al. [151], a convolutional neural network for malware detection that operates on the raw byte sequences of files. We will refer to this model as Bayesian MalConv (BMC). As exact Bayesian inference cannot be done for deep neural networks such as MalConv, approximate inference methods need to be used.

Gal and Ghahramani [62] introduced an easy to implement approach to variational inference in Bayesian neural networks. In particular, they showed that a neural network with dropout, a technique commonly used to reduce overfitting in neural networks by randomly dropping units during training [79, 174], applied before every weight layer is equivalent to an approximation of a deep Gaussian process [42], and that training with dropout effectively performs variational inference for the deep Gaussian process model. The posterior distribution can be sampled from

by leaving dropout on at test time. For Bayesian MalConv, we follow this approach and apply dropout before each fully connected layer of the MalConv model, with a dropout probability of $p = 0.1$. We use the Adam optimizer [98] to train the model, and we produce 16 samples at evaluation time using multiple forward passes on the trained model with dropout left on.

The second model we apply is a Bayesian logistic regression (BLR) model which takes as input the binary presence of 94,225 byte 8-grams [150, 148] selected using LASSO from the one million most common byte 8-grams. Dropout is used in a similar manner to Bayesian MalConv, with a dropout probability of $p = 0.1$ and 16 samples at evaluation time.

The third model we apply is an ensemble of 16 Light Gradient Boosting Machine (LGBM) models [93] trained with different random seeds on the EMBER features as described in Anderson and Roth [6].

#### 3.3.2.2 Sophos

We apply two models to the Sophos dataset that both rely on the EMBER2018 featurization. The first is an ensemble of 5 feed-forward neural network (FFNN) models as described in Harang and Rudd [74], a simplified version of the model from Rudd et al. [159], trained using different random seeds. The second is an ensemble of five LGBM models trained using different random seeds. We use the publicly available pre-trained models provided with the Sophos dataset as our ensemble members for both models. While 5 ensemble members may seem small,

Ovadia et al. [138] found that increasing ensemble sizes beyond 5 has diminishing returns with respect to the quality of the uncertainty estimates, so an ensemble size of 5 may be sufficient. Our results will show that not only are they sufficient, but for our goal of low FPR they can be significantly more effective than has been previously reported.

### 3.3.3 Uncertainty Estimation

The Bayesian framework allows for the principled modeling of uncertainty in machine learning and decision making. Within this framework, probabilities represent degrees of belief as opposed to the frequentist interpretation of probabilities as long run frequencies [110]. Bayesian inference uses Bayes' Theorem to update beliefs (that are represented in the form of probability distributions) when new data is observed.

In the context of machine learning, a Bayesian update takes the following form where $\theta$ represents model parameters, $D$ represents the data, and $M$ represents the model class: $\mathbb{P}(\theta|D,M) = \frac{\mathbb{P}(D|\theta,M)\mathbb{P}(\theta|M)}{\mathbb{P}(D|M)}$, where $\mathbb{P}(\theta|D,M)$ is the posterior belief about the model parameters given the data, $\mathbb{P}(D|\theta,M)$ is the likelihood of the data given the model parameters, $\mathbb{P}(\theta|M)$ is the prior belief about model parameters, and $\mathbb{P}(D|M)$ is the marginal likelihood or evidence. Bayesian inference is usually intractable due to the integrals involved, unless the prior distribution is conjugate to the likelihood distribution. Unfortunately, conjugate priors exist for only exponential family distributions [132] and so can't be directly applied to complex models

like Bayesian deep neural networks.

As exact Bayesian inference cannot be done for Bayesian deep learning models, approximate inference methods need to be used. Given sufficient compute time, Markov Chain Monte Carlo (MCMC) methods can be used to sample from the posterior [133]. Unfortunately, common sampling based approaches are difficult to scale to problems in the malware space where practical dataset sizes are measured in terabytes because they requires gradient computations over the entire dataset. As MCMC is hard to scale in practice, variational inference is often used instead which converts the integration problem into an optimization problem where the posterior is approximated using a simpler variational distribution [23]. Variational inference for neural networks was first introduced in the early nineteen nineties [78], and Graves [72] revived interest in variational inference for neural networks by introducing a stochastic variational method for inference in neural networks.

We note that complicated Bayesian inference is not necessarily needed to provide useful uncertainty estimates. Lakshminarayanan, Pritzel, and Blundell [105] introduce an alternative that trains an ensemble of randomly initialized models. These deep ensembles have been shown to produce competitive uncertainty estimates [138, 13] because they are able to explore different modes in function space [58]. Wilson and Izmailov [189] argue that deep ensembles are not a competing approach to Bayesian deep learning but rather are an effective approach for Bayesian model averaging.

Two kinds of uncertainty can be distinguished [60]. Aleatoric uncertainty is caused by inherent noise and stochasticity in the data. More training data will not

help to reduce this kind of uncertainty. Epistemic uncertainty on the other hand is caused by a lack of similar training data. In regions lacking training data, different model parameter settings that produce diverse or potentially conflicting predictions can be comparably likely under the posterior.

For classification tasks where epistemic and aleatoric uncertainty don't need to be differentiated, uncertainty can be measured using the predictive distribution entropy:

$$H[\mathbb{P}(y|x, D)] = -\sum_{y \in C} \mathbb{P}(y|x, D) \log \mathbb{P}(y|x, D)$$

Aleatoric uncertainty can be measured using expected entropy:

$$u_{alea} = \mathbb{E}_{\mathbb{P}(\theta|D)} H[\mathbb{P}(y|x, \theta)]$$

Mutual information can be used to measure epistemic uncertainty:

$$u_{epis} = I(\theta, y|D, x) = H[\mathbb{P}(y|x, D)] - \mathbb{E}_{\mathbb{P}(\theta|D)} H[\mathbb{P}(y|x, \theta)]$$

Monte Carlo estimates obtained by sampling from the posterior can be used to approximate the terms of these equations for our Bayesian models [169]. In particular, $\mathbb{P}(y|x, D) \approx \frac{1}{T} \sum_{i=1}^{T} \mathbb{P}(y|x, \theta_i)$ and $\mathbb{E}_{\mathbb{P}(\theta|D)} H[\mathbb{P}(y|x, \theta)] \approx \frac{1}{T} \sum_{i=1}^{T} H[\mathbb{P}(y|x, \theta_i)]$ where the $\theta_i$ are samples from the posterior over models and $T$ is the number of samples.

For our ensemble based models which are not explicitly Bayesian (because each ensemble member receives the same weight) but Bayesian inspired, uncertainties can

be computed in a similar way where the $\theta_i$ are no longer samples from a posterior, but instead multiple independent trainings of a model with $T$ different random seeds.

### 3.3.4 Classification Metrics

We use multiple metrics to evaluate and compare approaches.

Accuracy is defined as the percent of correct predictions made. Area under the receiver operating characteristic curve (AUC) is the probability that the classifier will rank a randomly selected malicious file higher in probability to be malicious than a randomly selected benign file. The true positive rate (TPR) is defined as the number of true positives over the sum of true positives and false negatives. The false positive rate (FPR) is defined as the number of false positives over the sum of false positives and true negatives.

An important contribution of our work is to recognize that the TPR obtained at any given FPR on the test set is not the actual measure of interest in malware detection, but an over-fit measure due to the implicit assumption that the correct decision threshold is known at test time. The threshold must be estimated during training or validation, and then applied to the test set. This means we have a target maximum FPR $T_{FPR}$ that we wish to obtain, and a separate *actualized FPR* that is obtained on the test set. In order to capture the trade-off between TPR and actualized FPR constraint satisfaction, we define the following combined metric

Equation 3.1 where $T_{FPR}$ is the desired maximum FPR.

$$C = TPR - \frac{\max(\text{actualized } FPR - T_{FPR}, 0)}{T_{FPR}} \tag{3.1}$$

This metric captures that we have a desired TPR, but penalizes the score based on the degree of violation of the FPR. This is done by a division so that the magnitude of the violation's impact grows in proportion to the target FPR shrinking. This matches the nature of desiring low FPR itself. For example, 90% TPR at a target FPR of 0.1% is still quite good if the actualized FPR is 0.11% ($C = 0.8$), but is unacceptably bad if the target FPR was 0.01% ($C = -9.1$).

## 3.4 Experiments and Discussion

Now that we have discussed the methods of our work and the metrics by which they will be examined, we will show empirical results demonstrating our primary contributions: 1) Evaluating test-set performance thresholds from the test set leads to misleading results at lower FPR, 2) Simple non-diverse ensembles can dramatically improve TPR at any given FPR rate, 3) we can further improve TPR@FPR by explicitly modeling Bayesian uncertainty estimates into our decision process, and 4) these uncertainty estimates have practical benefits to application by showing that errors and previously unseen malware families have uncertainty distributions that place more weight on higher uncertainties. For each of these we will include the empirical results on the EMBER2018 and the Sophos 2020 corpora, and include additional discussion and nuance to how these relate to practical deployment.

### 3.4.1 Misleading Evaluation

A currently accepted practice for evaluating malware detection models under FPR constraints is to report the test set ROC curve. Once the test set ROC curve is produced, the desired FPR rates from the curve are selected to show their associated TPR. This is misleading as in practice the test set is not available when choosing the decision threshold, causing this evaluation procedure to be invalid. Instead, we must recognize that there are a priori *target FPRs* that are the FP rates that we desire from the model, and the *actualized FPRs* which is what is obtained on the test (read, "production") data. Selecting the threshold from the test set hides that the target and actualized FPRs are different, especially for low FPRs that require large amounts of data to estimate. The valid approach to this scenario when evaluating a classifier at different FPRs is to select the thresholds using a validation set. Once the thresholds are selected that obtain the target FPRs, they can be applied to the test set to obtain the actualized FPRs and their associated TPRs. We show the impact this has on the entire TPR/FPR curve in Figure 3.1 which shows the absolute relative error in TPR for a given actualized FPR. Depending on the model and dataset, the resulting TPR for any actualized FPR can change by over 30%, and the relative error generally increases as the FPR decreases. This is expected because low FPRs naturally require more data to estimate: if you want an FPR of 1:1,000 and you want 1,000 FPRs to estimate the threshold from you would expect to need $1,000^2$ =1 million examples.

We note that the Sophos FFNN model seems to be particularly robust with the

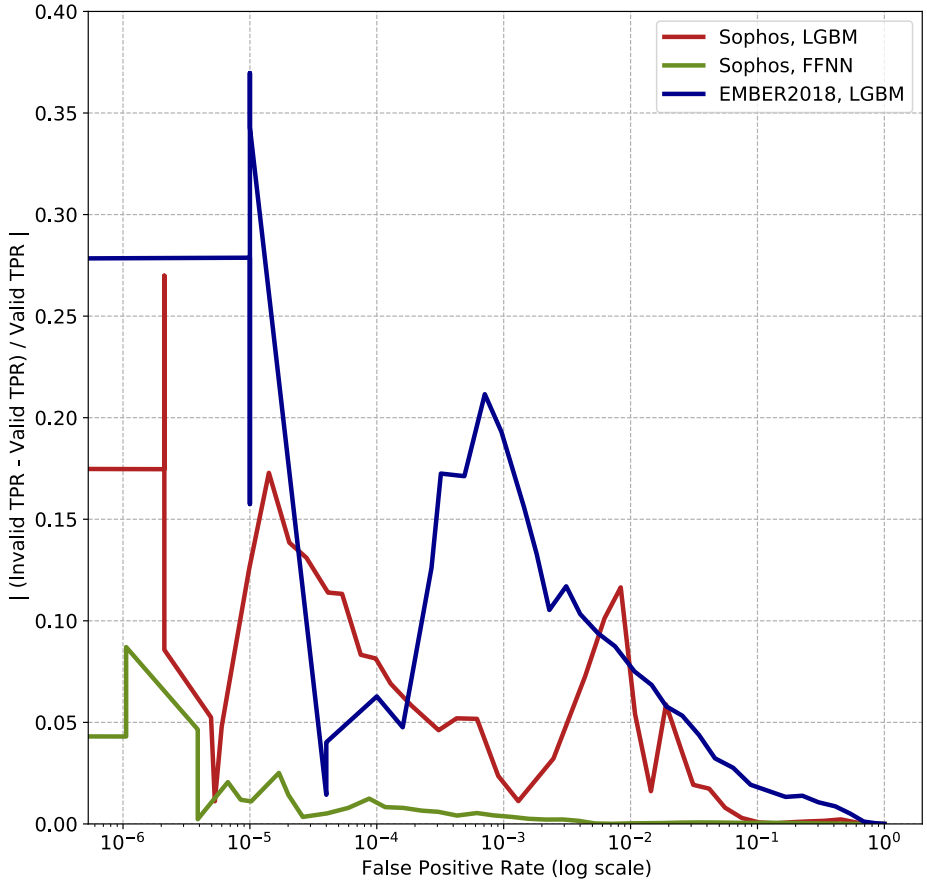



Figure 3.1: Absolute relative error in TPR when using the invalid evaluation protocol, for three different model and dataset combinations. A valid evaluation protocol will use a validation set ROC curve to select a threshold given a desired FPR. The evaluation protocol that is currently the norm in the malware detection literature is invalid because it uses the test set ROC curve, which is never available in practice, to select a threshold. The use of the invalid evaluation protocol can lead to over a 30 percent relative error in TPR. A relative error of zero would mean that the use of the invalid evaluation protocol is not a major issue, but this is clearly not the case.

lowest error in Figure 3.1. This is in part a testament to the FFNN approach, but more broadly a function on the magnitude of the Sophos dataset. With 2.5 million samples in the validation set and 4.2 million in the test set, the corpus is large enough to mitigate the impact of some inappropriate practices. To demonstrate the

impact the validation set can have, we show the same results in Figure 3.2 when only the validation set used to select the threshold is reduced by various orders of magnitude.

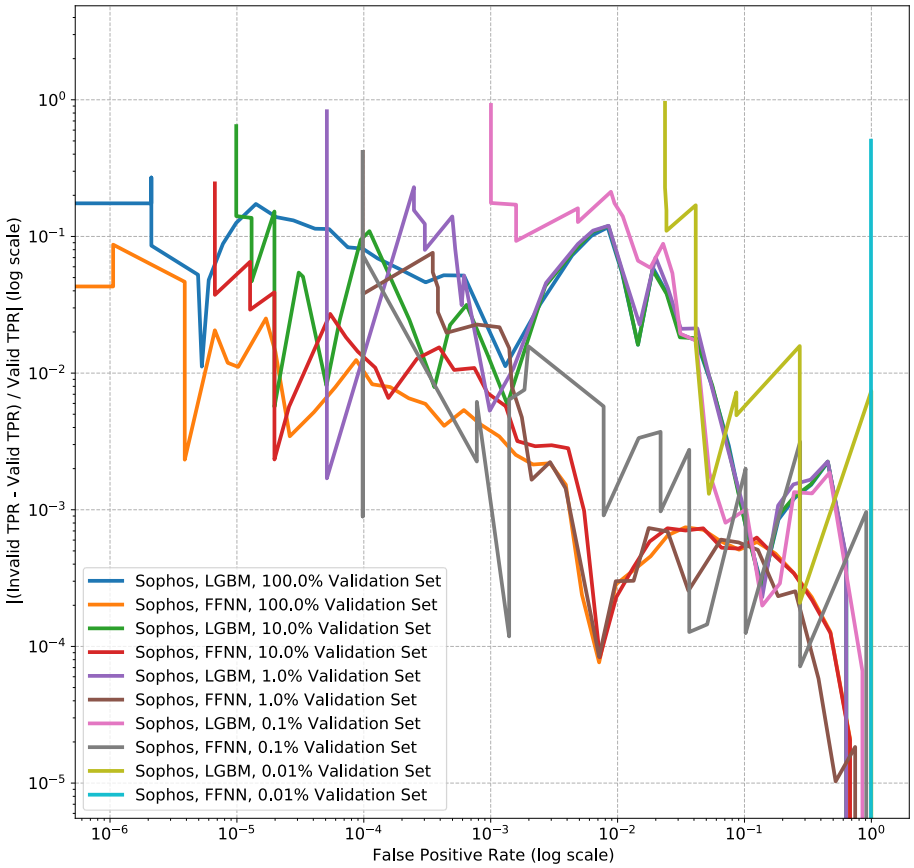



Figure 3.2: Absolute relative error in TPR when using the invalid evaluation protocol at various levels of subsampling of the validation set. As the validation set size decreases, the ability to estimate the FPR decreases. This causes more errors and a "shortening" of the curves as it becomes impossible to estimate lower desired FPR rates.

One can clearly see that as the validation set size decreases, the ability to estimate the FPR decreases. This causes more errors and a "shortening" of the

curves as it becomes impossible to estimate lower desired FPR rates. This last point is important as some prior works have reported FPRs lower than what their dataset could accurately estimate. If the test set size times the desired FPR is less than 100 samples, it is unlikely the TPR@FPR reported will be an accurate estimate (e.g., as done in [5]).

We make explicit to note that this distinction between invalid vs. valid approaches is *not* a critique on the evaluation of entire ROC curves. The fundamental distinction is whether we care about the entire ROC curve, or only specific points from the ROC curve. If we care about the entire ROC curve, evaluating the ROC on the test set is valid and appropriate. But because malware detection is concerned with particular points from the ROC curve, it becomes necessary to evaluate if the approach can hit its desired location on the curve (i.e., a specific FPR in production). There are also valid scenarios to consider just the ROC curve as a whole for malware analysis and its associated AUC, as it represents a metric of ability to rank that is applicable to other scenarios within malware detection and analysis. Our critique is for just those concerned with AV-like deployments that aim for low FPRs specifically.

### 3.4.2 Ensembles

We have now shown that the correct approach to developing a ROC curve when one wishes to evaluate specific points on the curve is to select the threshold from a validation set rather than the test set. We will apply this to the results

of this section to show that creating an ensemble of randomly seeded models can improve the obtained TPR at almost any actualized FPR, especially under extreme FPR constraints. Figure 3.3 shows the ROC curves for individual models as well as for the ensemble consisting of those individual models. The Sophos trained FFNN ensemble notably performs significantly better than any individual member of the ensemble, with the gap in performance widening as FPR becomes smaller.

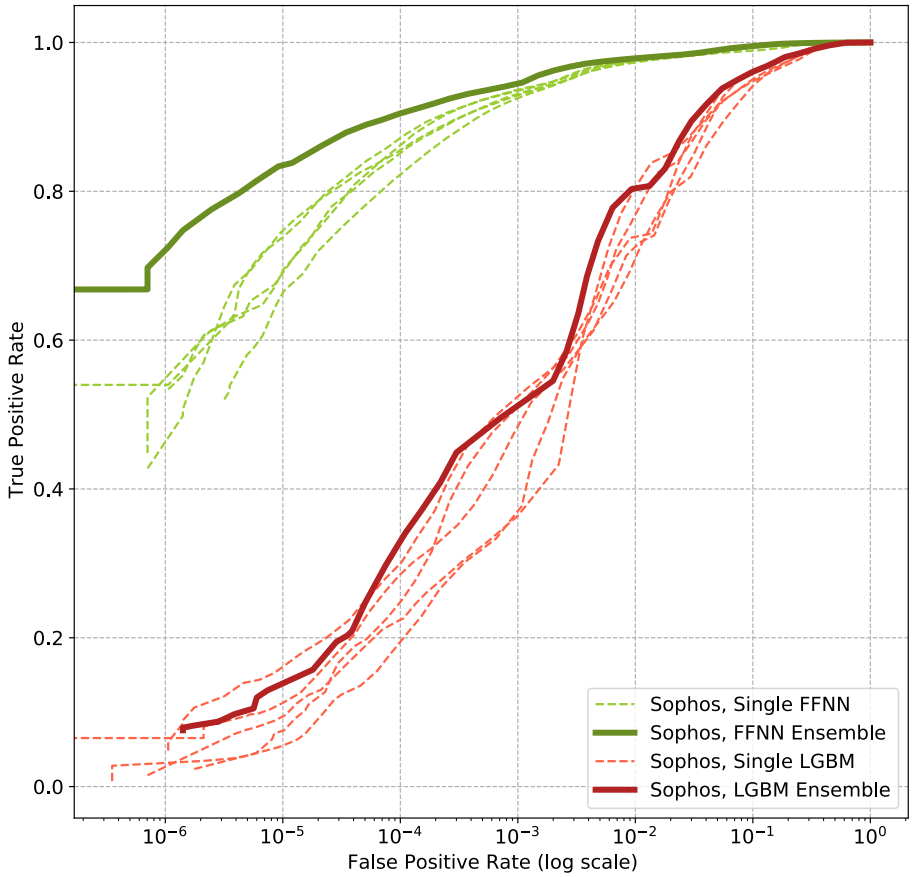



Figure 3.3: Ensembles that take the average of predictions from randomly seeded models can lead to significant TPR gains under extreme FPR constraints, compared to individual models. Note that the gap in performance grows as FPR becomes smaller, in particular for the FFNN model.

Table 3.1: EMBER2018 comparison of the standard global adjustment (labeled as (g)) to the uncertainty aware local adjustments from Equation 3.2 (labeled as (g+l)) and Equation 3.3 (labeled as (g+lv2)). Best combined score (TPR penalized for over-runs on target FPR) shown in **bold**.

| | Target FPR=1% | | | Target FPR=0.1% | | | Target FPR=0.01% | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | TPR | FPR | Comb. | TPR | FPR | Comb. | TPR | FPR | Comb. |
| BMC (g) | 7.602E-01 | 1.177E-02 | 5.832E-01 | 4.998E-01 | 8.100E-04 | 4.998E-01 | 2.422E-01 | 8.000E-05 | 2.422E-01 |
| BMC (g+l) | 7.617E-01 | 1.217E-02 | 5.447E-01 | 4.998E-01 | 8.300E-04 | 4.998E-01 | 2.431E-01 | 9.000E-05 | 2.431E-01 |
| BMC (g+lv2) | **7.594E-01** | **1.166E-02** | **5.934E-01** | **5.016E-01** | **8.200E-04** | **5.016E-01** | **2.434E-01** | **9.000E-05** | **2.434E-01** |
| BLR (g) | 7.778E-01 | 9.550E-03 | 7.778E-01 | 0.000E+00 | 0.000E+00 | 0.000E+00 | **0.000E+00** | **0.000E+00** | **0.000E+00** |
| BLR (g+l) | **7.781E-01** | **9.560E-03** | **7.781E-01** | **5.977E-02** | **7.500E-04** | **5.977E-02** | 1.000E-04 | 8.600E-04 | -7.600E+00 |
| BLR (g+lv2) | **7.781E-01** | **9.560E-03** | **7.781E-01** | 5.248E-02 | 7.800E-04 | 5.248E-02 | 9.000E-05 | 7.900E-04 | -6.900E+00 |
| LGBM (g) | 8.805E-01 | 1.805E-02 | 7.547E-02 | 6.954E-01 | 1.550E-03 | 1.454E-01 | 4.888E-01 | 8.000E-05 | 4.888E-01 |
| LGBM (g+l) | 8.680E-01 | 1.494E-02 | 3.740E-01 | **6.892E-01** | **1.390E-03** | **2.992E-01** | **5.142E-01** | **9.000E-05** | **5.142E-01** |
| LGBM (g+lv2) | **8.693E-01** | **1.488E-02** | **3.813E-01** | 6.917E-01 | 1.430E-03 | 2.617E-01 | **5.142E-01** | **9.000E-05** | **5.142E-01** |

The fact that these are all the same type of model, but with different random seeds at initialization, is an important and not previously recognized phenomena. Classical wisdom is that ensembles should maximize diversity to reduce correlation of predictions, and thus maximize accuracy. But in our scenario malware detection models are designed to be lightweight in model size, low latency, and high throughput, so that the AV system does not interrupt the users of a computing system. A classically diverse model with different types of algorithms or features, as done in prior work in this space, ends up including approaches that are many orders of magnitude larger and slower than the lighter weight approaches we study in this work. Because we can use multiple versions of the same type of model with the same features, we can maintain the high throughput, low latency, and size requirements while obtaining these large gains in TPR.

Table 3.2 compares the accuracy, AUC, and AUC @ $\leq 0.1\%$ FPR achieved by ensembles to the average of those achieved by individual ensemble members. In all cases, the ensemble has better performance than the expected performance

of individual ensemble members, even though they are using an ensemble of low diversity. Of particular importance is the performance of each model in the low FPR domain has a greater relative improvement (median 11% improvement) than one may have anticipated looking at more standard metrics like Accuracy and AUC (median improvements of 0.6% and 0.4% respectively). The only exception to this is the Logistic Regression approaches which have difficulty operating at the extremely low FPR ranges, which we will see repeated.

Table 3.2: Accuracy and AUC for each model and dataset combination. Ensembles are compared to the expected performance of their components. We note that the maximal value that $\text{AUC}_{\leq 0.1\% FPR}$ can take is 0.1.

| Dataset | Model | Accuracy | AUC | $\text{AUC}_{\leq 0.1\% FPR}$ |
|---|---|---|---|---|
| Ember | Bayesian MalConv | **91.64** | **97.47** | **0.04079** |
| | MalConv | 90.88 | 97.05 | 0.03288 |
| | Bayesian Log. Reg. | **94.72** | **98.15** | **0.0** |
| | Log. Reg. | 94.15 | 97.32 | **0.0** |
| | LightGBM Ensemble | **93.98** | **98.62** | **0.06054** |
| | LightGBM | 93.88 | 98.55 | 0.05433 |
| Sophos | FFNN Ensemble | **98.81** | **99.83** | **0.09274** |
| | FFNN | 98.56 | 99.75 | 0.08990 |
| | LightGBM Ensemble | **86.10** | **98.41** | **0.04459** |
| | LightGBM | 85.47 | 98.05 | 0.03637 |

### 3.4.3 Uncertainty Based Threshold Adjustments

While ensembling predictions by taking an average leads to improved results, there is more information within ensembles that can be leveraged to further ameliorate performance under extreme FPR constraints. In particular, estimates can be computed for epistemic ($u_{epis}$) and aleatoric ($u_{alea}$) uncertainty. We introduce a simple threshold adjustment approach that leverages data point specific uncertainty

to locally adjust the threshold based on the uncertainty. We explore two uncertainty aware local threshold adjustments:

$$\hat{y}_{adj} = \hat{y} + \alpha_1 \cdot u_{epis} + \alpha_2 \cdot u_{alea} \tag{3.2}$$

$$\hat{y}_{adj} = \hat{y} + \alpha_1 \cdot \exp(\alpha_3 \cdot u_{epis}) + \alpha_2 \cdot \exp(\alpha_4 \cdot u_{alea}) \tag{3.3}$$

where $\hat{y}$ is the original ensemble prediction for a data point, $u_{epis}$ is the epistemic uncertainty (mutual information) for a data point's prediction, $u_{alea}$ is the aleatoric uncertainty (expected entropy) for a data point's prediction, the $\alpha_i$ are learned scaling factors, and $\hat{y}_{adj}$ is the uncertainty adjusted prediction for a data point. The scaling factors are learned by iteratively optimizing each $\alpha_i$ to maximize TPR given a desired FPR, where after each scaling factor adjustment a new global adjustment is computed. Equation 3.2 and Equation 3.3 capture linear and exponential behaviors respectively.

Because TPR@FPR is not a differentiable metric, we use a gradient free approach to altering the weights $\boldsymbol{\alpha}$. In particular, we use a coordinate descent style approach where we take turns optimizing the individual $\alpha_i$ values while holding all others fixed, repeating the process until convergence. This is feasible thanks to the convex behavior that occurs with respect to the TPR scores. If $\alpha_i$ is set too large, then the associated feature (e.g, $u_{epis}$ or $u_{alea}$) becomes the only effective factor by overshadowing all other components, but is not sufficient on its own to make meaningful predictions, resulting in a low TPR when selecting the target FPR. If

the weight is too small then the associated feature has no impact, and the result is unchanged. This creates the two "low points" and a weight that results in a higher score (hopefully) exists between the two extrema. This can then be selected with high precision by using a golden search by treating the extrema as brackets on a solution. We use Brent's method to solve this because it allows faster searches by approximating the problem with a parabola when possible, and switching to golden search in the worst case, allowing it to solve the optimization quickly.[1]

This gives us an approach to directly optimize our target metric even though it is non-differentiable, and to do so with high precision in just a few minutes of computation. All optimization occurs on the validation set. The $\alpha_i$ are optimized in an alternating manner for Equation 3.2 and in a randomized sequential manner for Equation 3.3. Brent's method is used as the optimizer [27], with a bracketing interval of $[-100, 100]$ for Equation 3.2 and of $[-10, 10]$ for Equation 3.3. A ROC curve is then computed using the locally adjusted $\hat{y}_{adj}$ to obtain the final global threshold. For all methods, when fitting the global and local threshold adjustments using the validation set, a target FPR of 0.9 times the actual desired FPR is used in order to conservatively satisfy the constraint given differences in the test set data.

We briefly note that we had also tried optimizing the uncertainty scaling factors and thresholds jointly using a CMA-ES gradient-free optimization approach [73]. Unfortunately, the obtained solutions were not precise enough given the extremely small FPR constraints, leading us to use the iterative optimization of each variable using Brent's method.

[1] In our experience each call to Brent's method takes less than 100 optimization steps.

The results comparing our new uncertainty augmented local adjustments Equation 3.2 (g+l) and Equation 3.3 (g+lv2) against the naive approach (g) are provided in Table 3.1 and Table 3.3 for the EMBER2018 and Sophos datasets respectively. Bolded are the methods that performed best by the combined score Equation 3.1 which penalizes going over the target FPR. We note that across datasets, models, and target constraints, the inclusion of uncertainty based local adjustments (g+l and g+lv2) improves over the standard use of only a global threshold adjustment (g). Both approaches are statistically significant in their improvement (Wilcoxon-signed rank test p-values of 0.02 and 0.01 respectively). In almost all cases if (g+lv2) is the best performer, (g+l) is the second best. Similarly, when (g+lv2) is not the best, it is usually second best after g+l.

Table 3.3: Sophos comparison of the standard global adjustment (labeled as (g)) to the uncertainty aware local adjustments from Equation 3.2 (labeled as (g+l)) and Equation 3.3 (labeled as (g+lv2)). Best combined score (TPR penalized for over-runs on target FPR) shown in **bold**.

| Target FPR | Test Perf. | LGBM | | | FFNN | | |
|---|---|---|---|---|---|---|---|
| | | (g) | (g+l) | (g+lv2) | (g) | (g+l) | (g+lv2) |
| 1% | TPR | 8.060E-01 | 8.125E-01 | **8.150E-01** | **9.779E-01** | **9.779E-01** | **9.779E-01** |
| | FPR | 1.175E-02 | 1.123E-02 | **1.125E-02** | **8.664E-03** | **8.663E-03** | **8.666E-03** |
| | Comb. | 6.315E-01 | 6.899E-01 | **6.904E-01** | **9.779E-01** | **9.779E-01** | **9.779E-01** |
| 0.1% | TPR | 5.264E-01 | 5.318E-01 | **5.343E-01** | 9.440E-01 | **9.471E-01** | 9.450E-01 |
| | FPR | 1.493E-03 | 1.088E-03 | **9.699E-04** | 9.695E-04 | **9.473E-04** | 1.024E-03 |
| | Comb. | 3.338E-02 | 4.434E-01 | **5.343E-01** | 9.440E-01 | **9.471E-01** | 9.208E-01 |
| 0.01% | TPR | 2.296E-01 | 2.352E-01 | **2.371E-01** | 9.017E-01 | 9.037E-01 | **9.086E-01** |
| | FPR | 4.339E-05 | 5.751E-05 | **5.786E-05** | 8.855E-05 | 9.032E-05 | **8.961E-05** |
| | Comb. | 2.296E-01 | 2.352E-01 | **2.371E-01** | 9.017E-01 | 9.037E-01 | **9.086E-01** |
| 0.001% | TPR | 9.940E-02 | 1.007E-01 | **1.075E-01** | 8.022E-01 | **8.046E-01** | 8.043E-01 |
| | FPR | 3.881E-06 | 4.586E-06 | **4.939E-06** | 4.234E-06 | **5.292E-06** | 5.998E-06 |
| | Comb. | 9.940E-02 | 1.007E-01 | **1.075E-01** | 8.022E-01 | **8.046E-01** | 8.043E-01 |

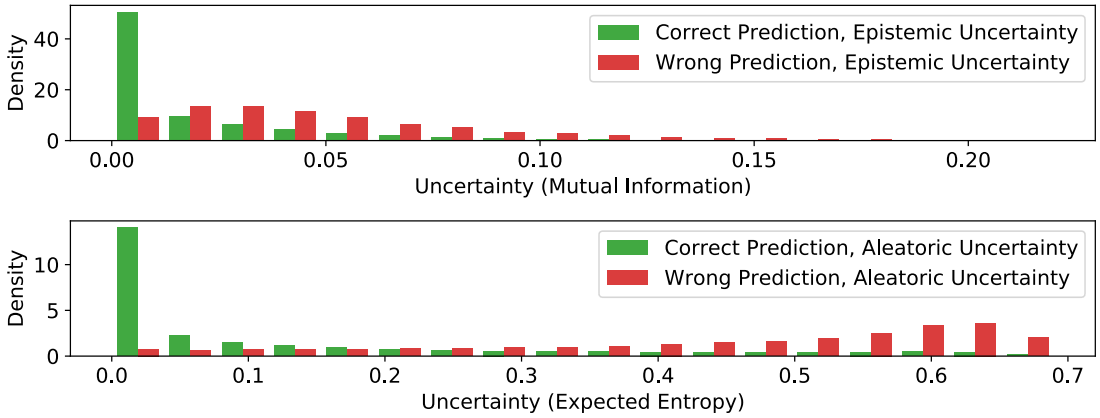
The only case where (g) performed best is when using the Bayesian Logistic

Regression (BLR) model on the EMBER corpus at a target FPR of 0.01%. In this one case we have pushed the model beyond what it is capable of achieving, and all three methods perform poorly - by happenstance the global threshold's degenerate solution of claiming that there is no malware receives a better score due to our uncertainty approaches failing to meet the FPR goal, which has a high penalty. However, we would argue our uncertainty based approaches are still preferable in this scenario because the degenerate model (g) is equivalent to having no anti-virus installed.

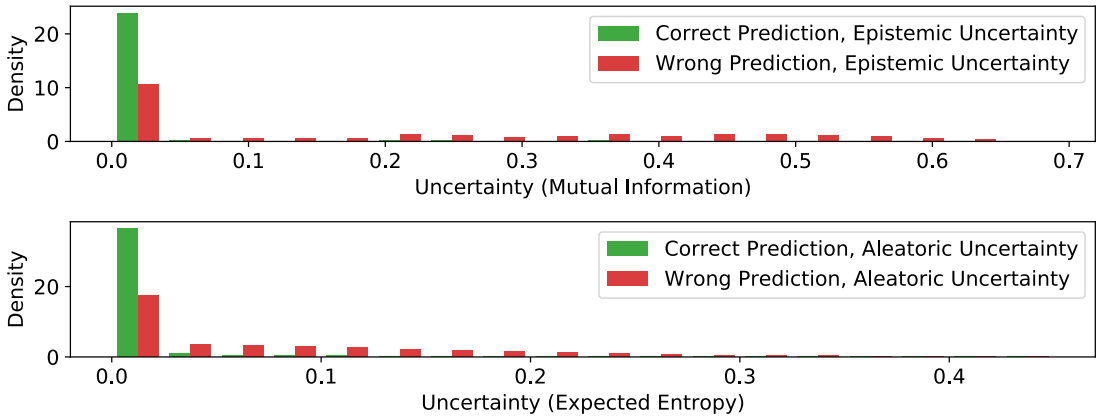
### 3.4.4 Uncertainty on Errors and New AV Classes

Our local threshold adjustments using epistemic and aleatoric uncertainty estimates show improved TPR for extremely low target FPRs. We further investigate how this is possible, and in doing so show that these uncertainty estimates provide an additional benefit to practical application. The errors of our models are correlated with both uncertainty measures. This means we can use the uncertainty measures not just as a means of adjusting the threshold, but as a sourcing tool for analysts. The data with the highest uncertainty scores are the most likely to be miss-predicted, and thus guide the analysts to the samples where their time is best spent.

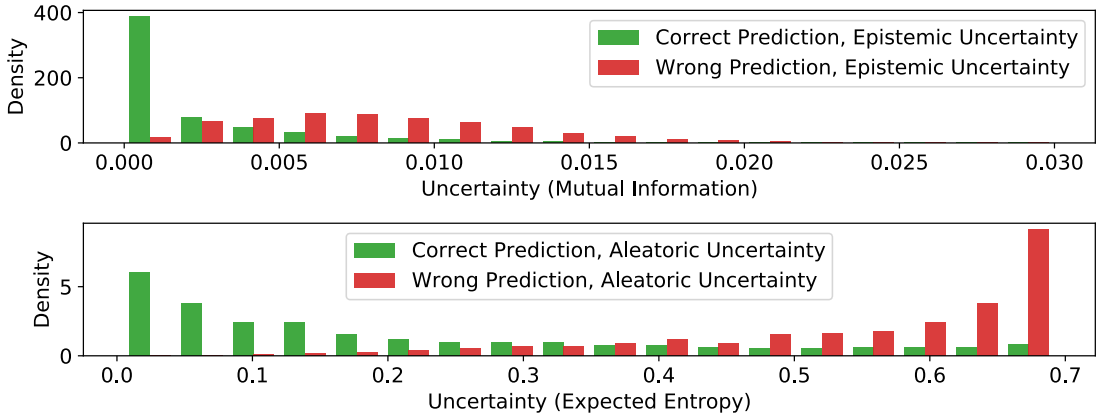
Figure 3.4a, Figure 3.4b, and Figure 3.4c on the EMBER2018 dataset and Figure 3.6b and Figure 3.6a on the Sophos data show that the uncertainty distributions for test samples that the models ultimately got wrong place most of their weight on




(a) EMBER, Bayesian MalConv




(b) EMBER, Bayesian Logistic Regression

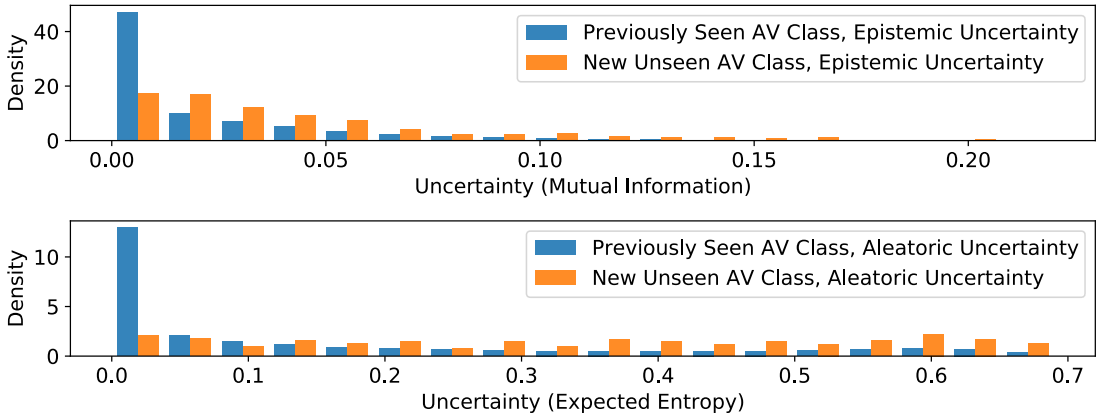



(c) EMBER, LGBM Ensemble

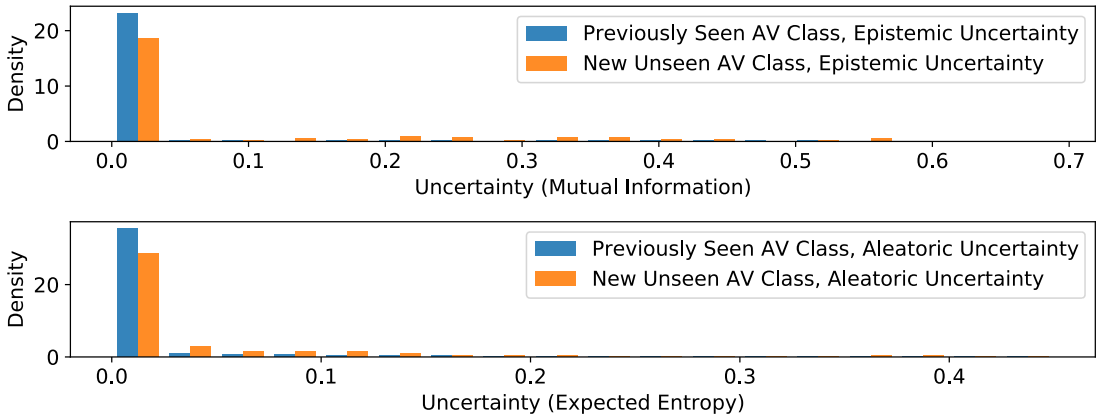
Figure 3.4: A comparison of uncertainty distributions for all three EMBER2018 models at test time between samples predicted correctly and incorrectly.

higher uncertainties. Consistently, the uncertainty distribution for test samples that a model ultimately got right places most of its weight on lower uncertainties. This suggests that overall system performance can be improved by leveraging uncertainty and flagging high uncertainty predictions for further processing and review. This explains the success of our approach, which can learn to use the uncertainty terms as a kind of additional offset. The more we want to lower the FPR rate, the less we should trust the model's outputs if uncertainty is high.

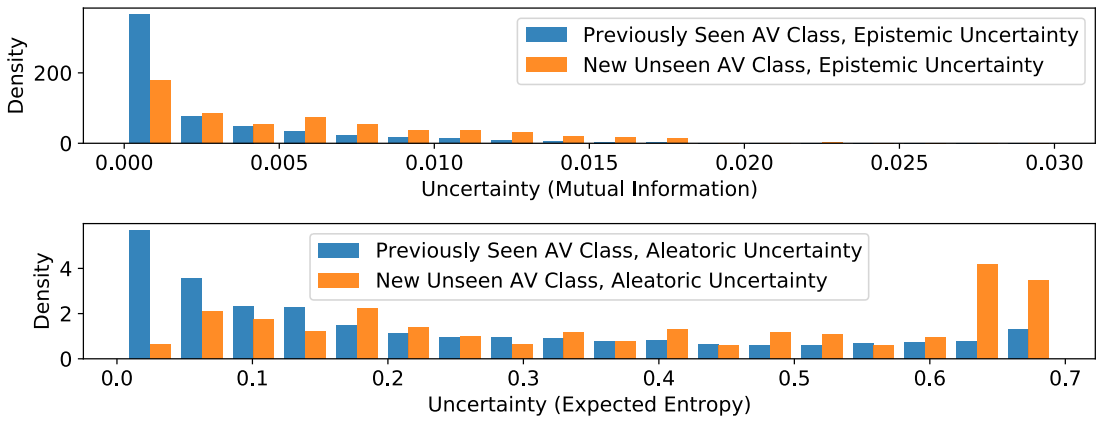
Of the 200,000 files in the EMBER2018 test set, 363 belong to new malware families that were not present in the train set (we note that all the test set are new files that did not exist prior, as the train/test split is a split in time). Figure 3.5a and Figure 3.5c show that the Bayesian MalConv and LightGBM ensemble uncertainty distributions for test samples from malware families seen during training place most of their weight on lower uncertainty values, whereas the uncertainty distributions for novel families not seen during training place most of their weight on higher uncertainty values. Figure 3.5b however shows that the Bayesian logistic regression model mostly does not exhibit this behavior, likely due to the simplicity of the model class which limits the extent to which predictions can disagree. Overall, these results suggest that for some models, uncertainties can be leveraged for the detection of new, out of training distribution, malware families.




(a) Bayesian MalConv

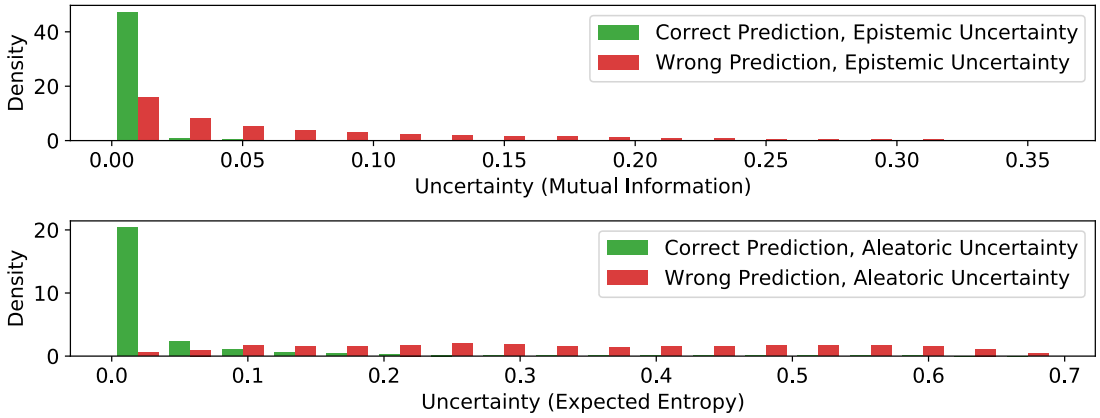



(b) Bayesian Logistic Regression

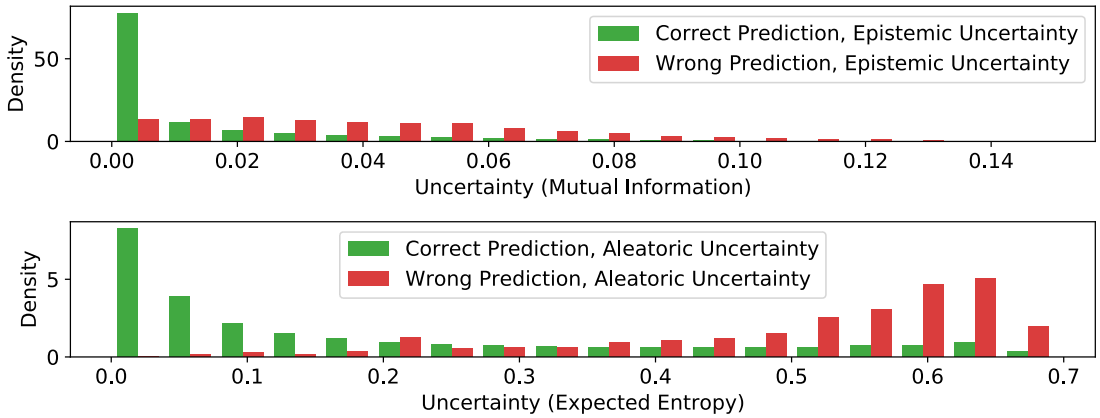



(c) LGBM

Figure 3.5: A comparison of uncertainty distributions for the EMBER2018 models at test time between malware families seen and unseen during training.




(a) Sophos, FFNN Ensemble




(b) Sophos, LGBM Ensemble

Figure 3.6: A comparison of uncertainty distributions for an ensemble of Sophos models at test time on Sophos data between samples predicted correctly and incorrectly.

## 3.5 Conclusions

We have provided evidence that uncertainty estimation using ensembling and Bayesian methods can lead to significant improvements in machine learning based malware detection systems. In particular, improvements were especially large under extreme false positive rate constraints which are common in deployed, production scenarios. Local uncertainty based threshold adjustments were shown to lead to

higher TPRs while satisfying desired FPR maximums. We additionally demonstrated how previous works have used an evaluation protocol that can lead to misleading results, and how uncertainty can be used to better detect model errors and new malware families.

Obtaining uncertainties has an inherent additional computational cost at prediction time which may limit use in resource limited deployed contexts. However, recent advances such as BatchEnsemble [185] have introduced new methods to avoid the computational and memory costs of naive ensembles.

We are currently working with professional malware analysts and teams that believe this approach may benefit them in production environments based on the evidence this work has provided. Future work includes leveraging uncertainty estimates to decide when to run more expensive malware analysis algorithms and techniques such as dynamic analysis, exploring and explaining malware specific drivers of uncertainty, and evaluating these methods over a long period of time in production.

While we have discussed how uncertainty can be used to improve decision making in malware detection algorithms, the question of when a machine learning algorithm should not make a decision is also important. We investigate this question in the next chapter.

# Chapter 4

# Out of Distribution Data Detection Using Dropout Bayesian Neural Networks

## 4.1 Introduction

Detecting out of distribution (OOD) data at test time is critical in a variety of machine learning applications. For example, in the context of malware classification [147], OOD data could correspond to the emergence of a new form of malicious attack. Gal and Ghahramani [62] developed an approach to variational inference in Bayesian neural networks (BNNs) that showed a neural network with dropout [79, 174], a technique commonly used to reduce overfitting in neural networks (NNs) by randomly dropping units during training, applied before every weight layer is equivalent to an approximation of a deep Gaussian process [42]. Training with dropout effectively performs variational inference for the deep Gaussian process model, and the posterior distribution can be sampled from by leaving dropout on at test time. This approach to Bayesian deep learning has been popular in practice as it is easy to implement and scales well.

Measures of uncertainty usually are a function of the sampled softmax outputs of such a BNN, for example predictive entropy and mutual information. There is however useful information at every intermediate layer of a dropout BNN. The

dropout based approach to Bayesian deep learning suffers, like most variational inference methods, from the tendency to fit an approximation to a local mode instead of to the full posterior because of a lack of representational capacity and because of the directionality of the KL divergence [169, 189]. This behavior however allows us to expect the randomized intermediate representation samples in a dropout BNN to be meaningfully related as they are sampled from a local mode.

In this chapter, we explore how to leverage additional information generated at every layer of the network for the task of OOD data detection at test time. In particular, we interpret the intermediate representation of a data point at a particular layer as a randomized embedding. The embedding is randomized due to the use of dropout at test time.

The idea to use a randomized embedding induced by the intermediate layers of a dropout BNN has been attempted previously, but can fail due to the underlying Euclidean distance metric used in previous work. The use of Euclidean distance does not account for the confounding variability caused by changes in embedding magnitudes. We will theoretically justify and empirically show that by instead using a measure based on cosine distance, this problem can be rectified. We then leverage this improved uncertainty estimation to show better OOD data identification across three highly different tasks to demonstrate the robustness of our approach.

The objective of this chapter is not to develop a state-of-the-art approach to OOD data detection, but rather in the context of dropout BNNs to: (1) show how to cheaply improve OOD data detection in systems where a dropout BNN is already deployed, by using intermediate computational results that are already

being computed but not fully leveraged, and (2) provide theoretical and practical evidence to highlight why it is valuable to deconflate angular information about embedding dispersion from embedding norm information. Additionally, previous works have evaluated OOD detection by assuming access to a large OOD dataset of similar size to the in distribution dataset. This is an unrealistic assumption as in areas like cyber security where OOD examples are limited and expensive. So, we also examine the effect of small dataset sizes for OOD detection in our experiments.

## 4.2 Related Work

Two kinds of uncertainty can be distinguished [94]. Aleatoric uncertainty is caused by inherent noise and stochasticity in the data. More training data will not help to reduce this kind of uncertainty. Epistemic uncertainty on the other hand is caused by a lack of similar training data. In regions lacking training data, different model parameter settings that produce diverse or potentially conflicting predictions can be comparably likely under the posterior. OOD data is expected to have higher uncertainty, epistemic in particular. Mukhoti et al. [130] prove that one cannot infer epistemic uncertainty from a deterministic model's softmax entropy, so additional information is needed to estimate epistemic uncertainty.

Uncertainty modeling using probabilistic embeddings has primarily been used for estimating aleatoric uncertainty [137, 167, 38, 33] in tasks such as determining the quality of a test input image. These methods do not easily translate to estimating epistemic uncertainty. For example, Oh et al. [137] try to apply their method on an

epistemic uncertainty estimation task and find that it did not work well for novel classes, and they leave the modeling of epistemic uncertainty as future work.

The only prior work we are aware of that looks at a randomized embedding approach similar to ours is by Terhörst et al. [175], who use dropout at test time to generate a stochastic embedding. They estimate face image quality through the stability of the embedding as measured using Euclidean distance. As we will show, the use of Euclidean distance is problematic as it does not account for factors affecting embedding norms and more generally, the assumptions made by Terhörst et al. [175] are not met in reality. We also note that they are actually estimating epistemic uncertainty (see [137] for an explanation) when test image quality is an inherently aleatoric uncertainty estimation problem. We will show both empirical evidence as well as mathematical grounding as to why our proposed approach, without the addition of any complexity, fixes these issues.

There is evidence that intermediate layers of a neural network contain information useful for epistemic uncertainty estimation and out of distribution detection. Postels et al. [146] establish a connection between the density of hidden representations and the information-theoretic surprise of observing a specific sample in the setting of a deterministic neural network. In particular, they suggest that the first layers of a neural network should be used to estimate epistemic uncertainty due to feature collapse, a phenomena where out-of-distribution data is mapped to in-distribution feature representations in later layers of a network [3, 130], though they also suggest that OOD data detection can benefit from aggregating uncertainty information from several layers. Our work differs from their work as we are not

fitting a density to representations of the training data, increasing the applicability of our approach to situations where fitting and storing a density is not an option for computational or regulatory reasons.

Other recent work has also looked at uncertainty estimation using a single forward pass of a neural network that has had its intermediate representations regularized to produce good uncertainty estimates [3, 111]. We note that many single forward pass based methods like Mukhoti et al. [130] and Liu et al. [111] require residual based networks in combination with spectral normalization to enforce a bi-Lipschitz inductive bias [17]. While the method of Amersfoort et al. [3] is not residual network constrained, it requires significant changes to the model and training procedure. While our approach requires multiple forward passes (as is the case with all dropout BNNs), it does not require any modifications to existing dropout BNNs, by only using information that is already being computed within a dropout BNN.

Mandelbaum and Weinshall [122] propose a confidence score that uses a data embedding derived from the penultimate layer of a neural network. The embedding is achieved using either a distance-based loss or adversarial training. Similarly to other methods, this method requires density estimation, and our work differs as our method does not involve a comparison to nearest neighbors from the training set, which may be difficult to deploy in practice due to both storage and regulatory constraints.

Many works have investigated OOD data detection in probabilistic contexts. Ovadia et al. [138] benchmarks Bayesian deep learning methods in the context of

dataset shift and OOD data at test time. Xiao, Gomez, and Gal [195] use epistemic uncertainty to detect OOD language data. Ren et al. [155] detect OOD data using likelihood ratios in the context of deep generative models and evaluate on OOD genomic sequences. Our work makes a contribution to probabilistic OOD identification by being the first work to systematically investigate the appropriate use of the randomized embeddings induced by the intermediate layers of a dropout BNN.

## 4.3 Methods

In a supervised setting, suppose a neural network structure with $N$ (non-linearity included) layers $f_i, i \in [1, N]$ where $x_1$ is the input and $x_{N+1}$ is the prediction: $x_{i+1} = f_i(x_i)$. Gal and Ghahramani [62] showed that a neural network with dropout [79, 174] applied before every weight layer is equivalent to an approximation of a deep Gaussian process [42], and that training with dropout effectively performs variational inference for the deep Gaussian process model. Dropout randomly omits neural network units. At test time, the posterior distribution can be sampled from by leaving dropout on. This gives us the network structure:

$$x_{i+1} = f_i(\text{dropout}(x_i)) \tag{4.1}$$

### 4.3.1 Randomized Embeddings

#### 4.3.1.1 Computing an Embedding

In the context of a trained dropout Bayesian neural network, we can use the intermediate representations from the various layers (the $x_{i+1}$ in Equation 4.1) as a randomized embedding of a data point. The embedding is randomized as multiple forward passes with dropout on will yield different embedding values. The variation in the embedding values could be used to measure epistemic uncertainty [137], allowing for the detection of OOD data and dataset shift.

#### 4.3.1.2 Measuring Uncertainty

A datum is embedded to a set of randomized embedding values at each layer. We can compute the maximum pairwise distance between the embeddings for a specific datum at a specific layer. This can be done at each layer in the BNN, giving us a feature for each layer that can then be used for tasks such as OOD identification. *All previous work has used Euclidean distance to compute the pairwise distances, without examining the appropriateness of Euclidean distance for the task.* Part of our contribution is an analysis in subsubsection 4.3.3.3 of why Euclidean distance is in fact not appropriate, and we introduce a preferable cosine distance based approach which we use in all of our experiments. A small value of 1e-6 was added to the embeddings to avoid numerical issues caused by corner-case zero normed embedding vectors.[1] In our experiments, embeddings from non-linear layers

[1] We also note that normalized Euclidean distance, where embedding vectors are normalized to unit length prior to computing Euclidean distance, could also be used in place of cosine distance

**Algorithm 1:** Computing Randomized Embedding Based Features for OOD Data Detection

**Input:** A datum $x$, a $N$ layer NN trained with dropout $\{f_1, ..., f_N\}$, and number of samples $T$.

**Output:** $N$ randomized embedding based features $z_1, ..., z_N$, each corresponding to a layer in the network, for a OOD data detection task.

**for** $t \leftarrow$ *1 to* $T$ **do**
  **for** $i \leftarrow$ *1 to* $N$ **do**
    $x_{i+1,t} \leftarrow f_i(\text{dropout}(x_{i,t}))$
**for** $i \leftarrow$ *1 to* $N$ **do**
  $z_i \leftarrow \max(\text{PairwiseCosineDistances}(x_{i,:}))$
**return** $z_1, ..., z_N$ `// Return features.`

(such as convolutions) are flattened prior to computing this metric. A summary of our approach can be found in algorithm 1. The intuition behind this approach is that if measured appropriately, the "spread" or maximal variation in a datum's embedding contains uncertainty information. If all embedding samples are realized to a same point in the embedding space, then there is less uncertainty than if the embedding samples are realized to wildly different parts of the embedding space.

### 4.3.2 Baseline Features

We compare the addition of our randomized embedding based features to a set of common baseline features. For classification tasks, uncertainty estimates in dropout BNNs are usually a function of the sampled softmax outputs. In particular, overall uncertainty can be measured using predictive distribution entropy:

$$H[\mathbb{P}(y|x, D)] = -\sum_{y \in C} \mathbb{P}(y|x, D) \log \mathbb{P}(y|x, D)$$

as its square can be shown to be proportional to cosine distance.

To isolate and measure epistemic uncertainty mutual information can be used:

$$I(\theta, y|D, x) = H[\mathbb{P}(y|x, D)] - \mathbb{E}_{\mathbb{P}(\theta|D)} H[\mathbb{P}(y|x, \theta)]$$

The terms of these equations can be approximated using Monte Carlo estimates obtained by sampling from the dropout BNN posterior [169]. In particular, $\mathbb{P}(y|x, D) \approx \frac{1}{T} \sum_{i=1}^{T} \mathbb{P}(y|x, \theta_i)$ and $\mathbb{E}_{\mathbb{P}(\theta|D)} H[\mathbb{P}(y|x, \theta)] \approx \frac{1}{T} \sum_{i=1}^{T} H[\mathbb{P}(y|x, \theta_i)]$ where the $\theta_i$ are samples from the posterior over models and $T$ is the number of samples. In addition to predictive distribution entropy and mutual information, we also use maximum softmax probability (the value of the largest element of $\mathbb{P}(y|x, D)$) as a feature, shown by Hendrycks and Gimpel [75] to be an effective baseline for the OOD data detection task.

### 4.3.3 How to Measure Embedding Dispersion

We will now explore why Euclidean distance as used by previous works is not appropriate to measure randomized embedding dispersion. We illustrate using a LeNet5 [200] model with added dropout before each layer trained on MNIST, with MNIST variants as OOD data. Further data, model, and experimental details correspond to those expanded upon in subsection 4.4.1.

#### 4.3.3.1 The Problem With Euclidean Distance

Terhörst et al. [175] suggest the Euclidean distance to measure when a data point is suitable for a downstream task, where lower variability in the stochastic

embedding induced by a dropout neural network suggests higher suitability for a data point. In particular, they use the sigmoid of the negative mean Euclidean distance between all stochastic embedding pairs for a data point as the measure of suitability. In other words, their hypothesis is that a form of uncertainty can be measured using the Euclidean distance between embedding samples.

We find that if Euclidean distance is used as the metric to measure distance between samples, their hypothesis holds only with excessive training and likely over-fitting. Figure 4.1a shows that with enough training to get to the accuracy plateau (10 epochs of training with a batch size of 64, with a test accuracy of 0.9885), we actually see the opposite effect. Embeddings for OOD data are actually less spread out than embeddings for in distribution data. Figure 4.1b shows that with excessive training (100 epochs of training, with a lower test accuracy of 0.9882), we see that the hypothesis holds better but note that there is still a good amount of overlap between the histograms, limiting the usefulness for OOD detection (and adding a difficult to select stopping criteria). We note that what we are observing is *not* feature collapse.

This points to two issues that we need to resolve. First, how can we get consistent behavior regardless of over/under-training? Second, how can we more usefully measure spread in a way that matches intuition?

#### 4.3.3.2 Spectral Normalization Stabilizes Behavior

Spectral normalization rescales the weights during training with the spectral norm of the weight matrix, enforcing a Lipschitz constraint that bounds the derivative of the learned function [127]. This helps to preserve distance as a data point makes its way through the network. Figure 4.1c shows that a spectral normalized version of the network results in consistent behavior even with longer training (100 epochs of training, with a test accuracy of 0.9927). So, there is a solution to the first problem. However, we still see that the spread for OOD data is lower than for in distribution data.

#### 4.3.3.3 Why Cosine Distance Is Needed To Properly Measure Embedding Dispersion

Previous research around OOD detection has noted that a lower maximal softmax output value is correlated with a data point being OOD [75]. One possible explanation could be logits (softmax inputs) of smaller norm. This would make intuitive sense as potentially, less neurons would activate for OOD data since OOD data would lack the in distribution features the network is looking for.

The squared Euclidean distance between vectors $\mathbf{u}$ and $\mathbf{v}$ can be written as, where $\theta$ is the angle between $\mathbf{u}$ and $\mathbf{v}$:

$$||\mathbf{u} - \mathbf{v}||^2 = ||\mathbf{u}||^2 + ||\mathbf{v}||^2 - 2\,||\mathbf{u}||\,||\mathbf{v}||\,\cos\theta \tag{4.2}$$

If embedding norms are inherently smaller for OOD data, then Euclidean distance which is norm dependent cannot be used to compare embedding spread across OOD and in distribution datasets, due to confounding. As shown in Equation 4.2, angular information is affected by norm in both an additive and multiplicative manner with Euclidean distance. So, assuming confounding caused by systematic norm differences, cosine distance should be used to isolate the angular information when measuring embedding dispersion. If Euclidean distance mostly captures information already captured by the norm, then the benefit of being Bayesian for this task is not fully leveraged as norm can be estimated with a single point estimate. *To take full advantage of a dropout BNN, angular information about embedding dispersion needs to be deconflated from embedding norm information.*

We explored this hypothesis and found it to be empirically true and formally justifiable. In Figure 4.2a, Euclidean distance is used to measure embedding dispersion, we see that dispersion is correlated with the logits norm and that the relationship is nearly identical for OOD and in distribution data. This means that measuring the spread of the embeddings using Euclidean distance conveys little extra information than just looking at the norm of the logits. In Appendix subsection A.3.2, we perform a simulation to further illustrate this problem in the case of a two layer ReLU activated network.

We want to measure spread in a way that is independent of the embedding norm. This can be done a couple of different ways. For example, a simple switch to cosine distance could be used, or the embeddings could be normalized prior to using Euclidean distance (which can be shown to be related to cosine distance). As

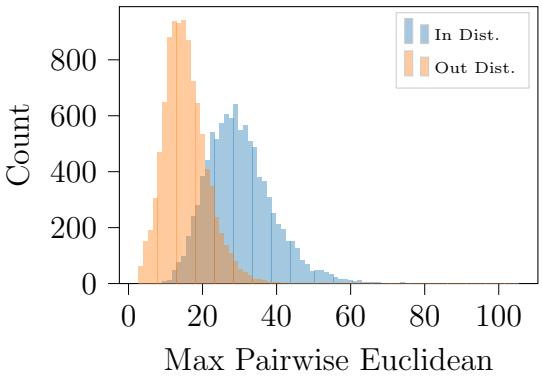
illustrated in Figure 4.2b, using cosine distance results in OOD and in distribution data having behaviors that are no longer identical. Appendix section A.3 shows similar results in an unsupervised setting involving a stacked denoising autoencoder variant.

Figure 4.1d shows the same information as Figure 4.1a, except a cosine distance based measure of spread is used instead of a Euclidean based one. With cosine distance, we now see the expected behavior of OOD having more spread than in distribution, and we see a better separation as well which is good for OOD detection. We have shown results for the last layer of a network but note that a similar analysis can be done for each layer. Having shown empirical evidence for why angular information needs to be isolated from norm information when measuring embedding dispersion, we next provide a formal analysis for why cosine distance allows for an additional source of information.

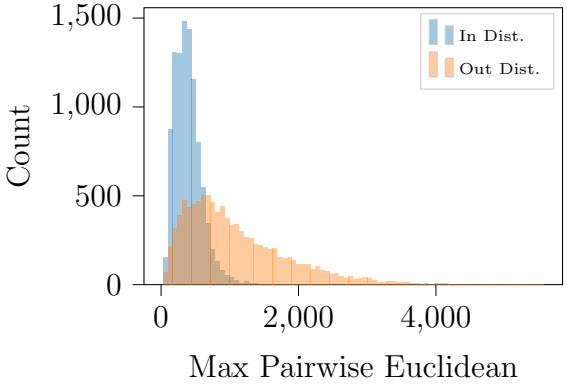
#### 4.3.3.4 Formal Analysis of Cosine Embedding Dispersion

We aim to compute a metric that is invariant to the relative magnitudes among embedding samples, and also accurately represents the dispersion of the embedding samples. In the following, we argue that the mutual information score is not satisfactory for these two objectives. Our goal is not to replace the mutual information as an uncertainty measure, but rather to demonstrate that our pairwise cosine similarity yields an additional source of information that is not captured otherwise.

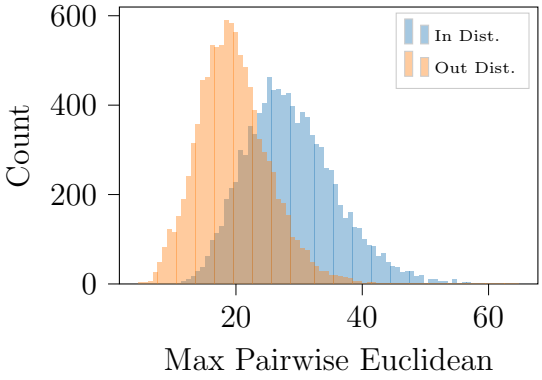
Let $\{z_i\}_{i=1}^{m}$ denote $m$ embedding vectors sampled through dropout. The mu-



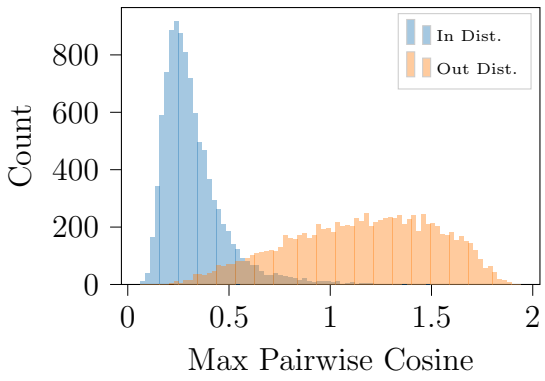

(a) LeNet5 with 10 epochs of training, Euclidean based measure of embedding dispersion.




(b) LeNet5 with 100 epochs of training, Euclidean based measure of embedding dispersion.




(c) Spectral Normalized LeNet5 with 100 epochs of training, Euclidean measure of dispersion.




(d) LeNet5 with 10 epochs of training, cosine based measure of embedding dispersion.

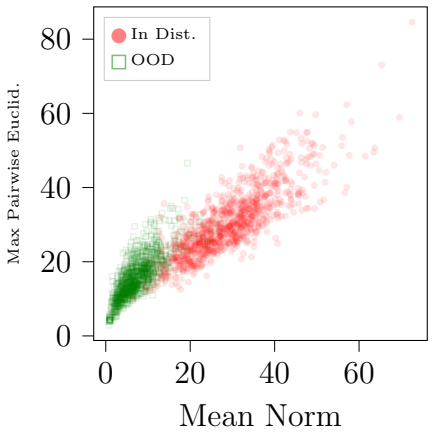
Figure 4.1: Comparison of last layer randomized embedding dispersion distributions for in distribution data (MNIST) and OOD data (Not-MNIST).

tual information score is defined as

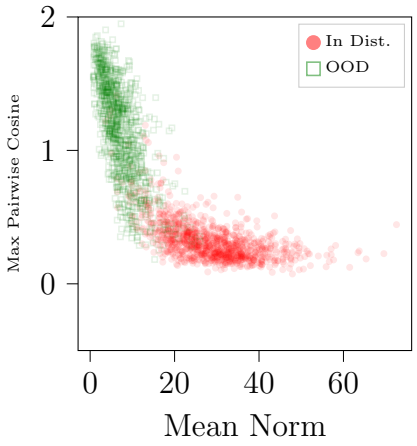
$$I(w, y|D, x) = H[p(y|x, D)] - \mathbb{E}_{p(w|D)} H[p(y|x, w)]$$

and is approximated by

$$\hat{I}(w, y|D, x) = H\left[\frac{1}{m}\sum_{i=1}^{m} \text{softmax}(z_i)\right] - \frac{1}{m}\sum_{i=1}^{m} H\left[\text{softmax}(z_i)\right]$$




(a) Euclidean distance.




(b) Cosine distance.

Figure 4.2: A comparison of the relationships between last layer randomized embedding mean norm and the maximum pairwise distance for Euclidean and cosine distances respectively, for in distribution data (MNIST) and OOD data (Not-MNIST). Both models using LeNet5 trained for 10 epochs.

where $H(\cdot)$ is the entropy function $H(y) = -\sum_i y_i \log y_i$.

We first introduce a theorem from Amos [4] that clarifies the geometric properties of the softmax function. The proof is readily shown using Lagrange multipliers.

**Theorem 4.3.1.** *The softmax function* $\text{softmax}(x)_j = \frac{\exp(x_j)}{\sum_i \exp(x_i)}$ *is a map from* $\mathbb{R}^d$ *to the* $(d-1)$*-simplex that satisfies*

$$\text{softmax}(x) = \underset{0<y<1}{\arg\min} -x^\top y - H(y) \quad \text{s.t.} \quad 1^\top y = 1$$

From this we see that the softmax solution is a balance between two competing objectives: maximizing $x^\top y$ which aims to place all weight on the coordinate with the largest $x_i$ value, and maximizing the entropy of $y$ which steers toward the uniform vector with value $1/d$. In addition, the softmax temperature changes the relative weighting, which allows us to evaluate the effect of the magnitude of the embedding

vector. We leverage this for a further Lemma and Theorem:

**Lemma 4.3.2.** *The softmax function with temperature $\alpha$, defined by* $\text{softmax}(x/\alpha)$, *satisfies*

$$\text{softmax}(x/\alpha) = \underset{0<y<1}{\arg\max}\, x^\top y + \alpha H(y) \quad \text{s.t.} \quad 1^\top y = 1$$

*Proof.* From the previous theorem we get $\text{softmax}(x/\alpha) = \arg\min_{0<y<1} -(x/\alpha)^\top y - H(y)$ s.t. $1^\top y = 1$. Multiplying by scalar $\alpha$ and switching the optimization to maximizing the negative does not change the optimal solution, yielding the statement above. □

These facts help indicate that softmax-based metrics are not suited for assessing the angular dispersion among vectors. We note that the mapped vector is $\alpha$-dependent and hence dependent on the $L_2$ magnitude of the input vector. Furthermore, arbitrary translations of the vector, which can completely change the direction of the vector, do not impact the softmax. These observations are formalized below.

**Theorem 4.3.3.** *The softmax function is invariant to translation of input vector $x$. It is not invariant to scaling $x$ except in the special case when $x_1 = x_2 = \ldots = x_d$. Furthermore, as the magnitude of $x$ increases (without changing direction), the softmax shifts weight to the vertex of the simplex corresponding to the largest coordinate in $x$.*

*Proof.* Invariance to translation follows from observing that $\text{softmax}(x + K) = \exp(x_j + K)/\sum_i \exp(x_j + K) = \exp(x_j)/\sum_i \exp(x_j) = \text{softmax}(x)$.

The dependence on scaling follows from Lemma 1.2. Consider two vectors

$x, x'$ such that $x' = x/\alpha$. The value of $\alpha$ adjusts the scale of the $H(y)$ term. Since the $\max x^\top y$ objective aims to shift weight in $y$ to the largest $x$ coordinate and the $\max H(y)$ objective aims to distribute weight evenly, their solutions do not coincide, giving softmax($x$) and softmax($x'$) different solutions. In the special case that $x_1 = x_2 = \ldots = x_d$ then $x^\top y$ is constant, so the optimization of $H(y)$ gives the uniform distribution vector. Otherwise, increasing the magnitude of $x'$ is equivalent to sending $\alpha \to 0$, which decreases the contribution of $H(y)$. This causes the solution vector to shift weight to the element with largest value in $x$. □

We confirm this analysis by simulation in Appendix subsection A.3.3, where we find that our new cosine-based feature adds an orthogonal measure of information that is not captured in previously used measures of uncertainty.

## 4.4 Experiments and Results

In this section, we evaluate the value of randomized embedding based features across three different OOD data detection tasks in the vision, language, and malware domains. All experiments were implemented in PyTorch [140], and neural networks were optimized using Adam with the default recommended settings [98]. A dropout probability of $p = 0.1$ was used, and when sampling from the base neural network models to compute features for OOD detection, 32 samples are used. Experiments were run on an 80 CPU core machine with 512GB of RAM using a single 16GB Tesla P100 GPU. Experiment specific details are described in their respective sections.

We explore the use of two model classes for the OOD detection algorithms.

The first model is an L2-regularized logistic regression (LR) with the regularization strength chosen using 3-fold cross-validation. We min-max scaled the input features for the LR model to the range $[0, 1]$ based on the training data. The second model is a 500 tree random forest (RF) classifier. We choose these two models to assess linear vs. non-linear behavior in the OOD detection task. We also explore the effect of varied, small training set sizes for the OOD task in all of our experiments. In many production contexts such as cyber security, examples of OOD data are limited and usually expensive to obtain.

### 4.4.1 Image Classification

For our vision experiments, similarly to the evaluation protocol from [3, 155, 146, 130] we explore MNIST variants as OOD data. In particular, we train our base model, a LeNet5 [200] with added dropout before each layer, on MNIST and use Kuzushiji-MNIST [39], notMNIST [28], and Fashion-MNIST [194] as OOD data. When training the downstream OOD data detection algorithms, we train the OOD detector on one of the OOD datasets and test on the other two. For example, we first train a digit classifier on MNIST. Then, we train an OOD data detector that uses randomized embedding based features from the digit classifier to classify MNIST vs. notMNIST. Then we test the OOD data detector on MNIST vs. Kuzushiji-MNIST and Fashion-MNIST.

Due to its importance in practical use, we will test the sample efficiency of the OOD tasks (i.e., how few samples of OOD are needed to detect future OOD data).

In particular, we evaluate performance, as measured by area under the receiver operating characteristic curve (captures desired data ordering performance) and accuracy (captures desired decision making value), using training datasets consisting of $n$=1000, 100, and just 10 data points from each class (in distribution and OOD). We note that this differs from most previous works which have evaluated by assuming access to a large OOD dataset of similar size to the in distribution dataset, an often unrealistic assumption. Each experiment was run 100 times with random training set samples, where all appropriate data not in the training set is included in the test set, and we report a mean and standard deviation for each. In all of our experiments, the standard deviations are much smaller than effect sizes, so we report only the means in this section, and standard deviations can be found in appendix section A.1.

#### 4.4.1.1 Detecting OOD Data

Table 4.1 compares performance with and without the cosine embedding spread features for various experimental configurations and OOD detection models for a dropout LeNet5 trained for 100 epochs. Features labeled as "Last" consist of common baseline features computed using softmax output samples from the network (predictive entropy, mutual information, and maximum softmax probability). Features labeled as "Last+Spread" consist of these baseline features plus our additional randomized embedding maximum cosine spread features for each layer.

The inclusion of the additional cosine spread features improves OOD detection performance consistently across datasets, training set sizes, and model types.

In limited cases where the "Spread" features do not improve the LR model, the RF model with "Spread" features performs the best overall, suggesting that the relationship is not necessarily linear. Table A.4 in the Appendix summarizes results from a similar experiment where the base model is a spectral normalized dropout LeNet5 trained for 100 epochs. A comparison of Table 4.1 and Appendix Table A.4 suggests that, while spectral normalization is not required to see an improvement from the inclusion of cosine spread features, spectral normalization does improve OOD detection performance consistently.

In Appendix section A.4, we further examine the need for a small amount of OOD training data, evaluate Euclidean based spread features, and investigate the feature importances associated with our cosine spread features.

### 4.4.2 Language Classification

Out of distribution data detection is also of interest in natural language processing, where systems are trained to work on specific languages, and inputs from other languages are considered OOD [195]. For these experiments, we train a Char-CNN [205] with dropout added before every layer to classify languages using the WiLI dataset [176]. Training consisted of 50 epochs with a batch size of 128, where the 100 most common characters in the training set (after stripping accents) were used as the vocabulary and each datum was truncated/padded to a length of 200 characters. We train the language classification model to distinguish between French, Spanish, German, English, Italian, and Portuguese text. We use Basque,

Table 4.1: Performance with and without the cosine randomized embedding spread features for various experimental configurations for a dropout LeNet5 trained on MNIST. Features labeled as "Last" consist of common baseline features computed using softmax output samples from the network (predictive entropy, mutual information, and maximum softmax probability). Features labeled as "Last+Spread" consist of these baseline features plus our additional randomized embedding maximum cosine spread features for each layer. Each experiment was repeated multiple times, and the mean is reported here while the standard deviation is reported in Appendix A. Best results are shown in **bold**.

| OOD | | | Num/Class | n=1000 | | n=100 | | n=10 | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Metric | AUC | Acc | AUC | Acc | AUC | Acc |
| Train | Test | Model | Features | | | | | | |
| Fashion | Kuzushiji | LR | Last | 0.969 | **0.914** | 0.967 | 0.909 | 0.963 | 0.884 |
| | | | Last+Spread | **0.979** | 0.914 | **0.973** | **0.911** | **0.967** | **0.901** |
| | | RF | Last | 0.960 | 0.917 | 0.952 | 0.905 | 0.942 | 0.884 |
| | | | Last+Spread | **0.979** | **0.922** | **0.974** | **0.921** | **0.969** | **0.907** |
| | notMNIST | LR | Last | 0.966 | 0.912 | 0.965 | 0.909 | 0.960 | 0.879 |
| | | | Last+Spread | **0.983** | **0.932** | **0.979** | **0.925** | **0.967** | **0.892** |
| | | RF | Last | 0.959 | 0.920 | 0.950 | 0.903 | 0.938 | 0.880 |
| | | | Last+Spread | **0.985** | **0.940** | **0.976** | **0.924** | **0.963** | **0.901** |
| Kuzushiji | Fashion | LR | Last | 0.973 | 0.920 | 0.972 | 0.917 | 0.967 | 0.899 |
| | | | Last+Spread | **0.989** | **0.948** | **0.983** | **0.937** | **0.978** | **0.922** |
| | | RF | Last | 0.964 | 0.920 | 0.956 | 0.907 | 0.946 | 0.896 |
| | | | Last+Spread | **0.986** | **0.943** | **0.978** | **0.931** | **0.967** | **0.914** |
| | notMNIST | LR | Last | 0.967 | 0.914 | 0.965 | 0.910 | 0.960 | 0.886 |
| | | | Last+Spread | **0.984** | **0.931** | **0.975** | **0.914** | **0.966** | **0.888** |
| | | RF | Last | 0.960 | 0.922 | 0.950 | 0.904 | 0.938 | 0.888 |
| | | | Last+Spread | **0.982** | **0.935** | **0.971** | **0.921** | **0.954** | **0.896** |
| notMNIST | Fashion | LR | Last | 0.966 | 0.911 | 0.957 | 0.906 | 0.959 | 0.893 |
| | | | Last+Spread | **0.978** | **0.937** | **0.969** | **0.928** | **0.977** | **0.925** |
| | | RF | Last | 0.960 | 0.910 | 0.955 | 0.904 | 0.946 | 0.887 |
| | | | Last+Spread | **0.988** | **0.943** | **0.982** | **0.935** | **0.978** | **0.920** |
| | Kuzushiji | LR | Last | 0.960 | **0.900** | 0.946 | **0.893** | 0.951 | 0.882 |
| | | | Last+Spread | **0.966** | 0.893 | **0.949** | 0.886 | **0.967** | **0.902** |
| | | RF | Last | 0.956 | **0.906** | 0.950 | 0.901 | 0.941 | 0.883 |
| | | | Last+Spread | **0.978** | 0.906 | **0.973** | **0.915** | **0.969** | **0.906** |

Polish, Luganda, Finnish, Tongan, and Xhosa as out of distribution languages. All of our in and out of distribution languages are chosen to use the Latin writing

system. For the OOD task, training sets consisted of $n$=100, 50, 25, and 10 data points from each class (in distribution and OOD). Each experiment was run 100 times with random training data subsamples, where all languages not trained on are tested on. Table 4.2 shows that the inclusion of our randomized embedding based features consistently improves OOD detection across experimental settings, with average and maximal AUC improvements of 0.06 and 0.15.

Table 4.2: Performance with and without the cosine randomized embedding spread features for a Char-CNN with dropout added before every layer trained to classify languages using the WiLI dataset. Standard deviations are reported in Appendix A, and best results are shown in **bold**.

| OOD | | | Num/Class | n=100 | | n=50 | | n=25 | | n=10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Metric | AUC | Acc | AUC | Acc | AUC | Acc | AUC | Acc |
| Train | Test | Model | Features | | | | | | | | |
| Basque | rest | LR | Last | 0.888 | 0.798 | 0.883 | 0.794 | 0.882 | 0.792 | 0.878 | 0.786 |
| | | | Last+Spread | **0.926** | **0.843** | **0.919** | **0.836** | **0.921** | **0.835** | **0.926** | **0.828** |
| | | RF | Last | 0.862 | 0.797 | 0.857 | 0.793 | 0.851 | 0.792 | 0.842 | 0.789 |
| | | | Last+Spread | **0.924** | **0.845** | **0.920** | **0.840** | **0.918** | **0.835** | **0.914** | **0.824** |
| Finnish | rest | LR | Last | 0.888 | 0.795 | 0.885 | 0.792 | 0.881 | 0.790 | 0.883 | 0.786 |
| | | | Last+Spread | **0.910** | **0.818** | **0.908** | **0.818** | **0.909** | **0.821** | **0.910** | **0.818** |
| | | RF | Last | 0.864 | 0.794 | 0.858 | 0.792 | 0.850 | 0.789 | 0.840 | 0.783 |
| | | | Last+Spread | **0.913** | **0.821** | **0.907** | **0.821** | **0.905** | **0.818** | **0.904** | **0.814** |
| Luganda | rest | LR | Last | 0.891 | 0.806 | 0.889 | 0.803 | 0.887 | 0.800 | 0.881 | 0.794 |
| | | | Last+Spread | **0.943** | **0.864** | **0.939** | **0.854** | **0.935** | **0.847** | **0.931** | **0.837** |
| | | RF | Last | 0.866 | 0.800 | 0.859 | 0.797 | 0.854 | 0.796 | 0.843 | 0.785 |
| | | | Last+Spread | **0.936** | **0.862** | **0.930** | **0.852** | **0.926** | **0.843** | **0.921** | **0.831** |
| Polish | rest | LR | Last | 0.900 | 0.824 | 0.897 | 0.821 | 0.891 | 0.816 | 0.887 | 0.812 |
| | | | Last+Spread | **0.939** | **0.866** | **0.938** | **0.864** | **0.935** | **0.860** | **0.934** | **0.852** |
| | | RF | Last | 0.870 | 0.793 | 0.860 | 0.787 | 0.854 | 0.783 | 0.850 | 0.780 |
| | | | Last+Spread | **0.937** | **0.871** | **0.932** | **0.863** | **0.928** | **0.855** | **0.922** | **0.841** |
| Tongan | rest | LR | Last | 0.857 | **0.815** | 0.841 | **0.811** | 0.815 | 0.800 | 0.791 | 0.771 |
| | | | Last+Spread | **0.886** | 0.811 | **0.877** | 0.810 | **0.884** | **0.819** | **0.880** | **0.813** |
| | | RF | Last | 0.765 | 0.699 | 0.766 | 0.695 | 0.769 | 0.684 | 0.785 | 0.701 |
| | | | Last+Spread | **0.915** | **0.847** | **0.913** | **0.845** | **0.906** | **0.836** | **0.903** | **0.823** |
| Xhosa | rest | LR | Last | 0.894 | 0.807 | 0.891 | 0.804 | 0.886 | 0.800 | 0.879 | 0.794 |
| | | | Last+Spread | **0.944** | **0.866** | **0.940** | **0.857** | **0.933** | **0.846** | **0.931** | **0.838** |
| | | RF | Last | 0.864 | 0.787 | 0.857 | 0.782 | 0.849 | 0.778 | 0.846 | 0.774 |
| | | | Last+Spread | **0.939** | **0.868** | **0.934** | **0.860** | **0.928** | **0.852** | **0.921** | **0.835** |

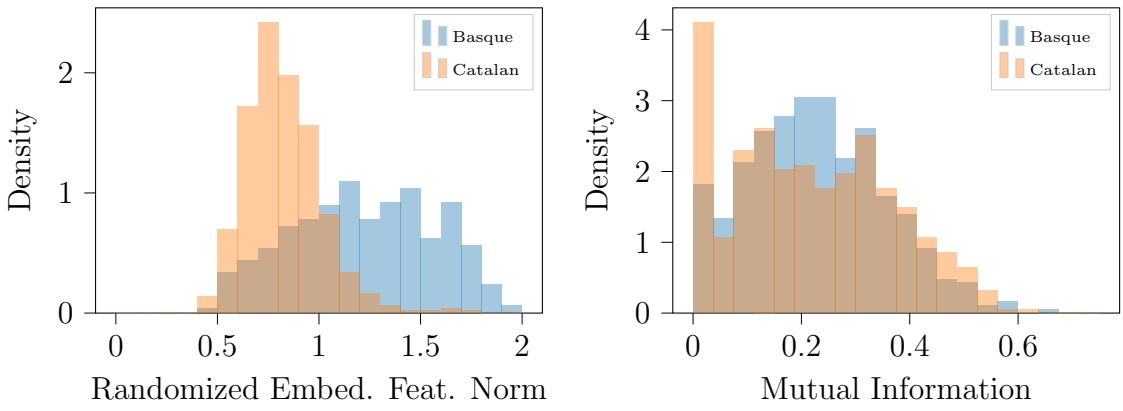



Figure 4.3: While Basque is a language isolate that linguistically does not share any significant similarities to any other languages, Catalan is a Romance language with many linguistic similarities to French and Italian. We expect good estimates of epistemic uncertainty to capture the property that Catalan is "less OOD" than Basque is. Our cosine based embeddings (left) show this desired property. Prior work using MI (right) is unable to meaningfully distinguish any difference between the languages.

We note that while OOD data detection is usually treated as a purely binary classification task by most previous work, OOD versus in distribution is a false binary. There are different levels and degrees of how OOD data can be. In the context of language, we can examine the nuances between different flavors of OOD data. While Basque is a language isolate that linguistically does not share any significant similarities to any other languages, Catalan is a Romance language with many linguistic similarities to French and Italian (and Spanish to a lesser extent). While both Basque and Catalan are considered OOD in our setting, we expect good estimates of epistemic uncertainty to capture the property that Catalan is "less OOD" than Basque is. Figure 4.3 shows that this desired property is captured by the norm of our randomized embedding features, while the mutual information distributions for Basque and Catalan are nearly indistinguishable.

### 4.4.3 Malware Detection

Finally, we evaluate the usefulness of our randomized embedding based features in the context of malware detection. Uncovering new or significantly different malware is of particular interest in the quickly evolving cyber security space. We use a dropout variant of the MalConv model [151], a convolutional NN for malware detection that operates on raw byte sequences. We apply dropout before each fully connected layer of MalConv. Applying dropout to only the last layers of a NN corresponds to using maximum a posteriori (MAP) estimates for the initial layers and Bayesian estimates for the later layers [61]. We train the dropout MalConv model for 5 epochs with a batch size of 32 on the EMBER2018 dataset which consists of portable executable files (PE files) scanned by VirusTotal in or before 2018 [6].

We run two experiments on the Bayesian MalConv model. First, of the 200000 files in the EMBER test set, 363 have as their top most likely malware family label (as labeled by AVClass [166]) a family that was not present in the train set. We evaluate OOD detection performance first on these unseen malware families. Second, we evaluate OOD detection performance on a different malware dataset containing malware samples obtained from a Brazilian financial entity [32]. The malware from this dataset could be considered as OOD due to differing geographical specificity and intent, leading to the use of malware tactics, techniques, and procedures likely specific to a Brazilian banking target. There are also temporal differences as the Brazilian samples were all collected before the EMBER dataset, and we additionally only used malware first seen by VirusTotal before 2012. OOD task training sets

consisted of $n$=100, 50, and just 25 data points from each class (in distribution and OOD). Each experiment was run 100 times with random train/test splits, where all of the data not in the training set is included in the test set. Results are summarized in Table 4.3, showing that the inclusion of our randomized embedding based features consistently improves OOD detection across experimental settings. Because of the high class imbalance in this use case, as access to good OOD data is more limited in the malware domain, we reported the ROC AUC and the recall for the OOD class in Table 4.3, noting that recall is often the primary metric of interest in practice for cyber security.

Table 4.3: Performance with and without the cosine randomized embedding spread features for a MalConv model with dropout added before each fully connected layer trained to detect malware using EMBER2018. Standard deviations are reported in Appendix A, and best results are **bolded**.

| OOD | | Num/Class | n=100 | | n=50 | | n=25 | |
|---|---|---|---|---|---|---|---|---|
| | | Metric | AUC | Recall | AUC | Recall | AUC | Recall |
| Experiment | Model | Features | | | | | | |
| EMBER2018 | LR | Last | 0.789 | 0.704 | **0.786** | 0.682 | **0.778** | 0.650 |
| | | Last+Spread | **0.793** | **0.718** | 0.783 | **0.689** | 0.766 | **0.658** |
| | RF | Last | 0.757 | 0.735 | 0.752 | 0.727 | 0.748 | 0.714 |
| | | Last+Spread | **0.791** | **0.784** | **0.782** | **0.764** | **0.770** | **0.743** |
| Brazilian | LR | Last | 0.685 | **0.645** | 0.680 | 0.607 | 0.668 | 0.584 |
| | | Last+Spread | **0.741** | 0.620 | **0.734** | **0.617** | **0.712** | **0.605** |
| | RF | Last | 0.724 | 0.693 | 0.705 | 0.674 | 0.679 | 0.652 |
| | | Last+Spread | **0.839** | **0.797** | **0.813** | **0.772** | **0.776** | **0.736** |

## 4.5 Conclusions

We have demonstrated why previous attempts at measuring randomized embedding dispersion using Euclidean distance are inherently flawed. Then we introduced and theoretically justified a cosine distance based, lightweight approach to test time OOD data detection in the context of dropout Bayesian neural networks. Information that is already computed is used as randomized embeddings, training dataset information does not need to be stored, additional regularization methods are not needed (though do help), and auxiliary neural networks do not need to be trained to take advantage of this additional information. While we note that our approach is limited to dropout BNNs, the popularity of the dropout approximation to BNNs and the existence of previous works exploring the use of stochastic embeddings based on dropout BNNs suggests the applicability of our approach to practice. Our approach can be deployed anywhere a dropout BNN is already deployed with minimal additional overhead. Future work includes the investigation of more elaborate features based off of the randomized embeddings.

We have discussed the use of uncertainty to improve malware detection as well as out of distribution data detection in single algorithm settings. In the next chapter, we evaluate the practical impact uncertainty quantification can have on multi-algorithm malware detection systems.

# Chapter 5

# When should we run more expensive analysis?

## 5.1 Introduction

The detection of malware is critical as malware has already caused billions in damages [8, 87] and cyber attacks on healthcare systems have directly led to deaths [51]. Detecting malware is a computationally challenging task due to the scale of the volume and velocity of incoming data. Many anti-virus (AV) vendors see over 2 million new malicious files each month [172], and benign files on a network tend to outnumber malicious files at a ratio of 80:1 [107].

A vast array of machine learning based malware detection algorithms have been developed with the goal of classifying new files as benign (not malware) or malicious (contains malware) [147]. Unsurprisingly, machine learning based malware detection algorithms vary widely in their strengths, weaknesses, and computational costs (associated with both model execution and especially feature extraction).

In the context of machine learning based malware detection systems that leverage multiple algorithms, we demonstrate how uncertainty quantification combined with auxiliary prediction targets can minimize compute costs while maximizing overall system accuracy. In particular, we estimate uncertainty using Bayesian and ensembling based approaches, and we train computationally cheaper algorithms to predict the success of more expensive algorithms. This results in an easy to imple-

ment approach applicable to a wide variety of AV systems.

We first demonstrate how CAPA based capability detection in executable files can be sped up by using cheaper machine learning algorithms capable of abstaining on uncertain predictions. We then demonstrate how malware detection can be improved by using a static analysis feature based machine learning algorithm that defers to a dynamic analysis feature based machine learning algorithm when the static model is unsure and predicts that the dynamic model will do well. This improves malware detection accuracy while minimizing the use of costly dynamic analysis.

## 5.2 Related Work

### 5.2.1 Learning to Reject/Defer

Backes and Nauman [16] suggest the running of more expensive malware analysis when the initial analysis results in a high uncertainty prediction as an interesting question for future malware detection research. This problem in its general form has been studied in the machine learning space, and we adapt and apply these ideas to the malware detection task.

Madras, Pitassi, and Zemel [121] introduce adaptive rejection learning, also known as learning to defer. Learning to defer extends and generalizes rejection learning [41, 37, 36, 15, 182] by learning to adapt to the strengths, weaknesses, and biases of the downstream decision maker. There is a connection to the gating network approach of adaptive mixtures of local experts, the difference being that

the decision maker is assumed to be a fixed expert under learning to defer [89]. Mozannar and Sontag [129] use the method of Madras, Pitassi, and Zemel [121] as a baseline and also tackle the problem of learning to defer. Similarly, Wilder, Horvitz, and Kamar [187] introduce methods to optimize teams consisting of humans and machine learning algorithms.

### 5.2.2 Active Feature Acquisition

An area related to learning to defer is active feature acquisition, the sequential decision process of whether to query more features or not under budget constraints. Noriega-Campero et al. [136] discuss active feature acquisition in the context of fair classification. Gao and Koller [66] introduce an active classification process that combines multiple classifiers and features at test time using an instance specific decision path. Similarly, Saar-tsechansky, Melville, and Provost [161] use value of information to rank feature acquisition options. There are many ways to measure the value of information such as information gain, classification loss, and decision robustness for example [34]. Xu et al. [197] and Xu et al. [198] develop algorithms that select features individually for each test data point and reduce trees of classifiers into cascades of classifiers. While cascades and trees of models are popular [112, 183], Xu, Weinberger, and Chapelle [196], instead of using a cascade of classifiers, extend stage-wise regression to directly incorporate feature extraction cost into the objective during training to minimize test time computation. Des Jardins, MacGlashan, and Wagstaff [47] constructs a cascaded ensemble of classifiers to selectively acquire

missing features at both train and test time. Wang et al. [182] develop prediction cascades that use additional augmenting classifiers that evaluate the distributional output of the earlier classifier to estimate its uncertainty. Many cascade based methods have an assumed ordering of cascade members, unable to adaptively reorder members based on the datum at hand, and Gao and Koller [66] differs by balancing expected classification gain with computational cost where observations are selected dynamically based on previous observations to achieve instance-specific decision paths. Active feature acquisition can also be formulated as a Markov Decision Process [168, 158, 48].

More broadly, this problem is fundamentally related to the task of estimating the value of information [54], a highly multidisciplinary problem with roots in information theory [124]. Behrens et al. [19] show evidence that humans modulate their learning rates based on environment volatility and uncertainty, and in a way that can be predicted by a Bayesian learner. This supports the hypothesis of Bayesian reasoning in humans. Feltham [54] provide a formal framework for measuring the value of information in the context of accounting. In particular, a framework to calculate the expected payoff for an information system from the perspective of the decision maker is developed.

## 5.3 Data and Models

As in chapter 3, we use the EMBER2018 dataset, we use a dropout Bayesian MalConv model when predicting on raw binaries, and a LGBM model is used when

predicting on the EMBER extracted features. In this chapter, we additionally develop a model for dynamic analysis logs, and we also extract capabilities in the executable files using CAPA with the SMDA recursive disassembler as a backend.[1]

For a comparison of computational costs, on an NVIDIA Tesla P100 GPU, a 16 sample Bayesian MalConv model can process about 51.2 EMBER2018 files a second on average (about 0.02 seconds per file for feature extraction, prediction, and sampling). The EMBER features take on average about 0.09 seconds to extract per file (note that this excludes the downstream prediction model, so the actual cost is slightly higher). CAPA feature extraction takes about 45.75 seconds per model on average (this also excludes the downstream model). Finally, running dynamic analysis on a file (excluding downstream prediction models) takes an average of 526 seconds (8 minutes and 46 seconds). In other words, running MalConv on a file is over $26,300$ times faster than running a dynamic analysis based model.

### 5.3.1 Dynamic Analysis Features and Model

We used a proprietary dynamic execution sandbox provided by a United States based software security company. The dynamic analysis tool uses libvirt to work with QEMU[2] to manage virtual machines (VMs) running Windows 7 32-bit.

The dynamic analysis tool first puts the sample for analysis into the VM, starts its custom monitoring application, starts the sample, saves the monitor's log file, and pulls the log from the VM to the host. Then the tool kills the VM image,

[1] https://github.com/mandiant/capa
[2] https://www.qemu.org/

and creates a new one from a template so that the VM is clean for every sample.

These dynamic analysis logs were featurized by taking a domain knowledge driven approach similar to that of Anderson and Roth [6]. The dynamic analysis engine captures broad information about DNS, registry, file, process, and security events from the dynamic execution of the sample. These events contain raw information which we used to then extract the following features for each file: event subtype counts, event subtype information counts, number of unique processes by user, PID counts by user, event key frequencies, count of registry bytes written, registry key path information, timing information, authentication signature information, hashed event message, hashed command line information, and counts of unnamed events. We then train a LGBM model on top of these features.

We note that it is possible that some executables did not exhibit all their behavior or functionality during dynamic analysis because they were not in the proper environment (for example if the executable targets a specific Windows build), did not have access to all the resources needed for execution (for example if the executable was unable to connect to the internet or to its command-and-control server), or had some built-in anti-reverse engineering techniques (for example the executable might wait a few days before executing). In other words, it is likely that some malicious files may not have necessarily exhibited malicious behaviors during dynamic analysis. As a result, even though dynamic analysis is more expensive than static analysis, it does not always result in better malware detection when used in isolation. We will show that judiciously running dynamic analysis on a subset of files will improve malware detection accuracy while keeping analysis costs low.

## 5.4 Predicting Capabilities in Executable Files

While most machine learning based approaches to malware have focused on the detection of maliciousness, there has been less work on the identification of behaviors, which is a critical component to malware analysis [170]. We investigated the prediction of CAPA outputs using an 8 sample Bayesian MalConv model as well as an 8 sample LGBM ensemble trained on the EMBER features. We used as prediction targets the 350 CAPA rules from the Mandiant standard collection [3] plus a target for failures, so a total of 351 non-exclusive labels for each file. Figure 5.1 shows detection accuracy for each CAPA rule for both MalConv and the LGBM model. We note that accuracy is high across rules, suggesting that CAPA may not need to be run for all samples when outcomes can be predicted. The LGBM model is consistently more accurate than MalConv across rules at the cost of higher cost associated with feature extraction. Figure 5.2 shows the AUC for each CAPA rule for both MalConv and the LGBM model. While LGBM generally does better than MalConv in terms of AUC, the LGBM model has worse worst cases.

Running CAPA only on files where a prediction model such as MalConv or LGBM is uncertain would greatly reduce computational costs (CAPA is more than 2200 times more expensive time-wise compared to MalConv mainly due to the costs associated with disassembly). In other words, we would like to enable MalConv and LGBM to abstain from making a prediction on high uncertainty samples that we then would run CAPA on. Figure 5.3 shows the performance of such an approach for

[3] https://github.com/mandiant/capa-rules

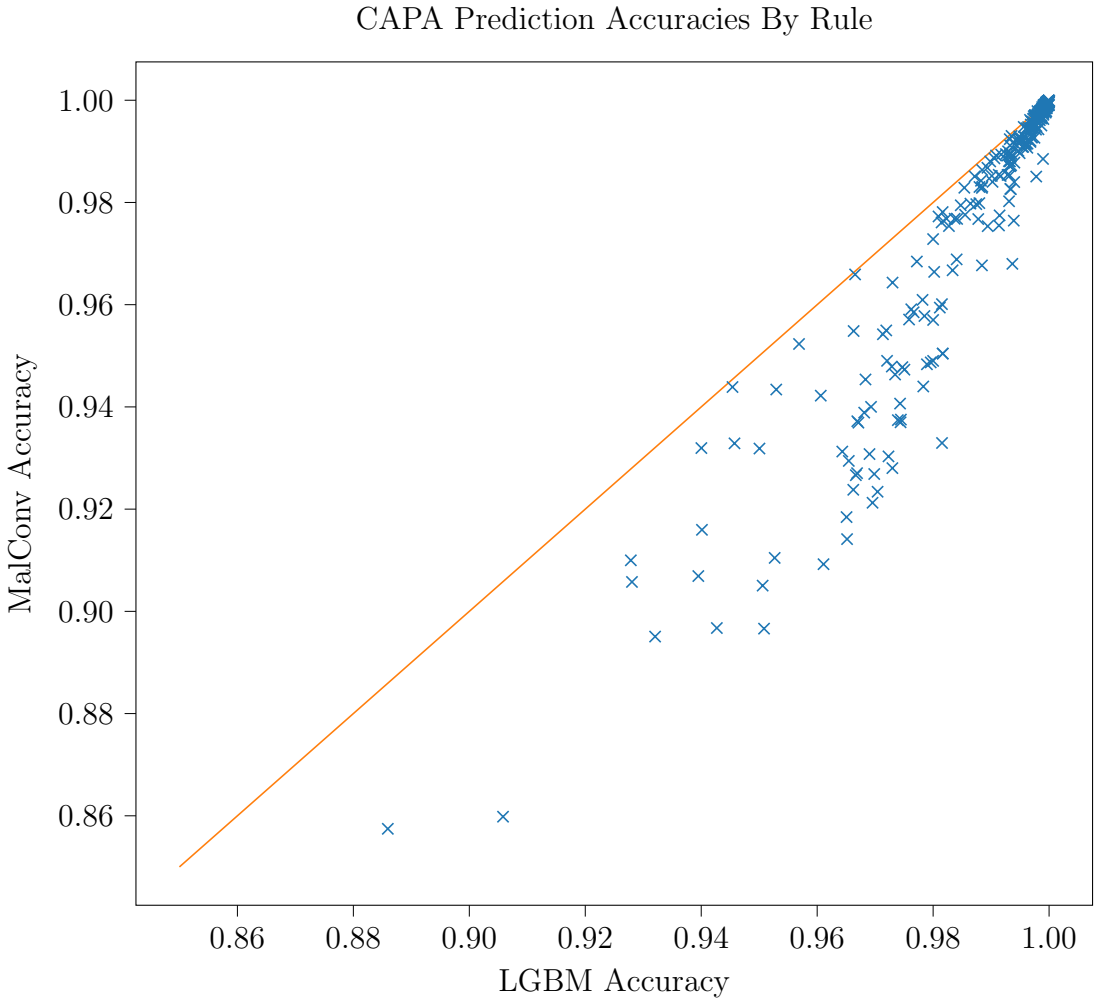



Figure 5.1: Detection accuracy for each CAPA rule for both MalConv and the LGBM model. The LGBM model is consistently more accurate than MalConv across rules at the cost of higher cost associated with feature extraction.

various uncertainty thresholds (as measured by predictive entropy) corresponding to proportions of the test data being run through CAPA. To simulate a real world deployment of such a system, we also plot the thresholds needed to achieve a 99.9% average percentage of CAPA rules correctly labeled across files as well as the actualized thresholds chosen using a validation set. The validation and test sets were temporally split to correspond to samples first seen in November 2018 and December 2018 respectively. We note that the ideal and actualized uncertainty thresholds are

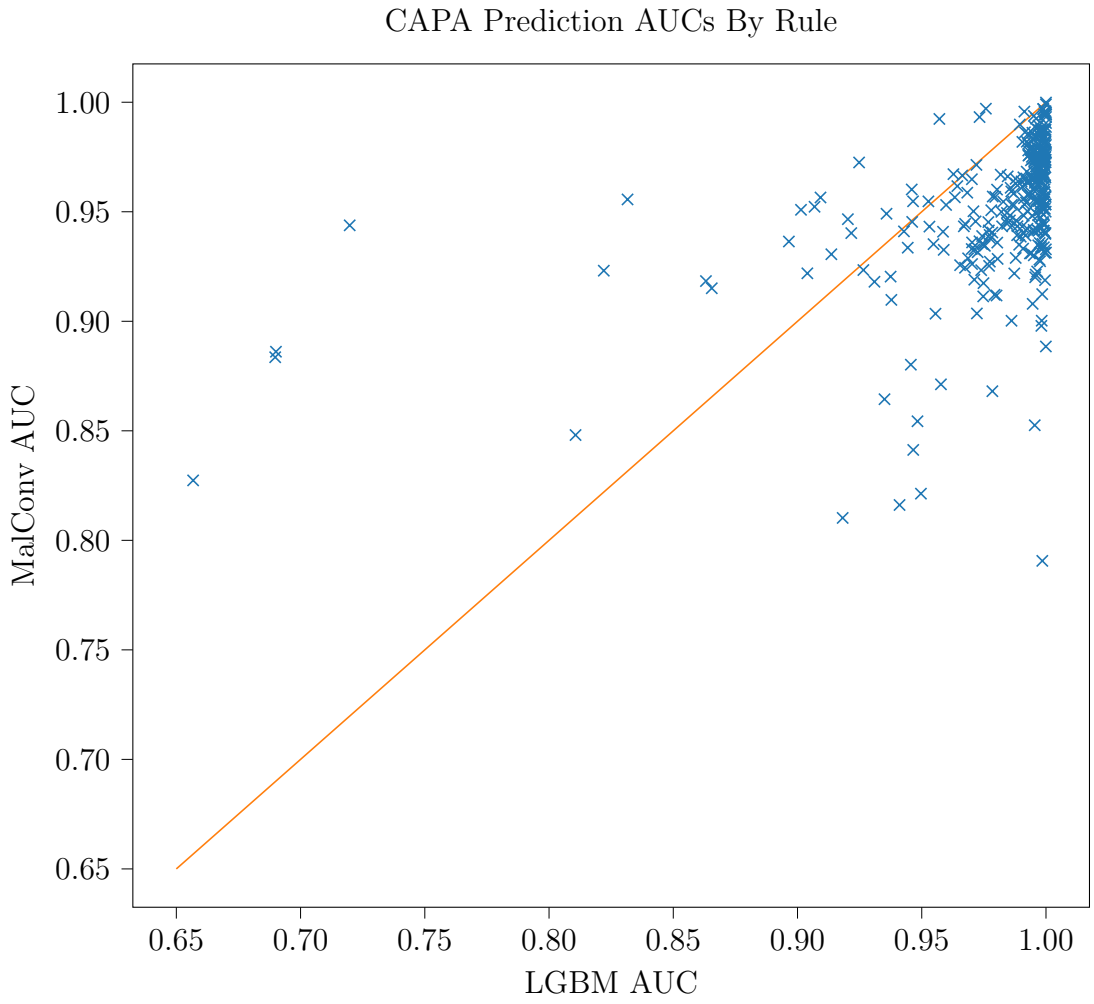



Figure 5.2: The AUC for each CAPA rule for both MalConv and the LGBM model. While LGBM generally does better than MalConv in terms of AUC, the LGBM model has worse worst cases.

close, and compute time spent on CAPA extraction can be cut by more than half. In other words, an approximate average percentage of CAPA rules correctly labeled of 99.9% can be attained while running only less than half of the files though CAPA, using a 2200 times faster MalConv prediction for the rest of the files. This leads to substantial gains in a deployed production setting where it is not feasible to run CAPA on all of the data. Furthermore, in highly compute constrained settings, a 98% average of CAPA rules correctly labeled can be achieved by using only MalConv

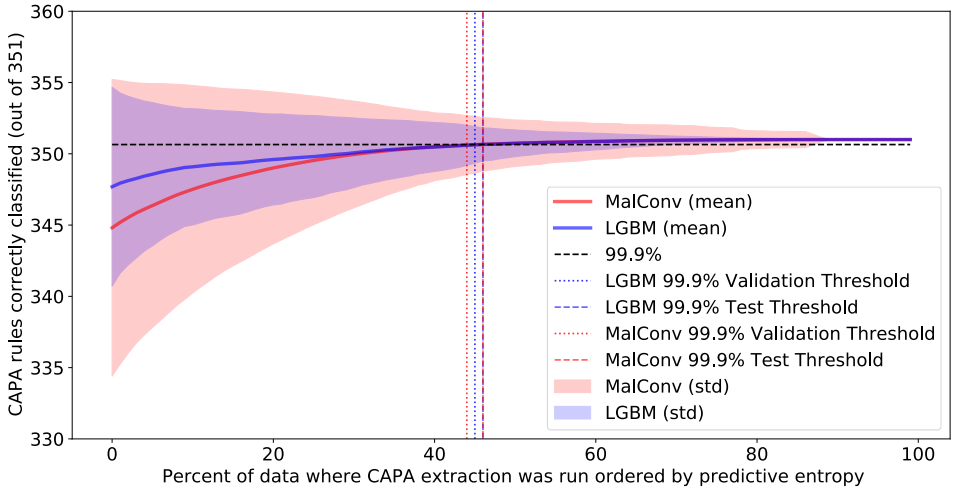



Figure 5.3: We would like to MalConv and LGBM to abstain from making a prediction on high uncertainty samples that we then would run CAPA on. This figure shows the performance of such an approach for various uncertainty thresholds (as measured by predictive entropy) corresponding to proportions of the test data being run through CAPA. To simulate a real world deployment of such a system, we also plot the thresholds needed to achieve a 99.9% average percentage of CAPA rules correctly labeled across files as well as the actualized thresholds chosen using a validation set. The x-axis corresponds to the percent of the data requiring expensive analysis. The far right corresponds to running CAPA on all of the data, and the far left corresponds to predicting on all of the data but never running CAPA. The figure shows that we can achieve a 99.9% average percentage of CAPA rules correctly labeled by running CAPA on less than half of the data. This is significant as CAPA is more than 2200 times more expensive time-wise compared to MalConv mainly due to the costs associated with disassembly.

and never actually running CAPA. This result also provides evidence that MalConv is capable of learning to pick up on features that are predictive of capabilities in executables.

## 5.5 Deferring to More Expensive Models

Dynamic malware analysis is more expensive than static malware analysis. Human malware analysis is more expensive than automated malware analysis. Ide-

ally, we would like to incur the minimal analysis cost per file needed to determine if that file is malicious or benign. In other words, we want to run cheap, automated static analysis first and only run more expensive analysis such as dynamic analysis if the static analysis model is uncertain. Additionally, we want to train the static analysis model to be adaptive to the dynamic analysis model's strengths and weaknesses. Even if uncertain, the static analysis model should make a prediction if it thinks the dynamic analysis model will perform worse on the kind of data being predicted on.

Using Bayesian MalConv as the static model, Figure 5.4 shows the performance of such an approach for various predictive entropy thresholds corresponding to various total runtimes. Following the naming conventions from Madras, Pitassi, and Zemel [121], "reject" corresponds to letting the dynamic model predict when Bayesian MalConv's uncertainty is above a certain threshold, and "defer" corresponds to letting the dynamic model predict only when Bayesian MalConv's uncertainty is above a certain threshold and Bayesian MalConv predicts that the dynamic model will make the correct prediction. To enable a Bayesian MalConv model to defer, we train it with dual equally weighted prediction tasks: the malware classification task as well as the prediction of whether or not the dynamic model will make a correct prediction on the sample.

Our approach differs from that of Madras, Pitassi, and Zemel [121] in two ways. First, while their approach can be seen as a mixture of Bernoullis where a binary deferral decision gating variable is used to both express uncertainty and to try to predict which model will have a lower loss, our approach explicitly decouples

uncertainty from the prediction of a downstream model's performance. This decoupling notably allows for an easier interpretation of the reasons for a deferral decision, and in many cases likely results in an easier learning problem as well. Second, the Bayesian uncertainty based approach described in Appendix F of Madras, Pitassi, and Zemel [121] uses only uncertainty to reject and, unlike our work, does not provide a way to incorporate the prediction of a downstream model's performance.

To simulate a real world deployment, we plot the actualized uncertainty thresholds chosen using a validation set to maximize malware detection accuracy. The validation and test sets were temporally split to correspond to samples first seen in November 2018 and December 2018 respectively. We also plot the accuracy achieved by an ensemble model that averages the predictions of the MalConv and dynamic models together. Unsurprisingly, the ensemble model outperforms both MalConv and the dynamic model used in isolation, but we note that both MalConv and dynamic analysis need to be run on every single file in order to compute the ensemble predictions.

We highlight that the rejection model with a threshold chosen on a held-out validation set achieves a test accuracy roughly equal to that of the ensemble model while requiring dynamic analysis to be run on only 13.2 percent of the test data, saving a year's worth of compute time compared to the ensemble model! The defer approach achieves an accuracy higher than that of the ensemble while requiring dynamic analysis to be run on only 17.3 percent of the test data. Even if the uncertainty threshold is set too high, the benefit of the defer approach is seen as accuracy never dips below that of the MalConv model, demonstrating that MalConv

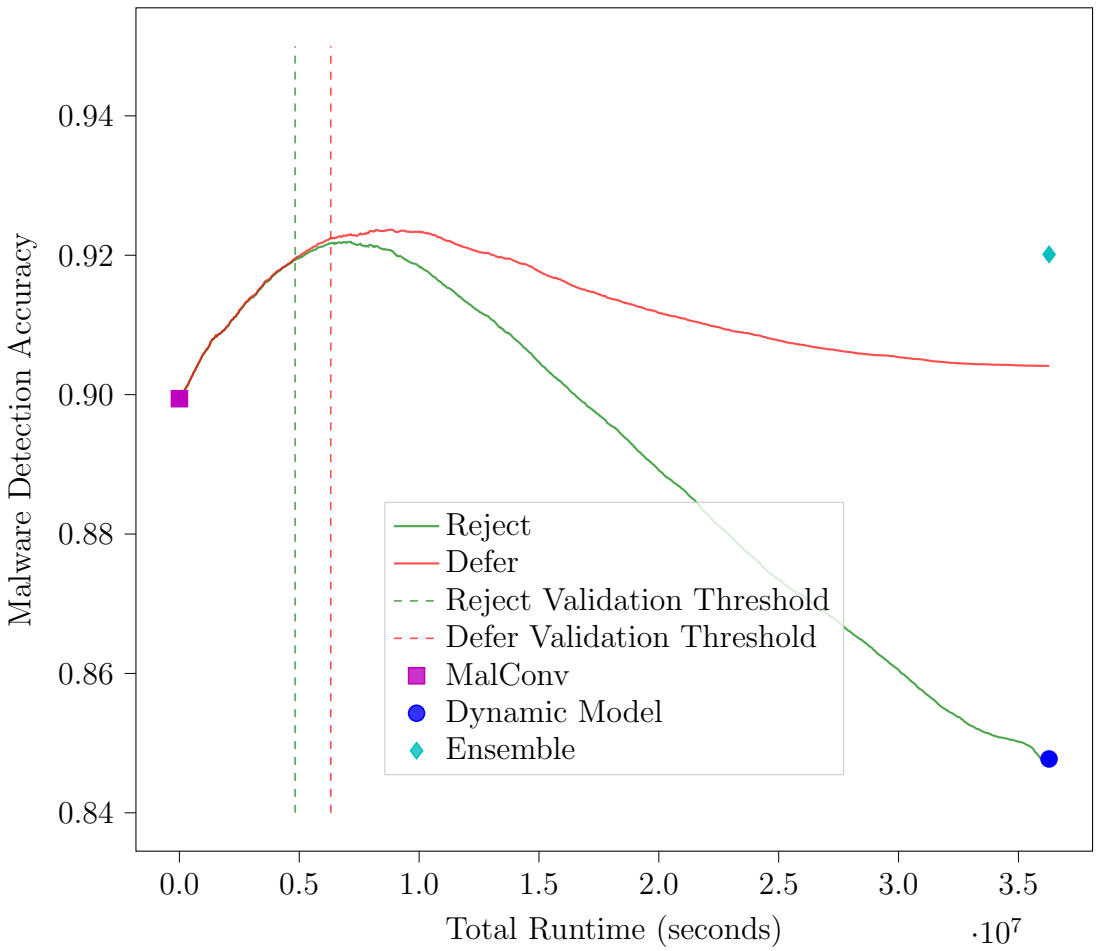



Figure 5.4: The performance of an approach that uses Bayesian MalConv as a cheap initial model that all files are run through, followed by a more expensive dynamic model for certain files, for various predictive entropy thresholds corresponding to various total runtimes. "Reject" corresponds to letting the dynamic model predict when Bayesian MalConv's uncertainty is above a certain threshold, and "Defer" corresponds to letting the dynamic model predict only when Bayesian MalConv's uncertainty is above a certain threshold and Bayesian MalConv predicts that the dynamic model will make the correct prediction. The rejection model with a threshold chosen on a held-out validation set achieves a test accuracy roughly equal to that of the ensemble model while requiring dynamic analysis to be run on only 13.2 percent of the test data, saving a year's worth of compute time compared to the ensemble model.

is able to accurately predict when the dynamic model will make a mistake. We note that this result in combination with MalConv's ability to accurately predict CAPA outputs suggests that MalConv is capable of uncovering useful features in the binary

data.

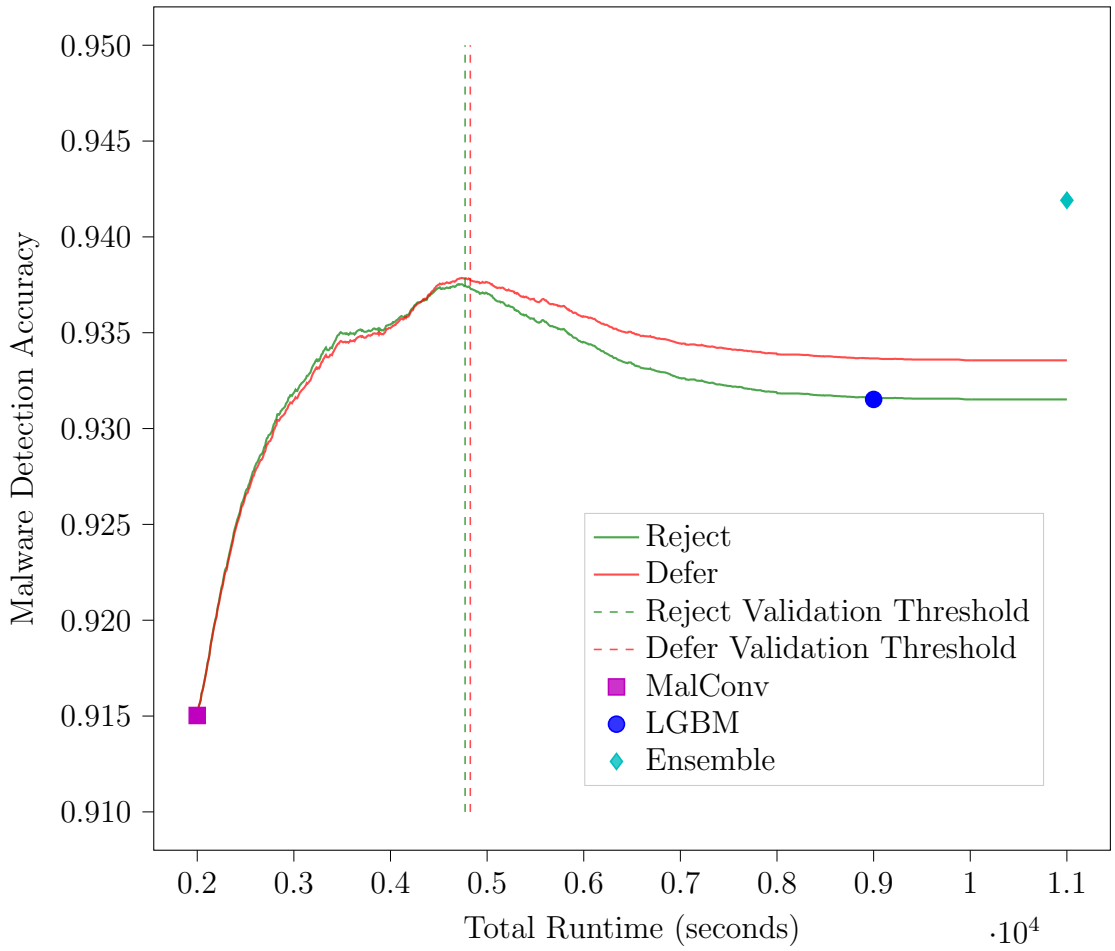



Figure 5.5: The performance of an approach that uses Bayesian MalConv as a cheap initial model that all files are run through, followed by a more expensive LGBM on EMBER features model for certain files, for various predictive entropy thresholds.

Finally, in Figure 5.5, we show a similar analysis where a LGBM model trained on EMBER features is used instead of the dynamic model as the more expensive downstream model. While in this case running the more expensive model on all of the data results in a best performing ensemble model, the reject and defer models show the desired behavior of improving accuracy in a compute budget limited situation.

## 5.6 Conclusions

We have shown that cheaper static analysis features can be used to predict the outcomes of more expensive disassembly-based analysis. We have also developed a dynamic analysis model that is complementary to static analysis-based models. In both cases, we showed how uncertainty can be used to minimize compute costs. Future work includes the investigation of chaining together a larger number of algorithms and approaches. In the next chapter, we will discuss broader avenues for future research in the context of this thesis in its entirety.

# Chapter 6

## Conclusions and Future Work

In this thesis, we have demonstrated how uncertainty estimation can benefit cyber defense. In particular, our experiments have shown how taking into account uncertainty can be especially beneficial for highly constrained and quickly evolving malware detection use cases. Leveraging uncertainty, we have improved malware detection rates under extreme false positive rate constraints, improved out of distribution data detection approaches, and greatly reduced the amount of compute time needed to take advantage of the benefits of dynamic analysis. Along the way, we have also illustrated why previous evaluation metrics can be misleading and demonstrated that executable file capabilities can be accurately predicted from raw byte sequences. Hopefully, this lays the groundwork for the increased adoption of uncertainty aware ML in the cybersecurity community.

Potential avenues for future work that follow directly from our research include exploring and explaining malware specific drivers of uncertainty, the investigation of more elaborate features based off of randomized embeddings, and the evaluation of these methods in a real world, deployed environment over a long period of time in production.

A clearly valuable direction of research we have not discussed is the use of uncertainty for active learning [116] in the context of malware detection. Much work

exists around uncertainty for active learning in the machine learning community. For example, Gal, Islam, and Ghahramani [64] and Pop and Fulop [145] demonstrate the use of ensembles of dropout neural networks for active learning. However, the application of such methods to cyber defense remains largely unexplored. This direction would be advantageous to explore as data labeling in cybersecurity is particularly difficult and expensive.

# Appendix A

# Out of Distribution Data Detection Using Dropout Bayesian Neural Networks Appendix

## A.1 Experimental Result Standard Deviations

We repeated each of our experiments multiple times and computed a mean and standard deviation for each experiment and evaluation metric. In all of our experiments, the standard deviations are much smaller than effect sizes, so we reported only the means in section 4.4. Here we report the complete results, which include standard deviations, for all of our experiments. Vision experiment results are summarized in Table A.1. Language experiment results are summarized in Table A.2. Malware experiment results are summarized in Table A.3.

Table A.1: Performance with and without the cosine randomized embedding spread features for various experimental configurations for a dropout LeNet5 trained on MNIST. Features labeled as "Last" consist of common baseline features computed using softmax output samples from the network (predictive entropy, mutual information, and maximum softmax probability). Features labeled as "Last+Spread" consist of these baseline features plus our additional randomized embedding maximum cosine spread features for each layer. Each experiment was repeated multiple times, and the mean and standard deviation are reported here. Best results are shown in **bold**.

| | | | Num/Class | n=1000 | | | | n=100 | | | | n=10 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Metric | AUC | | Acc | | AUC | | Acc | | AUC | | Acc | |
| | | | Statistic | avg | std | avg | std | avg | std | avg | std | avg | std | avg | std |
| OOD Train | OOD Test | OOD Model | Features | | | | | | | | | | | | |
| Fashion | Kuzushiji | LR | Last | 0.969 | 0.000 | **0.914** | **0.001** | 0.967 | 0.004 | 0.909 | 0.009 | 0.963 | 0.024 | 0.884 | 0.023 |
| | | | Last+Spread | **0.979** | **0.001** | 0.914 | 0.003 | **0.973** | **0.008** | **0.911** | **0.013** | **0.967** | **0.035** | **0.901** | **0.033** |
| | | RF | Last | 0.960 | 0.001 | 0.917 | 0.002 | 0.952 | 0.005 | 0.905 | 0.009 | 0.942 | 0.015 | 0.884 | 0.038 |
| | | | Last+Spread | **0.979** | **0.001** | **0.922** | **0.004** | **0.974** | **0.003** | **0.921** | **0.005** | **0.969** | **0.008** | **0.907** | **0.018** |
| | notMNIST | LR | Last | 0.966 | 0.001 | 0.912 | 0.002 | 0.965 | 0.003 | 0.909 | 0.011 | 0.960 | 0.004 | 0.879 | 0.024 |
| | | | Last+Spread | **0.983** | **0.001** | **0.932** | **0.003** | **0.979** | **0.005** | **0.925** | **0.015** | **0.967** | **0.011** | **0.892** | **0.021** |
| | | RF | Last | 0.959 | 0.002 | 0.920 | 0.004 | 0.950 | 0.005 | 0.903 | 0.010 | 0.938 | 0.014 | 0.880 | 0.038 |
| | | | Last+Spread | **0.985** | **0.001** | **0.940** | **0.004** | **0.976** | **0.004** | **0.924** | **0.006** | **0.963** | **0.009** | **0.901** | **0.020** |
| Kuzushiji | Fashion | LR | Last | 0.973 | 0.000 | 0.920 | 0.001 | 0.972 | 0.001 | 0.917 | 0.006 | 0.967 | 0.011 | 0.899 | 0.016 |
| | | | Last+Spread | **0.989** | **0.001** | **0.948** | **0.002** | **0.983** | **0.004** | **0.937** | **0.008** | **0.978** | **0.004** | **0.922** | **0.014** |
| | | RF | Last | 0.964 | 0.001 | 0.920 | 0.002 | 0.956 | 0.005 | 0.907 | 0.009 | 0.946 | 0.015 | 0.896 | 0.025 |
| | | | Last+Spread | **0.986** | **0.001** | **0.943** | **0.002** | **0.978** | **0.004** | **0.931** | **0.006** | **0.967** | **0.009** | **0.914** | **0.016** |
| | notMNIST | LR | Last | 0.967 | 0.000 | 0.914 | 0.001 | 0.965 | 0.002 | 0.910 | 0.010 | 0.960 | 0.005 | 0.886 | 0.020 |
| | | | Last+Spread | **0.984** | **0.001** | **0.931** | **0.004** | **0.975** | **0.008** | **0.914** | **0.020** | **0.966** | **0.007** | **0.888** | **0.021** |
| | | RF | Last | 0.960 | 0.001 | 0.922 | 0.003 | 0.950 | 0.005 | 0.904 | 0.010 | 0.938 | 0.017 | 0.888 | 0.026 |
| | | | Last+Spread | **0.982** | **0.001** | **0.935** | **0.003** | **0.971** | **0.006** | **0.921** | **0.006** | **0.954** | **0.010** | **0.896** | **0.019** |
| notMNIST | Fashion | LR | Last | 0.966 | 0.003 | 0.911 | 0.003 | 0.957 | 0.018 | 0.906 | 0.012 | 0.959 | 0.037 | 0.893 | 0.023 |
| | | | Last+Spread | **0.978** | **0.003** | **0.937** | **0.005** | **0.969** | **0.016** | **0.928** | **0.015** | **0.977** | **0.014** | **0.925** | **0.018** |
| | | RF | Last | 0.960 | 0.002 | 0.910 | 0.004 | 0.955 | 0.005 | 0.904 | 0.011 | 0.946 | 0.018 | 0.887 | 0.032 |
| | | | Last+Spread | **0.988** | **0.001** | **0.943** | **0.005** | **0.982** | **0.004** | **0.935** | **0.007** | **0.978** | **0.006** | **0.920** | **0.017** |
| | Kuzushiji | LR | Last | 0.960 | 0.006 | **0.900** | **0.007** | 0.946 | 0.030 | **0.893** | **0.021** | 0.951 | 0.057 | 0.882 | 0.035 |
| | | | Last+Spread | **0.966** | **0.006** | 0.893 | 0.010 | **0.949** | **0.028** | 0.886 | 0.031 | **0.967** | **0.030** | **0.902** | **0.035** |
| | | RF | Last | 0.956 | 0.002 | **0.906** | **0.005** | 0.950 | 0.006 | 0.901 | 0.012 | 0.941 | 0.020 | 0.883 | 0.033 |
| | | | Last+Spread | **0.978** | **0.001** | 0.906 | 0.008 | **0.973** | **0.004** | **0.915** | **0.012** | **0.969** | **0.008** | **0.906** | **0.023** |

Table A.2: Performance with and without the cosine randomized embedding spread features for a Char-CNN with dropout added before every layer trained to classify languages using the WiLI dataset. Best results are shown in **bold**.

| | | | Num/Class | n=100 | | | | n=50 | | | | n=25 | | | | n=10 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Metric | AUC | | Acc | | AUC | | Acc | | AUC | | Acc | | AUC | | Acc | |
| OOD Train | OOD Test | OOD Model | Statistic Features | avg | std | avg | std | avg | std | avg | std | avg | std | avg | std | avg | std | avg | std |
| Basque | rest | LR | Last | 0.888 | 0.005 | 0.798 | 0.008 | 0.883 | 0.014 | 0.794 | 0.010 | 0.882 | 0.015 | 0.792 | 0.011 | 0.878 | 0.029 | 0.786 | 0.019 |
| | | | Last+Spread | **0.926** | **0.019** | **0.843** | **0.019** | **0.919** | **0.033** | **0.836** | **0.023** | **0.921** | **0.034** | **0.835** | **0.025** | **0.926** | **0.026** | **0.828** | **0.023** |
| | | RF | Last | 0.862 | 0.008 | 0.797 | 0.007 | 0.857 | 0.013 | 0.793 | 0.011 | 0.851 | 0.015 | 0.792 | 0.013 | 0.842 | 0.015 | 0.789 | 0.021 |
| | | | Last+Spread | **0.924** | **0.008** | **0.845** | **0.011** | **0.920** | **0.011** | **0.840** | **0.013** | **0.918** | **0.012** | **0.835** | **0.017** | **0.914** | **0.013** | **0.824** | **0.018** |
| Finnish | rest | LR | Last | 0.888 | 0.003 | 0.795 | 0.006 | 0.885 | 0.006 | 0.792 | 0.008 | 0.881 | 0.013 | 0.790 | 0.011 | 0.883 | 0.008 | 0.786 | 0.015 |
| | | | Last+Spread | **0.910** | **0.019** | **0.818** | **0.017** | **0.908** | **0.032** | **0.818** | **0.022** | **0.909** | **0.035** | **0.821** | **0.024** | **0.910** | **0.041** | **0.818** | **0.028** |
| | | RF | Last | 0.864 | 0.006 | 0.794 | 0.007 | 0.858 | 0.009 | 0.792 | 0.011 | 0.850 | 0.014 | 0.789 | 0.015 | 0.840 | 0.020 | 0.783 | 0.024 |
| | | | Last+Spread | **0.913** | **0.011** | **0.821** | **0.012** | **0.907** | **0.013** | **0.821** | **0.012** | **0.905** | **0.014** | **0.818** | **0.014** | **0.904** | **0.016** | **0.814** | **0.015** |
| Luganda | rest | LR | Last | 0.891 | 0.002 | 0.806 | 0.005 | 0.889 | 0.004 | 0.803 | 0.008 | 0.887 | 0.006 | 0.800 | 0.009 | 0.881 | 0.026 | 0.794 | 0.015 |
| | | | Last+Spread | **0.943** | **0.006** | **0.864** | **0.016** | **0.939** | **0.008** | **0.854** | **0.017** | **0.935** | **0.009** | **0.847** | **0.014** | **0.931** | **0.009** | **0.837** | **0.017** |
| | | RF | Last | 0.866 | 0.006 | 0.800 | 0.007 | 0.859 | 0.010 | 0.797 | 0.011 | 0.854 | 0.014 | 0.796 | 0.013 | 0.843 | 0.027 | 0.785 | 0.038 |
| | | | Last+Spread | **0.936** | **0.005** | **0.862** | **0.009** | **0.930** | **0.008** | **0.852** | **0.014** | **0.926** | **0.009** | **0.843** | **0.015** | **0.921** | **0.011** | **0.831** | **0.018** |
| Polish | rest | LR | Last | 0.900 | 0.003 | 0.824 | 0.003 | 0.897 | 0.010 | 0.821 | 0.007 | 0.891 | 0.019 | 0.816 | 0.012 | 0.887 | 0.042 | 0.812 | 0.021 |
| | | | Last+Spread | **0.939** | **0.010** | **0.866** | **0.014** | **0.938** | **0.014** | **0.864** | **0.017** | **0.935** | **0.021** | **0.860** | **0.023** | **0.934** | **0.020** | **0.852** | **0.022** |
| | | RF | Last | 0.870 | 0.010 | 0.793 | 0.011 | 0.860 | 0.017 | 0.787 | 0.015 | 0.854 | 0.021 | 0.783 | 0.023 | 0.850 | 0.029 | 0.780 | 0.034 |
| | | | Last+Spread | **0.937** | **0.005** | **0.871** | **0.008** | **0.932** | **0.008** | **0.863** | **0.011** | **0.928** | **0.009** | **0.855** | **0.012** | **0.922** | **0.018** | **0.841** | **0.024** |
| Tongan | rest | LR | Last | 0.857 | 0.115 | **0.815** | **0.060** | 0.841 | 0.159 | **0.811** | **0.076** | 0.815 | 0.198 | 0.800 | 0.088 | 0.791 | 0.244 | 0.771 | 0.155 |
| | | | Last+Spread | **0.886** | **0.060** | 0.811 | 0.056 | **0.877** | **0.091** | 0.810 | 0.069 | **0.884** | **0.101** | **0.819** | **0.074** | **0.880** | **0.125** | **0.813** | **0.085** |
| | | RF | Last | 0.765 | 0.063 | 0.699 | 0.055 | 0.766 | 0.074 | 0.695 | 0.067 | 0.769 | 0.092 | 0.684 | 0.075 | 0.785 | 0.129 | 0.701 | 0.101 |
| | | | Last+Spread | **0.915** | **0.016** | **0.847** | **0.029** | **0.913** | **0.019** | **0.845** | **0.028** | **0.906** | **0.030** | **0.836** | **0.035** | **0.903** | **0.051** | **0.823** | **0.054** |
| Xhosa | rest | LR | Last | 0.894 | 0.004 | 0.807 | 0.008 | 0.891 | 0.007 | 0.804 | 0.008 | 0.886 | 0.019 | 0.800 | 0.013 | 0.879 | 0.046 | 0.794 | 0.022 |
| | | | Last+Spread | **0.944** | **0.009** | **0.866** | **0.014** | **0.940** | **0.014** | **0.857** | **0.019** | **0.933** | **0.020** | **0.846** | **0.024** | **0.931** | **0.028** | **0.838** | **0.029** |
| | | RF | Last | 0.864 | 0.009 | 0.787 | 0.013 | 0.857 | 0.016 | 0.782 | 0.020 | 0.849 | 0.021 | 0.778 | 0.027 | 0.846 | 0.028 | 0.774 | 0.032 |
| | | | Last+Spread | **0.939** | **0.006** | **0.868** | **0.011** | **0.934** | **0.009** | **0.860** | **0.013** | **0.928** | **0.012** | **0.852** | **0.017** | **0.921** | **0.024** | **0.835** | **0.021** |

Table A.3: Performance with and without the cosine randomized embedding spread features for a MalConv model with dropout added before each fully connected layer trained to detect malware using the EMBER2018 dataset. Best results are shown in **bold**.

| | | Num/Class | n=100 | | | | n=50 | | | | n=25 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Metric | AUC | | Recall | | AUC | | Recall | | AUC | | Recall | |
| Experiment | OOD Model | Statistic Features | avg | std | avg | std | avg | std | avg | std | avg | std | avg | std |
| EMBER2018 | LR | Last | 0.789 | 0.007 | 0.704 | 0.043 | **0.786** | **0.008** | 0.682 | 0.054 | **0.778** | **0.018** | 0.650 | 0.066 |
| | | Last+Spread | **0.793** | **0.008** | **0.718** | **0.042** | 0.783 | 0.013 | **0.689** | **0.067** | 0.766 | 0.027 | **0.658** | **0.080** |
| | RF | Last | 0.757 | 0.011 | 0.735 | 0.046 | 0.752 | 0.015 | 0.727 | 0.060 | 0.748 | 0.023 | 0.714 | 0.079 |
| | | Last+Spread | **0.791** | **0.011** | **0.784** | **0.045** | **0.782** | **0.014** | **0.764** | **0.057** | **0.770** | **0.018** | **0.743** | **0.084** |
| Brazilian | LR | Last | 0.685 | 0.007 | **0.645** | **0.054** | 0.680 | 0.010 | 0.607 | 0.072 | 0.668 | 0.042 | 0.584 | 0.078 |
| | | Last+Spread | **0.741** | **0.023** | 0.620 | 0.039 | **0.734** | **0.023** | **0.617** | **0.049** | **0.712** | **0.034** | **0.605** | **0.063** |
| | RF | Last | 0.724 | 0.016 | 0.693 | 0.038 | 0.705 | 0.023 | 0.674 | 0.055 | 0.679 | 0.035 | 0.652 | 0.081 |
| | | Last+Spread | **0.839** | **0.010** | **0.797** | **0.034** | **0.813** | **0.016** | **0.772** | **0.054** | **0.776** | **0.024** | **0.736** | **0.083** |

## A.2 Additional Spectral Normalization Results

We repeated the vision OOD data detection experiments from subsubsection 4.4.1.1 on a spectral normalized dropout LeNet5 trained for 100 epochs. While spectral normalization is not required to see an improvement from the inclusion of cosine spread features, spectral normalization improves OOD detection performance consistently, as shown in Table A.4 when compared to Table A.1.

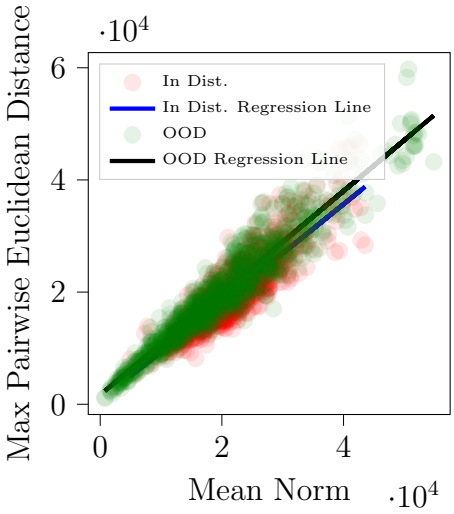
Table A.4: Performance with and without the cosine randomized embedding spread features for a dropout, spectral normalized LeNet5 trained on MNIST. Best results are shown in **bold**.

| | | | Num/Class | n=1000 | | | | n=100 | | | | n=10 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Metric | AUC | | Acc | | AUC | | Acc | | AUC | | Acc | |
| | | | Statistic | avg | std | avg | std | avg | std | avg | std | avg | std | avg | std |
| OOD Train | OOD Test | OOD Model | Features | | | | | | | | | | | | |
| Fashion | Kuzushiji | LR | Last | 0.982 | 0.003 | 0.921 | 0.005 | 0.978 | 0.016 | 0.921 | 0.016 | 0.980 | 0.040 | 0.908 | 0.032 |
| | | | Last+Spread | **0.984** | **0.002** | **0.926** | **0.006** | **0.980** | **0.015** | **0.926** | **0.019** | **0.984** | **0.008** | **0.915** | **0.024** |
| | | RF | Last | 0.979 | 0.001 | 0.936 | 0.004 | 0.972 | 0.004 | 0.931 | 0.010 | 0.963 | 0.014 | 0.919 | 0.024 |
| | | | Last+Spread | **0.985** | **0.001** | **0.939** | **0.004** | **0.982** | **0.002** | **0.940** | **0.007** | **0.981** | **0.008** | **0.932** | **0.017** |
| | notMNIST | LR | Last | **0.982** | **0.003** | 0.916 | 0.008 | 0.975 | 0.025 | 0.915 | 0.024 | 0.979 | 0.049 | 0.905 | 0.036 |
| | | | Last+Spread | 0.982 | 0.002 | **0.922** | **0.009** | **0.975** | **0.025** | **0.922** | **0.026** | **0.983** | **0.012** | **0.913** | **0.025** |
| | | RF | Last | 0.978 | 0.001 | 0.933 | 0.005 | 0.971 | 0.005 | 0.929 | 0.012 | 0.962 | 0.014 | 0.917 | 0.026 |
| | | | Last+Spread | **0.984** | **0.001** | **0.942** | **0.004** | **0.981** | **0.003** | **0.940** | **0.008** | **0.980** | **0.007** | **0.932** | **0.018** |
| Kuzushiji | Fashion | LR | Last | **0.988** | **0.000** | 0.948 | 0.002 | **0.987** | **0.003** | 0.944 | 0.008 | 0.979 | 0.045 | 0.918 | 0.033 |
| | | | Last+Spread | 0.987 | 0.001 | **0.949** | **0.002** | 0.985 | 0.003 | **0.944** | **0.007** | **0.981** | **0.041** | **0.932** | **0.037** |
| | | RF | Last | 0.980 | 0.001 | 0.946 | 0.002 | 0.972 | 0.004 | 0.938 | 0.007 | 0.966 | 0.009 | 0.930 | 0.021 |
| | | | Last+Spread | **0.985** | **0.001** | **0.950** | **0.002** | **0.982** | **0.004** | **0.947** | **0.003** | **0.983** | **0.002** | **0.943** | **0.007** |
| | notMNIST | LR | Last | **0.986** | **0.000** | 0.937 | 0.001 | **0.985** | **0.002** | 0.934 | 0.007 | 0.979 | 0.029 | 0.910 | 0.025 |
| | | | Last+Spread | 0.986 | 0.001 | **0.946** | **0.001** | 0.984 | 0.003 | **0.939** | **0.006** | **0.983** | **0.017** | **0.922** | **0.021** |
| | | RF | Last | 0.979 | 0.001 | 0.941 | 0.002 | 0.972 | 0.004 | 0.933 | 0.008 | 0.964 | 0.011 | 0.924 | 0.023 |
| | | | Last+Spread | **0.985** | **0.001** | **0.947** | **0.001** | **0.982** | **0.002** | **0.944** | **0.003** | **0.982** | **0.002** | **0.937** | **0.010** |
| notMNIST | Fashion | LR | Last | 0.988 | 0.001 | 0.946 | 0.002 | **0.986** | **0.003** | 0.941 | 0.009 | 0.983 | 0.018 | 0.916 | 0.027 |
| | | | Last+Spread | **0.988** | **0.001** | **0.951** | **0.002** | 0.985 | 0.006 | **0.944** | **0.010** | **0.986** | **0.002** | **0.932** | **0.016** |
| | | RF | Last | 0.980 | 0.001 | 0.945 | 0.002 | 0.971 | 0.005 | 0.938 | 0.007 | 0.966 | 0.010 | 0.931 | 0.017 |
| | | | Last+Spread | **0.987** | **0.001** | **0.952** | **0.001** | **0.984** | **0.002** | **0.947** | **0.004** | **0.983** | **0.004** | **0.941** | **0.010** |
| | Kuzushiji | LR | Last | 0.986 | 0.000 | 0.936 | 0.002 | 0.984 | 0.003 | 0.934 | 0.007 | 0.984 | 0.002 | 0.911 | 0.023 |
| | | | Last+Spread | **0.989** | **0.000** | **0.947** | **0.001** | **0.987** | **0.002** | **0.942** | **0.006** | **0.987** | **0.003** | **0.923** | **0.018** |
| | | RF | Last | 0.980 | 0.001 | 0.942 | 0.002 | 0.972 | 0.004 | 0.934 | 0.007 | 0.963 | 0.011 | 0.926 | 0.018 |
| | | | Last+Spread | **0.987** | **0.000** | **0.947** | **0.001** | **0.984** | **0.002** | **0.944** | **0.004** | **0.983** | **0.005** | **0.936** | **0.012** |

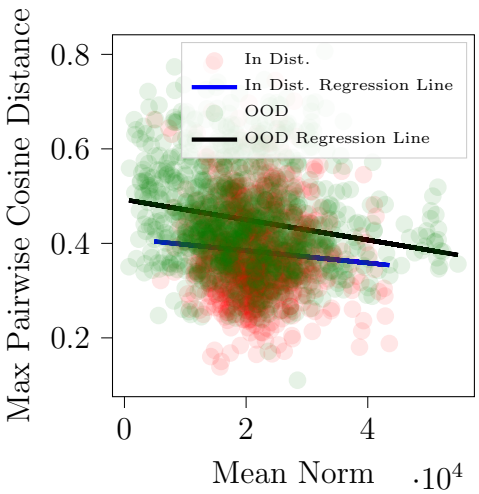
## A.3 Cosine Distance vs. Euclidean Distance for Unsupervised Embeddings

We also investigated cosine distance versus Euclidean distance for measuring randomized embedding dispersion in the unsupervised setting. In particular, we investigated a stacked denoising autoencoder variant [180, 181] where all layers are trained at the same time instead of stage-wise, and dropout with a dropout probability $p = 0.1$ is used as the corrupting process at each layer of the encoder. At test time, the dropout corruption is left on to generate randomized embeddings. The denoising autoencoder was trained on MNIST for 20 epochs with a batch size of 64 using the Adam optimizer with a learning rate of 0.001, the default recommended settings, and a weight decay of 0.01. Image inputs were flattened, and the encoder architecture consisted of 6 ReLU activated linear layers of output dimensions: 784, 400, 400, 120, 120, and 84. The decoder architecture is similar to the encoder architecture but in reverse order.

Figure A.1a and Figure A.1b show consistent results. Embedding dispersion as measured by Euclidean distance is related to mean norm in an identical manner across in distribution and OOD data. While not as well separated as in the supervised setting, in distribution data has lower embedding dispersion as measured by cosine distance when compared to OOD data.




(a) Denoising autoencoder, Euclidean distance.




(b) Denoising autoencoder, cosine distance.

Figure A.1: A comparison of the relationships between denoising autoencoder randomized embedding mean norm and the maximum pairwise distance for Euclidean distance and cosine distance respectively, for in distribution data (MNIST) and OOD data (Not-MNIST). Regression line fits are provided for each as well for easier comparison.

### A.3.1 Simulations

### A.3.2 Mean and Variance of the Embedding Norms

We perform a simulation to further illustrate the problem with the use of Euclidean distance in the case of a two layer ReLU activated network. As the depth of the BNN increases, the mean and variance of the embedding norms dramatically increase across layers, in particular as a consequence of the ReLU activation. This is known and bounds for this can be derived mathematically using the identity $max(x, 0) = 0.5(x + |x|)$ in the normal random matrix situation. However, we identify that the variance of the norms experiences a further increase due to the effect of dropout on preceding layers causing a carryover of variance into subsequent

layers. Because dropout samples are taken across all layers simultaneously, the signal representing the distance between two embedding samples in layer $N$ is diluted with the inflated norm caused by preceding dropout in layers 1 to $N-1$. This is confirmed by simulation on a two-layer neural network with dropout in Table A.5, where the variance of the final embedding norms (4526.2) is much higher than it would be if dropout were only applied on that embedding layer (3124.0). This can explain why the Euclidean distance measure fails to perform for OOD detection.

Table A.5: Mean and (variance) of the embedding norms in a simulated context.

| | Dropout only layer 1 | Dropout only layer 2 | Dropout both layers |
|---|---|---|---|
| Layer 1 embedding norm | 96.0 (58.6) | 118.6 (0.0) | 96.0 (58.6) |
| Layer 2 embedding norm | 599.7 (3328.0) | 606.1 (3124.0) | 501.0 (4526.2) |

### A.3.3 Correlation Analysis Between Measures of Uncertainty

To examine the relationships between the uncertainty features, we ran correlation analysis between all measures on the final embedding layer of a neural network, averaged over 1000 random matrix iterations. The embeddings form a $D \times B$ matrix, where $D$ is the embedding dimension and $B$ are the number of dropout samples, and we enforce a decaying correlation structure over the embedding dimensions. In Table A.6, we summarize the correlations between all predictive features.

This result indicates that the previously used features have higher inter-correlation than the max cosine pairwise distance, suggesting that our new feature adds an orthogonal measure of information that is not previously captured. This helps explain our improvement in OOD detection.

Table A.6: Correlation analysis between measures of uncertainty in a simulated setting.

| | mutual info. | pred entr. | max softmax | max cos pdist | max euclid pdist | mean embed. norm |
|---|---|---|---|---|---|---|
| mutual info. | 1.00 | -0.31 | 0.23 | 0.08 | 0.32 | 0.51 |
| pred entr. | -0.31 | 1.00 | -0.64 | 0.01 | -0.09 | -0.26 |
| max softmax | 0.23 | -0.64 | 1.00 | 0.01 | 0.06 | 0.13 |
| max cos pdist | 0.08 | 0.01 | 0.01 | 1.00 | 0.15 | -0.14 |
| max euclid pdist | 0.32 | -0.09 | 0.06 | 0.15 | 1.00 | 0.32 |
| mean embed. norm | 0.51 | -0.26 | 0.13 | -0.14 | 0.32 | 1.00 |

## A.4 Additional Experiments on MNIST Variants

### A.4.1 Is Some OOD Training Data Needed?

To compare with methods that do not require any OOD training data at all, we attempted the following where a linear kernel one class SVM and an Isolation Forest are used as outlier detectors that would hopefully capture OOD data. Results are shown in Table A.7. Generally, the best AUC is achieved using an Isolation Forest but the accuracy remains low. This is consistent with our conclusions that the relationship contains non-linear information and that some form of OOD data is needed to choose the appropriate threshold, and that as few as $n = 10$ OOD points can estimate that threshold with significantly greater accuracy and AUC.

### A.4.2 Results when using Euclidean Randomized Embedding Maximum Spread Features

To compare Euclidean distance features with cosine distance features, we ran experiments and found that cosine does empirically does better, as expected. In Table A.8 are the results for the MNIST experiments where the Spread features use

Table A.7: To compare with methods that do not require any OOD training data at all, we attempted the following where a linear kernel one class SVM and an Isolation Forest (IF) are used as outlier detectors.

| OOD Test | OOD Model | Metric Features | AUC | Acc |
|---|---|---|---|---|
| Kuzushiji | SVM | Last | 0.555574 | 0.539760 |
| | | Last+Spread | 0.190757 | 0.253412 |
| | IF | Last | 0.876447 | 0.804457 |
| | | Last+Spread | 0.858214 | 0.617729 |
| notMNIST | SVM | Last | 0.532724 | 0.523312 |
| | | Last+Spread | 0.307177 | 0.335988 |
| | IF | Last | 0.842468 | 0.766183 |
| | | Last+Spread | 0.869251 | 0.631671 |
| Fashion | SVM | Last | 0.526127 | 0.514582 |
| | | Last+Spread | 0.212349 | 0.262806 |
| | IF | Last | 0.860657 | 0.791121 |
| | | Last+Spread | 0.883436 | 0.649500 |

Euclidean distance.

### A.4.3 Classifier Feature Importances

To further understand the contribution of our cosine distance measure, we compute the mean and standard deviation of feature Gini importances for the random forest classifiers fit across our MNIST variant experiments. Results are shown in Figure A.2 and show that our spread based features are important with layer 3's spread having a Gini importance comparable to traditional features such as predictive entropy.

Table A.8: MNIST variant results when using Euclidean randomized embedding maximum spread features.

| OOD Train | OOD Test | OOD Model | Num/Class | n=1000 | | n=100 | | n=10 | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Metric | AUC | Acc | AUC | Acc | AUC | Acc |
| | | | Features | | | | | | |
| Fashion | Kuzushiji | LR | Last | 0.970909 | 0.915305 | 0.968269 | 0.910754 | 0.967937 | 0.891701 |
| | | | Last+Spread | 0.959380 | 0.906095 | 0.953365 | 0.897221 | 0.941887 | 0.864347 |
| | | RF | Last | 0.961252 | 0.916221 | 0.949761 | 0.899854 | 0.946880 | 0.880030 |
| | | | Last+Spread | 0.958593 | 0.896063 | 0.945695 | 0.892397 | 0.943430 | 0.879735 |
| | notMNIST | LR | Last | 0.966394 | 0.910832 | 0.965956 | 0.910824 | 0.961222 | 0.882841 |
| | | | Last+Spread | 0.966064 | 0.914921 | 0.955173 | 0.901126 | 0.922676 | 0.839990 |
| | | RF | Last | 0.958357 | 0.916837 | 0.947242 | 0.898442 | 0.939856 | 0.876058 |
| | | | Last+Spread | 0.966978 | 0.927221 | 0.953618 | 0.910864 | 0.945218 | 0.882916 |
| Kuzushiji | Fashion | LR | Last | 0.972502 | 0.919816 | 0.971007 | 0.917156 | 0.968018 | 0.900125 |
| | | | Last+Spread | 0.968836 | 0.923116 | 0.962746 | 0.916281 | 0.961164 | 0.898184 |
| | | RF | Last | 0.963407 | 0.920547 | 0.953377 | 0.903246 | 0.938179 | 0.885983 |
| | | | Last+Spread | 0.964453 | 0.919921 | 0.955725 | 0.908291 | 0.941819 | 0.894097 |
| | notMNIST | LR | Last | 0.967112 | 0.914289 | 0.965904 | 0.911111 | 0.960355 | 0.885173 |
| | | | Last+Spread | 0.975508 | 0.925026 | 0.966202 | 0.917970 | 0.938949 | 0.869555 |
| | | RF | Last | 0.959856 | 0.920416 | 0.948595 | 0.903442 | 0.928044 | 0.876868 |
| | | | Last+Spread | 0.968210 | 0.925442 | 0.958487 | 0.912819 | 0.934295 | 0.887989 |
| notMNIST | Fashion | LR | Last | 0.964710 | 0.911000 | 0.947280 | 0.902452 | 0.968372 | 0.894182 |
| | | | Last+Spread | 0.963373 | 0.918600 | 0.949651 | 0.909618 | 0.962364 | 0.887704 |
| | | RF | Last | 0.960297 | 0.909847 | 0.955611 | 0.903392 | 0.946035 | 0.888309 |
| | | | Last+Spread | 0.968320 | 0.920963 | 0.963052 | 0.906211 | 0.954943 | 0.906248 |
| | Kuzushiji | LR | Last | 0.960289 | 0.899979 | 0.938433 | 0.888839 | 0.967569 | 0.887464 |
| | | | Last+Spread | 0.956063 | 0.901247 | 0.938300 | 0.890312 | 0.950506 | 0.865883 |
| | | RF | Last | 0.958770 | 0.905458 | 0.953364 | 0.902915 | 0.940991 | 0.884212 |
| | | | Last+Spread | 0.959689 | 0.892047 | 0.957301 | 0.895558 | 0.946487 | 0.895423 |

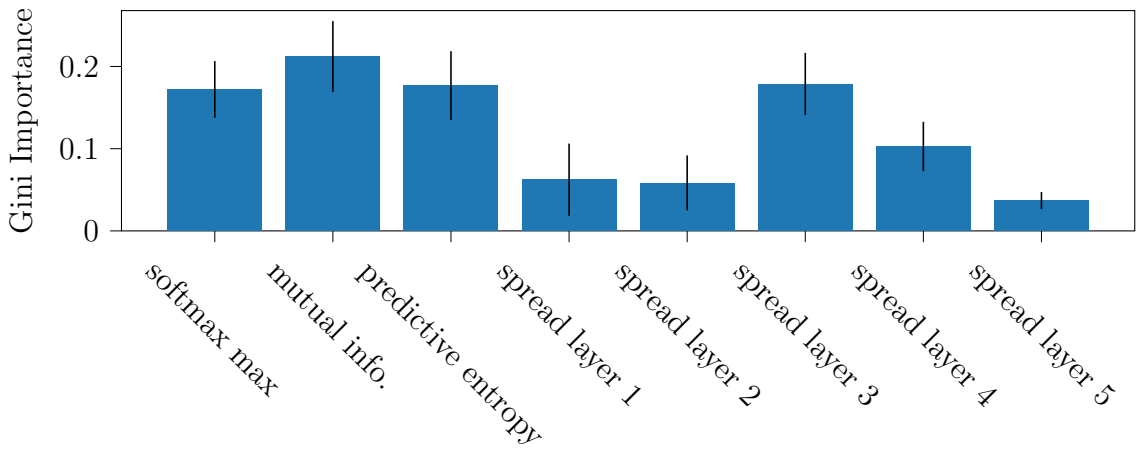



Figure A.2: Random Forest Gini feature importances for MNIST variant experiments. Means and standard deviations are shown.

## A.5 Embedding Component Variance

In the context of a linear layer with input $\mathbf{x}$ indexed by $i$, output $\mathbf{y}$ indexed by $j$, weight matrix $W$, bias $\mathbf{b}$, dropout with probability $p$ of *not* being dropped, the layer can be written as

$$y_j = \left( \sum_i D_i \cdot W_{ij} \cdot x_i \right) + b_j$$

where $D_i \sim \text{Bern}(p)$ are i.i.d. Bernoulli random variables with probability parameter $p$. The variance of an embedding component can be written as follows:

$$\text{Var}(y_j) = \text{Var}\left[ \left( \sum_i D_i \cdot W_{ij} \cdot x_i \right) + b_j \right]$$

Variance is invariant to changes in a location parameter, and the $D_i$ are i.i.d. allowing us to write:

$$\text{Var}(y_j) = \sum_i (W_{ij} \cdot x_i)^2 \, \text{Var}(D_i) = \sum_i (W_{ij} \cdot x_i)^2 \, p \, (1 - p)$$

## A.6 Dataset Links

Data used in the image classification experiments can be found here:

- `http://yann.lecun.com/exdb/mnist/`
- `https://github.com/davidflanagan/notMNIST-to-MNIST`
- `https://github.com/rois-codh/kmnist`

- https://github.com/zalandoresearch/fashion-mnist

Data used in the language classification experiments can be found here: https://zenodo.org/record/841984#.YK0r8S1h1pQ

Part of the data used in the malware detection experiments can be found here:

- https://github.com/elastic/ember
- https://github.com/fabriciojoc/brazilian-malware-dataset

The 1.1TB of raw PE files are not available as part of EMBER2018, but they can be downloaded via VirusTotal: https://www.virustotal.com/gui/

# Bibliography


[1] John Aldrich. "R. A. Fisher on Bayes and Bayes' Theorem". In: *Bayesian Analysis* 3.1 (2008), pp. 161–170. ISSN: 19360975. DOI: 10.1214/08-BA306.

[2] Joost van Amersfoort et al. "On Feature Collapse and Deep Kernel Learning for Single Forward Pass Uncertainty". In: (Feb. 2021). URL: http://arxiv.org/abs/2102.11409.

[3] Joost van Amersfoort et al. "Uncertainty Estimation Using a Single Deep Deterministic Neural Network". In: (Mar. 2020). URL: http://arxiv.org/abs/2003.02037.

[4] Brandon Amos. "Differentiable optimization-based modeling for machine learning". PhD thesis. Carnegie Mellon University, 2019.

[5] Blake Anderson, David Mcgrew, and Subharthi Paul. "Discovering Human and Machine Readable Descriptions of Malware Families". In: *The AAAI-16 workshop on Artificial Intelligence for Cyber Sercurity (AICS)* (2016), pp. 150–156.

[6] Hyrum S. Anderson and Phil Roth. "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models". In: (2018). URL: http://arxiv.org/abs/1804.04637.

[7] Hyrum S. Anderson et al. "Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning". In: (2018). URL: http://arxiv.org/abs/1801.08917.

[8] Ross Anderson et al. "Measuring the Changing Cost of Cybercrime". In: *Workshop on the Economics of Information Security (WEIS)* (2019). URL: https://weis2019.econinfosec.org/wp-content/uploads/sites/6/2019/05/WEIS_2019_paper_25.pdf.

[9] Ross Anderson et al. "Measuring the Cost of Cybercrime". In: *Workshop on the Economics of Information Security (WEIS)* (2012).

[10] Javier Antorán, James Urquhart Allingham, and José Miguel Hernández-Lobato. "Depth Uncertainty in Neural Networks". In: (2020). URL: http://arxiv.org/abs/2006.08437.

[11] Javier Antorán, James Urquhart Allingham, and José Miguel Hernández-Lobato. "Variational Depth Search in ResNets". In: (2020), pp. 1–13. URL: http://arxiv.org/abs/2002.02797.

[12] Daniel Arp et al. "Technical Report IFI-TB-2013-02 D REBIN : Efficient and Explainable Detection of Android Malware in Your Pocket". In: (2013).

[13] Arsenii Ashukha et al. "Pitfalls of In-Domain Uncertainty Estimation and Ensembling in Deep Learning". In: *International Conference on Learning Representations (ICLR)*. 2020.

[14] Amir Atapour-Abarghouei, Stephen Bonner, and Andrew Stephen McGough. "A King's Ransom for Encryption: Ransomware Classification using Augmented One-Shot Learning and Bayesian Approximation". In: *Proceedings - 2019 IEEE International Conference on Big Data, Big Data 2019* (2019), pp. 1601–1606. DOI: 10.1109/BigData47090.2019.9005540.

[15] Josh Attenberg, Panagiotis Ipeirotis, and Foster Provost. "Beat the Machine: Challenging Workers to Find the Unknown Unknowns". In: *AAAI* (2011).

[16] Michael Backes and Mohammad Nauman. "LUNA: Quantifying and Leveraging Uncertainty in Android Malware Analysis through Bayesian Machine Learning". In: *Proceedings - 2nd IEEE European Symposium on Security and Privacy, EuroS and P 2017* (2017), pp. 204–217. DOI: 10.1109/EuroSP.2017.24.

[17] Peter L. Bartlett, Steven N. Evans, and Philip M. Long. "Representing smooth functions as compositions of near-identity functions with implications for deep network optimization". In: (Apr. 2018). URL: http://arxiv.org/abs/1804.05012.

[18] Thomas Bayes. "An Essay towards solving a Problem in the Doctrine of Chances". In: *Transactions of the Royal Society* (1763).

[19] Timothy E.J. Behrens et al. "Learning the value of information in an uncertain world". In: *Nature Neuroscience* 10.9 (2007), pp. 1214–1221. ISSN: 10976256. DOI: 10.1038/nn1954.

[20] José M. Bernardo and Adrian F.M. Smith. *Bayesian Theory*. 2008. ISBN: 9780470316870. DOI: 10.1002/9780470316870.

[21] Michael Betancourt. "A Conceptual Introduction to Hamiltonian Monte Carlo". In: (2017). URL: http://arxiv.org/abs/1701.02434.

[22] Battista Biggio and Fabio Roli. "Wild patterns: Ten years after the rise of adversarial machine learning". In: *Pattern Recognition* 84 (2018), pp. 317–331. ISSN: 00313203. DOI: 10.1016/j.patcog.2018.07.023.

[23] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. "Variational Inference: A Review for Statisticians". In: *Journal of the American Statistical Association* 112.518 (2017), pp. 859–877. ISSN: 1537274X. DOI: 10.1080/01621459.2017.1285773.

[24] Charles Blundell et al. "Weight uncertainty in neural networks". In: *32nd International Conference on Machine Learning, ICML 2015* 2 (2015), pp. 1613–1622.

[25] John Bradshaw, Alexander G. de G. Matthews, and Zoubin Ghahramani. "Adversarial Examples, Uncertainty, and Transfer Testing Robustness in Gaussian Process Hybrid Deep Networks". In: (2017), pp. 1–33. URL: http://arxiv.org/abs/1707.02476.

[26] Leo Breiman. "Bagging predictors". In: *Machine Learning* 24.2 (1996), pp. 123–140. DOI: 10.1007/BF00058655.

[27] Richard P Brent. *Algorithms for Minimization Without Derivatives*. 1972.

[28] Yaroslav Bulatov. *notMNIST dataset*. 2011. URL: http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html.

[29] Trevor Campbell and Boyan Beronov. "Sparse Variational Inference : Bayesian Coresets from Scratch". In: NeurIPS (2019).

[30] Trevor Campbell and Tamara Broderick. "Automated Scalable Bayesian Inference via Hilbert Coresets". In: 20 (2019), pp. 1–38.

[31] Trevor Campbell and Tamara Broderick. "Bayesian Coreset Construction via Greedy Iterative Geodesic Ascent". In: (2018).

[32] Fabricio Ceschin et al. "The Need for Speed: An Analysis of Brazilian Malware Classifers". In: *IEEE Security and Privacy* 16.6 (Nov. 2019), pp. 31–41. ISSN: 15584046. DOI: 10.1109/MSEC.2018.2875369.

[33] Jie Chang et al. *Data Uncertainty Learning in Face Recognition*. Tech. rep. 2020.

[34] Suming Chen, Arthur Choi, and Adnan Darwiche. "Value of information based on decision robustness". In: *Proceedings of the National Conference on Artificial Intelligence* 5 (2015), pp. 3503–3510.

[35] Tianqi Chen, Emily B. Fox, and Carlos Guestrin. "Stochastic gradient Hamiltonian Monte Carlo". In: *31st International Conference on Machine Learning, ICML 2014* 5 (2014), pp. 3663–3676.

[36] C. K. Chow. *On optimum recognition error and reject tradeoff*. 1969.

[37] C.K. Chow. "An Optimum Character Recognition System Using Decision Functions". In: *IRE Transactions on Electronic Computers* EC-7.2 (1958), p. 180. ISSN: 03679950. DOI: 10.1109/TEC.1958.5222530.

[38] Sanghyuk Chun et al. "Probabilistic Embeddings for Cross-Modal Retrieval". In: (Jan. 2021). URL: http://arxiv.org/abs/2101.05068.

[39] Tarin Clanuwat et al. "Deep Learning for Classical Japanese Literature". In: (Dec. 2018). DOI: 10.20676/00000341. URL: http://arxiv.org/abs/1812.01718%20http://dx.doi.org/10.20676/00000341.

[40] Adam D. Cobb et al. "Introducing an Explicit Symplectic Integration Scheme for Riemannian Manifold Hamiltonian Monte Carlo". In: 3 (2019), pp. 1–15. URL: http://arxiv.org/abs/1910.06243.

[41] Corinna Cortes, Giulia DeSalvo, and Mehryar Mohri. "Learning with rejection". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9925 LNAI (2016), pp. 67–82. ISSN: 16113349. DOI: 10.1007/978-3-319-46379-7{_}5.

[42] Andreas C Damianou and Neil D Lawrence. "Deep Gaussian Processes". In: 31 (2013).

[43] Erik Daxberger et al. “Laplace Redux – Effortless Bayesian Deep Learning”. In: (June 2021). URL: http://arxiv.org/abs/2106.14806.

[44] Luca Demetrio et al. “Explaining vulnerabilities of deep learning to adversarial malware binaries”. In: *CEUR Workshop Proceedings* 2315 (2019). ISSN: 16130073.

[45] Stefan Depeweg et al. “Decomposition of Uncertainty in Bayesian Deep Learning for Efficient and Risk-sensitive Learning”. In: *35th International Conference on Machine Learning, ICML 2018* 3 (2018), pp. 1920–1934.

[46] Stefan Depeweg et al. “Learning and Policy Search in Stochastic Dynamical Systems with Bayesian Neural Networks”. In: (2016), pp. 1–14. URL: http://arxiv.org/abs/1605.07127.

[47] Marie Des Jardins, James MacGlashan, and Kiri L. Wagstaff. “Confidence-based feature acquisition to minimize training and test costs”. In: *Proceedings of the 10th SIAM International Conference on Data Mining, SDM 2010* (2010), pp. 514–524. DOI: 10.1137/1.9781611972801.45.

[48] Gabriel Dulac-Arnold et al. “Datum-Wise classification: A sequential approach to sparsity”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6911 LNAI (2011), pp. 375–390. ISSN: 16113349. DOI: 10.1007/978-3-642-23780-5{_}34.

[49] David Duvenaud and Ryan P Adams. “Black-Box Stochastic Variational Inference in Five Lines of Python”. In: (2014), pp. 1–4.

[50] Elad Eban et al. “Scalable Learning of Non-Decomposable Objectives”. In: *AISTATS*. Vol. 54. Proceedings of Machine Learning Research. 2017, pp. 832–840. URL: http://proceedings.mlr.press/v54/eban17a.html.

[51] Melissa Eddy and Nicole Perlroth. *Cyber Attack Suspected in German Woman’s Death*. Sept. 2021.

[52] Bradley Efron and Robert Tibshirani. “Bootstrap Methods for Standard Errors, Confidence Intervals, and Other Measures of Statistical Accuracy”. In: *Statistical Science* (1986).

[53] Manuel Egele et al. “A survey on automated dynamic malware-analysis techniques and tools”. In: *ACM Computing Surveys* 44.2 (2012). ISSN: 03600300. DOI: 10.1145/2089125.2089126.

[54] Gerald A. Feltham. “The value of information”. In: *The Accounting Review* (1968).

[55] Angelos Filos et al. “A Systematic Comparison of Bayesian Deep Learning Robustness in Diabetic Retinopathy Tasks”. In: NeurIPS (2019), pp. 1–12. URL: http://arxiv.org/abs/1912.10481.

[56] William Fleshman et al. “Non-Negative Networks Against Adversarial Attacks”. In: (2018). URL: http://arxiv.org/abs/1806.06108.

[57] William Fleshman et al. “Static Malware Detection & Subterfuge: Quantifying the Robustness of Machine Learning and Current Anti-Virus”. In: *MALWARE 2018 - Proceedings of the 2018 13th International Conference on Malicious and Unwanted Software* (2019), pp. 3–12. DOI: 10.1109/MALWARE.2018.8659360.

[58] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. “Deep ensembles: A loss landscape perspective”. In: *arXiv* (2019), pp. 1–15.

[59] Yoshiro Fukushima et al. “A behavior based malware detection scheme for avoiding false positive”. In: *IEEE Workshop on Secure Network Protocols*. 2010, pp. 79–84. DOI: 10.1109/NPSEC.2010.5634444.

[60] Yarin Gal. “Uncertainty in Deep Learning”. PhD thesis. University of Cambridge, 2016.

[61] Yarin Gal and Zoubin Ghahramani. “Dropout as a Bayesian Approximation: Appendix”. In: *33rd International Conference on Machine Learning, ICML 2016* 3 (2016), pp. 1661–1680.

[62] Yarin Gal and Zoubin Ghahramani. “Dropout as a Bayesian approximation: Representing model uncertainty in deep learning”. In: *33rd International Conference on Machine Learning, ICML 2016* 3 (2016), pp. 1651–1660.

[63] Yarin Gal, Jiri Hron, and Alex Kendall. “Concrete dropout”. In: *Advances in Neural Information Processing Systems* 2017-Decem (2017), pp. 3582–3591. ISSN: 10495258.

[64] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. “Deep Bayesian active learning with image data”. In: *34th International Conference on Machine Learning, ICML 2017* 3 (2017), pp. 1923–1932.

[65] Yarin Gal and Lewis Smith. “Sufficient Conditions for Idealised Models to Have No Adversarial Examples: a Theoretical and Empirical Study with Bayesian Neural Networks”. In: (2018), pp. 1–16. URL: http://arxiv.org/abs/1806.00667.

[66] Tianshi Gao and Daphne Koller. “Active classification based on value of classifier”. In: *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011* (2011), pp. 1–9.

[67] Marta Garnelo et al. “Neural Processes”. In: (2018). URL: http://arxiv.org/abs/1807.01622.

[68] Dragos Gavrilut, Razvan Benchea, and Cristina Vatamanu. “Optimized Zero False Positives Perceptron Training for Malware Detection”. In: *2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. 2012, pp. 247–253. DOI: 10.1109/SYNASC.2012.34.

[69] Daniel Gibert, Carles Mateu, and Jordi Planes. "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges". In: *Journal of Network and Computer Applications* 153.January (2020), p. 102526. ISSN: 10958592. DOI: 10.1016/j.jnca.2019.102526. URL: https://doi.org/10.1016/j.jnca.2019.102526.

[70] Ryan Giordano, Tamara Broderick, and Michael Jordan. "Linear response methods for accurate covariance estimates from mean field variational bayes". In: *Advances in Neural Information Processing Systems* 2015-Janua (2015), pp. 1441–1449. ISSN: 10495258.

[71] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: (2014), pp. 1–11. URL: http://arxiv.org/abs/1412.6572.

[72] Alex Graves. "Practical variational inference for neural networks". In: *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011* (2011), pp. 1–9.

[73] Nikolaus Hansen and Andreas Ostermeier. *Completely Derandomized Self-Adaptation in Evolution Strategies*. Tech. rep.

[74] Richard Harang and Ethan M. Rudd. "SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection". In: (2020). URL: http://arxiv.org/abs/2012.07634.

[75] Dan Hendrycks and Kevin Gimpel. "A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks". In: (2016), pp. 1–12. URL: http://arxiv.org/abs/1610.02136.

[76] José Miguel Hernández-Lobato and Ryan P. Adams. "Probabilistic backpropagation for scalable learning of Bayesian neural networks". In: *32nd International Conference on Machine Learning, ICML 2015* 3 (2015), pp. 1861–1869.

[77] José Miguel Hernández-Lobato et al. "Black-Box alpha-Divergence Minimization". In: 48 (2018).

[78] Geoffrey E. Hinton and Drew van Camp. "Keeping neural networks simple by minimizing the description length of the weights". In: (1993), pp. 5–13. DOI: 10.1145/168304.168306.

[79] Geoffrey E. Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: (2012), pp. 1–18. URL: http://arxiv.org/abs/1207.0580.

[80] Jennifer A Hoeting et al. "Bayesian Averaging Models". In: *Statistical Science* 14.4 (1999), pp. 382–417.

[81] Matthew D. Hoffman and Andrew Gelman. "The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo". In: *Journal of Machine Learning Research* 15.2008 (2014), pp. 1593–1623. ISSN: 15337928.

[82] Matthew D. Hoffman et al. “Stochastic variational inference”. In: *Journal of Machine Learning Research* 14 (2013), pp. 1303–1347. ISSN: 15324435.

[83] Giles Hooker. “Diagnosing extrapolation: Tree-based density estimation”. In: *KDD-2004 - Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2004), pp. 569–574.

[84] David John Henry Howie. “Interpretations of probability, 1919-1939: Harold Jeffreys, R. A. Fisher, and the Bayesian controversy”. PhD thesis. University of Pennsylvania, 1999. URL: https://repository.upenn.edu/dissertations/AAI9937734/.

[85] Gao Huang et al. *Snapshot Ensembles: Train 1, get M for free*. Tech. rep. 2017. URL: www.kaggle.com.

[86] Jonathan H Huggins, Trevor Campbell, and Tamara Broderick. “Coresets for Scalable Bayesian Logistic Regression”. In: Nips (2016).

[87] Paul Hyman. “Cybercrime: it’s serious, but exactly how serious?” In: *Communications of the ACM* 56.3 (2013), pp. 18–20. ISSN: 0001-0782. DOI: 10.1145/2428556.2428563.

[88] Pavel Izmailov et al. “Averaging weights leads to wider optima and better generalization”. In: *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018* 2 (2018), pp. 876–885.

[89] Robert A. Jacobs et al. “Adaptive Mixtures of Local Experts”. In: *Neural Computation* 3.1 (1991), pp. 79–87.

[90] Harold Jeffreys. *Theory of Probability*. 1939.

[91] Michael I. Jordan et al. “Introduction to variational methods for graphical models”. In: *Machine Learning* 37.2 (1999), pp. 183–233. ISSN: 08856125. DOI: 10.1023/A:1007665907178.

[92] Chanhyun Kang et al. “Ensemble Models for Data-driven Prediction of Malware Infections”. In: *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. 2016, pp. 583–592. DOI: 10.1145/2835776.2835834.

[93] Guolin Ke et al. *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*. Tech. rep. URL: https://github.com/Microsoft/LightGBM.

[94] Alex Kendall and Yarin Gal. “What uncertainties do we need in Bayesian deep learning for computer vision?” In: *Advances in Neural Information Processing Systems* 2017-Decem.Nips (2017), pp. 5575–5585. ISSN: 10495258.

[95] Jeffrey O Kephart et al. “Biologically Inspired Defenses Against Computer Viruses”. In: *IJCAI*. 1995, pp. 985–996.

[96] Mohammad Emtiyaz Khan et al. “Fast and Scalable Bayesian Deep Learning by Weight-Perturbation in Adam”. In: (2018).

[97] Khaled N. Khasawneh et al. “Ensemble Learning for Low-Level Hardware-Supported Malware Detection”. In: *Research in Attacks, Intrusions, and Defenses*. 2015.

[98] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: (2014), pp. 1–15. ISSN: 09252312. DOI: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503. URL: http://arxiv.org/abs/1412.6980.

[99] Clemens Kolbitsch et al. “Effective and Efficient Malware Detection at the End Host”. In: *USENIX Security Symposium*. USENIX Association, 2009.

[100] Bojan Kolosnjaji et al. “Adversarial malware binaries: Evading deep learning for malware detection in executables”. In: *European Signal Processing Conference* 2018-Septe (2018), pp. 533–537. ISSN: 22195491. DOI: 10.23919/EUSIPCO.2018.8553214.

[101] Marek Krčál et al. “Deep convolutional malware classifiers can learn from raw executables and labels only”. In: *6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings* 2016 (2018), pp. 2016–2019.

[102] Felix Kreuk et al. “Deceiving End-to-End Deep Learning Malware Detectors using Adversarial Examples”. In: (2018). URL: http://arxiv.org/abs/1802.04528.

[103] Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. “Learnable Uncertainty under Laplace Approximations”. In: (Oct. 2020). URL: http://arxiv.org/abs/2010.02720.

[104] Alp Kucukelbir and Andrew Gelman. “Automatic Differentiation Variational Inference”. In: 18 (2017), pp. 1–45.

[105] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. “Simple and scalable predictive uncertainty estimation using deep ensembles”. In: *Advances in Neural Information Processing Systems*. 2017.

[106] Pierre-Simon Laplace. *Théorie analytique des probabilitiés*. 1812.

[107] Bo Li et al. “Large-Scale Identification of Malicious Singleton Files”. In: *7TH ACM Conference on Data and Application Security and Privacy*. 2017.

[108] Guanjun Lin et al. “Software Vulnerability Detection Using Deep Neural Networks: A Survey”. In: *Proceedings of the IEEE* (2020), pp. 1–24. ISSN: 15582256. DOI: 10.1109/JPROC.2020.2993293.

[109] Che Hsun Liu, Zhi Jie Zhang, and Sheng De Wang. “An android malware detection approach using Bayesian inference”. In: *Proceedings - 2016 16th IEEE International Conference on Computer and Information Technology* (2017), pp. 476–483. DOI: 10.1109/CIT.2016.76.

[110] Han Liu and Larry Wasserman. “Bayesian Inference”. In: *Statistical Machine Learning*. In Preparation., 2014. Chap. 12, pp. 299–351.

[111] Jeremiah Zhe Liu et al. “Simple and principled uncertainty estimation with deterministic deep learning via distance awareness”. In: *arXiv* NeurIPS (2020).

[112] Li Ping Liu et al. “TEFE: A time-efficient approach to feature extraction”. In: *Proceedings - IEEE International Conference on Data Mining, ICDM* (2008), pp. 423–432. ISSN: 15504786. DOI: 10.1109/ICDM.2008.48.

[113] Liu Liu and Baosheng Wang. “Malware classification using gray-scale images and ensemble learning”. In: *International Conference on Systems and Informatics*. 2016.

[114] Yi An Ma, Tianqi Chen, and Emily B. Fox. “A complete recipe for stochastic gradient MCMC”. In: *Advances in Neural Information Processing Systems* 2015-Janua (2015), pp. 2917–2925. ISSN: 10495258.

[115] David J. C. MacKay. “A Practical Bayesian Framework for Backpropagation Networks”. In: *Neural Computation* 472.1 (1992), pp. 448–472.

[116] David J. C. MacKay. “Information-Based Objective Functions for Active Data Selection”. In: *Neural Computation* 4.4 (1992), pp. 590–604. ISSN: 0899-7667. DOI: 10.1162/neco.1992.4.4.590.

[117] David J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*. 7.2. Vol. 13. Cambridge University Press, 2005.

[118] David John Cameron MacKay. “Bayesian Methods for Adaptive Models”. In: *CIT theses* (1991).

[119] Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. “Early Stopping is Nonparametric Variational Inference”. In: 51 (2015), pp. 1070–1077. URL: http://arxiv.org/abs/1504.01344.

[120] Wesley Maddox et al. “A Simple Baseline for Bayesian Uncertainty in Deep Learning”. In: NeurIPS (2019), pp. 1–25. URL: http://arxiv.org/abs/1902.02476.

[121] David Madras, Toniann Pitassi, and Richard Zemel. “Predict responsibly: Improving fairness and accuracy by learning to defer”. In: *Advances in Neural Information Processing Systems* 2018-Decem (2018), pp. 6147–6157. ISSN: 10495258.

[122] Amit Mandelbaum and Daphna Weinshall. *Distance-based Confidence Score for Neural Network Classifiers*. Tech. rep. 2017.

[123] Gael M. Martin, David T. Frazier, and Christian P. Robert. “Computing bayes: Bayesian computation from 1763 to the 21st century”. In: *arXiv* (2020), pp. 1–47.

[124] J. McCarthy. “Measures of the Value of Information”. In: *Proceedings of the National Academy of Sciences* 42.9 (1956), pp. 654–655. ISSN: 0027-8424. DOI: 10.1073/pnas.42.9.654.

[125] Eitan Menahem et al. “Improving Malware Detection by Applying Multi-inducer Ensemble”. In: *Comput. Stat. Data Anal.* 53.4 (2009), pp. 1483–1494. DOI: 10.1016/j.csda.2008.10.015.

[126] Thomas P. Minka. "Expectation Propagation for approximate Bayesian inference". In: (2013). URL: http://arxiv.org/abs/1301.2294.

[127] Takeru Miyato et al. "Spectral Normalization for Generative Adversarial Networks". In: (Feb. 2018). URL: http://arxiv.org/abs/1802.05957.

[128] Abedelaziz Mohaisen and Omar Alrawi. "Unveiling zeus automated classification of malware samples". In: *WWW 2013 Companion - Proceedings of the 22nd International Conference on World Wide Web* (2013), pp. 829–832. DOI: 10.1145/2487788.2488056.

[129] Hussein Mozannar and David Sontag. "Consistent Estimators for Learning to Defer to an Expert". In: (2020). URL: http://arxiv.org/abs/2006.01862.

[130] Jishnu Mukhoti et al. "Deterministic Neural Networks with Appropriate Inductive Biases Capture Epistemic and Aleatoric Uncertainty". In: (Feb. 2021). URL: http://arxiv.org/abs/2102.11582.

[131] M Arthur Munson and W Philip Kegelmeyer. "Builtin vs . Auxiliary Detection of Extrapolation Risk". In: February (2013).

[132] Kevin P Murphy. *Machine learning: a probabilistic perspective (adaptive computation and machine learning series)*. The MIT Press, 2012. ISBN: 0262018020.

[133] Radford M. Neal. "Bayesian Learning for Neural Networks". PhD thesis. University of Toronto, 1995.

[134] Andre T. Nguyen et al. "Leveraging Uncertainty for Improved Static Malware Detection Under Extreme False Positive Constraints". In: *IJCAI-21 1st International Workshop on Adaptive Cyber Defense* (2021). URL: http://arxiv.org/abs/2108.04081.

[135] NISC. *NISC Survey Results*. Tech. rep. Neustar International Security Council, 2020. URL: https://www.nisc.neustar/nisc-survey-results/.

[136] Alejandro Noriega-Campero et al. "Active fairness in algorithmic decision making". In: *AIES 2019 - Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society* (2019), pp. 77–83. DOI: 10.1145/3306618.3314277.

[137] Seong Joon Oh et al. "Modeling Uncertainty with Hedged Instance Embedding". In: (Sept. 2018). URL: http://arxiv.org/abs/1810.00319.

[138] Yaniv Ovadia et al. "Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty Under Dataset Shift". In: NeurIPS (2019). URL: http://arxiv.org/abs/1906.02530.

[139] Nicolas Papernot. "Characterizing the Limits and Defenses of Machine Learning in Adversarial Settings". In: *ProQuest Dissertations and Theses* May (2018), p. 178.

[140] Adam Paszke et al. "PyTorch: An imperative style, high-performance deep learning library". In: *Advances in Neural Information Processing Systems* 32.NeurIPS (2019). ISSN: 10495258.

[141] Tim Pearce et al. "Uncertainty in Neural Networks: Approximately Bayesian Ensembling". In: (2018). URL: http://arxiv.org/abs/1810.05546.

[142] Roberto Perdisci, Andrea Lanzi, and Wenke Lee. "McBoost: Boosting Scalability in Malware Collection and Analysis Using Statistical Classification of Executables". In: *Annual Computer Security Applications Conference (ACSAC)*. 2008.

[143] Andreas Pitsillidis et al. "Botnet judo: Fighting spam with itself". In: *Symposium on Network and Distributed System Security (NDSS)*. 2010.

[144] Ponemon Institute. "2017 Cost of Data Breach Study: Global Overview". In: *IBM Security* June (2018), p. 47.

[145] Remus Pop and Patric Fulop. "Deep Ensemble Bayesian Active Learning : Addressing the Mode Collapse issue in Monte Carlo dropout via Ensembles". In: (2018), pp. 1–15. URL: http://arxiv.org/abs/1811.03897.

[146] Janis Postels et al. "The Hidden Uncertainty in a Neural Networks Activations". In: (Dec. 2020). URL: http://arxiv.org/abs/2012.03082.

[147] Edward Raff and Charles Nicholas. "A Survey of Machine Learning Methods and Challenges for Windows Malware Classification". In: (2020), pp. 1–48. URL: http://arxiv.org/abs/2006.09271.

[148] Edward Raff et al. "An investigation of byte n-gram features for malware classification". In: *Journal of Computer Virology and Hacking Techniques* 14.1 (2018). ISSN: 22638733. DOI: 10.1007/s11416-016-0283-1.

[149] Edward Raff et al. "Classifying Sequences of Extreme Length with Constant Memory Applied to Malware Detection". In: *AAAI*. 2021.

[150] Edward Raff et al. "KiloGrams: Very Large N-Grams for Malware Classification". In: *Proceedings of KDD 2019 Workshop on Learning and Mining for Cybersecurity (LEMINCS'19)*. 2019. URL: https://arxiv.org/abs/1908.00200.

[151] Edward Raff et al. "Malware Detection by Eating a Whole EXE". In: (2017). URL: http://arxiv.org/abs/1710.09435.

[152] M Zubair Rafique and Juan Caballero. "FIRMA: Malware Clustering and Network Signature Generation with Mixed Network Behaviors". In: *Proceedings of the 16th International Symposium on Research in Attacks, Intrusions, and Defenses - Volume 8145*. RAID 2013. New York, NY, USA: Springer-Verlag New York, Inc., 2013, pp. 144–163. DOI: 10.1007/978-3-642-41284-4_8.

[153] Rajesh Ranganath, Sean Gerrish, and David M. Blei. "Black box variational inference". In: *Journal of Machine Learning Research* 33 (2014), pp. 814–822. ISSN: 15337928.

[154] C E Rasmussen and C K I Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[155] Jie Ren et al. "Likelihood Ratios for Out-of-Distribution Detection". In: (June 2019). URL: http://arxiv.org/abs/1906.02845.

[156] Hippolyt Ritter, Aleksandar Botev, and David Barber. "A Scalable Laplace Approximation for Neural Networks". In: *ICLR*. 2018.

[157] Christian Rossow et al. "Prudent practices for designing malware experiments: Status quo and outlook". In: *Proceedings - IEEE Symposium on Security and Privacy* (2012), pp. 65–79. ISSN: 10816011. DOI: 10.1109/SP.2012.14.

[158] Thomas Rückstieß, Christian Osendorfer, and Patrick Van Der Smagt. "Sequential feature selection for classification". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7106 LNAI (2011), pp. 132–141. ISSN: 03029743. DOI: 10.1007/978-3-642-25832-9{_}14.

[159] Ethan M Rudd et al. *ALOHA: Auxiliary Loss Optimization for Hypothesis Augmentation*. ISBN: 9781939133069. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/rudd.

[160] Tim G. J. Rudner et al. "On the Connection between Neural Processes and Gaussian Processes with Deep Kernels". In: NeurIPS (2018), pp. 1–6.

[161] Maytal Saar-tsechansky, Prem Melville, and Foster Provost. "Active Feature-Value Acquisition". In: (), pp. 1–32. URL: papers://5e3e5e59-48a2-47c1-b6b1-a778137d3ec1/Paper/p1779.

[162] Riccardo Sartea, Alessandro Farinelli, and Matteo Murari. "Bayesian Active Malware Analysis". In: Aamas (2020), pp. 1206–1214.

[163] Joshua Saxe and Konstantin Berlin. "Deep neural network based malware detection using two dimensional binary program features". In: *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. 2015. DOI: 10.1109/MALWARE.2015.7413680.

[164] Rens van de Schoot et al. "Bayesian statistics and modelling". In: 0123456789 (2021). DOI: 10.1038/s43586-020-00001-2.

[165] Peter Schulam and Suchi Saria. "Can you trust this prediction? Auditing pointwise reliability after learning". In: *AISTATS 2019 - 22nd International Conference on Artificial Intelligence and Statistics* 89 (2020).

[166] Marcos Sebastián et al. *AVCLASS: A Tool for Massive Malware Labeling*. Tech. rep. 2016. URL: https://github.com/malicialab/avclass.

[167] Yichun Shi and Anil K Jain. *Probabilistic Face Embeddings*. Tech. rep. 2019. URL: https://github.com/seasonSH/.

[168] Hajin Shim, Sung Ju Hwang, and Eunho Yang. "Joint active feature acquisition and classification with variable-size set encoding". In: *Advances in Neural Information Processing Systems* 2018-Decem.NeurIPS (2018), pp. 1368–1378. ISSN: 10495258.

[169] Lewis Smith and Yarin Gal. "Understanding measures of uncertainty for adversarial example detection". In: *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018* 2 (2018), pp. 560–569.

[170] Michael R. Smith et al. "Mind the Gap: On Bridging the Semantic Gap between MachineLearning and Malware Analysis". In: *arXiv* (2020), pp. 49–60.

[171] Charles Smutz and Angelos Stavrou. "Malicious PDF detection using metadata and structural features". In: *Proceedings of the 28th Annual Computer Security Applications Conference*. 2012, pp. 239–248.

[172] Eugene C. Spafford. "Is Anti-virus Really Dead?" In: *Computers & Security* 44 (2014), p. iv. DOI: 10.1016/S0167-4048(14)00082-0.

[173] Michael Spreitzenbarth et al. "Mobile-sandbox: Having a deeper look into Android applications". In: *Proceedings of the ACM Symposium on Applied Computing* (2013), pp. 1808–1815. DOI: 10.1145/2480362.2480701.

[174] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958.

[175] Philipp Terhörst et al. "SER-FIQ: Unsupervised Estimation of Face Image Quality Based on Stochastic Embedding Robustness". In: *CVPR*. 2020.

[176] Martin Thoma. "The WiLI benchmark dataset for written language identification". In: (Jan. 2018). URL: http://arxiv.org/abs/1801.07779.

[177] Michalis K. Titsias and Miguel Lázaro-Gredilla. "Doubly stochastic variational bayes for non-conjugate inference". In: *31st International Conference on Machine Learning, ICML 2014* 5 (2014), pp. 4056–4069.

[178] Richard Eric Turner and Maneesh Sahani. "Two problems with variational expectation maximisation for time series models". In: *Bayesian Time Series Models* 9780521196 (2011), pp. 104–124. DOI: 10.1017/CBO9780511984679.006.

[179] Meet P. Vadera et al. "URSABench: Comprehensive Benchmarking of Approximate Bayesian Inference Methods for Deep Neural Networks". In: (July 2020). URL: http://arxiv.org/abs/2007.04466.

[180] Pascal Vincent et al. "Extracting and Composing Robust Features with Denoising Autoencoders". In: *ICML*. 2008.

[181] Pascal Vincent et al. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion". In: *Journal of Machine Learning Research* 11 (2010), pp. 3371–3408.

[182] Xin Wang et al. "IDK Cascades: Fast deep learning by learning not to Overthink". In: *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018* 2 (2018), pp. 580–590.

[183] David Weiss and Ben Taskar. "Structured prediction cascades". In: *Journal of Machine Learning Research* 9 (2010), pp. 916–923. ISSN: 15324435.

[184] Max Welling and Yee Whye Teh. "Bayesian Learning via Stochastic Gradient Langevin Dynamics". In: *Proceedings of the 28th International Conference on Machine Learning*. 2011.

[185] Yeming Wen, Dustin Tran, and Jimmy Ba. "Batchensemble: An alternative approach to efficient ensemble and lifelong learning". In: *arXiv:2002.06715* (2020), pp. 1–20.

[186] Florian Wenzel et al. "Hyperparameter Ensembles for Robustness and Uncertainty Quantification". In: *arXiv* NeurIPS (2020).

[187] Bryan Wilder, Eric Horvitz, and Ece Kamar. "Learning to Complement Humans". In: (2020), pp. 1526–1533. DOI: 10.24963/ijcai.2020/212.

[188] Christopher K.I. Williams. "Computing with infinite networks". In: *Advances in Neural Information Processing Systems* (1997), pp. 295–301. ISSN: 10495258.

[189] Andrew Gordon Wilson and Pavel Izmailov. "Bayesian Deep Learning and a Probabilistic Perspective of Generalization". In: 3 (2020). URL: http://arxiv.org/abs/2002.08791.

[190] Andrew Gordon Wilson et al. "Stochastic variational deep kernel learning". In: *Advances in Neural Information Processing Systems* Nips (2016), pp. 2594–2602. ISSN: 10495258.

[191] David H. Wolpert. "Stacked generalization". In: *Neural networks* 5 (1992), pp. 241–259.

[192] David H. Wolpert and William G. Macready. "No free lunch theorems for optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82. ISSN: 1089778X. DOI: 10.1109/4235.585893.

[193] Christian Wressnegger et al. "Automatically inferring malware signatures for anti-virus assisted attacks". In: *ASIA CCS 2017 - Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security* (2017), pp. 587–598. DOI: 10.1145/3052973.3053002.

[194] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: (Aug. 2017). URL: http://arxiv.org/abs/1708.07747.

[195] Tim Z. Xiao, Aidan N. Gomez, and Yarin Gal. "Wat zei je? Detecting Out-of-Distribution Translations with Variational Transformers". In: NeurIPS (2020), pp. 4–7. URL: http://arxiv.org/abs/2006.08344.

[196] Zhixiang Xu, Kilian Q. Weinberger, and Olivier Chapelle. "The Greedy Miser: Learning under test-time budgets". In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012* 2 (2012), pp. 1175–1182.

[197] Zhixiang Xu et al. “Cost-sensitive tree of classifiers”. In: *30th International Conference on Machine Learning, ICML 2013* 28.PART 1 (2013), pp. 133–141.

[198] Zhixiang E. Xu et al. “Classifier cascades and trees for minimizing feature evaluation cost”. In: *Journal of Machine Learning Research* 15 (2014), pp. 2113–2144. ISSN: 15337928.

[199] Jinpei Yan, Yong Qi, and Qifan Rao. “Detecting Malware with an Ensemble Method Based on Deep Neural Network”. In: *Security and Communication Networks* 2018 (2018). ISSN: 19390122. DOI: 10.1155/2018/7247095.

[200] Yann LeCun et al. “Gradient-Based Learning Applied to Document Recognition”. In: *IEEE* (1998).

[201] Jiayu Yao et al. “Quality of Uncertainty Quantification for Bayesian Neural Network Inference”. In: (2019). URL: http://arxiv.org/abs/1906.09686.

[202] Yanfang Ye et al. “Automatic Malware Categorization Using Cluster Ensemble”. In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2010, pp. 95–104. DOI: 10.1145/1835804.1835820.

[203] Suleiman Y. Yerima, Sakir Sezer, and Gavin McWilliams. “Analysis of Bayesian classification-based approaches for Android malware detection”. In: *IET Information Security* 8.1 (2014), pp. 25–36. ISSN: 17518709. DOI: 10.1049/iet-ifs.2013.0095.

[204] Scott Wen-tau Yih, Joshua Goodman, and Geoff Hulten. “Learning at Low False Positive Rates”. In: *Proceedings of the 3rd Conference on Email and Anti-Spam*. CEAS, 2006.

[205] Xiang Zhang, Junbo Zhao, and Yann LeCun. “Character-level Convolutional Networks for Text Classification”. In: (Feb. 2016). URL: http://arxiv.org/abs/1502.01710.

[206] Kenan Zhu and Baolin Yin. “Malware behavior classification approach based on Naive Bayes”. In: *Journal of Convergence Information Technology* 7.5 (2012), pp. 203–210. ISSN: 19759320. DOI: 10.4156/jcit.vol7.issue5.25.