Quantifying uncertainty in deep learning systems

Josiah Davis, Jason Zhu, PhD, Jeremy Oldfather – AWS Professional Services

Samual MacDonald, Maciej Trzaskowski, PhD – Max Kelsen

Table of Contents

Delivering machine learning (ML) solutions to production is difficult. It's not easy to know where to start, which tools and techniques to use, and whether you're doing it right. ML professionals use different techniques based on their individual experiences, or they use prescribed tools that were developed within their company. In either case, deciding what to do, implementing the solution, and maintaining it require significant investments in time and resources. Although existing ML techniques help speed up parts of the process, integrating these techniques to deliver robust solutions requires months of work. This guide is the first part of a content series that focuses on machine learning and provides examples of how you can get started quickly. The goal of the series is to help you standardize your ML approach, make design decisions, and deliver your ML solutions efficiently. We will be publishing additional ML guides in the coming months, so please check the AWS Prescriptive Guidance website for updates.

This guide explores current techniques for quantifying and managing uncertainty in deep learning systems, to improve predictive modeling in ML solutions. This content is for data scientists, data engineers, software engineers, and data science leaders who are looking to deliver high-quality, production-ready ML solutions efficiently and at scale. The information is relevant for data scientists regardless of their cloud environment or the AWS services they are using or are planning to use.

This guide assumes familiarity with introductory concepts in probability and deep learning. For suggestions on building machine learning competency at your organization, see Deep Learning Specialization on the Coursera website or the resources on the Machine Learning: Data Scientist page on the AWS Training and Certification website.

Introduction

If success in data science is defined by the predictive performance of our models, deep learning is certainly a strong performer. This is especially true for solutions that use non-linear, high-dimensional patterns from very large datasets. However, if success is also defined by the ability to reason with uncertainty and detect failures in production, the efficacy of deep learning becomes questionable. How do we best quantify uncertainty? How do we use these uncertainties to manage risks? What are the pathologies of uncertainty that challenge the reliability, and therefore the safety, of our products? And how can we overcome such challenges?

This guide:

- Introduces the motivation for quantifying uncertainty in deep learning systems
- Explains important concepts in probability that relate to deep learning

- Demonstrates current state-of-the-art techniques for quantifying uncertainty in deep learning systems, highlighting their associated benefits and limitations
- Explores these techniques within the transfer learning setting of natural language processing (NLP)
- Provides a case study inspired by projects performed in a similar setting

As discussed in this guide, when quantifying uncertainty in deep learning, a good rule of thumb is to use temperature scaling with deep ensembles.

- Temperature scaling is an ideal tool for interpreting uncertainty estimates when data can be considered in distribution (Guo et al. 2017).
- Deep ensembles provide state-of-the-art estimates of uncertainty of when data is out of distribution (Ovadia et al. 2019).

If the memory footprint of hosting models is a concern, you can use Monte Carlo (MC) dropout in place of deep ensembles. In the case of transfer learning, consider using either MC dropout or deep ensembles with MC dropout.

Concepts in uncertainty

Uncertainty represents the reliability of our inferences. Some statistics that proxy or approximate uncertainty include the softmax probability, predictive variance $\mathbb{V}[\mathbf{y}|\mathbf{x}]$, and Shannon's entropy of the softmax vector $\mathcal{H}(\mathbf{p})$. This section introduces these statistics and explains what information about our predictions they provide. The section also outlines major pathologies of uncertainty that data scientists should be aware of.

Beyond the softmax probability

In classification, data scientists often use the softmax probability score as a notion of confidence about whether or not a predicted class (as indicated by $i \in \{1, 2, \ldots, K\}$) is correct:

$$p_i = \max(\mathbf{p}), \text{ s.t. } \mathbf{p} = \mathbf{SoftMax}(\mathbf{u})$$

The softmax probability p_i is easily accessible, which is why it has widespread adoption. However, there are some caveats data scientists should be aware of, such as deterministic overconfidence and reliability degeneration in out-of-domain contexts.

Deterministic overconfidence

Gal and Ghahramani (2016) warned against interpreting softmax probabilities as confidence scores. They empirically showed that passing a point estimate through the softmax activation function yields large probabilities, whereas passing a distribution of estimates through the softmax yields more reasonable, lower confidence scores. This

deterministic overconfidence is part due to what motivates learning a predictive distribution $p(\mathbf{y}|\mathbf{x})$, instead of a single prediction $\mathbf{y} = \mathbf{f}(\mathbf{x})$.

Formally, the deterministic overconfidence conjecture can be detailed by the following inequality:

$$\mathcal{H}(\mathbf{p}^{(\text{deterministic})}) \leq \mathcal{H}(\mathbf{p}^{(\text{Bayesian})})$$

The $\mathcal{H}(.)$ operator represents Shannon's entropy, which is larger when elements of the input vector are more similar, and is therefore largest for uniform vectors. Thus, the previous equation states that the uncertainty, in terms of Shannon's entropy $\mathcal{H}(.)$, of the expected softmax probability vector from a Bayesian model $\mathbf{p}^{(\text{Bayesian})}$ (the average of a distribution), will be larger than or equal to the softmax probability vector from a deterministic model $\mathbf{p}^{(\text{deterministic})}$ (from a model that produces a single point estimate). For a proof and demonstration of the inequality in the previous equation ,see Appendix A.

Deterministic overconfidence affects the reliability and safety of our deep learning models. Consider the case where a model confidently predicts that an item on an assembly line isn't defective, whereas, in fact, it is, resulting in the item skipping the quality review process. This faulty item might then be embedded into a larger product, compromising its integrity. At best, the end result is an inefficiency if the defect is caught down the line, or worse, a total failure of the product, if the defect isn't found. Therefore, it is critical to understand and overcome deterministic overconfidence issues for the success of our projects, and for the future of deep learning.

Three ways to improve the quality of uncertainty measurements and overcome overconfidence are:

- Calibrating softmax probabilities, post-hoc, with temperature scaling (Guo et al. 2017)
- Approximating Bayesian inference by MC dropout (that is, keeping dropout on during inference) (Gal and Ghahramani 2016)
- Approximating Bayesian inference with deep ensembles (Lakshminarayanan, Pritzel, and Blundell 2017)

Deterministic overconfidence is a theory that applies to both in-distribution and out-of-distribution data.[1] The next sections explain how to split the total quantifiable uncertainty[2]

[1] In particular, rectified linear unit (ReLU) overconfidence has recently been found to be a significant contributor to overconfidence when data is far away from the decision boundary, especially when data becomes out of distribution (Hein et al. 2019). One suggested way to become robust against ReLU overconfidence is to model the information theoretic notion of aleatoric uncertainty (Gal and Ghahramani2016, Hein et al., 2019; van Amersfoort et al., 2020), which is explained later in this guide.

[2] Some fields decompose total uncertainty into uncertainty that is quantifiable, and uncertainty that is not quantifiable. The discussion in this guide is limited to quantifiable uncertainty; therefore, the terms *total uncertainty* and *total quantifiable uncertainty* are used interchangeably.

into its two constituent components: epistemic (model) uncertainty and aleatoric (data) uncertainty (Kendall and Gal 2017).

Decomposing uncertainty

Bayesian neural networks (BNNs) yield a predictive distribution $p(\mathbf{y}|\mathbf{x})$, which provides a set of different predictions from which you can estimate variance $\mathbb{V}(.)$; that is, total predictive uncertainty $\mathbb{V}(\mathbf{y}|\mathbf{x})$. The total predictive uncertainty can be split into these two components of uncertainty by using the law of total variance:

$$\mathbb{V}(\mathbf{y}|\mathbf{x}) = \mathbb{V}(\underbrace{\mathbb{E}[\mathbf{y}|\mathbf{x},\Theta]}_{\mathbf{f}(\mathbf{x},\Theta)}) + \mathbb{E}[\underbrace{\mathbb{V}(\mathbf{y}|\mathbf{x},\Theta)}_{\mathbf{s}^2(\mathbf{x},\Theta)}]$$

The expected value $\mathbb{E}(.)$ of a target variable $\mathbf{y}$, given input $\mathbf{x}$ and random parameters Θ that specify a BNN, $\mathbb{E}[\mathbf{y}|\mathbf{x},\Theta]$, is estimated by a BNN with a single forward propagation and denoted as $\mathbf{f}(\mathbf{x},\Theta)$. The variance of the target, given input and random parameters, $\mathbb{V}(\mathbf{y}|\mathbf{x},\Theta)$, is output by the BNN, too, and denoted as $\mathbf{s}^2(\mathbf{x},\Theta)$. Thus, the total predictive uncertainty is the sum of these two numbers:

- The variance about the BNN's predicted means $\mathbb{V}(\mathbf{f}(\mathbf{x},\Theta))$ — the epistemic uncertainty
- The average of the BNN's predicted variance $\mathbb{E}[\mathbf{s}^2(\mathbf{x},\Theta)]$ — the aleatoric uncertainty

The following formula demonstrates how to calculate total uncertainty in accordance with (Kendall and Gal 2017). BNNs input $\mathbf{x}$, generate a random parameter configuration Θ, and make a single forward propagation through the neural network to output a mean $\mathbf{f}(\mathbf{x},\Theta)$ and variance $\mathbf{s}^2(\mathbf{x},\Theta)$. We denote a random generation, or simulation, by $\sim$. With fixed $\mathbf{x}$, you can reiterate this process T many times to yield a set:

$$\{\mathbf{f}(\mathbf{x},\Theta_t), \mathbf{s}^2(\mathbf{x},\Theta_t)\}_{t=1}^{T} \sim \mathrm{BNN}_{\Theta_t}(\mathbf{x}), \ \mathbf{s.t.}\ \Theta_t \sim p(\Theta|\mathcal{D}).$$

These T many samples $\{\mathbf{f}(\mathbf{x},\Theta_t), \mathbf{s}^2(\mathbf{x},\Theta_t)\}_{t=1}^{T}$ provide the necessary statistics to ascertain uncertainties. You can do this by estimating epistemic uncertainty and aleatoric uncertainty separately, and then take their sum, as shown previously in the first equation in this section.

Epistemic uncertainty

Epistemic uncertainty refers to the uncertainty of the model (*epistemology* is the study of knowledge) and is often due to a lack of training data. Examples of epistemic uncertainty include underrepresented minority groups in a facial recognition dataset or the presence of rare words in a language modeling context.

The epistemic uncertainty is found by the definition of variance:

$$\begin{aligned}\mathbb{V}(\mathbf{f}(\mathbf{x}, \Theta)) &= \mathbb{E}_{\Theta_t \sim q}[\mathbf{f}(\mathbf{x}, \Theta_t)^2] - (\mathbb{E}_{\Theta_t \sim q}[\mathbf{f}(\mathbf{x}, \Theta_t)])^2 \\ &= \frac{1}{T}\sum_{i=1}^{T}\mathbf{f}(\mathbf{x}, \Theta_t)^2 - \left(\frac{1}{T}\sum_{i=1}^{T}\mathbf{f}(\mathbf{x}, \Theta_t)\right)^2\end{aligned}$$

where $\Theta_t \sim p(\Theta|\mathcal{D})$.

Epistemic uncertainty $\mathbb{V}(\mathbf{f}(\mathbf{x}, \Theta))$ of a trained model will decrease as the size of training data increases. $\mathbb{V}(\mathbf{f}(\mathbf{x}, \Theta))$ might also be affected by the suitability of model architecture. As such, the measure of epistemic uncertainty is of great value to the machine learning engineer. This is because large measures of epistemic uncertainty might suggest that inference is being made on data that the model has less experience with. Therefore, this epistemic uncertainty might correspond to erroneous predictions or outlier data.

Aleatoric uncertainty

Aleatoric uncertainty refers to the data's inherent randomness that cannot be explained away (*aleator* refers to someone who rolls the dice in Latin). Examples of data with aleatoric uncertainty include noisy telemetry data and low-resolution images or social media text. You can assume the aleatoric uncertainty $\mathbb{E}[\mathbf{s}^2(\mathbf{x}, \Theta)]$, the inherent randomness, to be either constant (*homoscedastic*) or variable (*heteroscedastic*), as a function of the input explanatory variables.

Homoscedastic aleatoric uncertainty

Homoscedastic aleatoric uncertainty, when $\mathbb{E}[\mathbf{s}^2(\mathbf{x}, \Theta)]$ is constant, is the simplest case and commonly encountered in regression under the modeling assumption that $y = f(x) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma\mathbf{I})$, where $\mathbf{I}$ is the identity matrix and σ is a constant scalar. It is highly restrictive to assume constant aleatoric risk—to assume that noise ϵ about a response y is independent from the explanatory variable x and constant—and rarely reflective of reality. Many phenomena in nature do not exhibit constant randomness. For example, uncertainty about outcomes in physical systems, such as fluid motion, are usually a function of kinetic energy. Consider the contrast between the turbulent water flow of a large waterfall and the laminar water flow of a decorative fountain. The stochasticity (randomness) of a water particle's trajectory is a function of the kinetic energy and therefore not constant. This assumption can lead to loss of valuable information when modeling relationships between targets and inputs that host variable noise, and cannot be explained with the observable information. As a consequence, in most cases, it is not sufficient to assume homoscedastic uncertainty. Unless the phenomena is known to be homoscedastic in nature, the inherent noise should be modeled as a function of the explanatory variables $\mathbf{x}$, if it can be done so.

Heteroscedastic aleatoric uncertainty

Heteroscedastic aleatoric uncertainty is when we consider the inherent randomness within data to be a function of the data itself $\mathbf{s}^2(\mathbf{x})$. To calculate this type of uncertainty, you average a sample set of the predictive variance:

$$\mathbf{s}^2(\mathbf{x}) = \mathbb{E}_{\Theta \sim q(\Theta|\mathcal{D})}[\mathbf{s}^2(\mathbf{x}, \Theta)] = \frac{1}{T}\sum_{t=1}^{T} \mathbf{s}^2(\mathbf{x}, \Theta_t)$$

with $\mathbf{s}^2(\mathbf{x}, \Theta)$ being estimated by a BNN. Learning aleatoric uncertainty during training encourages BNNs to encapsulate the inherent randomness within the data that can't be explained away. If there is no inherent randomness, $\mathbf{s}^2(\mathbf{x})$ should tend toward zero.

An information theoretic approach to uncertainty

The explanation of uncertainty in the previous section relies only on the variance notion of uncertainty, but information theoretic notions of uncertainty exist, too. Incorporating information theoretic aleatoric uncertainty improves robustness of the total uncertainty estimate (Gal 2016; Hein, Andriushchenko, and Bitterwolf 2019, van Amersfoort et al. 2020). Total uncertainty is measured by Shannon's entropy:

$$\mathcal{H}(\mathbf{p}) = -\sum_{i=1}^{K} p_i \log(p_i) = -\langle \mathbf{p}, \log(\mathbf{p}) \rangle$$

where $\langle ., . \rangle$ is the dot product operator and $\mathbf{K}$ is the number of classes.

The predictive entropy $\mathcal{H}(\mathbf{p})$ is available to both Bayesian and non-Bayesian neural networks. In order to decompose this total uncertainty into the epistemic and aleatoric components, you must estimate the mutual information $\mathcal{MI}(\mathbf{p}, \Theta)$, and this requires a Bayesian approach.

$$\underbrace{\mathcal{MI}(\mathbf{p}, \Theta)}_{\text{epistemic}} = \mathcal{H}(\mathbf{p}) - \underbrace{\sum_{t=1}^{T} \langle \mathbf{p}|\Theta_t, \log(\mathbf{p}|\Theta_t) \rangle}_{\text{aleatoric}}$$

Reliability degeneration in out-of-distribution contexts

Practitioners of deep learning often assume that test data and training data share the same distribution. Unfortunately, this assumption doesn't always hold in practice. The world evolves, and data generated from the future is often out-of-distribution (ood). Consequently, as context changes, the in-distribution assumption becomes less realistic, and so does the reliability of our predictions and uncertainties (Fort, Hu, and Lakshminarayanan, 2019; Nalisnick et al., 2019; Ovadia et al., 2019). In fact, predictive performance can decrease while measures of confidence increase, which causes a silent failure.

Methods for estimating uncertainty in deep learning

This section discusses three methods for quantifying uncertainty in deep learning networks, and provides general recommendations for using each method:

- Temperature scaling
- Monte Carlo dropout
- Deep ensembles

Temperature scaling

In classification problems, the predicted probabilities (softmax output) are assumed to represent the true correctness probability for the predicted class. However, although this assumption might have been reasonable for models a decade ago, it isn't true for today's modern neural network models (Guo et al. 2017). The loss of connection between model predicting probabilities and the confidence of model predictions would prevent the application of modern neural network models into real-world problems, as in decision-making systems. Precisely knowing the confidence score of model predictions is one of the most critical risk control settings required for building robust and trustworthy machine learning applications.

Modern neural network models tend to have large architectures with millions of learning parameters. The distribution of predicting probabilities in such models is often highly skewed to either 1 or 0, meaning that the model is overconfident and the absolute value of these probabilities could be meaningless. (This issue is independent of whether class imbalance is present in the dataset.) Various calibration methods for creating a prediction confidence score have been developed in the past ten years through post-processing steps to recalibrate the naïve probabilities of the model. This section describes one calibration method called *temperature scaling*, which is a simple yet effective technique for recalibrating prediction probabilities (Guo et al. 2017). Temperature scaling is a single-parameter version of Platt Logistic Scaling (Platt 1999).

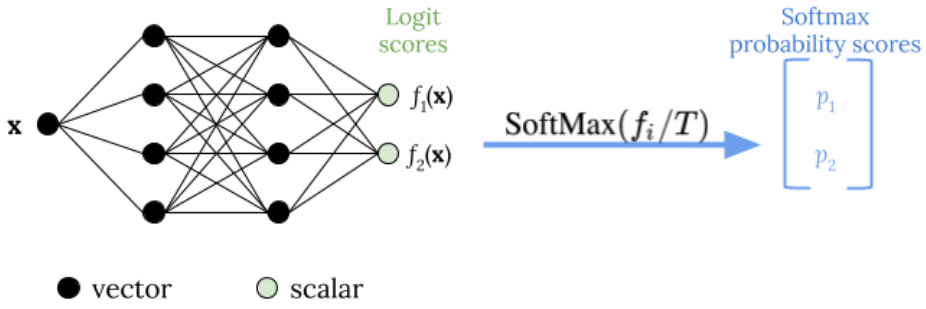
Temperature scaling uses a single scalar parameter $T > 0$, where T is the temperature, to rescale logit scores before applying the softmax function, as shown in the following figure. Because the same T is used for all classes, the softmax output with scaling has a monotonic relationship with unscaled output. When $T = 1$, you recover the original probability with the default softmax function. In overconfident models where $T > 1$, the recalibrated probabilities have a lower value than the original probabilities, and they are more evenly distributed between 0 and 1.

The method to get an optimal temperature T for a trained model is through minimizing the negative log likelihood for a held-out validation dataset.

$$\mathcal{L} = -\sum_{i=1}^{n} \log(\mathbf{f}(\mathbf{x}_i))$$

We recommend that you integrate the temperature scaling method as a part of the model training process: After a model training is completed, extract the temperature value T by using the validation dataset, and then rescale logit values by using T in the softmax function. Based on experiments in text classification tasks using BERT-based models, the temperature T usually scales between 1.5 and 3.

The following figure illustrates the temperature scaling method, which applies temperature value T before passing the logit score to the softmax function.



The calibrated probabilities by temperature scaling can approximately represent the confidence score of model predictions. This can be evaluated quantitatively by creating a reliability diagram (Guo et al. 2017), which represents the alignment between the distribution of expected accuracy and the distribution of predicting probabilities.

Temperature scaling has also been evaluated as an effective way to quantify total predictive uncertainty in the calibrated probabilities, but it is not robust in capturing epistemic uncertainty in scenarios like data drifts (Ovadia et al. 2019). Considering the ease of implementation, we recommend that you apply temperature scaling to your deep learning model output to build a robust solution for quantifying predictive uncertainties.

Monte Carlo dropout

One of the most popular ways to estimate uncertainty is by inferring predictive distributions with Bayesian neural networks. To denote a predictive distribution, use:

$$p(\mathbf{y}|\mathbf{x}, \mathcal{D}),$$

with target $\mathbf{y}$, input $\mathbf{x}$, and N many training examples $\mathcal{D} = (\mathbf{x}_i, \mathbf{y}_i)_{i=1}^{N}$. When you obtain a predictive distribution, you can inspect the variance and uncover uncertainty. One way to learn a predictive distribution requires learning a distribution over functions, or, equivalently, a distribution over the parameters (that is, the parametric posterior distribution $p(\Theta|\mathcal{D})$).

The Monte Carlo (MC) dropout technique (Gal and Ghahramani, 2016) provides a scalable way to learn a predictive distribution. MC dropout works by randomly switching off neurons in a neural network, which regularizes the network. Each dropout configuration corresponds to a different sample from the approximate parametric posterior distribution $q(\Theta|\mathcal{D})$:

$$\Theta_t \sim q(\Theta|\mathcal{D}),$$

where Θ_t corresponds to a dropout configuration, or, equivalently, a simulation ~, sampled from the approximate parametric posterior $q(\Theta|\mathcal{D})$, as shown in the following figure. Sampling from the approximate posterior $q(\Theta|\mathcal{D})$ enables Monte Carlo integration of the model's likelihood, which uncovers the predictive distribution, as follows:

$$p(\mathbf{y}|\mathbf{x}) \overset{\text{VI}}{\approx} \int_{\Omega} \underbrace{p(\mathbf{y}|\mathbf{x},\Theta)}_{\text{likelihood}} \underbrace{q(\Theta|\mathcal{D})}_{\text{p. posterior}} d\Theta$$
$$\overset{\text{MC}}{\approx} \frac{1}{T}\sum_{t=1}^{T} p(\mathbf{y}|\mathbf{x},\Theta_t), \text{ s.t. } \Theta_t \sim q(\Theta|\mathcal{D})$$

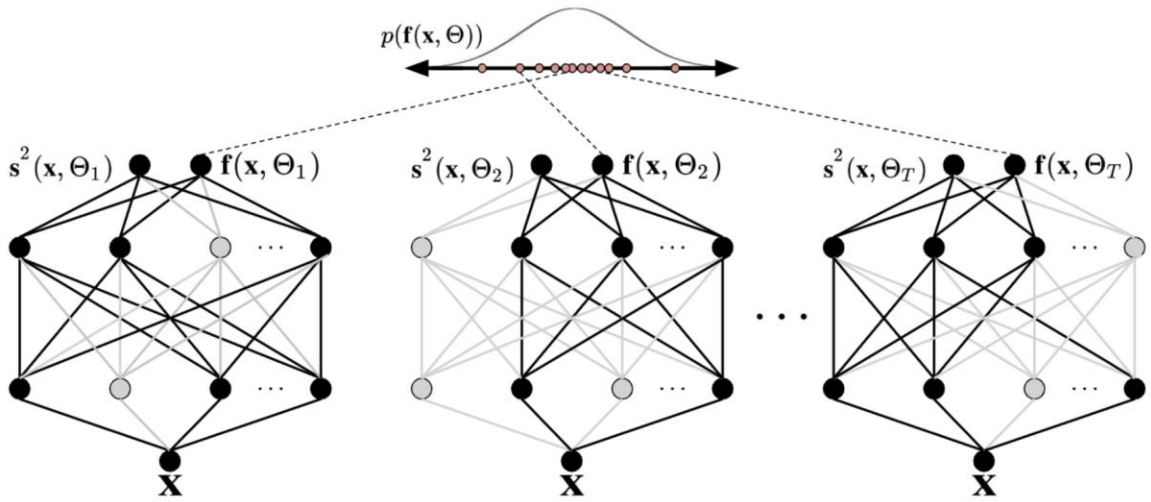
For simplicity, the likelihood may be assumed to be Gaussian distributed:

$$p(\mathbf{y}|\mathbf{x},\Theta) = \mathcal{N}(\mathbf{f}(\mathbf{x},\Theta), \mathbf{s}^2(\mathbf{x},\Theta)),$$

with the Gaussian function $\mathcal{N}$ specified by the mean $\mathbf{f}(\mathbf{x},\Theta)$ and variance $\mathbf{s}^2(\mathbf{x},\Theta)$ parameters, which are output by simulations from the Monte Carlo dropout BNN:

$$\mathbf{f}(\mathbf{x},\Theta), \mathbf{s}^2(\mathbf{x},\Theta) \sim \text{MonteCarloDropout}(\mathbf{x}).$$

The following figure illustrates MC dropout. Each dropout configuration yields a different output by randomly switching neurons off (grey circles) and on (black circles) with each forward propagation. Multiple forward passes with different dropout configurations yield a predictive distribution over the mean *p*(f(**x**, ø)).



The number of forward passes through the data should be evaluated quantitatively, but 30-100 is an appropriate range to consider (Gal and Ghahramani 2016).

Deep ensembles

The core idea behind ensembling is that by having a committee of models, different strengths will complement one another, and many weaknesses will cancel each other out. This is the guiding intuition behind 18th century French mathematician Nicolas de Condorcet's famous jury theorem (Estlund 1994): If each juror has a probability that's greater than 50% of arriving at the true verdict, and if the jurors make independent decisions, the probability of a correct group verdict increases to 100% as the number of jurors increases.

Moving to recent history, the process of ensembling ML models includes two steps: training different models and combining the predictions. You can obtain different models by using different feature subsets, training data, training regimes, and model architectures. You can combine predictions by averaging them, training a new model on top of the predictions (*model stacking*), or using custom voting rules that you can tune to a specific context (see the case study for one such example). Two of the initial ensembling techniques for machine learning are *boosting* (Freund and Schapire 1996) and *random forests* (Breiman 2001). These are two complementary approaches.

The idea behind boosting is to sequentially train weak learners. Each subsequent model focuses on a subset of the data and is boosted by the errors previously observed during training. In this way each sequential tree is trained on a new training set that was previously unseen. At the end of training, predictions are averaged across the weak learners.

The idea behind random forests is training multiple decision tree models without pruning, on bootstrapped samples of the data and by selecting random feature subsets. Breiman

showed that the generalization error has an upper bound that is a function of the number and decorrelation of the individual trees.

In deep learning, dropout is designed as a regularization technique and can also be interpreted as an ensemble of multiple models (Srivastava et al. 2014). The realization that dropout could be used to effectively quantify uncertainty (Gal and Ghahramani 2016) motivated a further exploration of ensembles in deep learning models for the same purpose. Deep ensembles have been shown to outperform MC dropout in quantifying uncertainty in a variety of datasets and tasks in regression and classification (Lakshminarayanan., Pritzel, and Blundell 2017). Additionally, deep ensembles have been shown to be state-of-the-art in out-of-distribution settings (such as perturbations of the data or the introduction of new classes unseen during training). They outperform MC dropout and other methods (Ovadia et al. 2019). The reason why deep ensembles perform so well in out-of-distribution settings is that their weight values and loss trajectories are very different from one another, and, as a result, they lead to diverse predictions (Fort, Hu, and Lakshminarayanan 2019).

Neural networks often have hundreds of millions more parameters than training data points. This means that they include a large space of possible functions that might sufficiently approximate the data generating function. Consequently, there are many low-loss valleys and regions that all correspond to good, but different, functions. Viewed from a Bayesian perspective (Wilson and Izmailov 2020), these candidate functions correspond to different hypotheses that identify the true underlying function. As such, the more candidate functions you ensemble, the more likely you are to represent the truth, and therefore achieve a robust model that shows low confidence when you extend inference out of distribution. Ensembles essentially settle in many distant low-loss valleys, yielding *a distribution of diverse functions* (Fort,Hu, and Lakshminarayanan 2019). On the other hand, alternative methods such as MC dropout and alternative Bayesian approaches will hone in to just one valley, yielding *a distribution of similar functions*. Therefore, just a few independently trained neural networks from the ensemble—(Lakshminarayanan, Pritzel, and Blundell 2017) and (Ovadia et al. 2019) suggest that five models are sufficient—will more accurately recover the true marginal likelihood (predictive distribution), when compared with sampling around a single low-loss region, which will host a lot of redundancy (because functions will all be similar).

In summary, to improve your accuracy and to maximize the reliability of your uncertainties, ensemble your models.

Quantitative comparison of uncertainty methods

This section describes how we compared the methods for estimating uncertainty by using the Corpus of Linguistic Acceptability (CoLA) (Warstadt, Singh, and Bowman 2019) dataset. The CoLA dataset consists of a collection of sentences along with a binary indicator of whether they are acceptable. Sentences can be labeled as unacceptable for a variety of reasons, including improper syntax, semantics, or morphology. These sentences are taken from examples in linguistic publications. There are two validation sets. One validation set is

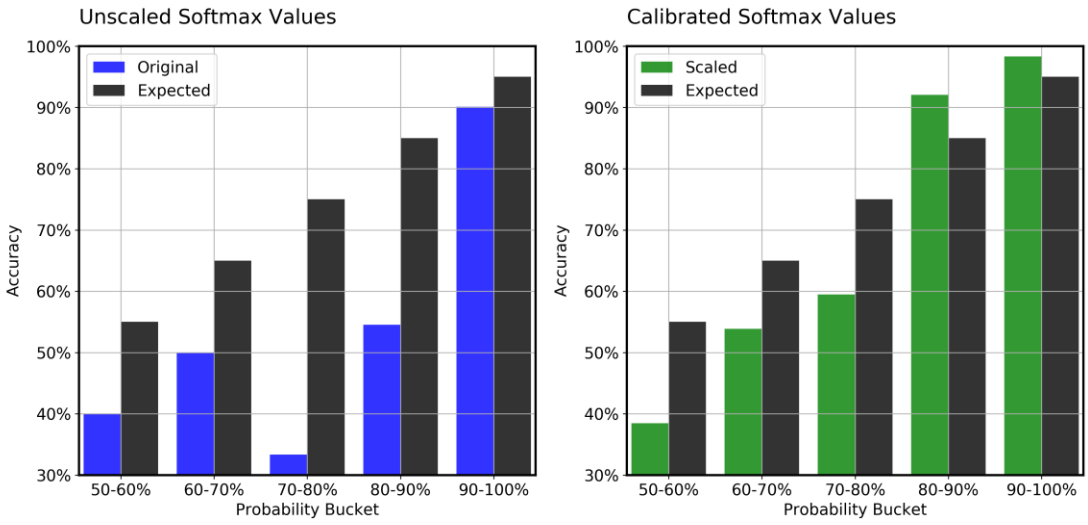
taken from the same sources used in forming the training dataset (in domain), and the other validation set is taken from sources that aren't contained in the training set (out of domain). The following table summarizes this information.

Dataset	Total size	Positive	Negative
Training	8551	6023	2528
Validation (in domain)	527	363	164
Validation (out of domain)	516	354	162

The comparison uses a RoBERTa (Liu et al., 2019) base architecture with pretrained weights and a randomly initialized head with a single hidden layer. Hyperparameters are mostly suggested in the RoBERTa paper with a few minor modifications.

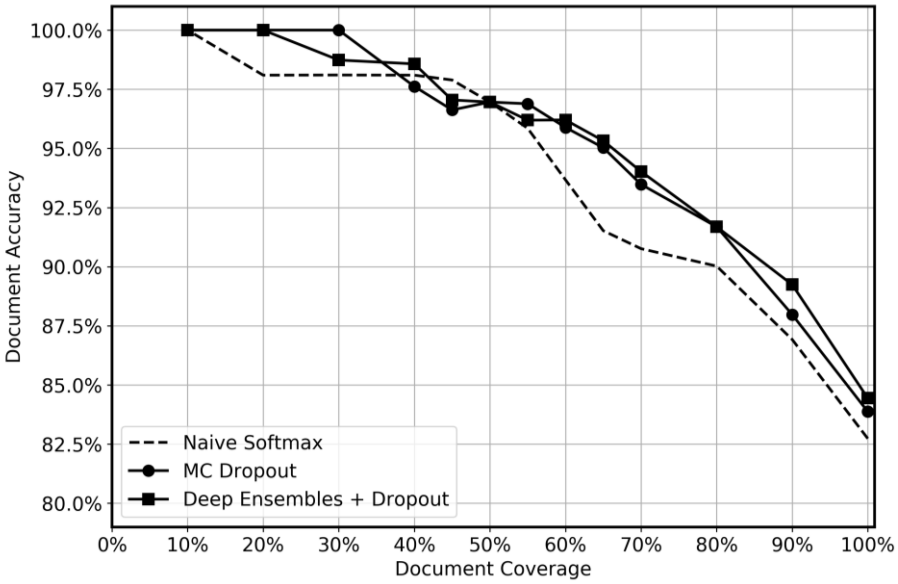
Temperature scaling

We took the average of five different values, obtaining a value of $T = 2.62$ across different training seeds. The following charts show calibration before and after temperature scaling. As the first chart shows, unscaled softmax values revealed major discrepancies. For example, the 70-80% confidence bucket contains predictions that are less than 50% accurate. After scaling, the calibration improves substantially. For example, the 70-80% bucket corresponds to 72% accuracy. Consequently, we used the temperature-scaled values for subsequent experiments.



Document coverage and accuracy — in domain

We compared the predictive performance of deep ensembles with dropout applied at test time, MC dropout, and a naïve softmax function, as shown in the following graph. After inference, predictions with the highest uncertainties were dropped at different levels, yielding remaining data coverage that ranged from 10% to 100%. We expected the deep ensemble to more efficiently identify uncertain predictions due to its greater ability to quantify epistemic uncertainty; that is, to identify regions in the data where the model has less experience. This should be reflected in higher accuracy for different data coverage levels. For each deep ensemble, we used 5 models and applied inference 20 times. For MC dropout, we applied inference 100 times for each model. We used the same set of hyperparameters and model architecture for each method.

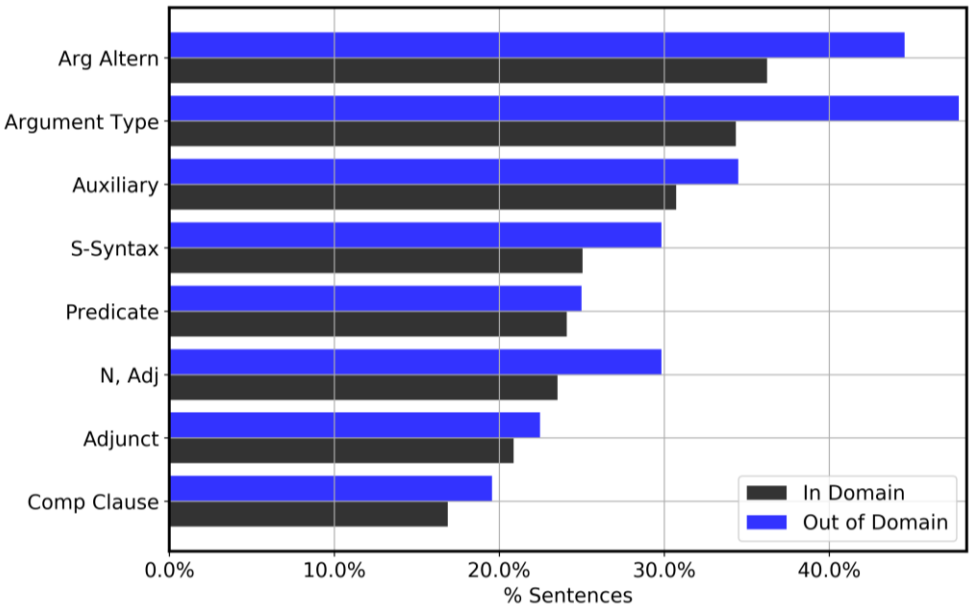


The graph appears to show a slight benefit to using deep ensembles and MC dropout compared with naïve softmax. This is most notable in the 50-80% data coverage range. Why is this not greater? As mentioned in the deep ensembles section, the strength of deep ensembles comes from the different loss trajectories taken. In this situation, we are using pretrained models. Although we fine-tune the entire model, the overwhelming majority of the weights are initialized from the pretrained model, and only a few hidden layers are randomly initialized. Consequently, we conjecture that pretraining of large models can cause an overconfidence due to little diversification. To our knowledge, the efficacy of deep ensembles has not been previously tested in transfer learning scenarios, and we see this as an exciting area for future research.

Document coverage and accuracy — out of domain

We also examined out-of-domain data, which was taken from syntax textbooks that weren't used to source the training data. However, we didn't observe a noticeable difference in relative performance. This is perhaps because the quantitative content of linguistic features would likely differ very little, although sentences are sourced from different textbooks.

The following chart provides a comparison of the most frequent linguistic features across the two data sources. It shows very little difference between the distributions of the in-domain and out-of-domain datasets. Furthermore, with respect to vocabulary, the model had at least some exposure with out-of-domain language during training on in-domain examples. All words found in the out-of-domain set had a frequency count of at least 100 over the entire training set (Warstadt, Singh, and Bowman 2019). Thus, the out-of-domain data wasn't considered as truly out of distribution. For more information on the linguistic features, see Warstadt, Singh, and Bowman (2019).

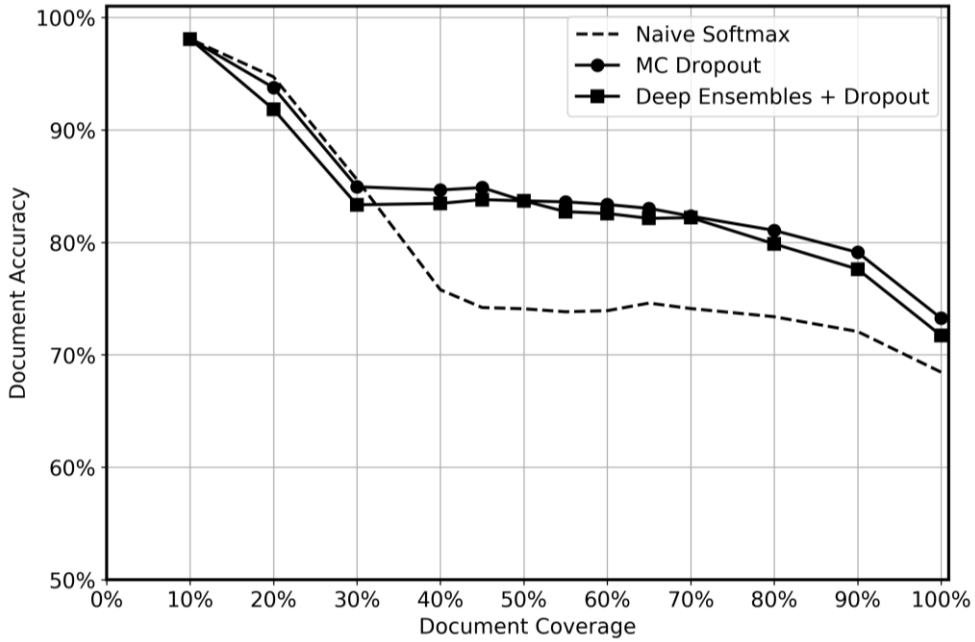


To gain a better sense of the deep ensemble's and Monte Carlo dropout's ability to estimate uncertainty in a truly out-of-distribution setting, we created three adversarial datasets that included random words injected into the sentences:

- One dataset with rare words not found in the training dataset
- One dataset with non-English words not found in the training dataset
- One dataset with a mixture of the previous two datasets

All of the injected words were present in the original vocabulary used for pretraining the model.

The following graph shows the correlation between accuracy and coverage for the third dataset. The first and second datasets show similar patterns.



The graph shows a clear benefit from using either MC dropout or deep ensembles with MC dropout for coverage levels above 40%. We suspect that these two methods show similar performance because the pretraining model doesn't include much diversification. This opens the way for further investigations. The significant performance degradation for the naïve softmax method that occurs above 40% document coverage is likely because we altered approximately 55% of the validation set with our adversarial data generation process. In the low coverage region, the methods have similar accuracy values, because these sets of data aren't out of distribution.

Case study

This section examines a real-world business scenario and application for quantifying uncertainty in deep learning systems. Suppose you want a machine learning model to automatically judge whether a sentence is grammatically unacceptable (negative case) or acceptable (positive case). Consider the following business process: If the model flags a sentence as grammatically acceptable (positive), you process it automatically, without human review. If the model flags the sentence as unacceptable (negative), you pass the sentence to a human for review and correction. The case study uses deep ensembles along with temperature scaling.

This scenario has two business objectives:

- **High recall for negative cases**. We want to catch all sentences that have grammatical errors.
- **Reduction of the manual workload**. We want to auto-process cases that have no grammatical errors as much as possible.

Baseline results

When applying a single model to the data with no dropout at test time, these are the results:

- For positive sample: recall=94%, precision=82%
- For negative sample: recall=52%, precision=79%

The model has much lower performance for negative samples. However, for business applications, recall for negative samples should be the most important metric.

Application of deep ensembles

To quantify model uncertainty, we used the standard deviations of individual model predictions across deep ensembles. Our hypothesis is that for false positives (FP) and false negatives (FN) we expect to see the uncertainty to be much higher than for true positives (TP) and true negatives (TN). Specifically, the model should have high confidence when it is correct and low confidence when it is wrong, so we can use uncertainty to tell when to trust the model's output.

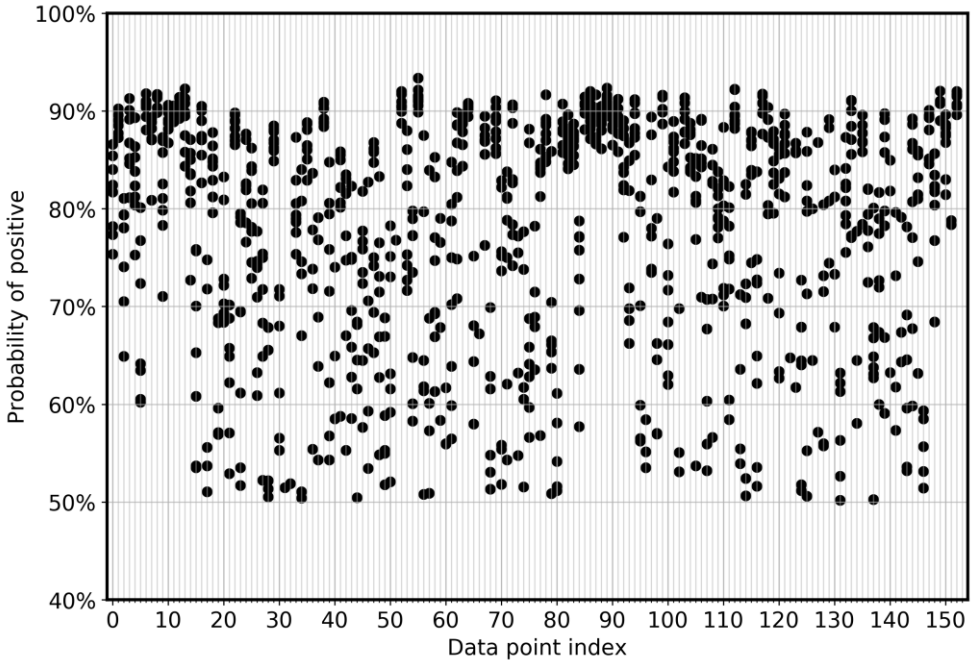
The following confusion matrix shows the uncertainty distribution across FN, FP, TN, and TP data. The probability of negative standard deviation is the standard deviation of the probability of negatives across models. The median, mean, and standard deviations are aggregated across the dataset.

Label	Probability of negative standard deviation		
	Median	Mean	Standard deviation
FN	0.061	0.060	0.027
FP	**0.063**	**0.062**	0.040
TN	0.039	0.045	0.026
TP	**0.009**	**0.020**	0.025

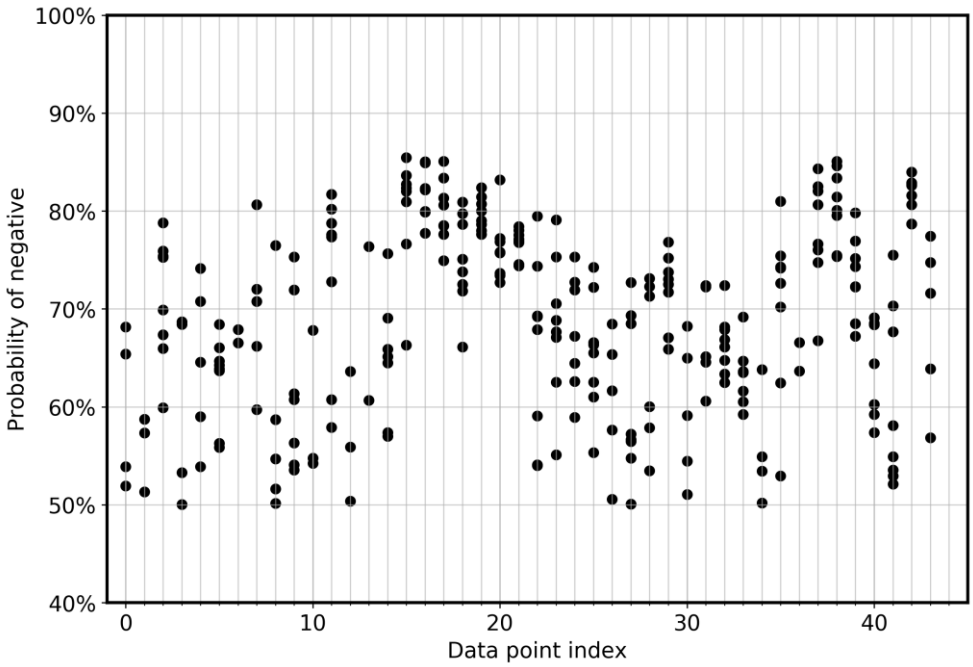
As the matrix shows, the model performed the best for TP, so that has the lowest uncertainty. The model performed the worst for FP, so that has the highest uncertainty, which is in line with our hypothesis.

To directly visualize the model's deviation among ensembles, the following graph plots probability in a scatter view for FN and FP for the CoLA data. Each vertical line is for one specific input sample. The graph shows eight ensemble model views. That is, each vertical line has eight data points. These points either perfectly overlap or are distributed in a range.

The first graph shows that for the FPs, the probability of being positive distributes between 0.5 and 0.925 across all eight models in the ensemble.



Similarly, the next graph shows that for the FNs, the probability of being negative distributes between 0.5 and 0.85 among the eight models in the ensemble.

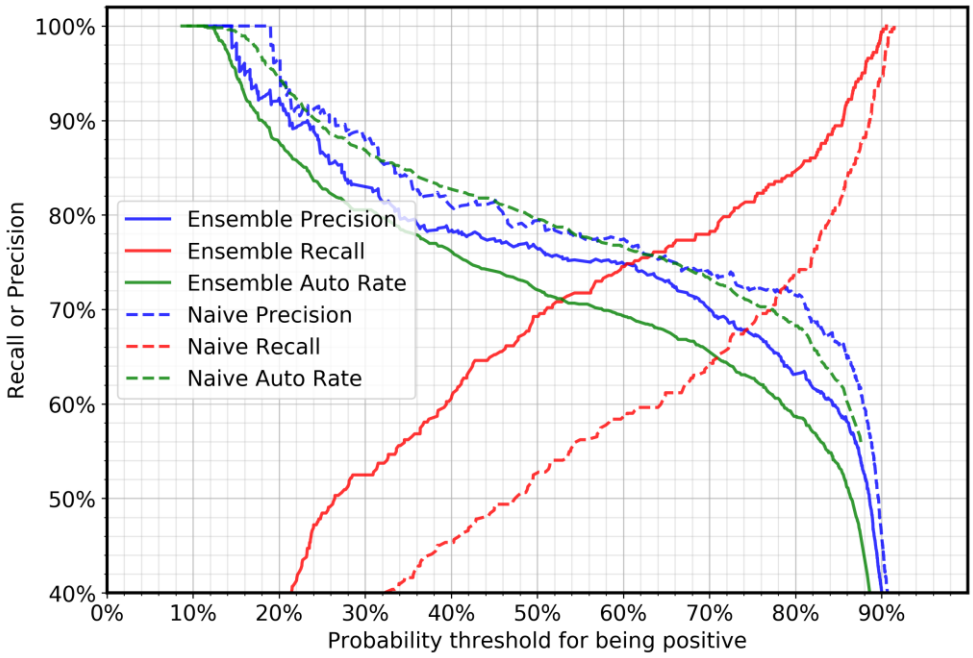


Defining a decision rule

To maximize the benefit of the results, we use the following ensemble rule: For each input, we take the model that has the lowest probability of being positive (acceptable) to make flagging decisions. If the selected probability is larger than, or equal to, the threshold value, we flag the case as acceptable and auto-process it. Otherwise, we send the case for human review. This is a conservative decision rule that is appropriate in highly regulated environments.

Evaluating the results

The following graph shows the precision, recall, and auto (automation) rate for the negative cases (cases with grammatical errors). The automation rate refers to the percentage of cases that will be automatically processed because the model flags the sentence as acceptable. A perfect model with 100% recall and precision would achieve a 69% (positive cases/total cases) automation rate, because only positive cases will be automatically processed.



The comparison between the deep ensemble and naïve cases shows that, for the same threshold setting, recall increases quite drastically and precision decreases slightly. (The automation rate depends on the positive and negative sample ratio in the test dataset.) For example:

- Using a threshold value of 0.5:
  - With a single model, the recall for negative cases will be 52%.
  - With the deep ensemble approach, the recall value will be 69%.

- Using a threshold value of 0.88:
  - With a single model, the recall for negative cases will be 87%.
  - With the deep ensemble approach, the recall value will be 94%.

You can see that deep ensemble can boost certain metrics (in our case, the recall of negative cases) for business applications, without a requirement to increase the size of the training data, its quality, or a change in the model's method.

Conclusion

This guide provided a conceptual overview of uncertainty in deep learning systems. It described experiments that extend the existing literature to cover the transfer learning scenario for natural language processing (NLP) in both in-distribution and out-of-distribution settings. Finally, it provided a case study that serves as a roadmap for how data scientists can apply these concepts in their work in a highly regulated industry.

When quantifying uncertainty in deep learning networks, our general recommendation is to use temperature scaling with deep ensembles. Temperature scaling provides interpretable uncertainty estimates when incoming data is in distribution. Therefore, temperature scaling addresses the total uncertainty by adjusting the softmax uncertainties so that they are not so overconfident. Temperature scaling should be performed on the validation dataset, after the model has been trained on the validation dataset.

Deep ensembles currently provide state-of-the-art estimates of uncertainty when data is out of distribution. They provide higher epistemic uncertainty estimates when presented with data that's different from the training data. This is due to the strength in diversity of the underlying models that comprise the deep ensemble. We suggest that five models will suffice in most situations.

In two scenarios, we recommend that you consider MC dropout as an alternative to deep ensembles: when hosting multiple models is a concern due to additional load to the infrastructure, and in transfer learning (that is, when using pretrained weights). When the hosting requirements for multiple models is a concern, MC dropout is a valid alternative to deep ensembles. If you're using MC dropout as a replacement for deep ensembles, you should be prepared to sacrifice some computational latency for the sake of more iterations through the data. We recommend 30-100 iterations as an appropriate range. In transfer learning, there will be less diversification among the ensembled base learners (that is, the underlying model weights will be more similar to one another). This is why total predictive uncertainty can be low in transfer learning, especially in settings with out-of-distribution data. As a result, in the transfer learning situation, consider supplementing or replacing deep ensembles with MC dropout.

References

Breiman, L. 2001. "Random Forests." *Machine Learning*. https://doi.org/10.1023/A:1010933404324.

Estlund, D. M. 1994. "Opinion Leaders, Independence, and Condorcet's Jury Theorem." *Theory and Decision*. https://doi.org/10.1007/BF01079210.

Fort, S., H. Hu, and B. Lakshminarayanan. 2019. "Deep Ensembles: A Loss Landscape Perspective." 2, 1–14. https://arxiv.org/abs/1912.02757.

Freund, Y. and R.E. Schapire. 1996. "Experiments with a New Boosting Algorithm." Proceedings of the *13th International Conference on Machine Learning*. https://dl.acm.org/doi/10.5555/3091696.3091715.

Gal, Y. 2016. "Uncertainty in Deep Learning." Department of Engineering. University of Cambridge.

Gal, Y., and Z. Ghahramani. 2016. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning." *33rd International Conference on Machine Learning (ICML 2016)*. https://arxiv.org/abs/1506.02142.

Guo, C., G. Pleiss, Y. Sun, and K.Q. Weinberger. 2017. "On Calibration of Modern Neural Networks." *34th International Conference on Machine Learning (ICML 2017)*. https://arxiv.org/abs/1706.04599.

Hein, M., M. Andriushchenko, and J. Bitterwolf. 2019. "Why ReLU Networks Yield High-Confidence Predictions Far Away From the Training Data and How to Mitigate the Problem." 2019. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (June 2019): 41–50. https://doi.org/10.1109/CVPR.2019.00013.

Kendall, A. and Y. Gal. 2017. "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" *Advances in Neural Information Processing Systems*. https://papers.nips.cc/paper/7141-what-uncertainties-do-we-need-in-bayesian-deep-learning-for-computer-vision.

Lakshminarayanan, B., A. Pritzel, and C. Blundell. 2017. "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles." *Advances in Neural Information Processing Systems*. https://arxiv.org/abs/1612.01474.

Liu, Y., M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. 2019. "RoBERTa: A Robustly Optimized BERT Pretraining Approach." https://arxiv.org/abs/1907.11692.

Nado, Z., S. Padhy, D. Sculley, A. D'Amour, B. Lakshminarayanan, and J. Snoek. 2020. "Evaluating Prediction-Time Batch Normalization for Robustness under Covariate Shift." https://arxiv.org/abs/2006.10963.

Nalisnick, E., A. Matsukawa, Y.W. Teh, D. Gorur, and B. Lakshminarayanan. 2019. "Do Deep Generative Models Know What They Don't Know?" *7th International Conference on Learning Representations (ICLR 2019)*. https://arxiv.org/abs/1810.09136.

Ovadia, Y., E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J.V. Dillon, B. Lakshminarayanan, and J. Snoek. 2019. "Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty Under Dataset Shift." *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*. https://arxiv.org/abs/1906.02530.

Platt, J., and others. 1999. "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods." *Advances in Large Margin Classifiers*. http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.1639.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. 2014. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal of*

Machine Learning Research.
https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf.

van Amersfoort, J., L. Smith, Y.W. Teh, and Y. Gal. 2020. "Uncertainty Estimation Using a Single Deep Deterministic Neural Network." *International Conference for Machine Learning*. https://arxiv.org/abs/2003.02037.

Warstadt, A., A. Singh, and S.R. Bowman. 2019. "Neural Network Acceptability Judgments." *Transactions of the Association for Computational Linguistics*. https://doi.org/10.1162/tacl_a_00290.

Wilson, A. G., and P. Izmailov. 2020. "Bayesian Deep Learning and a Probabilistic Perspective of Generalization." https://arxiv.org/abs/2002.08791.

Acknowledgments

The authors acknowledge the following people for their valuable feedback and support:

- Eden Duthie, AWS Professional Services, Practice Manager
- Verdi March, PhD, AWS Professional Services, Senior Data Scientist
- Yin Song, PhD, AWS Professional Services, Data Scientist
- Liam Hodgkinson, PhD, University of California, Berkeley, Postdoctoral Researcher
- Fred Roosta, PhD, University of Queensland, Australia, ARC DECRA Fellow
- Kaiah Stevens, Max Kelsen, Machine Learning Researcher
- Nick Therkelsen, Max Kelsen, CEO

Appendix A. Proof of deterministic overconfidence (binary classification)

Consider the softmax (logistic) function binary classification:

$$\text{SoftMax}(u) = (1 + \exp(-u))^{-1}$$

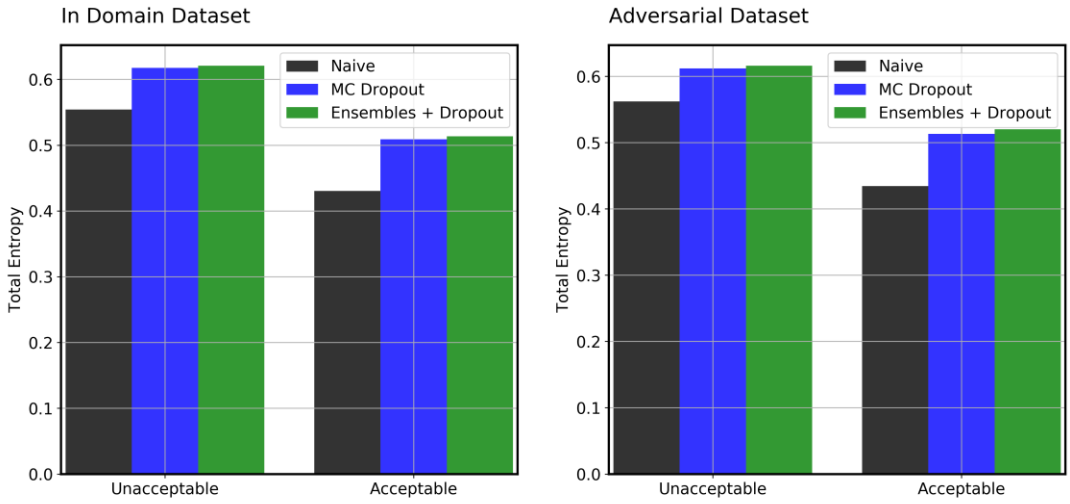
The softmax function is concave for $u \in \mathbb{R}^*_+$ and convex for $u \in \mathbb{R}^*_-$. Therefore, by Jensen's inequality:

$$\begin{aligned} \text{SoftMax}(\mathbb{E}_\Theta[u|\Theta]) &\geq \mathbb{E}_\Theta[\text{SoftMax}(u|\Theta)], \text{ if } u|\Theta \in \mathbb{R}^*_+ \\ \underbrace{\text{SoftMax}(\mathbb{E}_\Theta[u|\Theta])}_{p^{(D)}} &\leq \underbrace{\mathbb{E}_\Theta[\text{SoftMax}(u|\Theta)]}_{p^{(B)}}, \text{ if } u|\Theta \in \mathbb{R}^*_-. \end{aligned}$$

This implies $\mathcal{H}(p^{(D)}) \leq \mathcal{H}(p^{(B)})$ for all $u \in \mathbb{R}$. Equality is obtained when $p^{(D)} = p^{(B)} = 0.5$.

Appendix B. Empirical demonstration of deterministic overconfidence

To empirically support the theoretical evidence in Appendix A for deterministic overconfidence, we compared the total entropy from each modeling technique. We observed that there was higher total entropy for MC dropout and deep ensembles with MC dropout, when compared with the deterministic case. This holds for both acceptable and unacceptable sentences. Furthermore, it holds true for the dataset that was generated using adversarial techniques. The following charts show the total entropy comparison.



Appendix C. Other considerations and notable methods

This guide addresses the most practical and effective ways to ascertain reliable measures of uncertainty. It also addresses some of the major pathologies such as out-of-distribution degeneration and deterministic overconfidence. Other recent techniques include deterministic uncertainty quantification (DUQ) (van Amersfoort et al. 2020) and prediction-time batch normalization (Nado et al. 2020).

DUQs are a new kind of deep learning classifier that do not utilize the traditional softmax function. Instead, DUQs provide reliable uncertainty for out-of-distribution data. DUQs output a vector, f(x), which is transformed by a class-specific weight matrix, W_c, for mapping to a feature vector. The distance between this feature vector and learnt centroids (one centroid for each class) represents the corresponding uncertainties. The distance to the closest centroid is deemed the predictive uncertainty. Feature vectors are able to map far from centroids for out-of-distribution data by regularizing the model's smoothness. The novel regularization method tunes smoothness so that changes of output coincide with changes in input, without changing so much that it compromises generalization. DUQs are a promising new way for modeling uncertainty and provide an alternative to deep ensembles for reliable uncertainty in out-of-distribution settings. For details, see the publications in the References section.

Another method worth noting is prediction-time batch normalization for out-of-distribution robustness (Nado et al. 2020). This technique requires just a few lines of code to implement and claims to improve uncertainty reliability with out-of-distribution data in a way that is complementary to deep ensembles. An interesting caveat to this method is that the quality of uncertainty actually degenerates for pretraining settings, which raises questions for future work.

Document history

The following table describes significant changes to this content. If you want to be notified about future updates, you can subscribe to an RSS feed from the top navigation bar on this page.

Change	Description	Date
—	Initial publication	August 24, 2020