Uncertainty: a Tutorial

Eric Jang

December 22, 2018

Notions of **uncertainty** are tossed around in casual discussion of AI safety, risk management, portfolio optimization, scientific measurement, and insurance. Here are a few examples of colloquial use:

- "We want machine learning models to know what they don't know."
- "An AI responsible for diagnosing patients and prescribing treatments should tell us how confident it is about its recommendations."
- "Significant figures in scientific calculations represent uncertainty in measurements."
- "We want autonomous agents to explore areas where they are uncertain (about rewards or predictions) so that they may discover sparse rewards."
- "In portfolio optimization, we want to maximize returns while limiting risk."
- "US equity markets finished disappointingly in 2018 due to increased geopolitical uncertainty."

What exactly then, is uncertainty? **Uncertainty** measures reflect the amount of dispersion of a random variable. In other words, it is a scalar measure of how "random" a random variable is. In finance, it is often referred to as **risk**.

There is no single formula for uncertainty because there are many different ways to measure dispersion: standard deviation, variance, value-at-risk (VaR), and entropy are all appropriate measures. However, it's important to keep in mind that a single scalar number cannot paint a full picture of "randomness", as that would require communicating the entire random variable itself!

Nonetheless, it is helpful to collapse randomness down to a single number for the purposes of optimization and comparison. The important thing to remember is that "more uncertainty" is usually regarded as "less good" (except in simulated RL experiments).

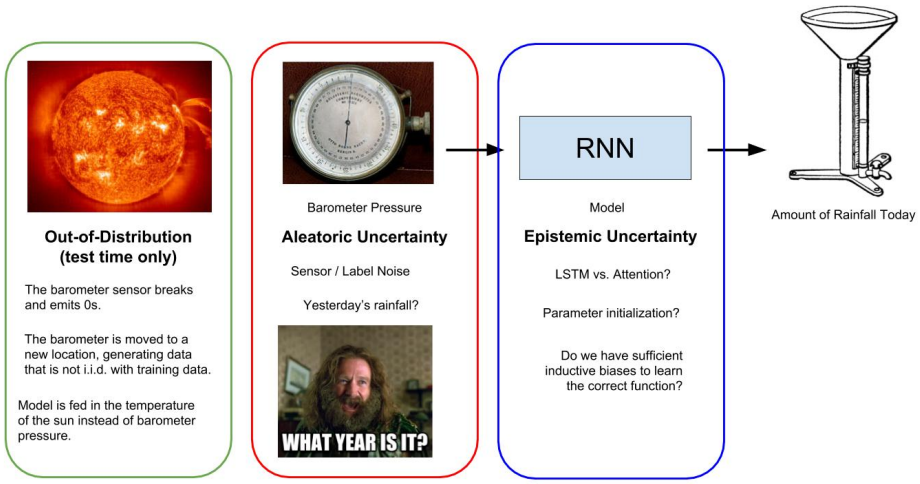

Figure 1: Uncertainty can be understood from a simple machine learning model that attempts to predict daily rainfall from a sequence of barometer readings. Aleatoric uncertainty is irreducible randomness that arises from the data collection process. Epistemic uncertainty reflects confidence that our model is making the correct predictions. Finally, out-of-distribution errors arise when the model sees an input that differs from its training data (e.g. temperature of the sun, anomalies).

1 Types of Uncertainty

Statistical machine learning concerns itself with the estimation of models $p(\theta|\mathcal{D})$, which in turn estimate unknown random variables $p(y|x)$. Multiple forms of uncertainty come into play here. Some notions of uncertainty describe inherent randomness that we should expect (e.g. outcome of a coin flip) while others describe our lack of confidence about our best guess of the model parameters.

To make things more concrete, let's consider a recurrent neural network (RNN) that predicts the amount of rainfall today from a sequence of daily barometer readings. A barometer measures atmospheric pressure, which often drops when its about to rain. Here's a diagram (Figure 1) summarizing the rainfall prediction model along with different kinds of uncertainty.

1.1 Aleatoric Uncertainty

Aleatoric Uncertainty draws its name from the Latin root *aleatorius*, which means the incorporation of chance into the process of creation. It describes randomness arising from the data generating process itself; noise that cannot be eliminated by simply drawing more data. It is the coin flip whose outcome you cannot know.

In our rainfall prediction analogy, aleatoric noise arises from imprecision of

the barometer. There are also important variables that the data collection setup does not observe: How much rainfall was there yesterday? Are we measuring barometric pressure in the present day, or the last ice age? These unknowns are inherent to our data collection setup, so collecting more data from that system does not absolve us of this uncertainty.

Aleatoric uncertainty propagates from the inputs to the model predictions. Consider a simple model $y = 5x$, which takes in normally-distributed input $x \sim \mathcal{N}(0, 1)$. In this case, $y \sim \mathcal{N}(0, 5)$, so the aleatoric uncertainty of the predictive distribution can be described by $\sigma = 5$. Of course, predictive aleatoric uncertainty is more challenging to estimate when the random structure of the input data x is not known.

One might think that because aleatoric uncertainty is irreducible, one cannot do anything about it and so we should just ignore it. No! One thing to watch out for when training models is to choose an output representation capable of representing aleatoric uncertainty correctly. A standard LSTM does not emit probability distributions, so attempting to learn the outcome of a coin flip would just converge to the mean. In contrast, models for language generation emit a sequence of categorical distributions (words or characters), which can capture the inherent ambiguity in sentence completion tasks.

1.2 Epistemic Uncertainty

"Good models are all alike; every bad model is wrong in its own way."

Epistemic Uncertainty is derived from the Greek root *epistēmē*, which pertains to knowledge about knowledge. It measures our ignorance of the correct prediction or model parameters.

Below is a plot of a Gaussian Process Regression model on some toy 1-dimensional dataset. The confidence intervals reflect epistemic uncertainty; the uncertainty is zero for training data (red points), and as we get farther away from training points, the model ought to assign higher standard deviations to the predictive distribution. Unlike aleatoric uncertainty, epistemic uncertainty can be reduced by gathering more data and "ironing out" the regions of inputs where the model lacks knowledge.

There is a rich line of inquiry connecting Deep Learning to Gaussian Processes. The hope is that we can extend the uncertainty-awareness properties of GPs with the representational power of neural networks. Unfortunately, GPs are challenging to scale to the uniform stochastic minibatch setting for large datasets.

If one wants maximum flexibility in choosing their model family, a good alternative to estimating uncertainty is to use ensembles, which is just a fancy way of saying "multiple independently learned models". While GP models analytically define the predictive distribution, ensembles can be used to compute the *empirical distribution* of predictions.

Any individual model will make some errors due to randomized biases that occur during the training process. Ensembling is powerful because other models

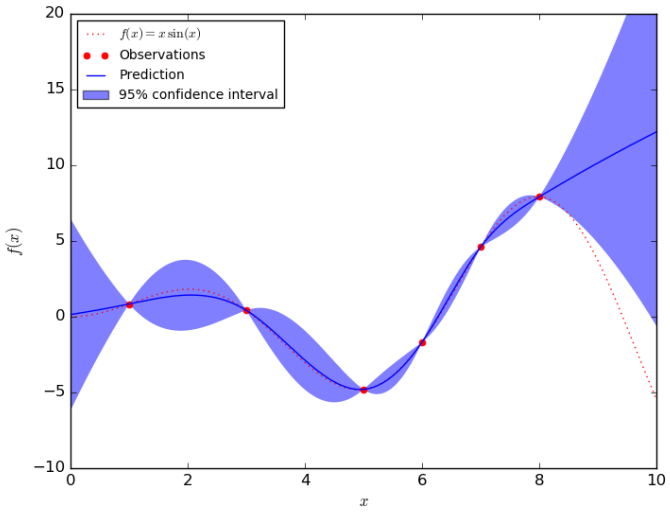

Figure 2: 1-D Gaussian Process Regression Model showcasing epistemic uncertainty for inputs outside its training set.

in the ensembles tend to expose the idiosyncratic failures of a single model while agreeing with the correctly inferred predictions.

How do we sample models randomly to construct an ensemble? In Ensembling with bootstrap aggregation, we start with a training dataset of size N and sample M datasets of size N from the original training set (with replacement, so each dataset does not span the entire dataset). The M models are trained on their respective datasets and their resulting predictions collectively form an empirical predictive distribution.

There is a line of interesting work that proposes to use Dropout training to approximate a model ensemble. However, a word of warning: introducing dropout involves an extra hyperparameter and can compromise single model performance (often unacceptable for real world applications where calibrated uncertainty estimation is secondary to accuracy).

Therefore, if one has access to plentiful computing resources (as one does at Google), it is often easier to just re-train multiple copies of a model. This also yields the benefits of ensembling without hurting performance. This is the approach taken by the Deep Ensembles paper. The authors of this paper also mention that the random training dynamics induced by differing weight initializations were sufficient to introduce a diverse set of model without having to resort to reducing the training set diversity via bootstrap aggregation. From a practical engineering standpoint, it's smart to bet on risk estimation methods that do not get in the way of the model's performance or whatever other ideas the researcher wants to try.

1.3 Out of Distribution Errors

For our rainfall predictor, what if instead of feeding in the sequence of barometer readings, we fed in the temperature of the sun? Or a sequence of all zeros? Or barometer readings from a sensor that reports in different units? The RNN will happily compute away and give us a prediction, but the result will likely be meaningless.

The model is totally unqualified to make predictions generated via some other procedure than how the training data was created. This is a failure mode that is often overlooked in benchmark-driven ML research, because we typically assume that the training, validation, and test sets consist entirely of clean i.i.d data.

Determining whether inputs are "valid" is a serious problem for deploying ML in the wild, and is known as the Out of Distribution (OoD) problem. OoD is also synonymous with model misspecification error (generalization) and anomaly detection.

Besides its obvious importance for hardening ML systems, anomaly detection models are an intrinsically useful technology. For instance, we might want to build a system that monitors a healthy patient's vitals and alerts us when "something goes wrong" without necessarily having seen that pathology before. Or we might be managing the "health" of a datacenter and want to know whenever unusual activity occurs (disks filling up, security breaches, hardware failures, etc.)

Since OoD inputs only occur at test-time, we should not presume to know the distribution of anomalies the model encounters. This is what makes OoD detection tricky - we have to harden a model against inputs it never sees during training! This is incidentally the standard attack scenario in Adversarial Machine Learning.

There are two ways to handle OoD inputs for a machine learning model: 1) catch the bad inputs before we even put them through the model 2) let the "weirdness" of model predictions imply to us that the input was probably malformed.

In the first approach, we assume nothing about the downstream ML task, and simply consider the problem of whether an input is in the training distribution or not. This is exactly what discriminators in Generative Adversarial Networks (GANs) are supposed to do. However, a single discriminator is not completely robust because the GAN discriminator only learns to discriminate between the true data distribution and whatever the generative distribution is; it can give arbitrary predictions for an input that lies in neither distribution.

Instead of a discriminator, we could build a density model of the in-distribution data, such as a kernel density estimator or fitting a Normalizing Flow to the data. Hyunsun Choi and I investigated this in our recent paper on using modern generative models to do OoD detection.

The second approach to OoD detection involves using the predictive (epistemic) uncertainty of the task model to tell us when inputs are OoD. Ideally, malformed inputs to a model ought to generate "weird" predictive distribution

$p(y|x)$. For instance, Hendrycks and Gimpel showed that the maximum softmax probability (the predicted class) for OoD inputs tends to be lower than that of in-distribution inputs. Here, uncertainty is inversely proportional to the "confidence" as modeled by the max sofmax probability. Models like Gaussian Processes give us these uncertainty estimates by construction, or we could compute epistemic uncertainty via Deep Ensembles.

In reinforcement learning, OoD inputs are actually assumed to be a *good thing*, because it represents inputs from the world that the policy function does not know about yet. Encouraging the policy to find its own OoD inputs implements "intrinsic curiosity" to explore regions the model predicts poorly in. This is all well and good, but I do wonder what would happen if such "curiousity-driven" agents are deployed in real world settings where sensors break easily, which also generate OoD inputs. How does a robot distinguish between "unseen states" (good) and "sensors breaking" (bad)? Might that result in agents that learn to interfere with their sensory mechanisms to generate maximum novelty?

2 Who Will Watch the Watchdogs?

As mentioned in the previous section, one way to defend ourselves against OoD inputs is to set up a likelihood model that "watchdogs" the inputs to a model. I prefer this approach because it de-couples the problem of OoD inputs from epistemic and aleatoric uncertainty in the task model. Makes things easy to analyze from an engineering standpoint.

But we should not forget that the likelihood model is also a function approximator, possibly with its own OoD errors! We show in our recent work on Generative Ensembles (and also showed in concurrent work by DeepMind), that under a CIFAR likelihood model, natural images from SVHN can actually be more likely than the in-distribution CIFAR images themselves!

However, all is not lost! It turns out that the *epistemic uncertainty* of likelihood models is an excellent OoD detector for the likelihood model itself. By bridging epistemic uncertainty with density estimation, we can use ensembles of likelihood models to protect machine learning models against OoD inputs in a model-agnostic way.

3 Calibration: the Next Big Thing?

A word of warning: just because a model is able to spit out a confidence interval for a prediction doesn't mean that the confidence interval actually reflects the actual probabilities of outcomes in reality!

Confidence intervals (e.g. 2σ) implicitly assume that your predictive distribution is Gaussian-distributed, but if the distribution you're trying to predict is multi-modal or heavy-tailed, then your model will not be well calibrated!

Suppose our rainfall RNN tells us that there will be $\mathcal{N}(4, 1)$ inches of rain today. If our model is *calibrated*, then if we were to repeat this experiment

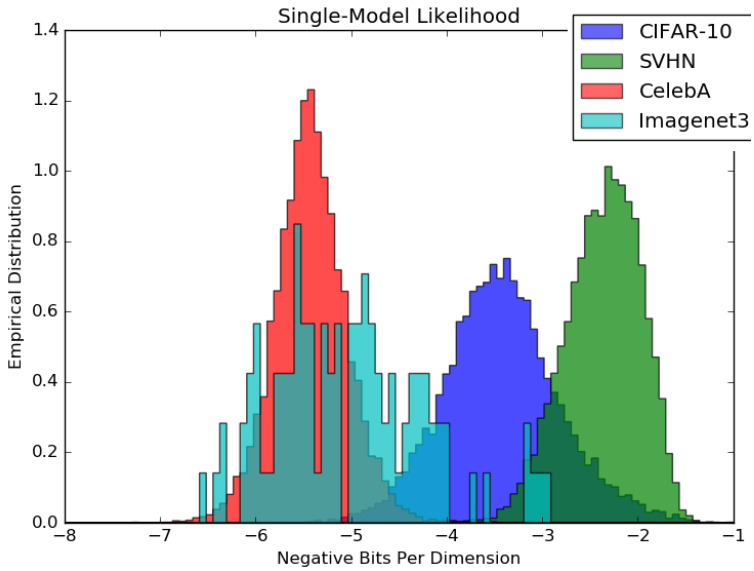

Figure 3: Likelihood estimation involves a function approximator that can itself be susceptible to OoD inputs. A likelihood model of CIFAR assigns higher probabilities to SVHN images than CIFAR test images!

over and over again under identical conditions (possibly re-training the model each time), we really would observe empirical rainfall to be distributed exactly $\mathcal{N}(4, 1)$.

Machine Learning models developed by academia today mostly optimize for test accuracy or some fitness function. Researchers are not performing model selection by deploying the model in repeated identical experiments and measuring calibration error, so unsurprisingly, our models tend to be poorly calibrated.

Going forward, if we are to trust ML models deployed in the real world (robotics, healthcare, etc.), I think a much more powerful way to "prove our models understand the world correctly" (in a statistical sense) is to test them for statistical calibration. Good calibration implies good accuracy, so it would be a strictly harder benchmark to optimize against.

4 Should Uncertainty be Scalar?

As useful as they are, scalar uncertainty measures will never be as informative as the random variables they describe. I find methods like particle filtering and Distributional Reinforcement Learning very cool because they are algorithms that operate on entire distributions, freeing us from resorting to simple normal distributions to keep track of uncertainty. Instead of shaping ML-based decision making with a single scalar of "uncertainty", we can now query the full structure

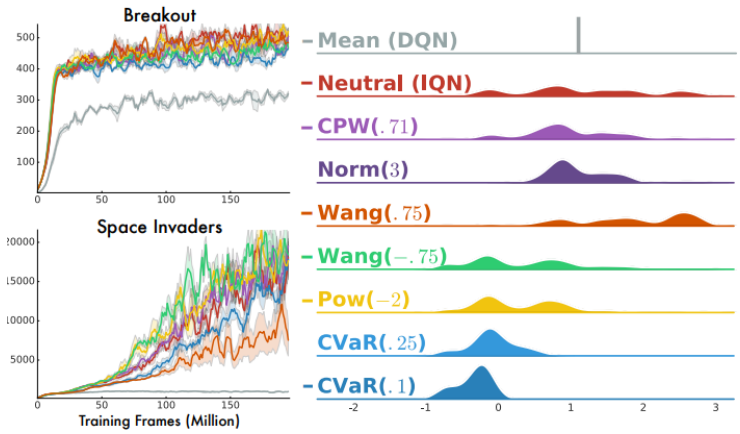

Figure 4: Performance of various risk measures on Atari games as reported by the IQN paper.

of distributions when deciding what to do.

The Implicit Quantile Networks paper (Dabney et al.) has a very nice discussion on how to construct "risk-sensitive agents" from a return distribution. In some environments, one might favor an opportunitistic policy that prefers to explore the unknown, while in other environments unknown things may be unsafe and should be avoided. The choice of Risk Measure essentially determines how to map the distribution of returns to a scalar quantity that can be optimized against. All risk measures can be computed from the distribution, so predicting full distributions enables us to combine multiple definitions of risk easily. Furthermore, supporting flexible predictive distributions seems like a good way to improve model calibration.

Risk measures are a deeply important research topic to financial asset managers. The vanilla Markowitz portfolio objective minimizes a weighted variance of portfolio returns $\frac{1}{2}\lambda w^T \Sigma w$. However, variance is an unintuitive choice of "risk" in financial contexts: most investors don't mind returns exceeding expectations, but rather wish to minimize the probability of small or negative returns. For this reason, risk measures like Value-at-Risk, Shortfall Probability, and Target Semivariance, which only pay attention to the likelihood of "bad" outcomes, are more useful objectives to optimize. However, they are analytically more difficult to work with. My hope is that research into distributional RL, Monte Carlo methods, and flexible generative models will allow us to build differentiable relaxations of risk measures that can play nicely with portfolio optimizers.

5 Summary

Here's a recap of the main points of this post:

- Uncertainty/risk measures are scalar measures of "randomness". Collapsing a random variable to a single number is done for optimization and mathematical convenience.
- Predictive uncertainty can be decomposed into aleatoric (irreducible noise arising from data collection process), epistemic (ignorance about true model), and out-of-distribution (at test time, inputs may be malformed).
- Epistemic uncertainty can be mitigated by softmax prediction thresholding or ensembling.
- Instead of propagating OoD uncertainty to predictions, we can use a task-agnostic filtering mechanism that safeguards against "malformed inputs".
- Density models are a good choice for filtering inputs at test time. However, it's important to recognize that density models are merely approximations of the true density function, and are themselves susceptible to out-of-distribution inputs.
- Generative Ensembles reduce epistemic uncertainty of likelihood models so they can be used to detect OoD inputs.
- Calibration is important and underappreciated in research models.
- Some algorithms (Distributional RL) extend ML algorithms to models that emit flexible distributions, which provides more information than a single risk measure.

6 Further Reading

- I especially recommend Chapter 3 ("Risk Measurement") of Modern Investment Management by Litterman et al. to learn about risk in a concrete way.
- http://uqpm2017.usacm.org/sites/default/files/DStarcuzzi_UQConf.pdf
- http://mlg.eng.cam.ac.uk/yarin/blog_2248.html