Uncertainty Quantification in Machine Learning

July 7, 2021

Dr. Matias Valdenegro Toro
German Research Center for Artificial Intelligence, Bremen

matias.valdenegro@dfki.de
@mvaldenegro
http://github.com/mvaldenegro

Contents

Motivation

- As ML is used for real processes, the testing set is effectively infinite, and there are needs to evaluate how certain are the predictions in completely unseen samples, as this additional information can be used to assert if the predictions are useful.
- **Autonomous Driving**: The decision making process while driving needs to know if the predictions from a perception model are reliable, and to *know when the model does not know*.
- **Medical Applications**: Low confidence predictions might indicate that additional tests might be required, instead of taking a decision over faulty data.

Motivation

- We need ML models that can answer the following questions:

Do I know that I do not know?
Can I refuse to provide an answer?

- This is related to out of distribution detection, a model can provide information to say that the input is dissimilar to the training set, or the task is very different, and refuse to answer.
- Classic example is to train a cat/dog model, and test with a bird image. The answer will always be wrong for a typical model.

Why Uncertainty?

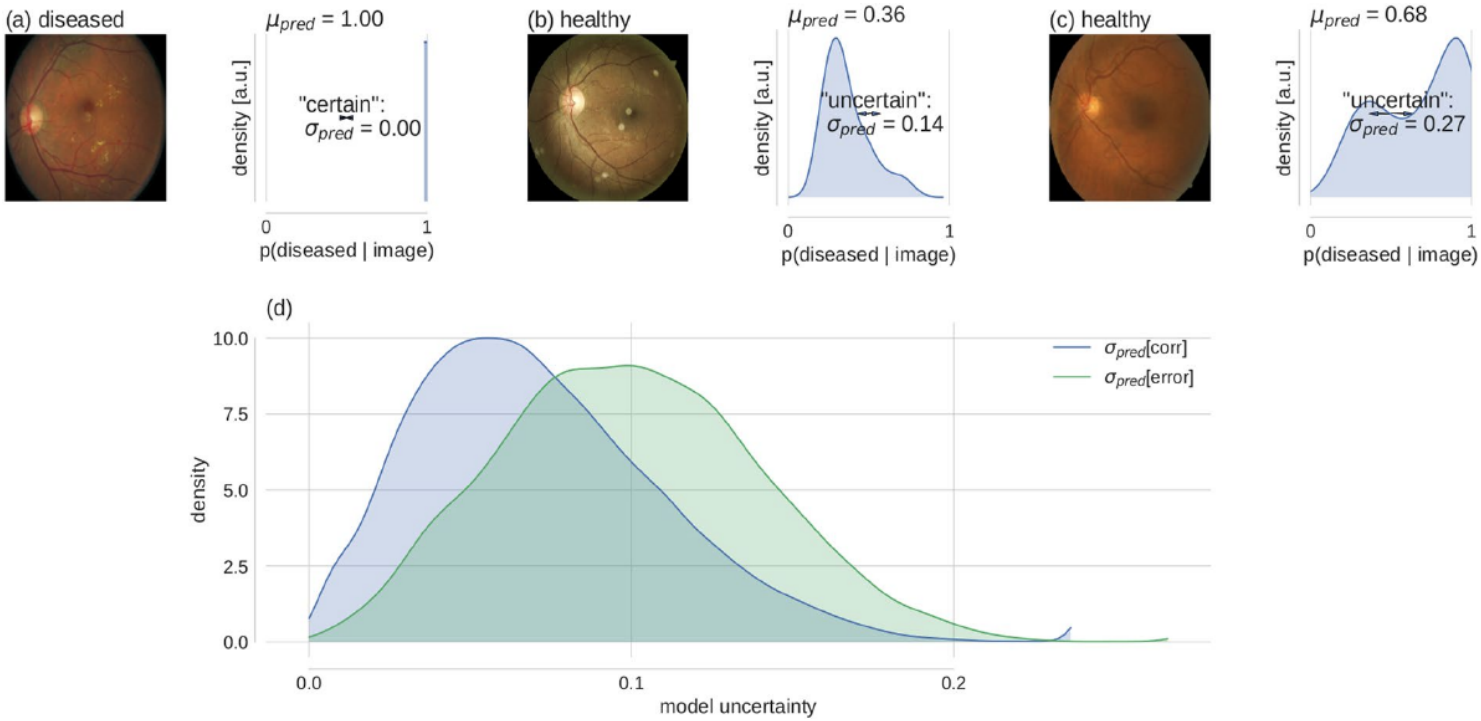


Figure 1. Bayesian model uncertainty for diabetic retinopathy detection. (**a–c**) **left:** Exemplary fundus images with human label assignments in the titles. (**a–c**) **right**: Corresponding approximate predictive posteriors (Eq. 6) over the softmax output values p(diseased | image) (Eq. 1). Predictions are based on μ_{pred} (Eq. 7) and uncertainty is quantified by σ_{pred} (Eq. 8). Examples are ordered by increasing uncertainty from left to right. (**d**) Distribution of uncertainty values for all Kaggle DR test images, grouped by correct and erroneous predictions. Label assignment for "diseased" was based on thresholding μ_{pred} at 0.5. Given a prediction is incorrect, there is a strong likelihood that the prediction uncertainty is also high.

Figure extracted from "Leveraging uncertainty information from deep neural networks for disease detection" by Leibig et al. 2017.

Why Uncertainty?

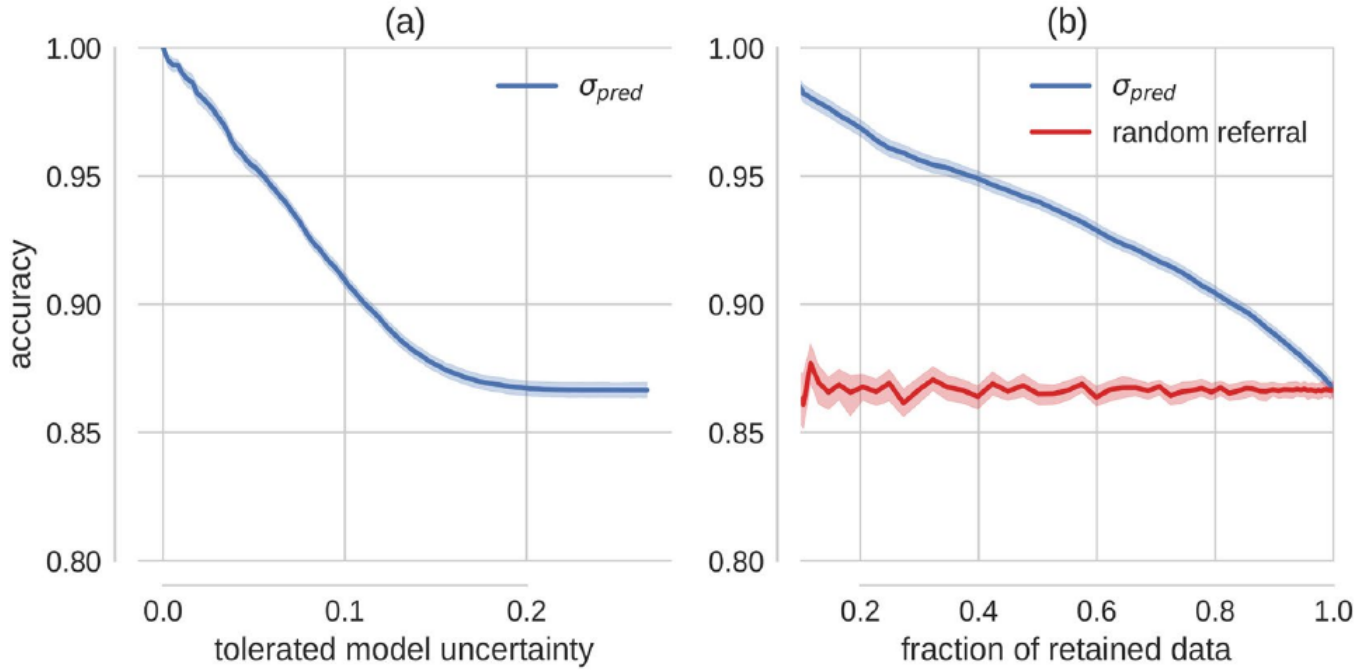


Figure 3. Improvement in accuracy via uncertainty-informed decision referral. (**a**) The prediction accuracy as a function of the tolerated amount of model uncertainty. (**b**) Accuracy is plotted over the retained data set size (test data set size - referral data set size). The red curve shows the effect of rejecting the same number of samples randomly, that is without taking into account information about uncertainty. Exemplarily, if 20% of the data would be referred for further inspection, 80% of the data would be retained for automatic diagnostics. This results in a better test performance (accuracy $\geq$ 90%, point on blue curve) on the retained data than on 80% of the test data sampled uniformly (accuracy $\approx$ 87%, point on red curve). Uncertainty informed decision referral derived from the conventional softmax output cannot achieve consistent performance improvements (Fig. 4).

Figure extracted from "Leveraging uncertainty information from deep neural networks for disease detection" by Leibig et al. 2017.

Why Uncertainty?

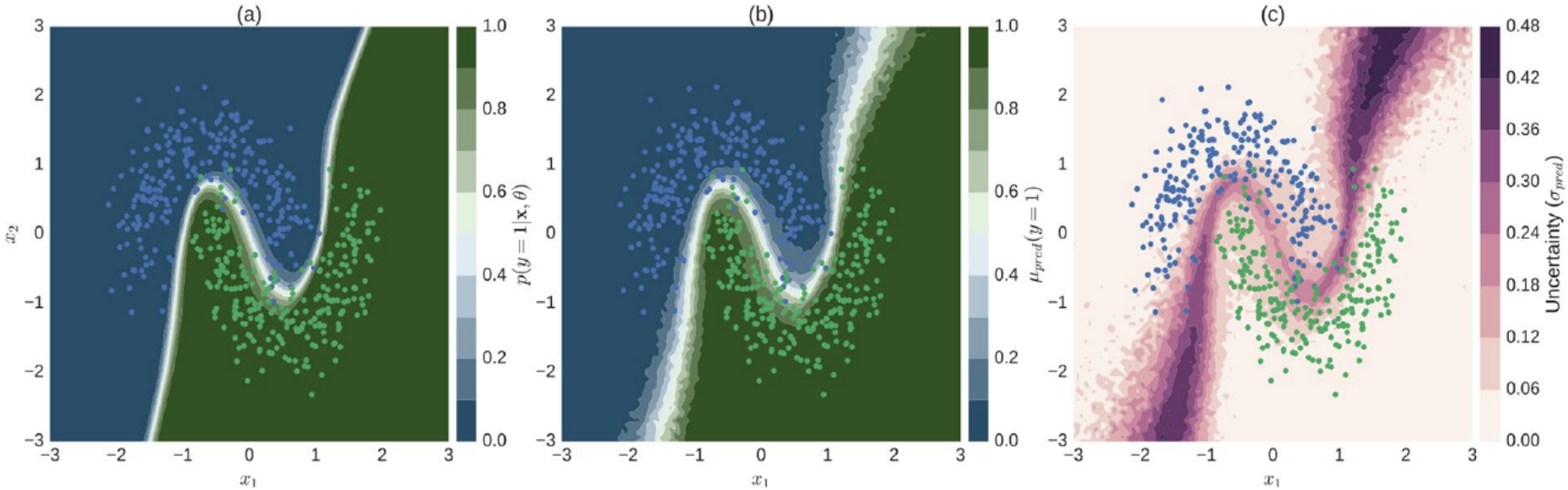
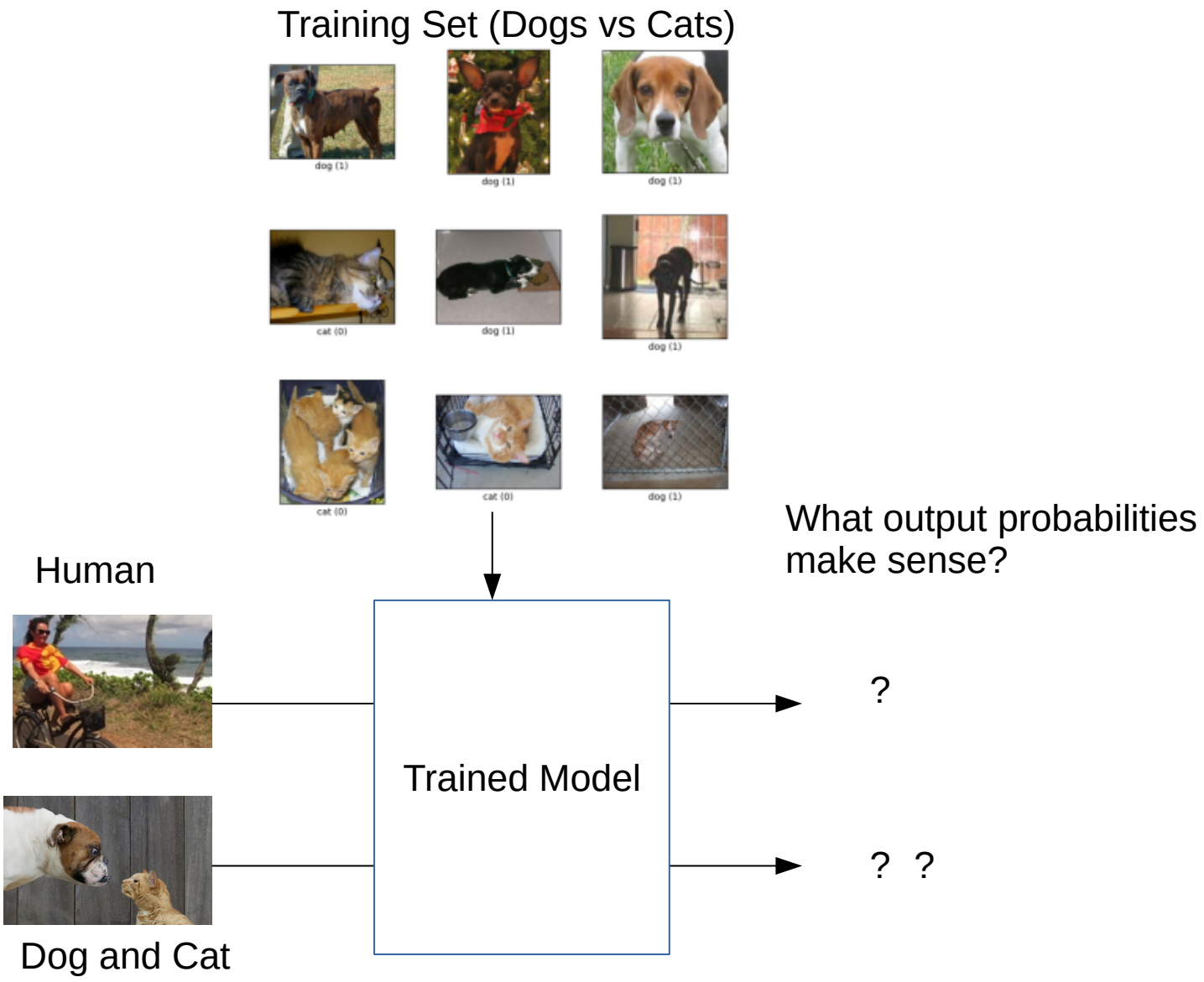


Figure 5. Illustration of uncertainty for a 2D binary classification problem. (**a**) Conventional softmax output obtained by turning off dropout at test time (eq. 1). (**b**) Predictive mean of approximate predictive posterior (eq. 7). (**c**) Uncertainty, measured by predictive standard deviation of approximate predictive posterior (eq. 8). The softmax output (**a**) is overly confident (only a narrow region in input space assumes values other than 0 or 1) when compared to the Bayesian approach (**b**,**c**). Uncertainty (**c**) tends to be high for regions in input space through which alternative separating hyperplanes could pass. Colour-coded dots in all subplots correspond to test data the network has not seen during training.

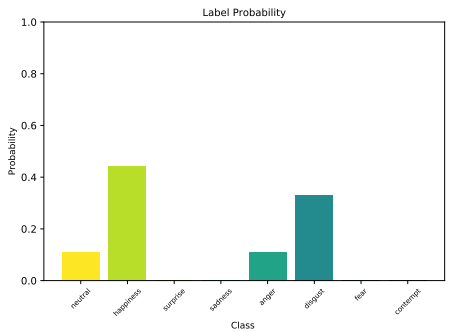
Figure extracted from "Leveraging uncertainty information from deep neural networks for disease detection" by Leibig et al. 2017.

What is Uncertainty in Machine Learning?

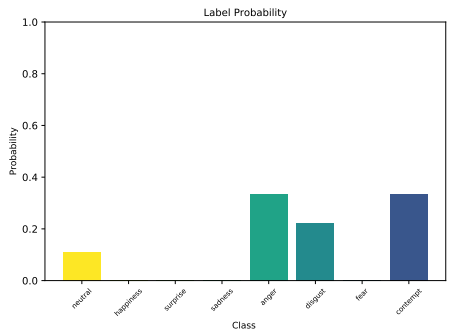


What is Uncertainty in Machine Learning?

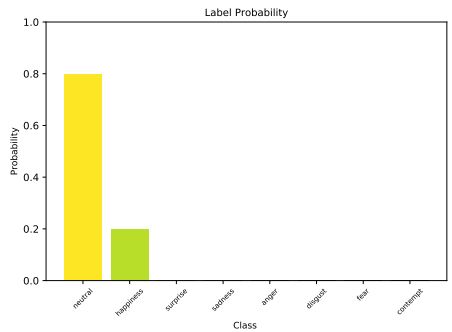
Happiness



Anger



Neutral



FER+ fataset, with crowd sourced labels for emotion recognition, over classes Neutral, Happinesss, Surprise, Sadness, Anger, Disgust, Fear, and Contempt.

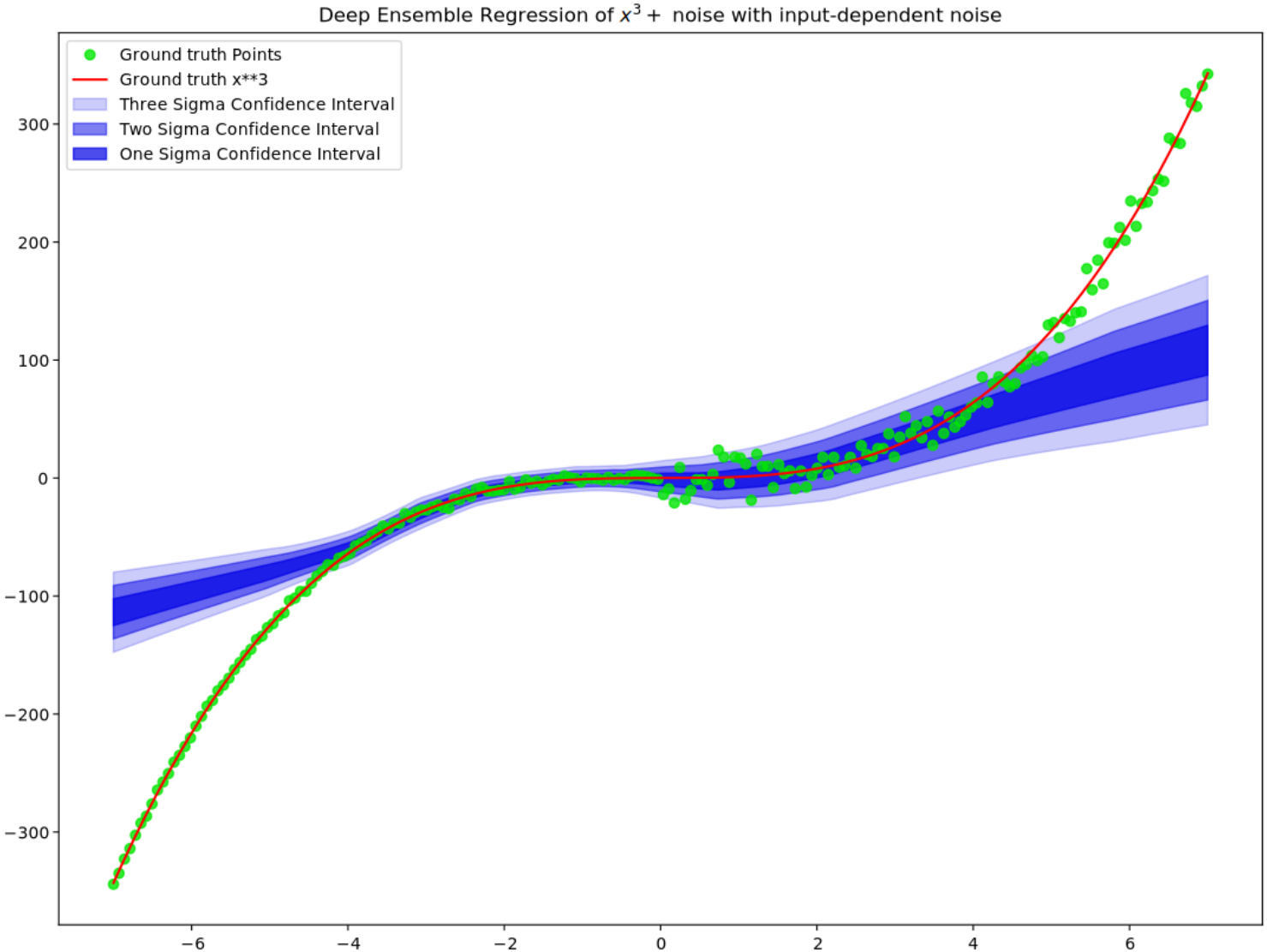
What is Uncertainty in ML?

- Real-world datasets are typically unbalanced, so confidences on each class should be different, reflecting the training data and model inferences.
- Real-world datasets might contain noise, like imprecise labels, ambiguous measurements, or sensor noise. A model should be aware of this.
- Most neural networks are overconfident, meaning that softmax confidences do not have a good probabilistic interpretation and could be misleading.

What do Classical Models Lack?

- Most machine learning models do not explicitly model uncertainty at their outputs.
- They produce `point-wise` predictions. A model with uncertainty outputs a `distribution`.
- A distribution can usually include more information than a single point-wise prediction, for example, mean and variance for a regression output instead of just a point prediction.
- Neural networks are often overconfident, producing wrong predictions with high confidence.

What do Classical Models Lack?



Practical Applications of Uncertainty

- Reliable confidence estimates can be used to detect misclassified examples or when the model is extrapolating.
- A model can reject to produce an output if the uncertainty is too high, for example, to require human processing instead of automated. This is called out of distribution detection.
- The confidence or uncertainty of a prediction tells the human how much it should really trust the prediction.
- Additional decision making can be made with a realistic confidence score, which is very important for medical and human-interaction applications.

Types of Uncertainty

Aleatoric Uncertainty

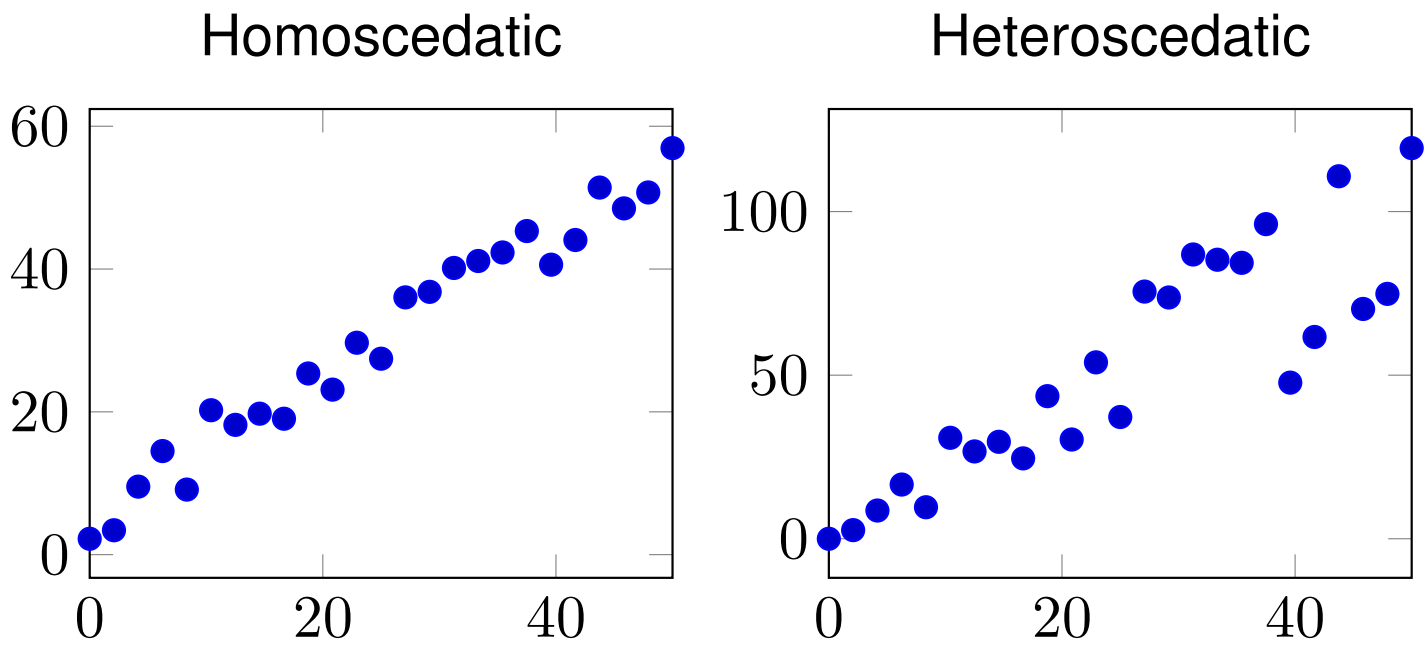
Uncertainty that is inherent to the data, for example, sensor noise, stochastic processes.
Cannot be reduced by adding more information.

Epistemic Uncertainty

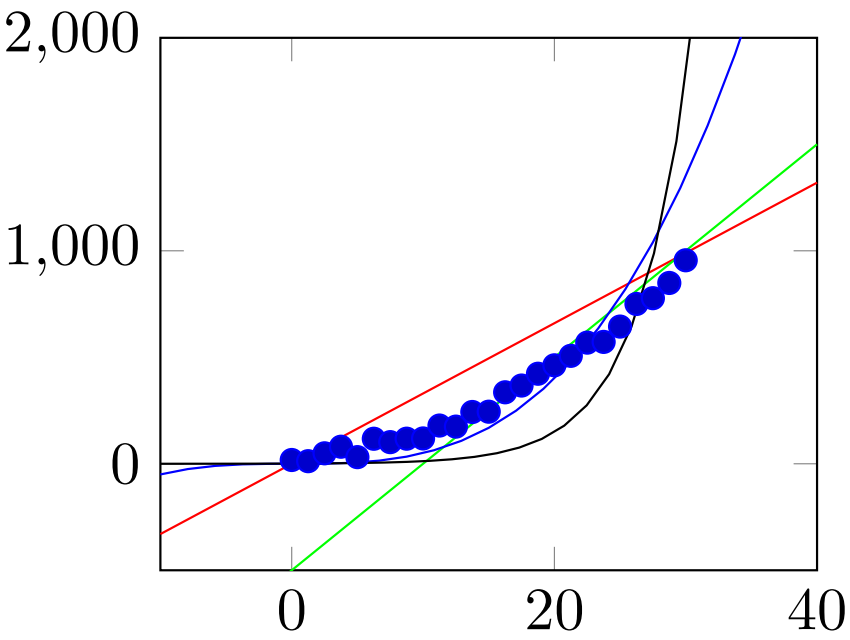
Uncertainty produced by the model, for example, model misspecification, class imbalance, lack of training data.
Can be reduced by adding more information to the training process.

Aleatoric Uncertainty

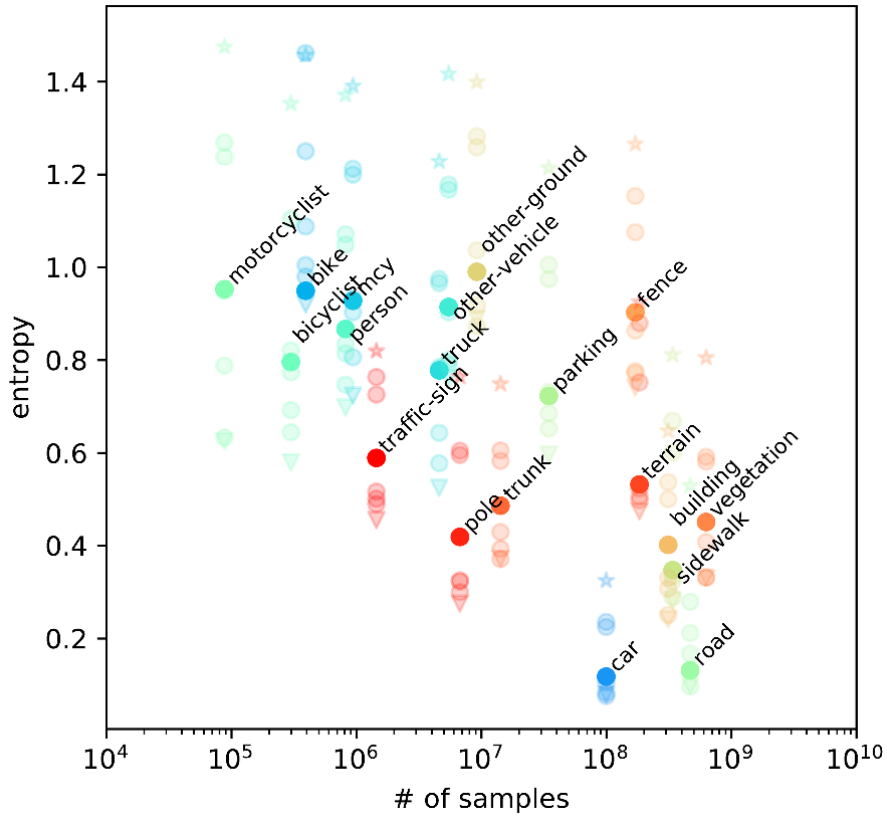
The simplest example of AU is measurements corrupted by additive noise, like $f(x) = x^3 + \epsilon$ Where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ and x^3 would be the true function.
If σ^2 is constant, this is called homoscedatic noise, if σ^2 is a function of the input or variable, then it is called heteroscedatic noise.



Epistemic Uncertainty



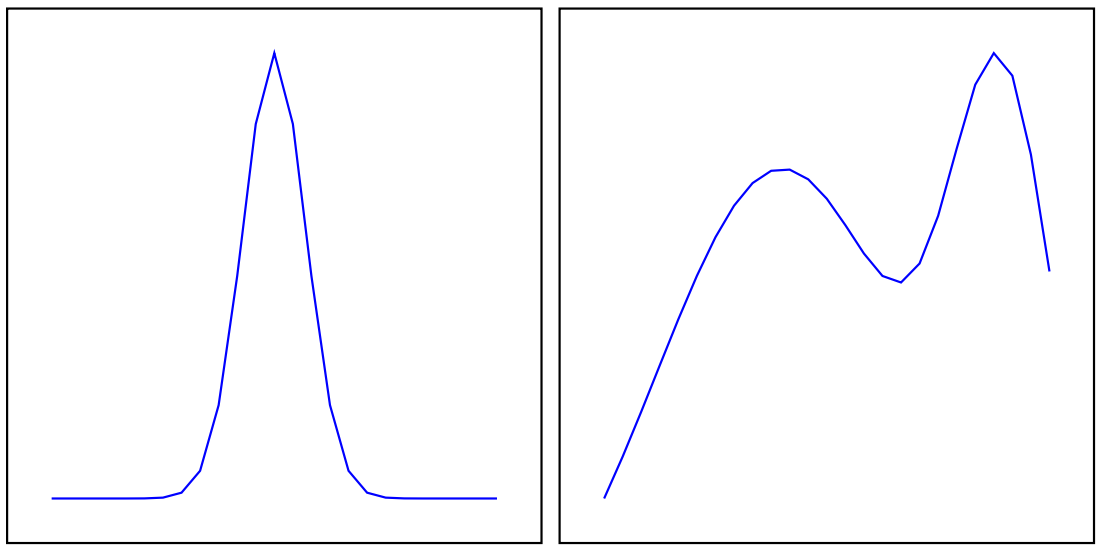
Model Misspecification



Variations of # Samples in Training Data

Uncertainty Representations

- Before starting with techniques, we need to discuss how uncertainty information is represented.
- In general, this is not so trivial as there are multiple representations, and it depends on the kind of task.
- The most generic representation is to use a probability distribution on the output. This distribution indirectly encodes uncertainty.



Uncertainty Representation - Regression

Assuming that the output is represented by $f(x)$, then there are two principal ways to represent uncertainty.

Confidence Intervals

The output is within some defined interval $[a, b]$:

$$f(x) \in [a, b]$$

Usually some methods give a probability that the output belongs to the interval.

Uncertainty Representation - Regression

Assuming that the output is represented by $f(x)$, then there are two principal ways to represent uncertainty.

Mean and Variance

Uncertainty is represented as the variation of the output from the mean ($f(x)$):

$$f(x) \pm \sigma$$

A equivalent interval would be $f(x) \in [f(x) - \sigma, f(x) + \sigma]$

Uncertainty Representation - Classification

- For classification, representing uncertainty is a bit more hard.
- The only robust representation is to use a discrete probability distribution.
- The easiest way to implement it is to use a softmax activation (for multi-class) or a sigmoid activation (for binary classification).

Overconfidence and Calibration

- Many models produce probabilities or confidence intervals that are not good, mostly are overconfident.
- Probabilities or confidence intervals must have a specific meaning, and this can be measured by calibration.

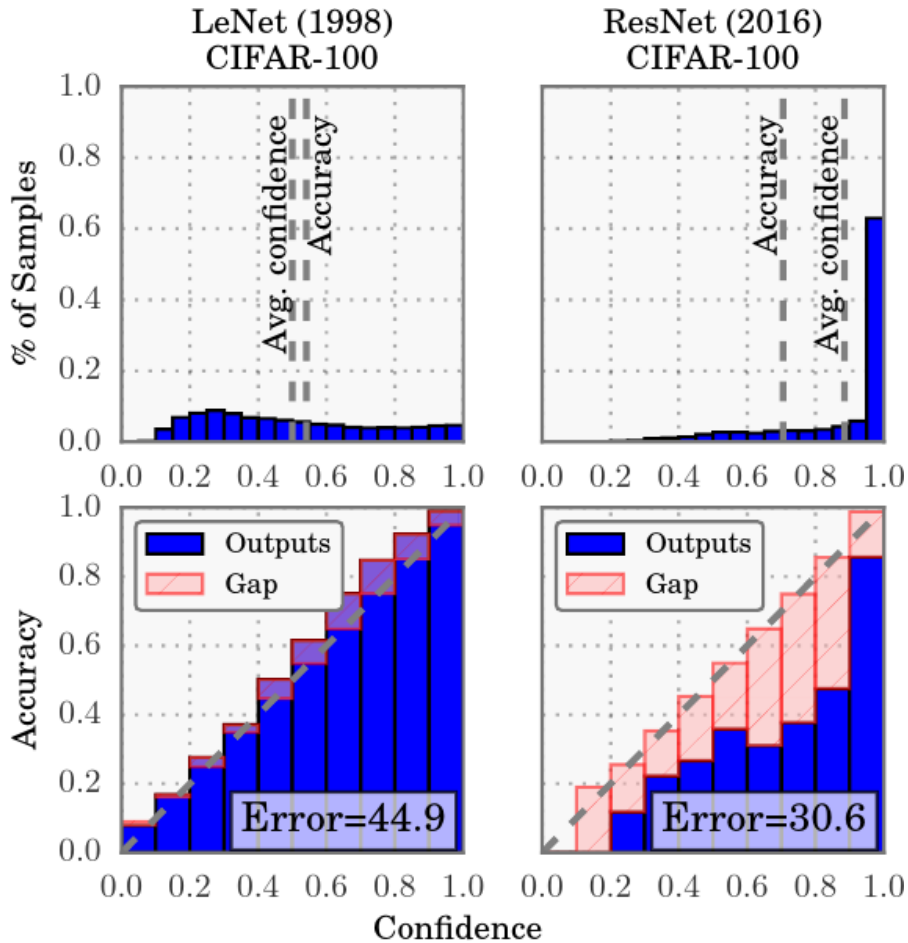


Figure 1. Confidence histograms (top) and reliability diagrams (bottom) for a 5-layer LeNet (left) and a 110-layer ResNet (right) on CIFAR-100. Refer to the text below for detailed illustration.

Entropy

Entropy is a measurement of the "information content" in a probability distribution. It is defined as:

$$H = -\sum_x P(x) \log P(x) = \sum_x P(x) \log \frac{1}{P(x)}$$

The entropy is important as it is directly related to uncertainty. The units of entropy are called bits if one uses the base-two logarithm.
The uniform distribution is the one that maximizes entropy, as its results are harder to predict. For a fixed mean and variance, the Gaussian distribution is the one that maximizes the entropy.

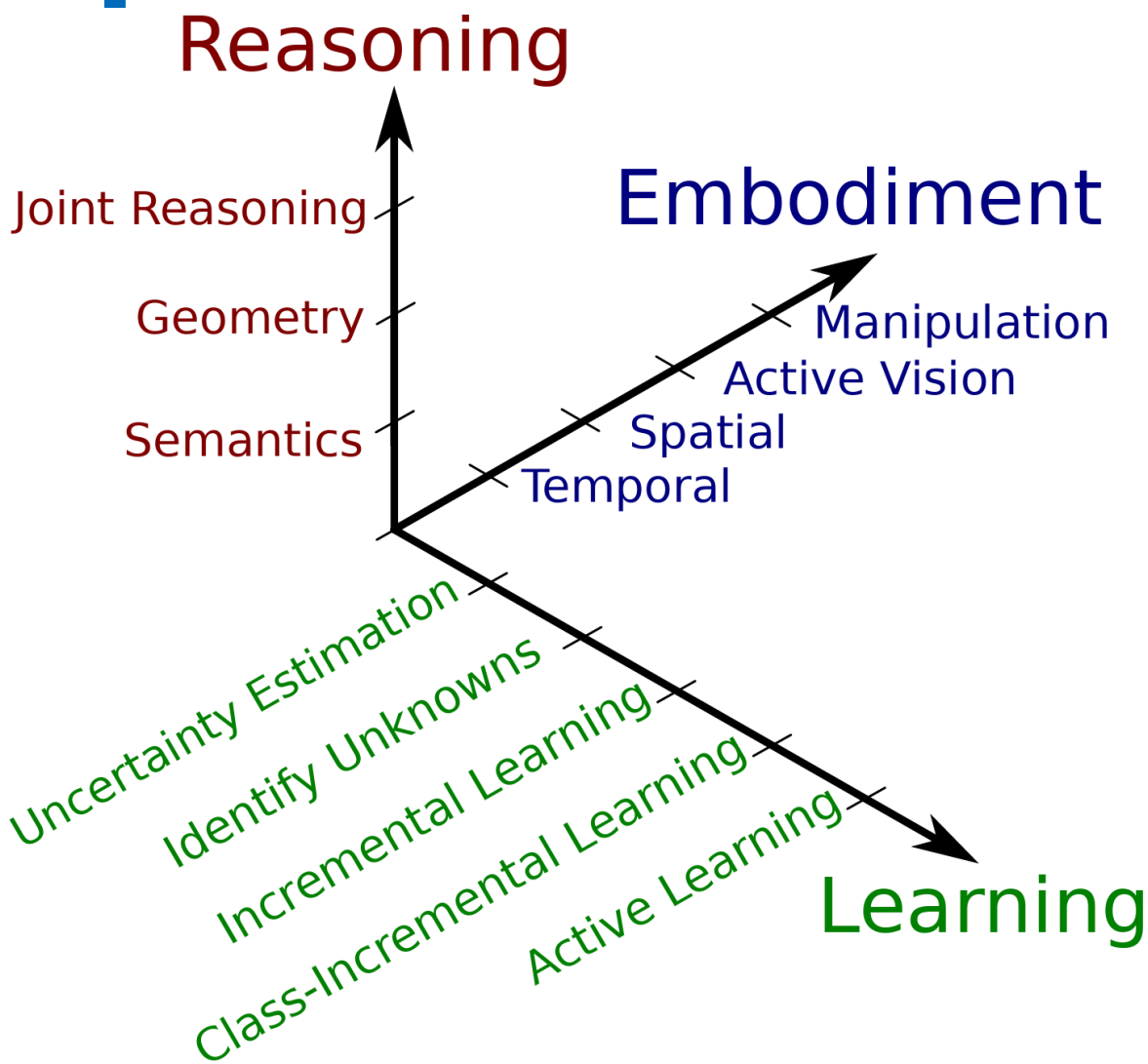
Entropy Examples

- For a event that always happens, its entropy is zero bits.
- A fair coin flip has 1.0 bits of entropy.
- A uniform distribution has the maximum entropy possible across distributions.
- In general, the more information content, the higher the entropy.

Challenges of DL in Robotics [Sünderhauf et al. 2018]

- Machine/Deep Learning and Computer Vision by itself is quite different from Robotics. The main difference is that a robot has a "body".
- A good description paper about this topic is "The Limits and Potentials of Deep Learning for Robotics" by Sünderhauf et al. 2018.
- Embodiment is the main difference between Robot Learning/Perception and their more theoretical fields of Machine/Deep Learning and Computer Vision.

Challenges of DL in Robotics [Sünderhauf et al. 2018]



Challenges of DL in Robotics - Learning [Sünderhauf et al. 2018]

Level	Name	Description
5	Active Learning	The system is able to select the most informative samples for incremental learning on its own in a data-efficient way. It can ask the user to provide labels.
4	Class-Incremental Learning	The system can learn *new* classes, preferably using low-shot or one-shot learning techniques, without catastrophic forgetting. The system requires the user to provide these new training samples along with correct class labels.
3	Incremental Learning	The system can learn off new instances of known classes to address domain adaptation or label shift. It requires the user to select these new training samples.
2	Identify Unknowns	In an open-set scenario, the robot can reliably identify instances of unknown classes and is not fooled by out-of distribution data.
1	Uncertainty Estimation	The system can correctly estimate its uncertainty and returns calibrated confidence scores that can be used as probabilities in a Bayesian data fusion framework. Current work on Bayesian Deep Learning falls into this category.
0	Closed-Set Assumptions	The system can detect and classify objects of classes known during training. It provides uncalibrated confidence scores.

Challenges and Applications

Medical Systems and Decision Making

Practically all medical applications require correct (epistemic) uncertainty estimates to be used with humans/animals, receive regulatory approval, and be useful for practitioning medical doctors to make decisions.

Robotics

Generally in Robotics, useful uncertainties are not modeled, for example uncertainty in dynamical systems (parameters), perception (object detection), or estimate when robot capabilities are being extrapolated. The best example is autonomous driving.

Challenges and Applications

Reinforcement Learning

In the same way, it is very important to have RL-learned policies that can estimate their own epistemic uncertainty, and not take an action when the environment is too different from the training one.

- RL in robots or real mechanisms, with safety constraints (Safe RL).
- RL in non-stationary environments (for example, dynamic or unpredictable obstacles).
- Reduce the sample complexity required for training through Active Learning and Exploration.

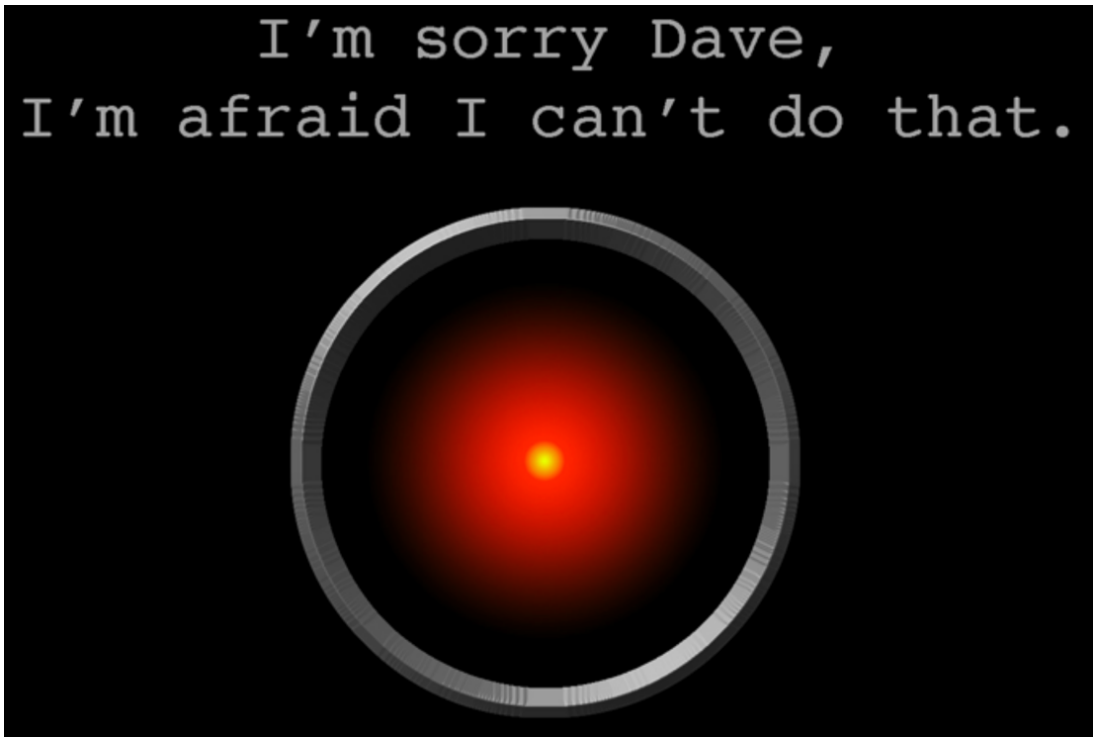
Challenges and Applications

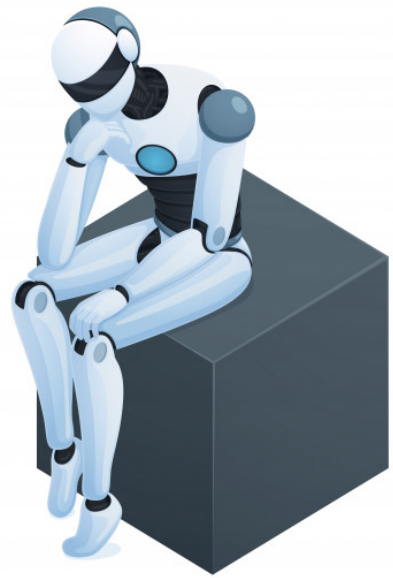
Autonomous Driving

Autonomous Driving is a very important application field, as the whole reason for AD to be desirable, is that it can provide safer driving than humans. But this is not automatic, safety has to be engineered and safe methods have to be developed and used.

- AD usually fails with variations of the test environment, like different weather, physical location of roads, and environmental conditions like snow and rain.
- A car using AD has to detect unusual and strange situations and alert the driver, and has to do this with nearly 100% precision.
- In many AD systems, usual situations are labeled by humans, which does not scale (And it is strange to claim super-human driving from human samples).

Objective - Safe and Trustable Robots





Objective - Safe and Trustable Robots

Examples

- Multiple incidents of experimental Autonomous Vehicles hitting human pedestrians and producing accidents, due to conditions not considered in development/training (similar to Kidnapped Robot Problem).
- Possible issues with Robots at care homes for the elderly. Algorithms should be tuned for maximum safety.
- Well known examples of face recognition being biased against some skin colors, OOD detection can help in preventing or alleviate these.
- AI/Robotics should be done for the social good.

Bayesian View of Uncertainty

Bayesian statistics has a particular view of uncertainty. Typically when learning a model, we wish to learn the probability distribution $P(\mathbf{y} \mid \mathbf{x})$, that is, the probability of some output given the input. But this does not consider the model parameters **w**, which in the end indirectly encode uncertainty.
Then we wish to learn $P(\mathbf{y} \mid \mathbf{x}, \mathbf{w})$ which directly considers the model parameters. As model parameters are learned from data, then we wish to learn $P(\mathbf{w} \mid \mathbf{D})$:

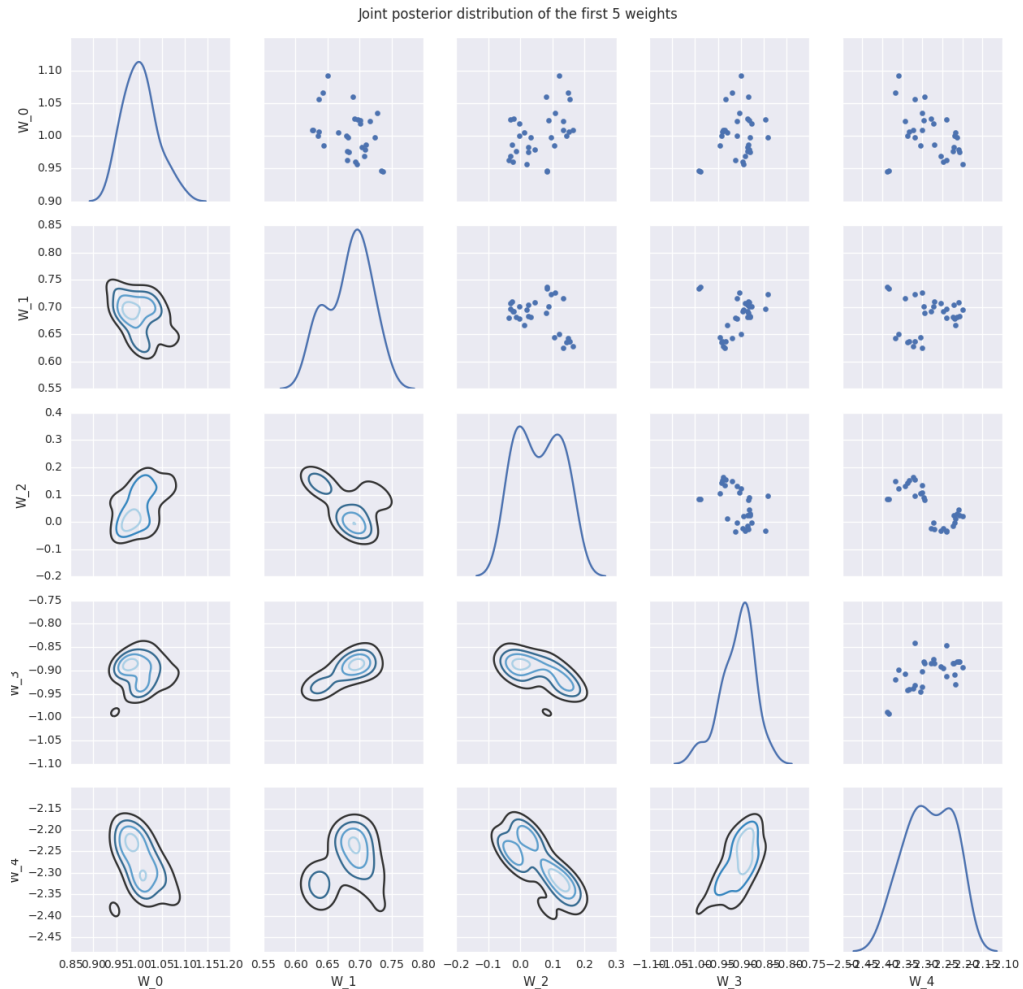
$$P(\mathbf{w} \mid \mathbf{D}) = \frac{P(\mathbf{D} \mid \mathbf{w})P(\mathbf{w})}{P(\mathbf{D})}$$

Here $\mathbf{D} = \{x_i, y_i\}$ is a labeled training set. Bayes Rule' allows us to decompose the probability of the weights given the data (which is unknown) into probability of the data given the weights (computed by the model).

Bayesian View of Uncertainty

- From the Bayesian POV, a model is defined by both the structure (the actual model equations) and the model's parameters.
- This means that for a Bayesian model, the model parameters encode the **model uncertainty** about each prediction. There are equations to compute the output probability distribution from the model and its inputs.
- Intuitively, this can be seen as a simple question. If we have several models trained on the same data, the model's parameters won't be the same. Some parameters might be very close, some will be equal, and some will be radically different.
- These variations on model parameters can be encoded as a probability distribution. Bayesian statistics can be used to estimate these distributions $P(\mathbf{w} \mid \mathbf{D})$

Bayesian View of Uncertainty (on MNIST with SGHMC)



Bayesian Learning

Maximum Likelihood Estimation Maximize the logarithm of the probability of the data given the model parameters.

$$\begin{aligned} w^{\text{MLE}} &= \text{argmax}_w \ \log P(\mathbf{D} \mid \mathbf{w}) \\ &= \text{argmax}_w \ \sum_i P(y_i \mid x_i, \ \mathbf{w}) \end{aligned}$$

Maximum A Posteriori MLE formulation plus a regularization term $P(\mathbf{w})$, which corresponds to a prior on the model parameters.

$$\begin{aligned} w^{\text{MAP}} &= \text{argmax}_w \ \log P(\mathbf{w} \mid \mathbf{D}) \\ &= \text{argmax}_w \ [P(\mathbf{D} \mid \mathbf{w}) + P(\mathbf{w})] \end{aligned}$$

Both of these formulations only learn what is called **point-wise** estimates of the parameters, instead of a full probability distribution, which in general it is very hard to estimate.

Bayesian Prediction

If the distribution over weights is obtained, then output distributions can be computed with the bayesian predictive posterior distribution:

$$P(\mathbf{y} \mid \mathbf{x}) = \int_{\mathbf{w}} P(\mathbf{y} \mid \mathbf{w}, \mathbf{x}) P(\mathbf{w} \mid \mathbf{x}) \, d\mathbf{w}$$

This equation is obtained by marginalizing over all possible model parameters **w**. This equation basically computes predictions **y** with different model parameters **w** and weights them by the probability of those parameters given the input **x**. This can also be considered as a form of Bayesian Model Averaging.

Bayesian Prediction Issues

- The biggest issue with Bayesian learning methods is how to encode or represent the posterior probability distributions over the weights. Since typical usable models have millions of parameters, the distribution is very high-dimensional.
- There is the issue of computational complexity. Integrating over millions of parameters and performing *multiple predictions* for each of these parameters is computationally infeasible.
- In general there are no closed form representations for posterior distribution over weights, and consequently there are also no closed form computations of the Bayesian predictive posterior distribution. The only alternative is to represent the distribution with samples (histograms), and to use Monte Carlo methods to sample from the posterior distribution.

Uncertainty Propagation

The most simple method is, given input values with uncertainty $\mathbf{x} \pm \sigma^2$, the output including uncertainty can be propagated through a non-linear function. This is done with a linear approximation from a taylor series:

$$f(\mathbf{x} \pm \sigma^2) = f(\mathbf{x}) \pm \mathbf{J}\sigma^2\mathbf{J}^T$$

Where $\mathbf{J}$ is the Jacobian matrix of $f(\mathbf{x})$ evaluated on $\mathbf{x}$.

$$\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j}$$

Note that this is just an approximation and it works considerably bad for very non-linear functions. It works better if the step size (the input uncertainty) is small. Better results can be obtained by using higher order taylor approximations.

Uncertainty Propagation

- Note that this is only an approximation, and usually one does not know how much "off" is the approximation from the uncertainty value.
- This method only considers aleatoric uncertainty, and does **not** compute the uncertainty of the model itself (the epistemic one).
- This makes this method only usable in some specific cases, for example if inputs are noisy and one wants to evaluate if predictions are also noisy.

Uncertainty in Random Forests and Ensembles

- Random Forests and Ensembles methods,also have an a uncertainty view related to them.
- Any ensemble method trained on the same dataset can produce direct uncertainty estimations by estimating the variance of the outputs across the ensemble. If each member of the ensemble learns slightly different parameters, then that uncertainty will propagate from the model into the output.
- The quality of the uncertainty estimates depends on: correlations in the training set for each member, number of member in the ensemble, and noise during learning.
- Note that usually to get valuable uncertainty estimates, a large number (over 10-100) ensemble members are needed.

Uncertainty in Random Forests and Ensembles

Regression Uncertainty

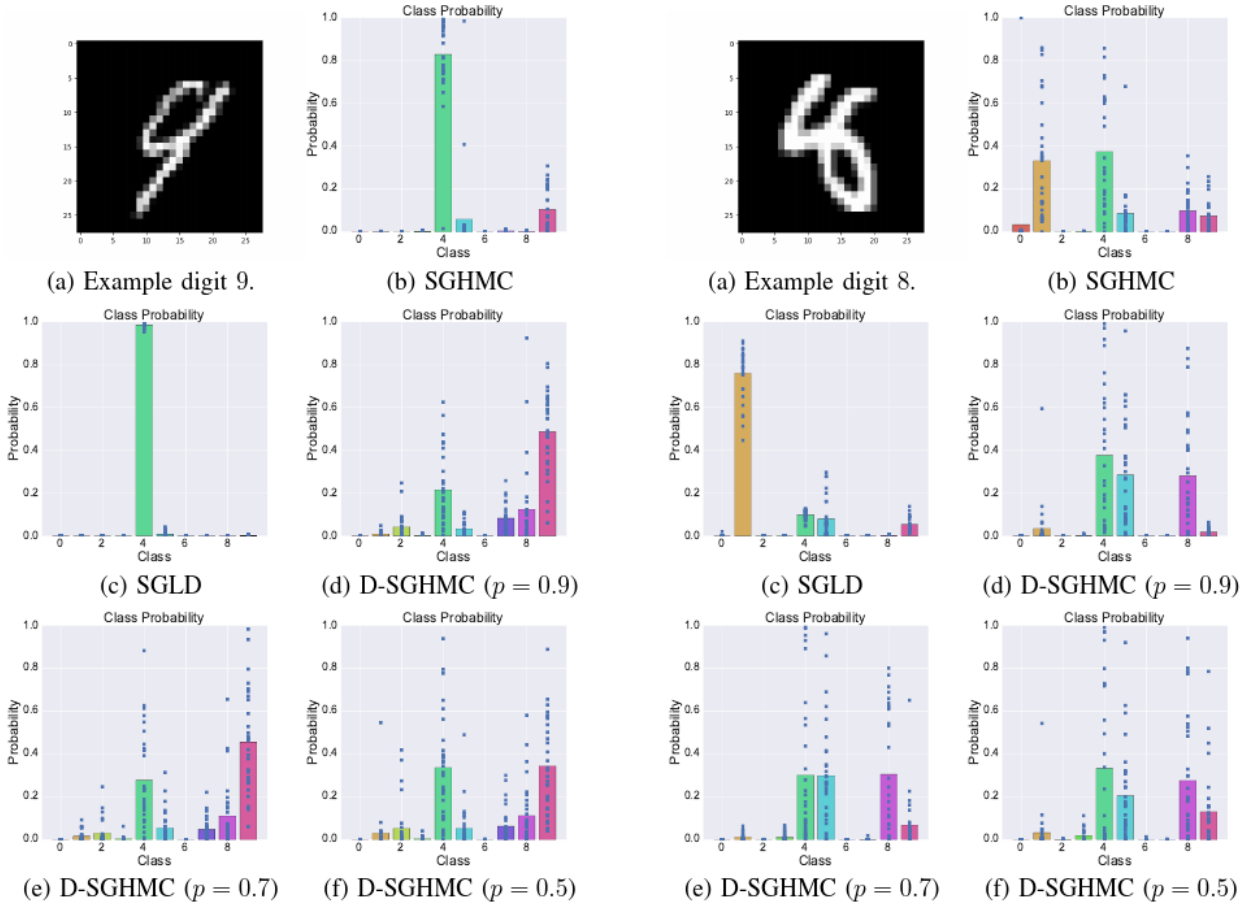
Compute the sample mean and variance from the predictions:

$$\mu(x) = N^{-1} \sum_i f_i(x) \qquad \sigma^2(x) = (N-1)^{-1} \sum_i (f_i(x) - \mu(x))^2$$

Classification Uncertainty

Estimate the distribution over the classes with a histogram. If a classifier with a softmax output is used, then the softmax probabilities can be averaged together.

Uncertainty in Classification with Histograms



(a) Example digit 9. (b) SGHMC (c) SGLD (d) D-SGHMC ($p = 0.9$) (e) D-SGHMC ($p = 0.7$) (f) D-SGHMC ($p = 0.5$)

Fig. 6: Bar plot of probabilities for digit 9 confusing example in MNIST, 1 chain. Squares represent the probability results by class for 30 samples obtained from the predictive distribution, bars shown the mean of those probabilities for each class.

(a) Example digit 8. (b) SGHMC (c) SGLD (d) D-SGHMC ($p = 0.9$) (e) D-SGHMC ($p = 0.7$) (f) D-SGHMC ($p = 0.5$)

Fig. 7: Bar plot of probabilities for digit 8 confusing example in MNIST, 1 chain. Squares represent the probability results by class for 30 samples obtained from the predictive distribution, bars shown the mean of those probabilities for each class.

Learning Uncertainty

- Another common idea is to train a ML estimator with two outputs, one for the prediction, and another for the uncertainty of that prediction.
- For a regression problem, one can use a MLE loss to learn the mean $\hat{y}$ and variance σ^2:

$$L_{\text{MLE}} = 0.5 \sum_i \left[\ln \sigma_i^2 + \frac{(y_i - \hat{y}_i)^2}{\sigma_i^2} \right]$$

- This loss only requires labels for the target outputs (y_i), and no labels for uncertainty are needed. In theory the uncertainty σ^2 balances out between the divisor and logarithm terms.
- No equivalent loss for classification. Sampling through softmax function can be used to regress logits.
- This method generally learns aleatoric uncertainty.

Ensembles

- Recent paper: "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles" by Lakshminarayanan et al. 2017.
- Its a method that uses neural networks and combines three methods we have previously seen.
- Its basically a neural network with two output "heads", one for the prediction and another for the uncertainty.
- The model is trained with:
  1. Adversarial examples as a kind of data augmentation.
  2. Dropout at both training and inference time.
  3. An MLE loss is used for supervision of the uncertainty output.
- Multiple models are trained on the same dataset, and an ensemble is built.

Ensembles

- Ensembles have also powerful uncertainty estimation properties.
- Ensembling consists of training M instances of the same model, but with different randomly drawn initial weights, and then combining their predictions.
- For regression, each ensemble member has two output heads, one for the mean $\mu_i(x)$ and one for the variance $\sigma_i^2(x)$, and a special loss is used for training:

$$-\log p(y_n|\mathbf{x}_n) = \frac{\log \sigma_i^2(\mathbf{x}_n)}{2} + \frac{(\mu_i(\mathbf{x}_n) - y_n)^2}{2\sigma_i^2(\mathbf{x}_n)} + C$$

- This loss is a negative log-likelihood with heteroscedastic variance, the model predicts a variance for each data point.

Ensembles - Combination

Where p_i is the output of the i-th member in the ensemble:

Classification

Ensemble output is average of the probabilities:

$$p_e(y \mid \mathbf{x}) = M^{-1} \sum_i p_i(y \mid \mathbf{x})$$

Ensembles - Combination

Where p_i is the output of the i-th member in the ensemble:

Regression

Ensemble output is a Gaussian mixture model:

$$p_e(y \mid \mathbf{x}) \sim \mathcal{N}(\mu_*(\mathbf{x}), \sigma_*^2(\mathbf{x}))$$

$$\mu_*(\mathbf{x}) = M^{-1} \sum_i \mu_i(\mathbf{x})$$

$$\sigma_*^2(\mathbf{x}) = M^{-1} \sum_i (\sigma_i^2(\mathbf{x}) + \mu_i^2(\mathbf{x})) - \mu_*^2(\mathbf{x})$$

Ensembles - Toy Regression

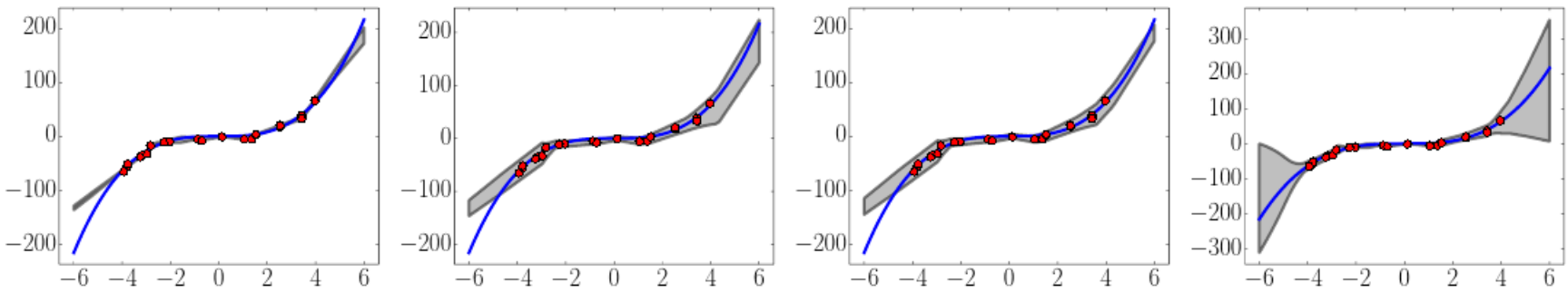
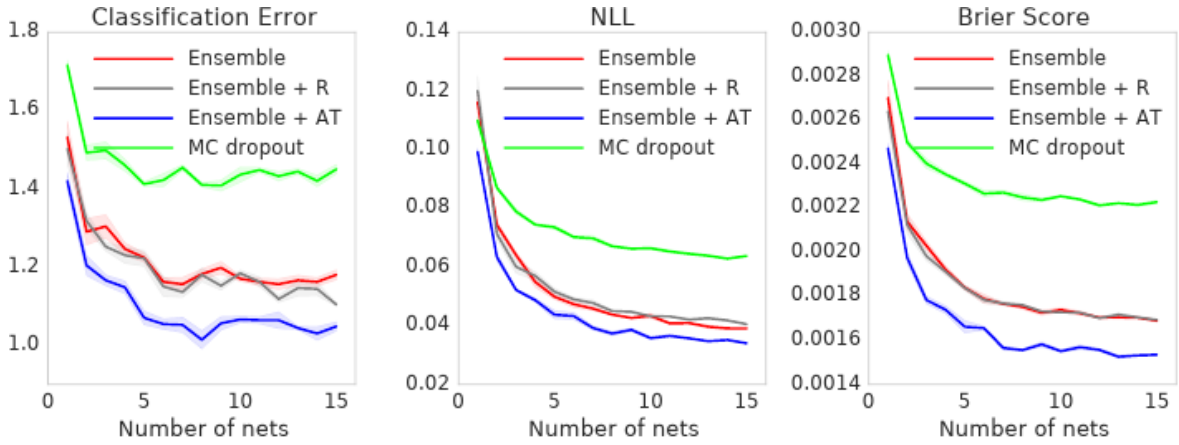
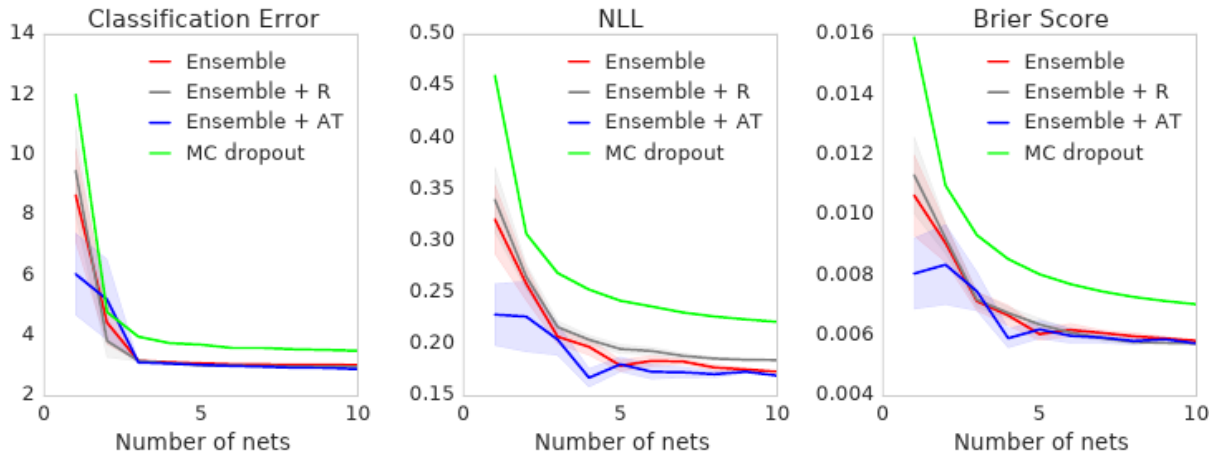


Figure 1: Results on a toy regression task: x-axis denotes x. On the y-axis, the blue line is the *ground truth* curve, the red dots are observed noisy training data points and the gray lines correspond to the predicted mean along with three standard deviations. Left most plot corresponds to empirical variance of 5 networks trained using MSE, second plot shows the effect of training using NLL using a single net, third plot shows the additional effect of adversarial training, and final plot shows the effect of using an ensemble of 5 networks respectively.

Ensembles - Classification

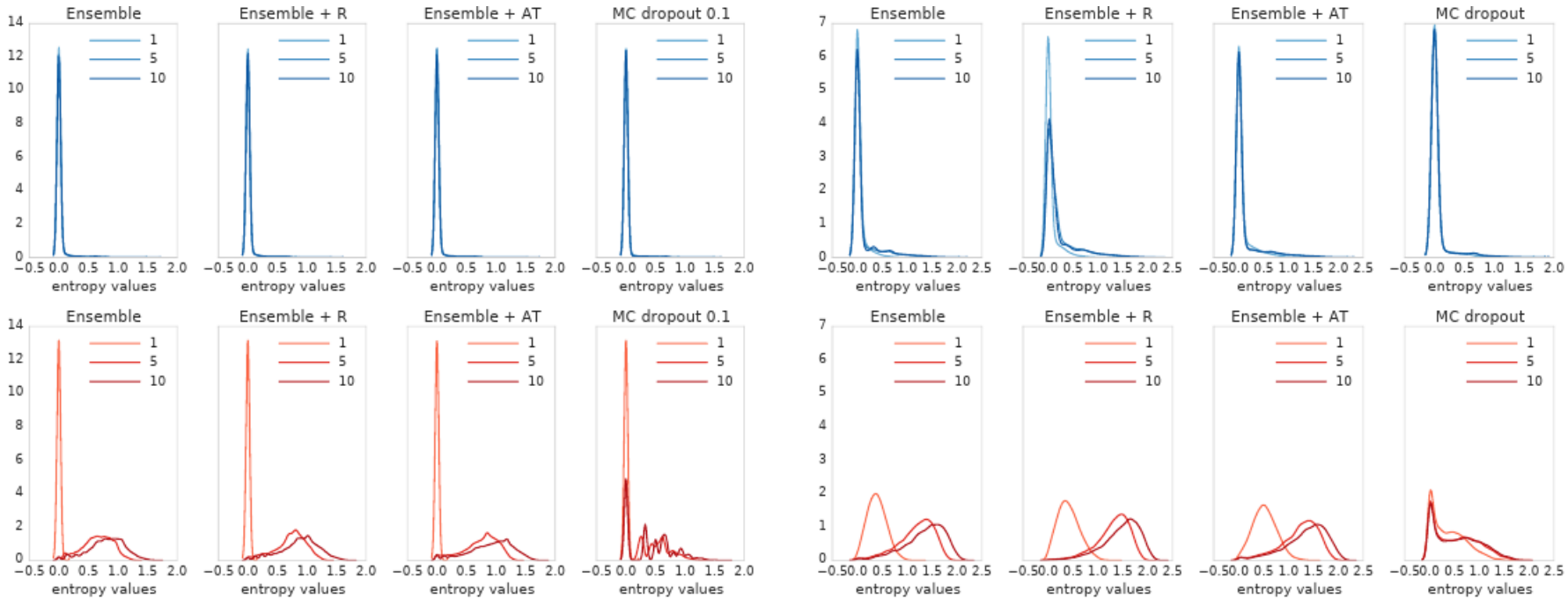


(a) MNIST dataset using 3-layer MLP



(b) SVHN using VGG-style convnet

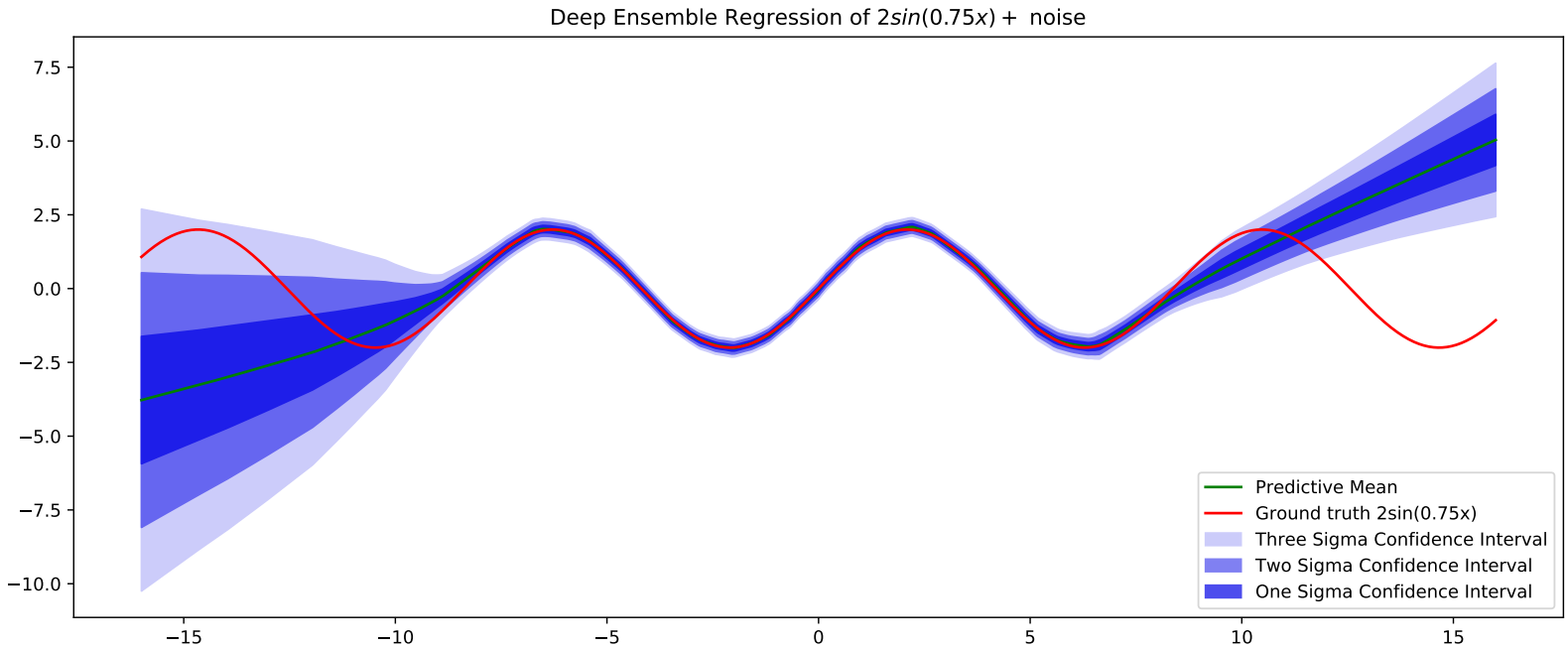
Ensembles - Unseen Examples



(a) MNIST-NotMNIST (b) SVHN-CIFAR10

Figure 3: : Histogram of the predictive entropy on test examples from known classes (top row) and unknown classes (bottom row), as we vary ensemble size M.

Ensembles - Sinusoid Regression



Sampling Methods for Uncertainty Quantification

- These are methods that approximate posterior probability distributions by sampling or by representing these distributions with histograms of samples.
- In general they are expensive, as they have to repeat a number of computations for each sample, but they provide very good estimates of uncertainty.
- Also generally a large number of samples (100-1000) are required to well approximate these posterior distributions, which also adds into the computation time.

Monte Carlo Dropout

- Its a interpretation of applying Dropout in a neural network, but at **inference time**.
- Yarin Gal made theoretical proofs that showed that using Dropout at inference time is approximately equivalent to sampling the predictive posterior distribution, under some assumptions.
- This has connections to Bayesian model averaging, as each forward pass using Dropout samples a different model, which may make different predictions. The mean and variance can be estimated from multiple samples as:

$$\mu(x) = N^{-1} \sum_i f_i(x) \qquad \sigma^2(x) = (N-1)^{-1} \sum_i (f_i(x) - \mu(x))^2$$

Monte Carlo Dropout

- Dropout is a well known technique for regularization of Neural Networks.
- During training, a mask $m_i \sim \text{Bernoulli}(p)$ is drawn and multiplied with the input activations, effectively making some of them zero.
- Dropout can also be enabled at inference time, where it has been proven that it works as an approximation of the predictive posterior distribution.

Monte Carlo DropConnect

- DropConnect is a variation of Dropout, where instead of applying a mask to the activations of a layer, it is applied to the `weights` of a layer.
- It has been proven to also produce an approximation of the predictive posterior distribution. It requires the implementation of new layers that use DropConnect internally.
- In some cases it outperforms MC Dropout in both task and uncertainty performance, but not always.

What about Monte Carlo?

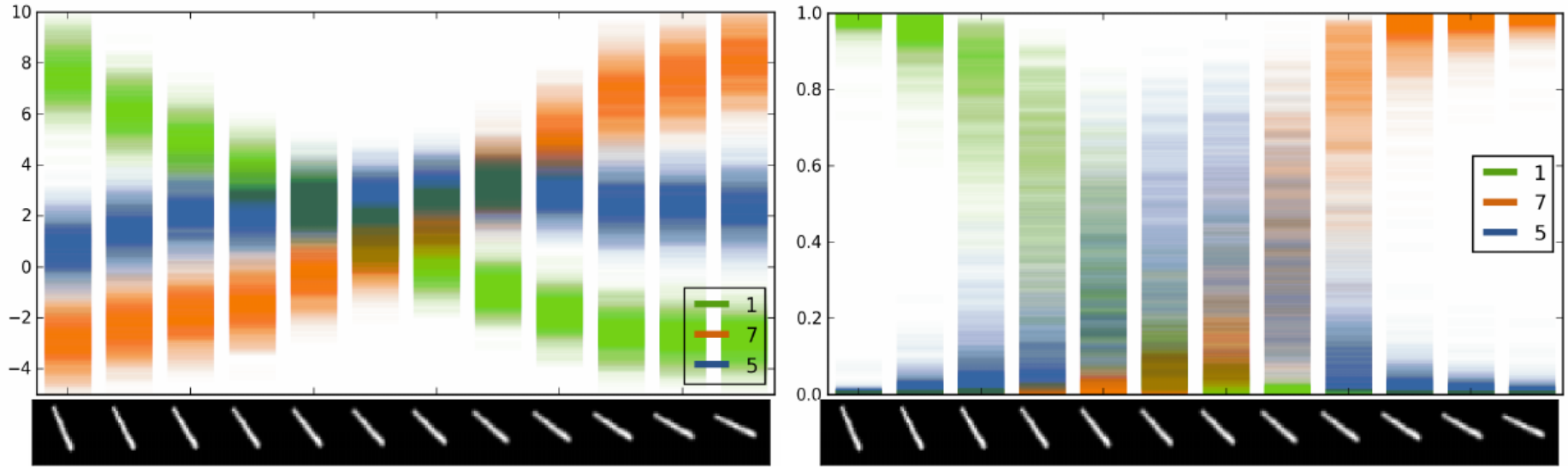
- Enabling Dropout or DropConnect at inference transforms the neural network into a stochastic model.
- This means each forward pass produces a different result, a sample from the predictive posterior distribution.
- The model with uncertainty can be evaluated by combining the predictions from M forward passes.
- This is the Monte Carlo version of the predictive posterior distribution:

$$p(y \mid x) \sim M^{-1} \sum_{i}^{M} p(y \mid x, \theta_i) \quad \text{where} \quad \theta_i \sim \Theta$$

Monte Carlo Dropout

- For classification, histograms or softmax probability averages can be used.
- It requires no modification to the network, other than it must be trained using Dropout.
- Very well received as its simple and theoretically grounded.
- But also do not forget that it is only an approximation. It assumes Gaussian errors, which means it has difficulty predicting distributions with more than one peak.

Monte Carlo Dropout - MNIST Example

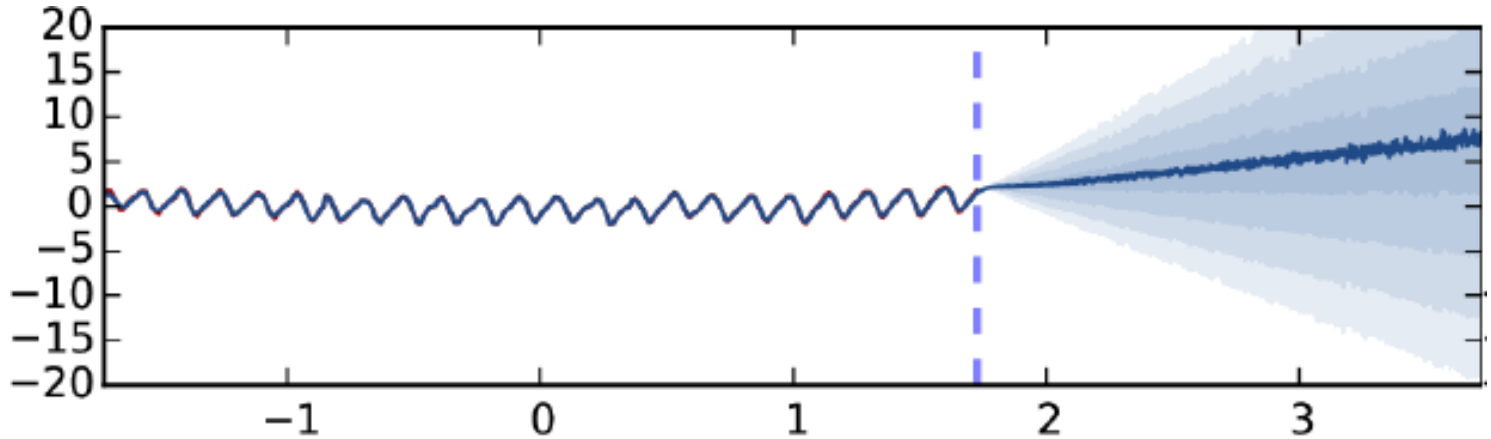


(a) Softmax *input* scatter

(b) Softmax *output* scatter

Figure 4. **A scatter of 100 forward passes of the softmax input and output for dropout LeNet.** On the X axis is a rotated image of the digit 1. The input is classified as digit 5 for images 6-7, even though model uncertainty is extremly large (best viewed in colour).

Monte Carlo Dropout - Regression on Mauna Loa CO2



(c) MC dropout with ReLU non-linearities

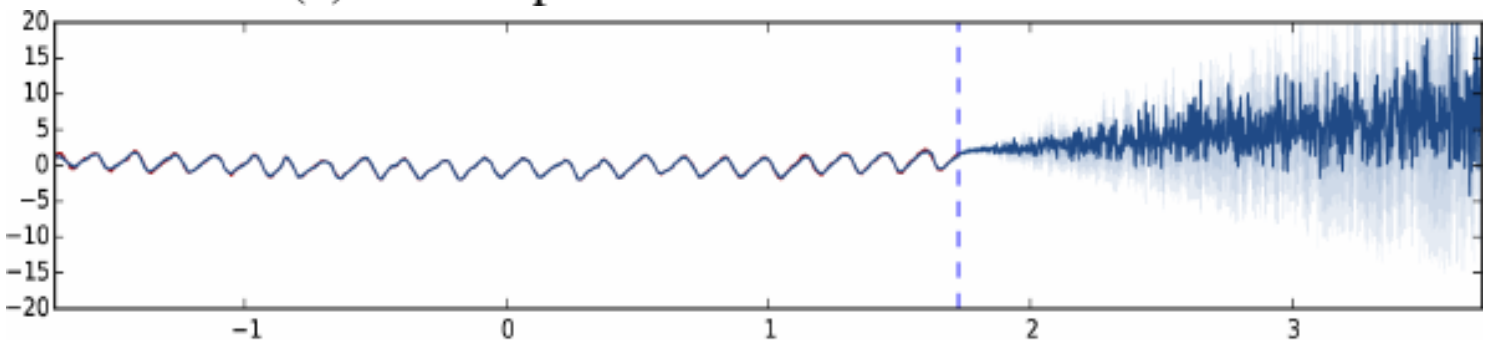
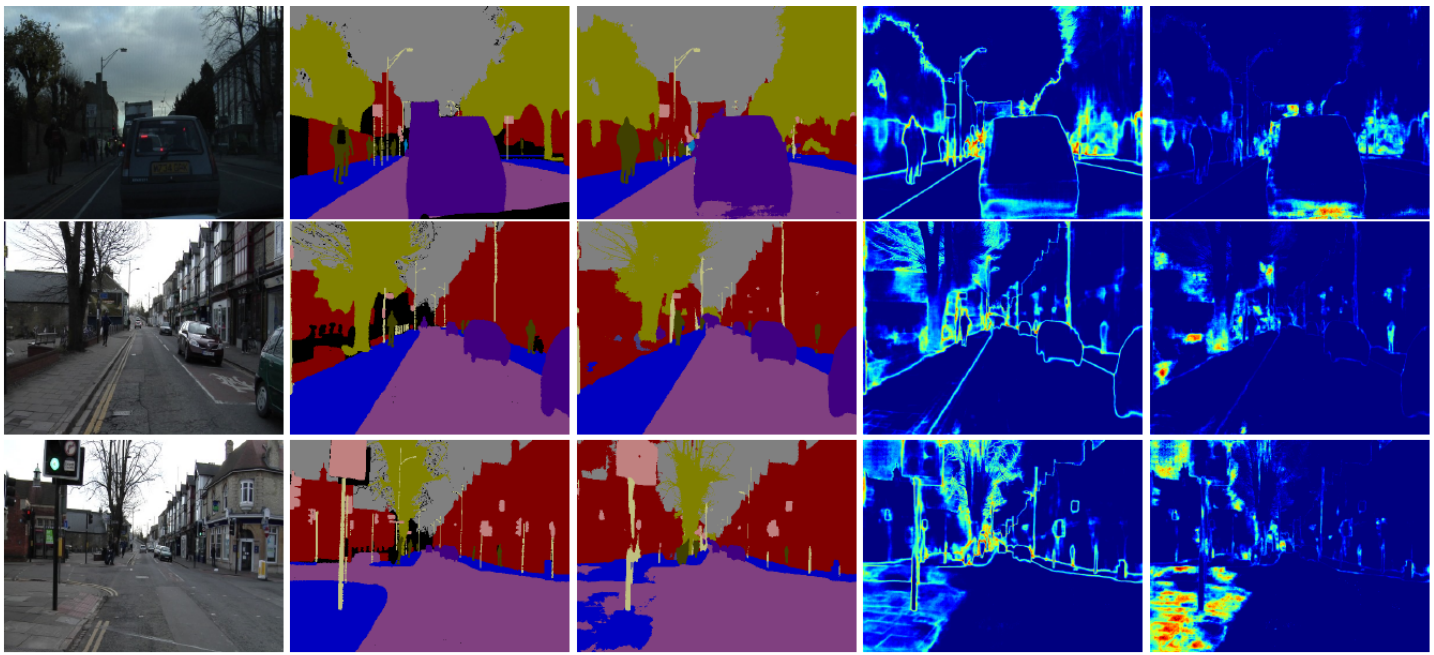


Figure 3. Predictive mean and uncertainties on the Mauna Loa CO_2 concentrations dataset for the MC dropout model with ReLU non-linearities, approximated with 10 samples.

Monte Carlo Dropout - Semantic Segmentation



(a) Input Image (b) Ground Truth (c) Semantic Segmentation (d) Aleatoric Uncertainty (e) Epistemic Uncertainty

Figure 1: **Illustrating the difference between aleatoric and epistemic uncertainty** for semantic segmentation on the CamVid dataset [8]. *Aleatoric* uncertainty captures noise inherent in the observations. In (d) our model exhibits increased aleatoric uncertainty on object boundaries and for objects far from the camera. *Epistemic* uncertainty accounts for our ignorance about which model generated our collected data. This is a notably different measure of uncertainty and in (e) our model exhibits increased epistemic uncertainty for semantically and visually challenging pixels. The bottom row shows a failure case of the segmentation model when the model fails to segment the footpath due to increased epistemic uncertainty, but not aleatoric uncertainty.

Monte Carlo Dropout - Depth Regression

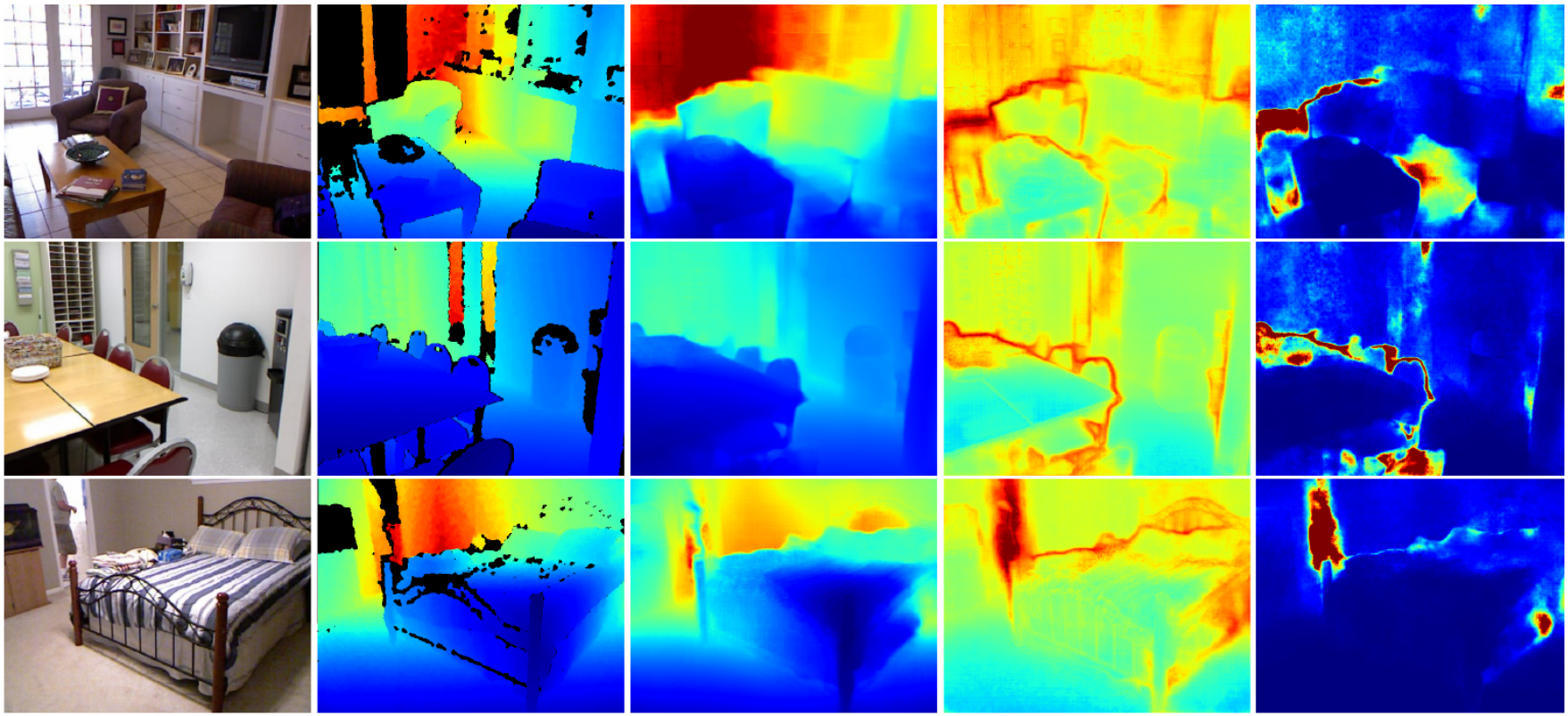
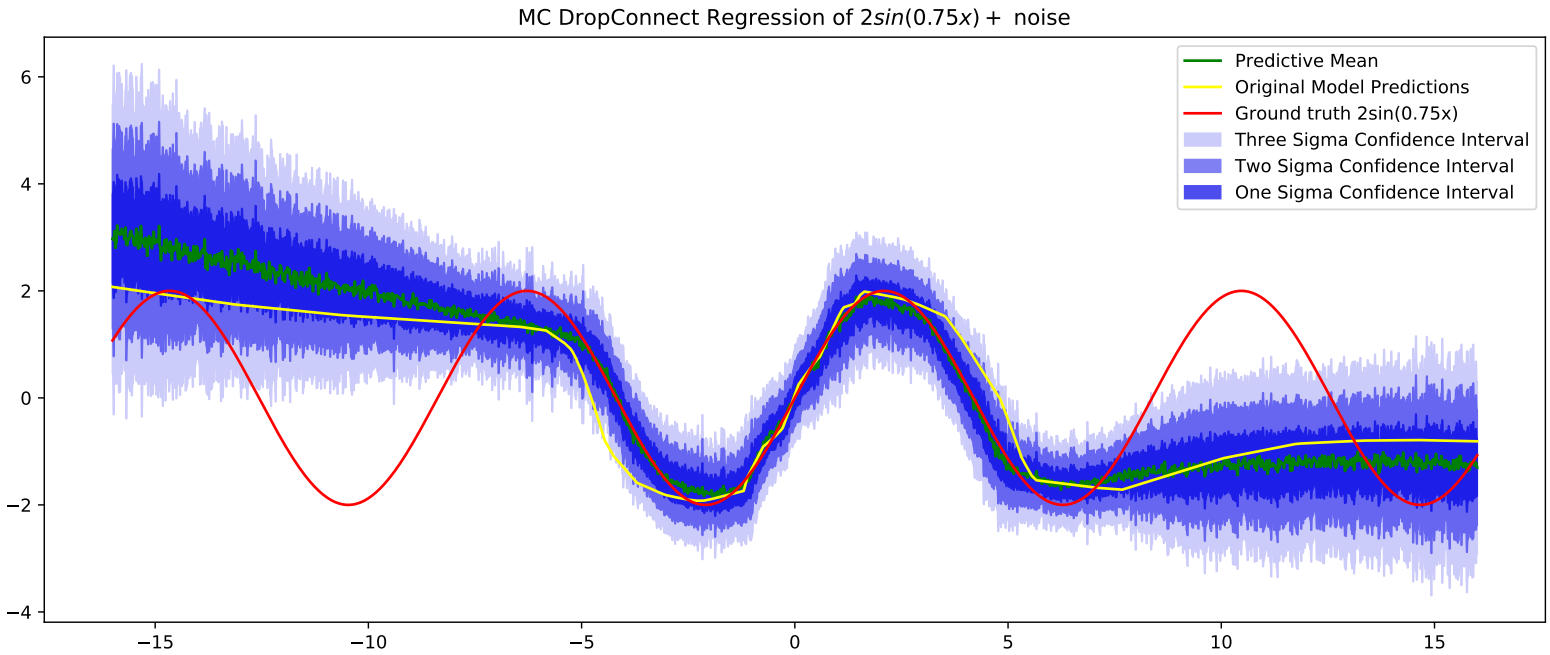


Figure 5: NYUv2 Depth results. From left: input image, ground truth, depth regression, aleatoric uncertainty, and epistemic uncertainty.

MC-DropConnect - Sinusoid Regression



Weight Uncertainty in Neural Networks

- This method approximates the distribution of each weight with a Gaussian one, using a variational approximation.
- The weight distribution is learned automatically from the data, without additional supervision.

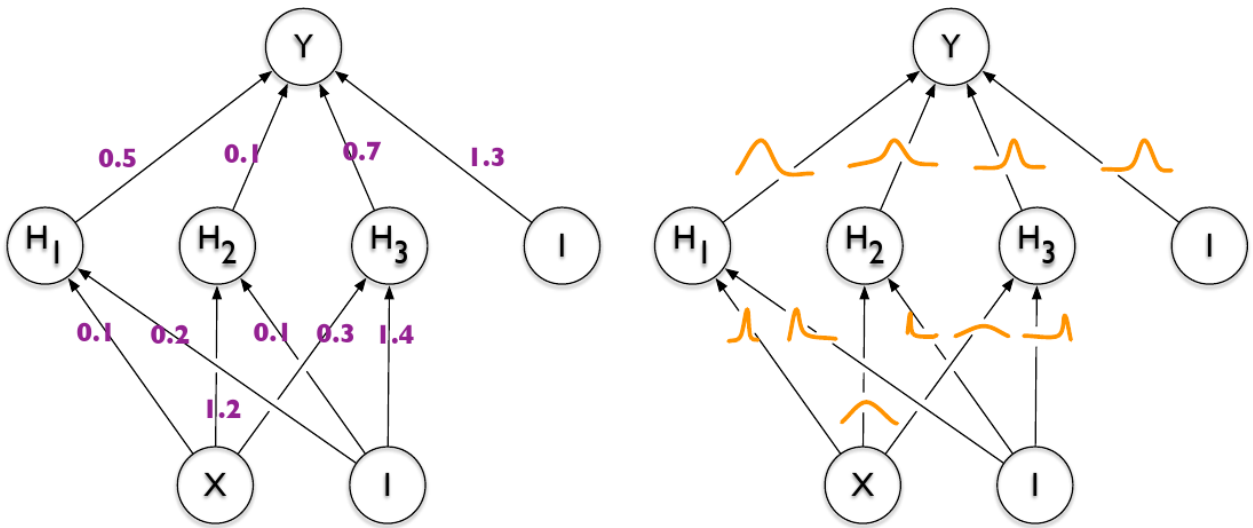


Figure 1. Left: each weight has a fixed value, as provided by classical backpropagation. Right: each weight is assigned a distribution, as provided by Bayes by Backprop.

Weight Uncertainty in Neural Networks

- Each weight in the model is no longer represented as a scalar (floating point number), but as a single Gaussian distribution with parameters $\mathcal{N}(\mu, \rho)$. This is called a variational approximation.
- The parameters of the variational distribution (μ, ρ) are updated using gradient descent.
- But since they weights are now distributions and not actual numbers, the gradient is approximated with a Monte Carlo gradient that is stochastic.
- Prediction outputs can be computed using the Bayesian predictive posterior distribution, but the authors don't specify exactly how it is done. I suspect that they just sampled weights from the variational distributions and performed one forward pass per sample.

Weight Uncertainty in Neural Networks

1. Sample $\epsilon \sim N(0, I)$
2. Compute $\mathbf{w} = \mu + \log(1 + e^{\rho}) \circ \epsilon$
3. Let $\theta = (\mu, \rho)$ and $f(\mathbf{w}, \theta) = \log q(\mathbf{w} \mid \theta) - \log P(\mathbf{w}) P(D \mid \mathbf{w})$ where $q(\mathbf{w} \mid \theta)$ is the parametrized variational distribution on the weights.
4. Compute gradients with respect to mean μ and parametrized standard deviation ρ:

$$\Delta_{\mu} = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \mu} \tag{1}$$

$$\Delta_{\rho} = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\epsilon}{1 + e^{-\rho}} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \rho} \tag{2}$$

5. Update variational distribution parameters:

$$\mu_{n+1} = \mu_n - \alpha \Delta_{\mu} \tag{3}$$

$$\rho_{n+1} = \rho_n - \alpha \Delta_{\rho} \tag{4}$$

Weight Uncertainty in Neural Networks: Regression

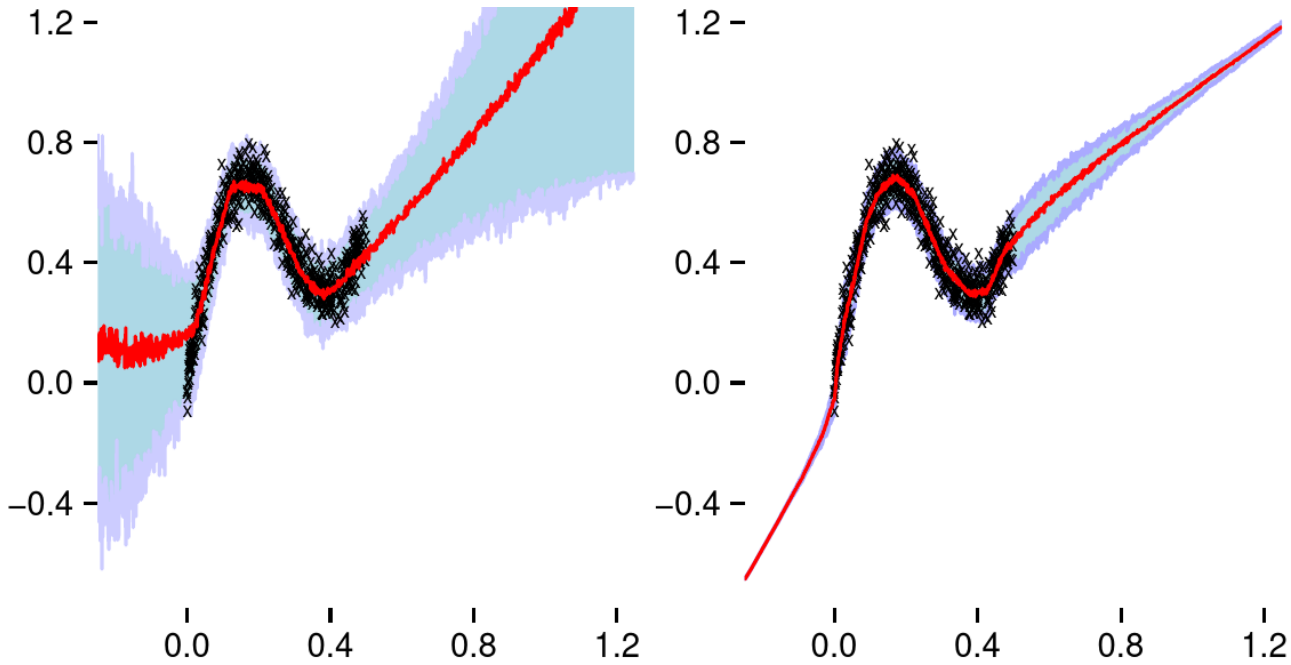


Figure 5. Regression of noisy data with interquatile ranges. Black crosses are training samples. Red lines are median predictions. Blue/purple region is interquartile range. Left: Bayes by Backprop neural network, Right: standard neural network.

Gaussian Processes

- A Gaussian process is the generalization of a Gaussian distribution to the **function space**.
- In other words, it is a stochastic process where the variables form a Multivariate Gaussian Distribution.
- They are considered a non-parametric method, meaning that training a GP actually stored the whole training data, instead of estimating parameters of the model. There is also no closed form equation for the model, it is completely data-driven.
- In order to make predictions, the whole training set is required, making practical use of this method difficult, specially for large datasets.
- For each prediction point/vector, a full distribution is output, which encodes uncertainty of the model.

Gaussian Processes

A Gaussian process is fully defined by its mean and covariance functions, given a real process $f(\mathbf{x})$:

$$m(\mathbf{x}) = E[f(\mathbf{x})] \tag{5}$$

$$k(\mathbf{x}, \mathbf{x}') = E[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))] \tag{6}$$

The covariance function is the most important part of the GP, and it can be considered as a "hyper-parameter" function. It defines how the similarity between data points is computed. Given a dataset x_i, the Covariance matrix is:

$$K = \begin{pmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \cdots & k(x_n, x_n) \end{pmatrix}$$

Gaussian Process Prediction: Noise Free

In the case of a noise-free model, predictions can be done by sampling from the following Multivariate Gaussian distribution. Assuming that the training inputs are x and targets are y.

$$P(\mathbf{y}_* \mid \mathbf{x}) \sim \mathcal{N}(K_* K^{-1} \mathbf{y}, K_{**} - K_* K^{-1} K_*^T)$$

Where we recognize $K_* K^{-1} \mathbf{y}$ is the mean, and $K_{**} - K_* K^{-1} K_*^T$ is the covariance matrix of the Gaussian distribution. Terms involving a star (*) are evaluated on the "test set" as follows:

$$K_* = \begin{pmatrix} k(x_*, x_1) & k(x_*, x_2) & \cdots & k(x_*, x_n) \end{pmatrix} \qquad K_{**} = K(x_*, x_*)$$

Gaussian Process Prediction: Noisy Observations

In the case of noisy observations $y = f(x) + \varepsilon$, assuming independent and identically distributed noise with $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. The predictive Gaussian Process $\mathcal{N}(\bar{\mathbf{f}}_*, \text{COV}(\mathbf{f}_*))$ then has mean $\bar{\mathbf{f}}_*$ and covariance $\text{COV}(\mathbf{f}_*)$ given by:

$$\begin{aligned}\bar{\mathbf{f}}_* =& K(X_*, X)\left[K(X, X) + \sigma^2 I\right]^{-1}\mathbf{y} \\ \text{COV}(\mathbf{f}_*) =& K(X_*, X_*) - K(X_*, X)\left[K(X, X) + \sigma^2 I\right]^{-1}K(X, X_*)\end{aligned}$$

In the case there is a single test point it can be simplified to:

$$\begin{aligned}\bar{\mathbf{f}}_* =& k_*^T(K(X, X) + \sigma^2 I)^{-1}\mathbf{y} \\ \text{COV}(\mathbf{f}_*) =& K(x_*, x_*) - k_*^T(K(X, X) + \sigma^2 I)^{-1}k_*\end{aligned}$$

Where $k_* = \begin{pmatrix} k(x_*, x_1) & k(x_*, x_2) & \cdots & k(x_*, x_n)\end{pmatrix}$.

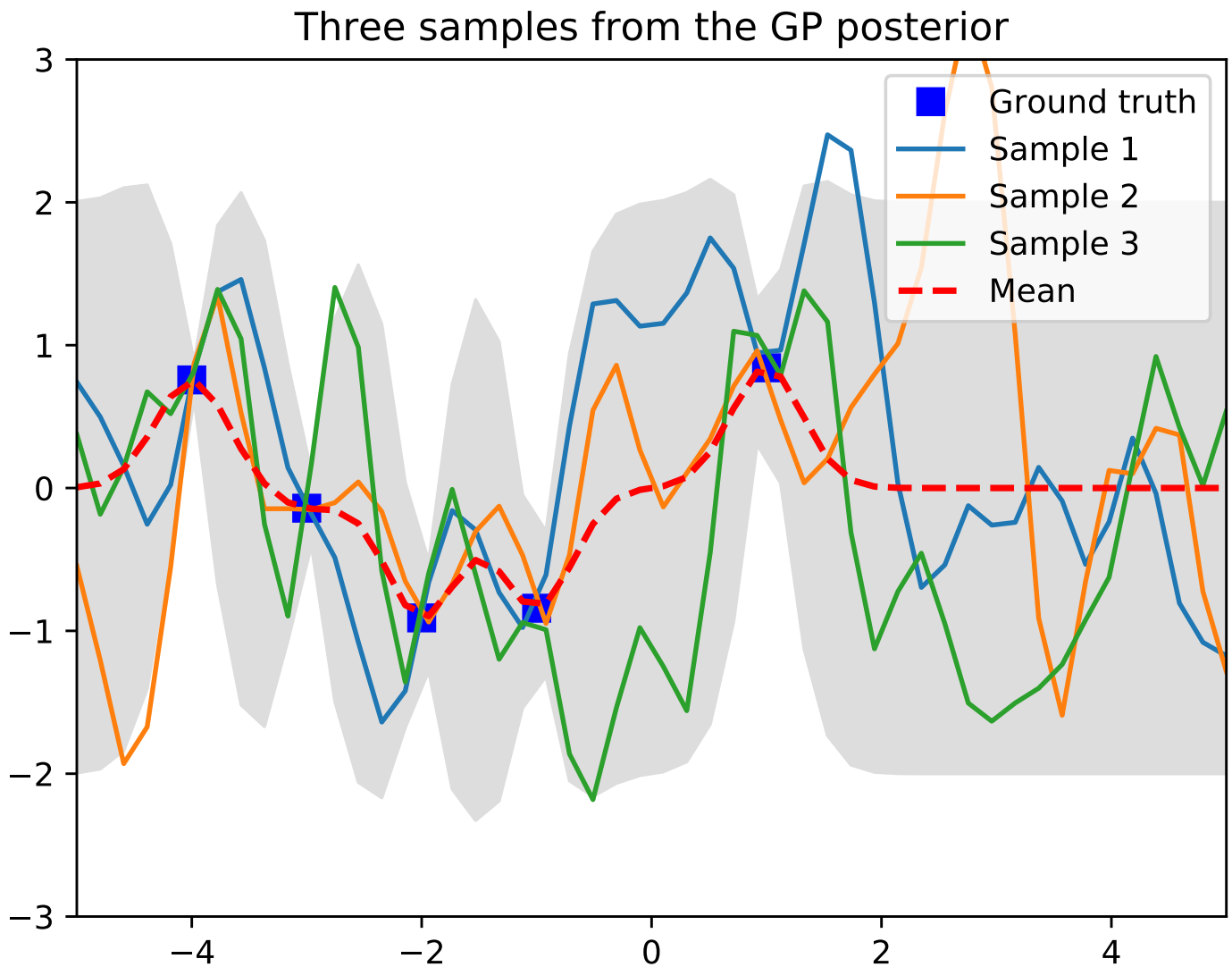
Gaussian Process Prediction: Linear Interpretation

The previous equations for the predictive mean can be seen as a linear combination of the kernel/covariance functions:

$$\bar{f}(x_*) = \sum_{i=0}^{n} \alpha_i k(x_i, x_*)$$

Where the vector $\alpha = (K(X, X) + \sigma^2 I)^{-1}\mathbf{y}$. Also note that the mean of the Gaussian Process depends on the training targets $\mathbf{y}$, but the covariance function does not depend on the target values. Assuming that a process has zero mean (for example, normalizing by substracting the mean), then a Gaussian Process can be completely defined by its covariance COV$(\mathbf{f}_*)$.

Samples from a Gaussian Process

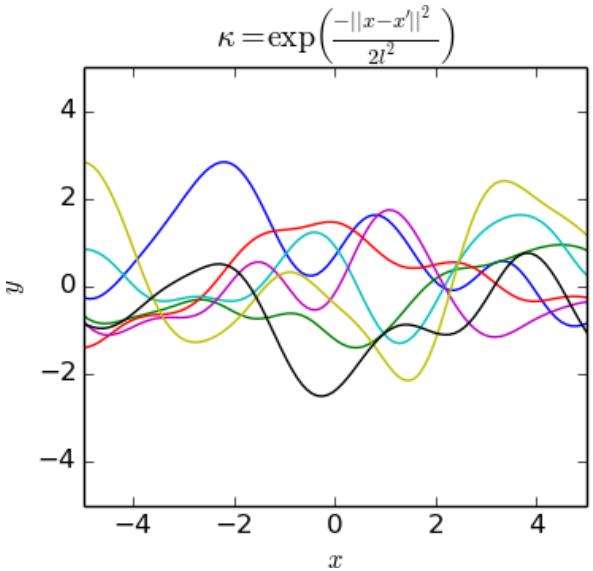


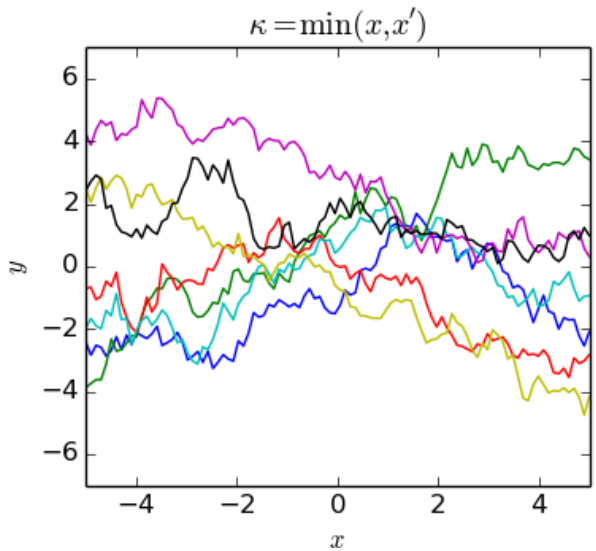
Kernel/Covariance Functions for Gaussian Processes

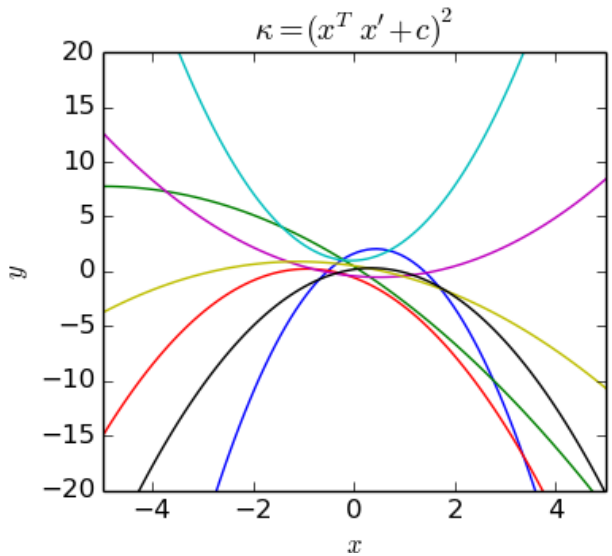
A kernel or covariance function is any function that produces a valid covariance matrix (that is, positive semi-definite). Some examples of Covariance functions are:

- **Linear**: $K_L(x, x') = x^T x'$
- **Squared Exponential**: $K_{SE}(x, x') = e^{-\frac{|x-x'|^2}{2\sigma^2}}$
- **Ornstein-Uhlenbec**: $K_{OU}(x, x') = e^{-\frac{|x-x'|}{\sigma}}$
- **Periodic**: $K_P(x, x') = e^{-\frac{2\sin^2(0.5|x-x'|)}{\sigma^2}}$
- **Rational Quadratic**: $K_{RQ}(x, x') = (1 + |x - x'|^2)^{-\alpha}$

Effect of Different Kernel/Covariance Functions







Classification with Gaussian Processes

- So far we have only mentioned regression with a GP.
- Classification can be easily performed by regressing the logits that can be given to a sigmoid (for binary classification) or softmax function (for multi-class classification).
- But then there are no observations for the logits, only for the class labels. The exact values of the logits are not important, just that they produce the right class labels and uncertainty.
- We won't cover now how this is exactly done, as it is a bit complicated and mathematical.

Learning Performance

The first thing is to define how to measure the performance of a learning model.

Losses

Objective function that guides learning during the optimization process. It defines the task to be learned and the quality of solutions. Usually it has to be differentiable.

Metrics

Measurements of quality that let the ML developer evaluate the learning process' success. Usually non-differentiable. Losses can be used as Metrics as well.

Probabilistic Classifiers

Most classifiers output a probability vector p of length C. The class integer class index c can be recovered by:

$$c = \text{argmax}_i \, p \tag{7}$$

Note that for C classes, their indices go from 0 to $C - 1$. For binary classification, only a single probability is required:

$$f(x) = P(y = 1) = 1 - P(y = 0) \tag{8}$$

In this case, the classifier outputs the probability of class 1 (usually the positive class), while the probability for class 0 (the negative class) can be recovered by subtracting with one.

Loss Functions - Classification

Categorical Cross-Entropy

For this loss, labels y^c should be one-hot encoded. Used for multi-class classification problems, where the model predictions are $\hat{y_i}^c$ are class probabilities that sum to 1.

$$L(y, \hat{y}) = -\sum_i \sum_c y_i^c \log(\hat{y}_i^c)$$

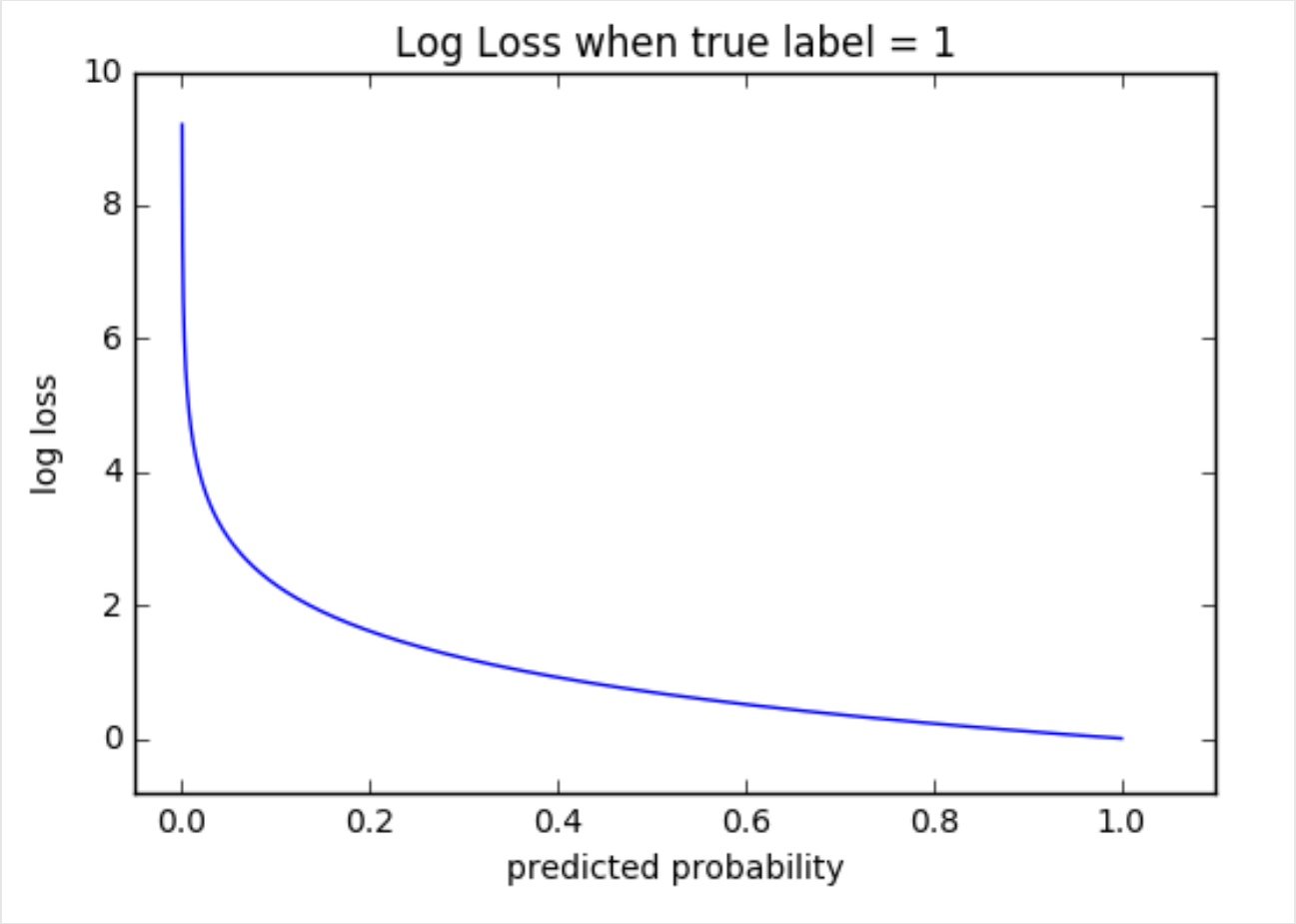
Binary Cross-Entropy

Used for binary classification problems with labels $y_i \in \{0, 1\}$

$$L(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

Loss Functions - Classification

Binary Cross-Entropy



Loss Functions - Negative Log-Likelihood

Log-likelihoods are a family of losses or objective functions. For regression with uncertainty, the following loss is commonly used.

$$-\log p(y_n|\mathbf{x}_n) = \frac{\log \sigma_i^2(\mathbf{x}_n)}{2} + \frac{(\mu_i(\mathbf{x}_n) - y_n)^2}{2\sigma_i^2(\mathbf{x}_n)} + C \tag{9}$$

Here the model outputs two variables. A mean $\mu(x)$ and variance $\sigma^2(x)$, and these are weighted. If the model is uncertain (large σ^2) then the squared error is ignored but the logarithm of variance counteracts this effect, and if the model is certain (small σ^2), then the opposite effect happens.

This loss assumes that the model outputs the parameters of a Gaussian distribution.

Loss Functions - Others

Kullback-Leibler Divergence

Distance measure between probability distributions p and q. The Cross-Entropy is a simplified version of this loss (with some assumptions).

$$L(p,q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)} dx\right) \qquad L(y,\hat{y}) = \sum_i y_i \log\left(\frac{y_i}{\hat{y}_i}\right)$$

Loss Functions with Uncertainty

Some special loss functions that I mentioned here do consider uncertainty.

- **Cross Entropy**. Special case of NLL for classification, also considers the probabilities/confidences of the correct class.
- **Gaussian NLL**. Special case of NLL for regression with Gaussian distributed output. Models uncertainty through the variance output.
- **KL Divergence**. General case of many losses, measures distance between probability distributions, which implicitly models uncertainty.

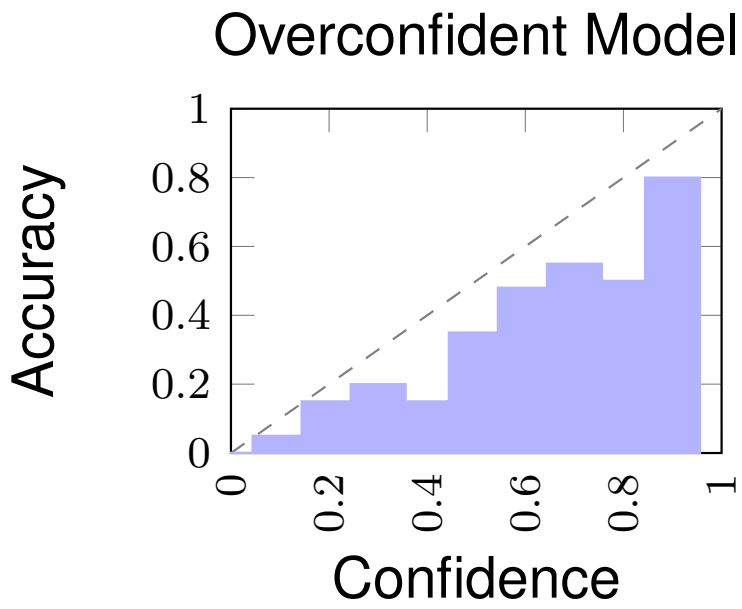
Calibration

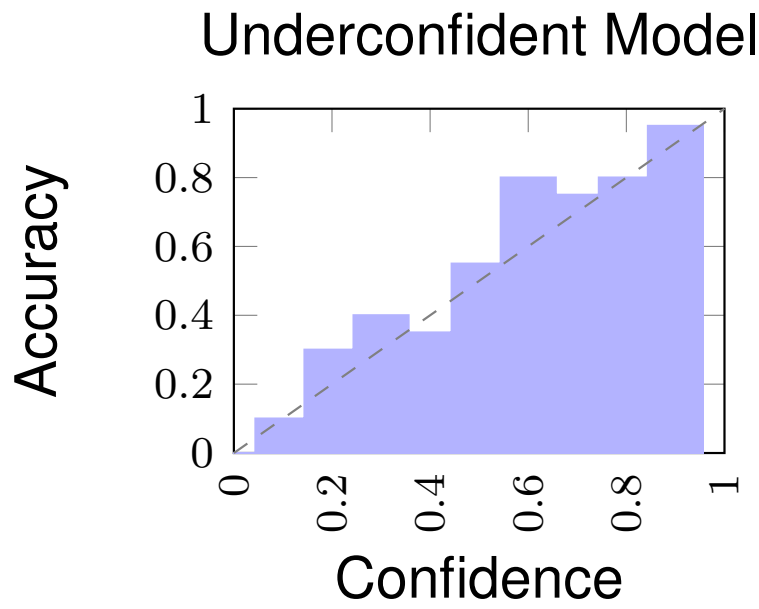
- We talked about a concept that indicates how much we can trust the confidence of a model.
- This can be formalized by comparing task performance (such as accuracy) as the confidence of predictions change.
- For example, if a prediction is made with 10% confidence, then we expect that such predictions will be correct 10% of the time.
- And correspondingly, if a prediction is made wit 90% confidence, then only 10% of such predictions will be incorrect.

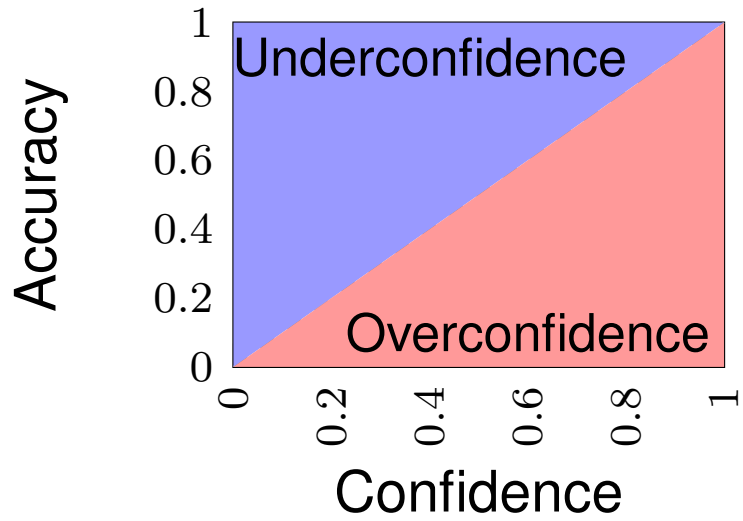
Calibration - Reliability Plots

- Calibration can be observed by making a Reliability plot.
- We take the predictions of a model over a dataset, divide the predictions by confidence values conf(B_i) into bins B_i, for each bin the accuracy acc(B_i) is computed, and then the values $(\text{conf}(B_i), \text{acc}(B_i))$ are plotted.
- Regions where conf$(B_i) <$ acc(B_i) indicate that the model is underconfident, while regions conf$(B_i) >$ acc(B_i) indicate overconfidence.
- The line conf$(B_i) =$ acc(B_i) indicates perfect calibration.

Calibration - Reliability Plots







Calibration - Metrics

Calibration Error

This is the standard metric to measure miscalibration. It is affected by variations in the number of samples in each bin.

$$\text{CE} = \sum_i |\text{acc}(B_i) - \text{conf}(B_i)|$$

Calibration - Metrics

Expected Calibration Error

In order to compensate for the varying number of elements in each bin B_i, ECE weights the error produced by each bin by the proportion of samples in that bin with respect of the total of samples.

$$\text{ECE} = \sum_i N^{-1} |B_i| \, |\text{acc}(B_i) - \text{conf}(B_i)|$$

This produces a metric that is much more stable and less prone to outliers. This is usually the metric that is commonly reported in scientific publications.

Calibration - Metrics

Maximum Calibration Error

For risk averse applications, the MCE computes the maximum level of miscalibration, so appropriate risk can be estimated and addressed by the application designed. Variations in miscalibration in each bin can hide in the overall mean or expectation.

$$\text{MCE} = \max_i |\text{acc}(B_i) - \text{conf}(B_i)|$$

For example, in Autonomous Driving, the maximum miscalibration should be close to zero.

Calibration - Reliability Plots with MC-Dropout on MNIST

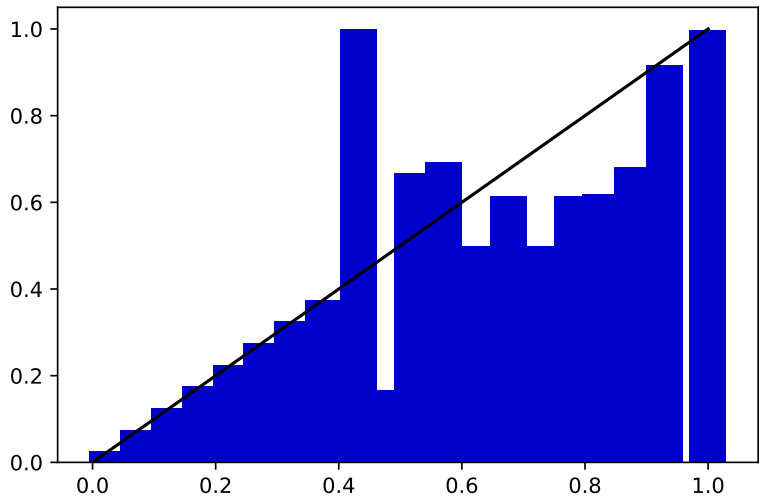


Figure 1: Classical NN, Calibration error is 0.18

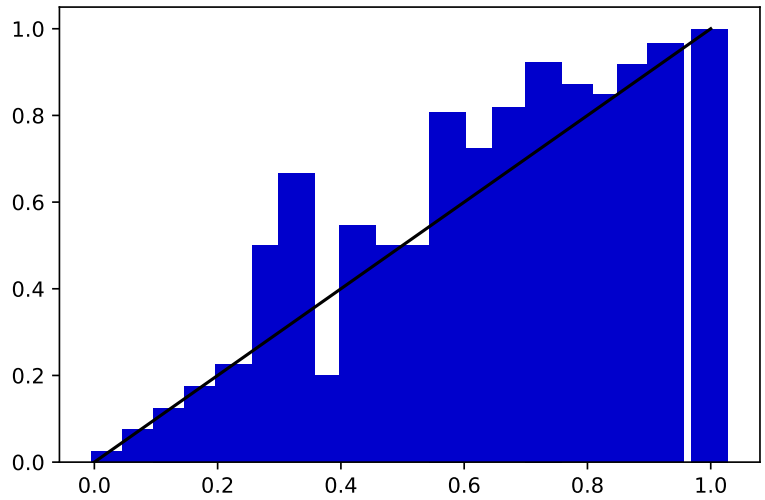


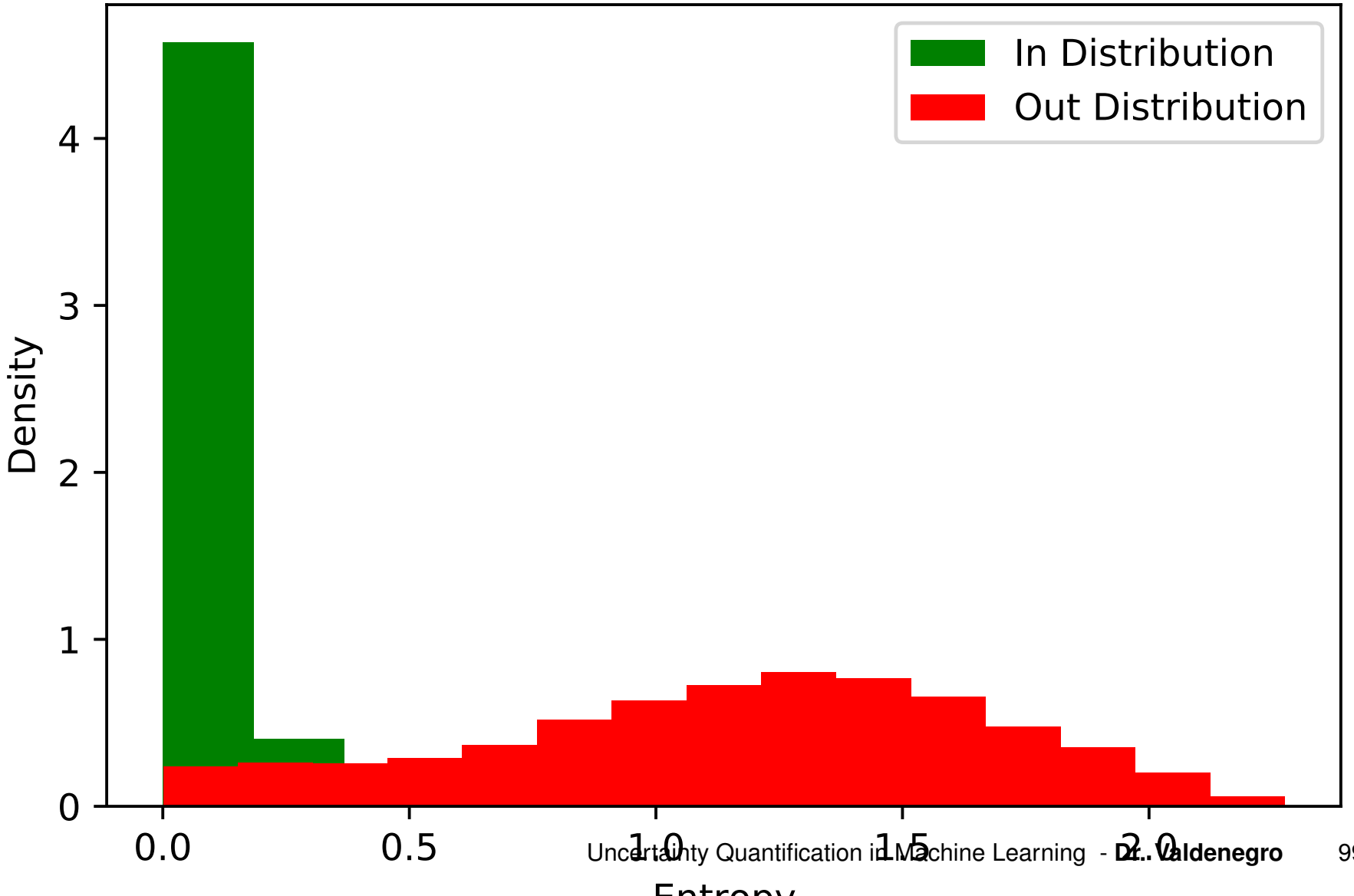
Figure 2: Bayesian NN with MC-Dropout, Calibration error is 0.11

Out of Distribution Detection

- It is the task of detecting when the input to the model is outside of the distribution of the training set used to train the model.
- This corresponds to the model rejecting to provide an output if its uncertain about it.
- Doing this is simple, reject to consider a model's output if the uncertainty is too large. The trick is to select an appropriate threshold.
- For regression, the standard deviation of the output can be used. For classification, entropy is preferred:

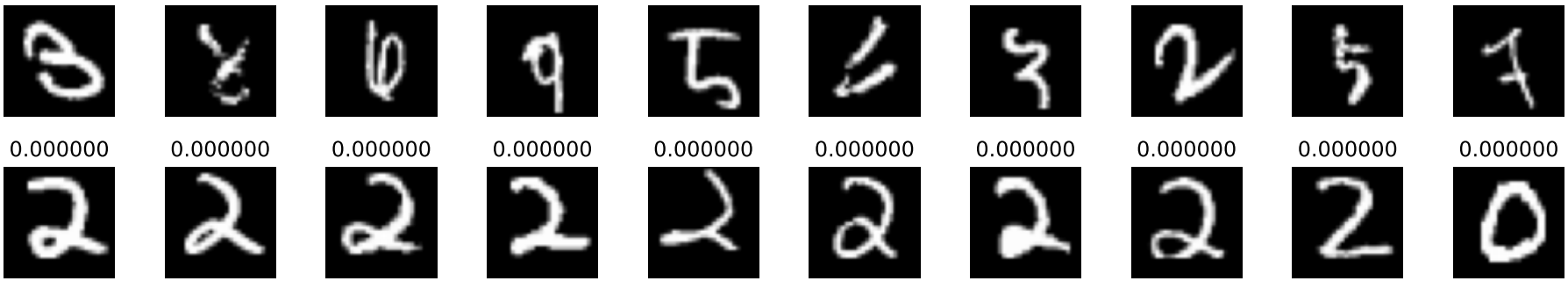
$$H(p(x)) = -\sum_i p(x)_i \log p(x)_i$$

Out of Distribution Detection - MNIST vs Fashion MNIST

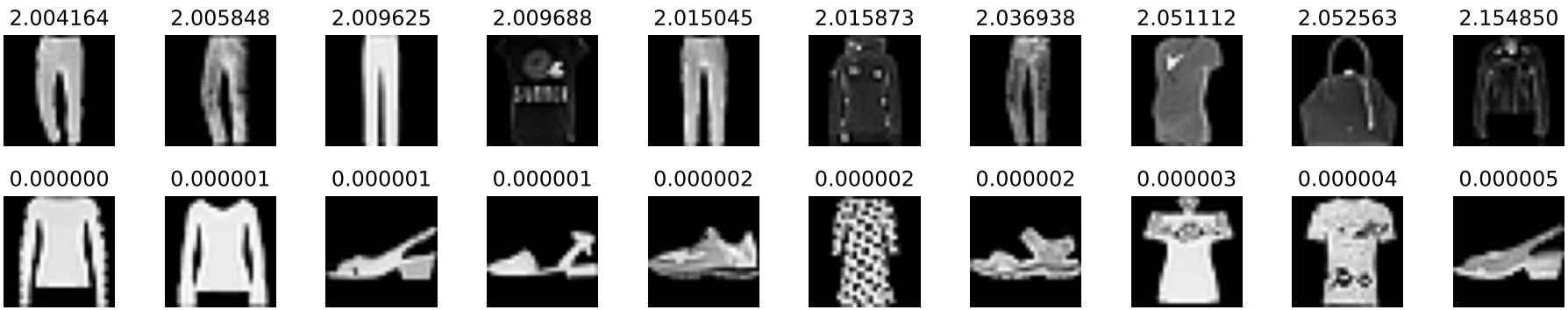


Out of Distribution Detection - MNIST vs Fashion MNIST

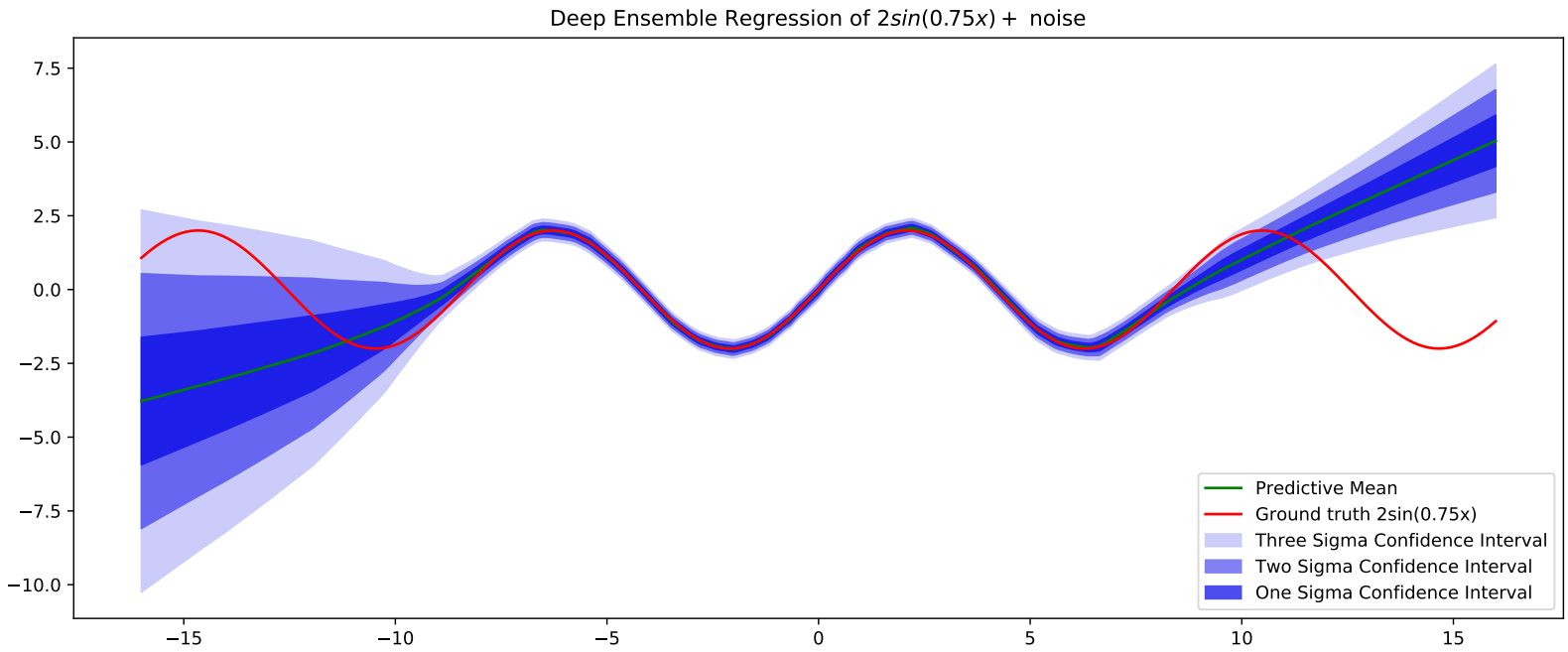
MNIST



Fashion MNIST



Out of Distribution Detection - Sinusoid Regression with Ensembles



In this example, the training set is $x \in [-8, 8]$, it can be visually seen that outside this range the standard deviation of the output (uncertainty) increases considerably, and increases as with the distance to that range.

Out of Distribution Detection - Pitfalls

- It is not easy to completely separate ID and OOD examples, as some ID examples have still high uncertainty, and sometimes OOD examples have low uncertainty. This is due to variability in classes.
- Choosing a threshold is not easy, as lots of analysis has to be performed.
- Unfortunately there are no guarantees on OOD performance, and there are known cases of bad effects. (See Ovadia et al.)
- Uncertainty should be used as additional information from where further human analysis can be decided, instead of enabling fully automatic processing.

Sub-Ensembles [Valdenegro. 2019]

- A great problem with Ensembles is that computational costs increase linearly with the number of members in the ensemble.
- A basic question is: Is it necessary that all ensemble members be independent? Can weights be shared across ensemble members?
- Turns out the answer is no and yes, weights on layers from the input can be shared, and last layers in the network ensembled, and this works as an approximation of the full ensemble.

Sub-Ensembles [Valdenegro. 2019]

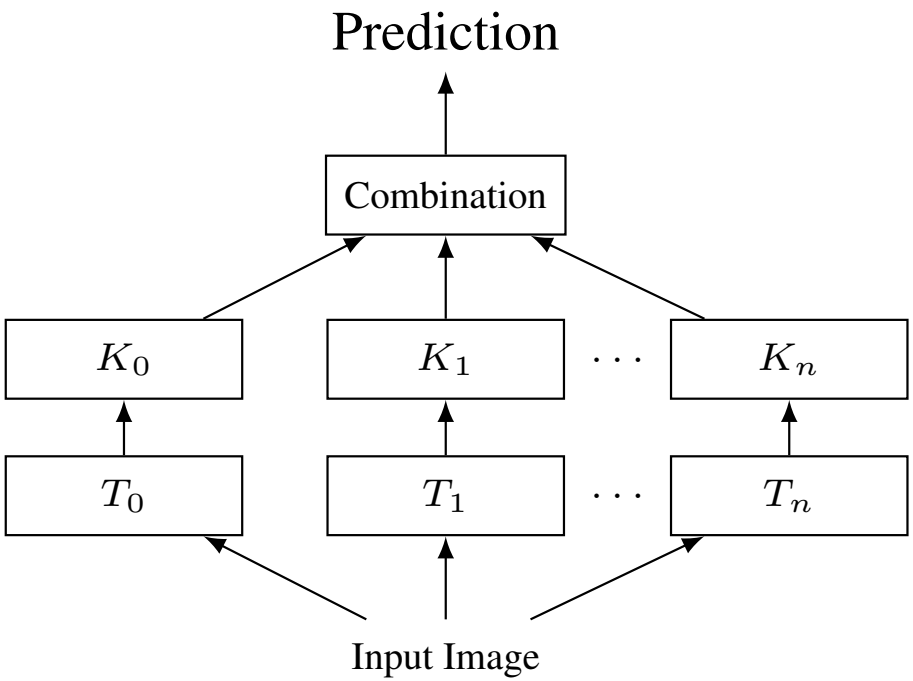


Figure 3: Ensemble

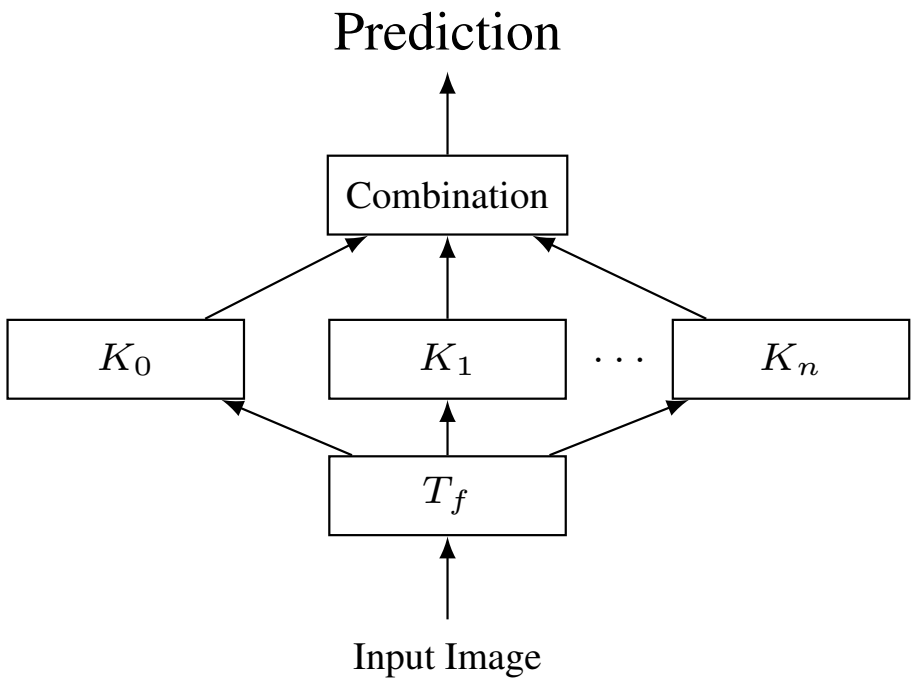
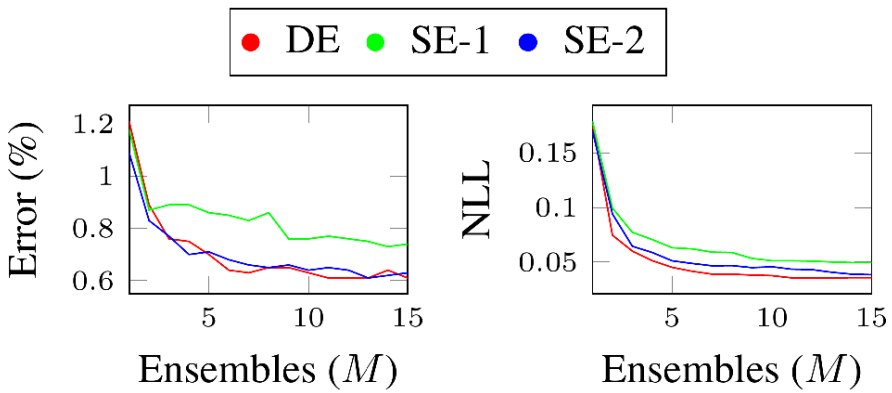
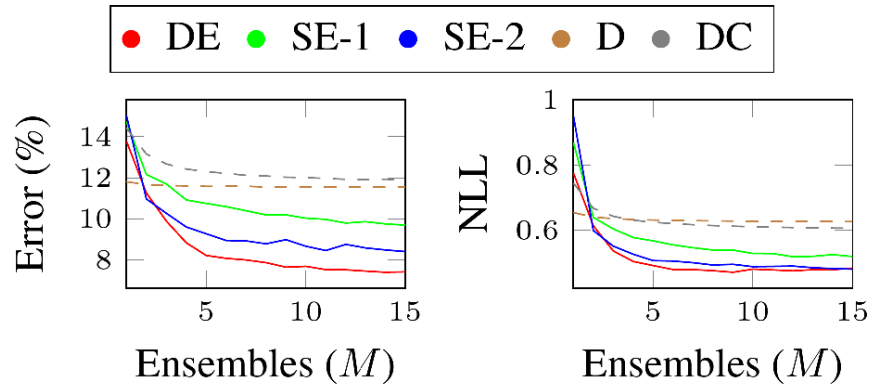


Figure 4: Sub-Ensemble

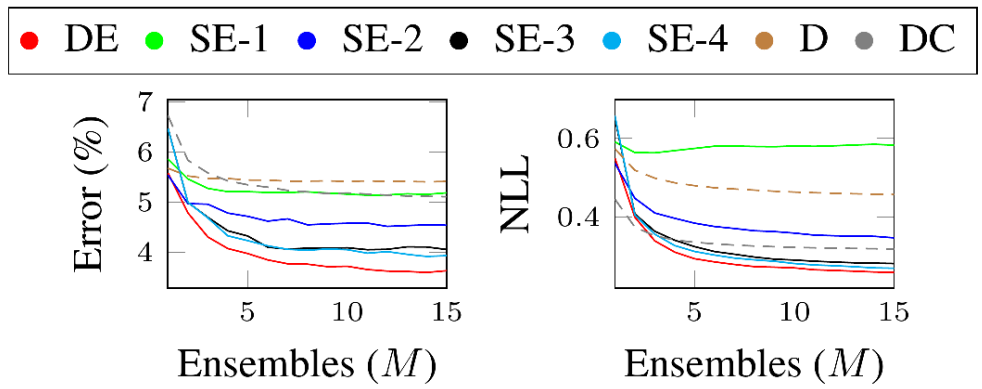
Sub-Ensembles - Performance



(a) MNIST



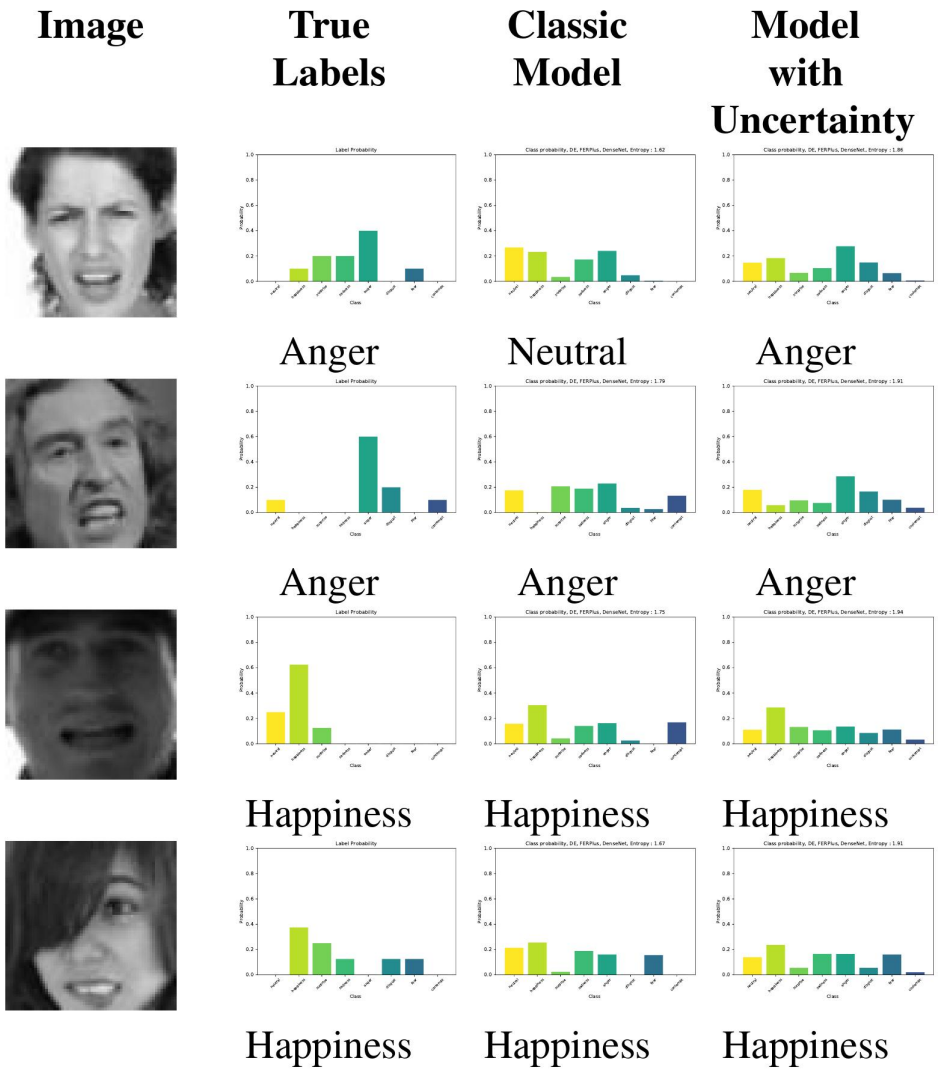
(b) CIFAR10



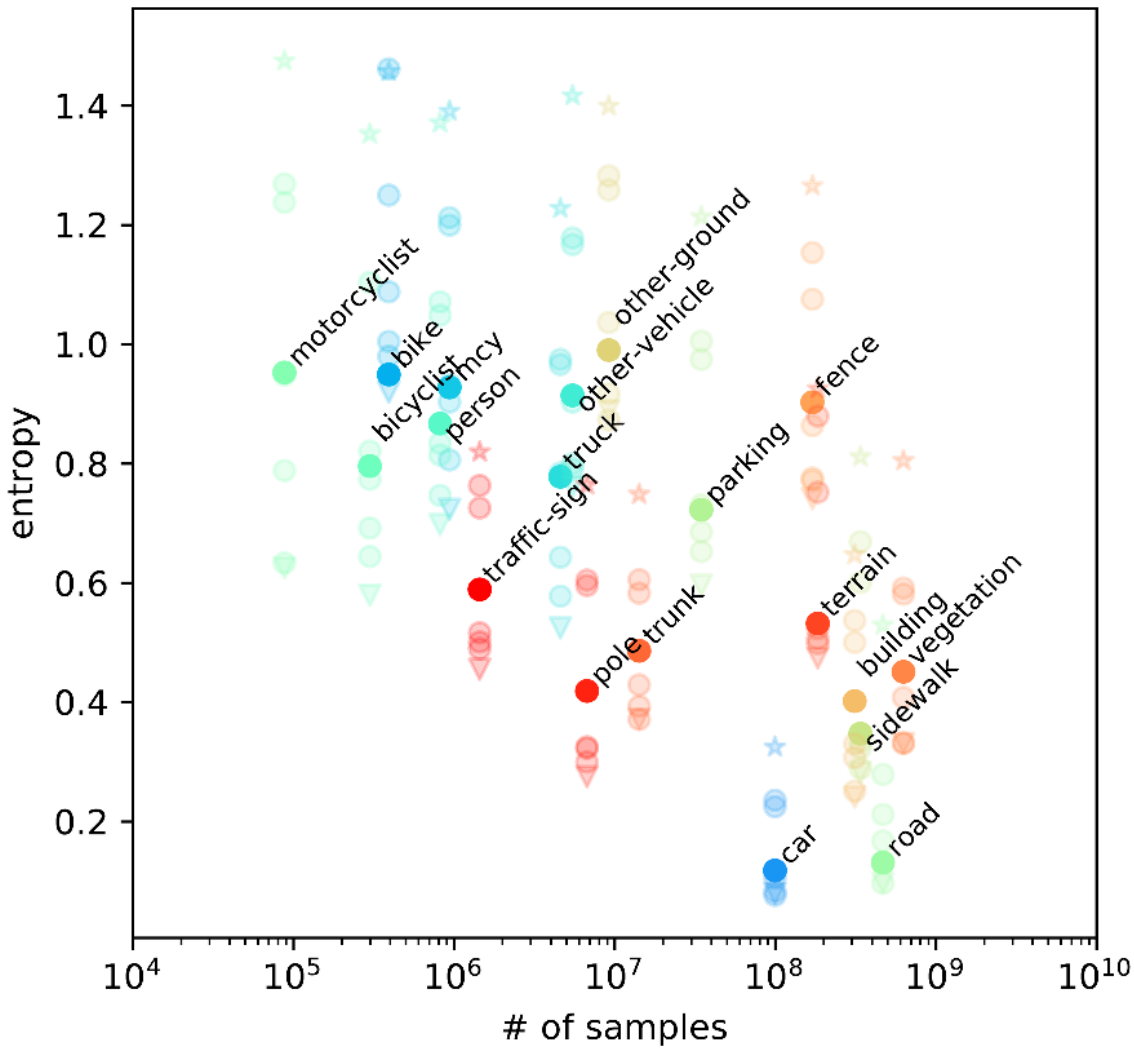
Results on SVHN using a batch normalized VGG-like network

Presented at the Bayesian Deep Learning Workshop @ NeurIPS 2019.

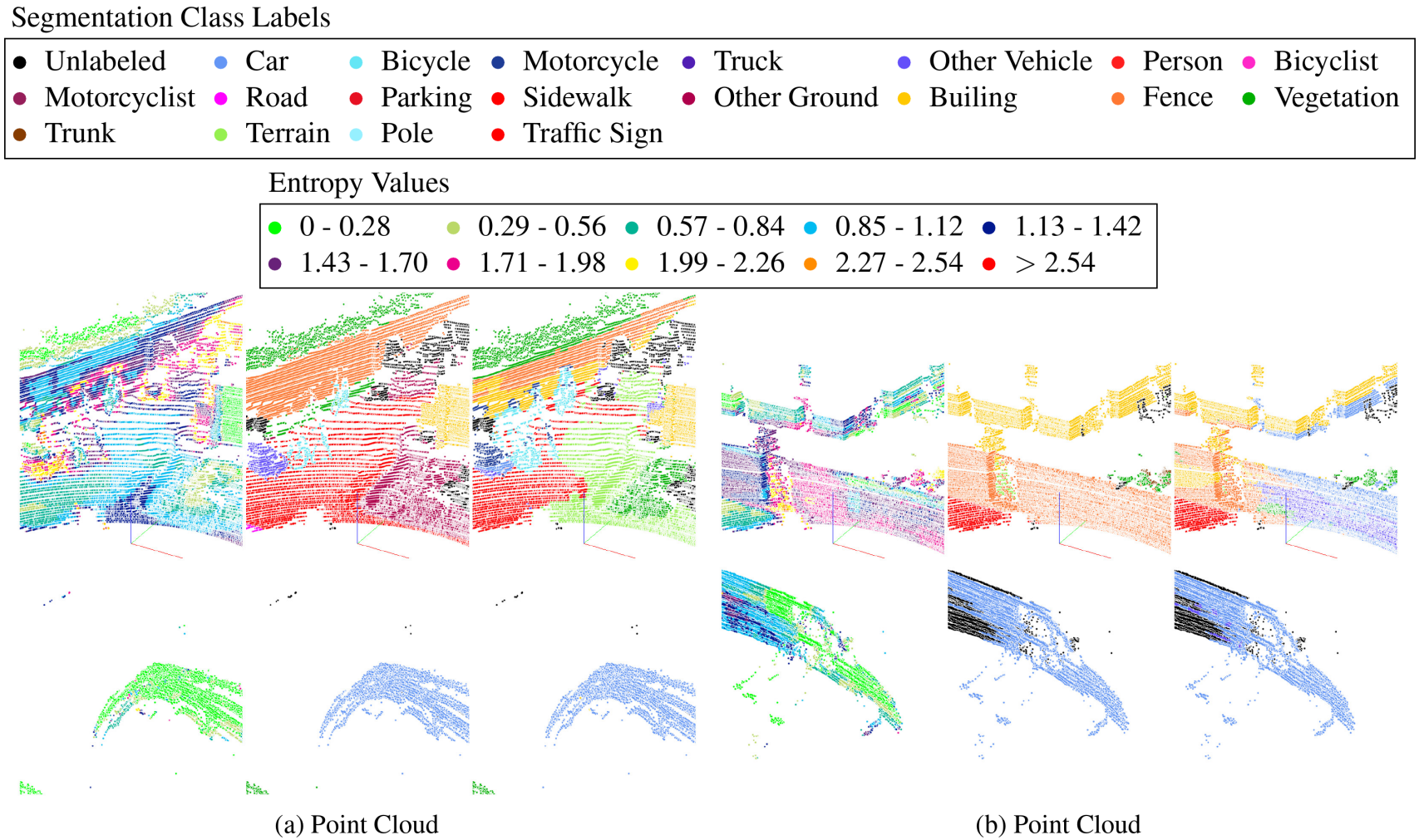
Uncertainty in Emotion Classification [Matin et al. 2020.]



Uncertainty in Point Cloud Segmentation [Bhandary et al. 2020]



Uncertainty in Point Cloud Segmentation [Bhandary et al. 2020]



(a) Point Cloud (b) Point Cloud

Entropy (Right) Ground truth (Center), Predictions (Right).

Unsupervised Difficulty Estimation [Arriaga & Valdenegro. 2020]

Idea. Look how the loss evolves for each sample on the train/val set, accumulating loss for each sample (as a metric).

Hypothesis. Difficult examples accumulate more loss than easy ones.

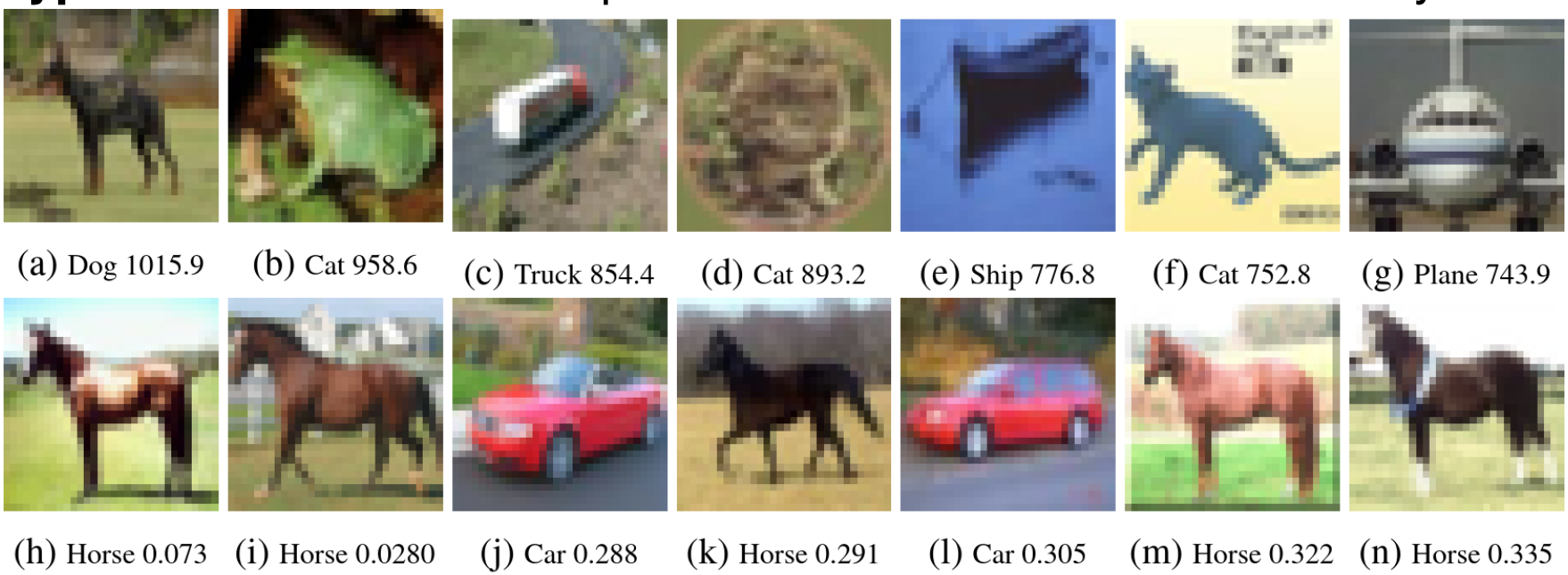
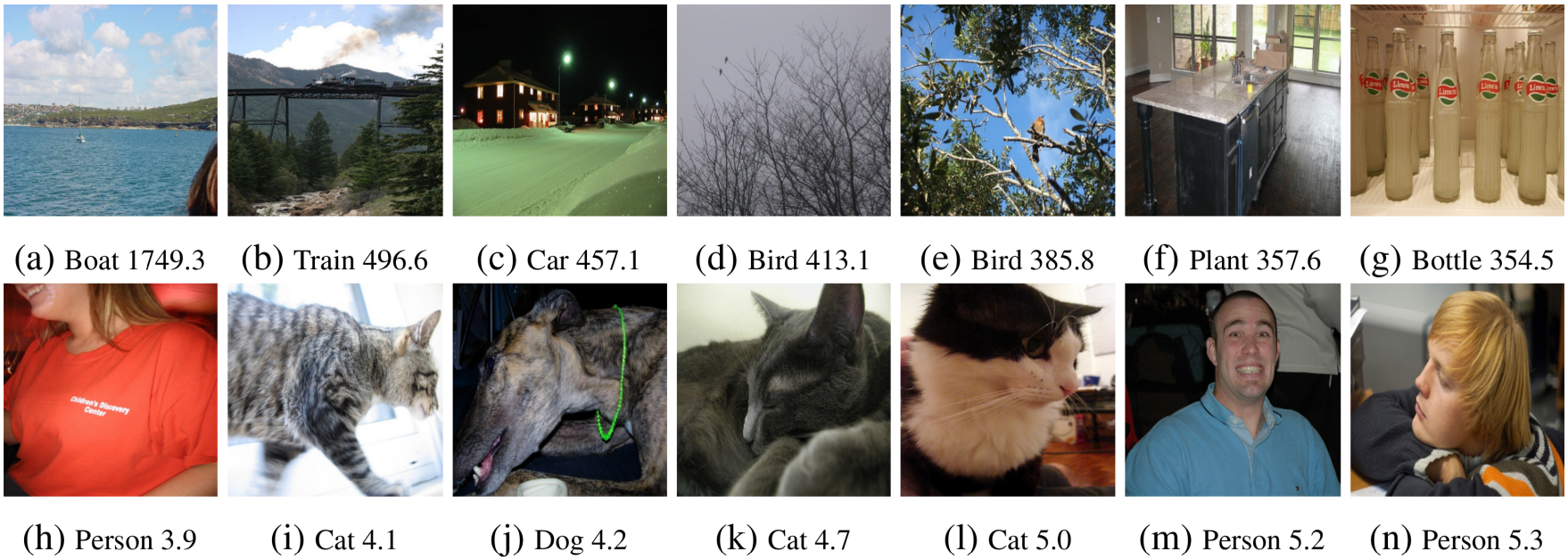


Figure 1: Most difficult (top-row) and easiest examples (bottom-row) in CIFAR10. Our proposed *action score* is displayed below each image as well as the true label.

Unsupervised Difficulty Estimation [Arriaga & Valdenegro. 2020]



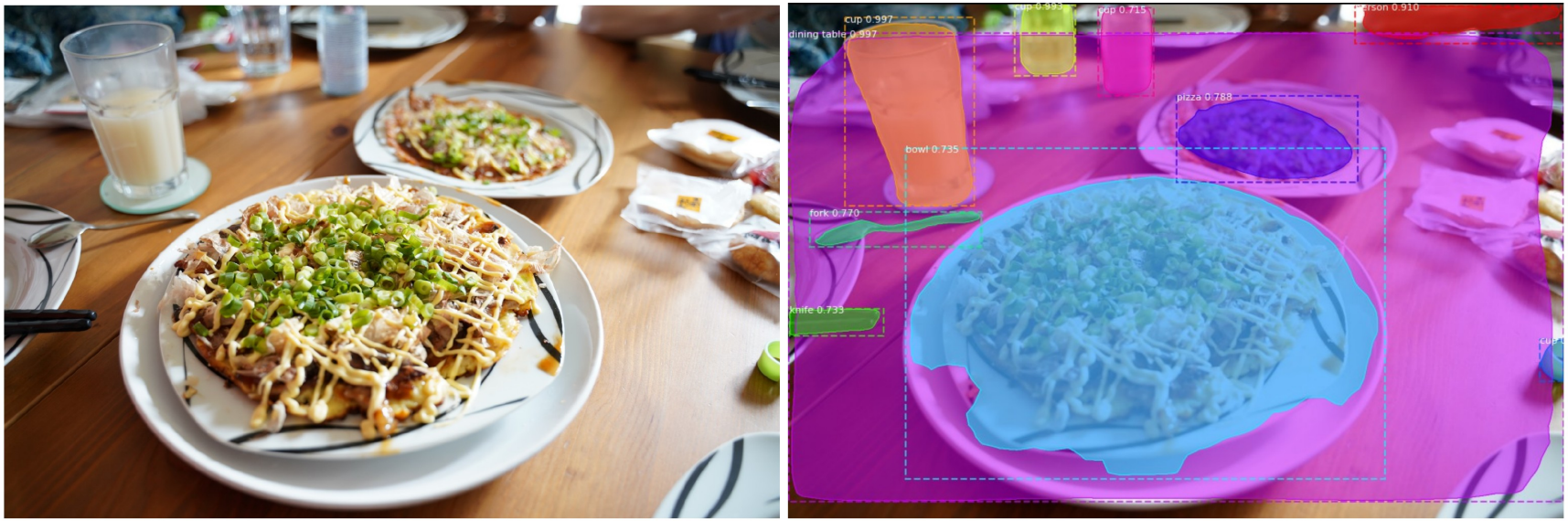
(a) Boat 1749.3 (b) Train 496.6 (c) Car 457.1 (d) Bird 413.1 (e) Bird 385.8 (f) Plant 357.6 (g) Bottle 354.5

(h) Person 3.9 (i) Cat 4.1 (j) Dog 4.2 (k) Cat 4.7 (l) Cat 5.0 (m) Person 5.2 (n) Person 5.3

Figure 2: Most difficult (top-row) and easiest examples (bottom-row) in the VOC 2007-VAL with the SSD localization loss. The *action scores* are displayed below each image as well as the true label.

We are also looking at relationship between action score and uncertainty (entropy), and possible predictions of model and data biases.

Lack of Uncertainty in Computer Vision



Incorrect detections with Mask R-CNN trained on COCO. Left presents the input image, and right the predictions. The Okonomiyaki is misclassified as a Bowl and a Pizza, and a Spoon is misclassified as a Fork, all with high confidence of $\sim 75\%$. The Bowl-Okonomiyaki has high variability in the predicted mask boundaries, signaling some amount of uncertainty.

M. Valdenegro, "I Find your Lack of Uncertainty in Computer Vision Disturbing.", Accepted at CVPR 2021 workshops.

Lack of Uncertainty in Computer Vision

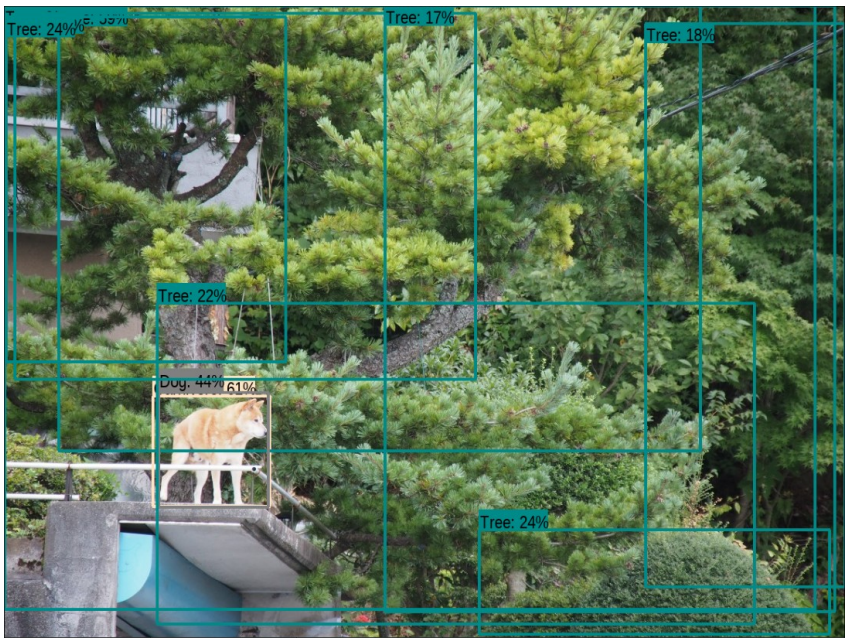


Figure 5: Multiple incorrect detections with low confidence. Shiba Dog is detected as 44% dog and 61% carnivorous, which is counterintuitive for humans.

Figure 6: Multiple incorrect detections with relatively high confidence, including detecting persons and bowls.

Keras-uncertainty

It is a small library I developed that implements common methods for evaluation and estimation of uncertainty in Keras models. It can be installed with: `pip install -user`

`git+https://github.com/mvaldenegro/keras-uncertainty.git`

There are many methods to estimate uncertainty in Neural Networks, I have only implemented the ones that are the most widely applicable and scale easily with the # of parameters of a network.

Keras-Uncertainty Modules

keras_uncertainty.models

Basic uncertainty meta-models such as `MCDropoutModel`, `DeepEnsembleRegressor/Classifier`, and `SimpleEnsemble`

keras_uncertainty.layers

Layers that can be used to build UQ models, such as `DropConnectDense`, `DropConnectConv1D/2D/3D`.

keras_uncertainty.utils

Miscellaneous utilities used in UQ, such as `classifier_calibration_error`, `classifier_calibration_curve`, `classifier_accuracy_confidence_curve`.

MC-Dropout API - Classification

```
from keras_uncertainty.models import MCDropoutClassifier

model = Sequential([
    Dense(32, activation="relu", input_shape=(1,)),
    Dropout(0.5),
    Dense(5, activation="softmax")
])

model.fit(x_train, y_train, epochs=10)

mc_model = MCDropoutClassifier(model)
probs = mc_model.predict(some_data, num_samples=10)
```

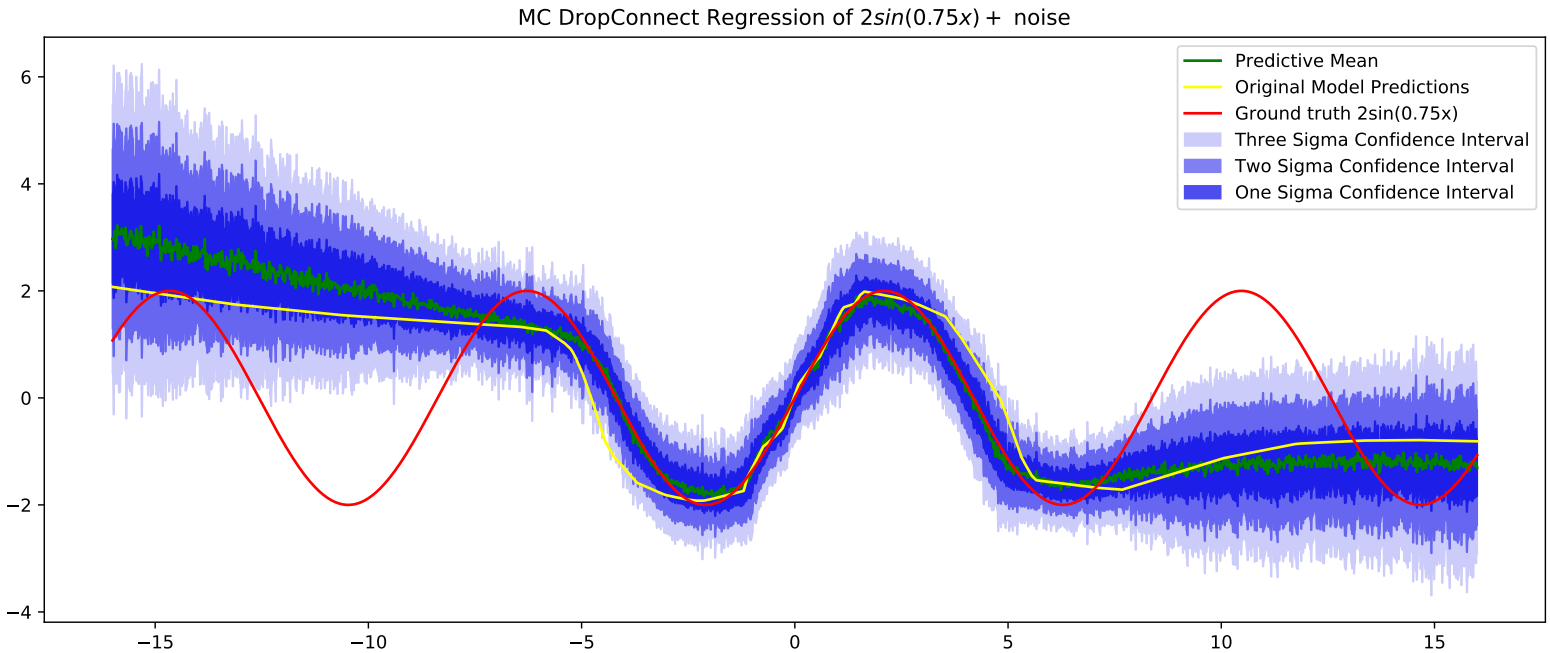
MC-Dropout API - Regression

```
from keras_uncertainty.models import MCDropoutRegressor

model = Sequential([
    Dense(32, activation="relu", input_shape=(1,)),
    Dropout(0.5),
    Dense(1, activation="linear")
])

model.fit(x_train, y_train, epochs=10)

mc_model = MCDropoutRegressor(model)
pred_mean, pred_std = mc_model.predict(some_data, num_samples=10)
```

MC-DropConnect API - Regression

```
from keras_uncertainty.layers import DropConnectDense,
                                     DropConnectConv2D

model = Sequential([
    DropConnectConv2D(32, 3, activation="relu",
                      input_shape=(28,28,1), prob=0.1),
    Flatten(),
    DropConnectDense(1, activation="linear", prob=0.1)
])

model.fit(x_train, y_train, epochs=10)

mc_model = MCDropoutRegressor(model)
pred_mean, pred_std = mc_model.predict(some_data, num_samples=10)
```

MC-DropConnect - Sinusoid Regression



Ensembles API - Classification

```
from keras_uncertainty.models import DeepEnsembleClassifier

def mlp_model():
    model = Sequential([
        Dense(32, activation="relu", input_shape=(1,)),
        Dense(3, activation="softmax"),
    ])

    return model

ens_model = DeepEnsembleClassifier(mlp_model, num_estimators=5)
ens_model.fit(x_train, y_train, epochs=200)

probs = ens_model.predict(some_data)
```

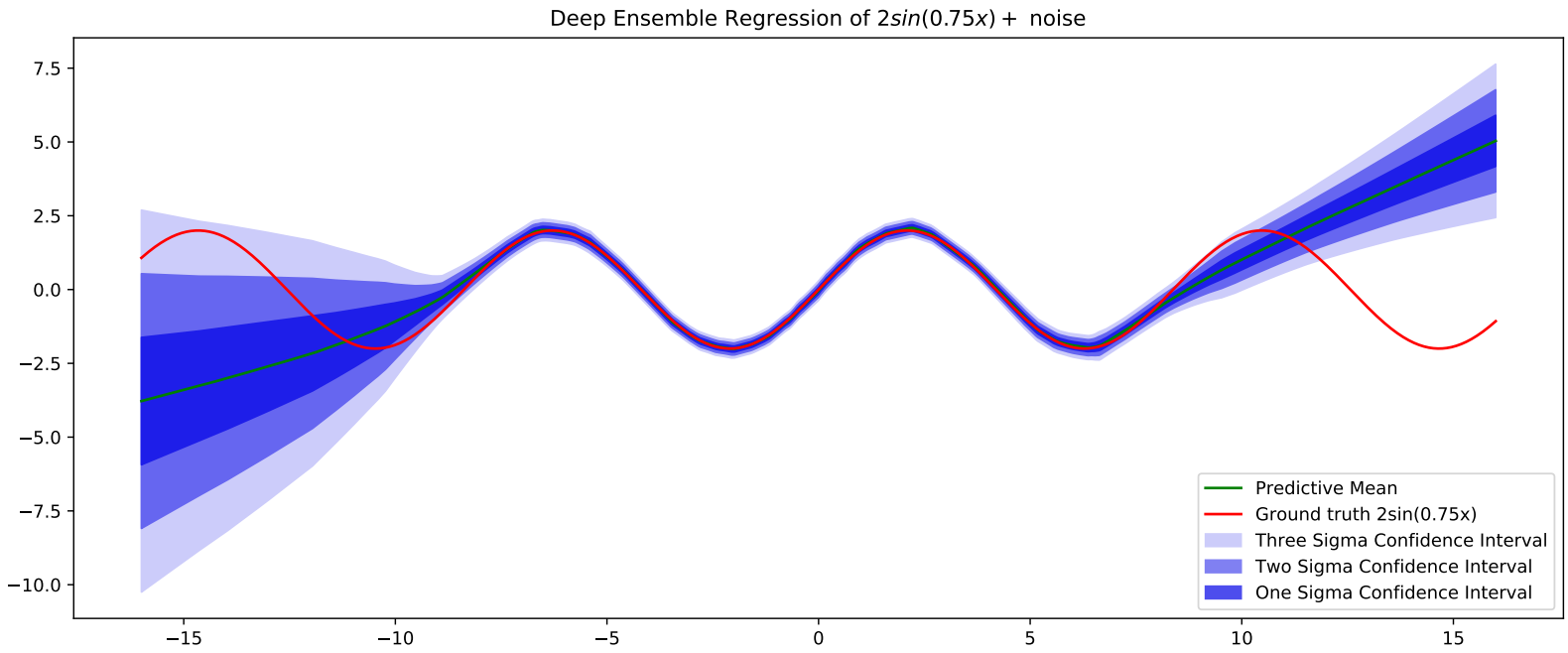
Ensembles API - Regression

```
from keras_uncertainty.models import DeepEnsembleRegressor,
                                     deep_ensemble_regression_nll_loss

def mlp_model():
    inp = Input(shape=(1,))
    x = Dense(10, activation="relu")(inp)
    mean = Dense(1, activation="linear")(x)
    var = Dense(1, activation="softplus")(x)

    tr_mdl = Model(inp, mean)
    pr_mdl = Model(inp, [mean, var])
    tr_mdl.compile(
        loss=deep_ensemble_regression_nll_loss(var),
        optimizer="adam"
    )

    return tr_mdl, pr_mdl
```

Ensembles API - Regression

```
ens_model = DeepEnsembleRegressor(mlp_model, num_estimators=5)
ens_model.fit(x_train, y_train, epochs=200)

pred_mean, pred_std = ens_model.predict(some_data)
```

Out of Distribution Detection - Sinusoid Regression with Ensembles



In this example, the training set is $x \in [-8, 8]$, it can be visually seen that outside this range the standard deviation of the output (uncertainty) increases considerably, and increases as with the distance to that range.

Calibration Implementation

```
from keras_uncertainty.metrics import classifier_calibration_curve
import matplotlib.pyplot as plt

y_pred = model.predict(x_data)
y_class = numpy.argmax(y_pred, axis=1)
y_confs = numpy.max(y_pred, axis=1)

conf, acc = classifier_calibration_curve(y_preds, y_class,
                                         y_confs, num_bins=20)

plt.plot(conf, acc)
plt.xaxis("Confidence")
plt.yaxis("Accuracy")
plt.show()
```

Calibration - Reliability Plots with MC-Dropout on MNIST

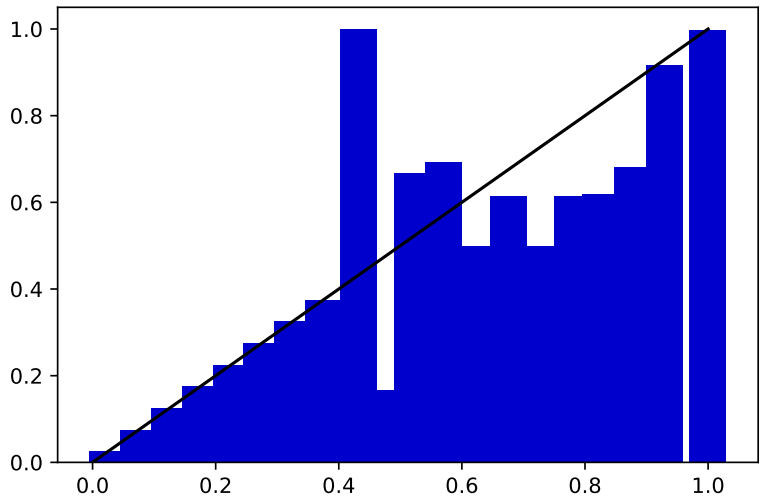


Figure 7: Classical NN, Calibration error is 0.18

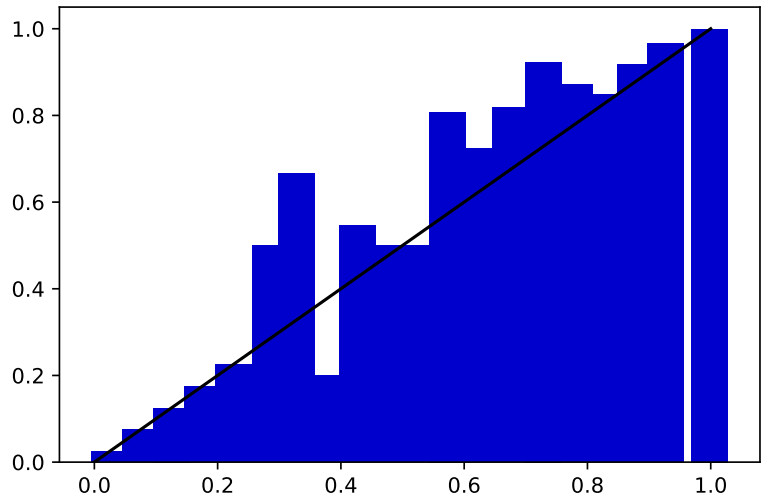
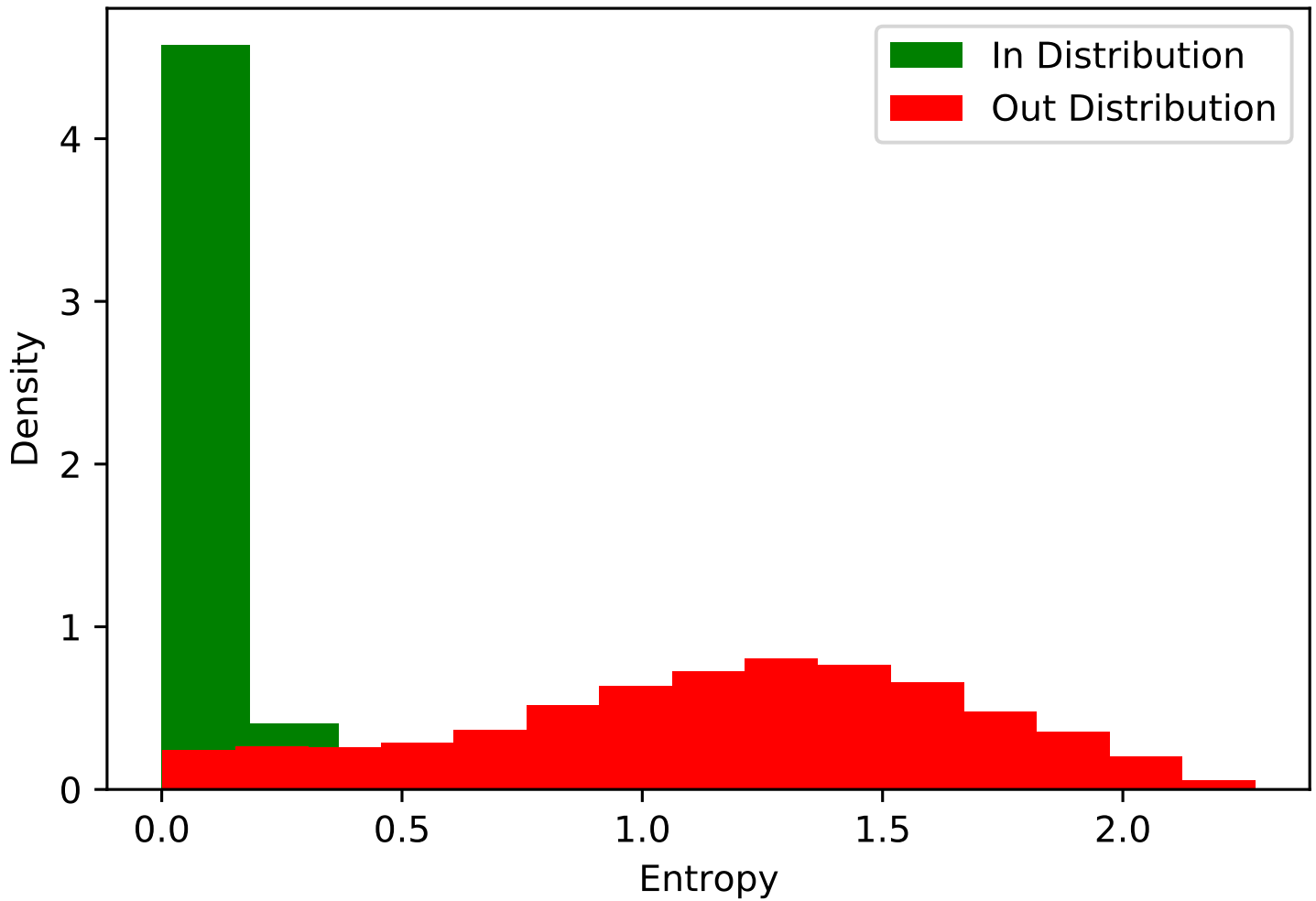


Figure 8: Bayesian NN with MC-Dropout, Calibration error is 0.11

Out of Distribution Detection - MNIST vs Fashion MNIST

```
mc_model = model_trained_mnist()
mnist_preds = mc_model.predict(mnist_data, num_samples=10)

fmnist_data =  fashion_mnist.load_data()

fmnist_preds = mc_model.predict(fmnist_data, num_samples=10)

fmnist_entropy = entropy(fmnist_preds)
mnist_entropy = entropy(mnist_preds)
```

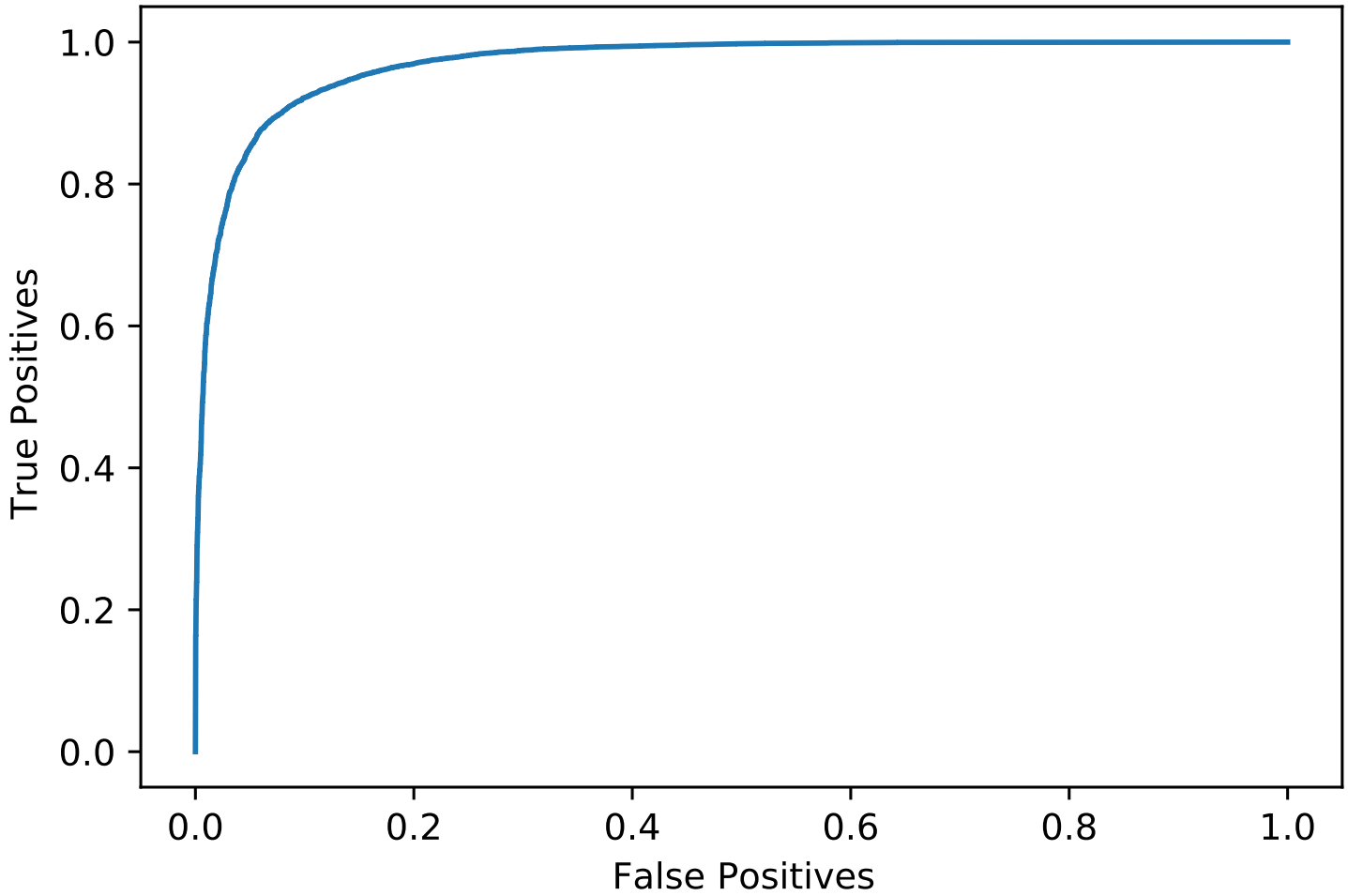
Out of Distribution Detection - MNIST vs Fashion MNIST



Out of Distribution Detection - ROC Curve

```
import numpy as np
from sklearn.metrics import roc_auc_score, roc_curve

mnist_preds = mc_model.predict(mnist_data, num_samples=10)
fmnist_preds = mc_model.predict(fmnist_data, num_samples=10)

fmnist_entropy = entropy(fmnist_preds)
mnist_entropy = entropy(mnist_preds)

id_labels = np.zeros_like(mnist_preds)
ood_labels = np.ones_like(fmnist_preds)

labels = np.concatenate([id_labels, ood_labels], axis=0)
scores = np.concatenate([mnist_entropy, fmnist_entropy], axis=0)

auc = roc_auc_score(labels, scores)
fpr, tpr = roc_curve(labels, scores)
```

Out of Distribution Detection - ROC Curve



Conclusions and Future Thoughts

- Uncertainty is a useful measure to detect misclassified and out of distribution examples.
- Bayesian neural networks are not often used in practice, and many applications would benefit from them. Computational performance is a big reason.
- It is important to spread these techniques and their possible applications, specially now that ML is used in real-world applications that require to estimate model limits.
- Robotics in particular is a great application field, for example with Bayesian Reinforcement Learning, Probabilistic Object Detection, etc.
- I expect increase use of these techniques in practice.