

Uncertainty Quantification

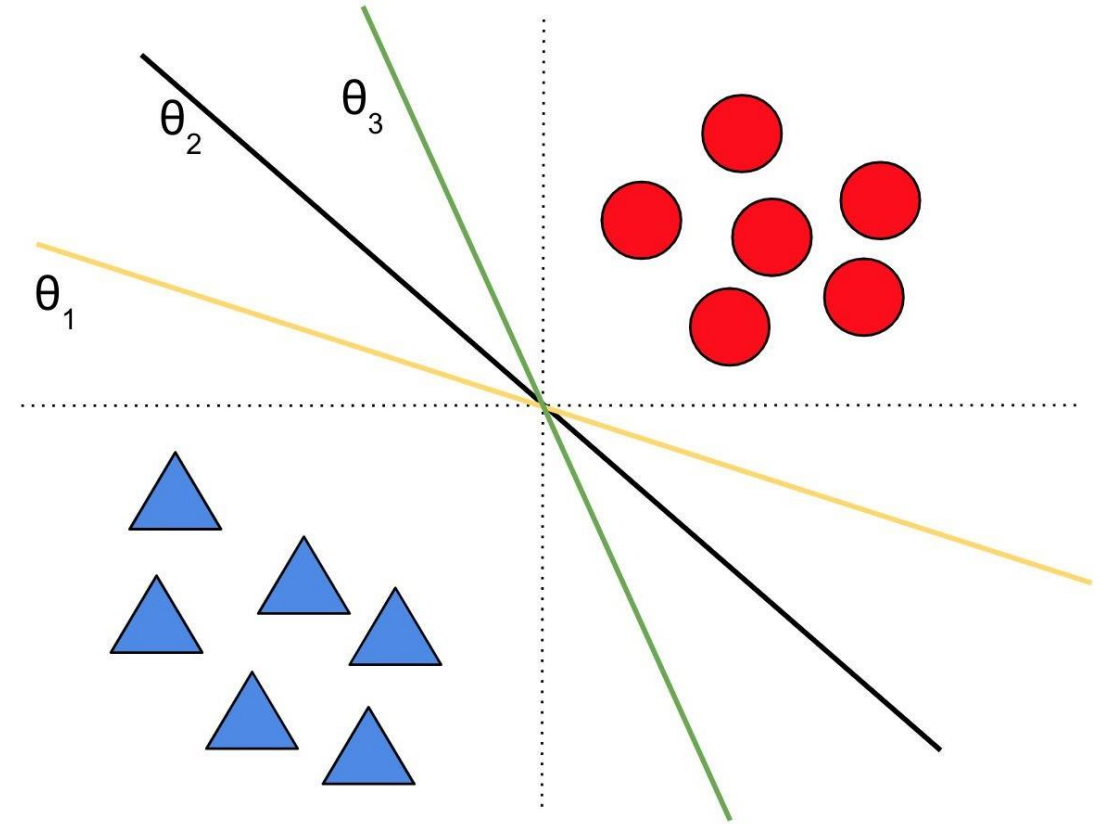
Sourangshu Bhattacharya
CSE, IIT Kharagpur.

Outline of this Part

- Bayesian Neural networks
- Ensemble Learning
- Neural Gaussian Process
- Post-hoc Calibration
- Conformal Prediction

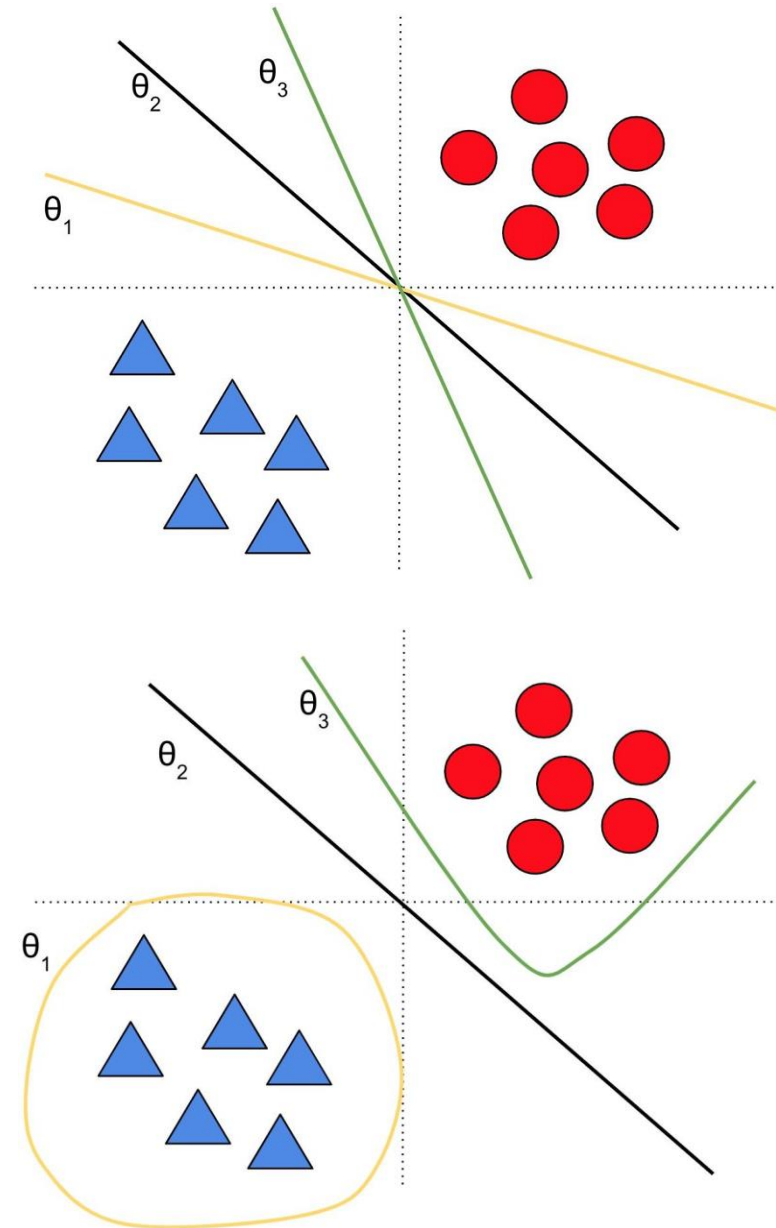
Sources of uncertainty: *Model uncertainty*

- Many models can fit the training data well
- Also known as *epistemic uncertainty*
- Model uncertainty is “**reducible**”
 - Vanishes in the limit of infinite data (subject to model identifiability)



Sources of uncertainty: *Model uncertainty*

- Many models can fit the training data well
- Also known as ***epistemic uncertainty***
- Model uncertainty is “**reducible**”
 - Vanishes in the limit of infinite data (subject to model identifiability)
- Models can be from same hypotheses class (e.g. linear classifiers in top figure) or belong to different hypotheses classes (bottom figure).



Sources of uncertainty: *Data uncertainty*

- Labeling noise (ex: human disagreement)

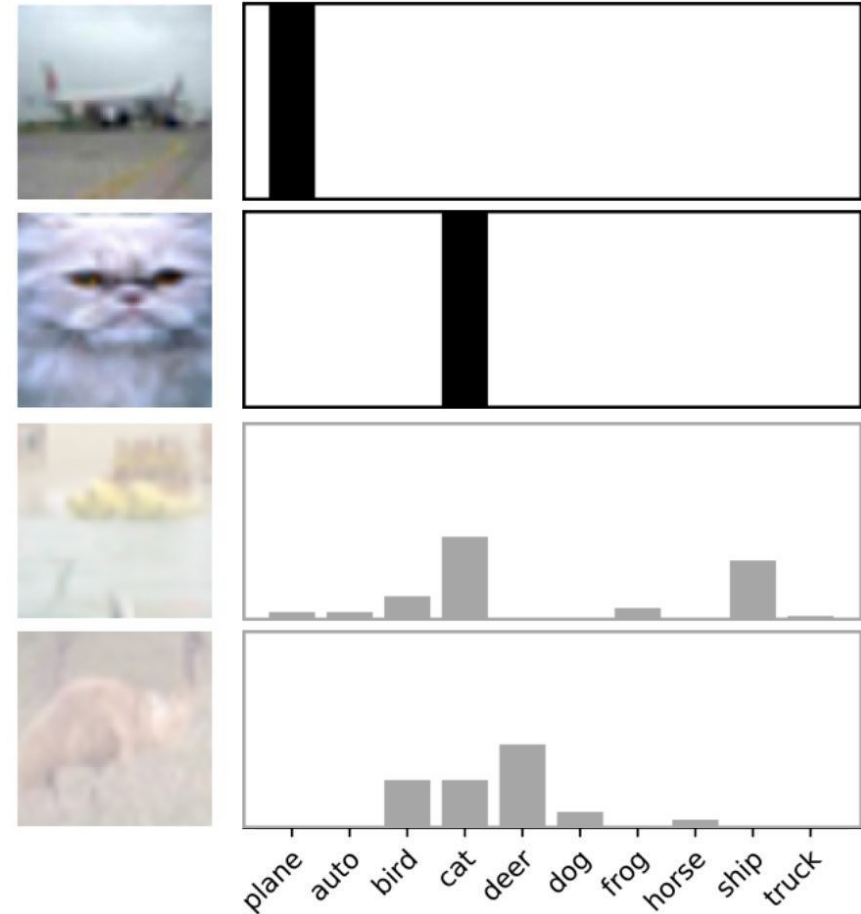


Image source: [Battleday et al. 2019](#) “Improving machine classification using human uncertainty measurements”

Sources of uncertainty: *Data uncertainty*

- Labeling noise (ex: human disagreement)

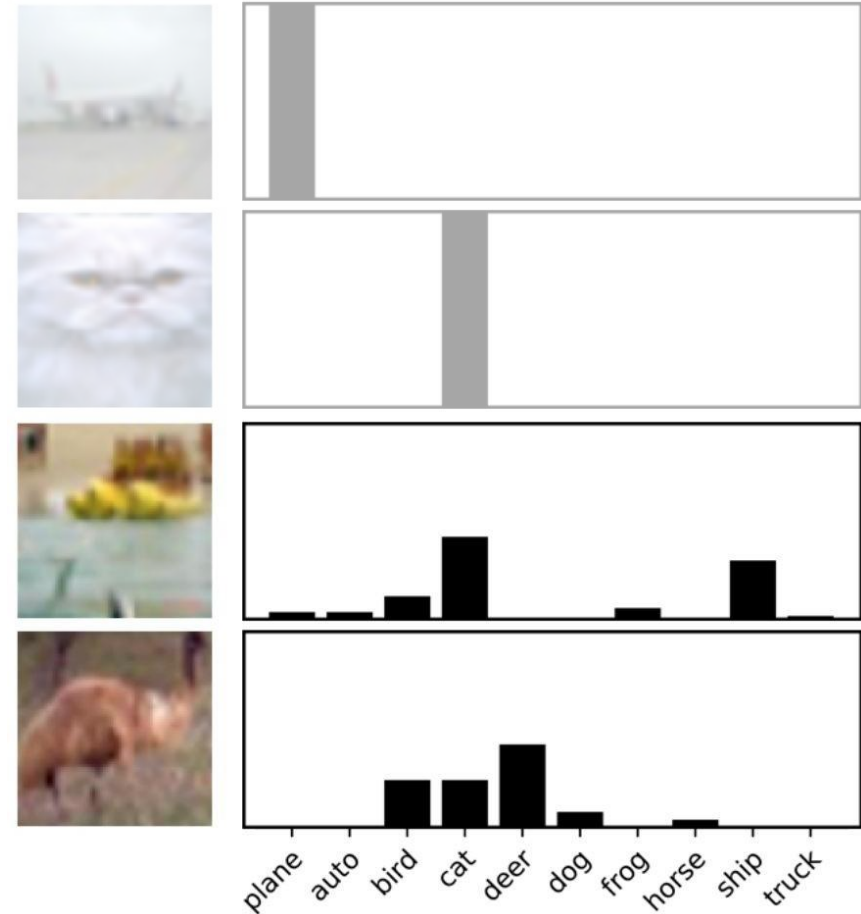


Image source: [Battleday et al. 2019](#) “Improving machine classification using human uncertainty measurements”

Sources of uncertainty: *Data uncertainty*

- **Labeling noise** (ex: human disagreement)
- **Measurement noise** (ex: imprecise tools)
- **Missing data** (ex: partially observed features, unobserved confounders)
- Also known as **aleatoric uncertainty**
- Data uncertainty is **“irreducible*”**
 - Persists even in the limit of infinite data
 - *Could be reduced with additional features/views

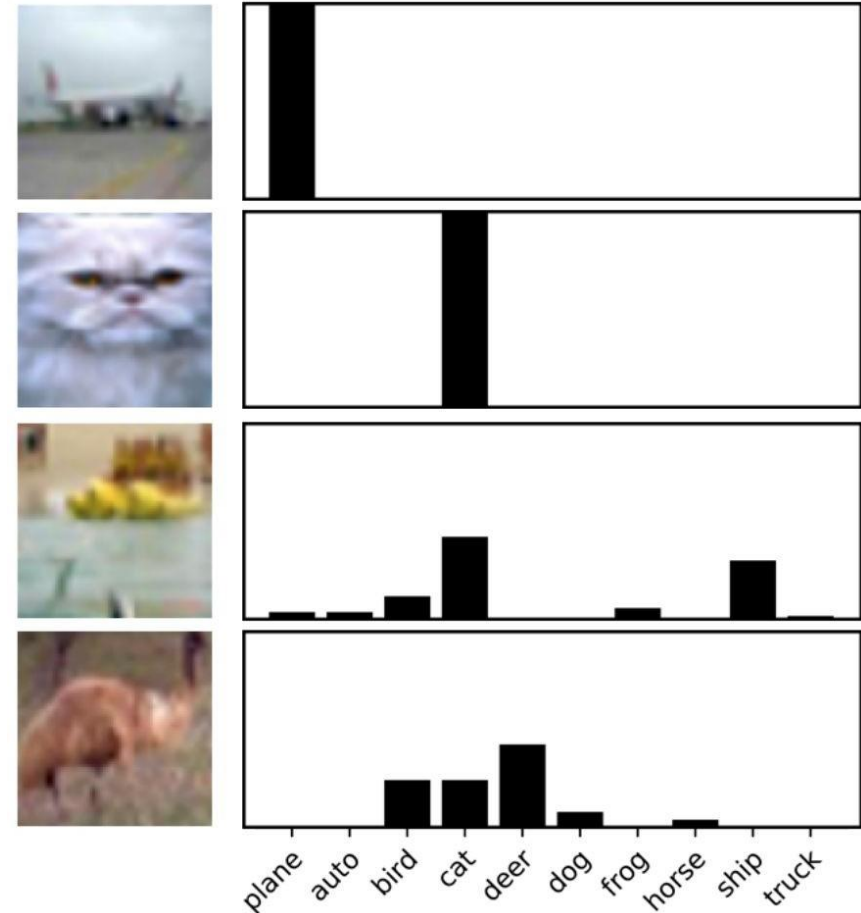


Image source: [Battleday et al. 2019](#) “Improving machine classification using human uncertainty measurements”

How do we measure the quality of uncertainty?

$$\text{Calibration Error} = |\text{Confidence} - \text{Accuracy}|$$

*predicted probability
of correctness*

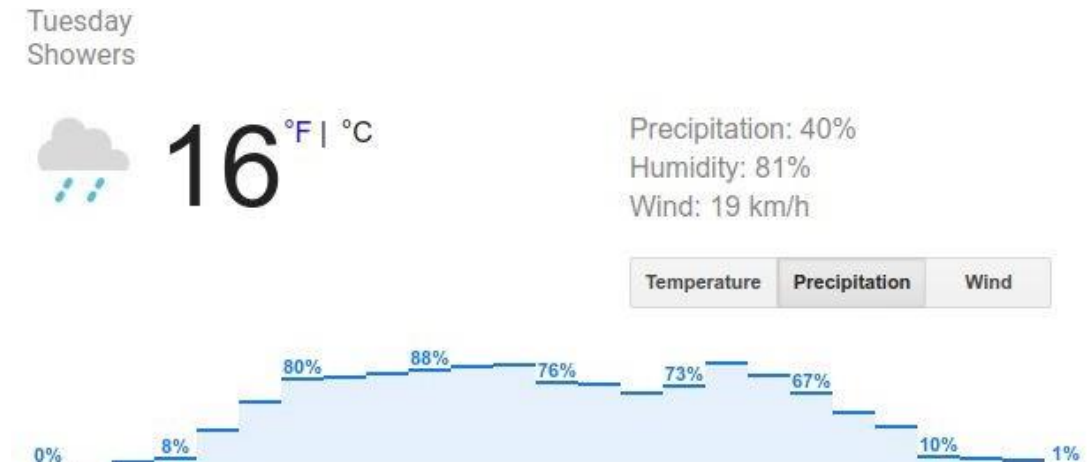
*observed frequency
of correctness*

How do we measure the quality of uncertainty?

$$\text{Calibration Error} = |\text{Confidence} - \text{Accuracy}|$$

Of all the days where the model predicted rain with 80% probability, what fraction did we observe rain?

- 80% implies perfect calibration
- Less than 80% implies model is overconfident
- Greater than 80% implies model is under-confident

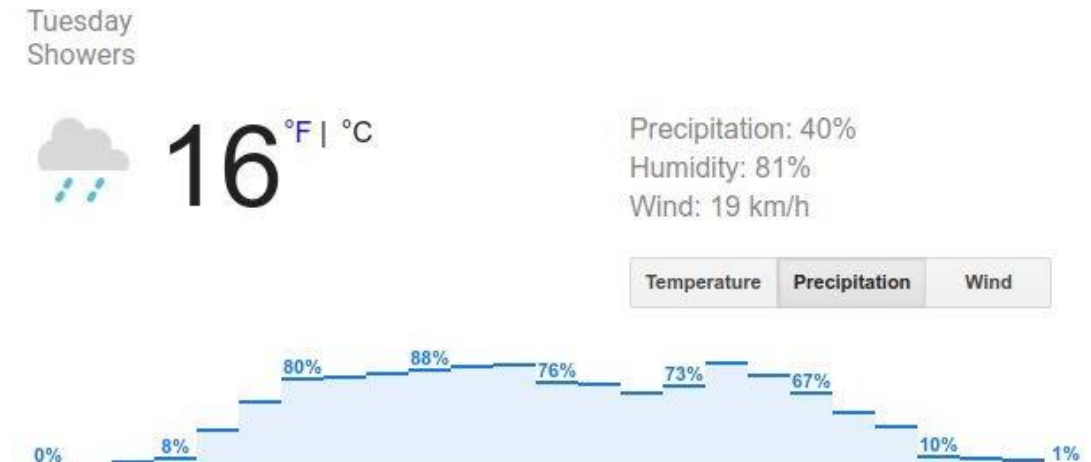


How do we measure the quality of uncertainty?

$$\text{Calibration Error} = |\text{Confidence} - \text{Accuracy}|$$

Of all the days where the model predicted rain with 80% probability, what fraction did we observe rain?

- 80% implies perfect calibration
- Less than 80% implies model is overconfident
- Greater than 80% implies model is under-confident



Intuition: For regression, calibration corresponds to coverage in a confidence interval.

How do we measure the quality of uncertainty?

Expected Calibration Error [\[Naeini+ 2015\]](#):

$$\text{ECE} = \sum_{b=1}^B \frac{n_b}{N} |\text{acc}(b) - \text{conf}(b)|$$

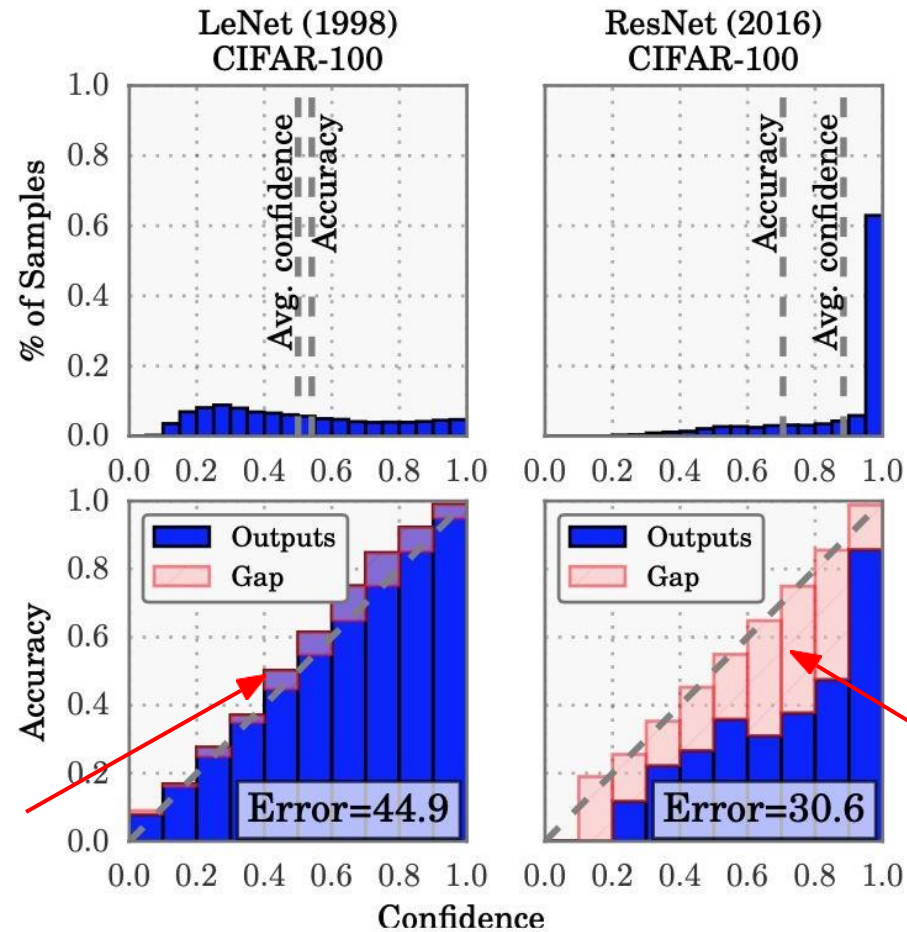
- Bin the probabilities into B bins.
- Compute the within-bin accuracy and within-bin predicted confidence.
- Average the calibration error across bins (weighted by number of points in each bin).

How do we measure the quality of uncertainty?

Expected Calibration Error [Naeini+ 2015]:

$$\text{ECE} = \sum_{b=1}^B \frac{n_b}{N} |\text{acc}(b) - \text{conf}(b)|$$

Confidence < Accuracy
 \Rightarrow Underconfident



Confidence > Accuracy
 \Rightarrow Overconfident

Image source: [Guo+ 2017](#) "On calibration of modern neural networks"



How do we measure the quality of uncertainty?

Expected Calibration Error [\[Naeini+ 2015\]](#):

$$\text{ECE} = \sum_{b=1}^B \frac{n_b}{N} |\text{acc}(b) - \text{conf}(b)|$$

Note: Does **not** reflect **accuracy**.

Predicting class frequency $p(y=1) = 0.3$ for all the inputs achieves perfect calibration.

| True label | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | Accurate? | Calibrated? |
|------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|---|
| Model prediction | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 |  |  |

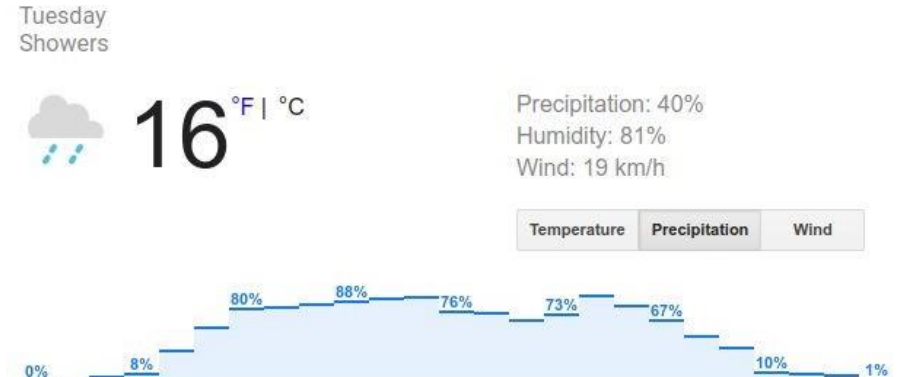
How do we measure the quality of uncertainty?

Proper scoring rules [\[Gneiting & Rahery 2007\]](#)

- Negative Log-Likelihood (NLL)
 - Also known as *cross-entropy*
 - Can overemphasize tail probabilities
- Brier Score
 - Quadratic penalty (bounded range [0,1] unlike log).

$$\text{BS} = \frac{1}{|\mathcal{Y}|} \sum_{y \in \mathcal{Y}} [p(y|\mathbf{x}_n, \theta) - \delta(y - y_n)]^2$$

- Can be numerically unstable to optimize.



How do we measure the quality of uncertainty?

Evaluate model on **out-of-distribution (OOD) inputs** which do not belong to any of the existing classes

- Max confidence
- Entropy of $p(y|x)$



CIFAR-10 (i.i.d test inputs)



SVHN (o.o.d test inputs)

CIFAR-10
classifier

Confidence on i.i.d inputs



Confidence on o.o.d inputs ?

How do we measure the quality of robustness?

Measure generalization to a *large collection of real-world shifts*. A large collection of tasks encourages *general robustness to shifts* (ex: [GLUE](#) for NLP).

- Novel textures in object recognition.
- Covariate shift (e.g. corruptions).
- Different sub-populations (e.g. geographical location).



Cartoon

Different renditions
(ImageNet-R)



Nearby video frames
(ImageNet-Vid-Robust, YTBB-Robust)

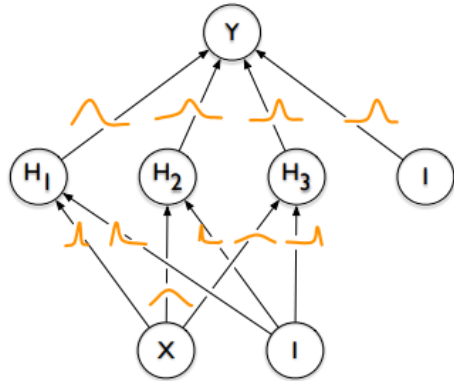


Multiple objects and poses
(ObjectNet)

Bayesian Neural Networks

Bayesian Neural Networks (BNNs)

- Impose probability distribution over model parameters θ to represent model uncertainty.



Training Time: infer the posterior distribution of weight parameters.

Prior: $p(\theta)$

Training data: \mathcal{D}

Posterior: $p(\theta|\mathcal{D}) = \frac{p(\theta)p(\mathcal{D}|\theta)}{\int p(\theta)p(\mathcal{D}|\theta)d\theta}$

Test Time: Bayesian model averaging.

$$p(y^*|\mathbf{x}^*, \mathcal{D}) = \int p(y^*|\mathbf{x}^*, \theta)p(\theta|\mathcal{D})d\theta$$

Bayesian Neural Networks (BNNs)

Each weight \mathbf{w} follows gaussian distribution with mean μ and standard deviation $\sigma = \log(1 + \exp(\rho))$

Parameters μ and ρ are updated using:

1. Sample $\epsilon \sim \mathcal{N}(0, I)$.
2. Let $\mathbf{w} = \mu + \log(1 + \exp(\rho)) \circ \epsilon$.
3. Let $\theta = (\mu, \rho)$.
4. Let $f(\mathbf{w}, \theta) = \log q(\mathbf{w}|\theta) - \log P(\mathbf{w})P(\mathcal{D}|\mathbf{w})$.
5. Calculate the gradient with respect to the mean

$$\Delta_{\mu} = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \mu}. \quad (3)$$

6. Calculate the gradient with respect to the standard deviation parameter ρ

$$\Delta_{\rho} = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\epsilon}{1 + \exp(-\rho)} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \rho}. \quad (4)$$

7. Update the variational parameters:

$$\mu \leftarrow \mu - \alpha \Delta_{\mu} \quad (5)$$

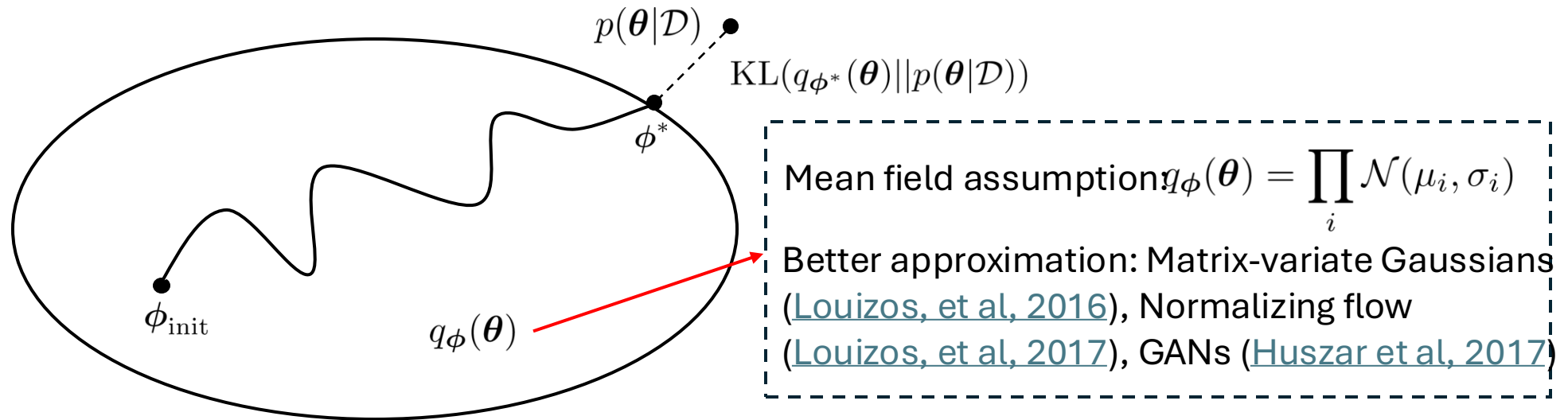
$$\rho \leftarrow \rho - \alpha \Delta_{\rho}. \quad (6)$$

BNN: Challenges

- Difficult to specify prior distribution of the weight parameters.
 - Parameters of DNN are uninterpretable.
 - Usually set as independent Gaussian.
- Exact Bayesian inference is intractable.
 - Approximate inference are needed.
 - Variational inference
 - Laplace approximation
 - SWAG
 - MCMC

BNNs: Variational Inference

- Minimize the KL-divergence between the approximate distribution within a given family and the true posterior

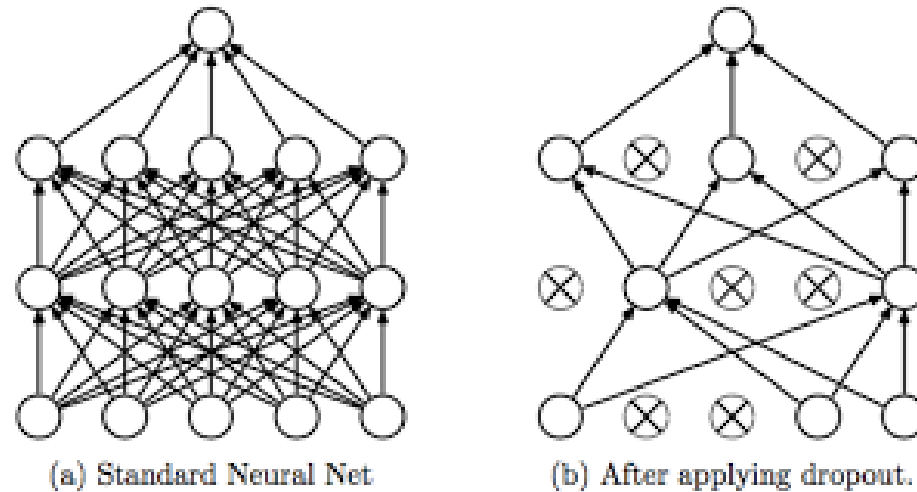


$$\text{ELBO} = \mathbb{E}_{q_{\phi}(\theta)}[\log p(\mathcal{D}|\theta)] - \text{KL}(q_{\phi}(\theta)||p(\theta)) \leq \text{KL}(q_{\phi^*}(\theta)||p(\theta|\mathcal{D}))$$

- Training with SGD to maximize the ELBO

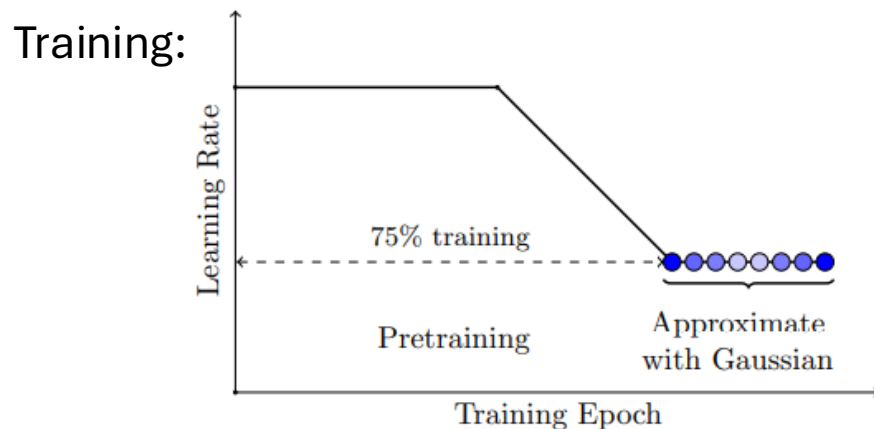
BNN: Monte Carlo Dropout

- Apply dropout at both training and testing time
- Run multiple forward passes with dropout at test time
- Equivalent to BNNs with a Bernoulli variational distribution

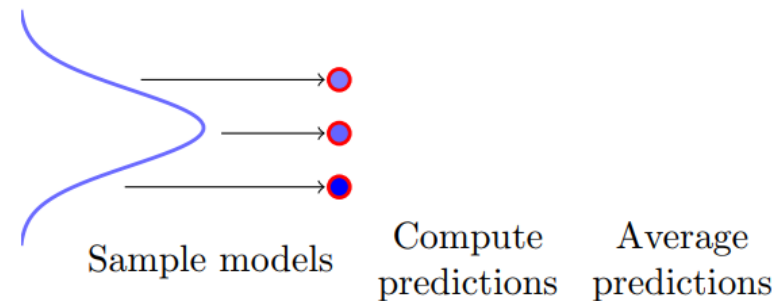


BNN: SWAG

- SGD with a constant learning rate is approximately sampling from a Gaussian distribution ([Mandt et al., 2017](#)).
- Compute first two moments of SGD trajectory.
- Use these moments to construct a Gaussian approximation in weight space.
- Sample from this Gaussian distribution to form a Bayesian model averaging.

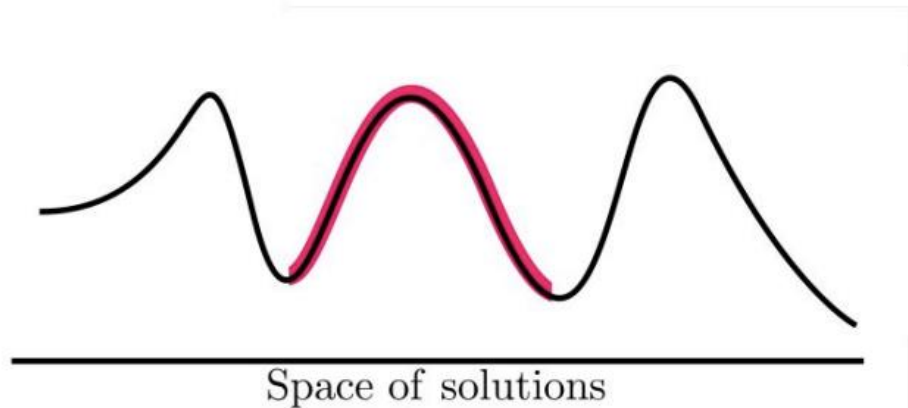


Test:



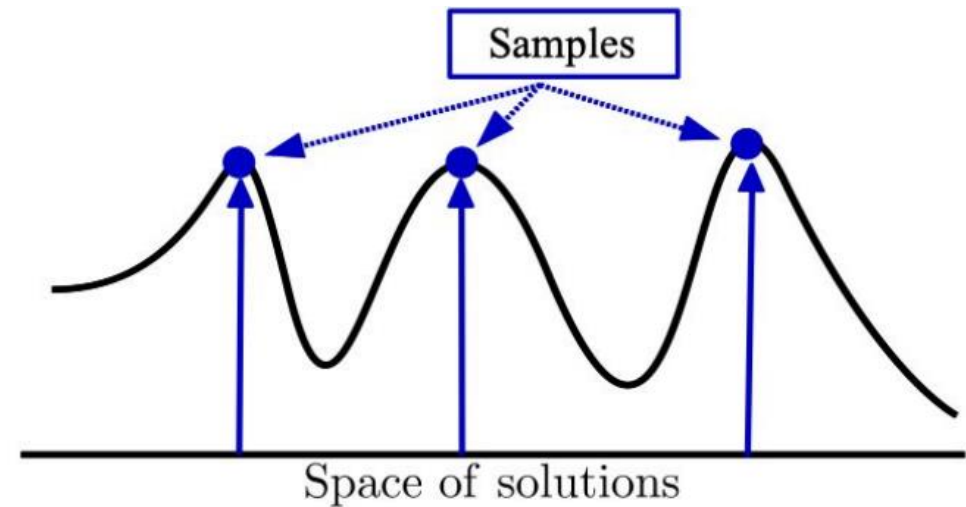
BNN: Summarization

Variational inference, Laplace approximation, SWAG...



- Local approximation with a simpler distribution
 - Simple but can only capture one mode.

MCMC

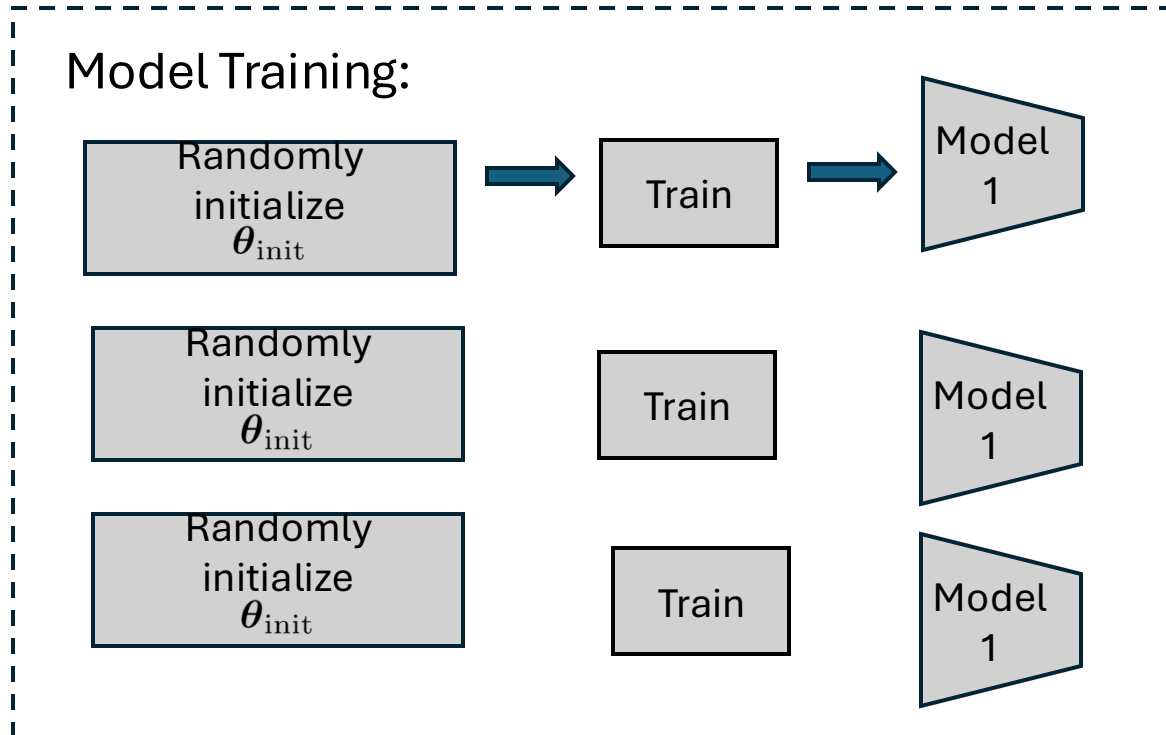


- Represent the distribution with samples
 - Can explore more modes but slow convergence

Ensemble Learning

Ensemble Learning: DeepEnsemble

- Train multiple DNNs with different weight initializations
- Average prediction at test time

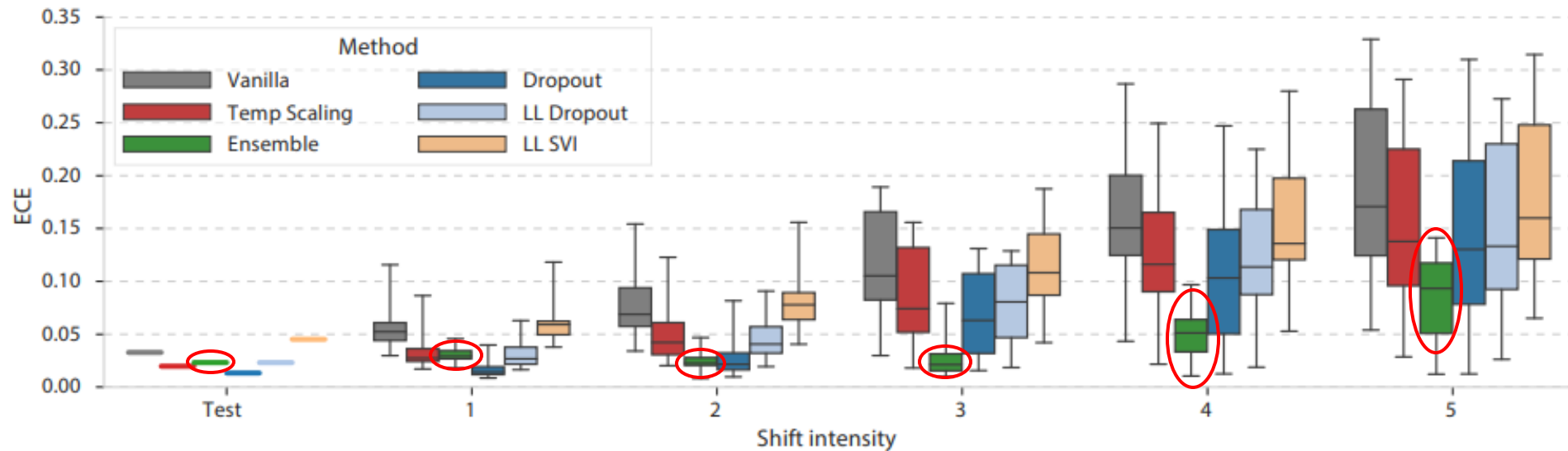


Model inference:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{M} \sum_{m=1}^M p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}_m)$$

DeepEnsemble: Empirical Results

- Simple but works surprisingly well!



DeepEnsemble works consistently well under different shift intensities.

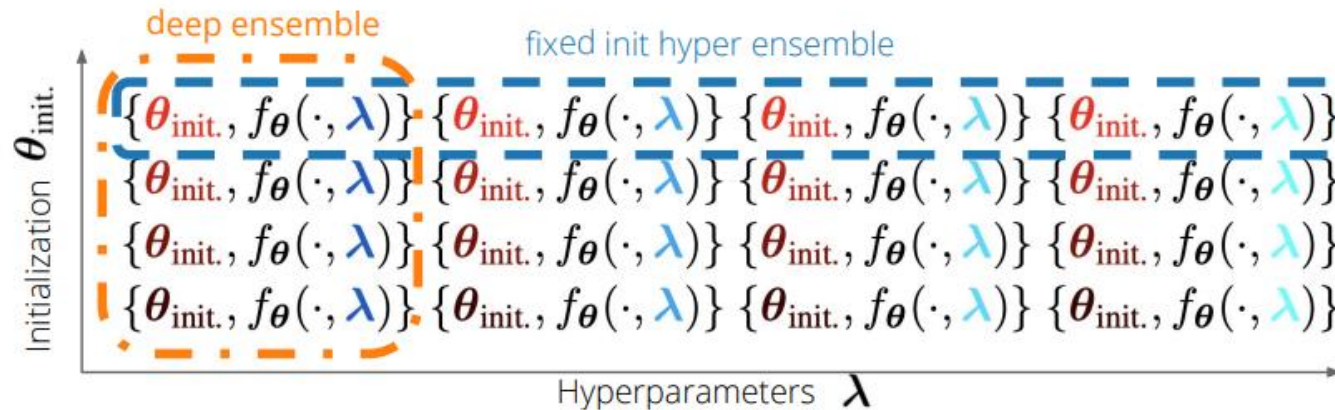
DeepEnsemble: Empirical Results

| | Tox21 | | | | ToxCast | | | | Average Ranking | | | |
|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|-----------------|--------------|--------------|--------------|
| | ROC↑ | ECE↓ | NLL↓ | BS↓ | ROC↑ | ECE↓ | NLL↓ | BS↓ | ROC↓ | ECE↓ | NLL↓ | BS↓ |
| DNN-RDKit | | | | | | | | | | | | |
| Deterministic | 0.7386 | 0.0417 | 0.2771 | 0.0779 | 0.6222 | 0.1168 | 0.4436 | 0.1397 | 25.75 | 17.25 | 24.13 | 22.88 |
| Temperature | 0.7386 | 0.0342 | 0.2723 | 0.0773 | 0.6220 | 0.1114 | 0.4882 | 0.1398 | 25.75 | 15.38 | 21.25 | 19.88 |
| Focal Loss | 0.7374 | 0.1058 | 0.3161 | 0.0871 | 0.6289 | 0.1264 | 0.4389 | 0.1396 | 25.88 | 24.38 | 22.38 | 24.00 |
| MC Dropout | 0.7376 | 0.0356 | 0.2727 | 0.0763 | 0.6248 | 0.1093 | 0.4319 | 0.1358 | 26.50 | 13.00 | 19.38 | 18.63 |
| SWAG | 0.7364 | 0.0438 | 0.2793 | 0.0790 | 0.6207 | 0.1175 | 0.4441 | 0.1400 | 26.38 | 18.50 | 25.63 | 23.50 |
| BBP | 0.7243 | 0.0422 | 0.2847 | 0.0814 | 0.6020 | 0.1443 | 0.4673 | 0.1510 | 22.75 | 19.13 | 19.38 | 21.38 |
| SGLD | 0.7257 | 0.1192 | 0.3455 | 0.0978 | 0.5319 | 0.3054 | 0.6685 | 0.2378 | 27.75 | 26.00 | 28.88 | 27.88 |
| Ensembles | 0.7540 | 0.0344 | 0.2648 | 0.0746 | 0.6486 | 0.0900 | 0.4008 | 0.1292 | 20.00 | 7.13 | 11.75 | 13.13 |
| ChemBERTa | | | | | | | | | | | | |
| Deterministic | 0.7542 | 0.0571 | 0.2962 | 0.0812 | 0.6554 | 0.1209 | 0.4313 | 0.1330 | 15.63 | 17.38 | 18.88 | 19.38 |
| Temperature | 0.7542 | 0.0424 | 0.2744 | 0.0792 | 0.6540 | 0.1067 | 0.4817 | 0.1313 | 15.88 | 12.00 | 13.88 | 15.38 |
| Focal Loss | 0.7523 | 0.0969 | 0.3052 | 0.0845 | 0.6442 | 0.1197 | 0.4243 | 0.1346 | 17.63 | 20.13 | 17.88 | 20.63 |
| MC Dropout | 0.7641 | 0.0423 | 0.2697 | 0.0744 | 0.6624 | 0.1069 | 0.4070 | 0.1276 | 12.50 | 10.75 | 10.63 | 10.00 |
| SWAG | 0.7538 | 0.0592 | 0.3008 | 0.0818 | 0.6556 | 0.1202 | 0.4305 | 0.1327 | 16.13 | 19.50 | 21.38 | 20.38 |
| BBP | 0.7433 | 0.0459 | 0.2765 | 0.0780 | 0.5814 | 0.1276 | 0.4545 | 0.1469 | 20.88 | 19.00 | 16.00 | 19.50 |
| SGLD | 0.7475 | 0.0504 | 0.2784 | 0.0795 | 0.5436 | 0.2238 | 0.5602 | 0.1881 | 21.13 | 19.88 | 19.13 | 18.63 |
| Ensembles | 0.7681 | 0.0440 | 0.2679 | 0.0750 | 0.6733 | 0.1037 | 0.3986 | 0.1258 | 12.38 | 13.00 | 11.63 | 12.25 |

DeepEnsemble works consistently well for various molecular property prediction tasks.

Hyperparameter Ensemble

- The ensemble member in DeepEnsemble differs only in random seed.
- We can also expand the space of hyperparameters we average over.

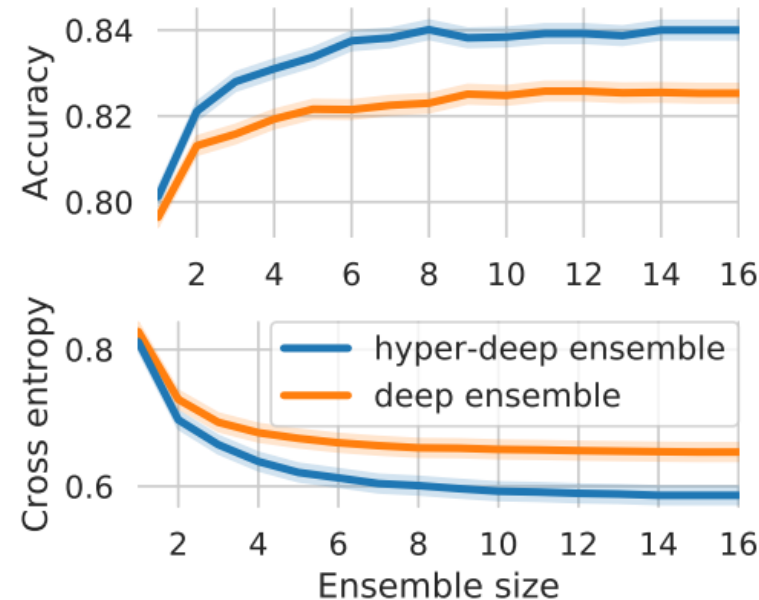


Hyperparameter ensemble search in the whole “block”, exploiting both initialization and hyperparameter diversity.

Hyperparameter Ensemble

- The ensemble member in DeepEnsemble differs only in random seed.
- We can also expand the space of hyperparameters we average over.

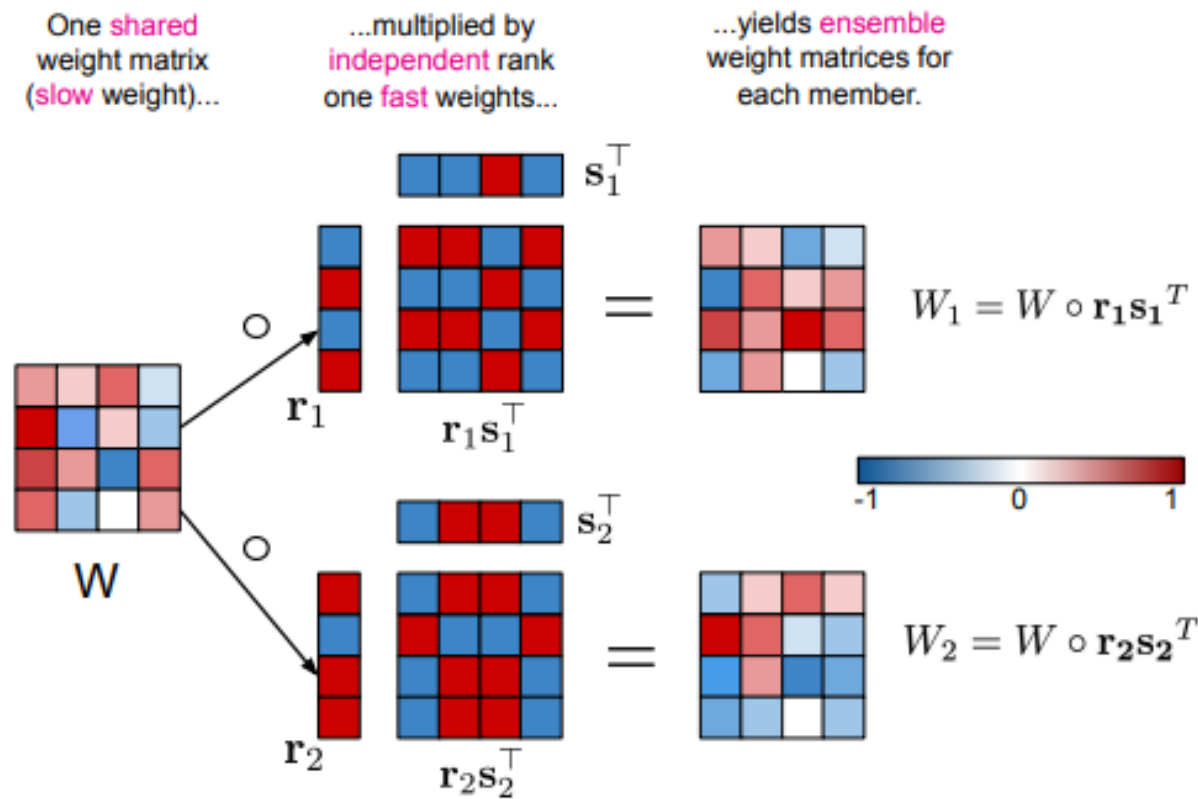
Results on CIFAR100:



Improved uncertainty estimates and accuracy than DeepEnsemble.

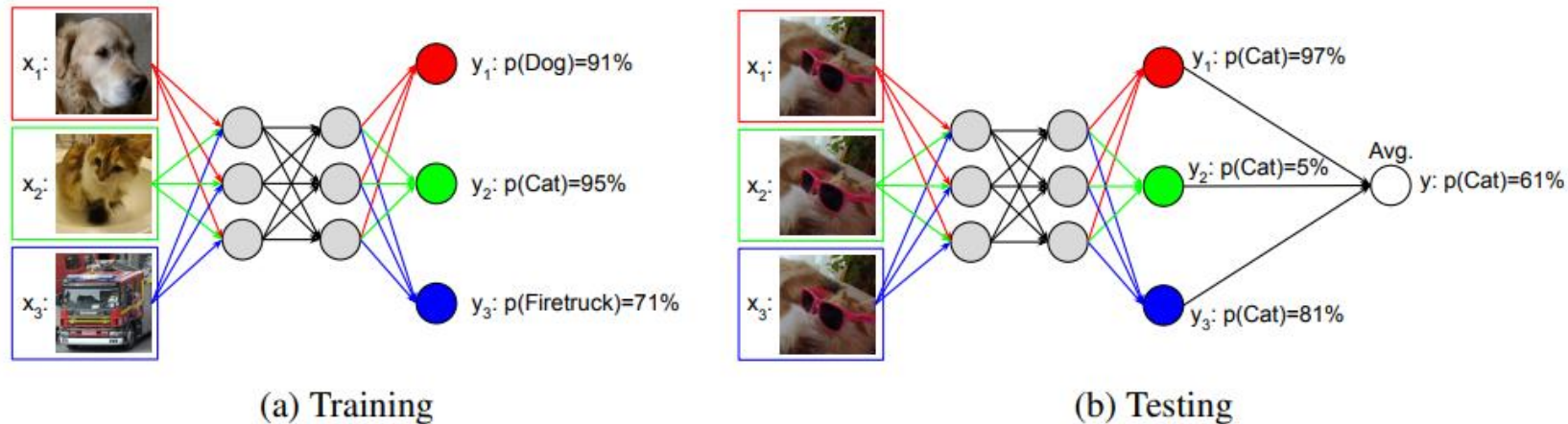
Efficient Ensemble: BatchEnsemble

- Exploit rank-one matrix for parameter-efficient ensemble learning



Efficient Ensemble: Multi-input Multi-output (MIMO)

- Concurrently train multiple independent subnetworks within one network



The black connections are shared by all subnetworks, while the colored connections are for individual subnetworks

- Training: independently sample the inputs from the training set
- Testing: repeat inputs M times and average the outputs in an ensemble

Efficient Ensemble: MIMO

Results on CIFAR100

| Name | Accuracy (\uparrow) | NLL (\downarrow) | ECE (\downarrow) | cAcc (\uparrow) | cNLL (\downarrow) | cECE (\downarrow) | Prediction time (\downarrow) | # Forward passes (\downarrow) |
|------------------------------|-------------------------|----------------------|----------------------|---------------------|-----------------------|-----------------------|----------------------------------|-----------------------------------|
| Deterministic | 79.8 | 0.875 | 0.086 | 51.4 | 2.700 | 0.239 | 0.632 | 1 |
| Monte Carlo Dropout | 79.6 | 0.830 | 0.050 | 42.6 | 2.900 | 0.202 | 0.656 | 1 |
| Naive mutlihead ($M = 3$) | 79.5 | 0.834 | 0.048 | 52.1 | 2.339 | 0.156 | 0.636 | 1 |
| MIMO ($M = 3$) (This work) | 82.0 | 0.690 | 0.022 | 53.7 | 2.284 | 0.129 | 0.639 | 1 |
| TreeNet ($M = 3$) | 80.8 | 0.777 | 0.047 | 53.5 | 2.295 | 0.176 | 0.961 | 1.5 |
| BatchEnsemble ($M = 4$) | 81.5 | 0.740 | 0.056 | 54.1 | 2.490 | 0.191 | 2.552 | 4 |
| Thin Ensemble ($M = 4$) | 81.5 | 0.694 | 0.017 | 53.7 | 2.190 | 0.111 | 0.823 | 4 |
| Ensemble ($M = 4$) | 82.7 | 0.666 | 0.021 | 54.1 | 2.270 | 0.138 | 2.536 | 4 |

Improved uncertainty estimates but much more efficient than DeepEnsemble!

Comparison between Ensemble & BNNs

- Both aggregate predictions over a collection of models. There are two core distinctions.

The space of models.

Bayes posits a prior that weighs different probability to different functions, and over an infinite collection of functions.

Ensembles weigh functions equally a priori and use a finite collection

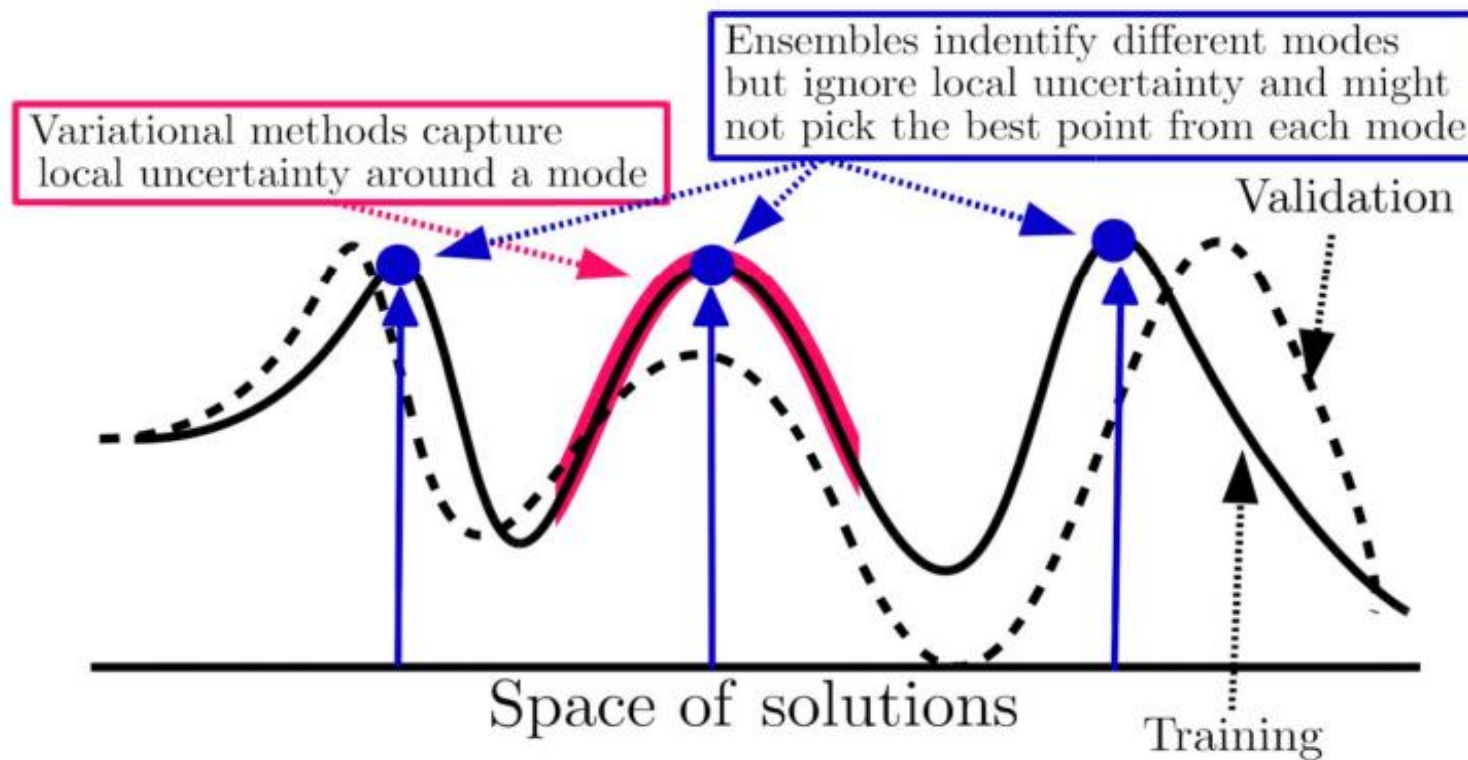
Model aggregation.

Bayesian models apply averaging, weighted by the posterior.

Ensembles can apply any strategy and have non-probabilistic interpretations.

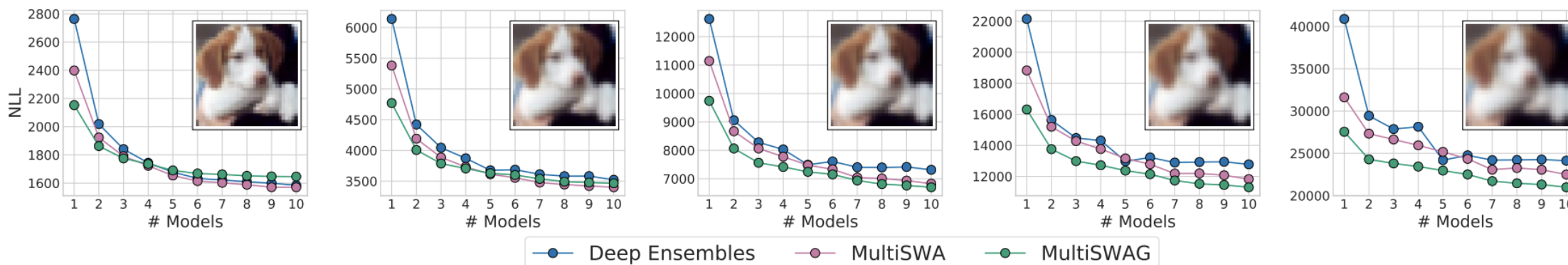
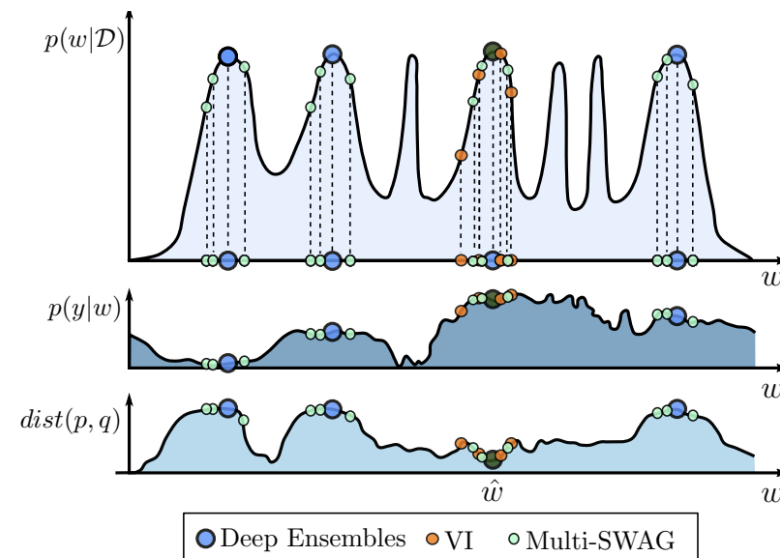
Comparison between Ensemble & BNNs

From Functional perspective



Comparison between Ensemble & BNNs

Ensemble + local approximate can outperform Deep Ensembles.



Neural Gaussian Process

Reminder: Gaussian Process

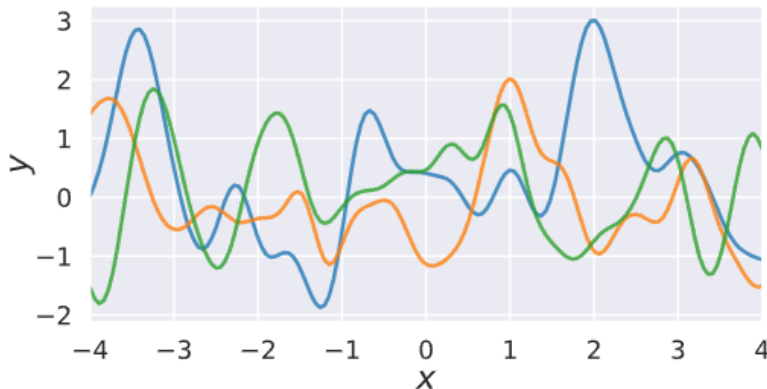
- Definition of a Gaussian process:

$z(x) \sim \mathcal{GP}(\mu, K)$, with mean and covariance functions $\mu(x)$, $K(x, x')$, if any finite set of draws, $[z(x_1), \dots, z(x_n)]^T$, follows $\mathcal{N}(\vec{\mu}, \mathbf{K})$ with

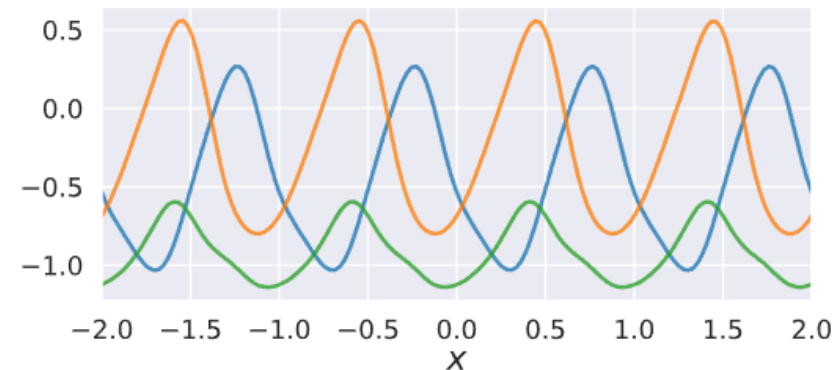
$$\vec{\mu} = \begin{bmatrix} \mu(x_1) \\ \vdots \\ \mu(x_n) \end{bmatrix}, \mathbf{K} = \begin{bmatrix} K(x_1, x_1) & \cdots & K(x_1, x_n) \\ \vdots & \ddots & \vdots \\ K(x_n, x_1) & \cdots & K(x_n, x_n) \end{bmatrix}$$

- Encode Prior from the functional space through kernel function.
 - Usually much easier than weight space!

RBF
Kernel:

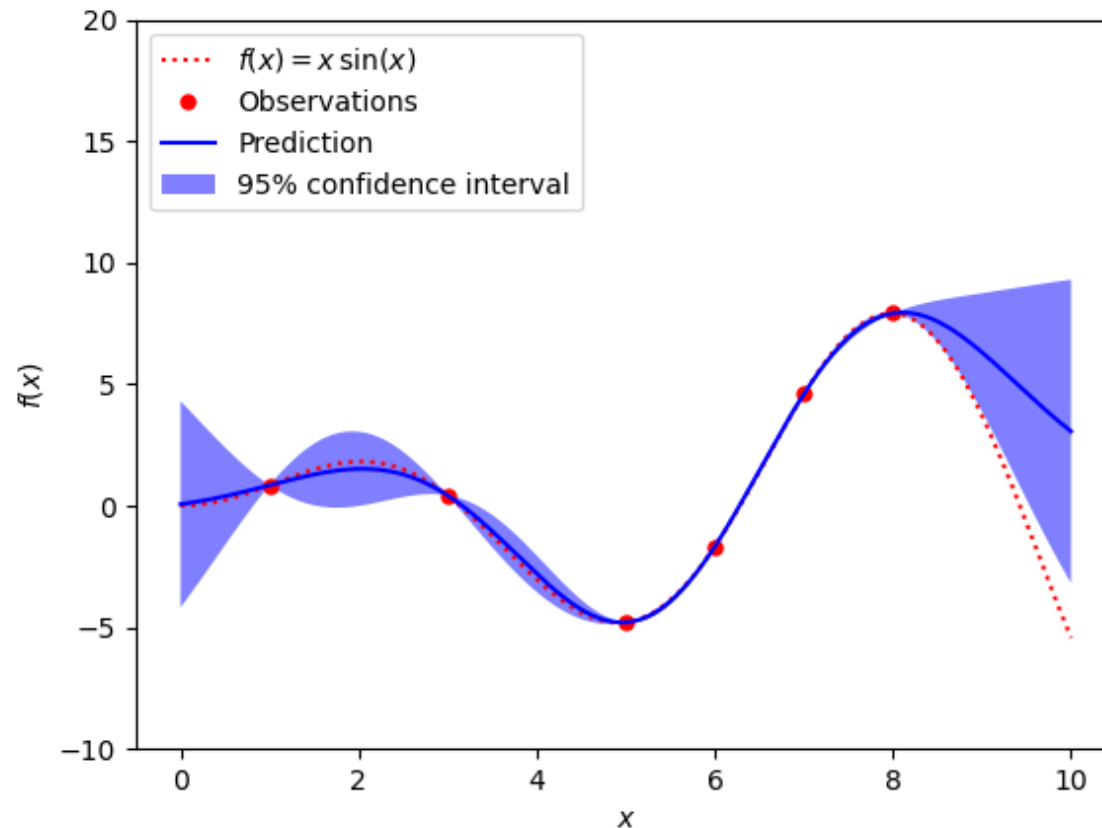


Periodic
kernel:



Reminder: Gaussian Process

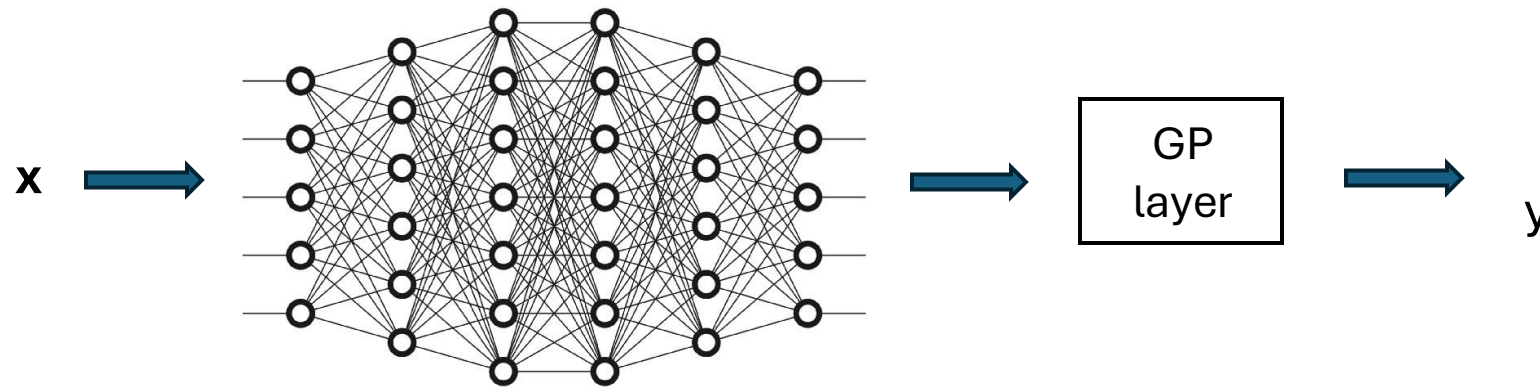
- Gaussian Process can provide reasonable uncertainty estimates.



[\[Image source\]](#)

Deep Kernel Learning

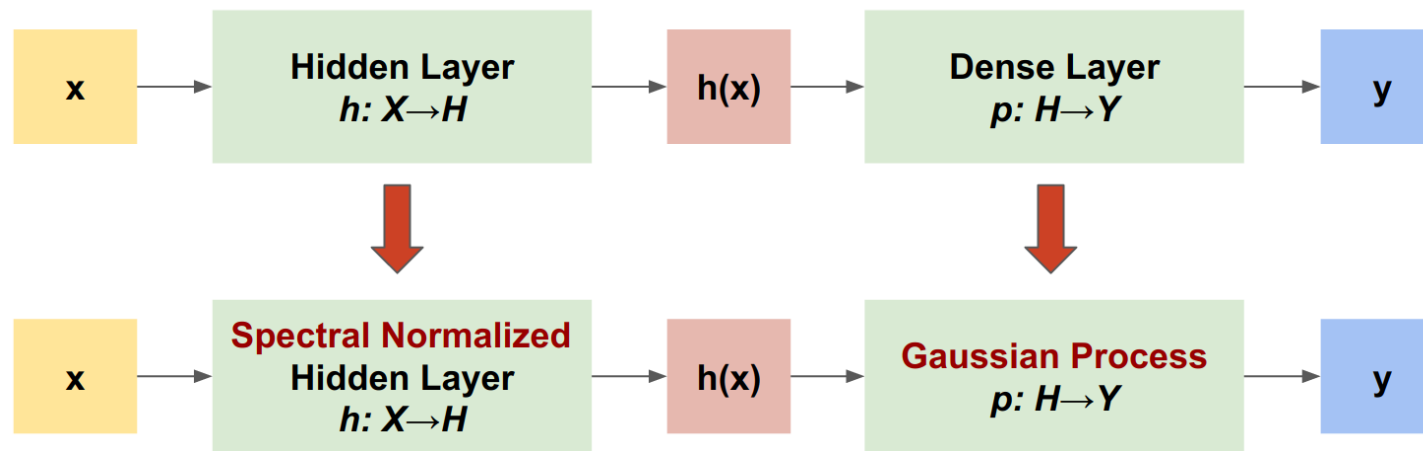
- Deep neural networks: high expressive power but **poor** uncertainty estimates
- Gaussian process: **low** expressive power but good uncertainty estimates
- Deep kernel learning wins the best of both worlds.
- Avoid the need of specifying the prior of the uninterpretable weights as in BNNs



Apply a GP layer on the features extracted by the DNN

Spectral Normalized Neural Gaussian Process

- Models should be distance aware: uncertainty increases farther from training data.
- Replace output layer with GP layer.
- Apply spectral normalization to preserve input distances within internal layers.



Apply Laplace approximation for fast inference

Spectral Normalized Neural Gaussian Process

Algorithm 1 SNGP Training

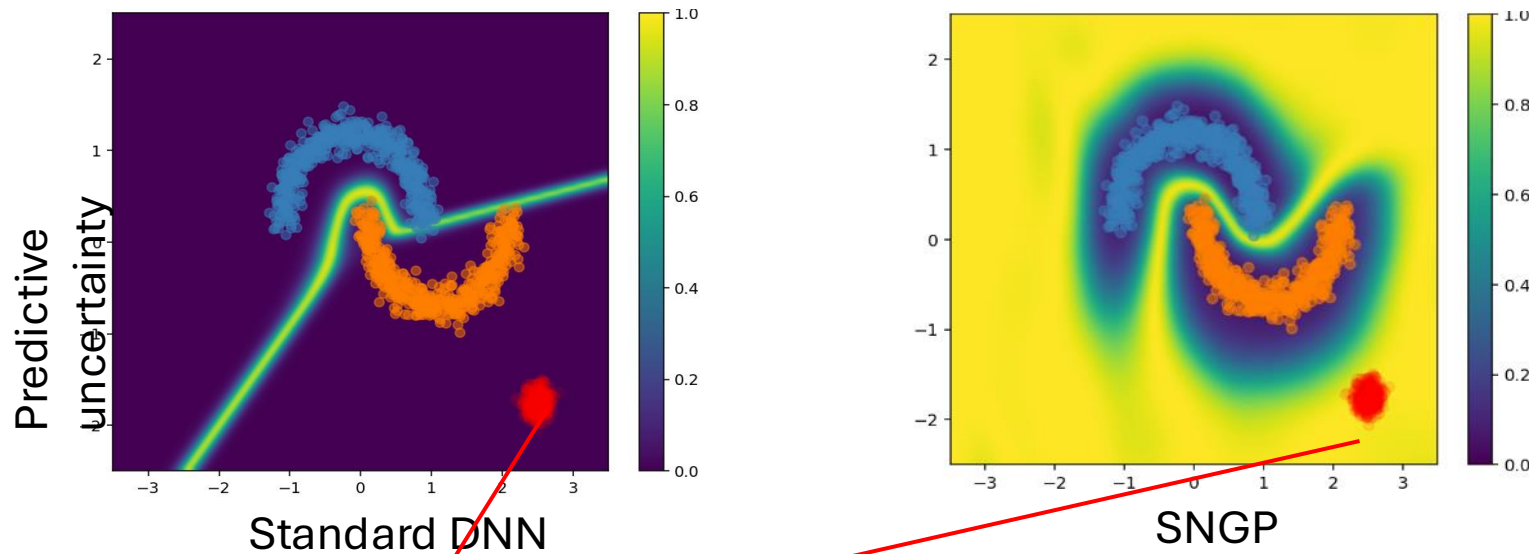
- 1: **Input:**
Minibatches $\{D_i\}_{i=1}^N$ for $D_i = \{y_m, \mathbf{x}_m\}_{m=1}^M$.
 - 2: **Initialize:**
 $\hat{\Sigma} = \mathbf{I}, \mathbf{W}_L \stackrel{iid}{\sim} N(0, 1), \mathbf{b}_L \stackrel{iid}{\sim} U(0, 2\pi)$
 - 3: **for** train_step = 1 **to** max_step **do**
 - 4: SGD update $\{\beta, \{\mathbf{W}_l\}_{l=1}^{L-1}, \{\mathbf{b}_l\}_{l=1}^{L-1}\}$
 - 5: Spectral Normalization $\{\mathbf{W}_l\}_{l=1}^{L-1}$ (10).
 - 6: **if** final_epoch **then**
 - 7: Update precision matrix $\{\hat{\Sigma}_k^{-1}\}_{k=1}^K$ (9).
 - 8: **end if**
 - 9: **end for**
 - 10: Compute posterior covariance $\hat{\Sigma}_k = \text{inv}(\hat{\Sigma}_k^{-1})$.
-

Algorithm 2 SNGP Prediction

- 1: **Input:** Testing example \mathbf{x} .
 - 2: Compute Feature:
 $\Phi_{D_L \times 1} = \sqrt{2/D_L} * \cos(\mathbf{W}_L h(\mathbf{x}) + \mathbf{b}_L),$
 - 3: Compute Posterior Mean:
 $\text{logit}_k(\mathbf{x}) = \Phi^\top \beta_k$
 - 4: Compute Posterior Variance:
 $\text{var}_k(\mathbf{x}) = \Phi^\top \hat{\Sigma}_k \Phi.$
 - 5: Compute Predictive Distribution:
$$p(y|\mathbf{x}) = \int_{m \sim N(\text{logit}(\mathbf{x}), \text{var}(\mathbf{x}))} \text{softmax}(m)$$
-

Spectral Normalized Neural Gaussian Process

- Models should be distance aware: uncertainty increases farther from training data.
- Replace output layer with GP layer.
- Apply spectral normalization to preserve input distances within internal layers.



Important for out-of-domain detection

Spectral Normalized Neural Gaussian Process

| Method | Accuracy (\uparrow) | ECE (\downarrow) | NLL (\downarrow) | OOD | | Latency (\downarrow) (ms / example) |
|---------------|-----------------------------------|-------------------------------------|------------------------------------|------------------------------------|------------------------------------|--|
| | | | | AUROC (\uparrow) | AUPR (\uparrow) | |
| Deterministic | 96.5 ± 0.11 | 0.024 ± 0.002 | 3.559 ± 0.11 | 0.897 ± 0.01 | 0.757 ± 0.02 | 10.42 |
| MC Dropout | 96.1 ± 0.10 | 0.021 ± 0.001 | 1.658 ± 0.05 | 0.938 ± 0.01 | 0.799 ± 0.01 | 85.62 |
| Deep Ensemble | 97.5 ± 0.03 | 0.013 ± 0.002 | 1.062 ± 0.02 | <u>0.964 ± 0.01</u> | <u>0.862 ± 0.01</u> | 84.46 |
| MCD-GP | 95.9 ± 0.05 | 0.015 ± 0.003 | 1.664 ± 0.04 | 0.906 ± 0.02 | 0.803 ± 0.01 | 88.38 |
| DUQ | 96.0 ± 0.04 | 0.059 ± 0.002 | 4.015 ± 0.08 | 0.917 ± 0.01 | 0.806 ± 0.01 | 15.60 |
| DNN-SN | 95.4 ± 0.10 | 0.037 ± 0.004 | 3.565 ± 0.03 | 0.922 ± 0.02 | 0.733 ± 0.01 | 17.36 |
| DNN-GP | 95.9 ± 0.07 | 0.075 ± 0.003 | 3.594 ± 0.02 | 0.941 ± 0.01 | 0.831 ± 0.01 | 18.93 |
| SNGP | <u>96.6 ± 0.05</u> | <u>0.014 ± 0.005</u> | <u>1.218 ± 0.03</u> | 0.969 ± 0.01 | 0.880 ± 0.01 | 17.36 |

Table 4: Results for BERT_{Base} on CLINC OOS, averaged over 10 seeds.

Post-hoc Calibration

Post-hoc Calibration

Histogram Binning

- All uncalibrated predictions \hat{p}_i are divided into mutually exclusive bins B_1, \dots, B_M .
- Each bin is assigned a calibrated score θ_m .

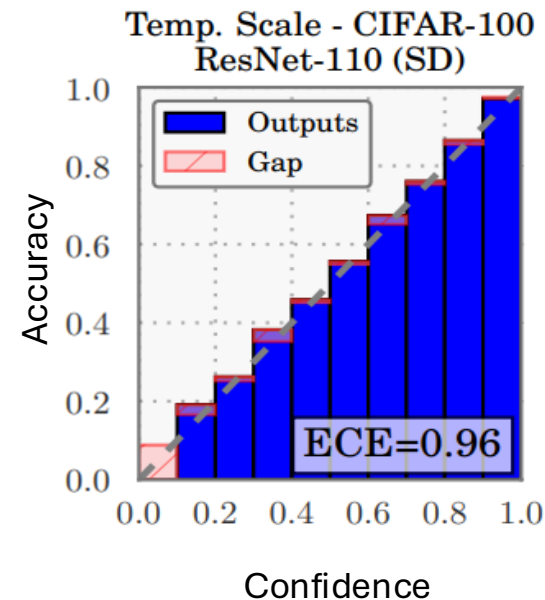
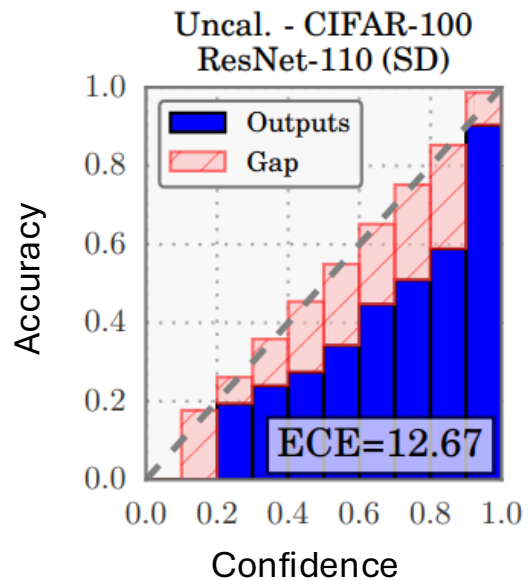
$$\min_{\theta_1, \dots, \theta_M} \sum_{m=1}^M \sum_{i=1}^n \mathbf{1}(a_m \leq \hat{p}_i < a_{m+1}) (\theta_m - y_i)^2,$$

Platt Scaling

- Platt scaling learns scalar parameters $a, b \in \mathbb{R}$
- Calibrated probability: $q_i = \sigma(az_i + b)$
- Parameters a and b are optimized using the NLL loss over the validation set.

Post-hoc Calibration: Temperature Scaling

- Rescale the logits of a pre-trained classifier
 - Initialize a temperature parameter T in the Softmax function:
$$p(y_i|\mathbf{x}) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$
 - Optimize T to minimize the negative log-likelihood on a hold-out calibration set.

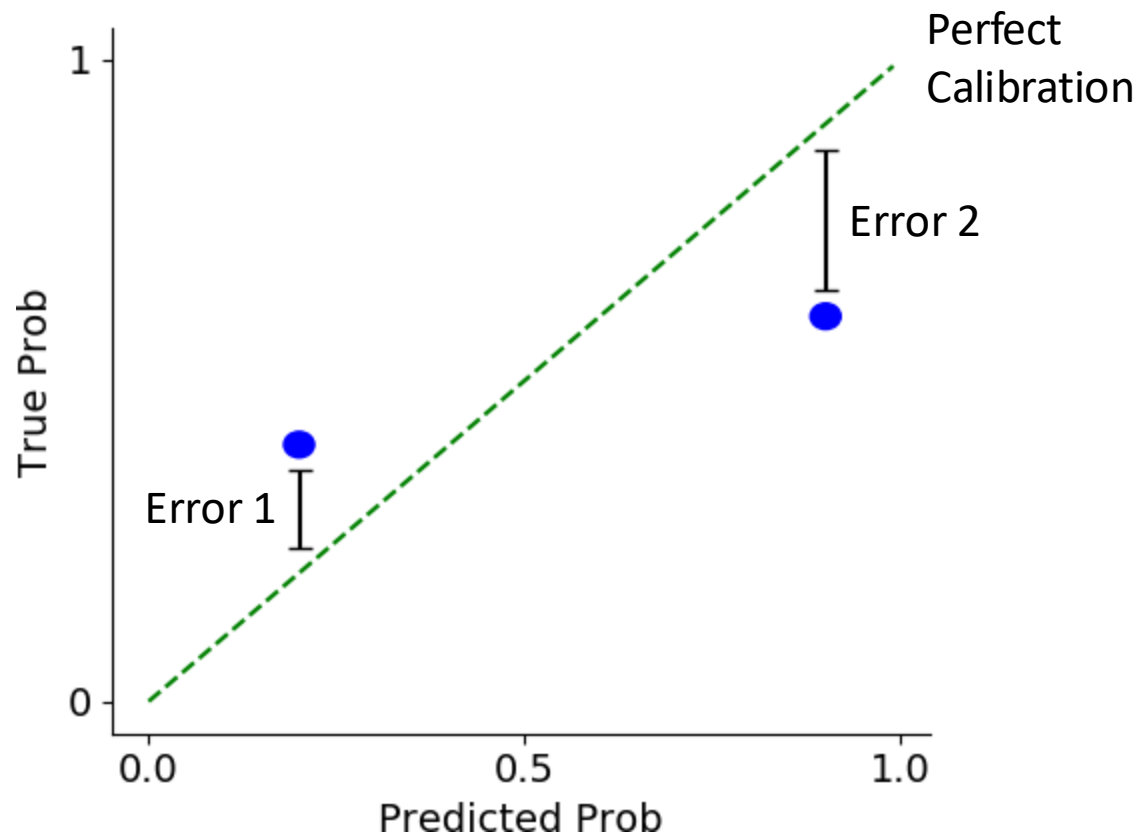


Post-hoc Calibration: Temperature Scaling

| Dataset | Model | Uncalibrated | Hist. Binning | Isotonic | BBQ | Temp. Scaling | Vector Scaling | Matrix Scaling |
|------------------|-----------------|--------------|---------------|--------------|--------------|---------------|----------------|----------------|
| Birds | ResNet 50 | 9.19% | 4.34% | 5.22% | 4.12% | 1.85% | 3.0% | 21.13% |
| Cars | ResNet 50 | 4.3% | 1.74% | 4.29% | 1.84% | 2.35% | 2.37% | 10.5% |
| CIFAR-10 | ResNet 110 | 4.6% | 0.58% | 0.81% | 0.54% | 0.83% | 0.88% | 1.0% |
| CIFAR-10 | ResNet 110 (SD) | 4.12% | 0.67% | 1.11% | 0.9% | 0.6% | 0.64% | 0.72% |
| CIFAR-10 | Wide ResNet 32 | 4.52% | 0.72% | 1.08% | 0.74% | 0.54% | 0.6% | 0.72% |
| CIFAR-10 | DenseNet 40 | 3.28% | 0.44% | 0.61% | 0.81% | 0.33% | 0.41% | 0.41% |
| CIFAR-10 | LeNet 5 | 3.02% | 1.56% | 1.85% | 1.59% | 0.93% | 1.15% | 1.16% |
| CIFAR-100 | ResNet 110 | 16.53% | 2.66% | 4.99% | 5.46% | 1.26% | 1.32% | 25.49% |
| CIFAR-100 | ResNet 110 (SD) | 12.67% | 2.46% | 4.16% | 3.58% | 0.96% | 0.9% | 20.09% |
| CIFAR-100 | Wide ResNet 32 | 15.0% | 3.01% | 5.85% | 5.77% | 2.32% | 2.57% | 24.44% |
| CIFAR-100 | DenseNet 40 | 10.37% | 2.68% | 4.51% | 3.59% | 1.18% | 1.09% | 21.87% |
| CIFAR-100 | LeNet 5 | 4.85% | 6.48% | 2.35% | 3.77% | 2.02% | 2.09% | 13.24% |
| ImageNet | DenseNet 161 | 6.28% | 4.52% | 5.18% | 3.51% | 1.99% | 2.24% | - |
| ImageNet | ResNet 152 | 5.48% | 4.36% | 4.77% | 3.56% | 1.86% | 2.23% | - |
| SVHN | ResNet 152 (SD) | 0.44% | 0.14% | 0.28% | 0.22% | 0.17% | 0.27% | 0.17% |
| 20 News | DAN 3 | 8.02% | 3.6% | 5.52% | 4.98% | 4.11% | 4.61% | 9.1% |
| Reuters | DAN 3 | 0.85% | 1.75% | 1.15% | 0.97% | 0.91% | 0.66% | 1.58% |
| SST Binary | TreeLSTM | 6.63% | 1.93% | 1.65% | 2.27% | 1.84% | 1.84% | 1.84% |
| SST Fine Grained | TreeLSTM | 6.71% | 2.09% | 1.65% | 2.61% | 2.56% | 2.98% | 2.39% |

Calibration Error (CE)

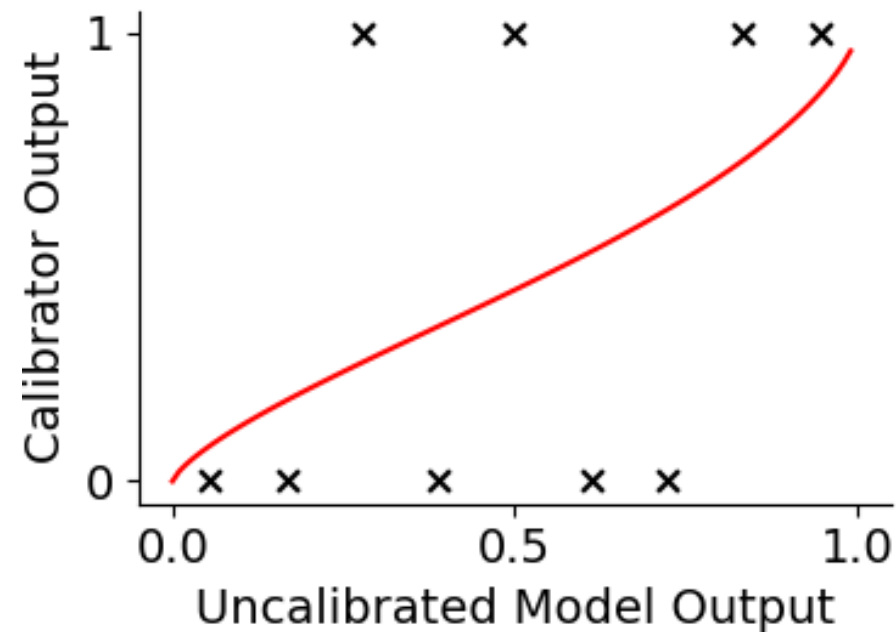
- Average difference between model's predicted prob and true prob



Details: $CE = \sqrt{E[(m - p)^2]}$
m is predicted prob
p is true prob

Platt Scaling

- Platt scaling, temperature scaling scale the model probabilities to improve them



X = labels

Is Platt Scaling calibrated?



Is Model' calibrated?

Impossible to measure calibration error of scaling

Definition 2.1 (Calibration error). *The calibration error of $f : \mathcal{X} \rightarrow [0, 1]$ is given by:*

$$\text{CE}(f) = \left(\mathbb{E} \left[|f(X) - \mathbb{E}[Y | f(X)]|^2 \right] \right)^{1/2}$$

Is Platt Scaling calibrated?

Definition 3.1. *The binned version of f outputs the average value of f in each bin I_j :*

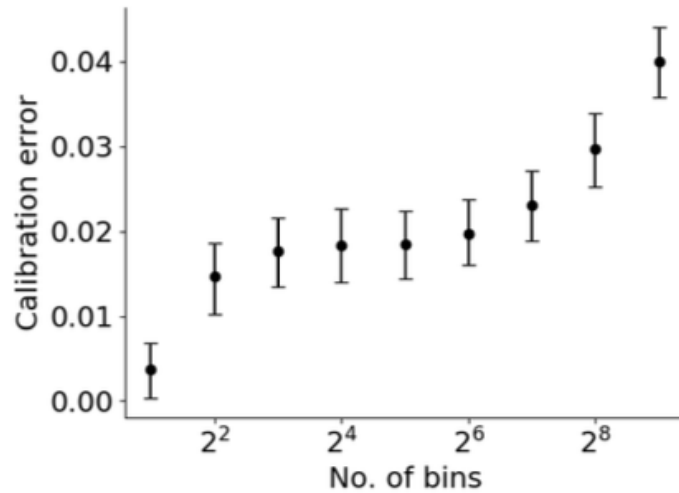
$$f_{\mathcal{B}}(x) = \mathbb{E}[f(X) \mid f(X) \in I_j] \quad \text{where } x \in I_j \quad (4)$$

Given \mathcal{B} , the binned calibration error of f is simply the calibration error of $f_{\mathcal{B}}$. A simple example shows that using binning to estimate the calibration error can severely underestimate the true calibration error.

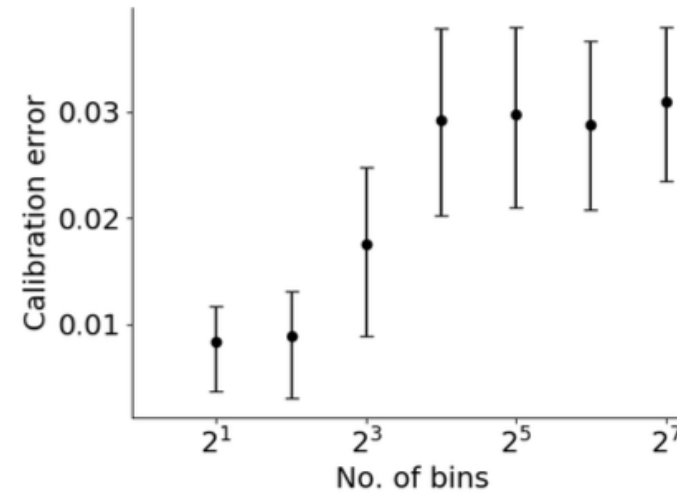
Example 3.2. *For any binning scheme \mathcal{B} , and continuous bijective function $f : [0, 1] \rightarrow [0, 1]$, there exists a distribution P over \mathcal{X}, \mathcal{Y} s.t. $\text{CE}(f_{\mathcal{B}}) = 0$ but $\text{CE}(f) \geq 0.49$. Note that for all f , $0 \leq \text{CE}(f) \leq 1$.*

The intuition of the construction is that in each interval I_j in \mathcal{B} , the model could underestimate the true probability $\mathbb{E}[Y \mid f(X)]$ half the time, and overestimate the probability half the time. So if we average over the entire bin the model appears to be calibrated, even though it is very uncalibrated. The formal proof is in Appendix [B](#), and holds for arbitrary ℓ_p calibration errors including the ECE.

Is Platt Scaling calibrated?



(a) ImageNet



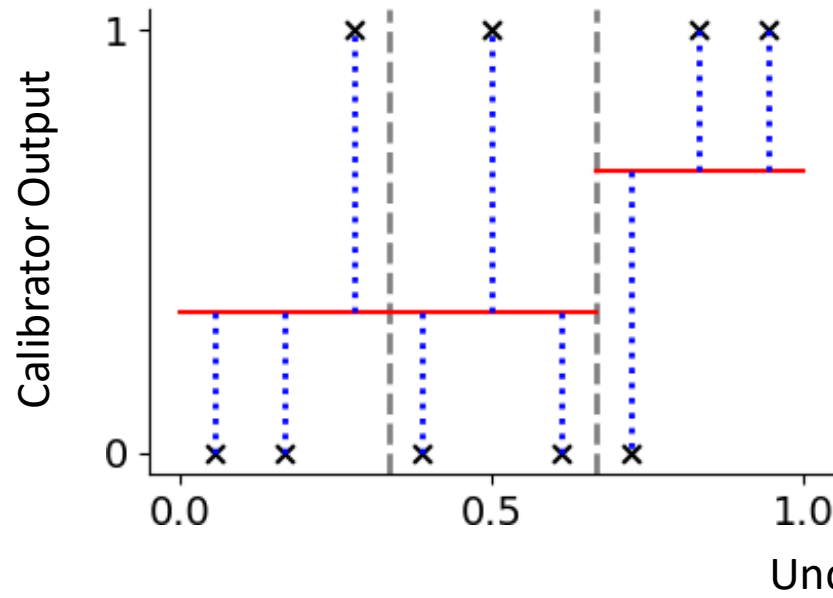
(b) CIFAR-10

Figure 2: Binned calibration errors of a recalibrated VGG-net model on CIFAR-10 and ImageNet with 90% confidence intervals. The binned calibration error increases as we increase the number of bins. This suggests that binning cannot be reliably used to measure the true calibration error.

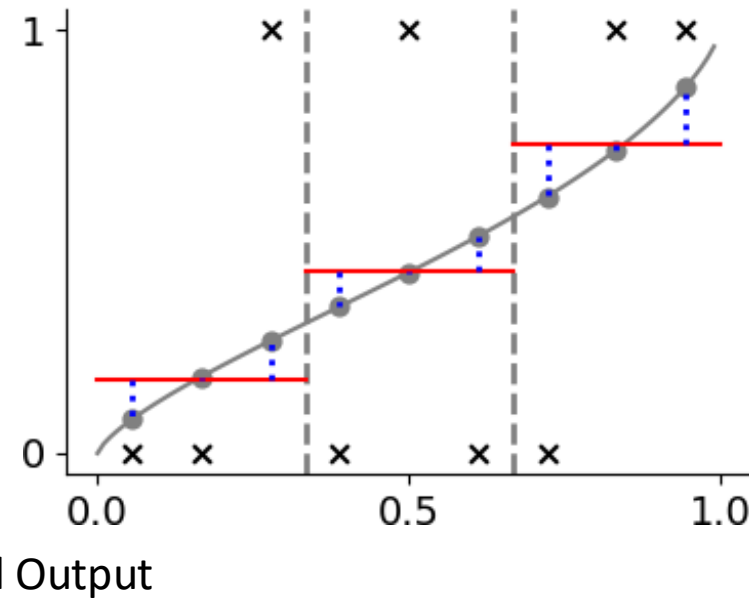
Scaling-Binning Calibrator

- Scaling-binning calibrator fits a function to data, and outputs average function value in each bin

X = labels — = calibrator output



(a) Histogram



(b) Scaling-binning (ours)

Scaling-Binning Calibrator

4.1 Algorithm

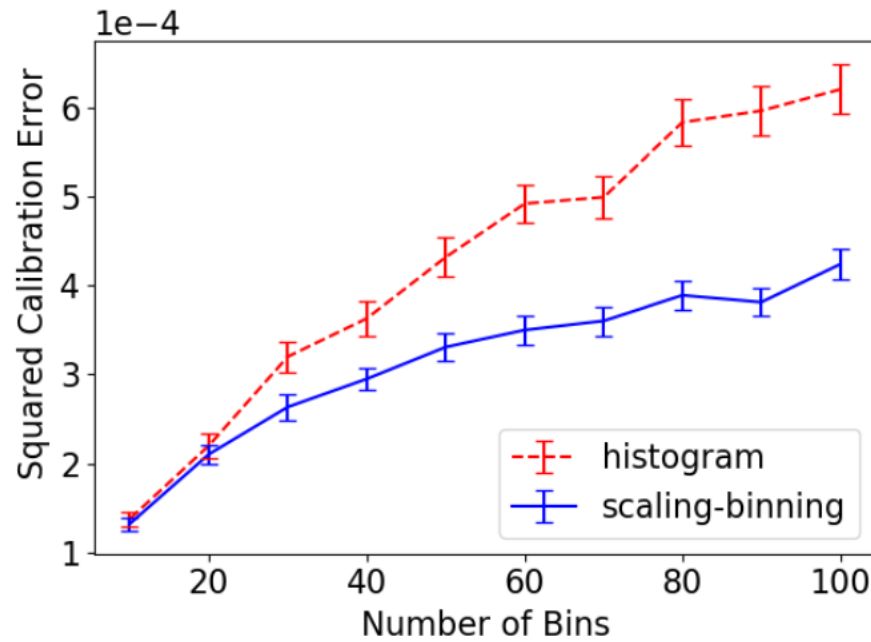
We split the recalibration data T of size n into 3 sets: T_1, T_2, T_3 . The scaling-binning calibrator, illustrated in Figure 1, outputs $\hat{g}_{\mathcal{B}}$ such that $\hat{g}_{\mathcal{B}} \circ f$ has low calibration error:

Step 1 (Function fitting): Select $g = \arg \min_{g \in \mathcal{G}} \sum_{(z,y) \in T_1} (y - g(z))^2$.

Step 2 (Binning scheme construction): We choose the bins so that an equal number of $g(z_i)$ in T_2 land in each bin I_j for each $j \in \{1, \dots, B\}$ —this uniform-mass binning scheme [10] as opposed to equal-width binning [9] is essential for being able to estimate the calibration error in Section 5.

Step 3 (Discretization): Discretize g , by outputting the average g value in each bin—these are the gray circles in Figure 1c. Let $\mu(S) = \frac{1}{|S|} \sum_{s \in S} s$ denote the mean of a set of values S . Let $\hat{\mu}[j] = \mu(\{g(z_i) \mid g(z_i) \in I_j \wedge (z_i, y_i) \in T_3\})$ be the mean of the $g(z_i)$ values that landed in the j -th bin. Recall that if $z \in I_j$, $\beta(z) = j$ is the interval z lands in. Then we set $\hat{g}_{\mathcal{B}}(z) = \hat{\mu}[\beta(g(z))]$ —that is we simply output the mean value in the bin that $g(z)$ falls in.

Scaling-Binning Calibrator



(a) Effect of number of bins on squared calibration error.

Scaling-Binning Calibrator

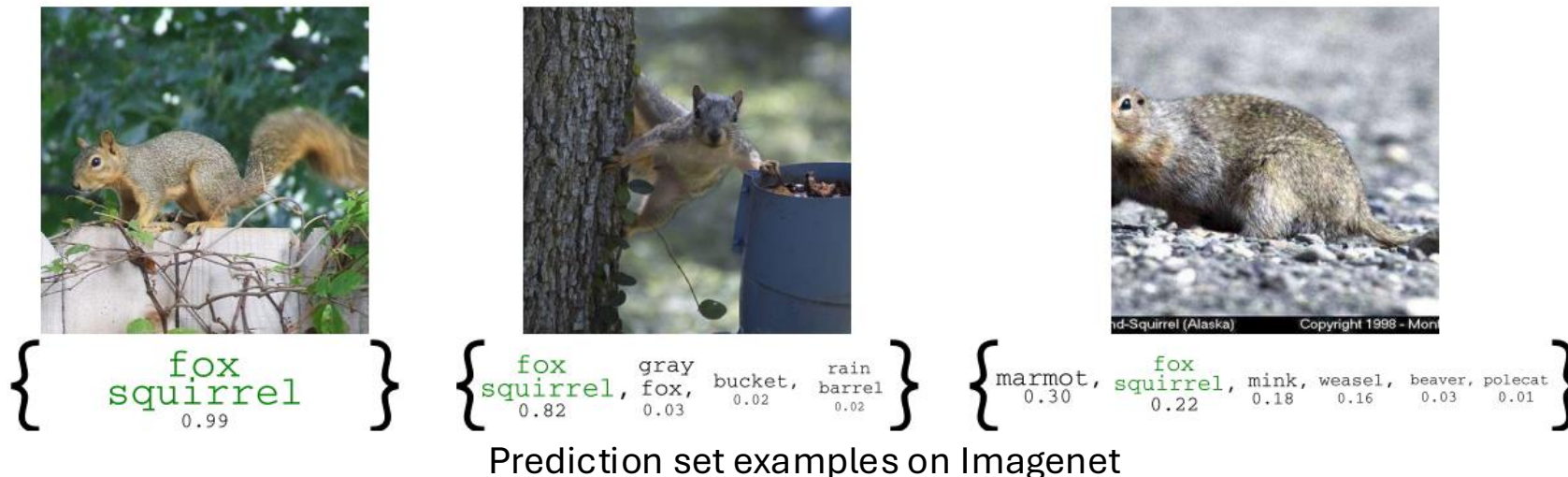
| <i>Recalibration Method</i> | <i>Samples Needed</i> | <i>Can Estimate Calibration?</i> |
|-----------------------------|--|----------------------------------|
| Platt Scaling | Few: $O\left(\frac{1}{\epsilon^2}\right)$ | ✗ |
| Histogram Binning | More: $O\left(\frac{B}{\epsilon^2}\right)$ | ✓ |
| Scaling-Binning (Ours) | Few: $O\left(\frac{1}{\epsilon^2} + B\right)$ | ✓ |

B = # Bins
 ϵ = desired CE

Conformal Prediction

Conformal Prediction: Motivation

- Predict a small set of plausible answers which provably contains the right one
 - Classification: a collection of discrete choices
 - Regression: a continuous interval
- Benefits
 - Can be applied on top of any pre-trained predictor.
 - **Finite-sample distribution free** coverage guarantees.



Conformal Prediction: Notations

- Given n i.i.d examples $(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{X} \times \mathcal{Y}$ and a user-chosen error rate $\alpha \in [0, 1]$ for a new input \mathbf{x}_{n+1} , return a set of predictions $\mathcal{C}(\mathbf{x}_{n+1}) \subseteq \mathcal{Y}$

- The predictor is **valid** if the prediction set contains the correct label with probability at least $1 - \alpha$:

$$\mathbb{P}(\mathbf{y}_{n+1} \in \mathcal{C}(\mathbf{x}_{n+1})) \geq 1 - \alpha$$

- The predictor is **efficient** if:

$$\mathbb{E}[|\mathcal{C}(\mathbf{x}_{n+1})|] \ll |\mathcal{Y}|$$

$\mathbf{x} =$



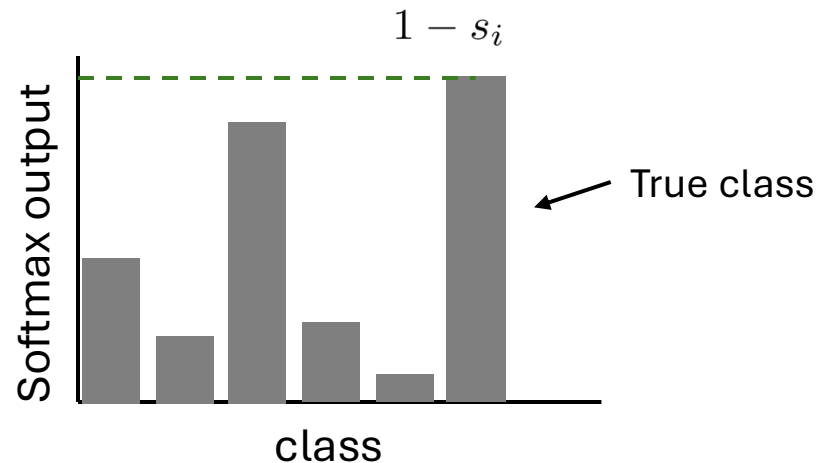
The prediction set

$$\mathcal{C}(\mathbf{x}) = \left\{ \begin{array}{l} \text{fox squirrel} \\ \text{gray fox} \\ \text{bucket} \\ \text{rain barrel} \end{array} \right\}$$

\mathbf{y} ←

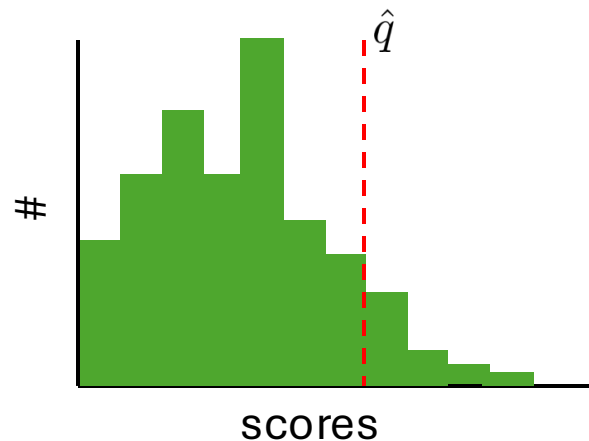
Conformal Prediction: Working Steps

- Given a pre-trained predictor, compute the conformal score $s(\mathbf{x}, y) \in \mathbb{R}$ on the calibration set.
 - Measure the discrepancy between model outputs and ground-truth labels.
 - Can simply be the loss of an uncalibrated pre-trained predictor.
 - E.g., one minus the softmax output of the true class.



Conformal Prediction: Working Steps

- Given a pre-trained predictor, compute the conformal score $s(\mathbf{x}, y) \in \mathbb{R}$ on the calibration set
- Compute \hat{q} as the $\frac{\lceil (n+1)(1-\alpha) \rceil}{n}$ quantile of the calibration scores
 $s_1 = s(\mathbf{x}_1, y_1), \dots, s_n = s(\mathbf{x}_n, y_n).$



Conformal Prediction: Working Steps

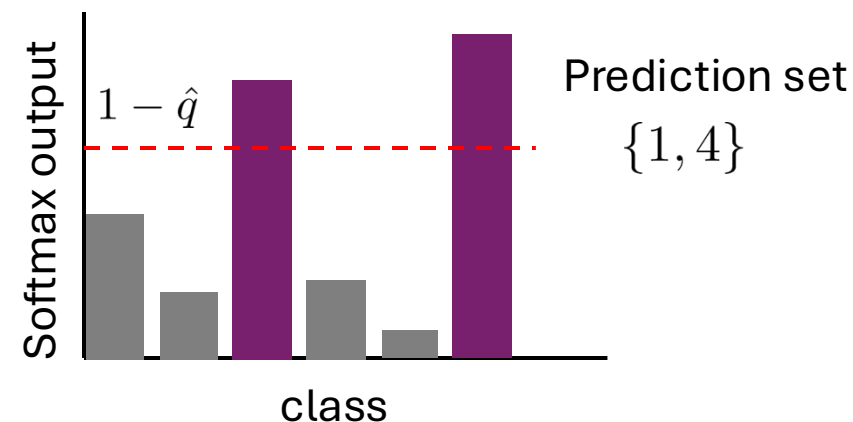
- Given a pre-trained predictor, compute the conformal score $s(\mathbf{x}, y) \in \mathbb{R}$ on the calibration set.

- Compute \hat{q} as the $\frac{\lceil (n+1)(1-\alpha) \rceil}{n}$ quantile of the calibration scores

$$s_1 = s(\mathbf{x}_1, y_1), \dots, s_n = s(\mathbf{x}_n, y_n).$$

- Use this quantile to form the prediction sets for new examples:

$$\mathcal{C}(\mathbf{x}_{n+1}) = \{y : s(\mathbf{x}_{n+1}, y) \leq \hat{q}\}$$



Conformal Prediction yields Valid Sets

- A key formal result in conformal prediction:

Theorem (Vovk et. al., 2005). *Let $(X_i, Y_i) \in \mathcal{X} \times \mathcal{Y}$, $i = 1, \dots, n+1$ be exchangeable random variables drawn from some distribution P . For any symmetric nonconformity measure \mathcal{S} and tolerance level $\epsilon \in (0, 1)$, define the conformal set (based on the first n samples) at $x \in \mathcal{X}$ as*

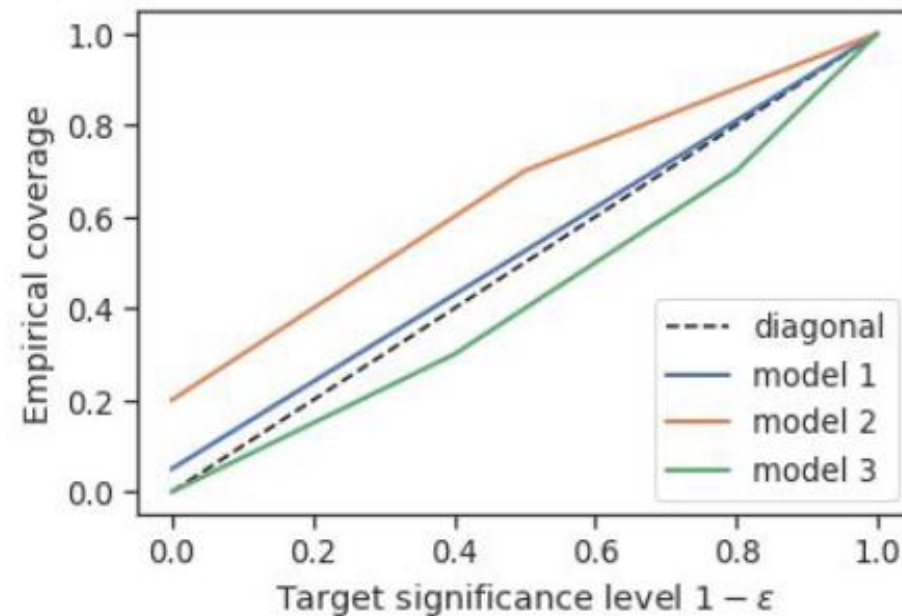
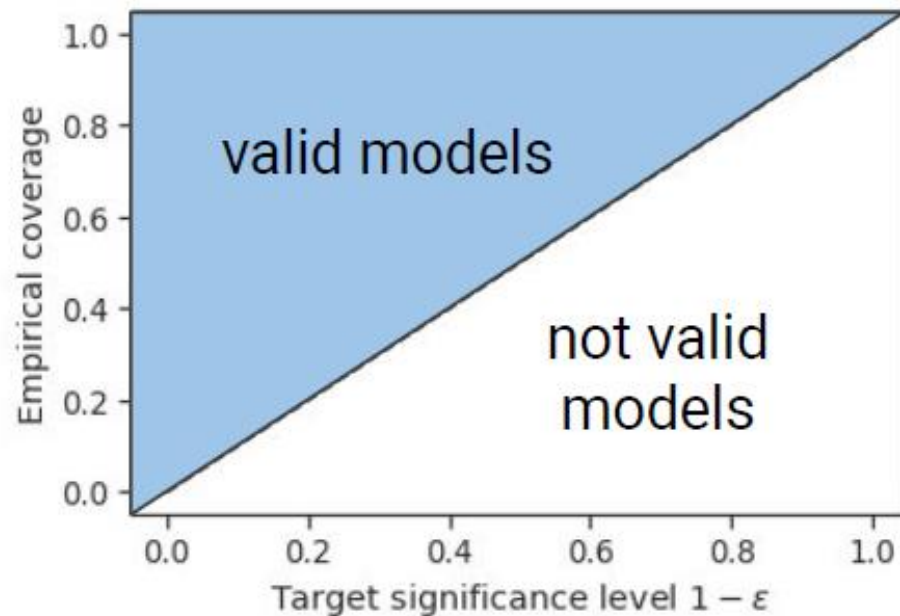
$$C_n(x) := \left\{ y \in \mathcal{Y} : V_{n+1}^{(x,y)} \leq \text{Quantile}(1 - \epsilon; V_{1:n}^{(x,y)} \cup \{\infty\}) \right\}, \quad (1)$$

where $V_i^{(x,y)}$ is the nonconformity score assigned by \mathcal{S} to point $(X_i = x, Y_i = y)$. Then C_n satisfies

$$\mathbb{P}(Y_{n+1} \in C_n(X_{n+1})) \geq 1 - \epsilon. \quad (2)$$

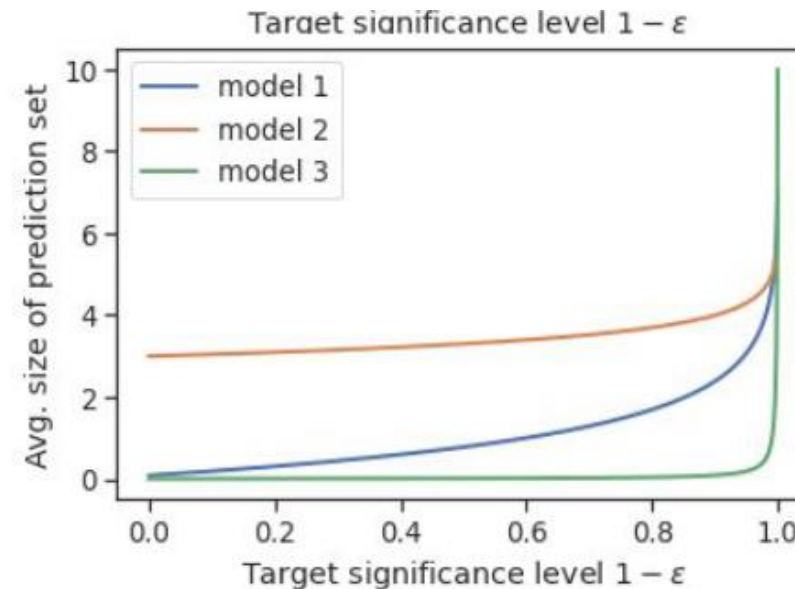
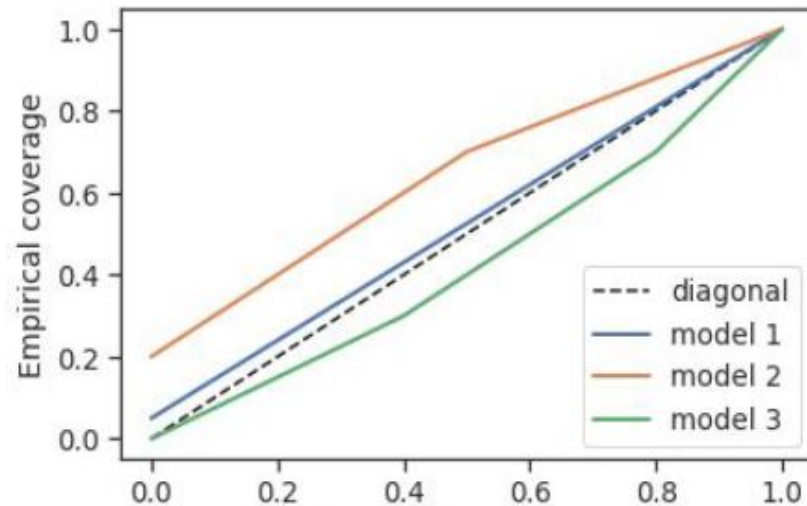
Evaluation of Conformal Prediction

- Empirically verifying the model's validity (anything above the diagonal)
 - Both model 1 and 2 are valid. Though, model 2 might be too conservative...
 - Model 3 is not valid.



Evaluation of Conformal Prediction

- Measuring the size of prediction sets (the lower the better).
 - While both models 1 and 2 are valid, model 1 is better because the prediction sets are smaller/tighter.
 - Model 3 is not valid, therefore, its small prediction sets aren't reliable.



Summary: Existing Methods for Predictive Models

- Bayesian neural networks:
 - Model parameters as probability distributions.
- Ensemble learning:
 - Train multiple models with different initializations.
- Neural Gaussian process:
 - Deep feature extractor + Gaussian process.
- Post-hoc Calibration:
 - Fit a calibration parameter on a held-out calibration dataset.
- Conformal Prediction:
 - Output set that contains correct answer with high probability.

No clear best method, choose the right one based on your problem and data!

References

- Primary references:
 - KDD 2023 tutorial: https://lingkai-kong.com/kdd23_tutorial/
 - Neurips 2020 tutorial: <https://neurips.cc/virtual/2020/tutorial/16649>
- Other references (good to read):
 - Coling 2022 tutorial: <https://sites.google.com/view/uncertainty-nlp>