Uncertainty Quantification in Deep Learning

Gianni FRANCHI,[†] Olivier Laurent

[†] *gianni.franchi@ensta-paris.fr*

Helmholtz AI 2024

Planning of the tutorial

Planning of the tutorial

- 30 min - Efficient Ensemble presentation
- 20 min - Efficient Ensemble hands on with TorchUncertainty
- 20 min - Pause
- 30 min - Bayesian Neural Network presentation
- 45 min - Bayesian Neural Network hands on with TorchUncertainty

Why Quantify Uncertainty in Deep Neural Networks?

Context

- Deep Neural Networks (DNNs) have achieved remarkable success in various applications, but their predictions are not infallible.
- Recognizing and quantifying uncertainty is crucial for enhancing the reliability and trustworthiness of DNNs.

Motivation

- **Real-world Consequences:** In critical applications such as healthcare or autonomous systems, incorrect predictions can have severe consequences.
- **Decision-Making:** Users and decision-makers need to understand the confidence levels associated with DNN predictions.

Types of Uncertainty in Machine Learning

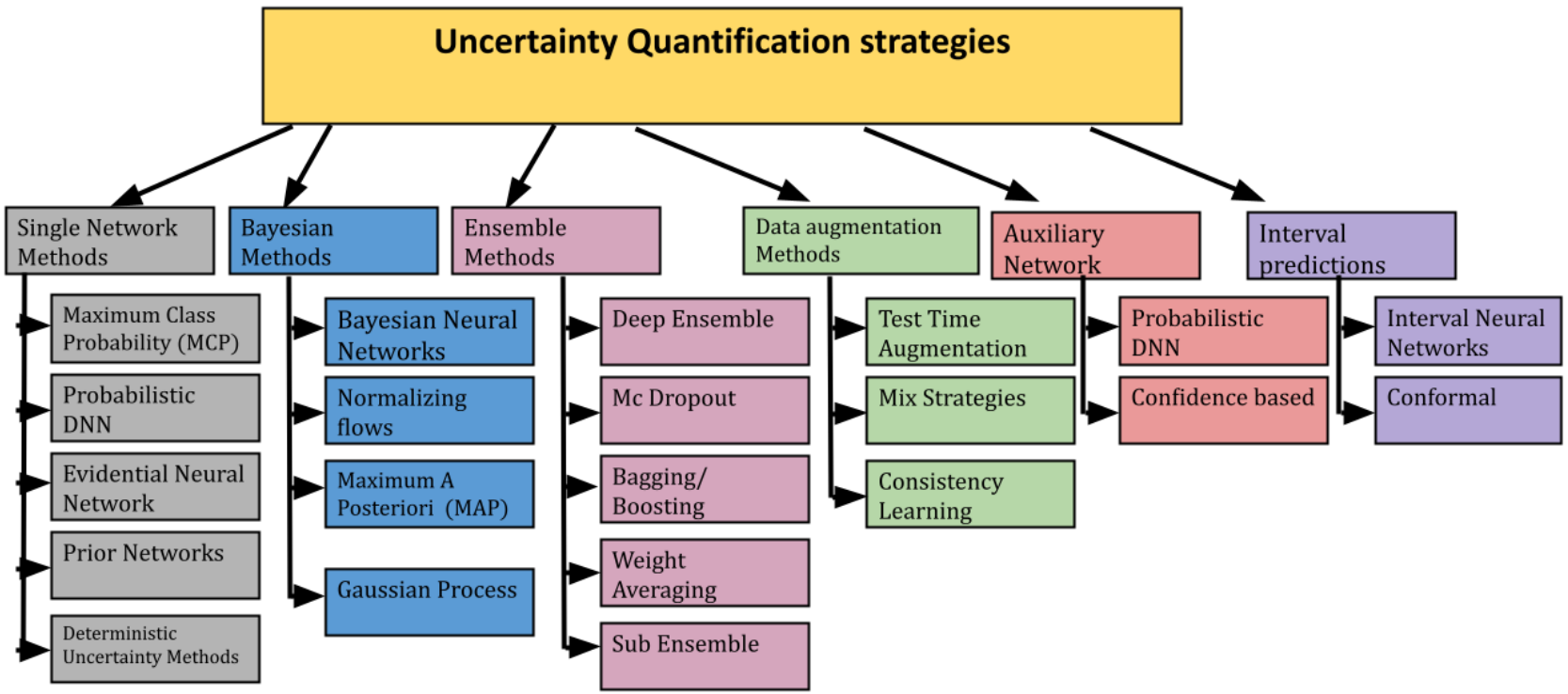
Aleatoric Uncertainty

- **Data Uncertainty:** Arises from inherent variability in the data. It can be further classified into homoscedastic (constant variance) and heteroscedastic (varying variance) uncertainty.
- **Measurement Uncertainty:** Associated with errors in the measurement process, impacting the reliability of observed data.

Epistemic Uncertainty

- **Model Uncertainty:** Arises from a lack of knowledge about the true model structure. It can be reduced with more data and better model architecture.
- **Inherent Model Limitations:** Uncertainty arising from the inability of the model to capture all relevant aspects of the underlying data distribution.
- **Parameter Uncertainty:** Related to uncertainty in the values of model parameters, often addressed through techniques like Bayesian modeling.

Uncertainty Quantification Strategies



Single Network Methods

Notations

- We consider that we have a training dataset $\mathcal{D} := \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\} \subset \mathcal{X} \times \mathcal{Y}$,
- $(\boldsymbol{x}_i, y_i)$ are assumed i.i.d. according to some unknown probability measure $P_{\mathcal{X}\times\mathcal{Y}}$ on $\mathcal{X} \times \mathcal{Y}$
- We denote $f_\omega(\boldsymbol{x})$ the prediction a DNN model with weights ω. We consider that $f_\omega(\boldsymbol{x}) = P(y|\boldsymbol{x}, \boldsymbol{\omega})$

Maximum Likelihood Estimation for Classification

Our goal is to find $\boldsymbol{\omega}$ that maximizes the Likelihood $P(\mathcal{D}|\boldsymbol{\omega})$.
Let us consider the case of i.i.d. samples from the conditional distribution.
Then, we can write the likelihood function of $\boldsymbol{\omega}$:

$$\boldsymbol{\omega} = \arg\max_{\boldsymbol{\omega}} P(\mathcal{D}|\boldsymbol{\omega}) \approx \arg\max_{\boldsymbol{\omega}} \sum_{i=1}^{N} \log P(y_i|\boldsymbol{x}_i, \boldsymbol{\omega}) \tag{1}$$

Ensemble Methods Overview

Ensemble Methods Overview

- Ensemble methods combine multiple base models to create a stronger, more robust predictive model.
- They are widely used to improve generalization and performance in various machine learning tasks.
- Can be used to quantify the uncertainty

Ensemble Methods Overview

Ensemble Methods Overview

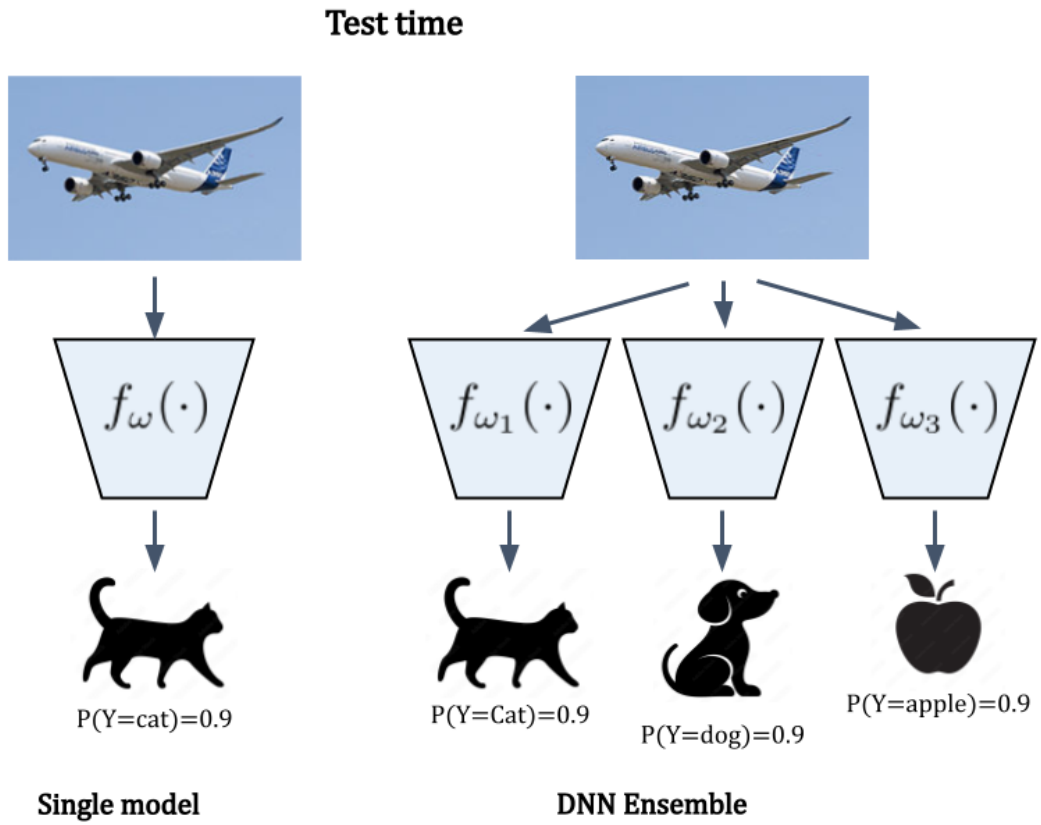
- **Bagging (Bootstrap Aggregating):** Involves training multiple instances of the same model using different subsets of the training data and averaging their predictions.
- **Random Forests:** Adapts the traditional random forest concept to neural networks, creating an ensemble of decision trees or models.
- **Boosting:** Sequentially trains multiple weak learners, giving more weight to misclassified instances in each iteration to improve overall model performance.
- **Stacking:** Involves training multiple diverse models and combining their predictions using another model (meta-learner).
- **Dropout:** During training, it randomly drops out neurons, effectively training an ensemble of slightly different models

Ensemble Methods Overview

Ensemble Methods Overview

- **Snapshot Ensembles:** Involves saving multiple snapshots of a model during training and using these snapshots as an ensemble for making predictions.
- **Bayesian Neural Networks (BNNs):**Introduces uncertainty by treating weights as probability distributions, providing a Bayesian approach to ensembling.
- **Deep Ensembles:** It ensembles multiple independently trained neural networks to improve generalization and reduce overfitting.
- **Weak Ensembles:** Methods that extend the ensemble concept to the batch dimension during training, and that ensemble smaller DNNs.

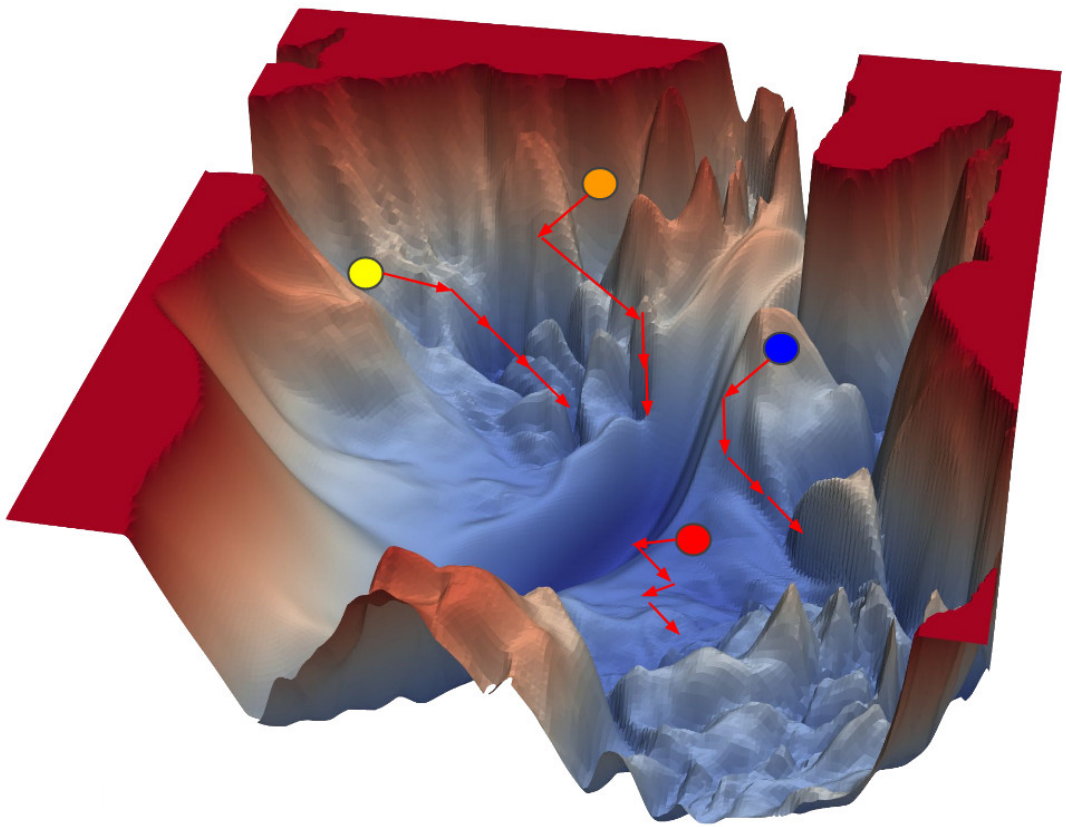
Motivation for Ensemble Methods



Deep Ensembles (DE) [3]

The authors of DE [3] propose to average the predictions of several DNNs with different initial seeds:

$$\mathcal{P}(y^*|x^*) = \frac{1}{N_{\text{model}}} \sum_{j=1}^{N_{\text{model}}} \mathcal{P}(y^*|\omega^j(t^*), x^*) \tag{2}$$



Deep Ensembles [11]

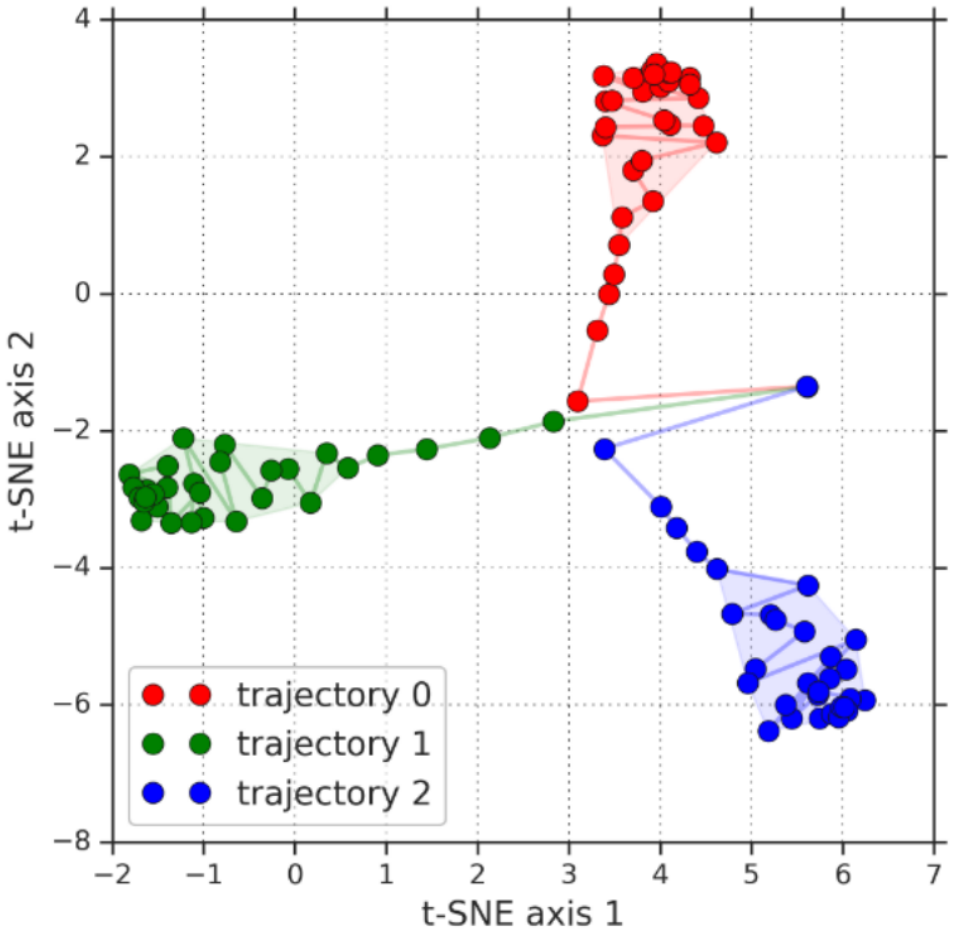


Figure: t-SNE plot of predictions from checkpoints corresponding to 3 different randomly initialized trajectorie

Deep Ensembles [11]

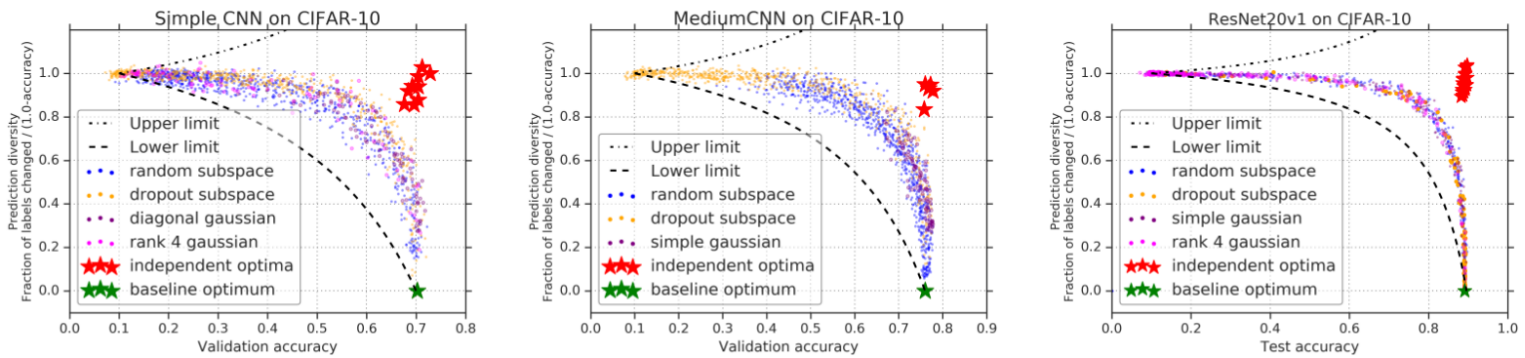


Figure: Diversity versus accuracy plots for 3 models trained on CIFAR-10

Overview of Weak Ensemble Methods

Overview of Weak Ensemble Methods

- While Deep Ensemble is frequently considered state-of-the-art (SOTA), it comes with significant computational demands.
- Weak Ensemble methods offer a faster alternative to achieve comparable results.
- Weak Ensemble methods can be performed on a reduced dataset, or/and with fewer neurons, or/and for a shorter duration.

BatchEnsemble Overview [6]

What is BatchEnsemble?

- **Definition:** BatchEnsemble is an ensemble learning technique designed for improving the performance and robustness of neural networks.
- **Inspiration:** Inspired by ensemble methods, BatchEnsemble extends the concept to the batch dimension during training.

How BatchEnsemble Works

- **Batch-Level Ensembling:** Instead of ensembling models across different training runs, BatchEnsemble ensembles within the same training batch.
- **Variability Across Batches:** Introduces diversity by training multiple instances of the model within each batch, enhancing robustness.

BatchEnsemble Overview [6]

They [6] propose to approximate the average of the predictions of several DNN with different initial seeds by using a DNN with two king of weights. For simplicity is the ω has two set of weight ω^{slow} , ω^{fast} For simplicity let us consider a DNN with just one fully connected layer and let us write $\omega = \{\omega_j\}_{j=1}^{N_{\text{model}}} = \{W_j\}_{j=1}^{N_{\text{model}}}$ and $\omega^{\text{slow}} = W$ and $\omega^{\text{slow}} = \{F_j\}_{j=1}^{N_{\text{model}}}$. We have $W_j = W \cdot F = W \cdot (r_j s_j^t)$

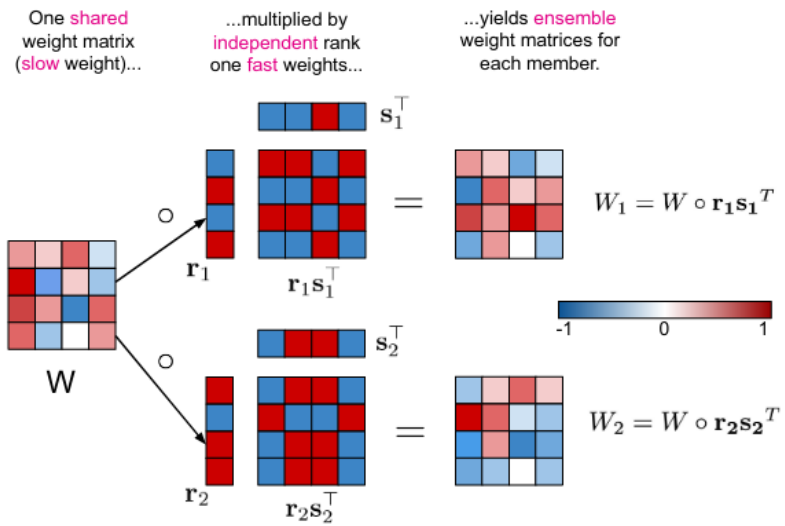


Figure: An illustration on how to generate the ensemble weights for two ensemble members

BatchEnsemble Overview [6]

We have a set of weight $W_j = W \cdot F = W \cdot (r_j s_j^t)$ with W that sees all images and $(r_j s_j^t)$ that does not see all the same images. If we denote ϕ an activation function then when we apply the BatchEnsemble on an image we perform:

$$y = \phi\left(W_j^t x\right) = \phi\left((W^t \cdot (r_j s_j^t))^t x\right) = \phi\left((W^t (x \cdot r_j) \cdot s_j)\right)$$

Similarly to Deep Ensembles, to perform inference we just perform ensembling :

$$\mathcal{P}(y^*|x^*) = \frac{1}{N_{\text{model}}} \sum_{j=1}^{N_{\text{model}}} \mathcal{P}(y^*|\omega_j, x^*) \tag{3}$$

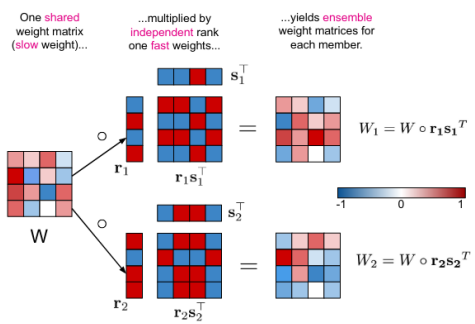


Figure: An illustration on how to generate the ensemble weights for two

MIMO Overview [12]

What is MIMO?

- **Definition:** MIMO stands for Multiple Input Multiple Output .
- **Objective:** MIMO aims to utilize a single model's capacity to train multiple subnetworks that independently learn the task at hand.

Key Concepts of MIMO [12]

How MIMO Works

- **MIMO principle:** The lottery ticket hypothesis shows that one can prune away 70-80% of the connections in a DNN without adversely affecting performance
- **MIMO Idea:**The basic Idea is that a neural network has sufficient capacity to fit 3-4 independent subnetworks simultaneously. Hence they just need to modify the input and output to handle this 3-4 subnetworks.

Key Concepts of MIMO [11]

```
class MIMOModel(nn.Module):
    def __init__(self, hidden_dim: int = 784, ensemble_num: int = 3):
        super(MIMOModel, self).__init__()
        self.input_layer = nn.Linear(hidden_dim, hidden_dim * ensemble_nu
        self.backbone_model = BackboneModel(hidden_dim, ensemble_num)
        self.ensemble_num = ensemble_num
        self.output_layer = nn.Linear(128, 10 * ensemble_num)

    def forward(self, input_tensor: torch.Tensor):
        input_tensor = input_tensor.transpose(1, 0).view(
            batch_size, self.ensemble_num, -1)
        # (batch_size, ensemble_num, hidden_dim)
        input_tensor = self.input_layer(input_tensor)
        # (batch_size, ensemble_num, hidden_dim * ensemble_num)
        # usual model forward
        output = self.backbone_model(input_tensor)  # (batch_size, ensemb
        output = self.output_layer(output)  # (batch_size, ensemble_num,
        output = output.reshape(
            batch_size, ensemble_num, -1, ensemble_num
        )  # (batch_size, ensemble_num, 10, ensemble_num)
        output = torch.diagonal(output, offset=0, dim1=1, dim2=3).transp
# (batch_size, ensemble_num, 10)
        output = F.log_softmax(output, dim=-1)  # (batch_size, ensemble_n
        return output
```

Key Concepts of MIMO [12]

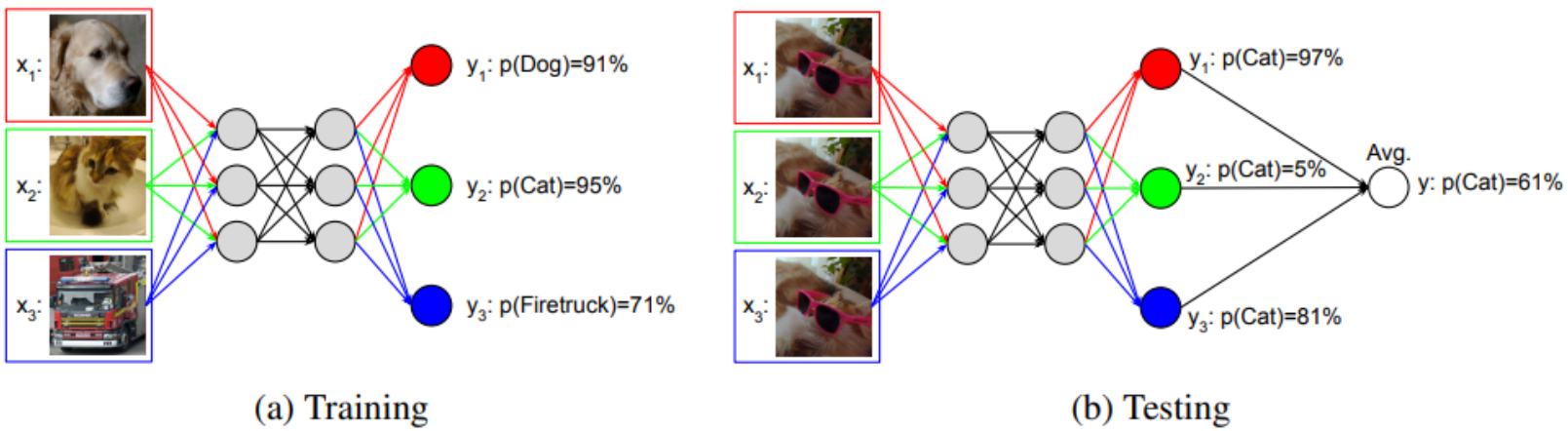


Figure: The multi-input multi-output (MIMO) configuration, the network takes M = 3 inputs and gives M outputs [12]

Packed-Ensembles Overview [13]

Seamless training of ensembles with Packed-Ensembles

- **Definition:** Packed-Ensembles estimate the posterior distributions restraining their support to smaller networks in a computationally efficient manner with grouped convolutions.
- **Objective:** Get the benefits of *deep ensembles* with reduced costs.

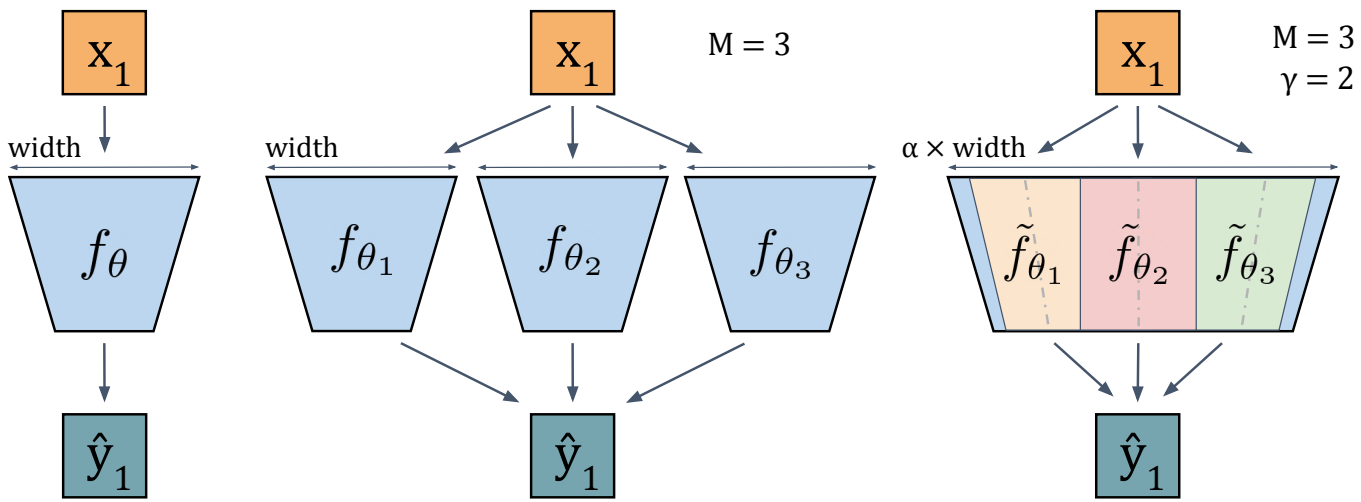


Figure: **Left:** A standard network, **Center:** A *deep ensembles*, **Right:** The corresponding Packed-Ensembles

How well does Packed-Ensembles perform? [13]

Performance of Packed-Ensembles

- **Performance:** For sufficiently large networks, Packed-Ensembles is equivalent to *deep-ensembles* in performance and UQ.
- **Computational efficiency:** Use Packed-Ensembles with *float16* to benefit from grouped-convolutions better.

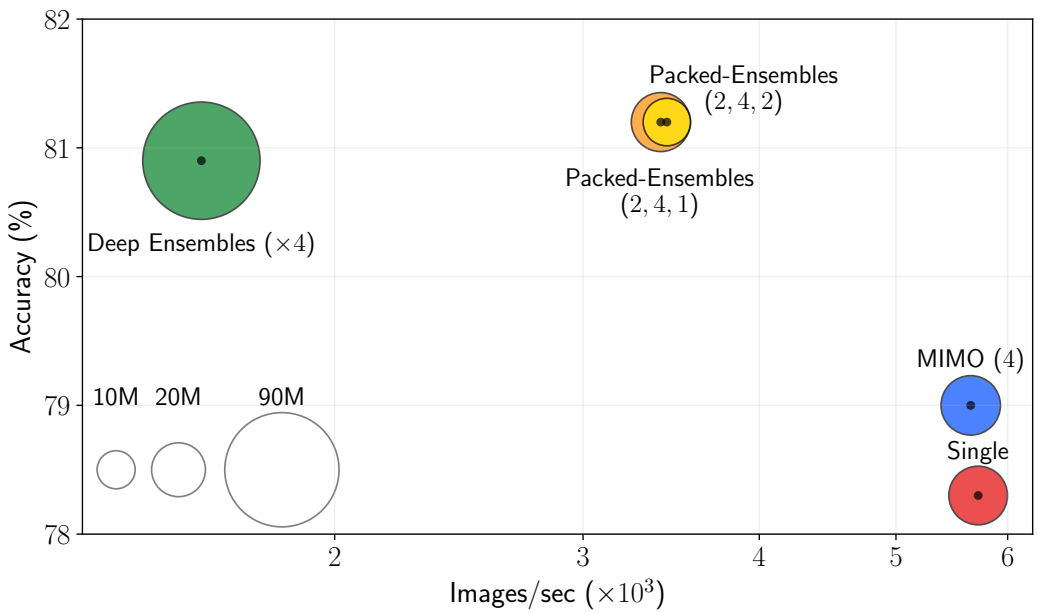


Figure: Performance (accuracy) wrt. the image throughput

Sources of stochasticity in deep ensembles [13]

Stochasticity			ResNet-50				
ND	**DI**	**DB**	**Acc** (↑)	**ECE** (↓)	**AUPR** (↑)	IDMI	OODMI
-	-	-	77.63±0.23	0.0825±0.0018	89.19±0.65	0±0	0±0
✓	-	-	80.94±0.10	0.0179±0.0010	90.23±0.62	0.1513	0.4022
-	✓	-	81.01±0.06	0.0202±0.0011	91.10±0.39	0.1524	0.4088
-	-	✓	80.87±0.10	0.0178±0.0010	90.80±0.30	0.1505	0.4115
✓	✓	✓	81.08±0.08	0.0198±0.0013	90.68±0.25	0.1534	0.4031

Figure: Impact of the three sources of stochasticity, non-deterministic backdrop. kernels (**ND**), different initialization (**DI**), and different batches (**DB**).

Uncertainty-sources are equivalent!

No source of stochasticity during training seems to single out. Having one source is sufficient, and adding more does not seem to affect the performance or the quantitative functional diversity (Mutual information).

Quantifying Uncertainty in DNNs

Criteria for Uncertainty Assessment

- **Single DNN:**
  - **Maximum Class Probability (MCP):**
    - $max_{k \subset \mathcal{Y}} P(Y = k|x, w)$
    - Higher MCP implies higher confidence, while lower MCP indicates increased uncertainty.
  - **Entropy:**
    - $\mathrm{H}(P(Y|x, w)) := -\sum_{k \in \mathcal{Y}} P(Y = k|x, w) \log P(Y = k|x, w))$
    - Higher entropy signifies higher uncertainty as it reflects a more uniform distribution of probabilities. (related to aleatoric uncertainty)
- **Ensemble of DNNs:**
  - **Maximum Class Probability (MCP):** Similar to single DNN, but now considering the MCP across the marginalized distribution $P(Y = k|x) = \int \mathcal{P}(Y|X, \boldsymbol{\omega})\mathcal{P}(\boldsymbol{\omega}|\mathcal{D})d\boldsymbol{\omega}$.
  - **Entropy:** $\mathrm{H}(P(Y|x)) := -\sum_{k \in \mathcal{Y}} P(Y|x) \log P(Y|x))$
  - **Mutual Information:** Measures the shared information between predictions of individual models, offering insights into epistemic uncertainty: $I(Y|x) = \mathrm{H}(P(Y = k|x)) - E_{\mathcal{P}(\boldsymbol{\omega}|\mathcal{D}}\mathrm{H}(P(Y = k|\boldsymbol{\omega}, x))$

Evaluating Uncertainty Quantification in DNNs

Evaluating Uncertainty in DNNs

- Evaluating the quality of Uncertainty quantification is crucial for reliable deep learning models.
- We distinguish between aleatoric uncertainty, epistemic uncertainty, and distribution shift, each requiring specific evaluation metrics.

Introduction to ECE

Expected Calibration Error (ECE) for classification

- **Definition:** The Expected Calibration Error (ECE) is a metric used to assess the calibration of predicted probabilities in classification tasks.
- **Importance:** Calibration is crucial for models that provide probability estimates, ensuring that predicted confidence scores align with actual outcomes.

Introduction to ECE

ECE Formula and Interpretation

- **Formula:** We begin by partitioning the data into m bins based on the confidence scores of the DNN predictions. B_i represents the collection of samples whose predicted probabilities fall within the i-th bin.

$$ECE = \sum_{i=1}^{m} \frac{|B_i|}{N} \cdot |\text{accuracy}(B_i) - \text{confidence}(B_i)|$$

- **Interpretation:**
  - A perfectly calibrated model has ECE = 0, indicating precise alignment between predicted and actual probabilities.
  - Higher ECE values suggest miscalibration, revealing discrepancies between predicted confidence and true outcomes.

Introduction to ECE

ECE: Usage and Considerations

- **Usage:**
  - ECE provides a global measure of calibration across the entire range of predicted probabilities.
  - Visualization through a reliability diagram aids in understanding calibration performance.
- **Considerations:**
  - ECE is sensitive to bin sizes; proper binning is crucial for meaningful results.
  - Lower ECE values indicate better-calibrated models with more accurate confidence scores.

Aleatoric Uncertainty Evaluation (Classification)

Quantifying Aleatoric Uncertainty (Classification)

- **Negative Log Likelihood (NLL):** Measures the likelihood of the true class under the predicted probability distribution.
- **Expected Calibration Error (ECE):** Measures the calibration of predicted probabilities against true outcomes.
- **Accuracy:** Essential for assessing the correctness of predictions.

Epistemic Uncertainty Evaluation (Classification)

Quantifying Epistemic Uncertainty (OOD Detection)

- Quantifying Epistemic Uncertainty is hard so often we consider just Out-of-Distribution (OOD) Detection.
- **Out-of-Distribution (OOD) Detection:** Evaluates the model's ability to detect samples outside the training distribution.
- **Transform to 2-Class Classification:** Detecting ID vs. OOD samples, based on DNN confidence scores.

Epistemic Uncertainty Evaluation (Classification)

Quantifying Epistemic Uncertainty (metrics)

We use metrics for binary classification assessment:

- **AUROC** (Area Under the Receiver Operating Characteristic Curve): Measures the trade-off between true positive rate (sensitivity) and false positive rate (1-specificity) across different probability thresholds. A higher value (closer to 1) indicates better performance.
- **AUPR** (Area Under the Precision-Recall Curve): Focuses on the precision-recall trade-off, emphasizing positive class prediction performance, especially in imbalanced datasets. A higher value (closer to 1) indicates better performance.
- **FPR95** (False Positive Rate at 95% True Positive Rate): Evaluates the model's performance at a high sensitivity level (95% true positive rate). A lower value (closer to 0) indicates better performance.

Key Takeaways

Summary of Insights

- **Understanding Sources:** We explored various sources impacting DNNs, acknowledging the inherent uncertainties.
- **Distinguishing Types:** The distinction between aleatoric and epistemic uncertainty provided clarity on different uncertainty manifestations.
- **Quantification Techniques:** We delved into diverse methods for quantifying uncertainty in DNNs.
- **Evaluation Approaches:** Different techniques for evaluating the effectiveness of uncertainty quantification were discussed.

Practical session

Link to the third practical session: https://drive.google.com/file/d/1lmjpNy1UAwLzdAZivD91m-dQW9KBlLSa

Short link: https://tinyurl.com/HelmHUQ1

Maximum Likelihood Estimation for classification

The Goal of DNN is to find $P(y|\boldsymbol{x}, \boldsymbol{\omega})$, most of the classic approaches find $\boldsymbol{\omega}$ that maximizes the likelihood.

$$\boldsymbol{\omega} = \arg\max_{\boldsymbol{\omega}} \log P(\mathcal{D}|\boldsymbol{\omega}) \tag{4}$$

$$\boldsymbol{\omega} = \arg\max_{\boldsymbol{\omega}} \sum_{i=1}^{N} \log P(y_i|\boldsymbol{x}_i, \boldsymbol{\omega}) \tag{5}$$

$$\boldsymbol{\omega} = \arg\max_{\boldsymbol{\omega}} 1/N \sum_{i=1}^{N} \log P(y_i|\boldsymbol{x}_i, \boldsymbol{\omega}) \tag{6}$$

$$\boldsymbol{\omega} = \arg\max_{\boldsymbol{\omega}} \mathbb{E}_{(\boldsymbol{x},y)\sim P(\mathcal{D})} \log P(y|\boldsymbol{x}, \boldsymbol{\omega}) \tag{7}$$

$$\boldsymbol{\omega} = \arg\min_{\boldsymbol{\omega}} H\left[P(\mathcal{D}), P(y|\boldsymbol{x}, \boldsymbol{\omega})\right] \tag{8}$$

With H, the cross entropy.

Bayesian approach and DNN

The Goal of DNN is to find $P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\omega})$. In the classic Bayesian approach, we find $\boldsymbol{\omega}$ such that we have the maximum a posteriori (MAP).

$$\boldsymbol{\omega} = \arg\max_{\boldsymbol{\omega}} \log P(\boldsymbol{\omega}|\mathcal{D}) \tag{9}$$

$$\boldsymbol{\omega} = \arg\max_{\boldsymbol{\omega}} \log P(\mathcal{D}|\boldsymbol{\omega}) + \log P(\boldsymbol{\omega}) \tag{10}$$

This leads to L2 regularization.

Bayesian Deep Neural Networks [1]

Bayesian DNNs are based on marginalization rather than MAP optim.:

$$P(y|\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{\omega}\sim P(\boldsymbol{\omega}|\mathcal{D})}\left[P(y|\boldsymbol{x},\boldsymbol{\omega})\right] \tag{11}$$

$$P(y|\boldsymbol{x}) = \int P(y|\boldsymbol{x},\boldsymbol{\omega})P(\boldsymbol{\omega}|\mathcal{D})d\boldsymbol{\omega} \tag{12}$$

In practice:

$$P(y|\boldsymbol{x}) \simeq \frac{1}{N_{\text{model}}}\sum_i P(Y|X,\boldsymbol{\omega}_i), \text{ with } \boldsymbol{\omega}_i \sim P(\boldsymbol{\omega}|\mathcal{D}) \tag{13}$$

$\Rightarrow$ Different methods to estimate $P(\boldsymbol{\omega}|\mathcal{D})$.

Posterior "Landscape" and Ensembles

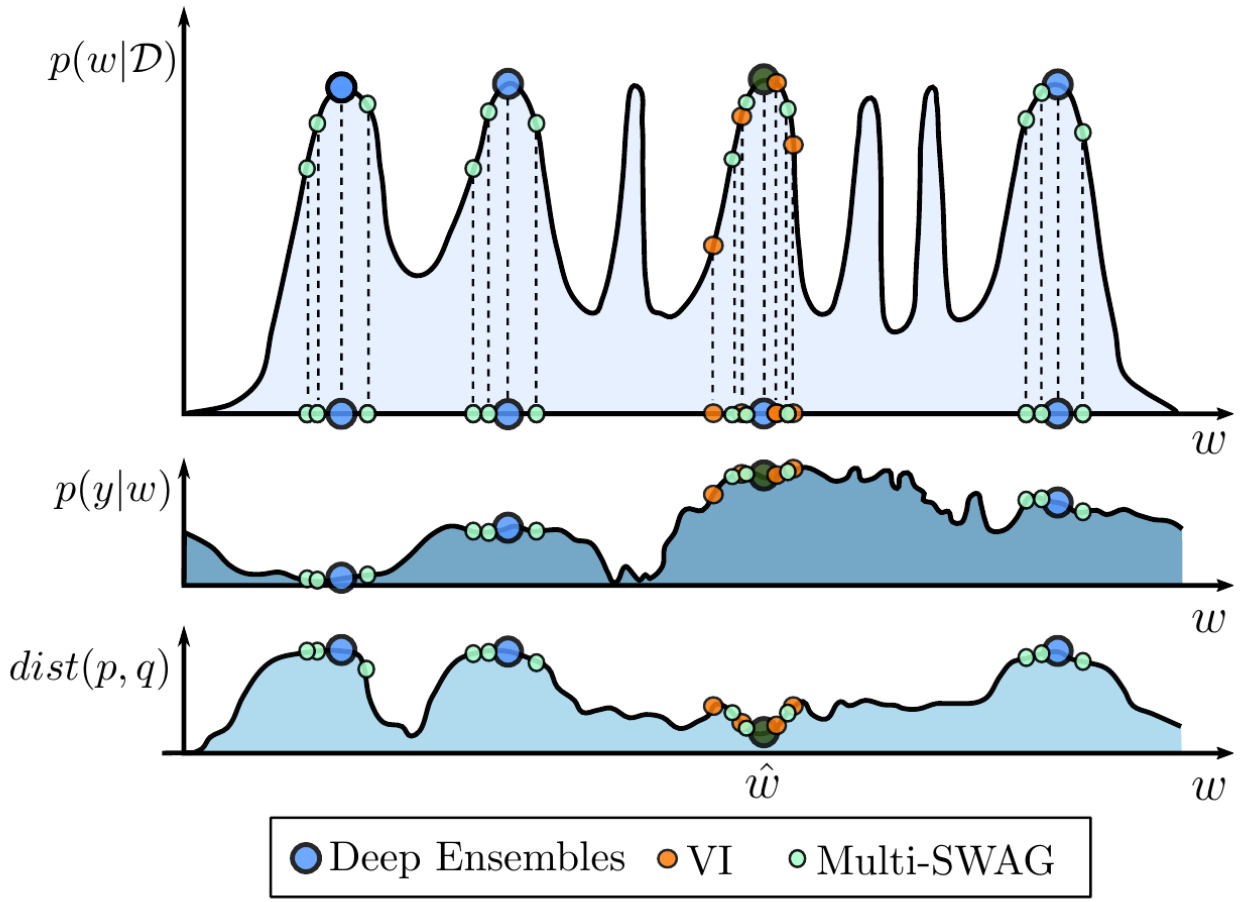


Figure: Top: $P(\omega|\mathcal{D})$, with representations from VI (orange), deep ensembles (blue), multiBNN (red). Middle $P(y|x,\omega)$ (from Wilson & Izmailov [15])

How to estimate the Posterior of BNN?

Markov Chain Monte Carlo (MCMC)

- MCMC methods sample from the posterior distribution by constructing a Markov chain that converges to the desired distribution.
- Popular MCMC algorithms for Bayesian neural networks include Metropolis-Hastings [16], Langevin Dynamics [15], and Hamiltonian Monte Carlo [14].
- While MCMC provides accurate posterior estimates, it can be computationally expensive for large-scale networks.

How to estimate the Posterior of BNN?

Dropout Variational Inference

- Dropout is a regularization technique commonly used in neural networks.
- Dropout Variational Inference (DVI) extends dropout to approximate Bayesian inference.
- It interprets dropout as sampling from a variational distribution over the weights, enabling uncertainty estimation.

Variational Inference (VI)

- VI [1] approximates the posterior distribution with a simpler distribution, typically from a parametric family.
- It formulates an optimization problem to minimize the divergence between the true posterior and the approximating distribution.
- While VI provides an interpretable way to estimate the posterior, it can be computationally expensive, unstable, and collapse to a single DNN without variance.

Variational inference

Variational inference approximates the posterior $\mathcal{P}(\boldsymbol{\omega}|\mathcal{D}_l)$ with a family of distributions $q_\lambda(\boldsymbol{\omega}/\mathcal{D}_l)$. The variational parameter λ indexes the family of distributions. For example, if q were Gaussian, it would be the mean and variance of the latent variables for each datapoint $\lambda_{x_i} = (\mu_{x_i}, \sigma^2_{x_i}))$.
Question : How can we know how well our variational posterior $q_\lambda(w/\mathcal{D}_l)$ approximates the true posterior $\mathcal{P}(\boldsymbol{\omega}|\mathcal{D}_l)$?

Variational inference

Question : How can we know how well our variational posterior $q_\lambda(\omega/\mathcal{D}_l)$ approximates the true posterior $\mathcal{P}(\omega|\mathcal{D}_l)$?
We can use the Kullback-Leibler divergence, which measures the information lost when using q to approximate $\mathcal{P}$:

$$\begin{aligned}\mathbb{KL}(q_\lambda(\omega/\mathcal{D}_l) \,||\, \mathcal{P}(\omega|\mathcal{D}_l)) &= \int_\omega \left(q_\lambda(\omega/\mathcal{D}_l) \log(\frac{q_\lambda(\omega/\mathcal{D}_l)}{\mathcal{P}(\omega|\mathcal{D}_l)}) \right) d\omega \\ &= \int_\omega \left(q_\lambda(\omega/\mathcal{D}_l) \log(\frac{q_\lambda(\omega/\mathcal{D}_l)}{\mathcal{P}(\mathcal{D}_l)\mathcal{P}(\mathcal{D}_l,\omega)}) \right) d\omega \\ &= \mathrm{E}_q[\log q_\lambda(\omega/\mathcal{D}_l)] - \mathrm{E}_q[\log \mathcal{P}(\omega,\mathcal{D}_l)] + \log \mathcal{P}(\mathcal{D}_l)\end{aligned}$$

Our goal is to find the variational parameters λ that minimize this divergence. The optimal approximate posterior is thus

Variational inference

The optimal approximate posterior is thus

$$q_\lambda^*(\boldsymbol{\omega}/\mathcal{D}_l) = \arg\min_\lambda \mathbb{KL}(q_\lambda(\boldsymbol{\omega}/\mathcal{D}_l) \,||\, \mathcal{P}(\boldsymbol{\omega}|\mathcal{D}_l)).$$

This is impossible to compute directly due to $\mathcal{P}(\mathcal{D}_l)$ that appears in the divergence. So, we consider the following function:

$$\begin{aligned} ELBO(\lambda) &= \mathsf{E}_q[\log \mathcal{P}(\boldsymbol{\omega}, \mathcal{D}_l)] - \mathsf{E}_q[\log q_\lambda(\boldsymbol{\omega}/\mathcal{D}_l)] \\ &= -\int_{\boldsymbol{\omega}} \left(q_\lambda(\boldsymbol{\omega}/\mathcal{D}_l) \log(\frac{q_\lambda(\boldsymbol{\omega}/\mathcal{D}_l)}{\mathcal{P}(\boldsymbol{\omega})\mathcal{P}(\mathcal{D}_l|\boldsymbol{\omega})}) \right) d\boldsymbol{\omega} \\ &= \mathsf{E}_q[\log \mathcal{P}(\mathcal{D}_l|\boldsymbol{\omega})] - \mathbb{KL}(q_\lambda(\boldsymbol{\omega}/\mathcal{D}_l) \,||\, \mathcal{P}(\boldsymbol{\omega})) \end{aligned}$$

Note that $\mathbb{KL}(q_\lambda(\boldsymbol{\omega}/\mathcal{D}_l) \,||\, \mathcal{P}(\boldsymbol{\omega}|\mathcal{D}_l)) = \log \mathcal{P}(\mathcal{D}_l) - ELBO(\lambda)$.

Variational inference: Reparametrization trick

theorem: Let ϵ be a random variable having a probability density given by $q(\epsilon)$ and let $\boldsymbol{\omega} = t(\lambda, \epsilon)$. Suppose that $q_\lambda(\boldsymbol{\omega}/\mathcal{D}_I)$, is such that $q(\epsilon)d\epsilon = q_\lambda(\boldsymbol{\omega}/\mathcal{D}_I)d\boldsymbol{\omega}$. Then for a function f with derivatives in $\boldsymbol{\omega}$:

$$\frac{\partial}{\partial \lambda} \mathsf{E}_{q_\lambda(\boldsymbol{\omega}/\mathcal{D}_I)} f(\boldsymbol{\omega}, \lambda) = \mathsf{E}_{q(\epsilon)} \left[\frac{\partial f(\boldsymbol{\omega}, \lambda)}{\partial \boldsymbol{\omega}} \frac{\partial \boldsymbol{\omega}}{\partial \lambda} + \frac{\partial f(\boldsymbol{\omega}, \lambda)}{\partial \lambda} \right].$$

Variational inference [1]

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

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

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

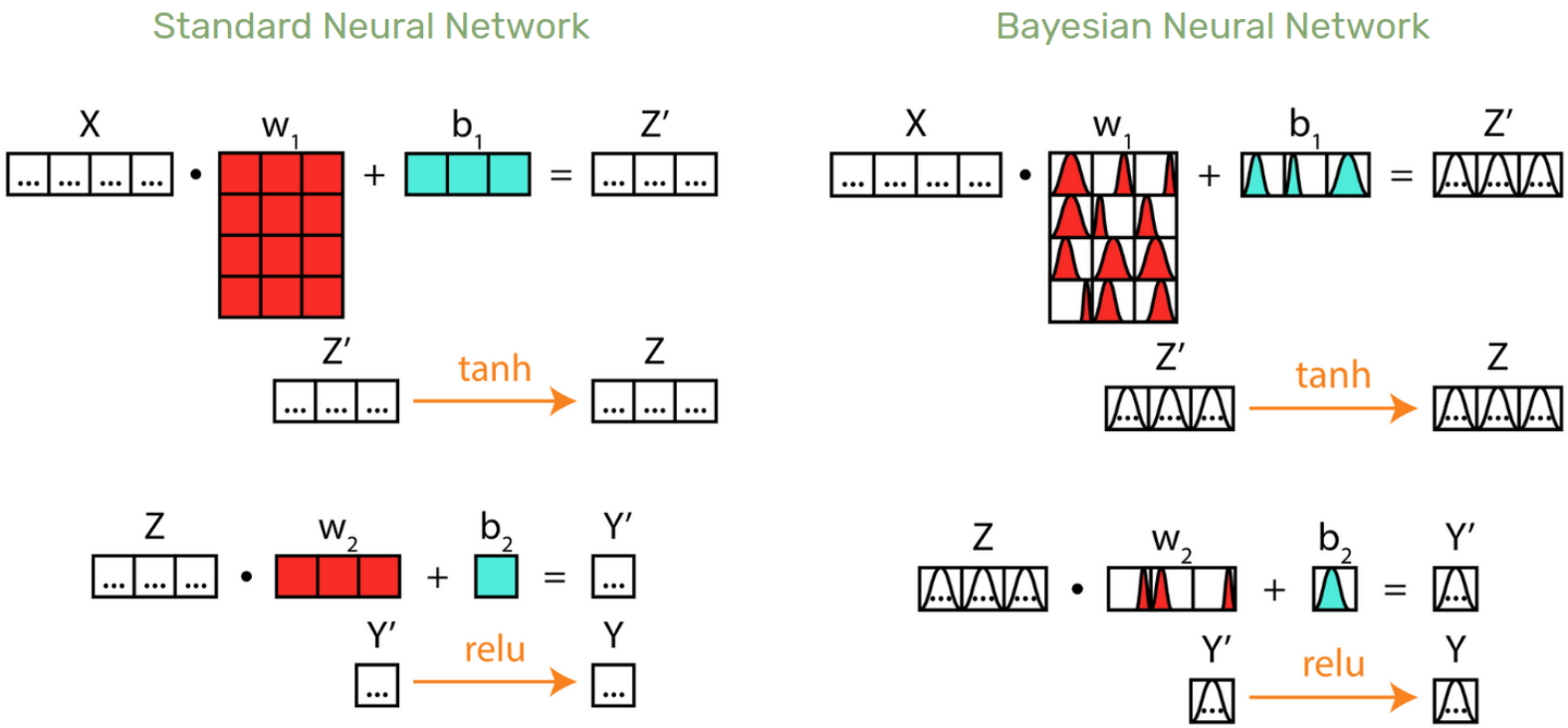
$$\Delta_\rho = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\epsilon}{1 + \exp(-\rho)} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \rho}. \tag{4}$$

7. Update the variational parameters:

$$\mu \leftarrow \mu - \alpha \Delta_\mu \tag{5}$$

$$\rho \leftarrow \rho - \alpha \Delta_\rho. \tag{6}$$

Weight Uncertainty in Neural Networks [1][1]



[1] Image credit: Eric Ma

TRADI [10]

- $\boldsymbol{\omega}(0)$ is the initial set of weights $\{\omega_k(0)\}_{k=1}^K$ following $\mathcal{N}(0, \sigma_k^2)$, where σ_k^2 are fixed as in [2].
- $\mathcal{L}(\boldsymbol{\omega}(t), y_i)$ is the loss function used to measure the dissimilarity between the output $g_{\boldsymbol{\omega}(t)}(x_i)$ of the DNN and the expected output y_i. One can use different loss functions.
- Weights on different layers are assumed to be independent of one another at all times.
- Each weight $\omega_k(t)$, $k = 1, \ldots, K$, follows a non-stationary Normal distribution (e.g. $W_k(t) \sim \mathcal{N}(\mu_k(t), \sigma_k^2(t))$) whose two parameters are tracked.

TRADI [10]

We had following state and measurement equations for the mean $\mu_k(t)$:

$$\begin{cases} \mu_k(t) = \mu_k(t-1) - \eta \nabla \mathcal{L}_{\omega_k(t)} + \varepsilon_\mu \\ \omega_k(t) = \mu_k(t) + \tilde{\varepsilon}_\mu \end{cases} \tag{14}$$

with ε_μ being the state noise, and $\tilde{\varepsilon}_\mu$ being the observation noise, as realizations of $\mathcal{N}(0, \sigma_\mu^2)$ and $\mathcal{N}(0, \tilde{\sigma}_\mu^2)$ respectively.

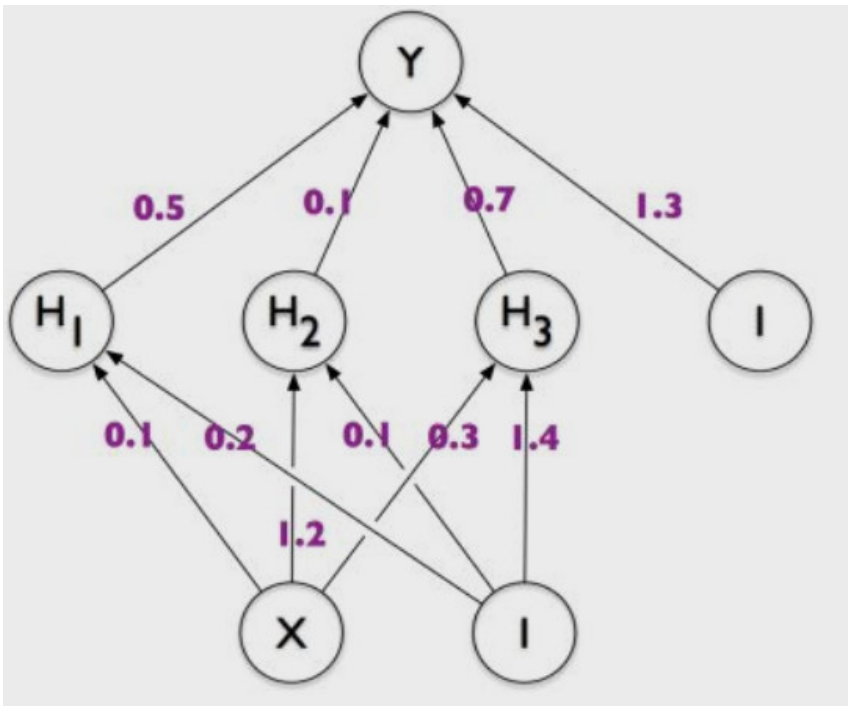
TRADI [10]

The state and measurement equations for the variance σ_k are given by:

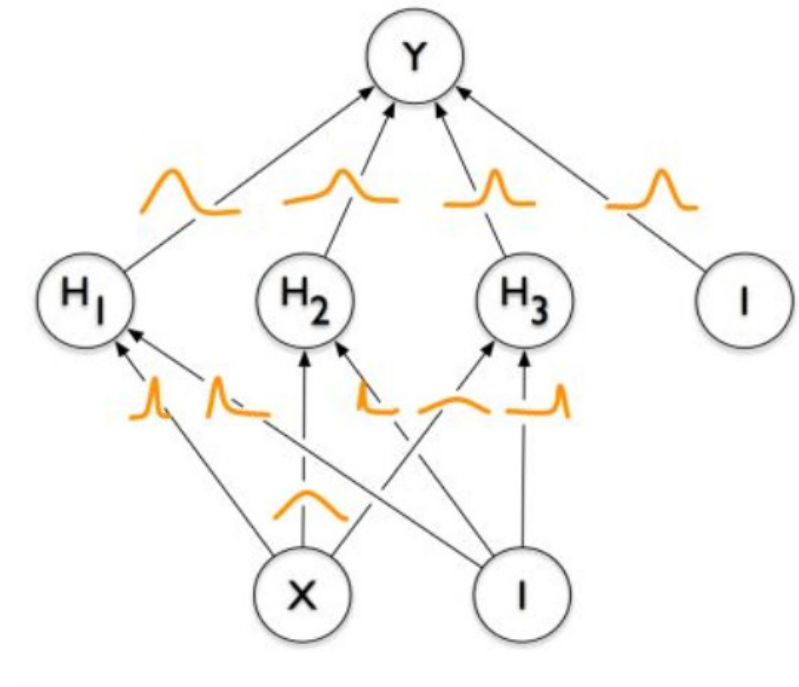
$$\begin{cases} \sigma_k^2(t) = \sigma_k^2(t-1) + \left(\eta \nabla \mathcal{L}_{\omega_k(t)}\right)^2 - \eta^2 \mu_k(t)^2 + \varepsilon_\sigma \\ z_k(t) = \sigma_k^2(t) - \mu_k(t)^2 + \tilde{\varepsilon}_\sigma \\ \text{with } z_k(t) = \omega_k(t)^2 \end{cases} \tag{15}$$

with ε_σ being the state noise, and $\tilde{\varepsilon}_\sigma$ being the observation noise, as realizations of $\mathcal{N}(0, \sigma_\sigma^2)$ and $\mathcal{N}(0, \tilde{\sigma}_\sigma^2)$, respectively.

TRADI [10]



(Normal DNN)



(Bayesian DNN)

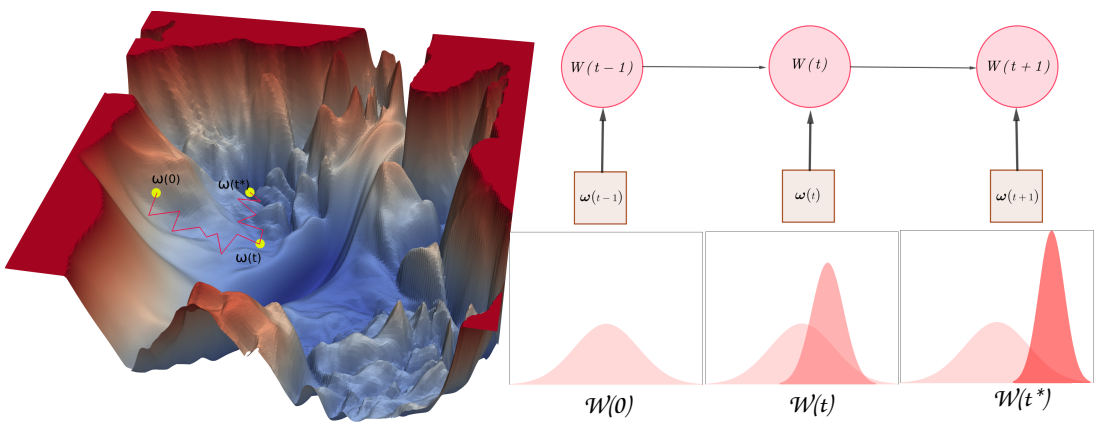
TRADI [10]

We sample new realizations of $W(t*)$ using the following formula:

$$\tilde{\omega}(t^*) = \mu(t^*) + \Sigma^{1/2}(t^*) \times \boldsymbol{m}_1 \text{ with } \Sigma \text{ the covariance matrix.} \quad (16)$$

$\boldsymbol{m}_1$ is a realization of the multivariate Gaussian $\mathcal{N}(0_K, \boldsymbol{I}_K)$. Then we take the expectation over this distribution :

$$\mathcal{P}(y^*|x^*) = \frac{1}{N_{\text{model}}} \sum_{j=1}^{N_{\text{model}}} \mathcal{P}(y^*|\tilde{\omega}^j(t^*), x^*) \quad (17)$$



How to estimate the Posterior of BNN?

Classical VI-BNN

Using the "reparametrization trick", a layer j of an MLP can be written:

$$\begin{aligned} \boldsymbol{u}_j &= \text{norm}\left(\left[W_\mu^{(j)} + \boldsymbol{\epsilon}_j W_\sigma^{(j)}\right] \boldsymbol{h}_{j-1}, \beta_j, \gamma_j\right), \text{ and} \\ \boldsymbol{a}_j &= a(\boldsymbol{u}_j), \end{aligned} \tag{18}$$

where the matrices $W_\mu^{(j)}$ and $W_\sigma^{(j)}$ denote the mean and standard deviation of the posterior distribution of layer j, $\boldsymbol{\epsilon}_j \sim \mathcal{N}(0, \mathbb{1})$ and the operator $\text{norm}(\cdot, \beta_j, \gamma_j)$, of trainable parameters β_j and γ_j, can refer to any batch, layer, or instance normalization.

How to turn a DNN into a BNN?

ABNN [17]

Our objective differs from VI-BNN, which requires training the posterior distribution parameters from scratch. Instead, our approach entails leveraging and converting an existing DNN into a BNN.

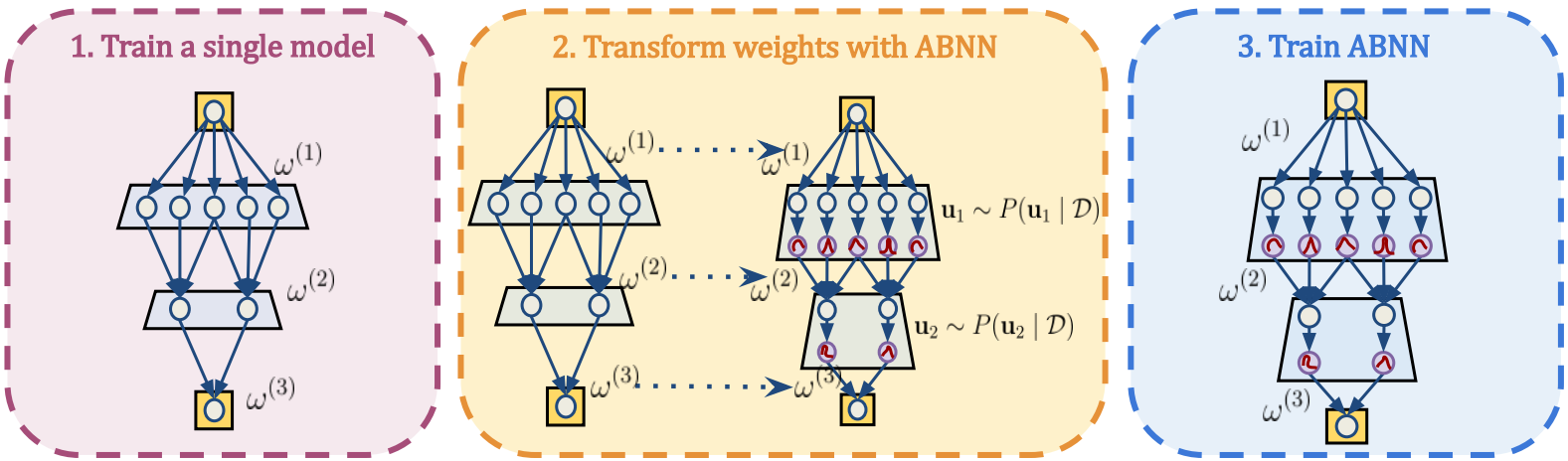


Figure: **Illustration of the training process for the ABNN.** The procedure begins with training a single DNN $\boldsymbol{\omega}_{\mathrm{MAP}}$, followed by architectural adjustments to transform it into an ABNN. The final step involves fine-tuning the ABNN model.

How to turn a DNN into a BNN?

ABNN [17]

Formally, our BNN relies on a new layer **BNL**$(\cdot)$:

$$\begin{aligned} \boldsymbol{u}_j &= \mathbf{BNL}\left(W^{(j)}\boldsymbol{h}_{j-1}\right), \text{ and} \boldsymbol{a}_j = a(\boldsymbol{u}_j), \text{ with} \\ \mathbf{BNL}(\boldsymbol{h}_j) &= \frac{\boldsymbol{h}_j - \hat{\mu}_j}{\hat{\sigma}_j} \times \gamma_j(1+\boldsymbol{\epsilon}_j) + \beta_j. \end{aligned} \tag{19}$$

This can be seen as adding a Gaussian dropout on the normalization layer and finetuning the DNN. We propose to train multiple of these ABNNs to have multiple modes of the posterior.

ABNN during evaluation

During evalution, for each sample from ABNN $\boldsymbol{\omega}_m$, we augment the number of samples by independently sampling multiple $\boldsymbol{\epsilon}_j \sim \mathcal{N}(0, \mathbb{1})$.

$$P(y \mid \boldsymbol{x}, \mathcal{D}) \approx \frac{1}{ML} \sum_{l=1}^{L} \sum_{m=1}^{M} P(y \mid \boldsymbol{x}, \boldsymbol{\omega}_m, \boldsymbol{\epsilon}_l). \tag{20}$$

Classification Results

	Method	CIFAR-10 Acc ↑	CIFAR-10 NLL ↓	CIFAR-10 AUPR ↑	CIFAR-10 AUC ↑	CIFAR-10 FPR95 ↓	CIFAR-100 Acc ↑	CIFAR-100 NLL ↓	CIFAR-100 AUPR ↑	CIFAR-100 AUC ↑	CIFAR-100 FPR95 ↓	Time (h) ↓
ResNet-50	Single Model	95.1	0.211	95.2	91.9	23.6	78.3	0.905	87.4	77.9	57.6	**1.7**
	BatchEnsemble	93.9	0.255	94.7	91.3	20.1	66.6	1.788	85.2	74.6	60.6	17.2
	LPBNN	94.3	0.231	92.7	86.7	54.9	78.5	1.02	88.2	77.8	73.5	17.2
	MCDropout	94.4	0.190	93.1	86.9	43.8	76.9	0.858	87.8	77.1	64.1	**1.7**
	MCBN	95.0	0.168	95.7	92.6	20.1	78.4	0.83	86.8	77.5	57.7	**1.7**
	Deep Ensembles	**96.0**	**0.136**	**97.0**	**94.7**	**80.9**	**0.713**	**2.6**	89.2	80.8	52.5	6.8
	Laplace	95.3	0.160	96.0	93.3	78.2	0.99	14.2	89.2	81.0	51.8	**1.7**
	ABNN	95.0	0.160	96.5	93.9	17.5	77.8	0.828	**90.0**	**82.0**	**51.3**	2.0
WideResNet-28×10	Single Model	95.4	0.200	96.1	93.2	20.4	80.3	0.963	81.0	64.2	80.1	**4.2**
	BatchEnsemble	95.6	0.206	95.5	92.5	22.1	82.3	0.835	88.1	78.2	69.8	25.6
	LPBNN	95.1	0.249	95.4	91.2	29.5	79.7	0.831	79.0	70.1	71.4	23.3
	MCDropout	95.7	0.138	96.2	93.5	12.8	79.2	0.758	**89.4**	**80.1**	58.6	**4.2**
	MCBN	95.5	**0.133**	96.5	94.2	14.6	80.4	0.749	80.4	67.8	63.1	**4.2**
	Deep Ensembles	**95.8**	0.143	**97.8**	**96.0**	**82.5**	0.903	22.9	81.6	67.9	71.3	16.6
	Laplace	95.6	0.151	95.0	90.7	31.9	80.1	0.942	83.4	72.1	59.9	**4.2**
	ABNN	94.5	0.171	0.7	96.8	94.6	80.0	**0.734**	86.7	75.7	59.4	5.0

→ ABNN improves uncertainty quantification with small computational overhead

→ Most of the gains are linked to improved epistemic uncertainty (as measured by OOD detection)

Semantic segmentation Results

	Method	mIoU ↑	ECE ↓	AUPR ↑	AUC ↑	FPR95 ↓
StreetHazards	Single Model	53.9	6.5	6.9	86.6	35.7
	TRADI	52.5	6.3	6.9	87.4	38.3
	Deep Ensembles	55.6	5.3	**8.3**	87.9	**30.3**
	BatchEnsemble	**56.2**	6.1	7.6	88.2	32.9
	LP-BNN	54.5	**5.2**	7.2	88.3	32.6
	ABNN	53.8	6.1	7.9	**88.4**	32.0
BDD-Anomaly	Single Model	47.6	17.7	4.5	85.2	28.8
	TRADI	44.3	16.6	4.5	84.8	36.9
	Deep Ensembles	**51.1**	14.2	5.2	84.8	**28.6**
	BatchEnsemble	48.1	16.9	4.5	84.3	30.2
	LP-BNN	49.0	17.2	4.5	85.3	29.5
	ABNN	48.8	**14.0**	**6.0**	**85.7**	29.0
MUAD	Single Model	57.3	6.1	26.0	86.2	39.4
	MC-Dropout	55.6	6.5	22.3	84.4	45.8
	Deep Ensembles	58.3	5.9	**28.0**	87.1	37.6
	BatchEnsemble	57.1	6.0	25.7	86.9	38.8
	ABNN	**62.0**	**5.6**	24.4	**91.6**	**21.7**

→ ABNN also performs well in the segmentation setting

Conclusions

Exploring Further

- **Contribute to Torch Uncertainty:** If you want to advance the field, consider contributing to TorchUncertainty.

  `https://github.com/ENSTA-U2IS-AI/torch-uncertainty`

- **Explore Our Resources:** Check out our curated list of resources on Uncertainty, available at our "awesome of uncertainty" repository.

  `https://github.com/ENSTA-U2IS-AI/awesome-uncertainty-deeplearning`

Practical session

Link to the fourth practical session: `https://drive.google.com/file/d/1vUVsO5gDP6M7NnpfRr0LjxG7P2d-76A5`

Short link: `https://tinyurl.com/HelmHUQ2`

Bibliography:

1 Blundell, Charles, et al. "Weight uncertainty in neural networks." arXiv preprint arXiv:1505.05424 (2015)

2 A.G. Wilson, P. Izmailov. Bayesian Deep Learning and a Probabilistic Perspective of Generalization. Advances in Neural Information Processing Systems, 2020.

3 Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles." Advances in neural information processing systems. 2017.

4 Gal, Yarin, and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning." international conference on machine learning. 2016.

Bibliography:

5 A.G. Wilson, P. Izmailov. Bayesian Deep Learning and a Probabilistic Perspective of Generalization. Advances in Neural Information Processing Systems, 2020.

6 Wen, Yeming, Dustin Tran, and Jimmy Ba. "Batchensemble: an alternative approach to efficient ensemble and lifelong learning." arXiv preprint arXiv:2002.06715 (2020).

7 Franchi, G., Yu, X., Bursuc, A., Tena, A., Kazmierczak, R., Dubuisson, S., ... & Filliat, D. (2022). MUAD: Multiple Uncertainties for Autonomous Driving, a benchmark for multiple uncertainty types and tasks. arXiv preprint arXiv:2203.01437.

8 Gawlikowski, J., Tassi, C.R.N., Ali, M., Lee, J., Humt, M., Feng, J., Kruspe, A., Triebel, R., Jung, P., Roscher, R. and Shahzad, M., 2023. A survey of uncertainty in deep neural networks. Artificial Intelligence Review, 56(Suppl 1), pp.1513-1589.

Bibliography:

9 Zhang, R., Li, C., Zhang, J., Chen, C., & Wilson, A. G. (2019). Cyclical stochastic gradient MCMC for Bayesian deep learning. arXiv preprint arXiv:1902.03932.

10 Franchi, G., Bursuc, A., Aldea, E., Dubuisson, S., & Bloch, I. (2020). TRADI: Tracking deep neural network weight distributions. In ECCV 2020.

11 Fort, Stanislav, Huiyi Hu, and Balaji Lakshminarayanan. "Deep Ensembles: A Loss Landscape Perspective." arXiv preprint arXiv:1912.02757 (2019).

12 Havasi, M., Jenatton, R., Fort, S., Liu, J. Z., Snoek, J., Lakshminarayanan, B., ... & Tran, D. (2020). Training independent subnetworks for robust prediction. arXiv preprint arXiv:2010.06610.

13 Laurent, O., Lafage, A., Tartaglione, E., Daniel, G., Martinez, J. M., Bursuc, A., and Franchi, G. (2022). Packed-Ensembles for Efficient Uncertainty Estimation. In ICLR 2023.

Bibliography:

14 Levy, Daniel, Matthew D. Hoffman, and Jascha Sohl-Dickstein. "Generalizing hamiltonian monte carlo with neural networks." arXiv preprint arXiv:1711.09268 (2017).

15 Welling, Max, and Yee W. Teh. "Bayesian learning via stochastic gradient Langevin dynamics." Proceedings of the 28th international conference on machine learning (ICML-11). 2011.

16 Angelopoulos, Nicos, and James Cussens. "Bayesian learning of Bayesian networks with informative priors." Annals of Mathematics and Artificial Intelligence 54 (2008): 53-98.

17 Franchi, G., Laurent, O., LeguÃ©ry, M., Bursuc, A., Pilzer, A., & Yao, A. (2023). Make Me a BNN: A Simple Strategy for Estimating Bayesian Uncertainty from Pre-trained Models. arXiv preprint arXiv:2312.15297.