Encoding the latent posterior of Bayesian neural networks for Uncertainty Quantification

Gianni FRANCHI

U2IS, ENSTA Paris

Presentation 2021

16/12/2021

Plan

What is uncertainty in machine/deep learning

- We make observations using the sensors in the world (e.g. camera)
- Based on the observations, we intend to learn a model that makes decisions
- Given the **same** observations, the decision should be the **same**

However,

- The world **changes**, observations **change**, our sensors **change**, the output should not change!
- We would like to know how confident we can be about the decisions

Why Uncertainty is important?

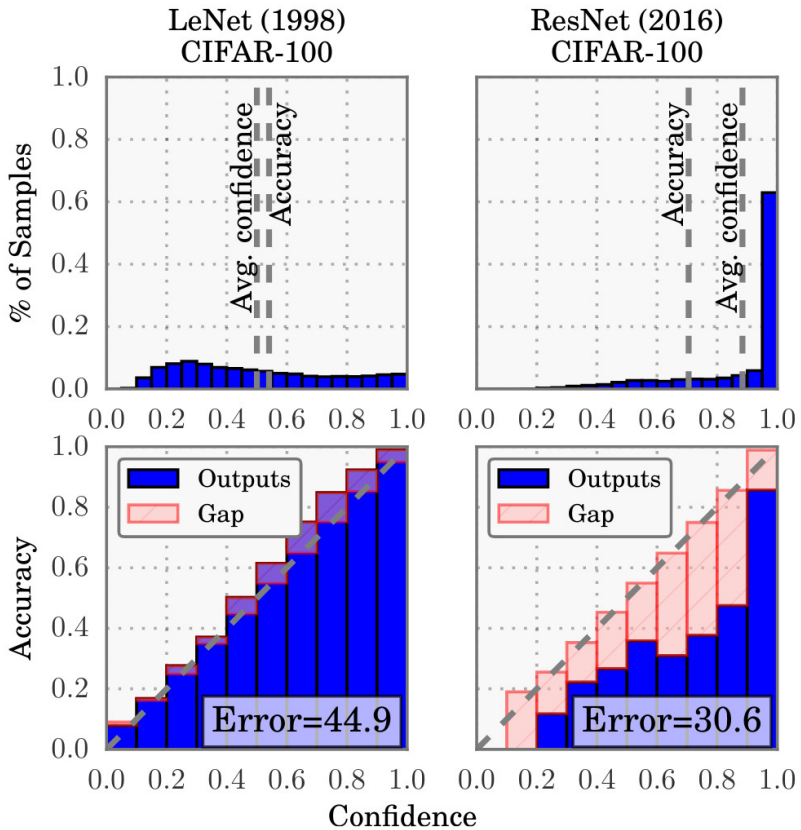
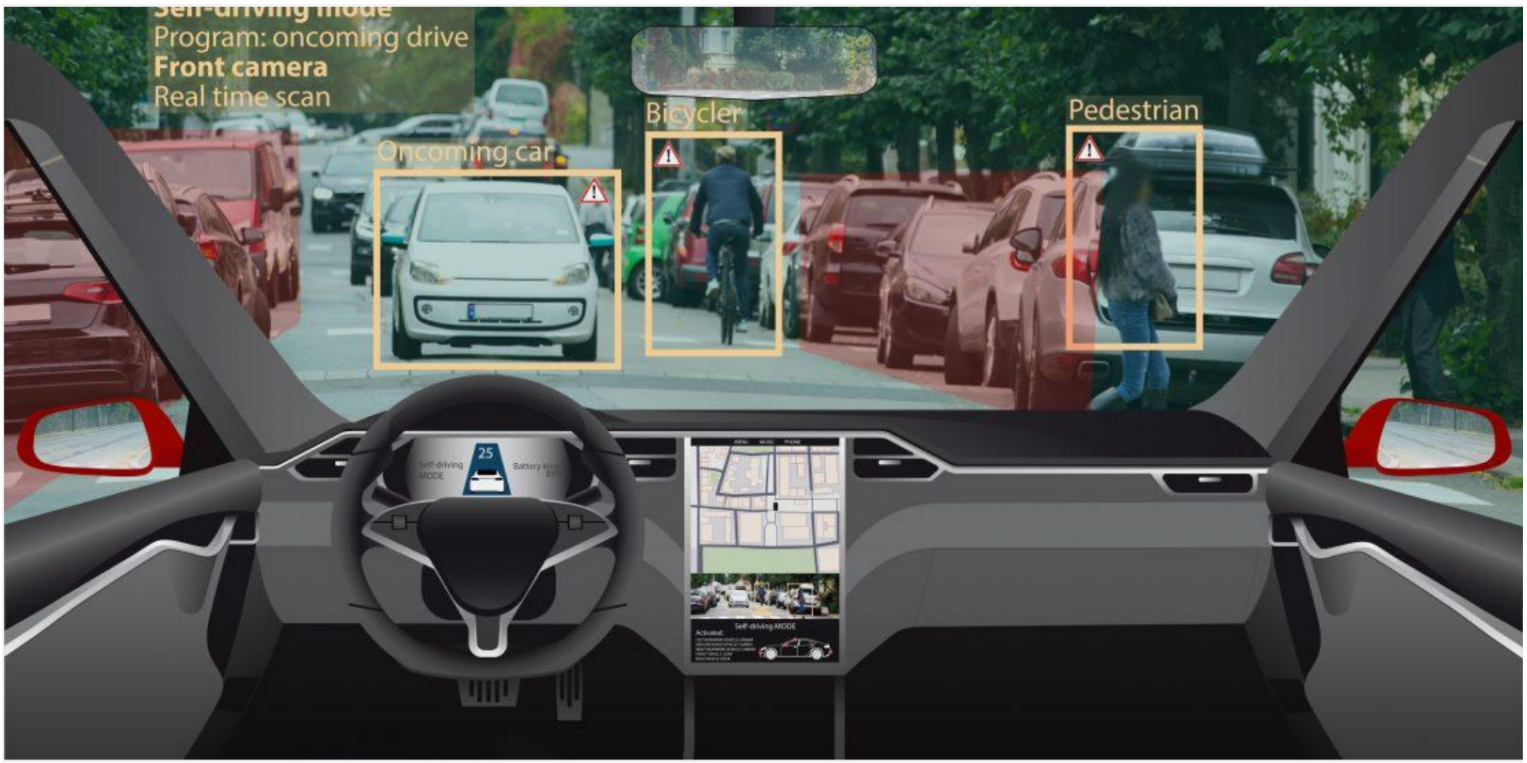


Figure: Confidence histograms (top) and reliability diagrams(bottom) for a 5-layer LeNet (left) and a 110-layer ResNet (right)on CIFAR-100. [1]

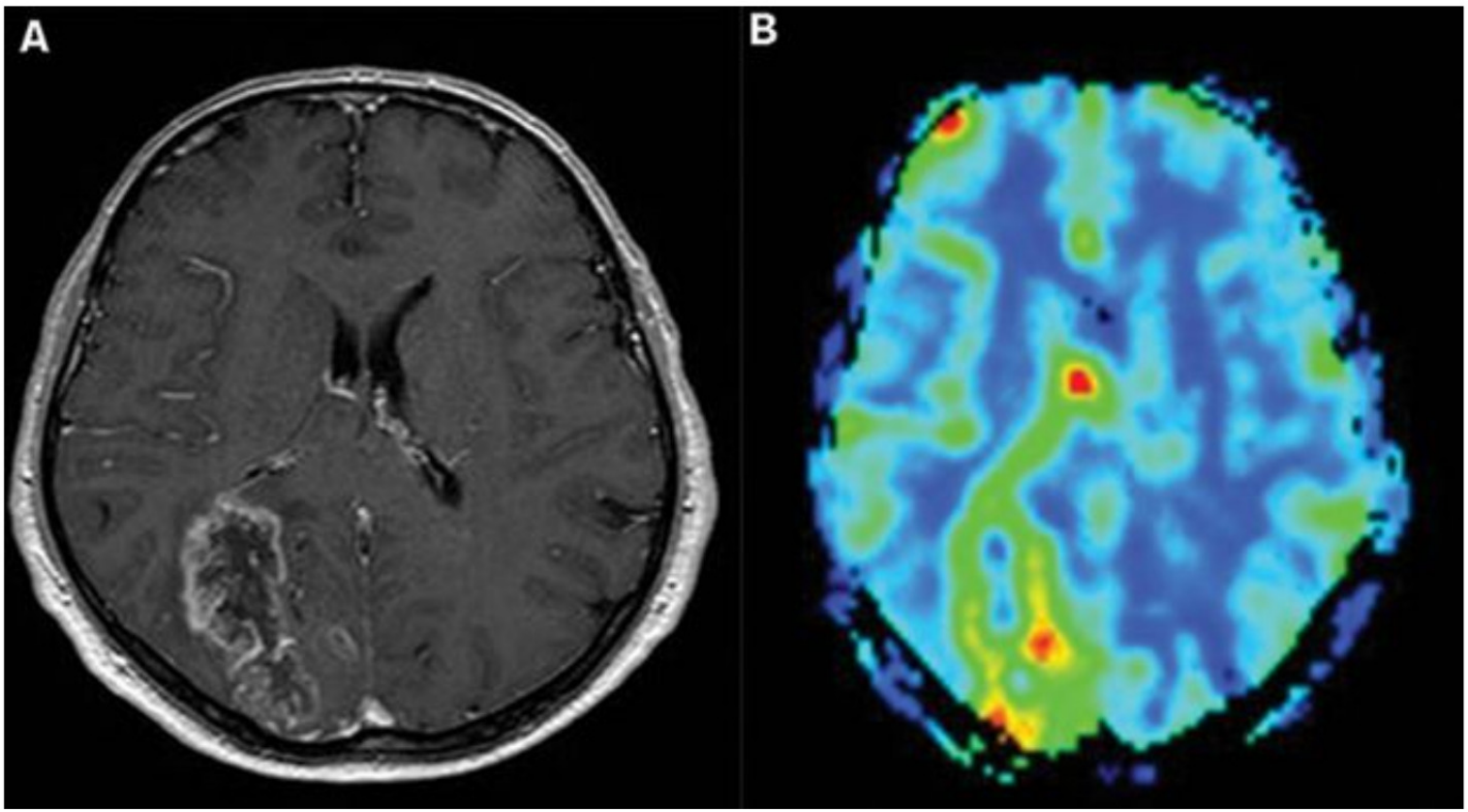
Why Uncertainty is important?

Imagine an autonomous car with a perception system based on Deep learning without Uncertainty:



Why Uncertainty is important?

Imagine a medical diagnostics based on Deep learning without Uncertainty:



Why Uncertainty is important?

We build models for predictions, can we trust them? Are they certain?

Aleatoric uncertainty

Aleatoric uncertainty can further be categorized into homoscedastic and heteroscedastic uncertainties:

- Homoscedastic uncertainty relates to the uncertainty that a particular task might cause. It stays constant for different inputs.
- Heteroscedastic uncertainty depends on the inputs to the model, with some inputs potentially having more noisy outputs than others.

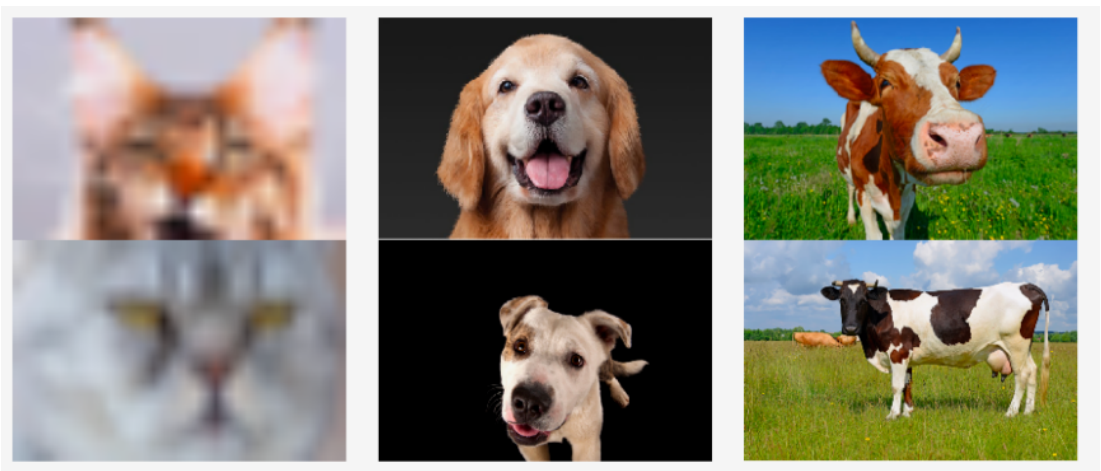
Types of Uncertainty: Case 1[1]

Let us consider a neural network model trained with several pictures of dogs. We ask the model to decide on a dog using a photo of a cat. What would you want the model to do?

[1] Credits: Gille Louppe

Types of Uncertainty: Case 2[2]

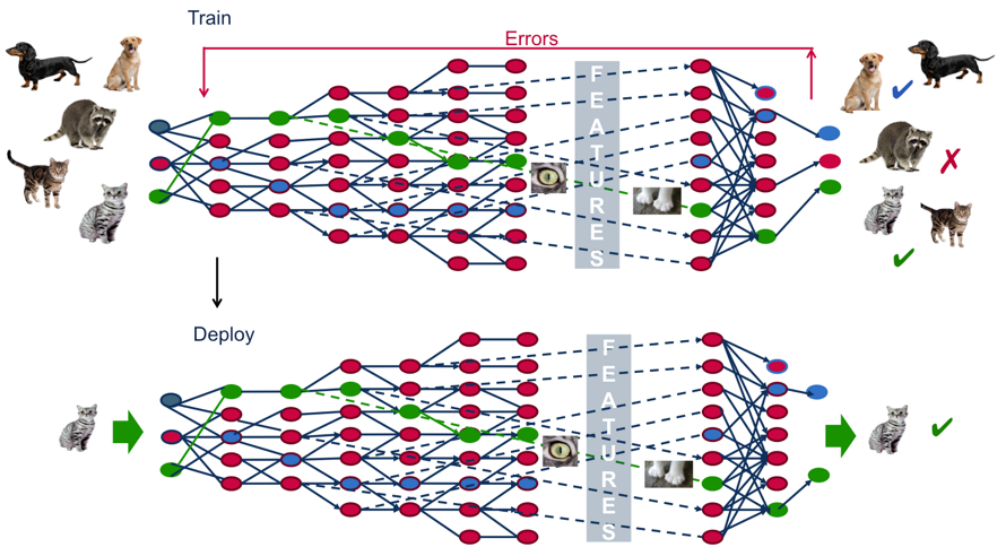
We have three different types of images to classify, cat, dog, and cow, some of which may be noisy due to the limitations of the acquisition instrument.



[2] Credits: Gille Louppe

Deep Learning

Deep learning systems are neural network (or convolutional neural network) models similar to those popular in the '80s and '90s, with algorithmic inovations, software inovations, and larger data sets.



Deep Learning notations

- Training/Testing sets are denoted respectively by $\mathcal{D}_l = (x_i, y_i)_{i=1}^{n_l}$, $\mathcal{D}_\tau = (x_i, y_i)_{i=1}^{n_\tau}$. Without loss of generality we consider the observed samples $\{x_i\}_{i=1}^{n}$ and the corresponding labels $\{y_i\}_{i=1}^{n}$ as vectors. Data in $\mathcal{D}_l$ and $\mathcal{D}_\tau$ are assumed to be i.i.d. distributed according to their respective unknown joint distribution $\mathcal{P}_t$ and $\mathcal{P}_\tau$.
- The Deep Neural Networks (DNN) are functions parameterized by a vector containing the K trainable weights $\boldsymbol{\omega} = \{\omega_k\}_{k=1}^{K}$.
- During training, $\boldsymbol{\omega}$, is iteratively updated for each mini-batch, we denote $W(t)$ the random variable associated with the weights.
- We view a neural network g_ω as a probabilistic model $g_\omega(x) = \mathcal{P}(y|x, \boldsymbol{\omega})$.

Bayesian approach and DNN

The Goal of DNN is to find ω optimal for $\mathcal{P}(y|x,\omega)$, most of the classical approach find ω that maximize the likelihood.

$$\omega = \arg\max_{\omega} \log \mathcal{P}(\mathcal{D}_I|\omega)$$

$$\omega = \arg\max_{\omega} \sum_{i=1}^{n_I} \log \mathcal{P}(y_i|x_i,\omega)$$

$$\omega = \arg\max_{\omega} 1/n_I \sum_{i=1}^{n_I} \log \mathcal{P}(y_i|x_i,\omega)$$

$$\omega \approx \arg\max_{\omega} \mathbb{E}_{(X,Y)\sim\mathcal{P}_t} \log \mathcal{P}(Y|X,\omega)$$

$$\omega \approx \arg\min_{\omega} H\left[\mathcal{P}_t, \mathcal{P}(Y|X,\omega)\right]$$

With H the cross entropy.

Bayesian approach and DNN

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

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

This leads to l2 regularization.

Bayesian DNN

Bayesian DNN is based on marginalization instead of MAP optimization.

$$\mathcal{P}(Y|X) = \mathbb{E}_{\omega \sim \mathcal{P}(\omega|\mathcal{D}_l)}\left(\mathcal{P}(Y|X,\omega)\right)$$
$$\mathcal{P}(Y|X) = \int \mathcal{P}(Y|X,\omega)\mathcal{P}(\omega|\mathcal{D}_l)d\omega$$

In practice:

$$\mathcal{P}(y|x) \simeq (1/N_{\text{model}}).\sum_{j=1}^{N_{\text{model}}} \mathcal{P}(y|x,\omega_j) \text{ with } \omega_j \sim \mathcal{P}(\omega|\mathcal{D}_l)$$

Different techniques to estimate $\mathcal{P}(\omega|\mathcal{D}_l)$.

Variational inference

Variational inference approximates the posterior $\mathcal{P}(\boldsymbol{\omega}|\mathcal{D}_I)$ with a family of distributions $q_\lambda(\boldsymbol{\omega}|\mathcal{D}_I)$ 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 $q_\lambda(w|\mathcal{D}_I)$ approximates the true posterior $\mathcal{P}(\boldsymbol{\omega}|\mathcal{D}_I)$?

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:

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

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 [4]

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

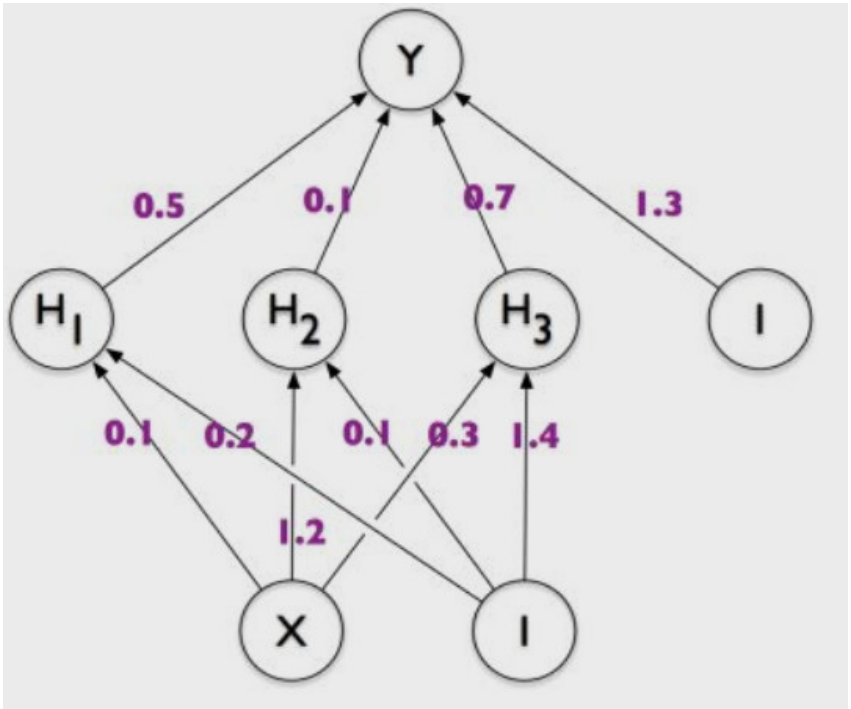
$$\Delta_\rho = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\epsilon}{1 + \exp(-\rho)} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \rho}. \tag{4}$$

7. Update the variational parameters:

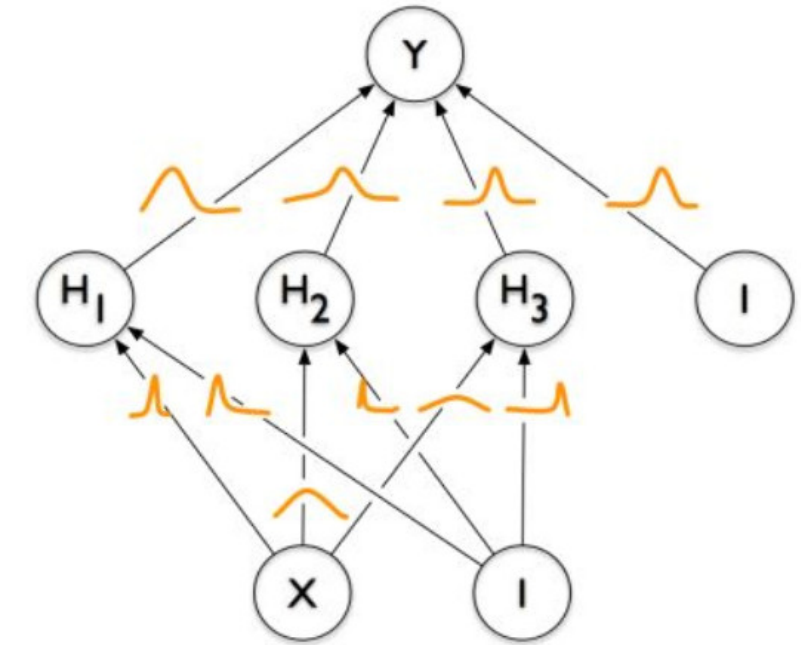
$$\mu \leftarrow \mu - \alpha \Delta_\mu \tag{5}$$

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

Weight Uncertainty in Neural Networks [4]



(Normal DNN)



(Bayesian DNN)

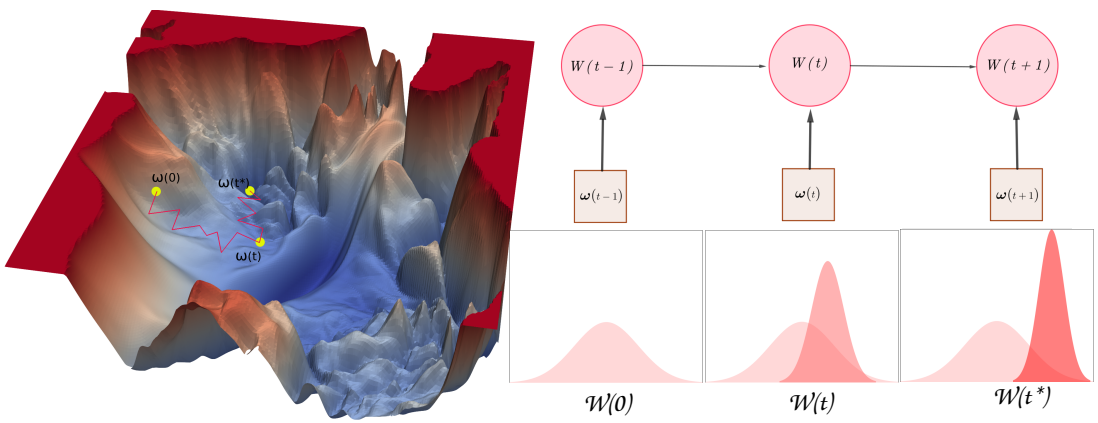
TRADI[9]

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

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

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

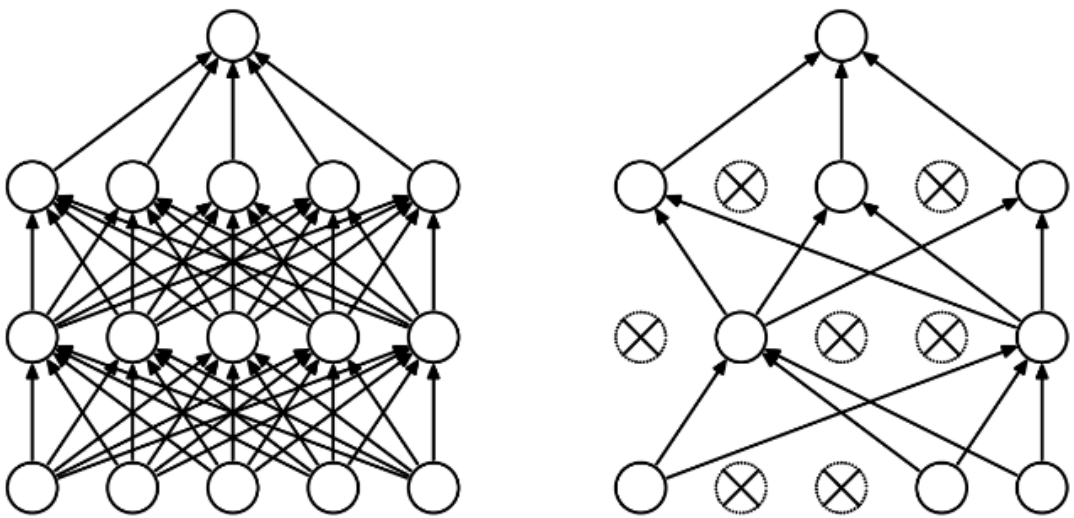
$$\mathcal{P}(y^*|x^*) = \frac{1}{N_{\text{model}}} \sum_{j=1}^{N_{\text{model}}} \mathcal{P}(y^*|\tilde{\omega}^j(t^*), x^*) \quad (2)$$



Dropout[3]

Dropout is an empirical technique that was proposed to avoid overfitting in CNN.
At each training step (i.e., for each sample within a mini-batch)

- Remove each node in the network with a probability p
- Update the weights of the remaining nodes with backpropagation.

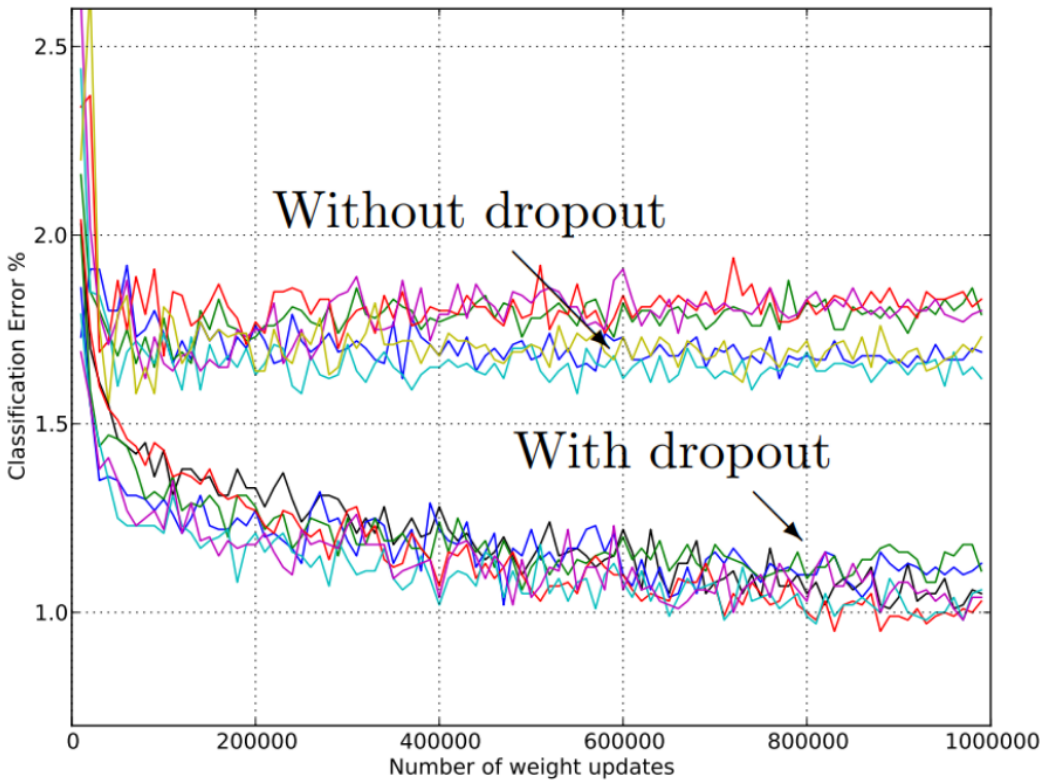


(a) Standard Neural Net

(b) After applying dropout.

[3]Image credit: G. Louppe

MC dropout[4]



Why does dropout work?

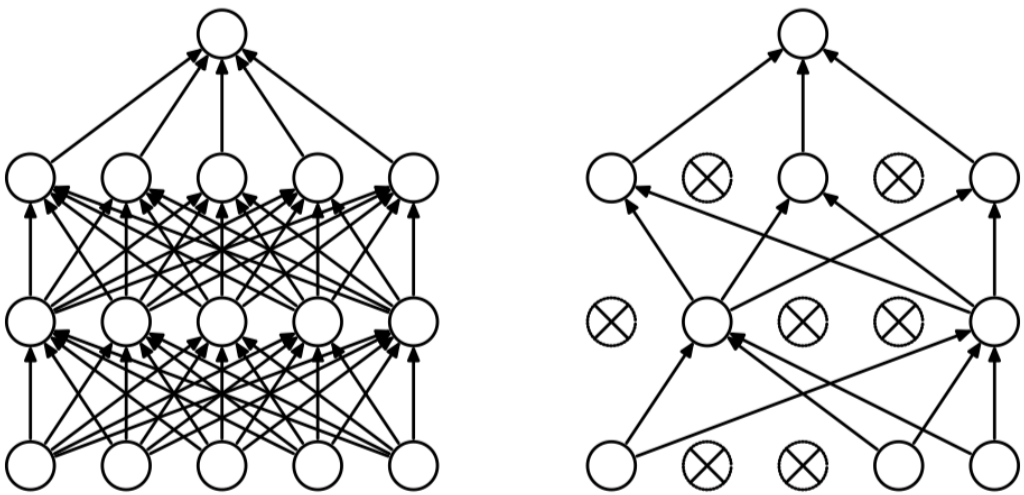
[4]Image credit: G. Louppe

MC dropout [2]

They [2] propose to average the predictions of several DNN where they apply the dropout:

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

with b^j a vector of the same size of $\omega(t^*)$ which is a realization of a binomial distribution.



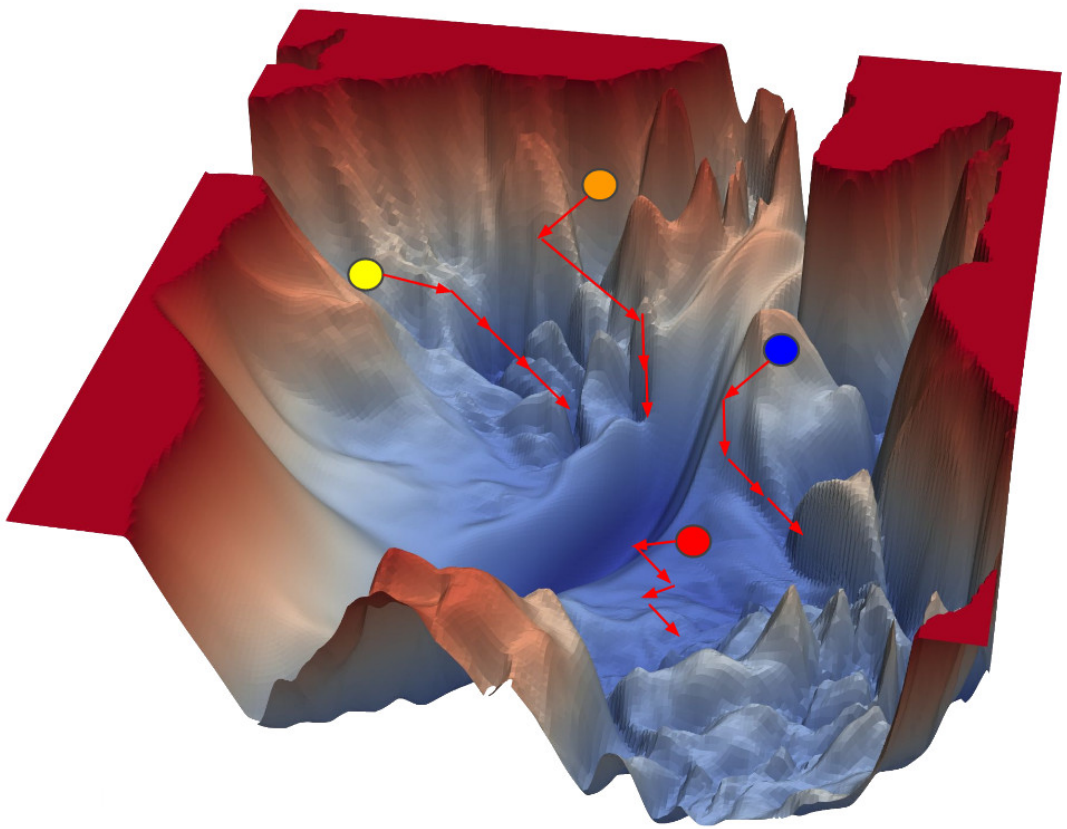
(a) Standard Neural Net

(b) After applying dropout.

Deep Ensembles[3]

They [3] propose to average the predictions of several DNN with different initial seeds:

$$\mathcal{P}(y^*|x^*) = \frac{1}{N_{\text{model}}} \sum_{j=1}^{N_{\text{model}}} \mathcal{P}(y^*|\omega^j(t^*), x^*) \tag{4}$$



Deep Ensembles[5]

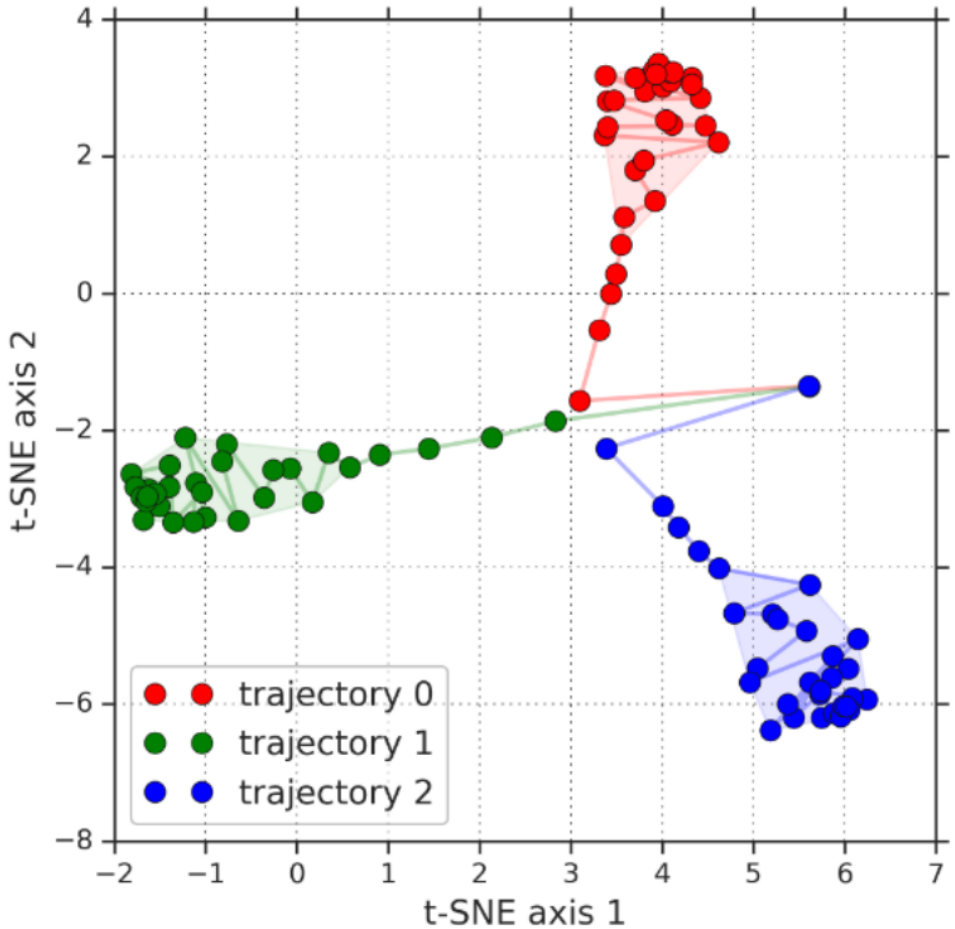


Figure: t-SNE plot of predictions from checkpoints corresponding to 3 different randomly initialized trajectorie

Deep Ensembles[5]

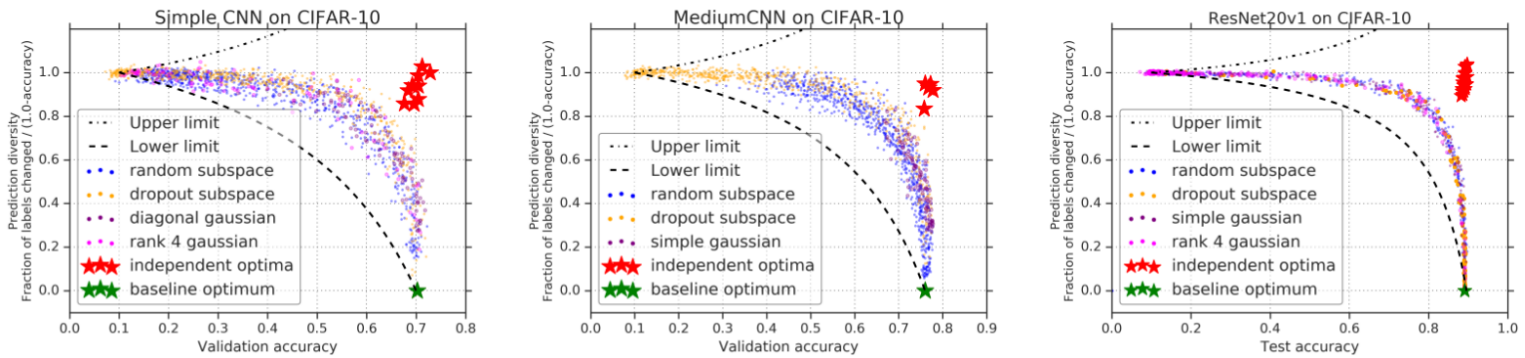


Figure: Diversity versus accuracy plots for 3 models trained on CIFAR-10

BatchEnsemble[7]

They [7] propose to approximate the average of the predictions of several DNN with different initial seeds by using a DNN with two kinds of weights. For simplicity ω has two sets of weights 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 $\omega^{\text{slow}} = \mathrm{W}_{\text{share}}$ and $\omega^{\text{fast}} = \{F_j\}_{j=1}^{N_{\text{model}}}$. We have $W_j = \mathrm{W}_{\text{share}} \cdot F_j = \mathrm{W}_{\text{share}} \cdot (r_j s_j^t)$

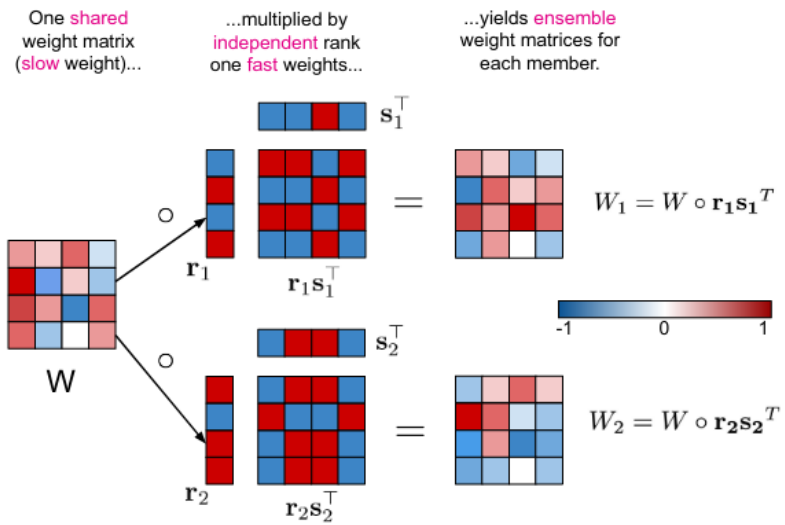


Figure: An illustration on how to generate the ensemble weights for two ensemble members

BatchEnsemble[7]

We have a set of weight $W_j = \mathrm{W}_{\text{share}} \cdot F_j = \mathrm{W}_{\text{share}} \cdot (r_j s_j^t)$ with $\mathrm{W}_{\text{share}}$ 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:

$$y = \phi\left(W_j^t x\right) = \phi\left((\mathrm{W}_{\text{share}}^t \cdot (r_j s_j^t))^t x\right) = \phi\left((\mathrm{W}_{\text{share}}^t (x \cdot r_j) \cdot s_j)\right)$$

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

$$\mathcal{P}(y^*|x^*) = \frac{1}{N_{\text{model}}} \sum_{j=1}^{N_{\text{model}}} \mathcal{P}(y^*|\omega^j, x^*) \tag{5}$$

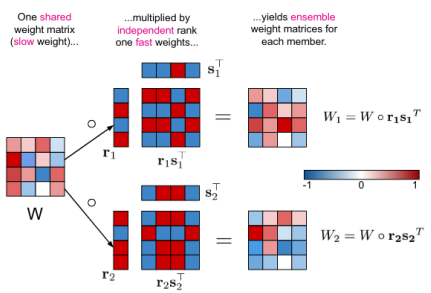
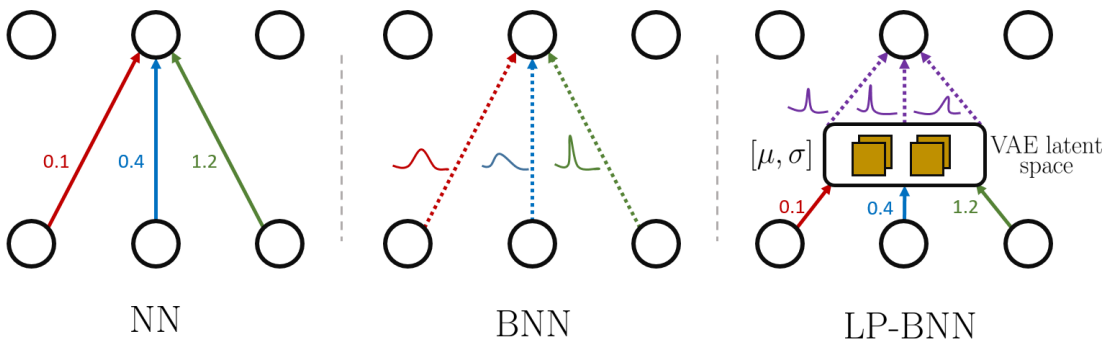


Figure: An illustration on how to generate the ensemble weights for two ensemble members

LP-BNN [8]

In classical BNN al



LP-BNN [8]

We proposed similarly to BatchEnsemble to use $W_j = \mathrm{W}_{\text{share}} \cdot F_j = \mathrm{W}_{\text{share}} \cdot (\hat{r}_j s_j^t)$ with $\mathrm{W}_{\text{share}}$ that sees all images and $(\hat{r}_j s_j^t)$ that does not see all the same images. If we denote ϕ an activation function then when we apply the LPBNN on an image we perform:

$$y = \phi\left(W_j^t x\right) = \phi\left((\mathrm{W}_{\text{share}}^t \cdot (\hat{r}_j s_j^t))^t x\right) = \phi\left((\mathrm{W}_{\text{share}}^t (x \cdot \hat{r}_j) \cdot s_j)\right)$$

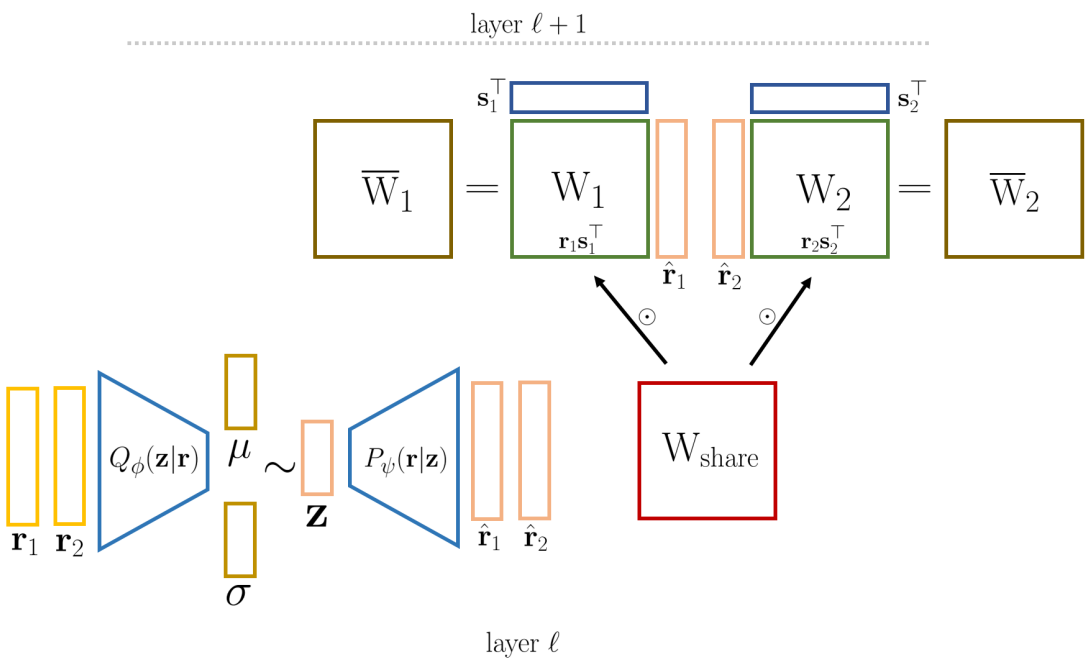
BUT we use $\hat{r}_j$ and not r_j, where $\hat{r}_j$ is the output of a VAE.
Hence, $\hat{r}_j = g_\psi^{\text{dec}}(z_j)$ and with $z_j \sim Q_\phi(z \mid r) = \mathcal{N}(z; \mu_j, \sigma_j^2 \mathrm{I})$ where $(\mu_j, \sigma_j) = g_\phi^{\text{enc}}(r_j)$.

LP-BNN [8]

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

$$\mathcal{P}(y^*|x^*) = \frac{1}{N_{\text{model}}} \sum_{j=1}^{N_{\text{model}}} \mathcal{P}(y^*|\theta^{\text{slow}}, s_j, \hat{r}_j, x^*) \tag{6}$$

with $\theta^{\text{slow}} = \{\mathrm{W}_{\text{share}}\}$



Metrics[1]

First we group predictions into M bins, each of size $1/M$. Let B_m be the set of indices of samples whose prediction confidence falls into the interval $I_m =]m - 1/M, m/M]$.
The accuracy of a set B_m is defined as:

$$\text{acc}(B_m) = 1/|B_m| \sum_{i \in B_m} \delta_{y_i}(\hat{y}_i) \tag{7}$$

The average confidence in B_m is defined as:

$$\text{conf}(B_m) = 1/|B_m| \sum_{i \in B_m} \hat{p}_i \tag{8}$$

where $\hat{p}_i$ is the confidence for sample i.

Metrics [1]

Expected Calibration Error (ECE) measures the difference in expected accuracy and expected confidence. It is defined as:

$$ECE = \sum_{m}^{M} \frac{|\text{acc}(B_m) - \text{conf}(B_m)|}{|B_m|} \tag{9}$$

Metrics[6]

The dataset is divided in two:

- Out of distribution
- in distribution

The confidence score $\hat{p}_i$ for sample i $\hat{p}_i$ is used to detect OOD data. To eveluate the quality we can use :

- Area Under the ROC Curve $\rightarrow$ AUC
- Area Under the Average Precision Curve $\rightarrow$ AUPR
- FPR at 95% TPR can be interpreted as the probability that a negative (out-of-distribution)example is misclassified as positive (in-distribution) when the true positive rate (TPR) is as high as 95%. True positive rate can be computed by TPR = TP / (TP+FN) and , the false positive rate (FPR) can be computed by FPR =FP / (FP+TN).

Classification

Method	CIFAR-10 Acc ↑	AUC ↑	AUPR ↑	FPR-95-TPR ↓	ECE ↓	cA ↑	cE ↓	CIFAR-100 Acc ↑	ECE ↓	cA ↑	cE ↓
MCP + cutout	96.33	0.9600	0.9767	0.115	0.0207	32.98	0.6167	80.19	0.1228	19.33	0.7844
MC dropout	95.95	0.9126	0.9511	0.282	0.0172	32.32	0.6673	75.40	0.0694	19.33	0.5830
MC dropout +cutout	96.50	0.9273	0.9603	0.242	0.0117	32.35	0.6403	77.92	0.0672	27.66	0.5909
DUQ†	87.48	0.7083	0.8114	0.698	0.3983	64.89	0.2542	-	-	-	-
DUQ Resnet18‡	93.36	0.8994	0.9213	0.1964	0.0131	69.01	0.5059	-	-	-	-
EDL†	85.73	0.9002	0.9198	0.247	0.0904	59.54	0.3412	-	-	-	-
MIMO	94.96	0.9387	0.9648	0.175	0.0300	**69.99**	0.1846	0.7869	0.1018	0.4735	0.2832
Deep Ensembles + cutout	**96.74**	**0.9803**	**0.9896**	**0.071**	**0.0093**	68.75	0.1414	**83.01**	**0.0673**	47.35	**0.2023**
BatchEnsembles + cutou	96.48	0.9540	0.9731	0.132	0.0167	**71.67**	0.1928	81.27	0.0912	47.44	0.2909
LP-BNN (ours) + cutout	95.02	**0.9691**	**0.9836**	**0.103**	**0.0094**	**69.51**	**0.1197**	79.3	0.0702	**48.40**	**0.2224**

Table: **Comparative results for image classification tasks**. We evaluate on CIFAR-10 and CIFAR-100 for the tasks: in-domain classification, out-of-distribution detection with SVHN (CIFAR-10 only), robustness to distribution shift (CIFAR-10-C, CIFAR-100-C). We run all methods ourselves in similar settings using publicly available code for related methods. Results are averaged over three seeds. †: We did not manage to scale these methods to WRN-28-10 on CIFAR-100.

Out of distribution (Results on the BDD anomaly experiments)

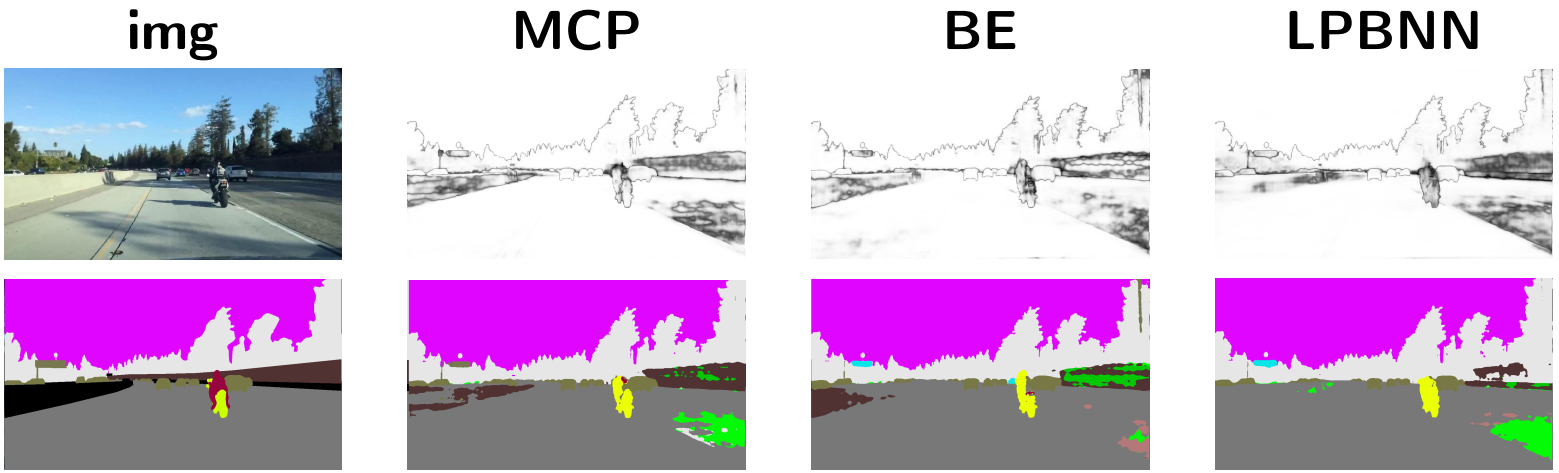


Figure: **Visual assessment on two BDD-Anomaly test images containing a motorcycle (OOD class).** For each image: *on the first row*, input image and confidence maps from Maximum Class Probability (MCP), BatchEnsemble (BE), and latent posterior of Bayesian neural networks (LP-BNN); *on the second row*, ground-truth segmentation and segmentation maps from MCP, BE, and LP-BNN. LP-BNN is less confident on the OOD objects.

Out of distribution (Results on the BDD anomaly experiments)

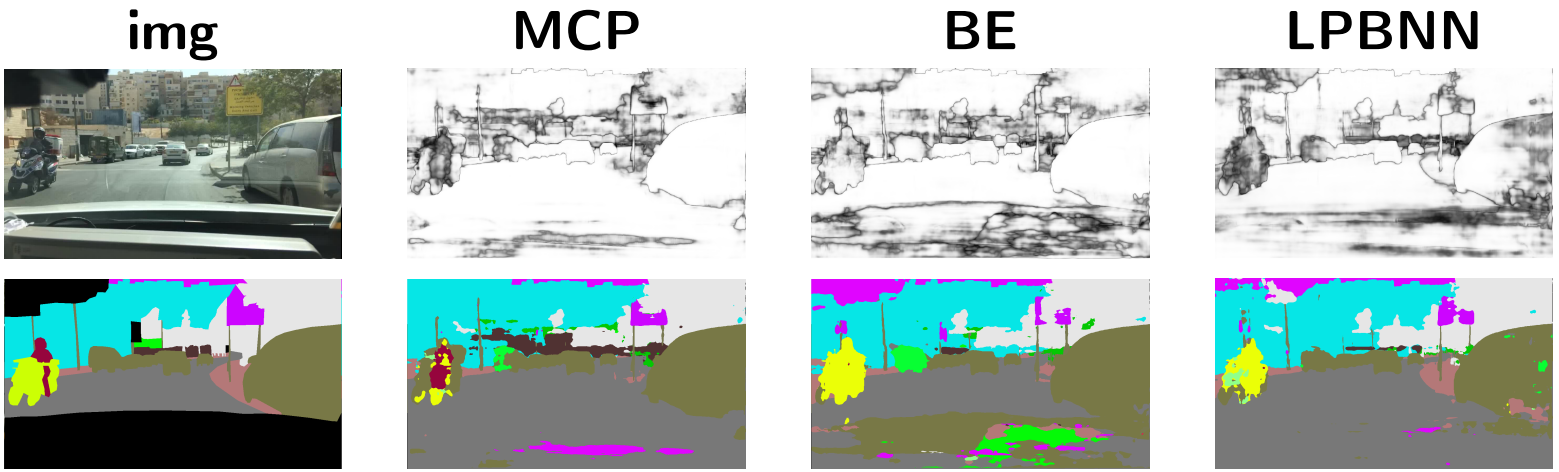


Figure: **Visual assessment on two BDD-Anomaly test images containing a motorcycle (OOD class).** For each image: *on the first row*, input image and confidence maps from Maximum Class Probability (MCP), BatchEnsemble (BE), and latent posterior of Bayesian neural networks (LP-BNN); *on the second row*, ground-truth segmentation and segmentation maps from MCP, BE, and LP-BNN. LP-BNN is less confident on the OOD objects.

Conclusions

We proposed a technique that learns the posterior of the DNN that is stable and improves the uncertainty quantification.
Thanks for your attention.

Bibliography:

1 Guo, Chuan, et al. "On calibration of modern neural networks." Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017.

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

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

Bibliography:

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

6 Hendrycks, Dan, et al. "A Benchmark for Anomaly Segmentation." arXiv preprint arXiv:1911.11132 (2019)

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

8 Franchi, G., Bursuc, A., Aldea, E., Dubuisson, S., & Bloch, I. (2020). Encoding the latent posterior of Bayesian Neural Networks for uncertainty quantification. arXiv preprint arXiv:2012.02818.

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