# Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles

*2020. 01. 23*

*JoonHyung Park*

*deepjoon@kaist.ac.kr*

# Table of Contents

# Background

- Goal: Quantifying predictive uncertainty in NNs
  - Bayesian NNs are the SOTA for estimating predictive uncertainty
  - Learn a distribution over weights
- Drawbacks of Bayesian NNs
  - Modifications to the training procedure
  - Computationally expensive

→ **Non-Bayesian approach**

# Contributions

- Training <u>probabilistic NNs</u> using a proper scoring rule as the training criterion

    Modeling predictive distributions

    - Simple to implement
    - Requires few modifications to standard NNs

- Investigate the effect of two modifications to the training pipeline
    - Ensembles & Adversarial training
    - Well suited for distributed computation
    - Attractive for large-scale deep learning

# Details of the model

- Problem setup
    - Regression problem
    - Dataset: $\mathcal{D} = \{x_n, y_n\}$ $(n = 1, \ldots, N)$
    - We use a NN to model the probabilistic predictive distribution $p_\theta(y|x)$, where $\theta$ are the parameters of the NN

- Simple recipe for predictive uncertainty estimation
    1. Use a proper scoring rule as the training criterion
    2. Use adversarial training to smooth the predictive distributions
    3. Train an ensemble

# Proper scoring rules

- Measure the quality of predictive uncertainty (Higher is better)

$$\text{S: } p_\theta, (x, y) \Longrightarrow \mathcal{R}$$

- Evaluates the quality relative to an event $y|x \sim q(y|x)$

True distribution on (y, x)

- Expected scoring rule

$$S(p_\theta, q) = \int q(y, x) S(p_\theta, (y, x)\,) dydx$$

$$S(p_\theta, q) \leq S(q, q), \text{ with equality if and only if } p_\theta(y|x) = q(y|x)$$

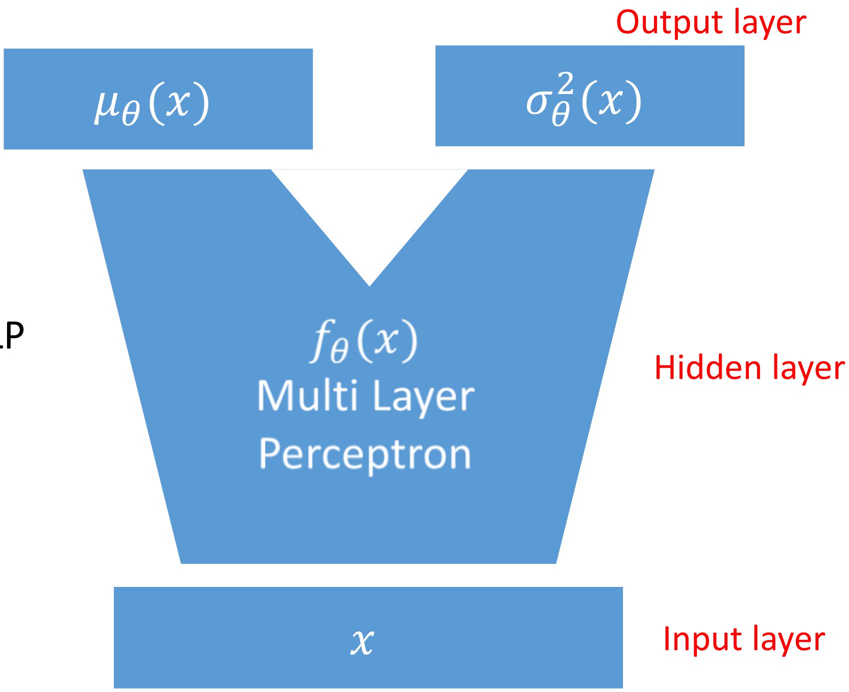
$$\text{Loss: } \mathcal{L}(\theta) = -S(p_\theta, q)$$

- It turns out many common NN loss functions are proper scoring rules!
  - Maximizing log-likelihood: $S(p_\theta, q)$ = log $p_\theta(y|x)$
  - NLL is used as the loss function for regression: $\mathcal{L}(\theta)$ = - log $p_\theta(y|x)$

# Training criterion for regression

- NNs usually output a single value $\mu(x)$, and the parameters are optimized to minimize the MSE
  - MSE does not capture predictive uncertainty
- We use a network that outputs two values in the final layer

Gaussian MLP

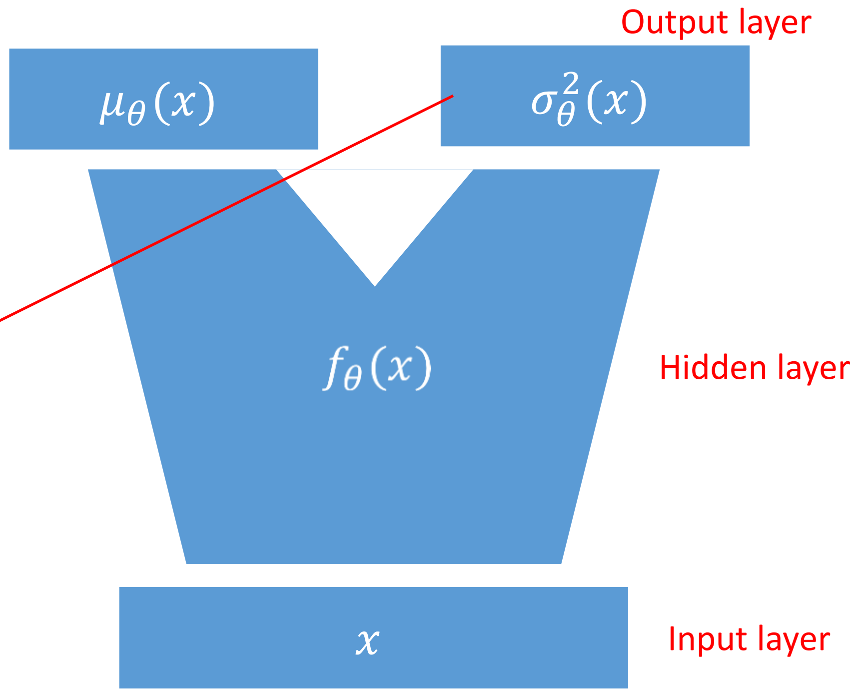


# Training criterion for regression

- Treating the observed value as a sample from a Gaussian distribution with the predicted mean $\mu_\theta(x)$ and variance $\sigma_\theta(x)$
- Minimize the NLL criterion using gaussian approximation

$$\mathcal{L}(\theta) = -\log p_\theta(y|x)$$
$$= \frac{log\sigma_\theta^2(x)}{2} + \frac{\left(y-\mu_\theta(x)\right)^2}{2\sigma_\theta^2(x)}$$

[Positive constraint]
variance = softplus(variance) + $10^{-6}$



# Adversarial training

- Adversarial examples
  - ‘Close’ to the original training examples
  - But examples are misclassified by the NN

- Fast gradient sign method
  - Fast solution to generate adversarial examples
  - $x' = x + \epsilon sign\left(\nabla_x \mathcal{L}(\theta, x, y)\right)$

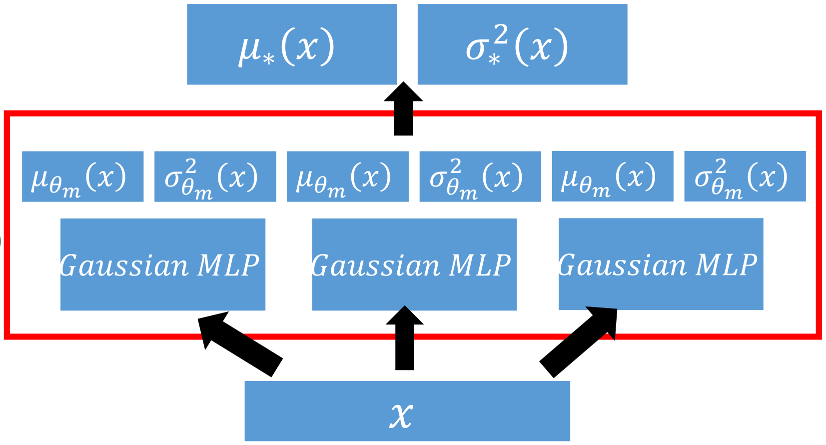
- Adversarial examples can be used to augment the original training set
  - By treating $(x', y)$

# Ensembles

- Bagging
  - For parallel computing
  - Ensemble members are trained on ~~different samples~~ of the original training set
    All of training set (In this paper)

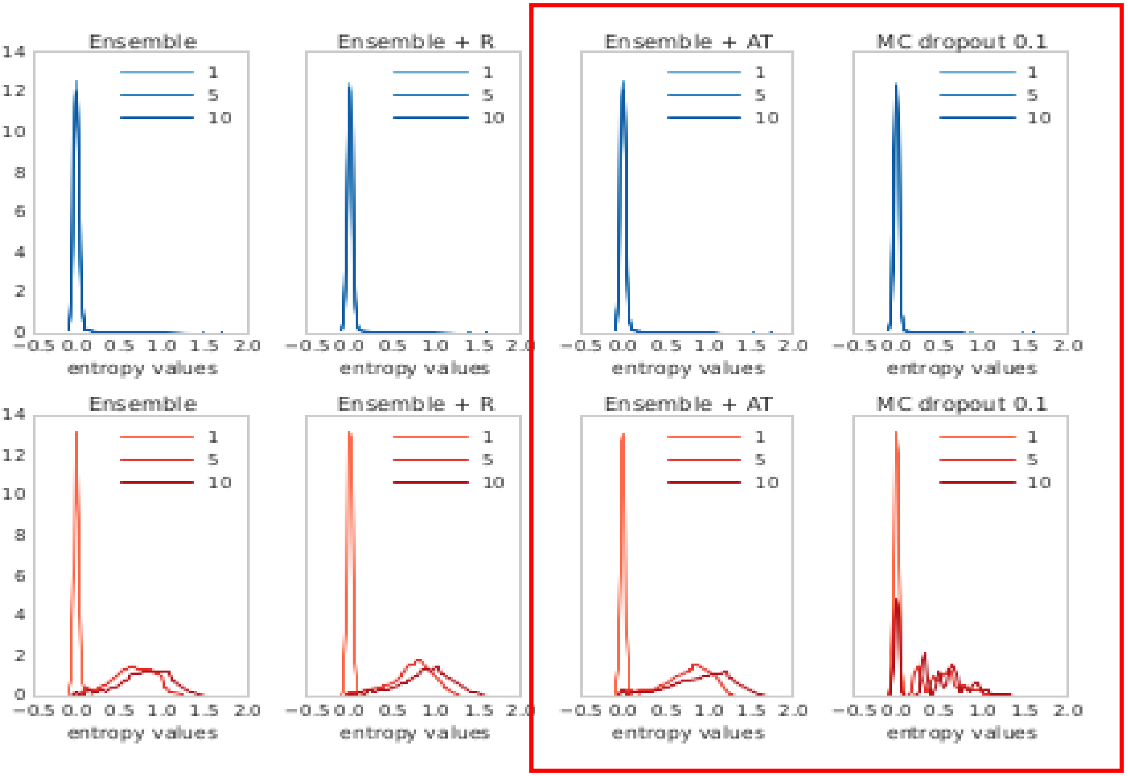
- Treat the ensemble as a uniformly-weighted mixture model
  - Gaussian mixture model
  - $M^{-1} \sum_{m=1}^{M} \mathcal{N}(\mu_{\theta_m}(x), \sigma_{\theta_m}^2(x))$ ($M$: # of NNs in ensemble)

- Mean and variance of a mixture
  - $\mu_*(x) = M^{-1} \sum_{m=1}^{M} \mu_{\theta_m}(x)$
  - $\sigma_*^2(x) = M^{-1} \sum_{m=1}^{M} (\sigma_{\theta_m}^2(x) + \mu_{\theta_m}^2(x)) - \mu_*^2(x)$



# Experimental Results

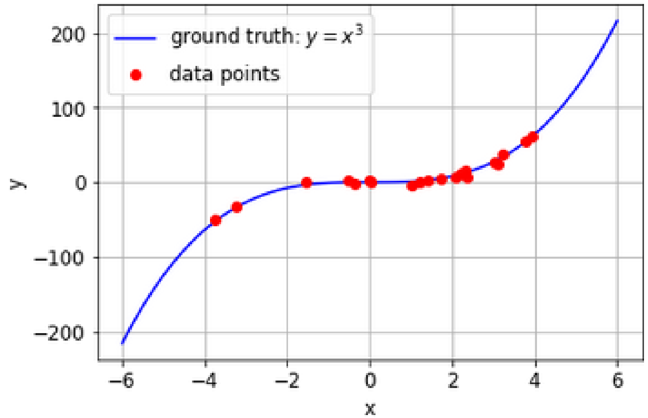
- Uncertainty evaluation
  - Test examples from **known vs unknown classes**
  - Train data/ Test data(known class): MNSIT data
  - Test data(unknown class): Same size as MNIST, however the labels are **alphabets**




(a) MNIST-NotMNIST

# TODO: Experiment 3.2

- Reproduce the graphs of experiment 3.2
- Dataset (Done)
  - Consists of 20 training examples drawn as $y = x^3 + \epsilon,\ \epsilon \sim \mathcal{N}(0, 3^2)$




- Drawing graph(Done)

```
# Training a Gaussian MLP(single network) with NLL score rule
# TODO:Draw Fig1.2
elif args.fig == 2:

    # Have to calculte predicted mean and var
    # 'mean' have to be a numpy array with shape [100,1]
    # 'var'  have to be a numpy array with shape [100,1]
    mean = np.random.randn(100,1)
    var = np.random.randn(100,1)
    draw_graph(x,x_set,y_set,mean, np.sqrt(var))
```

- Calculate predicted mean and variance for 4 cases

https://github.com/JoonHyung-Park/DeepEnsemble



MLILAB Machine Learning & Intelligence

# TODO: Experiment 3.2

- Reproduce the experiment 3.2

[code]
Training data: x_set, y,set
Test data: x

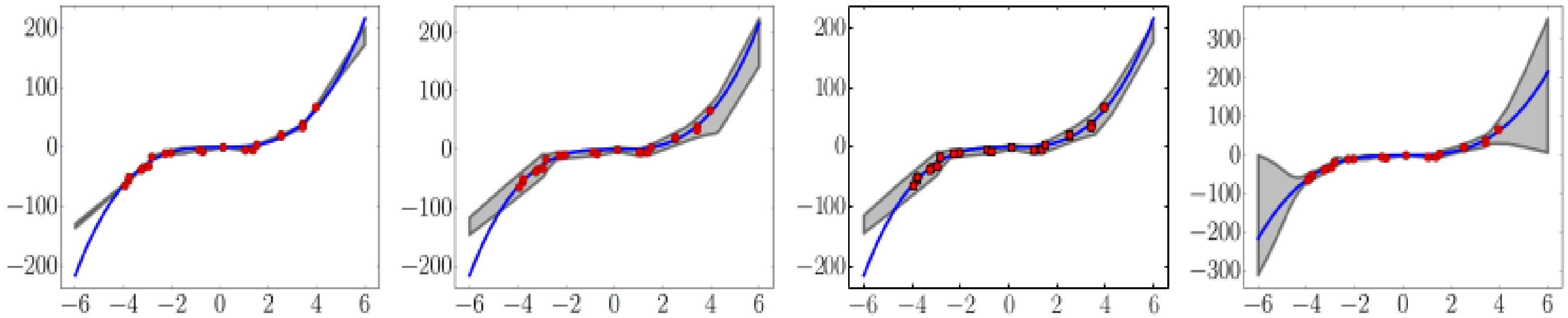
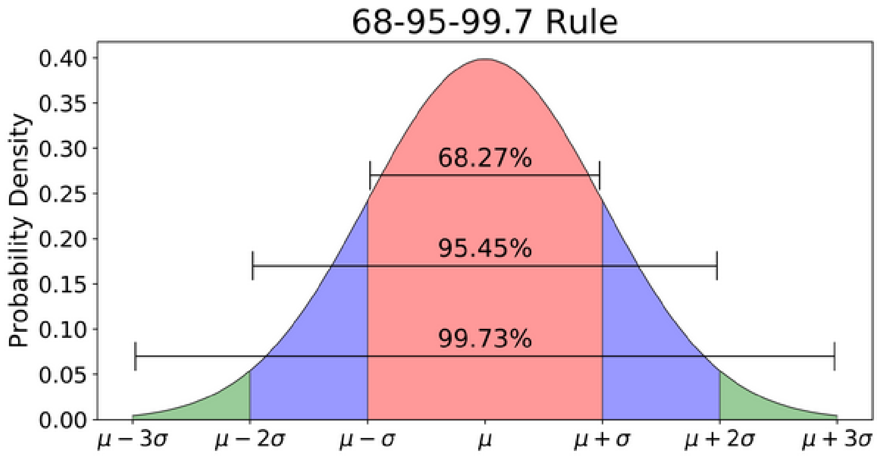


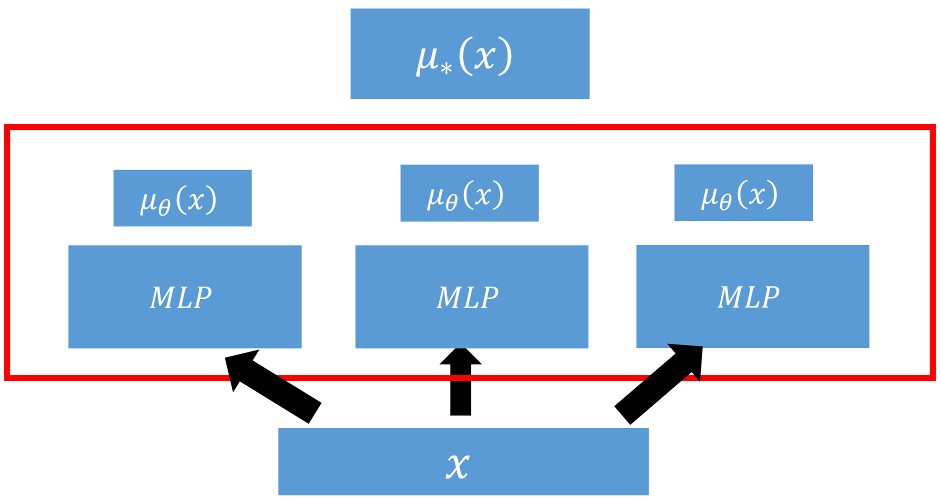

Fig 1 Fig 2 Fig 3 Fig 4

- Gray lines correspond to the predicted mean along with three standard deviations



# TODO: Fig. 1

- Training an ensemble of 5 networks with MSE
- Variance?
  - Obtain multiple points predictions and use the empirical variance of the networks' predictions as an approximate measure of uncertainty




Prediction mean (Network 1)

| | | | | |
|---|---|---|---|---|
| 8.4 | 5.5 | 65.6 | 125.3 | 1.4 |

Prediction mean (Network 2)

| | | | | |
|---|---|---|---|---|
| 7.9 | 6.1 | 64.3 | 121.6 | 0.8 |

Prediction mean (Network 3)

| | | | | |
|---|---|---|---|---|
| 8.8 | 6.3 | 66.9 | 131.9 | 1.9 |

Predicted mean (Ensemble)

| | | | | |
|---|---|---|---|---|
| 8.36 | 5.96 | 65.6 | 126.2 | 1.36 |

Standard deviation (Ensemble)

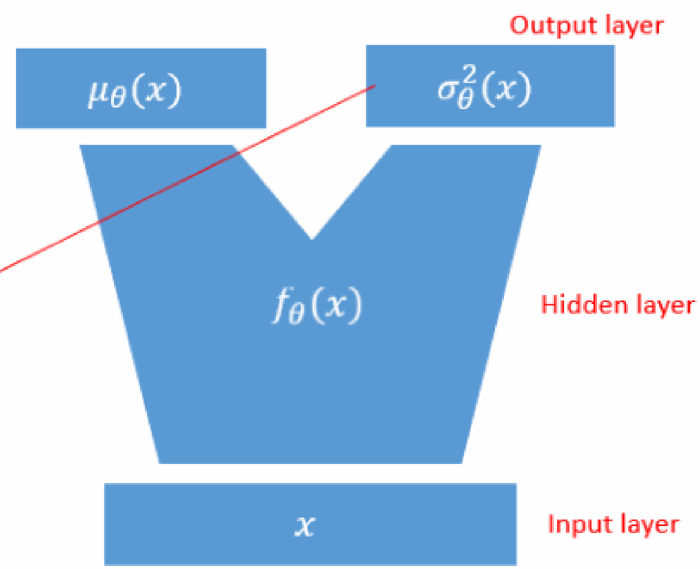
| | | | | |
|---|---|---|---|---|
| 0.45 | 0.41 | 1.3 | 5.21 | 0.55 |

# TODO: Fig. 2

- Training a Gaussian MLP (single network) with NLL score rule

$$\mathcal{L}(\theta) = -\log p_\theta(y|x)$$
$$= \frac{\log\sigma_\theta^2(x)}{2} + \frac{(y-\mu_\theta(x))^2}{2\sigma_\theta^2(x)}$$

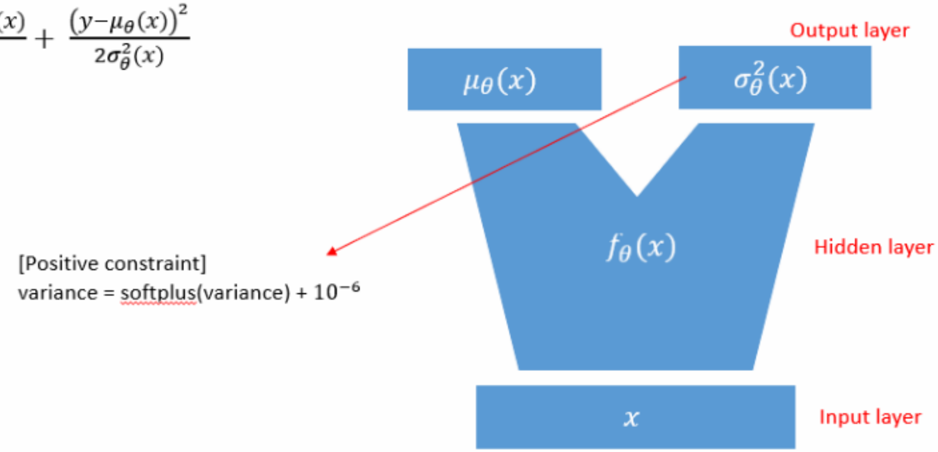
[Positive constraint]
variance = softplus(variance) + $10^{-6}$



# TODO: Fig. 3

- Training a Gaussian MLP (single network) with NLL score rule & Adversarial training

$$\mathcal{L}(\theta) = -\log p_\theta(y|x) = \frac{\log\sigma_\theta^2(x)}{2} + \frac{(y-\mu_\theta(x))^2}{2\sigma_\theta^2(x)}$$

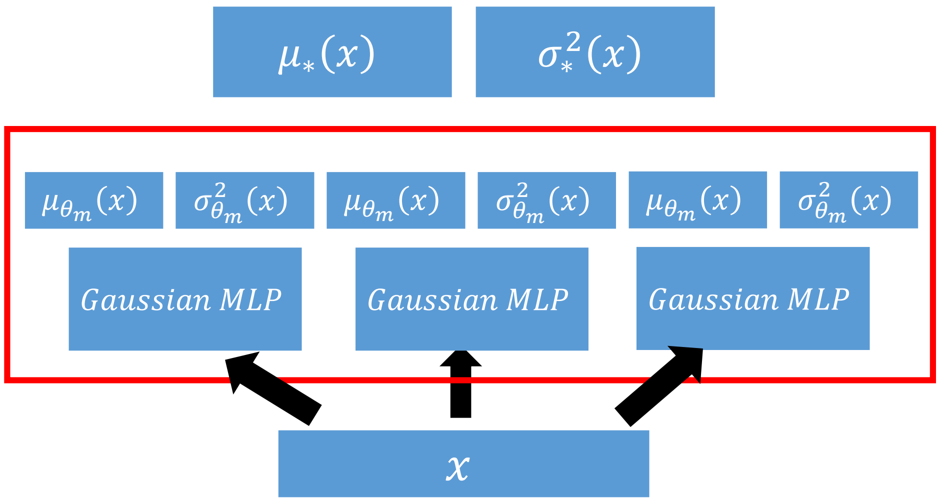



## Fast gradient sign method

- Fast solution to generate adversarial examples
- $x' = x + \epsilon sign\left(\nabla_x \mathcal{L}(\theta, x, y)\right)$

# TODO: Fig. 4

- Training a Gaussian mixture MLP (Deep Ensemble) with NLL & Adversarial training




Mean and variance of a mixture

- $\mu_*(x) = M^{-1} \sum_{m=1}^{M} \mu_{\theta_m}(x)$
- $\sigma_*^2(x) = M^{-1} \sum_{m=1}^{M} (\sigma_{\theta_m}^2(x) + \mu_{\theta_m}^2(x)) - \mu_*^2(x)$

# **Thank you !**

Any Questions ?