Uncertainty quantification in Machine learning

Presenter: R. Lehe

Day 8

Outline

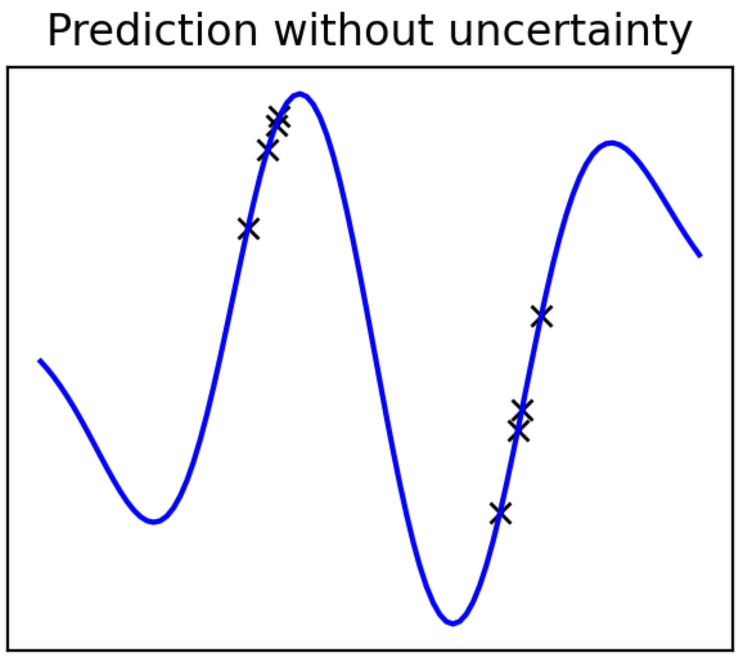
- Uncertainty in ML: definition and motivation

- Methods to estimate uncertainty
  - Gaussian processes: reminder
  - Ensemble methods
  - Monte Carlo drop-out
  - Bayesian neural networks
  - Quantile regression

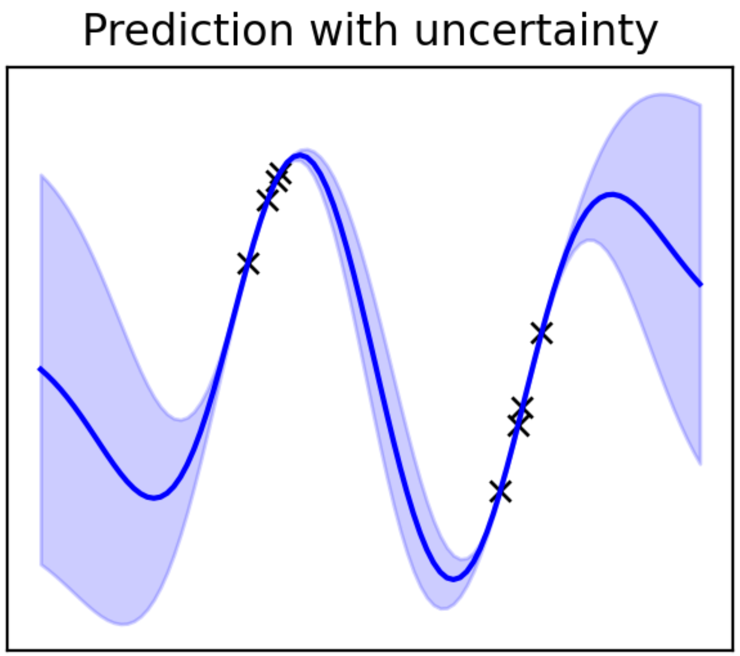
- Evaluating and calibrating uncertainty

Uncertainty in Machine Learning

Idea: The ML model should output a **prediction** and the corresponding **uncertainty.**



e.g. neural networks

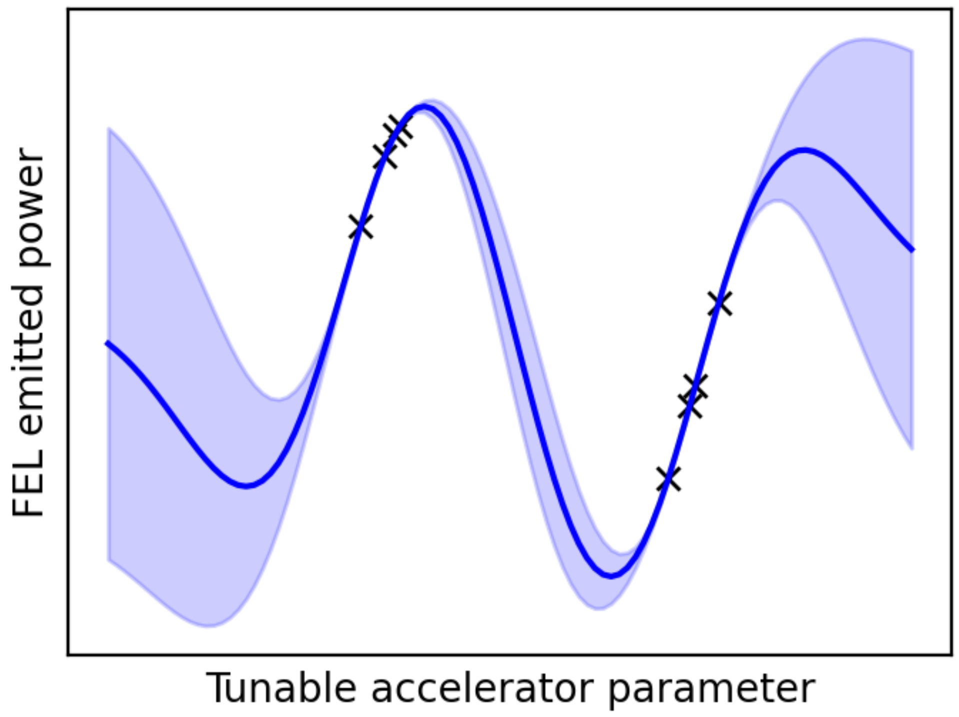


e.g. Gaussian processes

The uncertainty indicates the **probable interval** within which an actual evaluation may be.
(e.g. actual measurement or simulation)

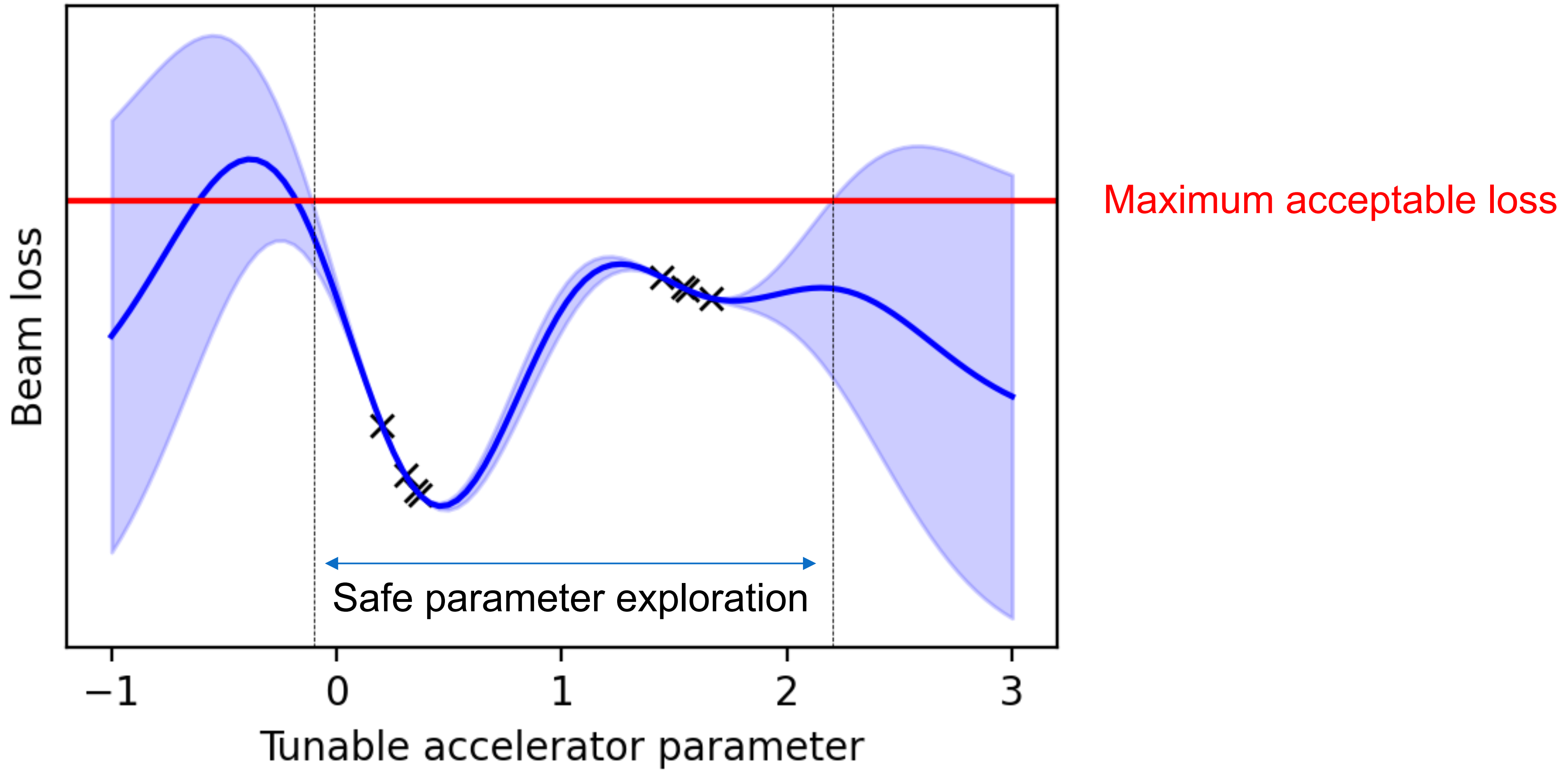
Motivation for accelerators: optimization

In the context of **model-based optimization** of accelerators:
uncertainty allows to balance **exploration and exploitation**.
(e.g. by calculating upper confidence bound, expected improvement)



Motivation for accelerators: safe operation

For **safe operation** of accelerators:
uncertainty helps ensure that **important constraints** are not **violated**.



Scope of this lecture

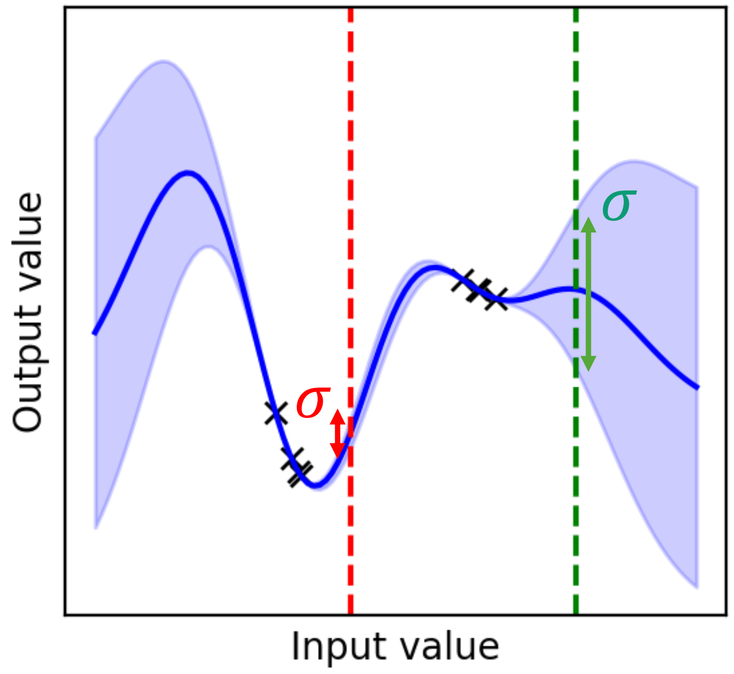
- Reliably evaluating the uncertainty in ML is very much still a **topic of research**.

- This lecture will describe different **well-known methods**, so that you can more easily navigate the corresponding ML literature in the future.

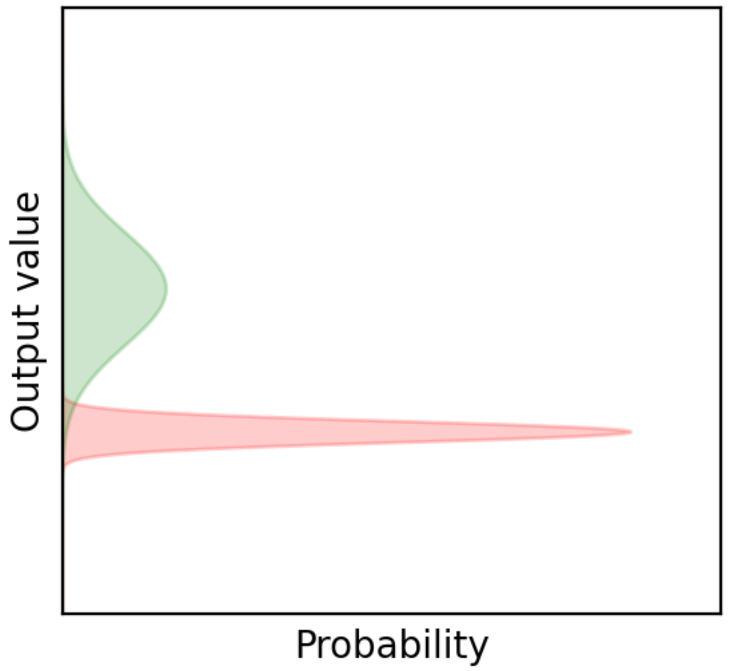
Uncertainty in Machine learning

Several representations for the uncertainty:

Standard deviation
(Single scalar)



Probability distribution
(Full function)



The **probability distribution** is a much more **complete** description, but few ML method provide it.

Epistemic and aleatoric uncertainty

Evaluations can often be modeled as:

$$f(\boldsymbol{x}) = \tilde{f}(\boldsymbol{x}) + \eta$$

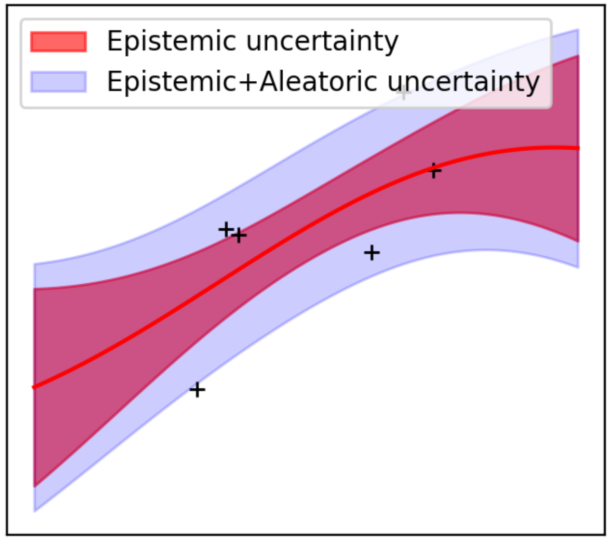
Underlying function
always gives the same result, for a given ***x***

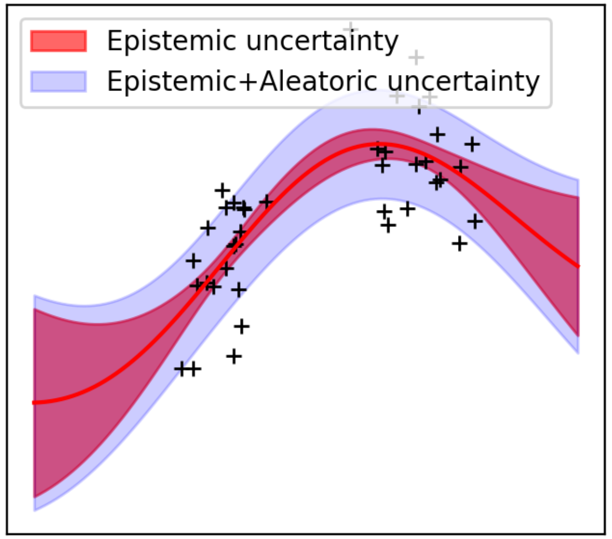
Intrinsic noise
value changes for each evaluation

Epistemic uncertainty:
uncertainty on underlying function

- increases when making predictions far from known data
- decreases when acquiring more data

Aleatoric uncertainty:
estimates the amplitude of the noise





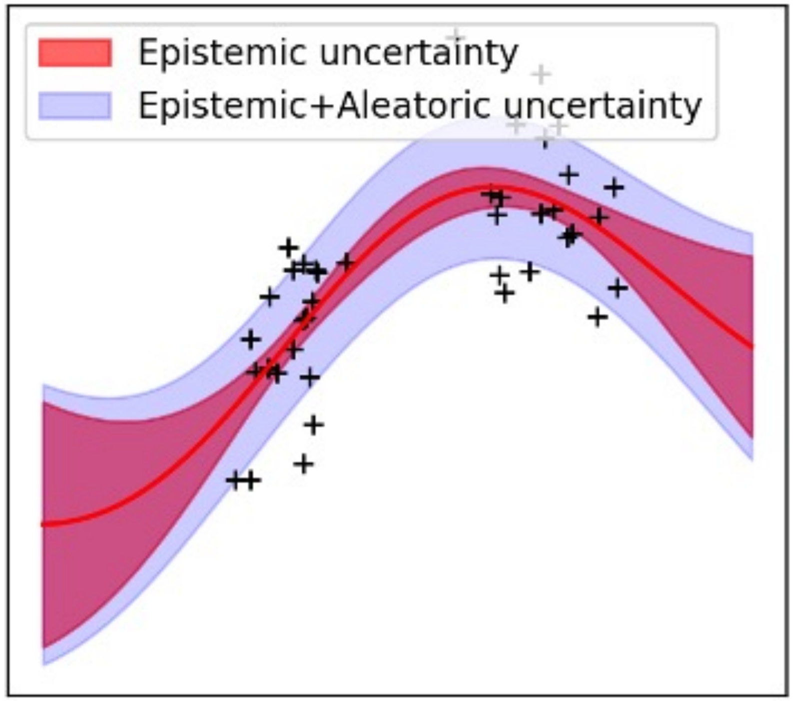
Epistemic and aleatoric uncertainty

Depending on the application, one may or may not want to include the **aleatoric part**:

Examples:

- **Optimizing beam size, with noisy beam size measurements:** the aim is to optimize the underling function $\tilde{f}$; the aleatoric part should not be included.

- **Keeping fluctuating beam loss under a threshold:** take into account aleatoric part, in order to evaluate the "worst-case scenario".



Outline

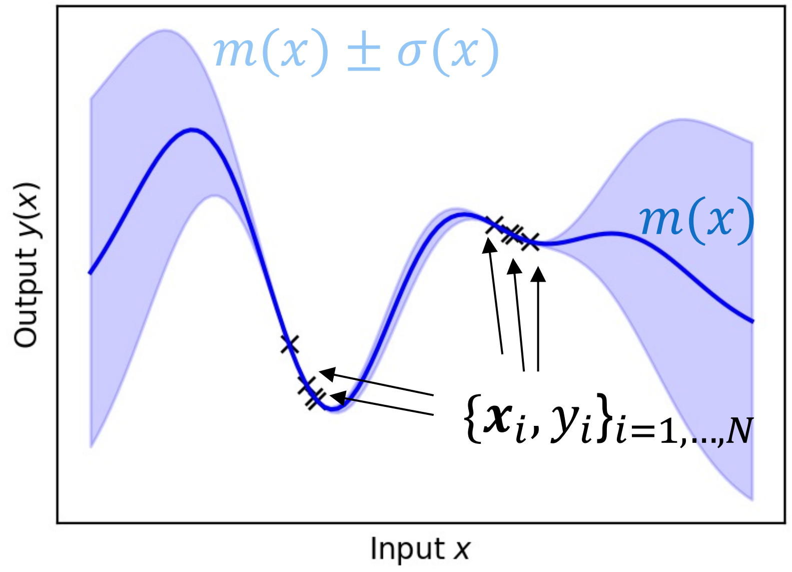
- Uncertainty in ML: definition and motivation

- Methods to estimate uncertainty
  - **Gaussian processes: reminder**
  - Ensemble methods
  - Monte Carlo drop-out
  - Bayesian neural networks
  - Quantile regression

- Evaluating and calibrating uncertainty

Reminder on Gaussian processes

Given N **previous evaluations** $\{\boldsymbol{x}_i, y_i\}_{i=1,\ldots,N}$, the **probability distribution** of $y(\boldsymbol{x}^*)$ at a new input $\boldsymbol{x}^*$ is predicted to be Gaussian: $y(\boldsymbol{x}^*) \sim \mathcal{N}\left(m(\boldsymbol{x}^*), \sigma^2(\boldsymbol{x}^*)\right)$



$$m(\boldsymbol{x}^*) = \boldsymbol{k}^{*T}(K + \sigma_\eta^2 I)^{-1}\boldsymbol{y}$$
$$\sigma^2(\boldsymbol{x}^*) = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}^{*T}(K + \sigma_\eta^2 I)^{-1}\boldsymbol{k}^*$$

(Rasmussen & Williams, "GP for ML", Eqns. (2.22)-(2.24))

$k(.,.)$: chosen kernel function (e.g. SE: $k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp(-\frac{(\boldsymbol{x}-\boldsymbol{x}')^2}{\ell^2})$)

σ_η: estimated noise level

Determined by **hyperparameter tuning** (e.g. maximization of marginal log-likelihood)

K: matrix of size $N \times N$, defined by $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$

$\boldsymbol{y}$: vector of size N, containing evaluations y_i

$\boldsymbol{k}^*$: vector of size N, defined by $k_i^* = k(\boldsymbol{x}_i, \boldsymbol{x}^*)$

Limitations of Gaussian processes

- Scales badly for **high-dimensional input**:
  - Suffers from **curse of dimensionality**,
    i.e. needs exponentially more data for high dimension
  - As more data is added, **computational cost** scales as n^3
  - Difficulties capturing **correlated input dimensions**
    (i.e. need many more hyperparameters in kernel)

- Inefficient for **high-dimensional output:**
  (essentially need to build a separate GP for each output)

- Predicted probability distribution is always Gaussian.
  Cannot predict distributions with long tails.

Outline

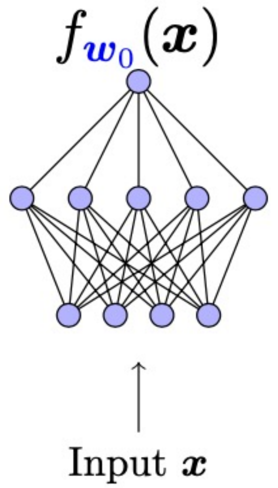
- Uncertainty in ML: definition and motivation

- Methods to estimate uncertainty
  - Gaussian processes: reminder
  - **Ensemble methods**
  - Monte Carlo drop-out
  - Bayesian neural networks
  - Quantile regression

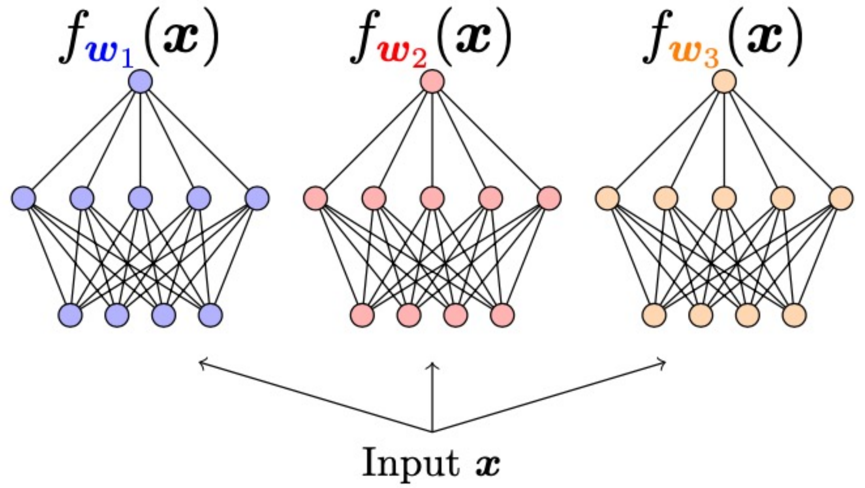
- Evaluating and calibrating uncertainty

Ensemble of neural networks

Regular neural network



Ensemble of neural network (N=3)


$f_{\boldsymbol{w}_1}(\boldsymbol{x})$
$f_{\boldsymbol{w}_2}(\boldsymbol{x})$
$f_{\boldsymbol{w}_3}(\boldsymbol{x})$
Input $\boldsymbol{x}$

- Due to **randomness** in initialization and training, each neural network has **different weights**, and gives a **different answer**.

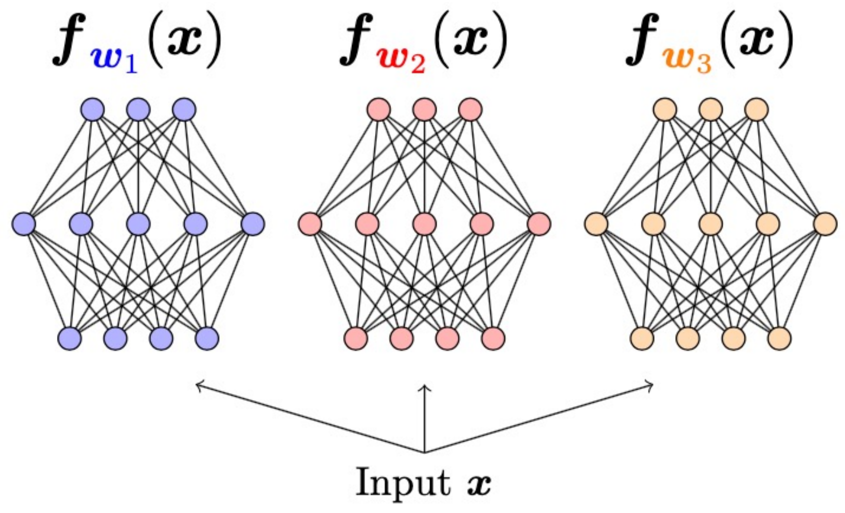
$$f(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} f_{\boldsymbol{w}_i}(\boldsymbol{x})$$

- Use the **mean** as the **prediction**
  Use the **standard deviation** as the **uncertainty**

$$\sigma_f(\boldsymbol{x}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(f_{\boldsymbol{w}_i}(\boldsymbol{x}) - f(\boldsymbol{x})\right)^2}$$

Ensemble of neural networks

Easily scales to **high-dimensional output**

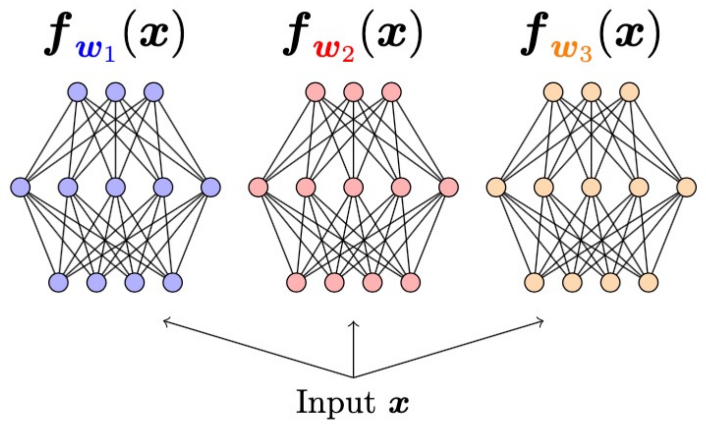

$\boldsymbol{f}_{\boldsymbol{w}_1}(\boldsymbol{x})$
$\boldsymbol{f}_{\boldsymbol{w}_2}(\boldsymbol{x})$
$\boldsymbol{f}_{\boldsymbol{w}_3}(\boldsymbol{x})$
Input $\boldsymbol{x}$

$$f_j(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} f_{j,\boldsymbol{w}_i}(\boldsymbol{x})$$

$$\sigma_{f_j}(\boldsymbol{x}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(f_{j,\boldsymbol{w}_i}(\boldsymbol{x}) - f_j(\boldsymbol{x})\right)^2}$$

Use **per-component** mean and standard deviation

Ensemble: how to make the models different?



Use randomness in **initialization** and/or **training data**.

Several possible methods:

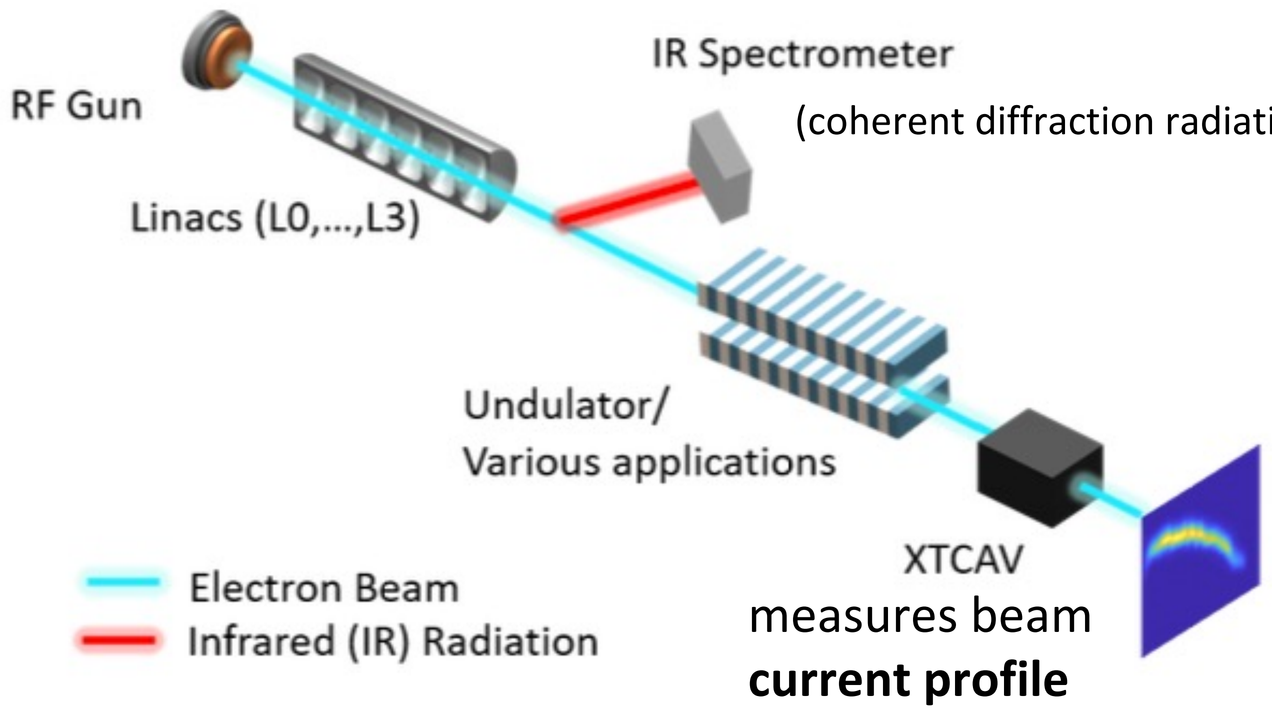
- Initialize **weights** of each network with a different random seed
  (Train all networks on the same data.)

- Randomly divide the data into **N partitions**
  Train each network on a **different partition** (with same initial weights)

- Different random initial weights **and** draw different random subsets of the data
  (“Boostrap AGGregatING” or “bagging”)

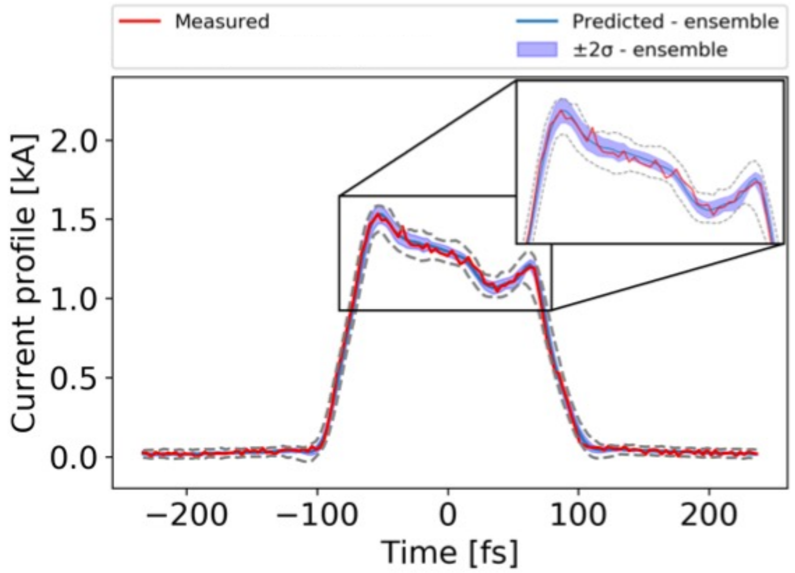
Example: uncertainty on virtual diagnostic for beam current

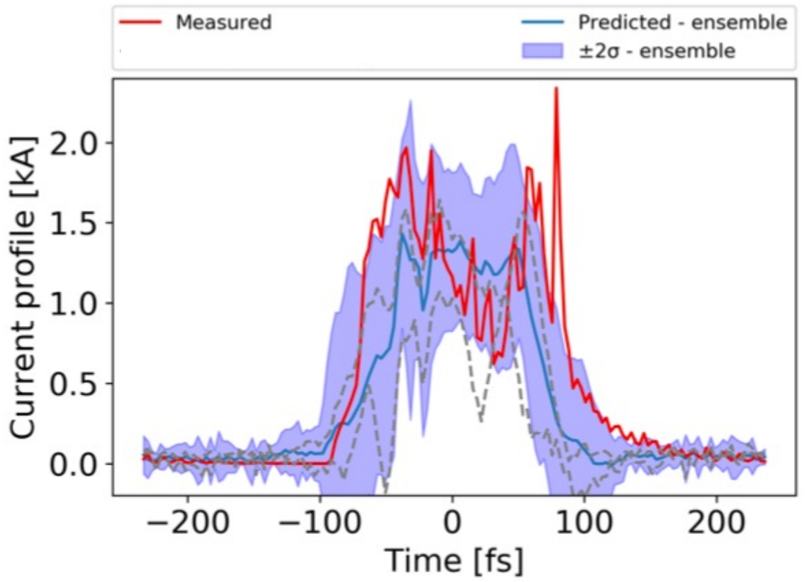
O. Convery et al., arXiv:2105.04654v1 (2021)



measures beam **current profile**

2 different shots





Ensemble of **16 independent neural networks**, trained with **bagging:**

- input: full IR spectrum
- output: 1d beam current profile

Outline

- Uncertainty in ML: definition and motivation

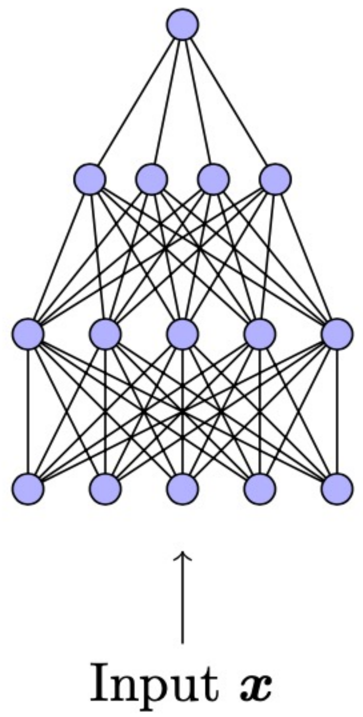
- Methods to estimate uncertainty
  - Gaussian processes: reminder
  - Ensemble methods
  - **Monte Carlo drop-out**
  - Bayesian neural networks
  - Quantile regression

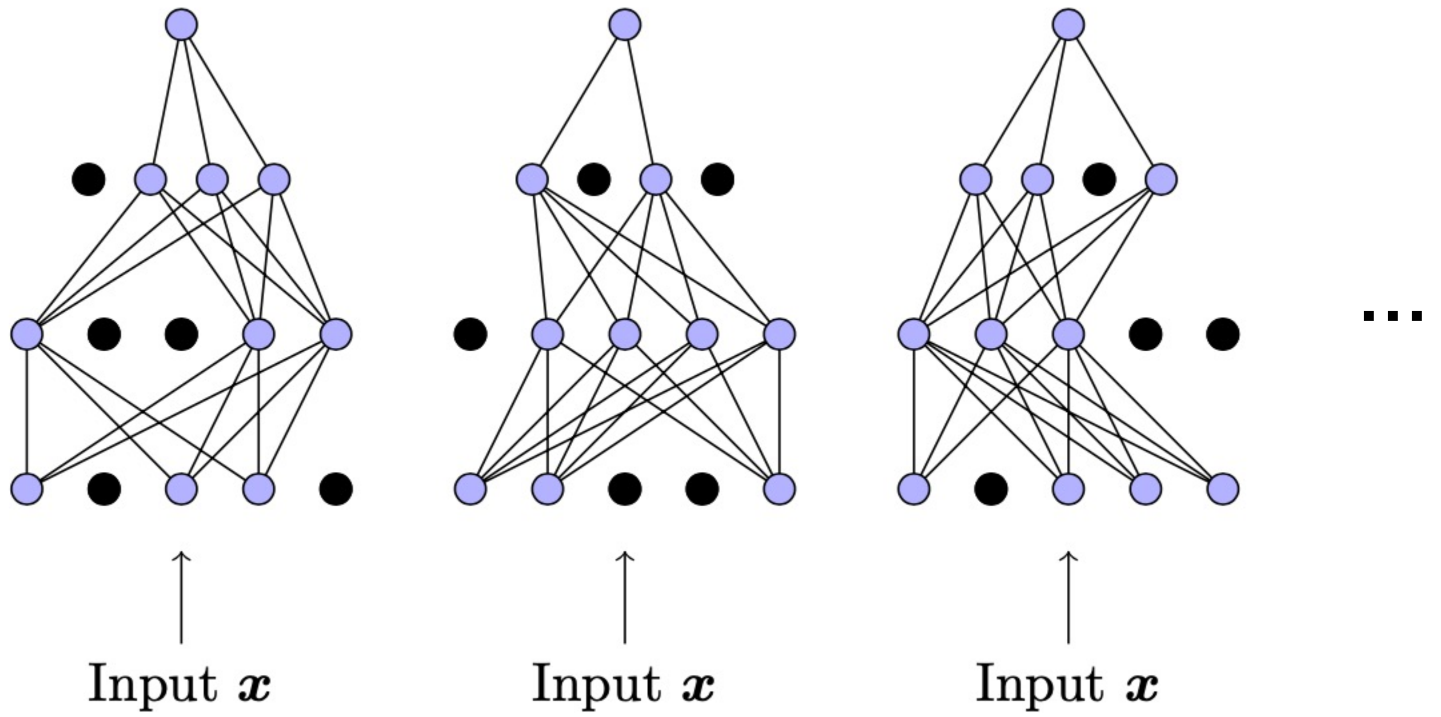
- Evaluating and calibrating uncertainty

Dropout neural network

Regular neural network

Drop-out neural network: repeated evaluations …





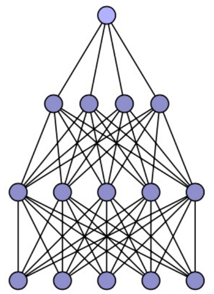
For each neuron, randomly set the activation to 0 with fixed probability *p*
(generate different random draw for each evaluation of the neural network)

Standard dropout vs. Monte-Carlo dropout

Standard Dropout:
(default behavior in `pytorch`, `keras`)

- Dropout is only applied during **training**
- During **inference** (i.e. for predictions), the activations are multiplied by *(1-p)* to represent the **"average behavior"**

During inference, repeated evaluations with the same input **x** give the **same result**.

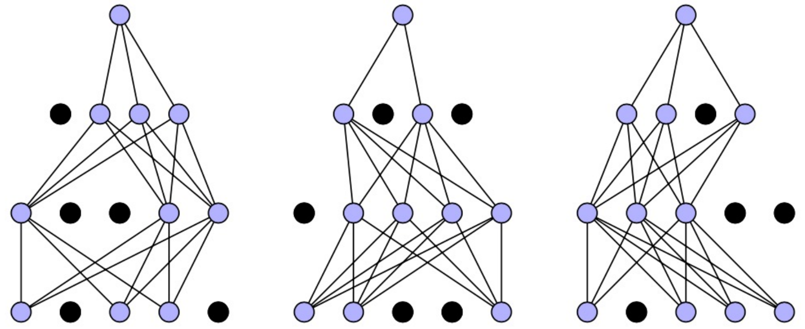


Monte-Carlo dropout (MC dropout):

Dropout is applied **both** during **training** and **inference**.

During inference, repeated evaluations with the same input **x** give **different results**.

Use the **mean** as the **prediction**
Use the **standard deviation** as the **uncertainty**



Outline

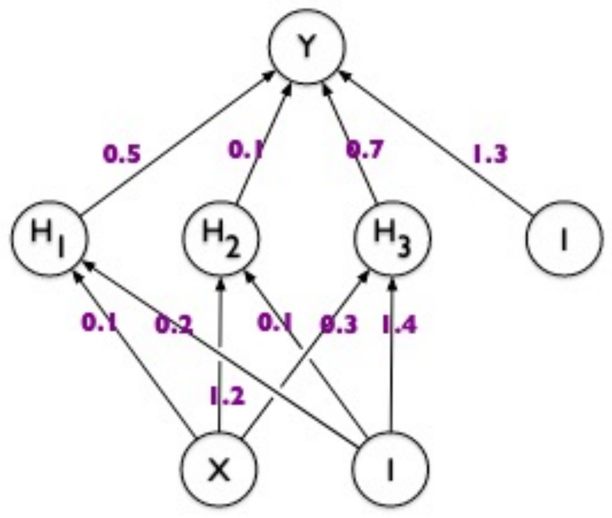
- Uncertainty in ML: definition and motivation
- Methods to estimate uncertainty
  - Gaussian processes: reminder
  - Ensemble methods
  - Monte Carlo drop-out
  - **Bayesian neural networks**
  - Quantile regression
- Evaluating and calibrating uncertainty

Bayesian neural networks

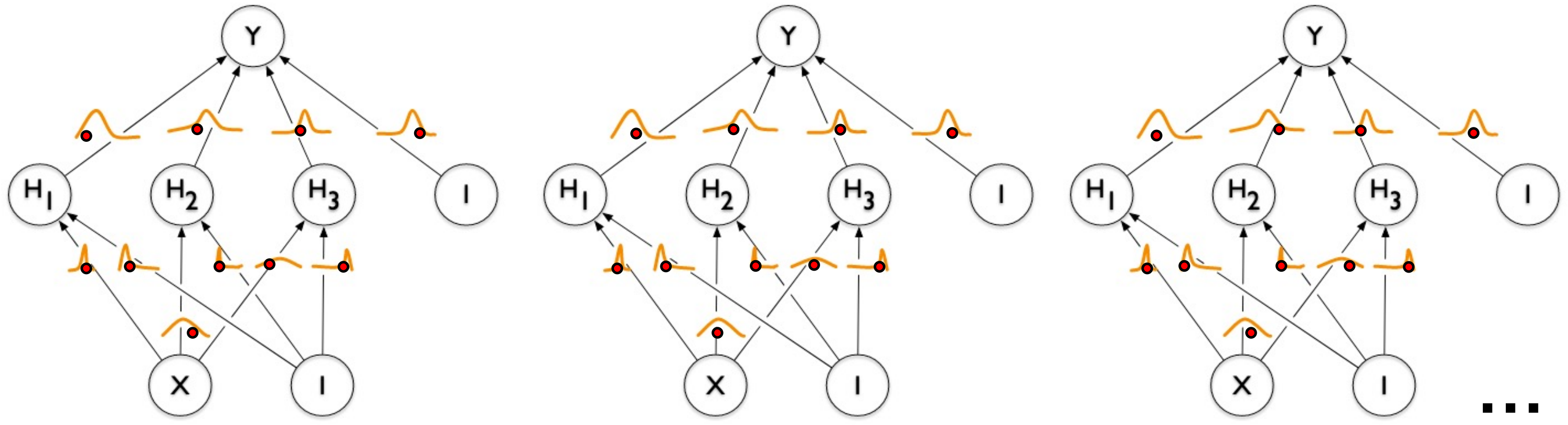
- Corresponds to a **whole family of methods**, where:
  - Weights are **randomly drawn** from a probability distribution, for each evaluation.
  - The probability distribution is **tuned** during training, according to Bayesian rules.
- As for drop-out, the **prediction** and **uncertainty** are evaluated by **averaging over repeated evaluation** of the network.
- Here we focus on one type of Bayesian neural network: "Bayes by Backprop", Blundell et al., arXiv:1505.05424 (2015)

Bayes by Backprop: inference

Regular neural network:
Weights are fixed.



Bayes by backprop:
Weights are drawn from **Gaussian distributions.**
The Gaussian distributions are fixed during inference,
but the weights change (randomly) for each evaluation.



Each weight w_i has a different Gaussian distribution, parameterized by μ_i, ρ_i:

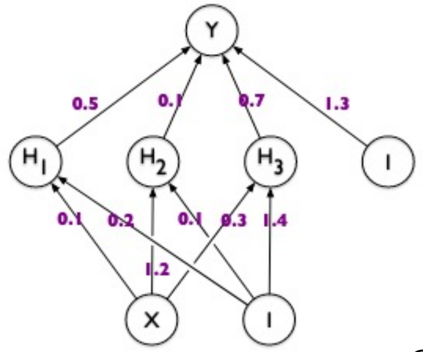
$$w_i = \mu_i + \sigma_i \epsilon_i$$

$$\epsilon_i \sim \mathcal{N}(0,1) \quad \sigma_i = \log(1 + e^{\rho_i})$$

Figures adapted from
Blundell et al., arXiv:1505.05424 (2015)

Bayes by Backprop: training

Regular neural network:
The weights themselves are updated.



$$w_i' = w_i - \alpha \frac{\partial \mathcal{L}}{\partial w_i}$$

Loss function:
Average error over the training data set

$$\mathcal{L} = \frac{1}{N} \sum_{j=1}^{N} (y_j - f_{\boldsymbol{w}}(\boldsymbol{x}_j))^2$$

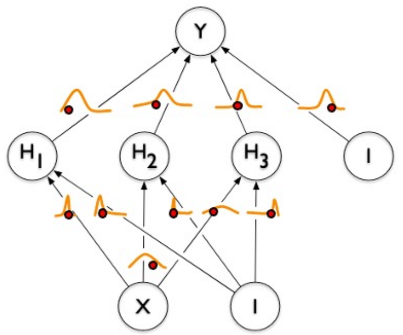
Number of examples in training set

Neural network prediction

Bayes by Backprop:
The parameters of the probability distribution (μ_i and ρ_i) are updated.

Step 1: Draw random weights

$$w_i = \mu_i + \epsilon_i \log(1 + e^{\rho_i})$$
$$\epsilon_i \sim \mathcal{N}(0,1)$$



Step 2: Update parameters

$$\mu_i' = \mu_i - \alpha \left(\frac{\partial \tilde{\mathcal{L}}}{\partial w_i} + \frac{\partial \tilde{\mathcal{L}}}{\partial \mu_i} \right)$$

$$\rho_i' = \rho_i - \alpha \left(\frac{\partial \tilde{\mathcal{L}}}{\partial w_i} \frac{\epsilon_i}{(1 + e^{-\rho_i})} + \frac{\partial \tilde{\mathcal{L}}}{\partial \rho_i} \right)$$

$$\tilde{\mathcal{L}} = \frac{1}{N} \sum_{j=1}^{N} (y_j - f_{\boldsymbol{w}}(\boldsymbol{x}_j))^2 + \frac{1}{N} \left(\sum_i \log \left(\frac{e^{\frac{(w_i - \mu_i)^2}{\sigma_i^2}}}{\sigma_i} \right) - \log(P_0(\boldsymbol{w})) \right)$$

P_0: Prior on the weights

Bayes by Backprop: ELBO loss function (“evidence lower bound”)

$$\tilde{\mathcal{L}} = \frac{1}{N}\sum_{j=1}^{N}(y_j - f_{\boldsymbol{w}}(\boldsymbol{x}_j))^2 + \frac{1}{N}\left(\sum_i \log\left(\frac{e^{\frac{(w_i-\mu_i)^2}{\sigma_i^2}}}{\sigma_i}\right) - \log(P_0(\boldsymbol{w}))\right)$$

Accuracy term:

- Depends on the training data
- Makes the neural network **fit the data**
- Amplitude stays roughly constant when increasing the number of training examples *N*

Regularization term:

- Independent of the training data
- Tends to make the Gaussian distribution of weights **similar to the prior** (Typical prior: Gaussian mixture)

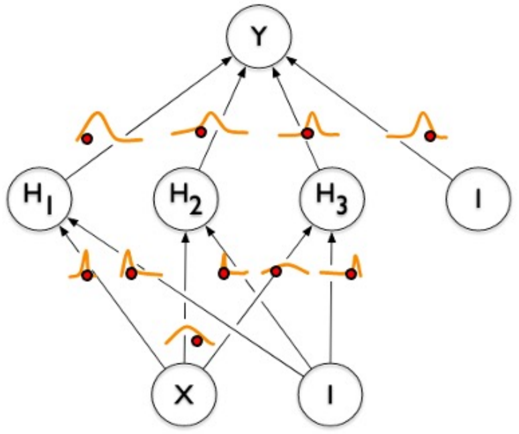
$$P_0(\boldsymbol{w}) \propto \Pi_i \left(\pi \frac{e^{-w_i^2/\sigma_1^2}}{\sigma_1} + (1-\pi)\frac{e^{-w_i^2/\sigma_2^2}}{\sigma_2}\right)$$

- Amplitude decreases when increasing the number of training examples *N*

As more training data is added (*N* increases), the Gaussian distribution on the weights **departs from the prior** and **fits the training data**.

Bayes by Backprop: summary

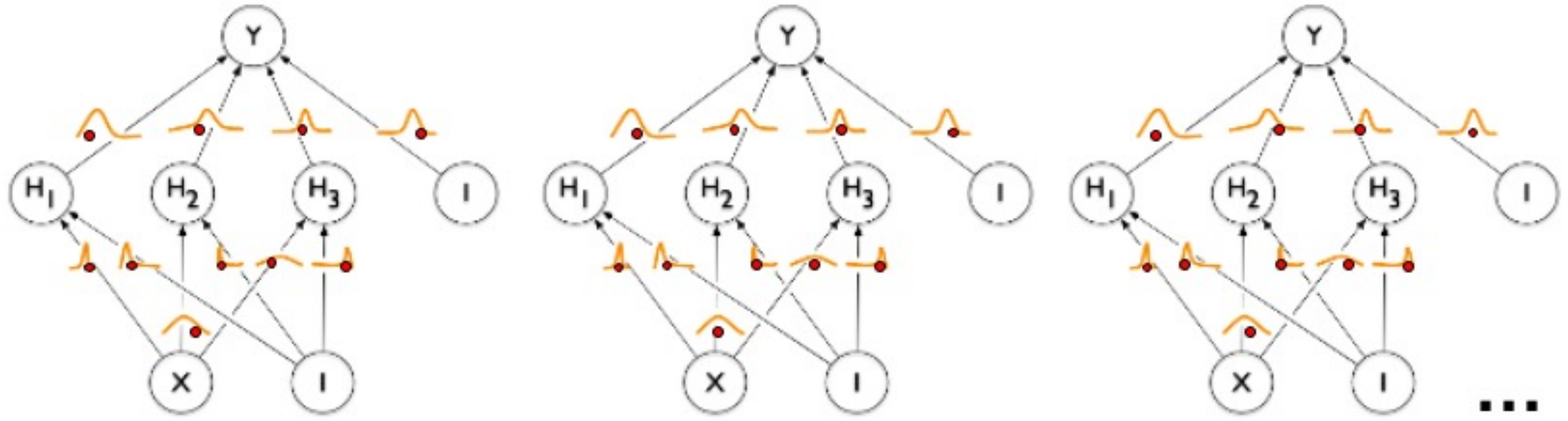
Training:
Tune the Gaussian probability distribution of the weights



$$\mu_i' = \mu_i - \alpha \left(\frac{\partial \tilde{\mathcal{L}}}{\partial w_i} + \frac{\partial \tilde{\mathcal{L}}}{\partial \mu_i} \right)$$

$$\rho_i' = \rho_i - \alpha \left(\frac{\partial \tilde{\mathcal{L}}}{\partial w_i} \frac{\epsilon_i}{(1 + e^{-\rho_i})} + \frac{\partial \tilde{\mathcal{L}}}{\partial \rho_i} \right)$$

Inference:
Draw random weights for each evaluation
Use **mean** and **standard deviation** to evaluate **prediction** and **uncertainty**



$$w_i = \mu_i + \epsilon_i \log(1 + e^{\rho_i})$$
$$\epsilon_i \sim \mathcal{N}(0, 1)$$

Bayesian neural network: practical considerations

Compared to regular NN:

- Requires 2x more parameters (μ_i, ρ_i instead of w_i)
- Added stochasticity during training due to random draw of weights
- Training is more difficult:
  e.g. much more sensitive to hyperparameters, such as the prior

Compared to Gaussian processes:

- Need to **tune** training hyperparameters (learning rate, number of epochs, etc.)
- But scales better to high dimension

Bayesian neural networks: theory

Aim: find **probability distribution** of the weights (given the training data), so that weights ***w*** can be **sampled randomly** for each evaluation

- Default assumption for probability of data, conditioned on the weights:

$$P(\{\boldsymbol{x}_i, y_i\}|\boldsymbol{w}) \propto \exp\left(-\sum_j (y_j - f_{\boldsymbol{w}}(\boldsymbol{x}_j))^2\right)$$

- The probability of the weights, conditioned on the data, can be found by Bayes theorem:

$$P(\boldsymbol{w}|\{\boldsymbol{x}_i, y_i\}) = \frac{P(\{\boldsymbol{x}_i, y_i\}|\boldsymbol{w})\ P_0(\boldsymbol{w})}{P(\{\boldsymbol{x}_i, y_i\})}$$

Prior on the weights ***w***

Prior on data (often ignored, because it does not depend on ***w***)

Bayesian neural networks: theory

Aim: find **probability distribution** of the weights (given the training data), so that weights ***w*** can be **sampled randomly** for each evaluation

- Default assumption for probability of data, conditioned on the weights:

$$P(\{\boldsymbol{x}_i, y_i\}|\boldsymbol{w}) \propto \exp\left(-\sum_j (y_j - f_{\boldsymbol{w}}(\boldsymbol{x}_j))^2\right)$$

- The probability of the weights, conditioned on the data, can be found by Bayes theorem:

$$P(\boldsymbol{w}|\{\boldsymbol{x}_i, y_i\}) \propto P_0(\boldsymbol{w}) \exp\left(-\sum_j (y_j - f_{\boldsymbol{w}}(\boldsymbol{x}_j))^2\right)$$

- **Problem:** Difficult to randomly sample weights ***w*** from this probability distribution, (due to the complex dependency on ***w*** through the neural network function $f_{\boldsymbol{w}}$)

Bayesian neural networks: theory

- $\boldsymbol{w}$ cannot be sampled from the true probability distribution

$$P(\boldsymbol{w}|\{\boldsymbol{x}_i, y_i\}) \propto P_0(\boldsymbol{w}) \exp\left(-\sum_j (y_j - f_{\boldsymbol{w}}(\boldsymbol{x}_j))^2\right)$$

- $\boldsymbol{w}$ is **instead** sampled from a simpler, **approximate probability** distribution $q(\boldsymbol{w}, \boldsymbol{\theta})$, that depends on hyperparameters $\boldsymbol{\theta}$

  e.g. "Bayes by backprop": $$q(\boldsymbol{w}, \boldsymbol{\theta}) = \Pi_j \frac{1}{\sqrt{2\pi}\log(1+e^{\rho_j})} \exp\left(-\frac{(w_j - \mu_j)^2}{2\log(1+e^{\rho_j})^2}\right)$$

$$\boldsymbol{\theta} = \{\mu_j, \rho_j\}$$

  Other Bayesian networks can be obtained by changing $q(\boldsymbol{w}, \boldsymbol{\theta})$ e.g. "concrete dropout"

- The hyperparameters $\boldsymbol{\theta}$ are tuned so that $q(\boldsymbol{w}, \boldsymbol{\theta})$ becomes **as close as possible** to the true probability distribution $P(\boldsymbol{w}|\{\boldsymbol{x}_i, y_i\})$ ("**variational approximation**")

Bayesian neural networks: theory

- **“as close as possible”:** tune $\boldsymbol{\theta}$ to minimize the Kullback-Leibler divergence between the **true distribution** *P* and the **approximate distribution** *q*

$$KL(q||P) = \left\langle \log\left(\frac{q(\boldsymbol{w}|\boldsymbol{\theta})}{P(\boldsymbol{w}|\{y_j,\boldsymbol{x}_j\})}\right)\right\rangle_{\boldsymbol{w}\sim q(\boldsymbol{w}|\boldsymbol{\theta})}$$

$$P(\boldsymbol{w}|\{\boldsymbol{x}_i, y_i\}) \propto P_0(\boldsymbol{w}) \exp\left(-\sum_j (y_j - f_{\boldsymbol{w}}(\boldsymbol{x}_j))^2\right)$$

$$= \left\langle \underbrace{\sum_j (y_j - f_{\boldsymbol{w}}(\boldsymbol{x}_j))^2}_{\textbf{Accuracy term}} + \underbrace{\log(q(\boldsymbol{w}|\boldsymbol{\theta})) - \log(P_0(\boldsymbol{w}))}_{\textbf{Regularization term}} \right\rangle_{\boldsymbol{w}\sim q(\boldsymbol{w}|\boldsymbol{\theta})}$$

Corresponds to the modified loss function $\tilde{\mathcal{L}}$ mentioned earlier.

Outline

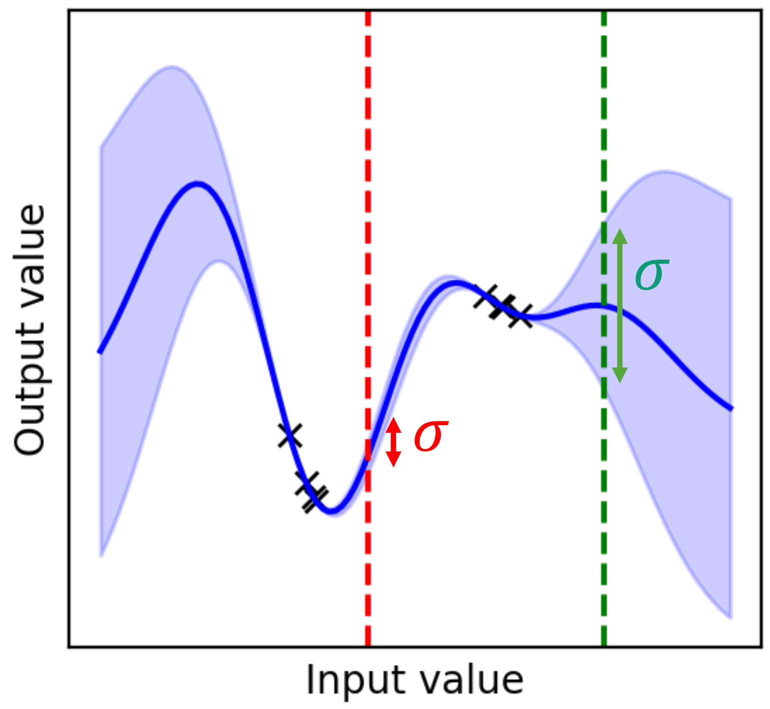
- Uncertainty in ML: definition and motivation

- Methods to estimate uncertainty
  - Gaussian processes: reminder
  - Ensemble methods
  - Monte Carlo drop-out
  - Bayesian neural networks
  - **Quantile regression**

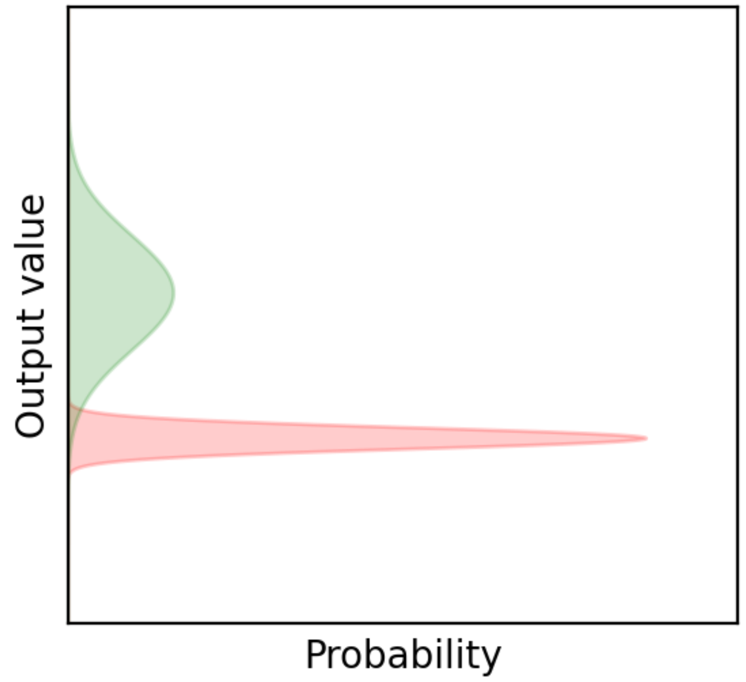
- Evaluating and calibrating uncertainty

How to obtain the probability distribution

Standard deviation
(Single scalar)

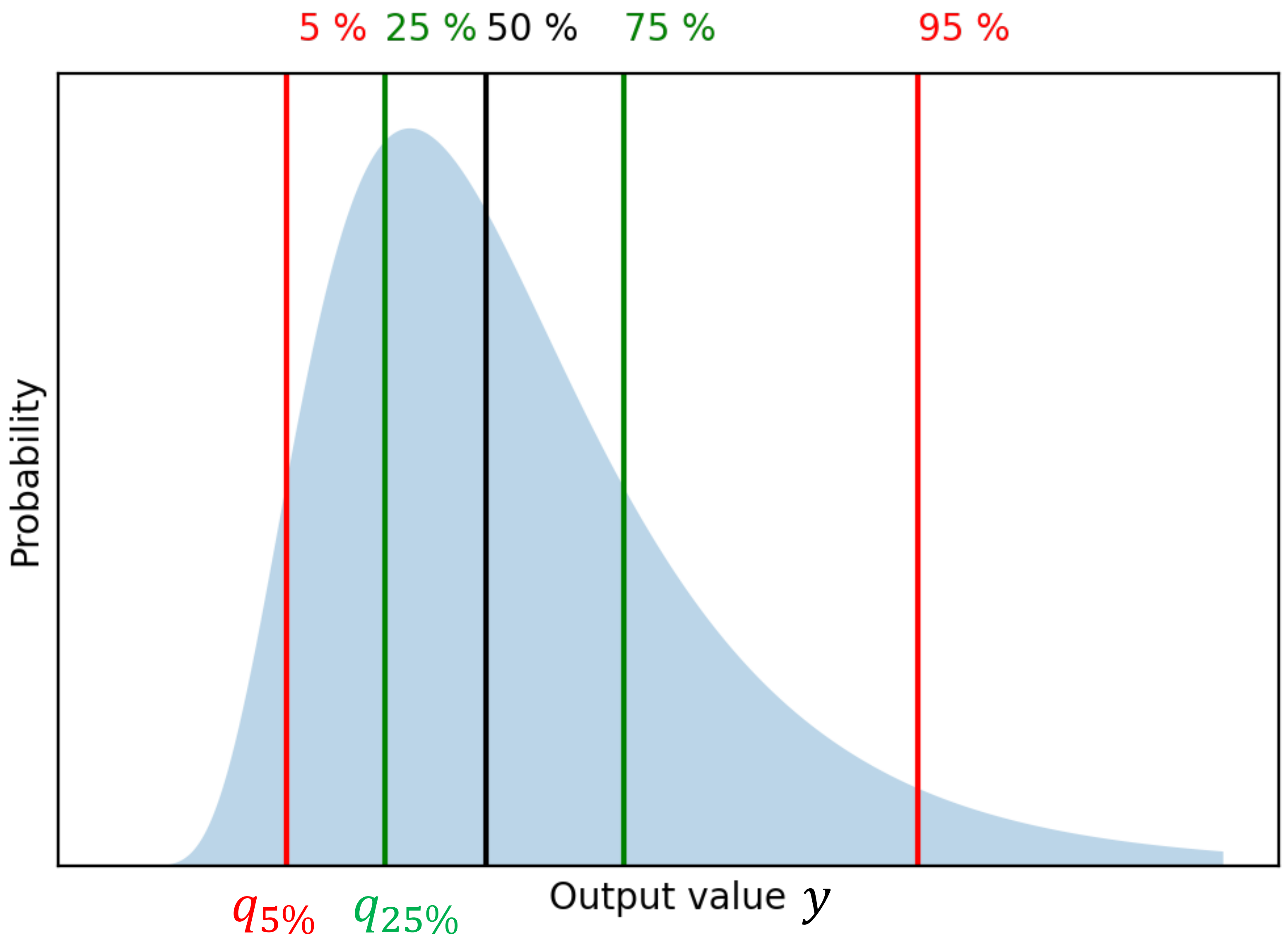


Probability distribution
(Full function)



- The methods seen so far (ensembles, MC dropout, Bayesian NN) only provide the standard deviation.
- By default, often assume that the corresponding distribution is Gaussian.
- What if the distribution of the data (e.g. noise) is significantly non-Gaussian?

Quantiles: a way to describe the probability distribution



Quantile definition:

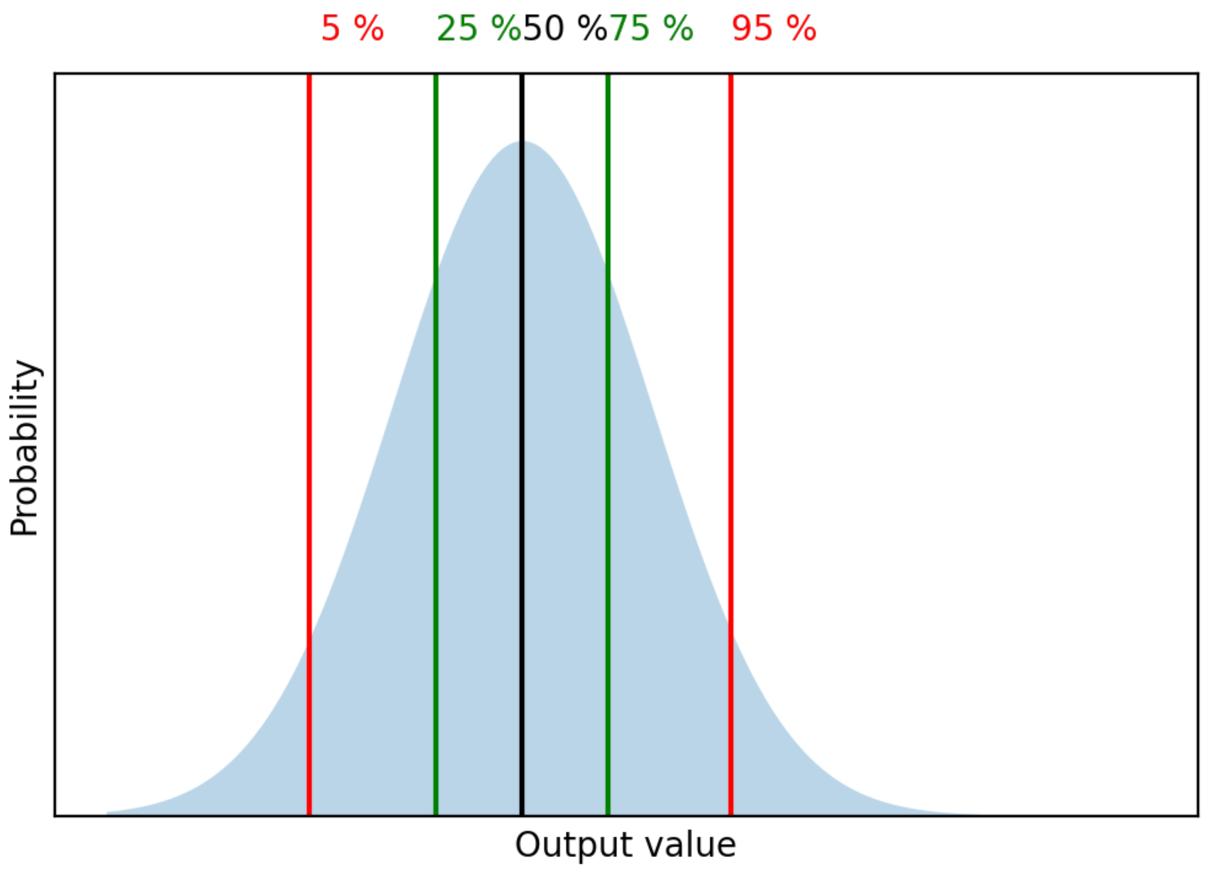
Value q_τ such that a fraction τ of the values y are **below** q_τ

In terms of probability:

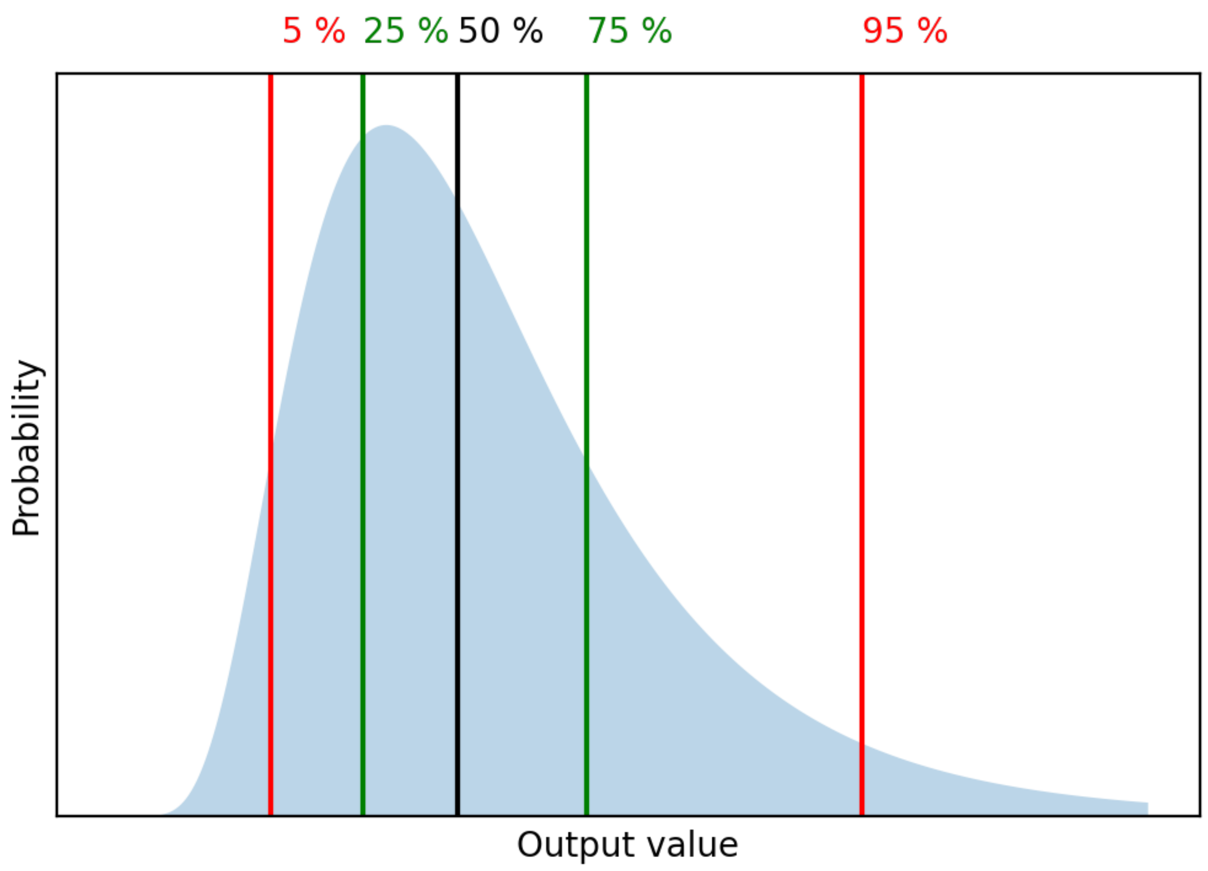
$$P(y \leq q_\tau) = \tau$$

Quantiles allow to capture non-Gaussian distributions

Gaussian

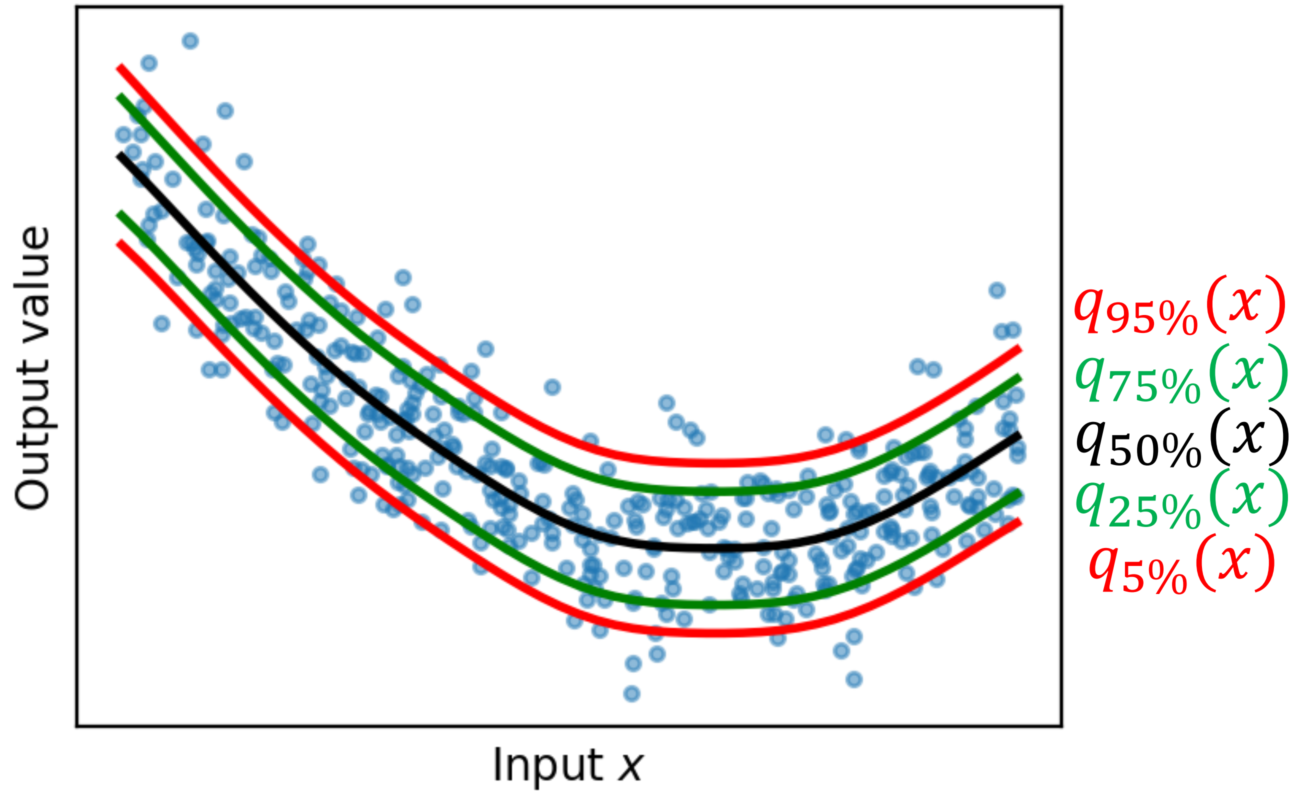


Log-normal (non-Gaussian)



Conditional quantiles

We would like an ML model that can predict the position of the quantiles as a function of the input ***x***.



Conditional quantile definition:

Value $q_\tau(x)$ such that a fraction τ of the output values y **corresponding to a given intput $\boldsymbol{x}$** are below q_τ.

In terms of conditional probability:

$$P(y \leq q_\tau | x) = \tau$$

Advantage: quantitative error bars that take into account non-Gaussian noise.

Quantile regression: loss function

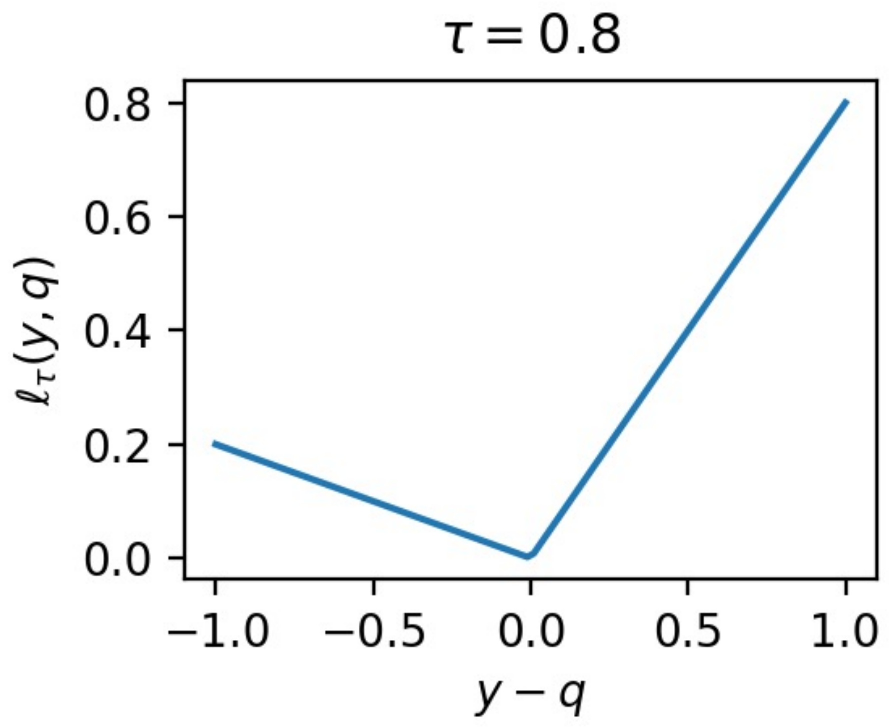
The quantile q_τ can **alternatively** be defined as the minimum of a specific loss function ("pinball loss"):

$$\mathcal{L}(q) = \langle \ell_\tau(y, q) \rangle$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \ell_\tau(y_i, q)$$

Sum over evaluated data points

$$\ell_\tau(y, q) = \begin{cases} (1-\tau)(q-y) & \text{if } y \leq q \\ \tau(y-q) & \text{if } y > q \end{cases}$$



Note: $\ell_{0.5} = 0.5|y - q|$

“Demonstration” of the equivalence between the different definitions

- The loss function can be written as:

$$\mathcal{L}(q) = \langle \ell_\tau(y,q) \rangle \equiv \int_{-\infty}^{\infty} \mathrm{d}y \; p(y) \ell_\tau(y,q)$$

$$= \int_{-\infty}^{q} \mathrm{d}y \, p(y)(1-\tau)(q-y) + \int_{q}^{+\infty} \mathrm{d}y \, p(y) \tau (y-q)$$

- The minimum q_τ satisfies $\frac{\partial \mathcal{L}}{\partial q}(q_\tau) = 0$

$$\int_{-\infty}^{q_\tau} \mathrm{d}y \; p(y)(1-\tau) + \int_{q_\tau}^{+\infty} \mathrm{d}y \; p(y)\tau(-1) = 0$$

$$\int_{-\infty}^{q_\tau} \mathrm{d}y \; p(y) = \tau \left(\int_{-\infty}^{q_\tau} \mathrm{d}y \; p(y) + \int_{q_\tau}^{+\infty} \mathrm{d}y \; p(y) \right)$$

$$P(y \leq q_\tau) = \tau \int_{-\infty}^{+\infty} \mathrm{d}y \; p(y) = \tau$$

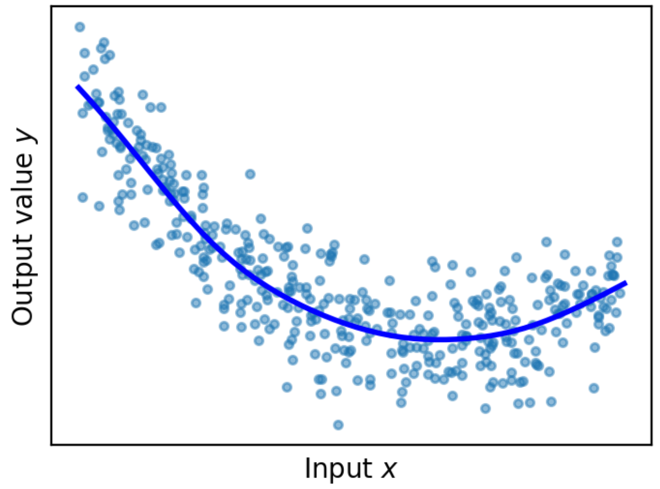
Training quantile regression neural networks

Standard neural network

Train by minimizing the loss function

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2$$

After training, the prediction of the neural network $f(x)$ corresponds to the **average of the data** at point *x*.



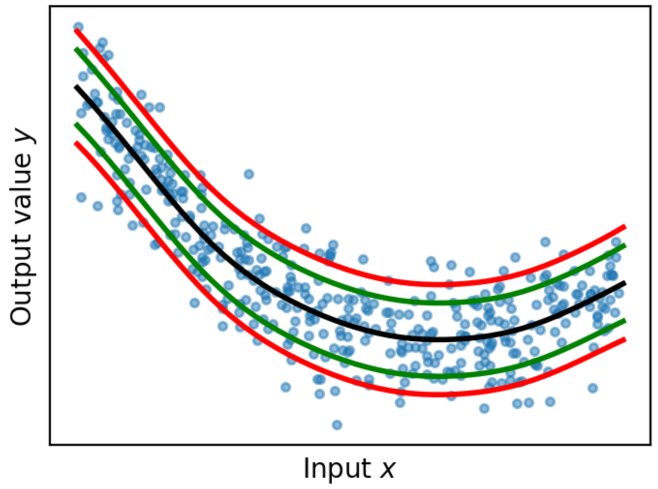
Quantile regression neural network

Train by minimizing the loss function

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\ell_\tau(y_i, f(x_i))$$

for a given τ.

After training, the prediction of the neural network $f(x)$ corresponds to the **τ-quantile** at point *x*.
(Use a separate neural network for each τ.)



FEL example

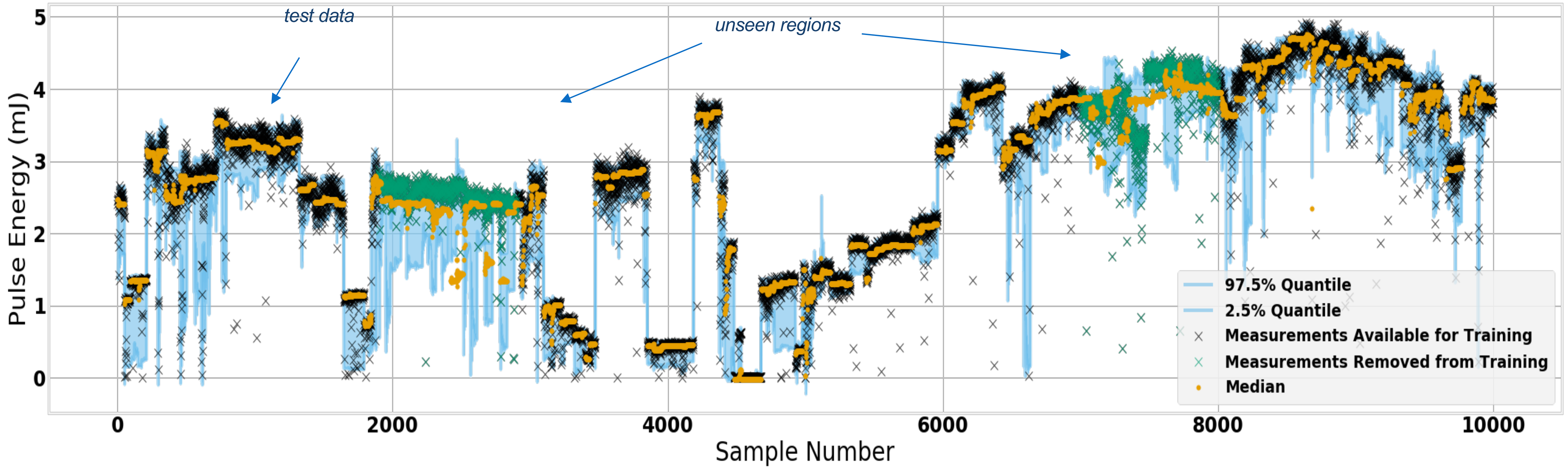
Input:

70+ quantities, incl:

- Strength of quadrupole and steering magnets
- Linac phases and amplitudes
- Laser properties in photo-injector
- Undulator properties

Output:

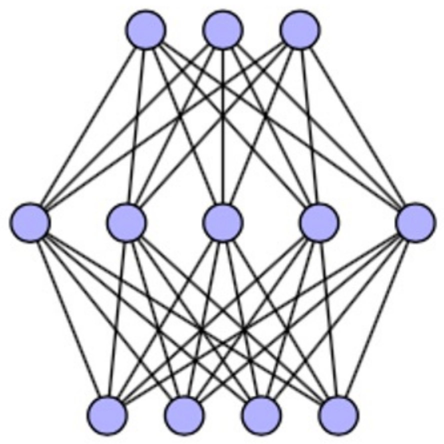
FEL pulse energy



Generalization to multi-dimensional output

Quantile regression neural network easily generalize to high-dimensional output: sum over dimensions in cost function.

$y^{(1)}$ $y^{(2)}$ $y^{(3)}$



e.g. beam size at different locations

Input x

e.g. accelerator parameters

$$\mathcal{L} = \sum_j \frac{1}{N} \sum_{i=1}^{N} \ell_\tau(y_i^{(j)}, f^{(j)}(x_i))$$

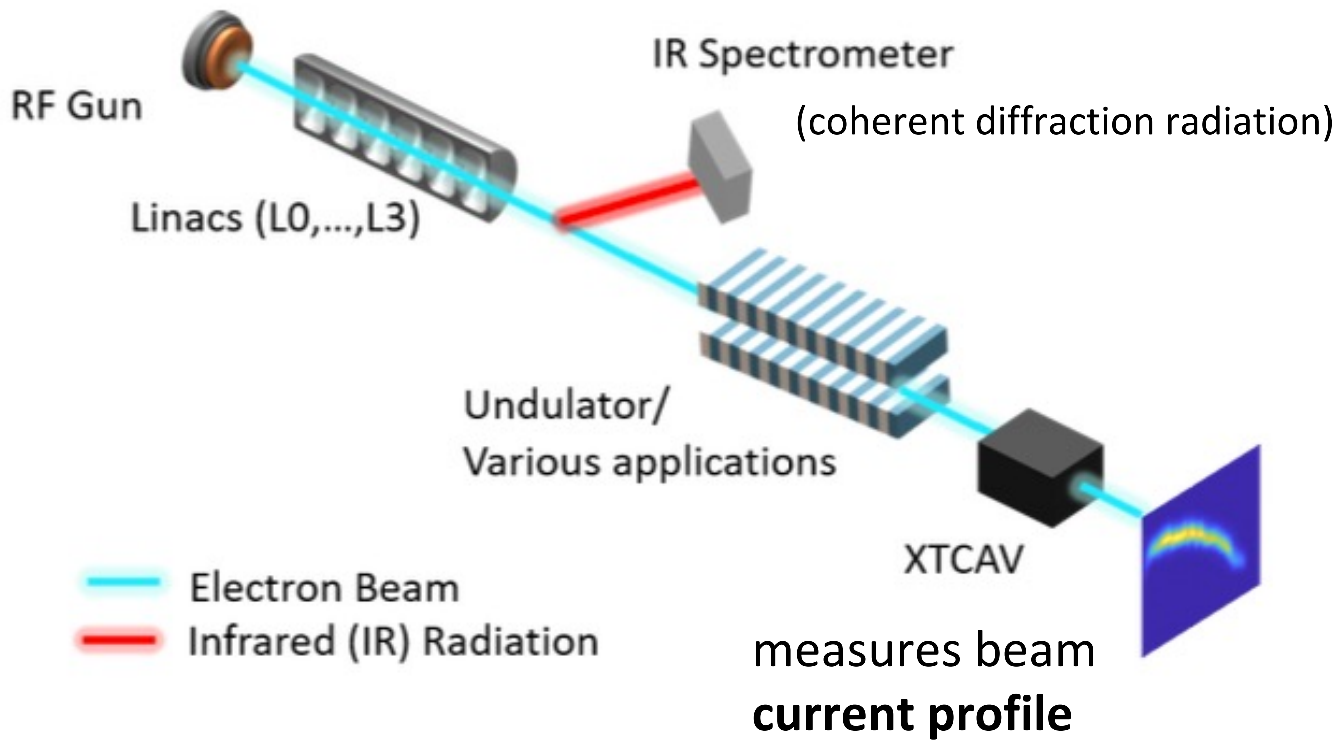
Sum over dimensions of the output

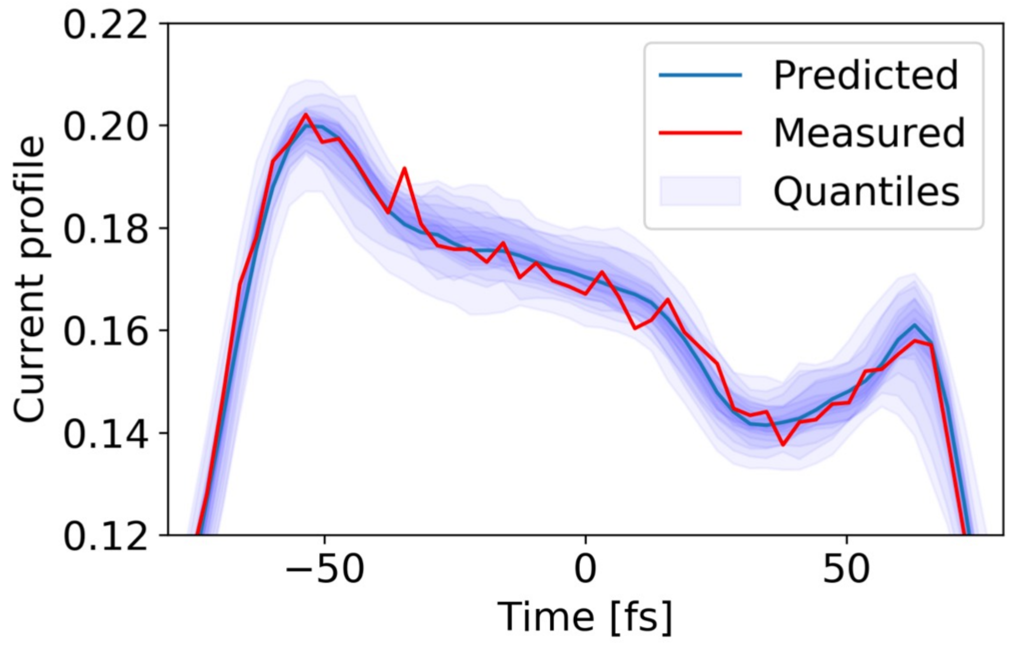
Sum over data points

After training, $f^{(j)}(x)$ corresponds to the τ-**quantile** for $y^{(j)}$ at point *x*.

Example: uncertainty on virtual diagnostics for beam current

O. Convery et al., arXiv:2105.04654v1 (2021)





Neural networks for 19 quantiles (0.05 to 0.95)

- Input $\boldsymbol{x}$: full IR spectrum
- Output $y^{(j)}$: 1d beam current profile

Trained on ~3,000 shots

Outline

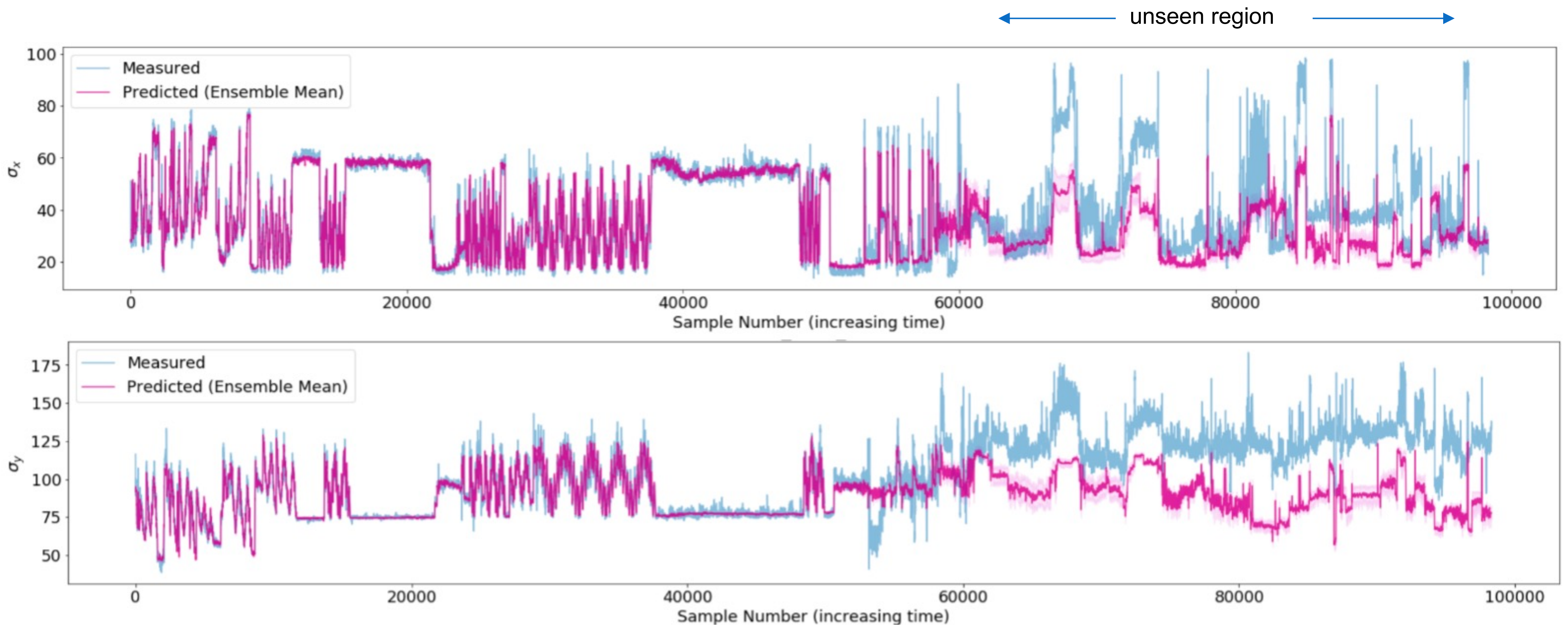
- Uncertainty in ML: definition and motivation

- Methods to estimate uncertainty
  - Gaussian processes: reminder
  - Ensemble methods
  - Monte Carlo drop-out
  - Bayesian neural networks
  - Quantile regression

- **Evaluating and calibrating uncertainty**

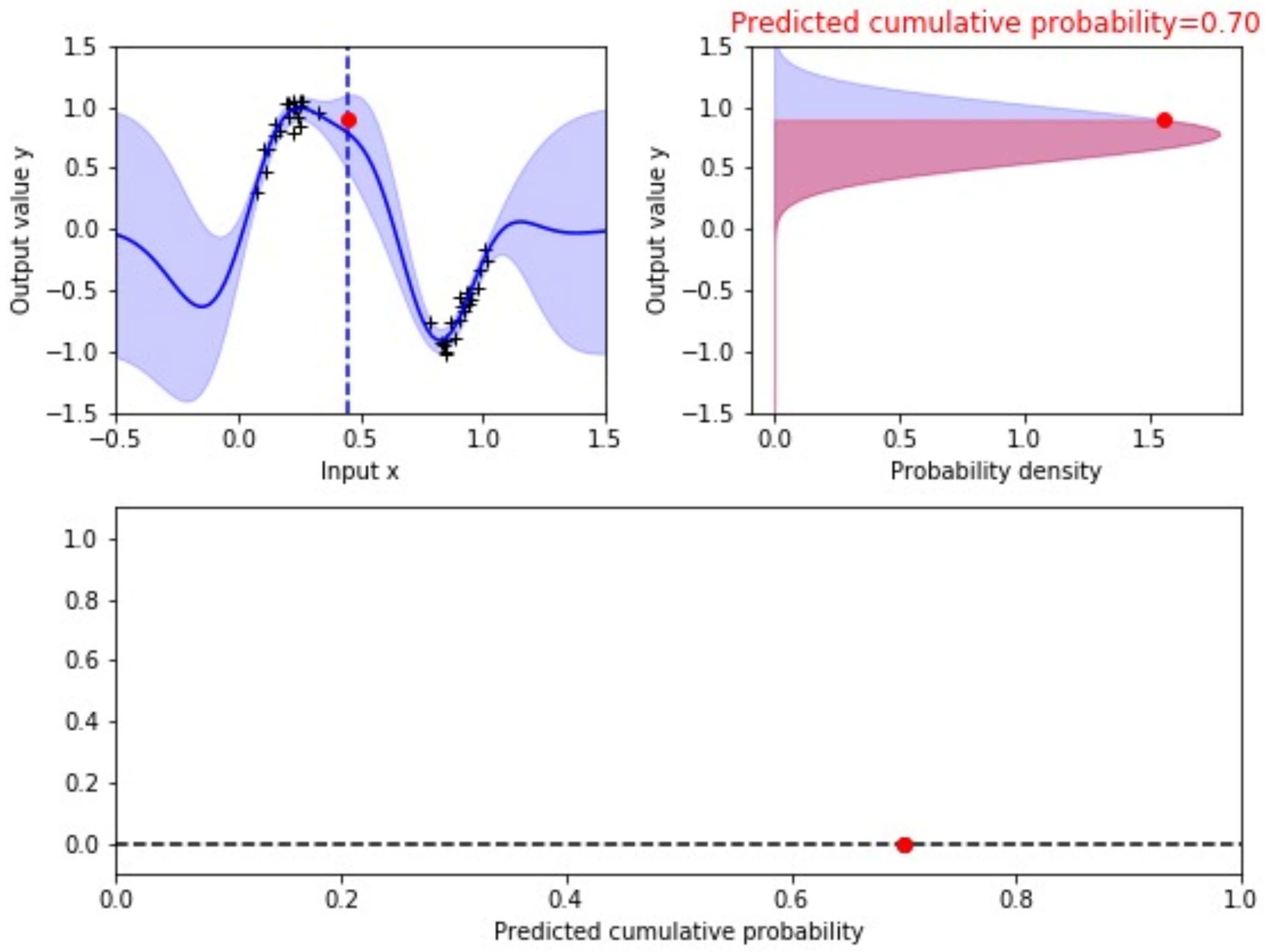
Validating uncertainty

Uncertainty estimate (and confidence intervals) are not always **quantitively** accurate.



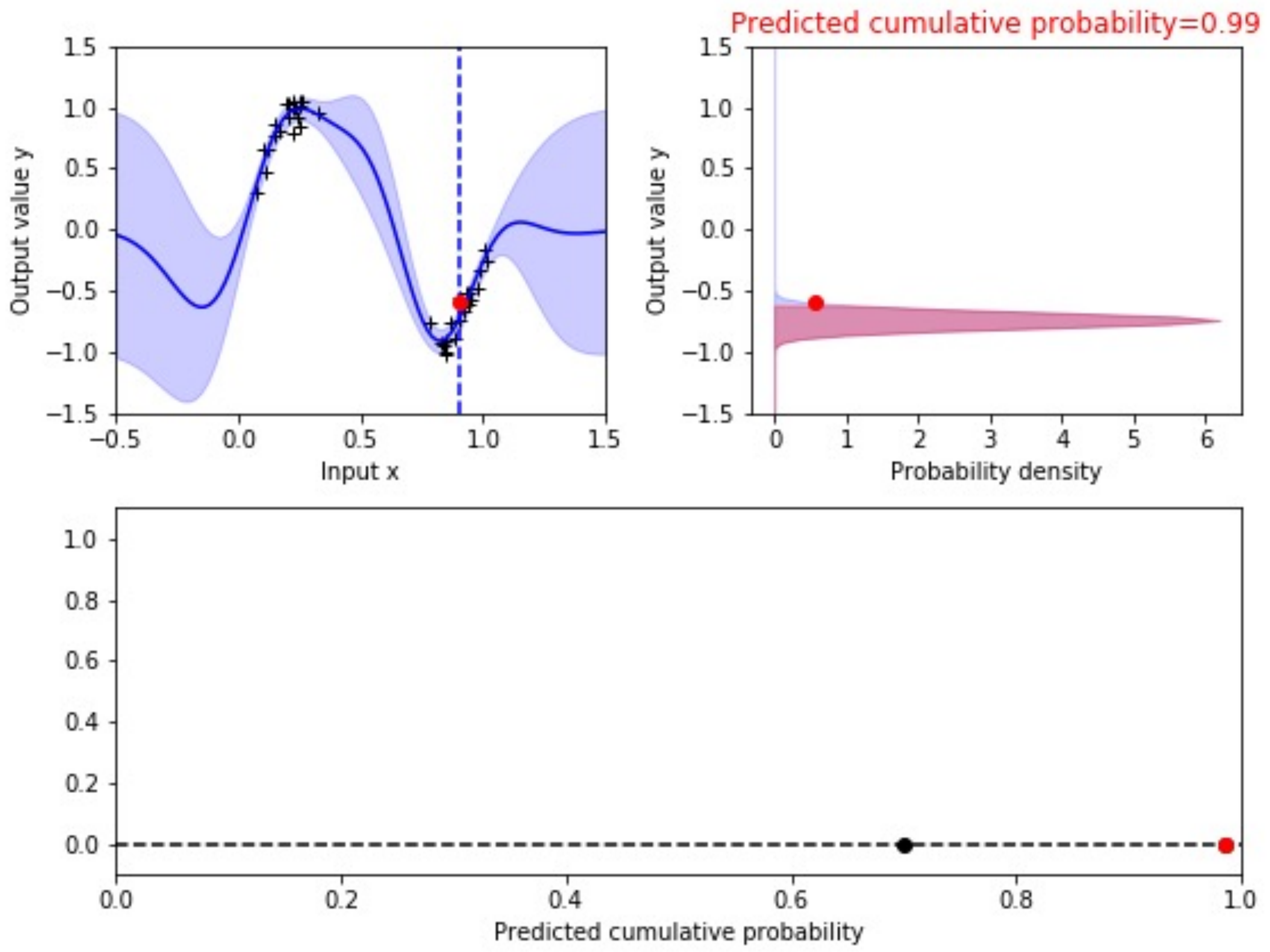
Calibration curve

- Use test data
  (unseen during training)
- For each point in the test data:
  Record the **predicted cumulative probability** of the data point, as predicted by the ML model



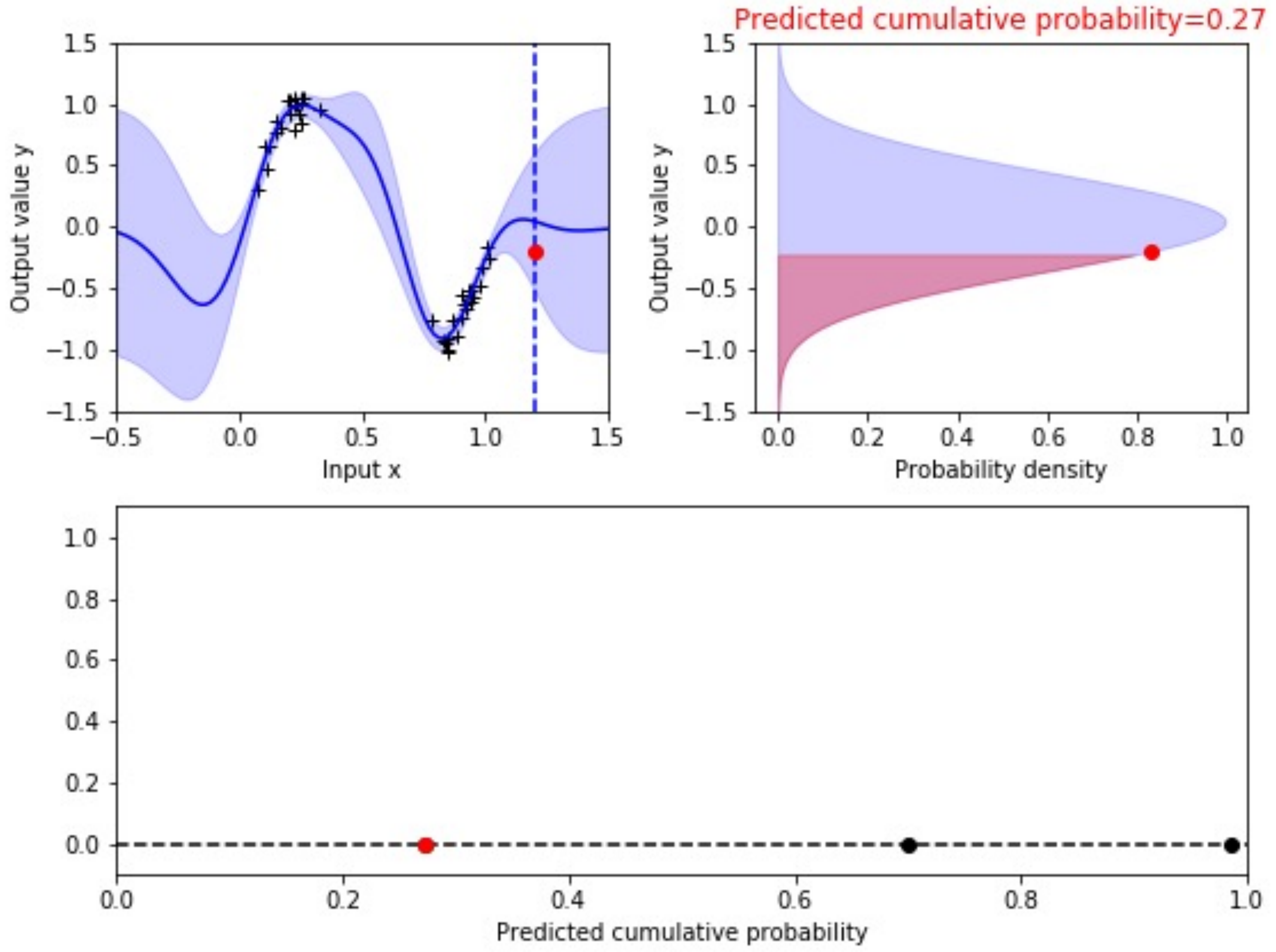
Calibration curve

- Use test data
  (unseen during training)
- For each point in the test data:
  Record the **predicted cumulative probability** of the data point, as predicted by the ML model



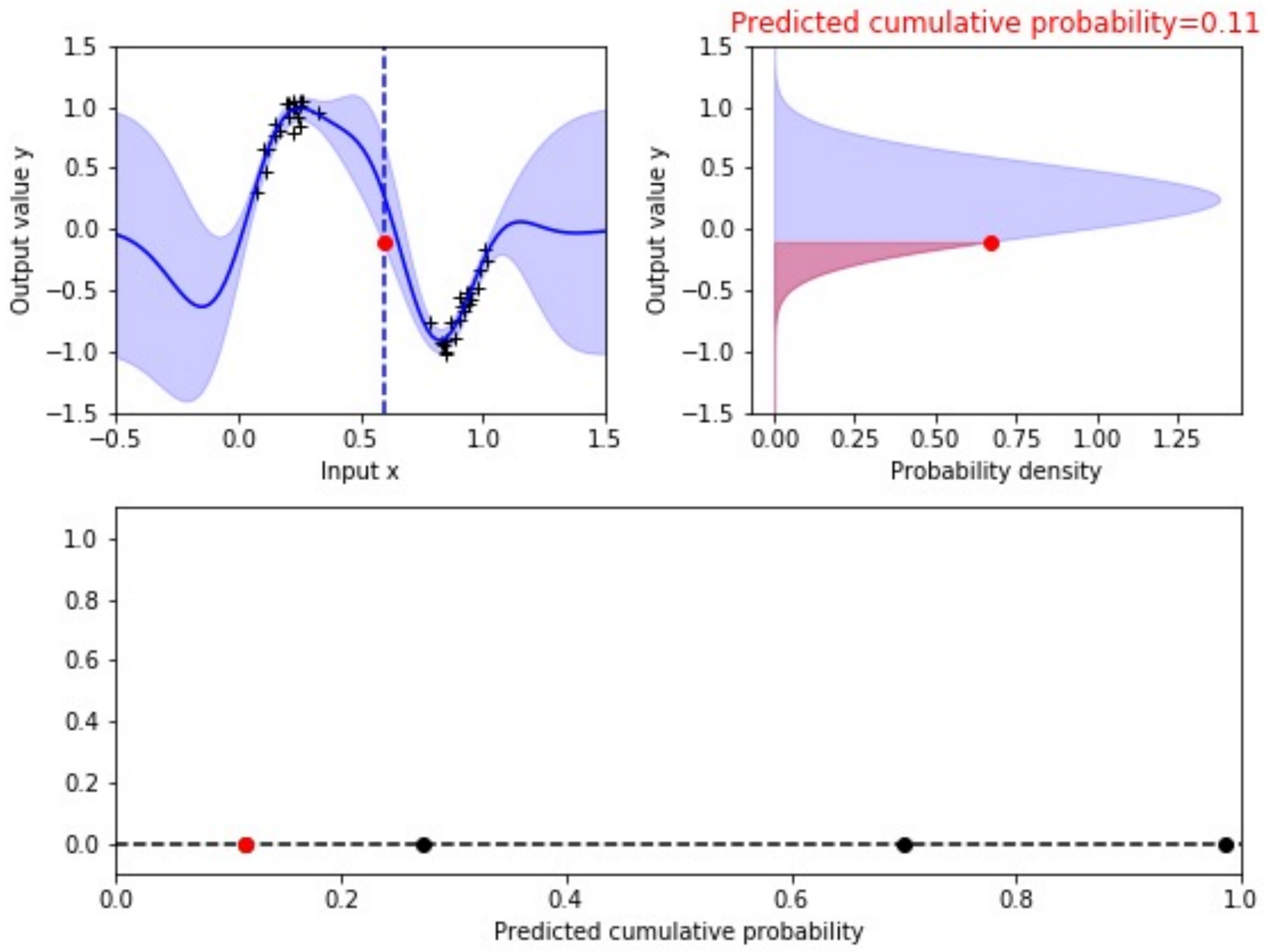
Calibration curve

- Use test data (unseen during training)
- For each point in the test data: Record the **predicted cumulative probability** of the data point, as predicted by the ML model



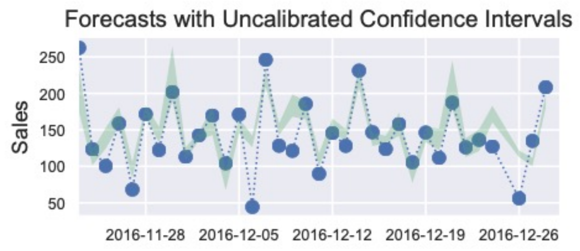
Calibration curve

- Use test data (unseen during training)
- For each point in the test data: Record the **predicted cumulative probability** of the data point, as predicted by the ML model

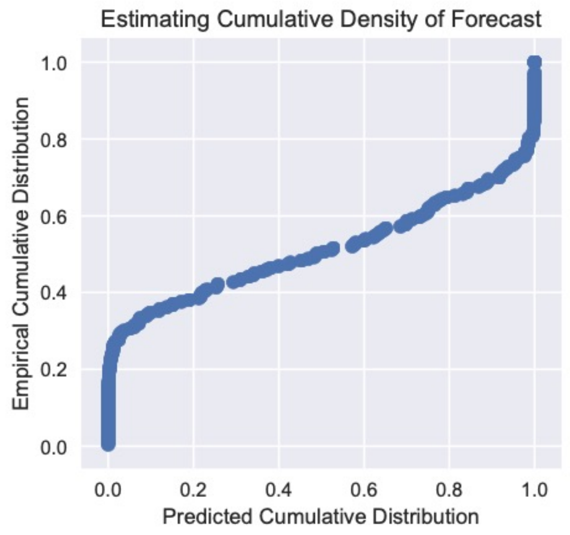


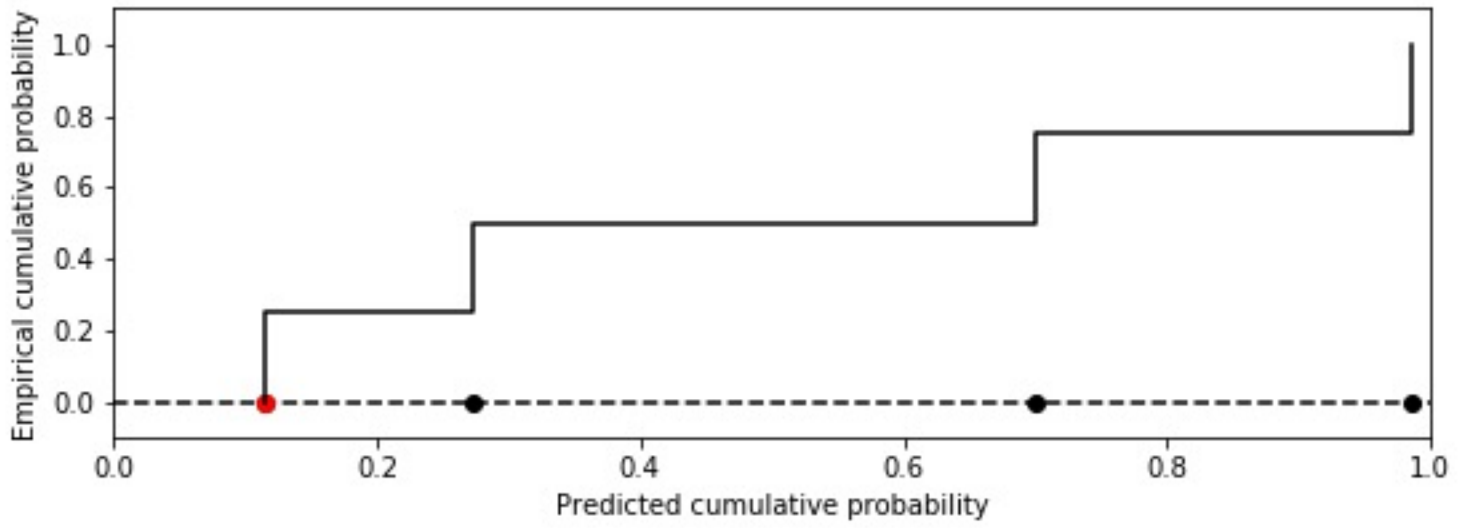
Calibration curve

- Use test data
  (unseen during training)
- For each point in the test data:
  Record the **predicted cumulative probability** of the data point, as predicted by the ML model
- Plot the corresponding **empirical cumulative probability**
- For a large number of points:
  this should tend towards a **straight line** if the model is **well calibrated.**



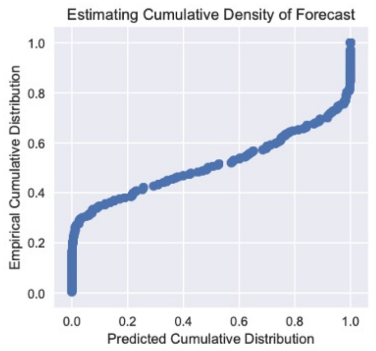
Kuleshov et al., “Accurate Uncertainties for Deep Learning Using Calibrated Regression”, 2018



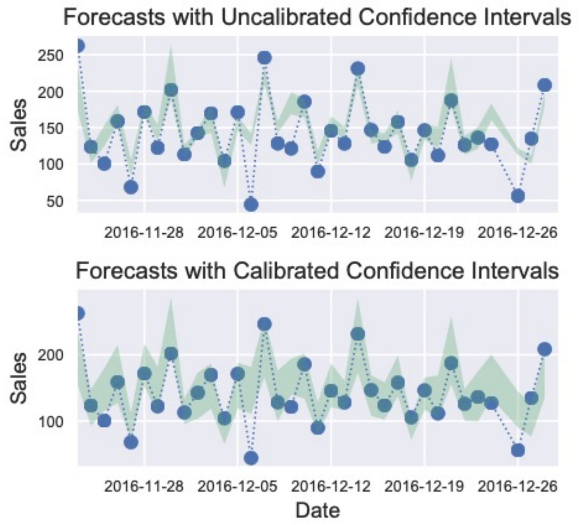


Recalibration: correct the predicted cumulative probability

Useful when **quantitative estimates** of the uncertainty are important.



Input **x** $\rightarrow$ ML model $\rightarrow$ Predicted cumulative distribution $P(y|\boldsymbol{x})$ $\rightarrow$ Fit of the calibration curve $f\,(P)$ $\rightarrow$ Corrected cumulative distribution $f\;(\;P(y|\boldsymbol{x})\;)$



Kuleshov et al., "Accurate Uncertainties for Deep Learning Using Calibrated Regression", 2018

Questions?

- Uncertainty in ML: definition and motivation
- Methods to estimate uncertainty
    - Gaussian processes: reminder
    - Ensemble methods
    - Monte Carlo drop-out
    - Bayesian neural networks
    - Quantile regression
- Evaluating and calibrating uncertainty