Machine Intelligence:: Deep Learning
Week 8

Beate Sick, Jonas Brändli, Oliver Dürr

Bayesian approaches for improving the performance and uncertainty estimates of NN models by taking into account the epistemic uncertainty.

Outline:

- Bayesian Modelling
  - Introduction to Bayesian Statistics
  - A simple example (Bayes the Hacker’s way)
  - Variational Approximation

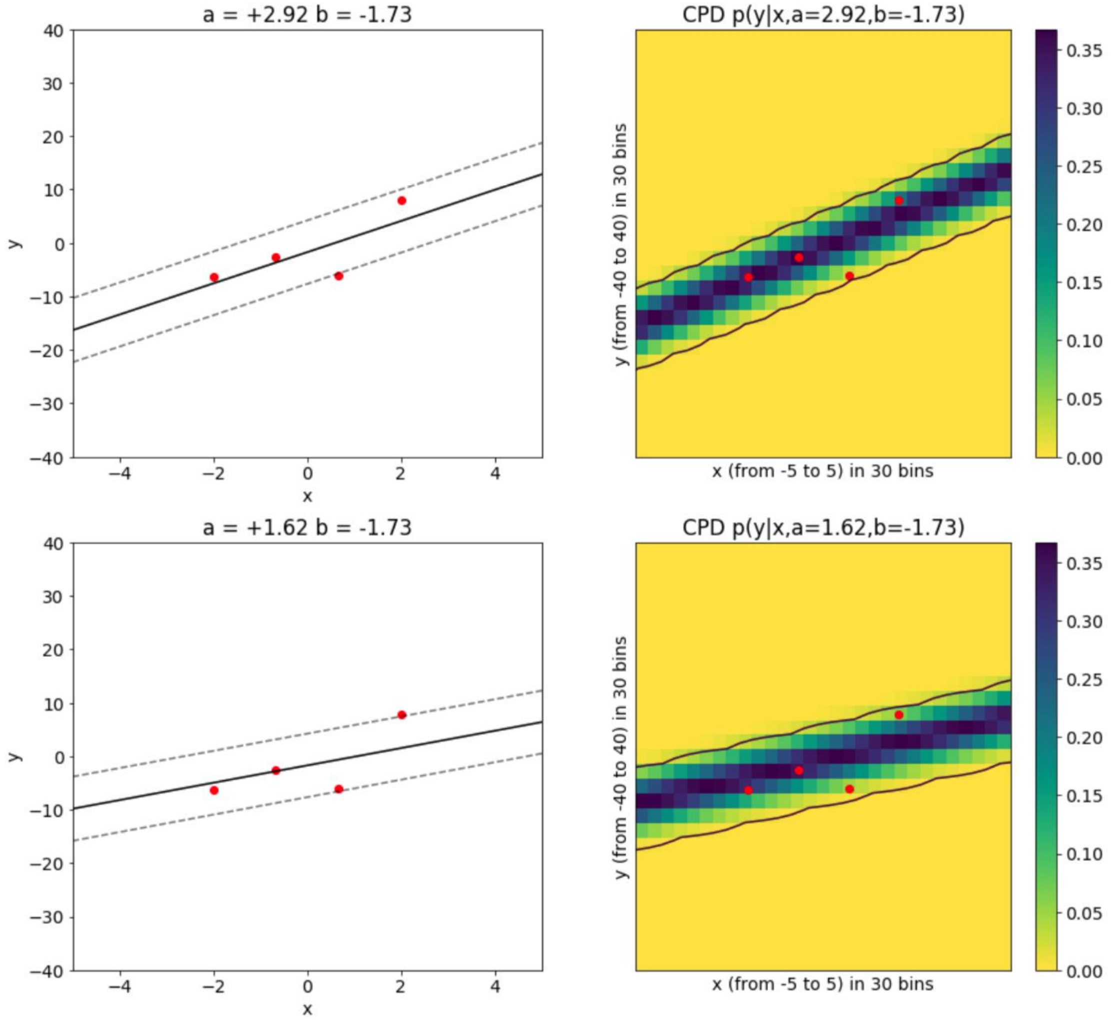
Bayesian Deep Learning

→ Improving prediction performance

→ Providing epistemic uncertainty measures

Bayes the hackers' way

Let's look at good old linear regression to understand the gist of the Bayes idea. Assume $\sigma = 3$ to be known.

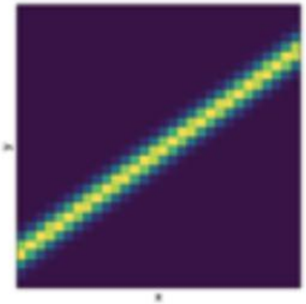
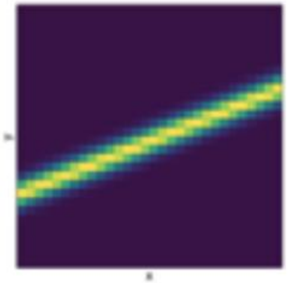
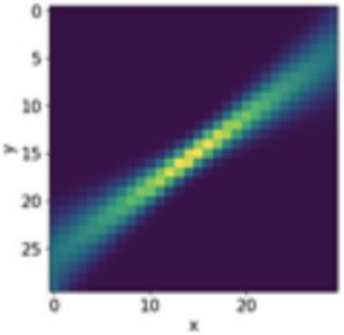


MaxLike Solution

A bit off the MaxLike Solution

Combining different fits

Also take the other fits with different parameters into account and weight them

0.09  + 0.001  + …= 

$p(y|x,D)$

Hightest weight for MaxLike

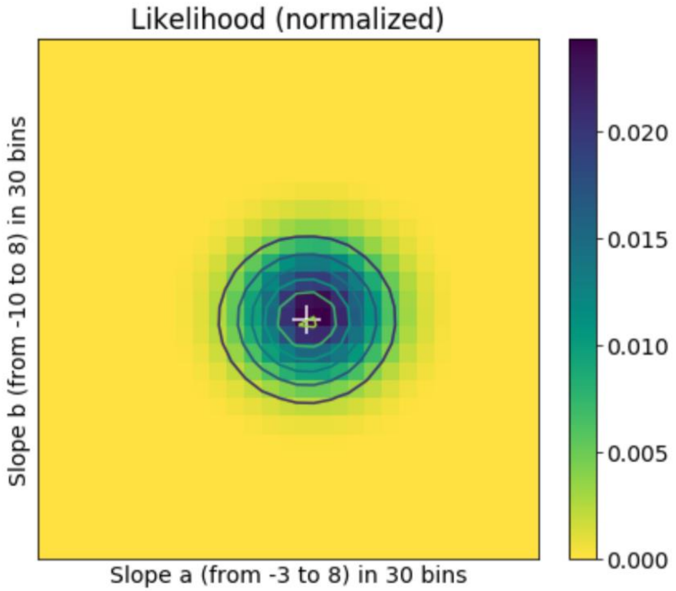
Less likely solutions get
a smaller weight

Question: How to get the weight?

Idea: use the (normalized) likelihood $p_{\mathrm{norm}}(D|(a,b))$!

Don't put all egg's in one Basket

- Also take other solutions for parameters a, b into account



$$p_{\text{norm}}(D|(a,b)) = \frac{p(D|(a,b))}{\sum_w p(D|(a,b))}$$

$$\text{p}(y|x,D) = \sum_a \sum_b p(y|x,(a,b)) \cdot p_{\text{norm}}(D|(a,b))$$

Likelihood at 30x30 different positions of a and b. Normalized to be one.
https://github.com/tensorchiefs/ dl_book/blob/master/chapter_07 /nb_ch07_02.ipynb

Result

$$\mathrm{p}(y|x,D) = \sum_{a}\sum_{b} p(y|x,(a,b)) \cdot p_{\mathrm{norm}}(D|(a,b))$$

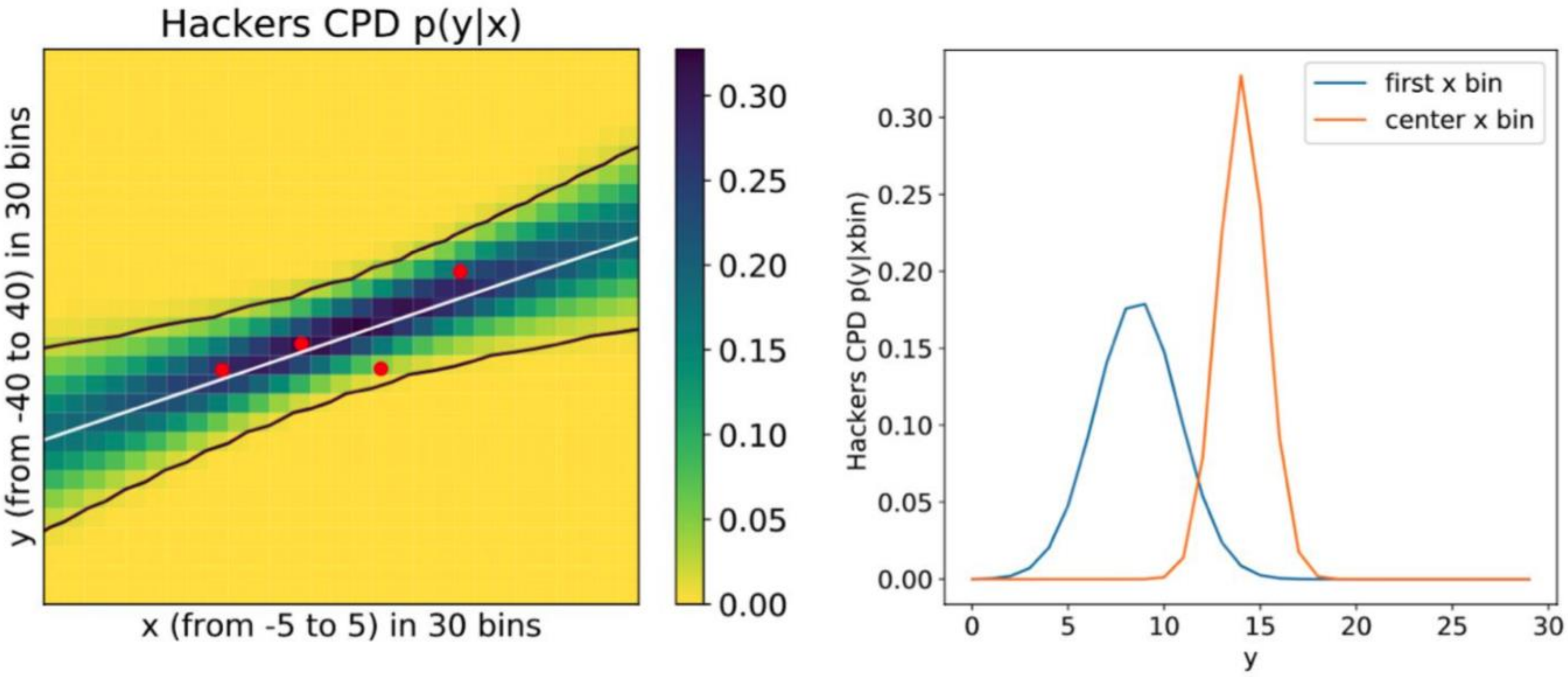


Figure 7.6 The predictive distribution for the Bayesian linear regression model, trained with the four data points shown on the left side by the color-code and on the right as conditional distribution at two different *x* positions. You can clearly see that the uncertainty gets larger when leaving the *x*-regions where there's data.

Bayesian statistics

- The Bayesian Theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

- Applied to $A = \theta$ and $B = D$
  - Parameters θ of a model e.g. weights *w* of NN
  - Training data *D*

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\sum_\theta p(D|\theta)p(\theta)} \sim p(D|\theta)p(\theta)$$

Revisiting the Hackers' example

- Hacker's way
  - $\mathrm{p}(y|x,D) = \sum p(y|x,w) \cdot p_{\mathrm{norm}}(D|w)$
    - $p_{\mathrm{norm}}(D|w) = \frac{p(D|w)}{\sum_w p(D|w)}$
- Bayes
  - $\mathrm{p}(y|x,D) = \sum_w p(y|x,w) \cdot p_{\mathrm{norm}}(w|D)\ )$
  - $p(w|D) = \frac{p(D|w)p(w)}{p(D)} = \frac{p(D|w)p(w)}{\sum_w p(D|w)p(w)}$

→ The Hacker's way is Bayes, if we assume a uniform prior $p(w) = \mathrm{const}$

Bayesian modeling has less problems with complex models

Frequentist's strategy:
You can only use a complex model if you have enough data!

Bayesian's strategy:

Use the model complexity you believe in.

Do not just use the best fitting model.

Do use the full posterior distribution over parameter settings leading to vague predictions since many different parameter settings have significant posterior probability.

$$\mathrm{p}(y|x^*, \mathrm{traindata}) = \int p(y|x^*, \theta) \cdot p_{\mathrm{norm}}(\theta|\mathrm{traindata})\ d\theta$$

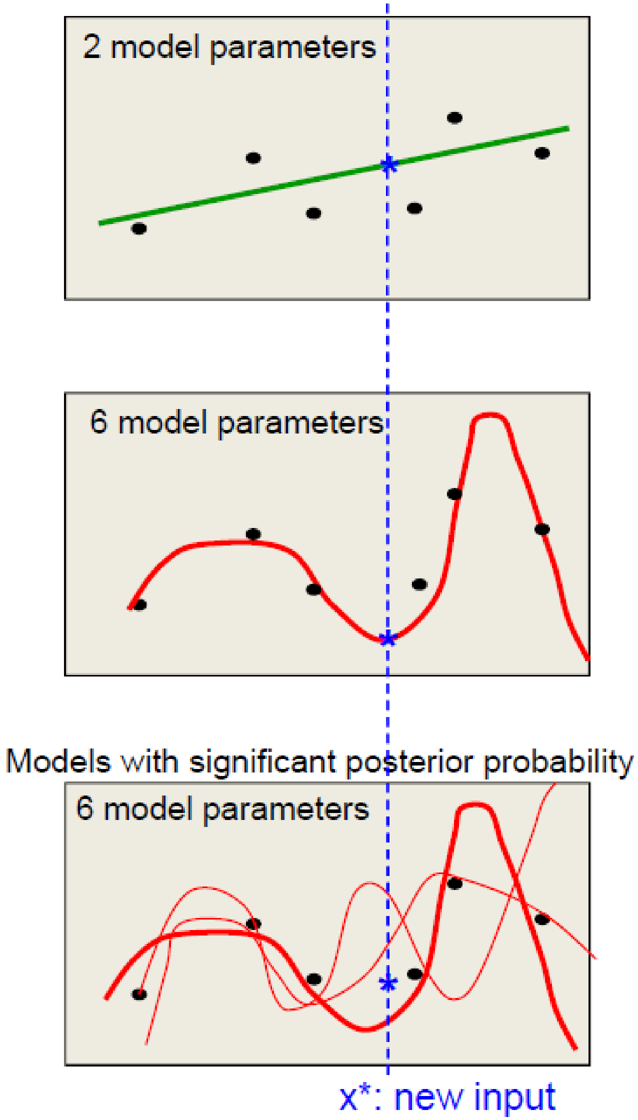


Image credits: Hinton coursera course

Bayesian terminology

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\sum_\theta p(D|\theta)p(\theta)} \sim p(D|\theta)p(\theta)$$

– $p(\theta|D)$	posterior	see next slide
– $p(D|\theta)$	likelihood	our good old friend
– $p(\theta)$	prior	see next slide
– $p(D)$	evidence	just a normalization constant

The Bayesian Mantra (say it loud)

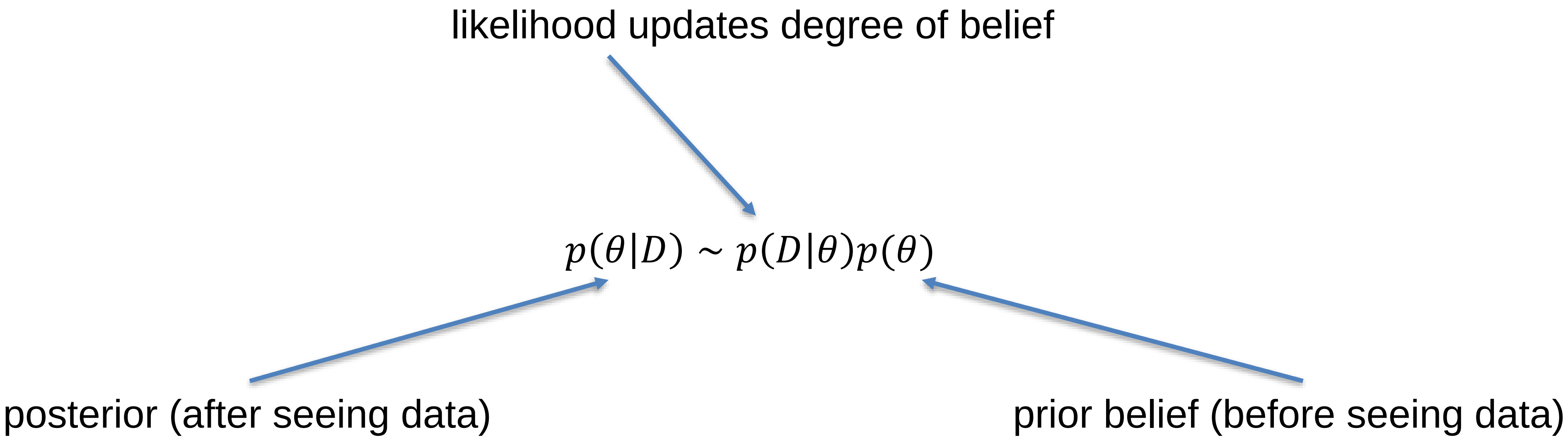
$$\boldsymbol{p}(\boldsymbol{\theta}|\boldsymbol{D}) \sim \boldsymbol{p}(\boldsymbol{D}|\boldsymbol{\theta})\boldsymbol{p}(\boldsymbol{\theta})$$

“The posterior is proportional to the likelihood, times the prior”

Interpretation updating the degree of belief

- Parameters θ are random variables, following a distribution
  - This distribution reflects our belief about which values are probable for θ

    Geben Sie hier eine Formel ein.
- The Bayes formula is seen as an update of our belief in the light of data



likelihood updates degree of belief

$$p(\theta|D) \sim p(D|\theta)p(\theta)$$

posterior (after seeing data)

prior belief (before seeing data)

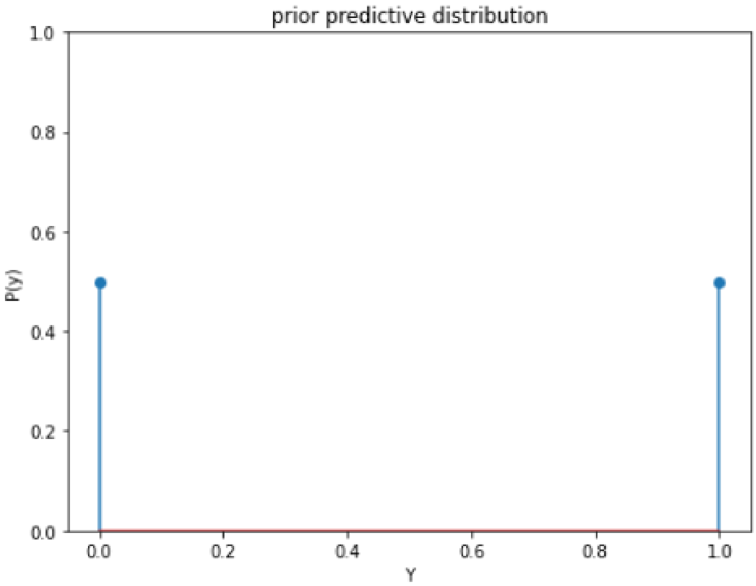
Example Coin Toss

Head or tail?

$Y_i \in \{0, 1\}$, here 0 stands for tail and 1 for head

$Y \sim \mathrm{Ber}(\theta)$, with 1 parameter $\theta = P(Y = 1)$

For a fair coin $\theta = 0.5$



But how to know if a coin is fair?

Analyzing a Coin Toss Experiment

- We do an experiment and observe 3 times head → D='3 heads'
- θ parameter for the Bernoulli-distribution (probability of head)
- Before the experiment we assume all value of θ are equally likely $p(\theta) = \text{const}$
- Calculate likelihood p$(D|\theta) = p(y=1)\ \cdot p(y=1)\ \cdot p(y=1)\ =\ \theta \cdot \theta \cdot \theta\ =\ \theta^3$
- Posterior changes to head

Posterior (after)

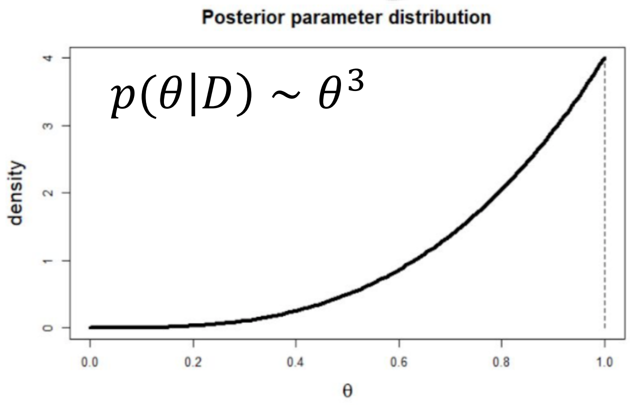
Prior (before)

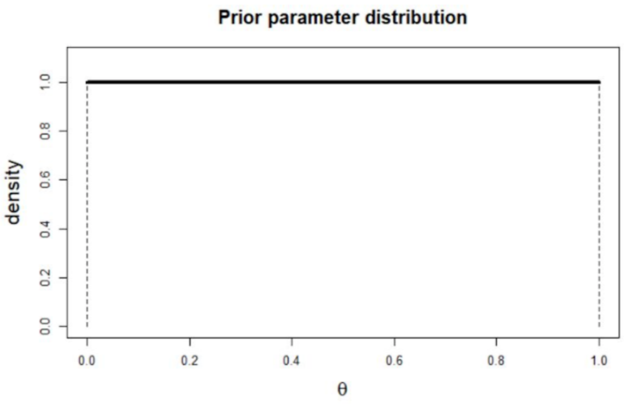
$$p(\theta|D) \sim p(D|\theta)p(\theta)$$

$p(D|\theta)$

θ^3

$p(\theta)$





$p(\theta|D) = 4 \cdot \theta^3$ (the factor 4 is needed for normalization so that the posterior integrates to 1)

Posterior Predictive Distribution

Posterior predictive distribution

$$\mathrm{p}(y|x,D) = \int_\theta p(y|x,\theta) \cdot p(\theta|D)\ d\theta$$

The coin example is unconditional (there is no predictor x)

$$\mathrm{p}(y|D) = \int_\theta p(y|\theta) \cdot p(\theta|D)\ d\theta$$

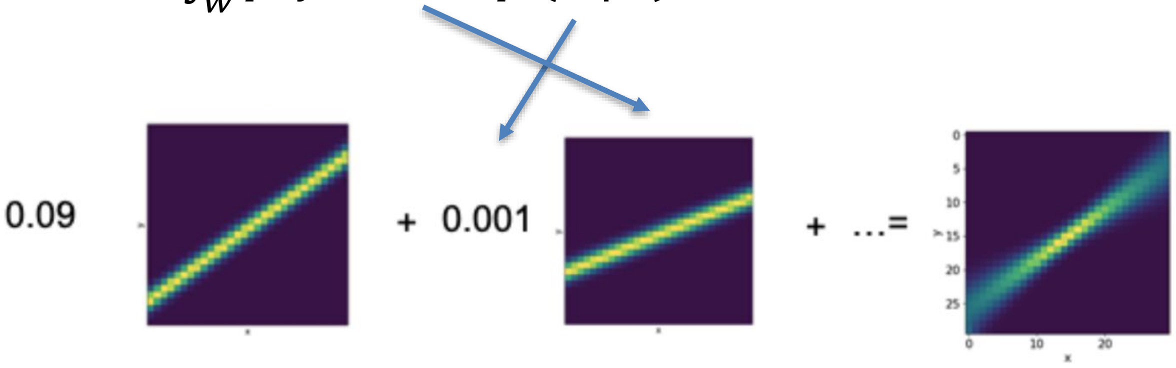
For the coin, we are interested in $\mathrm{p}(y=1|D)$, the probability for head

- Likelihood: $p(y=1|\theta) = \theta$
- Posterior$p(\theta|D) = 4 \cdot \theta^3$

Posterior predictive distribution $\mathrm{p}(y=1|D) = \int_\theta \theta\ \cdot 4 \cdot \theta^3 d\theta = 0.8$

Summary

- $p(y|x,D) = \int_w p(y|x,w) \cdot p\,(w|D)\ dw$



$p(y|x,D)$

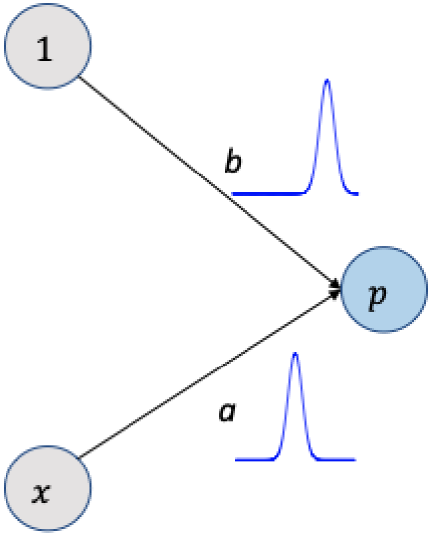
- Not just a single solution
  - “Integration instead of optimization”

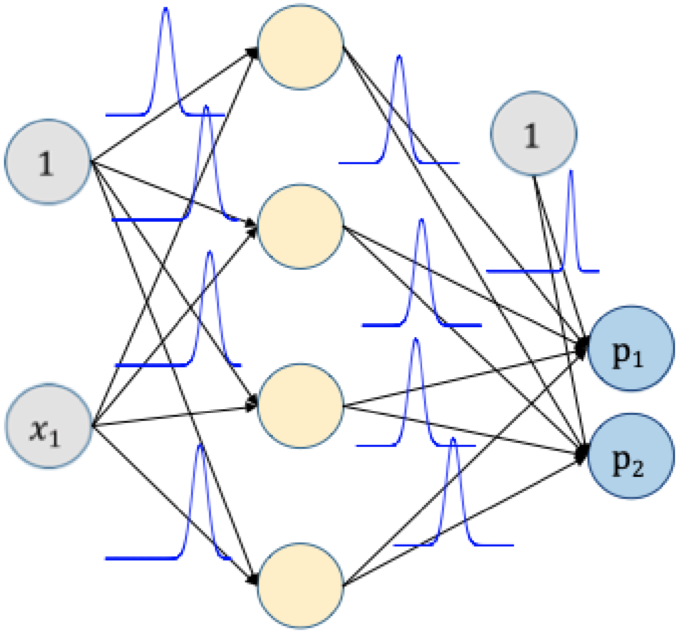
- Bayes usually achieves
  - Better uncertainty estimates
  - Better predictive performance

Bayesian Neural Networks

Bayesian Neural* Networks (BNN)

- Linear Regression with Gaussian Prior and fixed Sigma can be solved analytically





- Bayesian Neural Network cannot be solved analytically

*Don’t confuse Bayesian Networks (DAGs) with Bayesian Neural Networks

Approximations to BNN

- A BNN would require to calculate

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\sum_\theta p(D|\theta)p(\theta)}$$

- Usually no analytical solution exists (only for simple problems)
- Computing $\sum_\theta p(D|\theta)p(\theta)$ is impossible for high-dimension θ

Approximations

- MCMC (only for very small NN feasible)
  - Sample from $p(\theta|D)$ with knowledge of $p(D|\theta')p(\theta')$ / $p(D|\theta'')p(\theta'')$
- Gaussian variational Inference VI
  - Approximate $p(w|D)$ by a Gaussian $N(\mu, \sigma\ )$ and tune μ, σ
- MC-Dropout
  - MC-Dropout during predictions (magically) samples from a variational approximation

Variational Inference

The principle of VI

- Replace $p(\theta|D)$ with $q_\lambda(\theta)$ (Variational Ansatz)
- Typically independent Gaussian for each weight $\lambda = (\mu, \sigma)$
  - $p(\theta|D) = q_{\mu,\sigma}(\theta)$

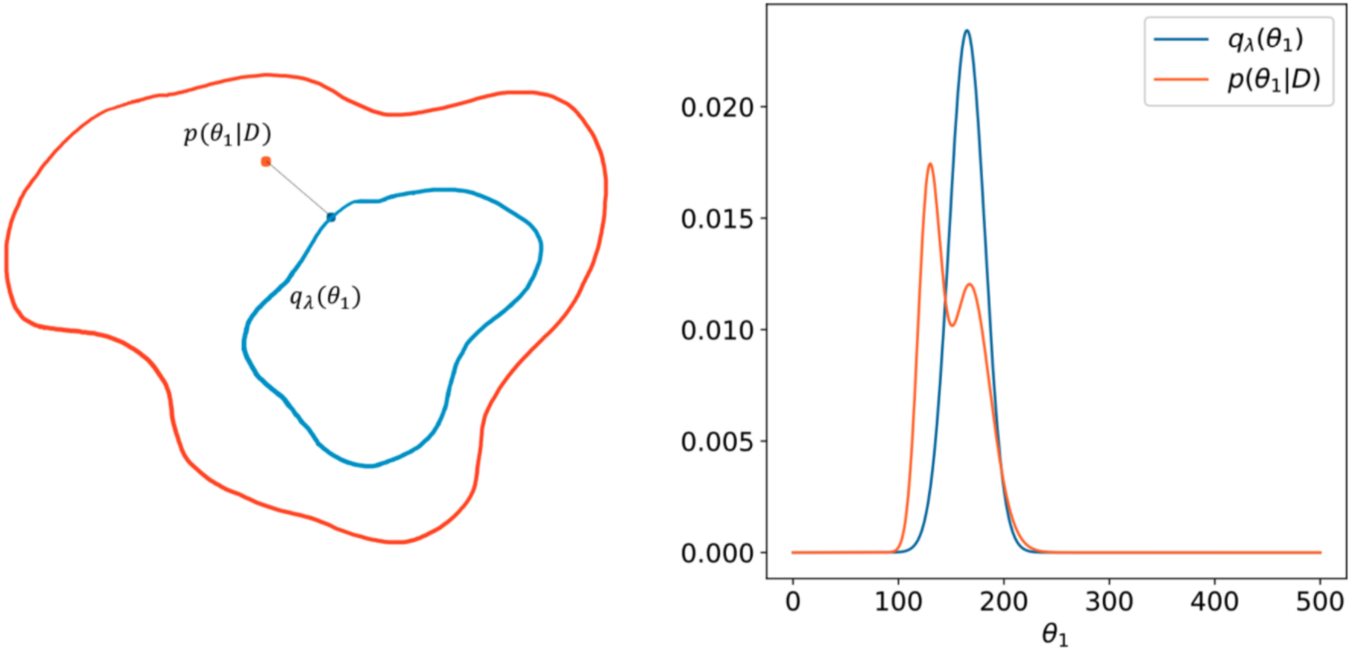


Figure 8.3 The principle idea of variational inference (VI). The larger region on the left depicts the space of all possible distributions, and the dot in the upper left represents the posterior $p(\theta_1|D)$ (corresponding to the dotted density on the right panel). The inner region depicts the space of possible variational distributions $q_\lambda(\theta_1)$. The optimized variational distribution $q_\lambda(\theta_1)$ (illustrated by the point in the inner loop in the left panel, corresponding to the solid density on the right panel) has the smallest distance to the posterior (shown by the dotted line on the right).

Particularies

- Layers for VI:
  - `DenseReparameterization`
  - `Convolution{1D,2D,3D}Reparameterization`
  - `Further a method called Flipout to speed up training`

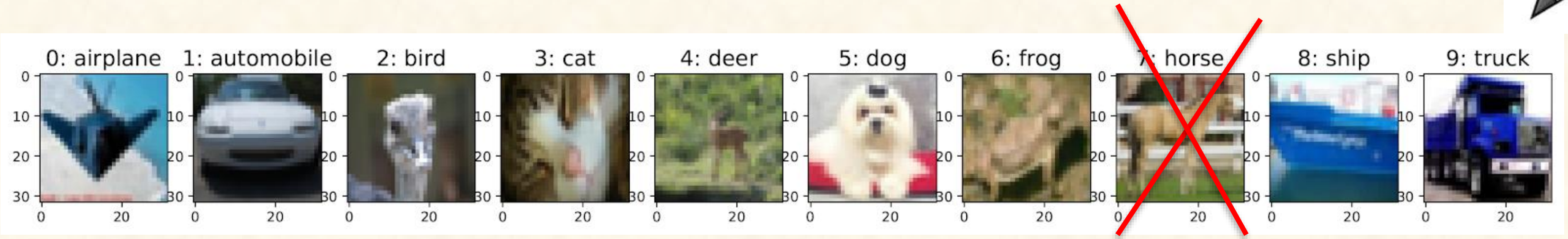
From documentation (Convolution2DFlipout)

When doing minibatch stochastic optimization, make sure to scale this loss such that it is applied just once per epoch (e.g. if kl is the sum of losses for each element of the batch, you should pass kl / num_examples_per_epoch to your optimizer)

```
kl = tfp.distributions.kl_divergence
divergence_fn=lambda q, p, _: kl(q, p) / (num * 1.0)

DenseReparameterization(1,kernel_divergence_fn=divergence_fn)
```

Hands-on Time cntd.: Fit the VI Bayesian NN



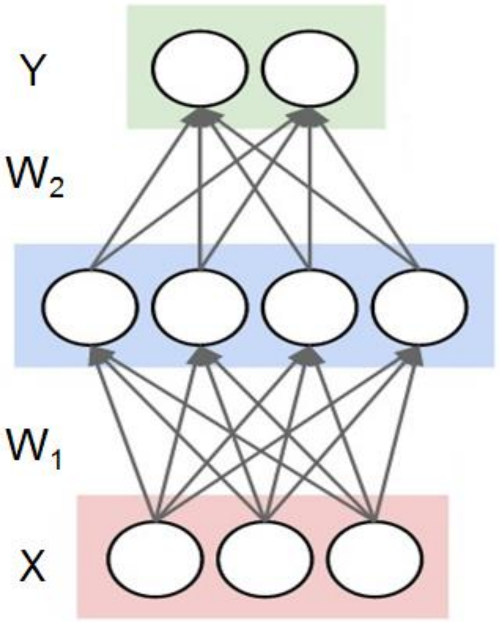
Train a CNN with only 9 of the 10 classes and investigate if the uncertainties are different when predicting images from known or unknown classes.

https://github.com/tensorchiefs/dl_course_2021/blob/master/notebooks/20_cifar10_classification_mc_and_vi.ipynb

Comparing non-Bayesian with Bayesian NN

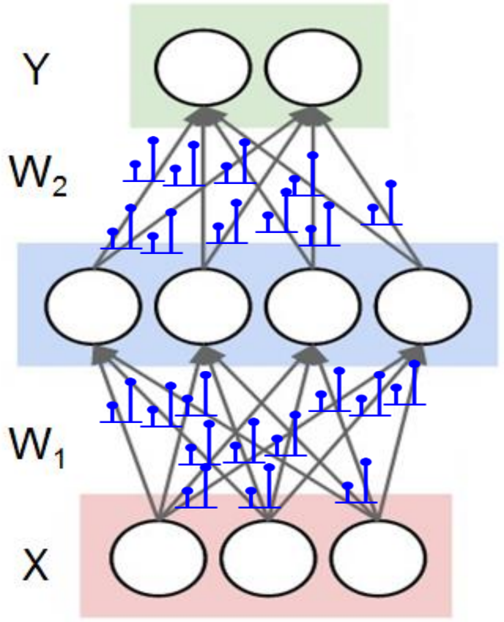
Non-Bayesian and Bayesian NNs

Non-Bayesian
NN



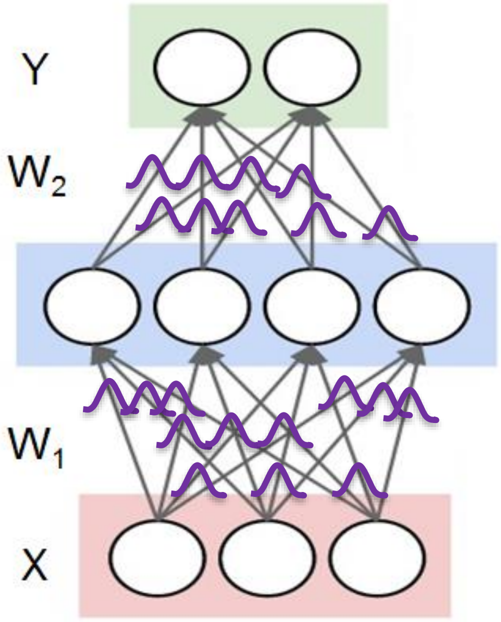
Weights are fixed

MC dropout
Bayesian NN



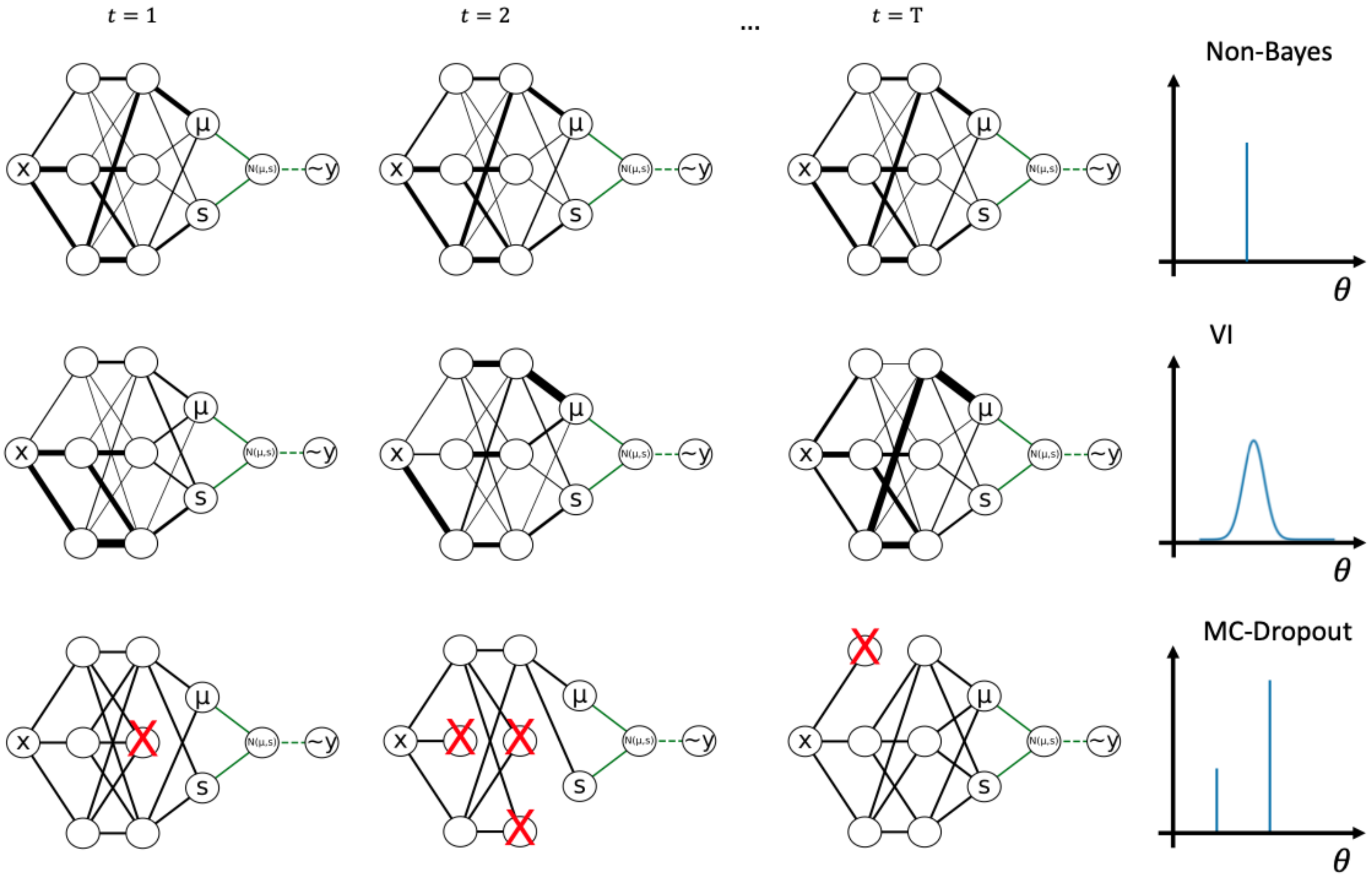
Weights have Bernoulli-kind distribution

VI
Bayesian NN



Weights have Gaussian distribution

Comparing different Network types



https://www.youtube.com/watch?v=mQrUcUoT2k4

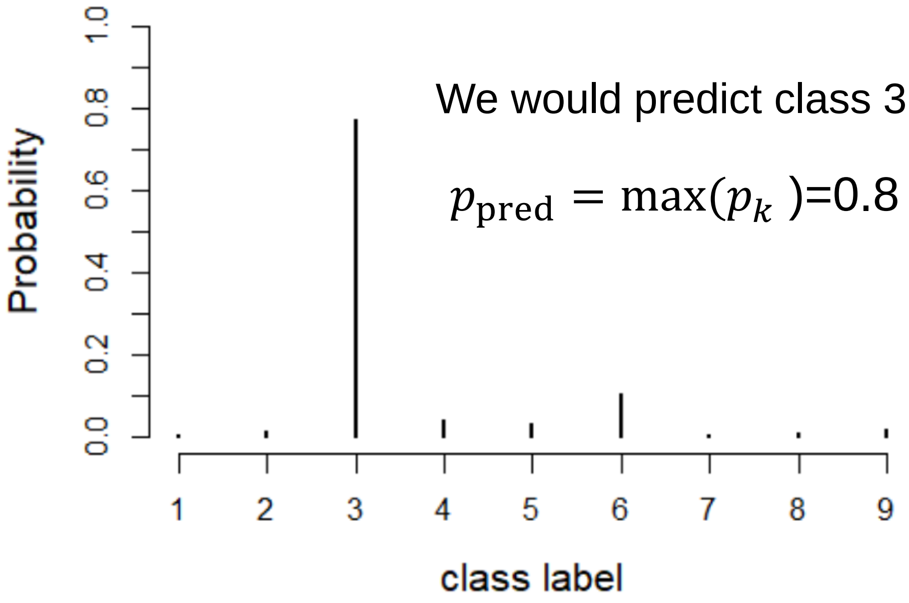
A Non-Baysian NN learns one set of weights: the same input same output
A Bayesian NN learns distribution of weights: same input different outputs

Uncertainty measures in classification

Uncertainty in non-Bayesian classification

Multinomial CPD

$\mathrm{MN}(p_1(x,w), p_2(x,w), \ldots, p_9(x,w))$



We would predict class 3

$p_{\text{pred}} = \max(p_k)$=0.8

In a non-Bayesian NN we make for each input x ONE CPD:

Image x
MN(p1(x,w), …, p9(x,w))

Uncertainty measures capturing the **aleatoric** uncertainty :

Negative log-Likelihood: $NLL = -\log(p_{\text{pred}})$

Entropy: $H = -\sum_{k=1}^{9} p_k \cdot \log(p_k)$

Uncertainty in Bayesian classification

In a Bayesian NN we sample T-times from the weight distributions and get each time a slightly different multinomial CPD

predict_no	**Image x**
1	MN(p1(x,w1), …, p9(x,w1))
2	MN(p1(x,w2), …, p9(x,w2))
…	
T	MN(p1(x,wT), …, p9(x,wT))

For each class k ($k \in \{1,2,\dots,9\}$) we determine the mean probability: $p_k^* = \frac{1}{T}\sum_{i=1}^{T} p_{k_i}$

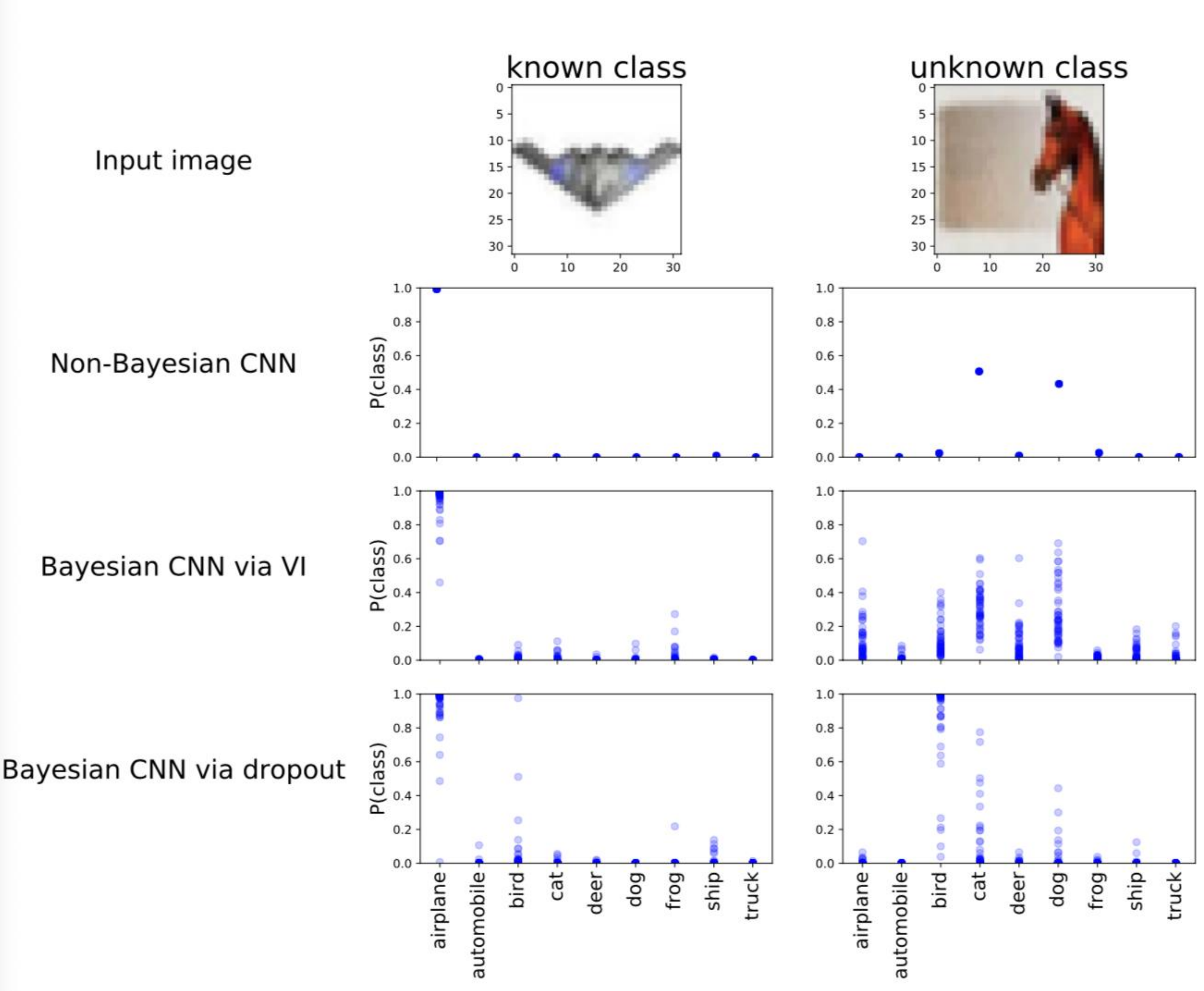
We predict the class with the highest mean probability: $p_{pred}^* = \max(p_k^*)$

Uncertainty measures including **aleatoric and epistemic** contributions:

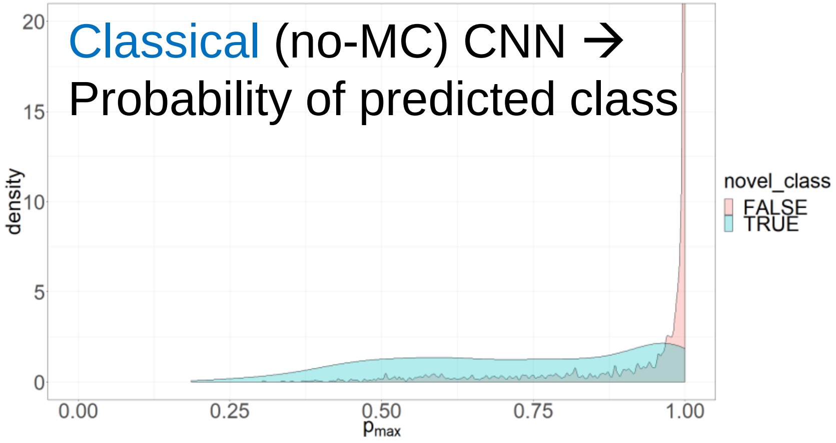
Entropy: $H^* = -\sum_{k=1}^{9} p_k^* \cdot \log(p_k^*)$

Total variance: $V_{tot}^* = \sum_{k=1}^{9} \mathrm{var}(p_k) = \sum_{k=1}^{9} \sum_{i=1}^{T} (p_{kt} - p_k^*)^2$

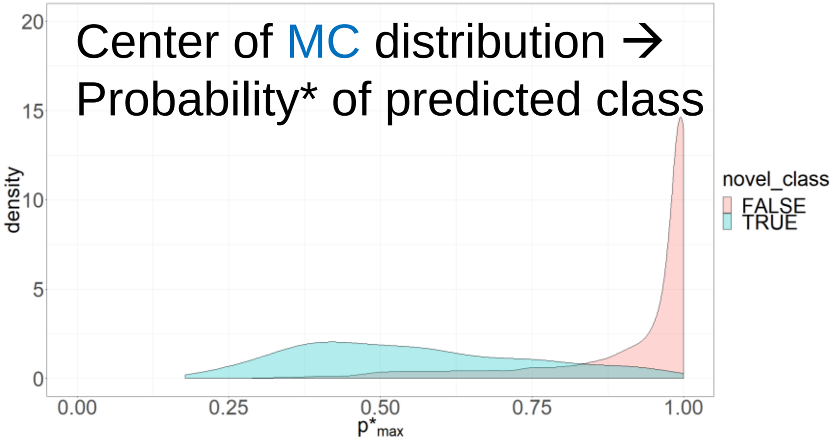
Looking at the predictive distribution!



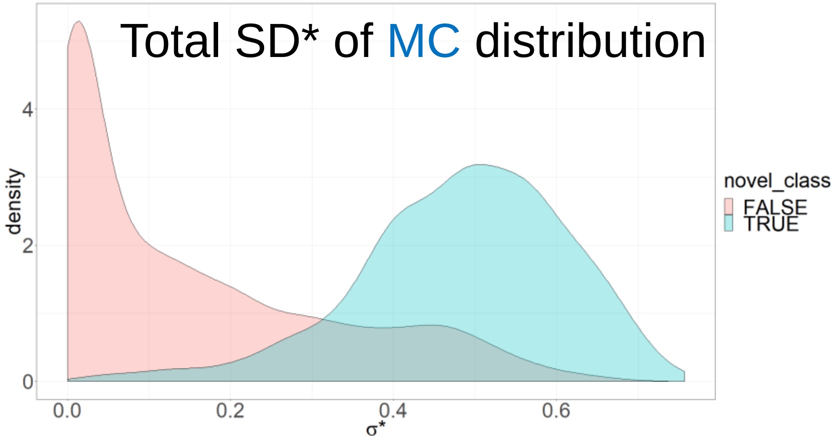
Do known/novel classes yield different values for probability and uncertainty measures?



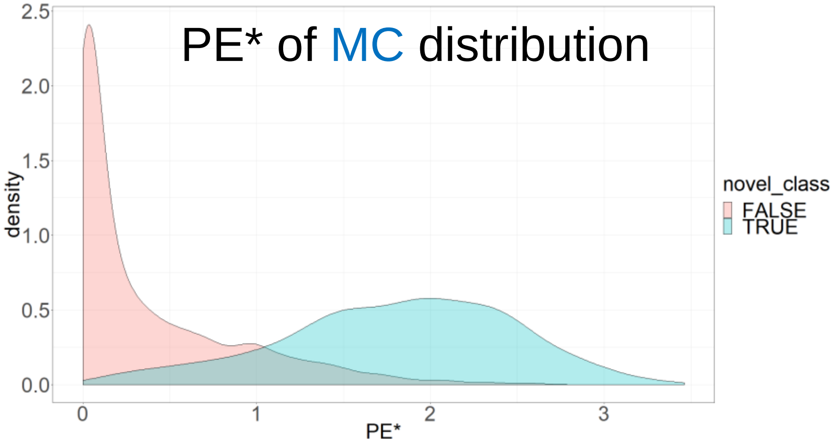
Classical (no-MC) CNN → Probability of predicted class



Center of MC distribution → Probability* of predicted class



Total SD* of MC distribution



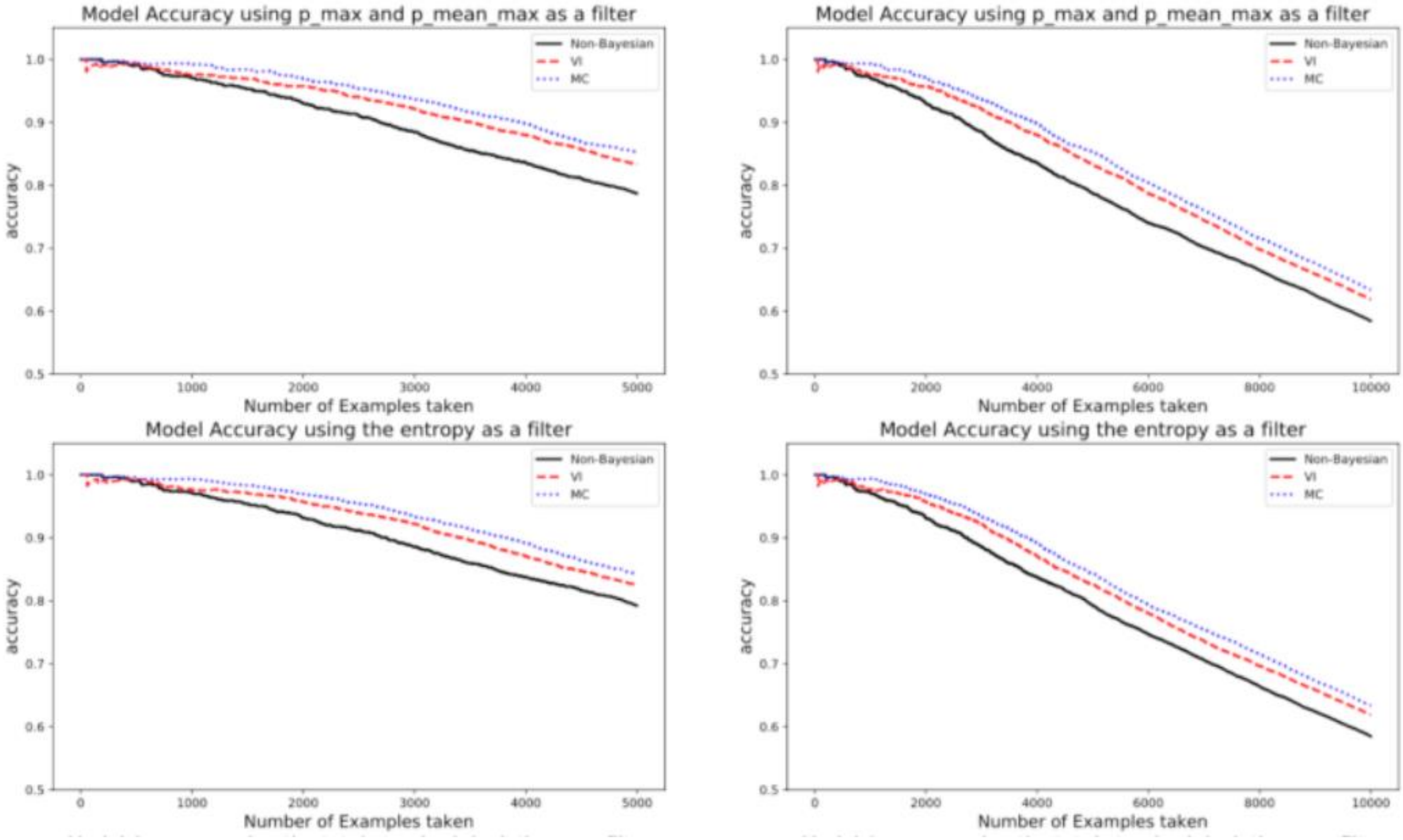
PE* of MC distribution

Filtering experiment based on uncertainty

Goal: Get higher accuracy by filter only predictions which are quite certainly correct

- Each prediction has an attached uncertainty measure
- Sort predictions according to the uncertainty measures
- A set of predictions with very low uncertainties should achieve a high accuracy
- By successively adding predictions with increasing uncertainties should yield predictions sets with decreasing accuracies.

Filtering experiment to compare uncertainty measures



Uncertainty from non-Bayesian NN is less good in filtering out wrong classifications than uncertainty measures from Bayesian variants of the NN.

Uncertainty measures in regression

Uncertainty in non-Bayesian NN

We do predictions for 400 x-values between -10 and 30 yielding for each x a Gaussian CPD.

x1= -10	x2= -9.9	...	x400= 30
$N\left(\mu_{x_{1,w}}, \sigma_{x1,w}\right)$	$N\left(\mu_{x_{2,w}}, \sigma_{x2,w}\right)$		$N\left(\mu_{x_{400,w}}, \sigma_{x400,w}\right)$

Uncertainty measures capturing the **aleatoric** uncertainty at $\boldsymbol{x}$:

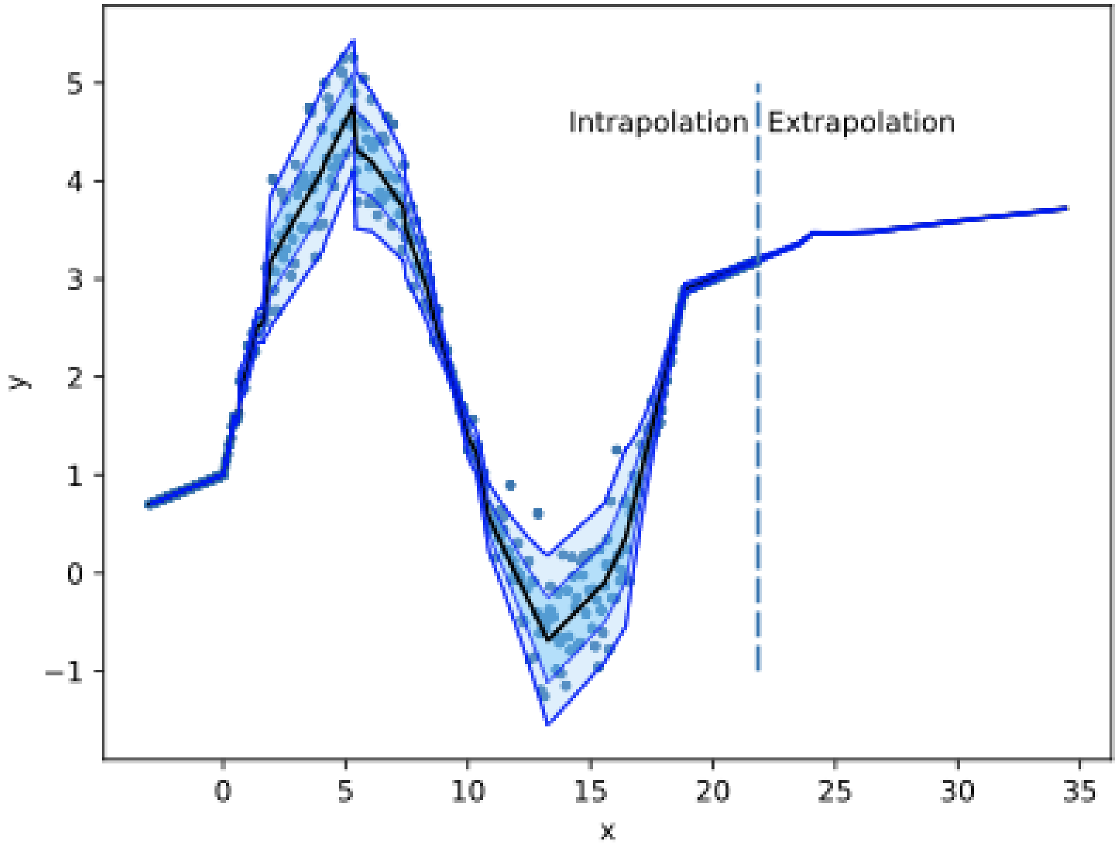
Standard deviation: σ_x

95% PI: $[q_{0.025}; q_{0.975}] = [\mu_x - 1.96 \cdot \sigma_x\,;\, \mu_x + 1.96 \cdot \sigma_x]$

Remark:
We could also estimate the 95% PI at position x by sampling several times from the CPD and determine the 0.025 and 0.975 quantiles, yielding :
95% PI: $[q_{0.025}; q_{0.975}]$

The problem of non-Bayesian NN

Problem:
A non-Bayesian NN does extrapolation with very small uncertainty



Uncertainty in Bayesian regression NN

In a Bayesian NN we sample T-times from the weight distributions and get each time a slightly different CPD. In regression the CPD is often Gaussian.

We do predictions for 400 x-values between -10 and 30 yielding in each of the T runs a different Gaussian CPD at each x-position.

predict_no	x1= -10	x2= -9.9	**…**	x400= 30
1	N(x1,w1,x1,w1)	N(x2,w1,x2,w1)		N(x400,w1,x400,w1)
2	N(x1,w2,x1,w2)	N(x2,w2,x2,w2)		N(x400,w2,x400,w2)
…				
T	N(x1,wT,x1,wT)	N(x2,wT,x2,wT)		N(x400,wT,x400,wT)

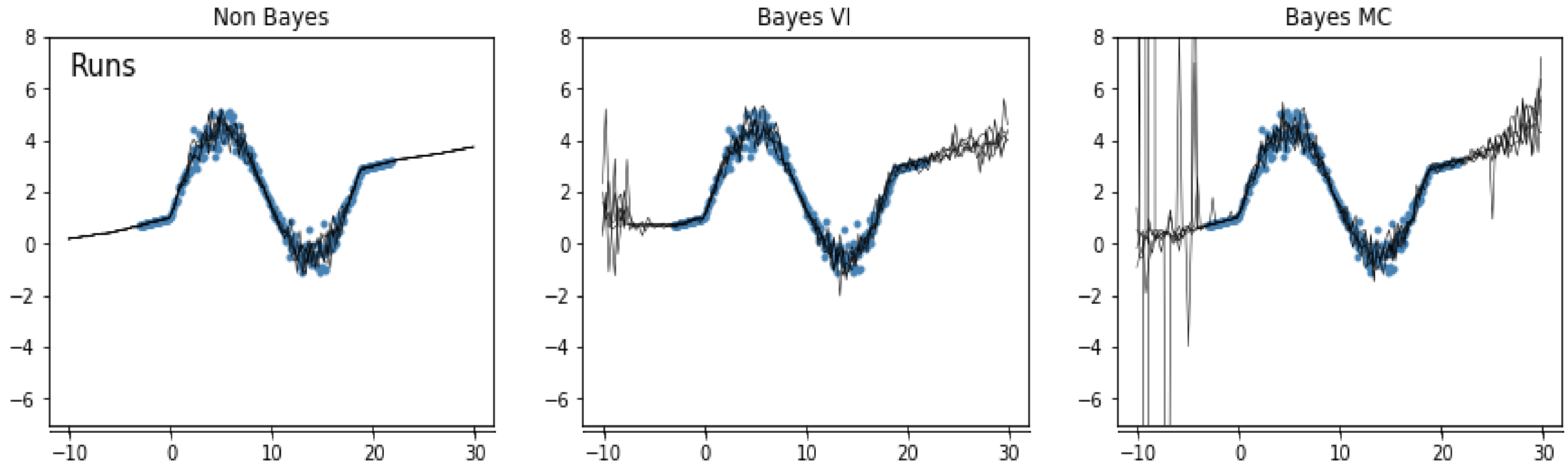
Uncertainty measures including **aleatoric and epistemic** contributions:

To estimate the 95% PI at position x we sample y-values from each of the T CPDs and determine from the samples the 0.025 and 0.975 quantiles, yielding :

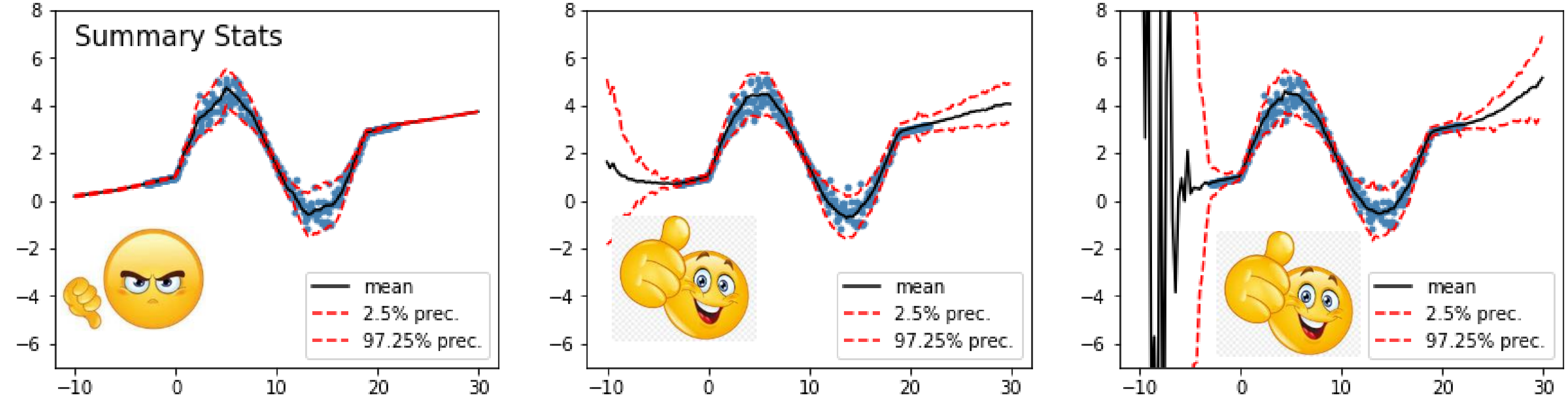
95% PI: $[q_{0.025}; q_{0.975}]$

Can we see enhanced uncertainty in extrapolation

The solid lines show five predicted y-vectors corresponding to 5 CPDs at each x-position.



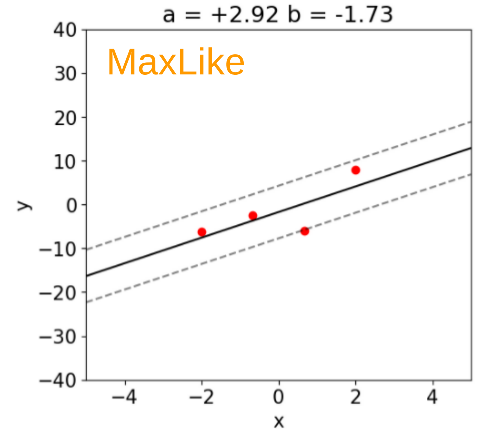
https://youtu.be/FO5avm3XT4g https://youtu.be/mQrUcUoT2k4 https://youtu.be/0-oyDeR9HrE

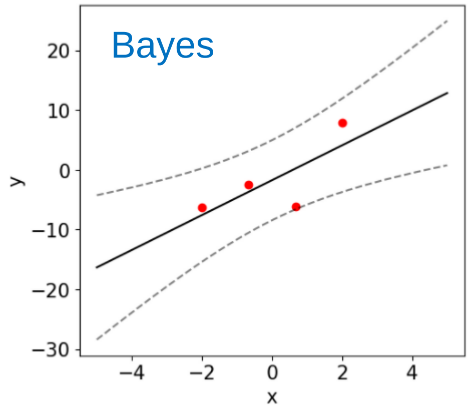


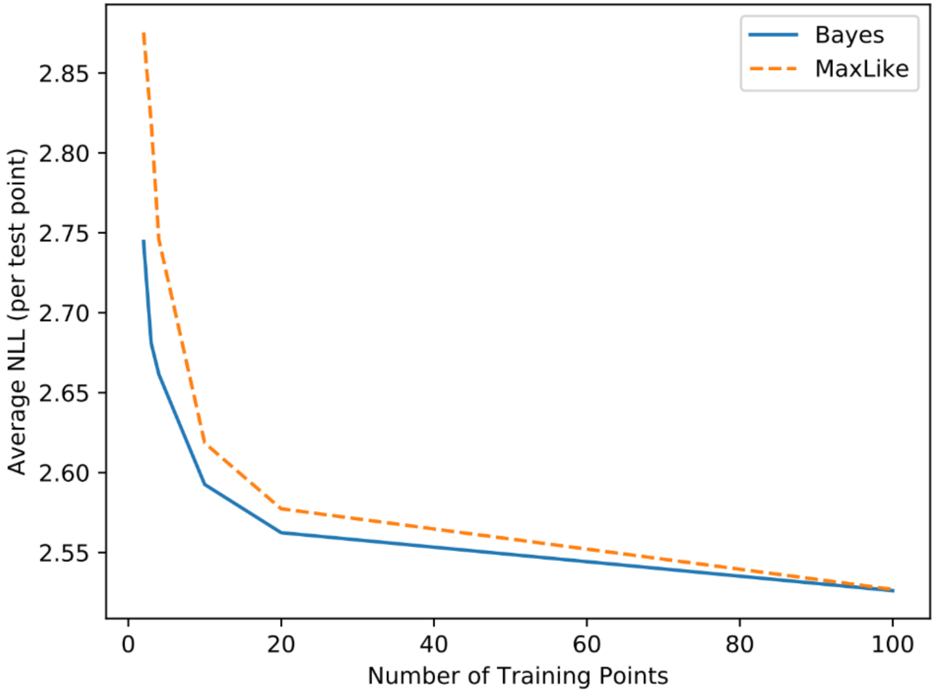
With Bayes we get better prediction performance

Train on 4 data points a linear regression via Maximum Likelihood and via Bayes

Evaluate the prediction performance of both models on 100 test data points sampled from the same distribution as the4 train data points.







Conclusion

- Standard neural networks (NNs) fail to express their uncertainty (can’t talk about the elephant in the room).
- Bayesian neural networks (BNNs) can express their uncertainty.
- BNNs often yield better performance than their non-Bayesian variant.
- Novel classes can be better identified with BNNs, which combine epistemic and aleatoric uncertainties compared to standard NNs.
- Variational inference (VI) and Monte Carlo dropout (MC dropout) are approximation methods that allow you to fit deep BNNs.
- TFP provides easy to use layers for fitting a BNN via VI.
- MC dropout can be used in Keras for fitting BNNs.

VI in TensorFlow probability

```
model = tf.keras.Sequential([
  tfp.layers.DenseReparameterization(1,input_shape=(None,1)),
  tfp.layers.DenseReparameterization(2),
  tfp.layers.DenseReparameterization(3)
])
```

This builds a network, with hidden layers having:

1 Node

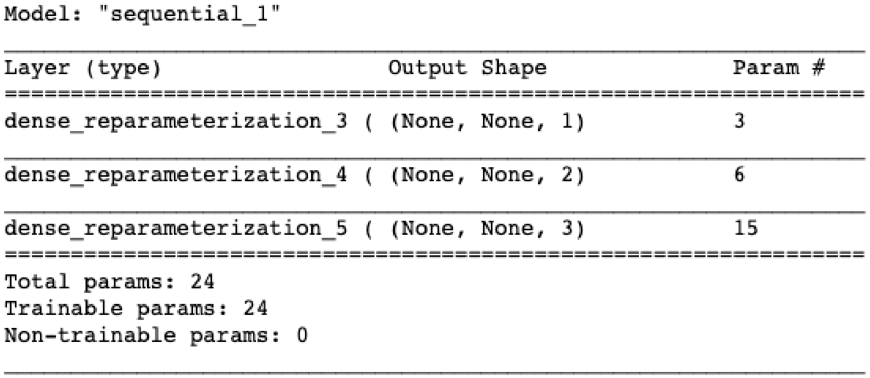
2 Node

3 Node

Question: How many parameters does this network have? Note that biases are included but don’t have variational distributions. [3 minutes]

Solution

```
model = tf.keras.Sequential([
  tfp.layers.DenseReparameterization(1, input_shape=(None,1)),
  tfp.layers.DenseReparameterization(2),
  tfp.layers.DenseReparameterization(3)
])
```



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_reparameterization_3 ( (None, None, 1)           3
_________________________________________________________________
dense_reparameterization_4 ( (None, None, 2)           6
_________________________________________________________________
dense_reparameterization_5 ( (None, None, 3)           15
=================================================================
Total params: 24
Trainable params: 24
Non-trainable params: 0
_________________________________________________________________
```