Machine Intelligence:: Deep Learning
Week 7

Beate Sick, Jonas Brändli, Oliver Dürr

Improving the performance and uncertainty estimates of NN models by taking into account the epistemic uncertainty.

Outline:

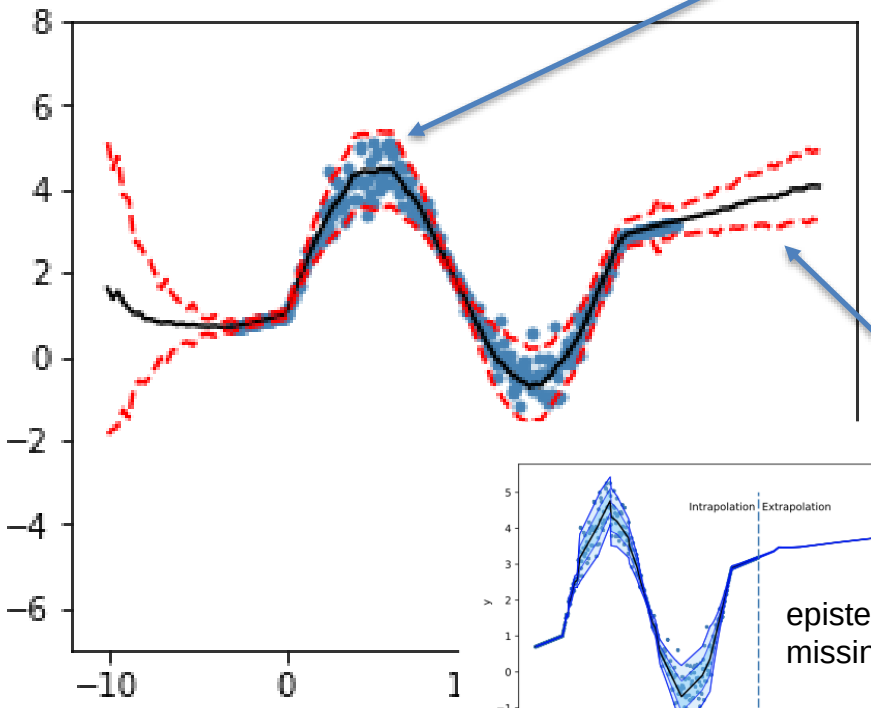
- Issues with current DL approach
  - No uncertainty for the fitted weights
    → epistemic uncertainty is ignored causing different problems:
    - No increased uncertainty in case of extrapolation
    - Deficits in prediction performance

- Approaches to take epistemic uncertainty into account:
  - Deep Ensembling
  - MC Dropout
  - Bayesian Modelling
    - Introduction to Bayesian Statistics
    - A simple example (Bayes the Hacker's way)
    - Variational Approximation

Aleatroic vs. Epistemic Uncertainty

Much spread in train data → aleatroic uncertainty (from latin "Alea Acta est"))

Caesar: Die Würfel sind gefallen!



We understand a dice perfectly (no epistemic uncertainty), but still there is a aleatoric uncertainty about the the number that will show up next when rolling the dice.

epistemic uncertainty is missing in classical NN

No (or few) train data
→ no (or few) "knowledge"
(from Ancient Greek ἐπιστήμη *(epistḗmē)* 'knowledge')
→ epistemic uncertainty

- *Aleatoric* uncertainty is due to the uncertainty, that is inherent in the data.
- The uncertainty when leaving the 'known ground' is called *epistemic* uncertainty.

The elephant in the room

A high performant NN (trained on imageNet data) does not see the elephant!

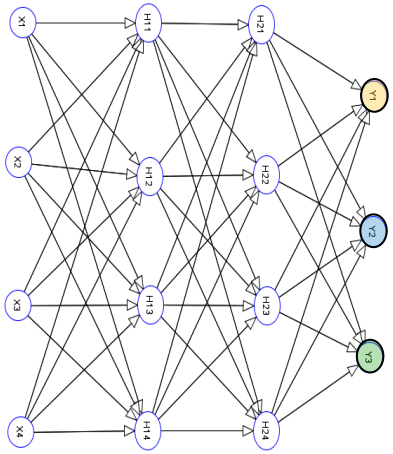
General VIEW DOCS

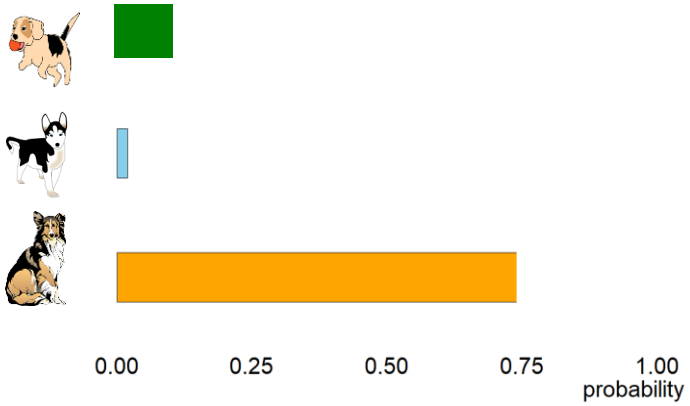
PREDICTED CONCEPT	PROBABILITY
group	0.979
adult	0.977
people	0.976
furniture	0.960
room	0.957
business	0.903
indoors	0.901
man	0.896
seat	0.895

Elephant in the room

- Aufgabe: https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/18_elephant_in_the_room.ipynb

CNNs have high performance on in-distribution examples

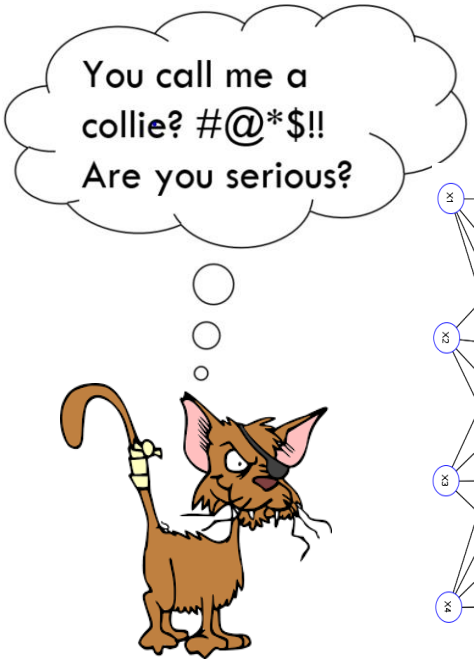


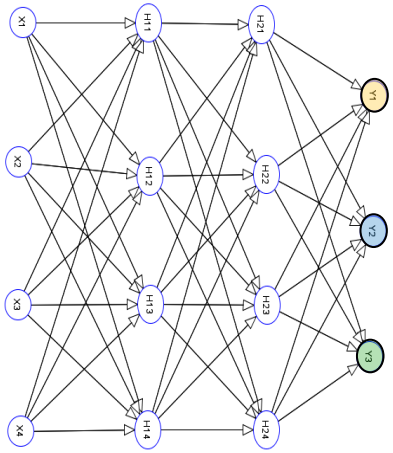


CNNs yield high accuracy and calibrated probabilities, but…

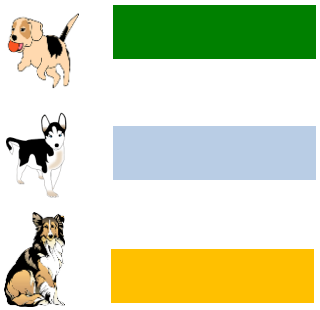
A classical NN cannot ring the alarm in case of out-of-distribution (OOD) examples

What happens if we present a novel class to the CNN?





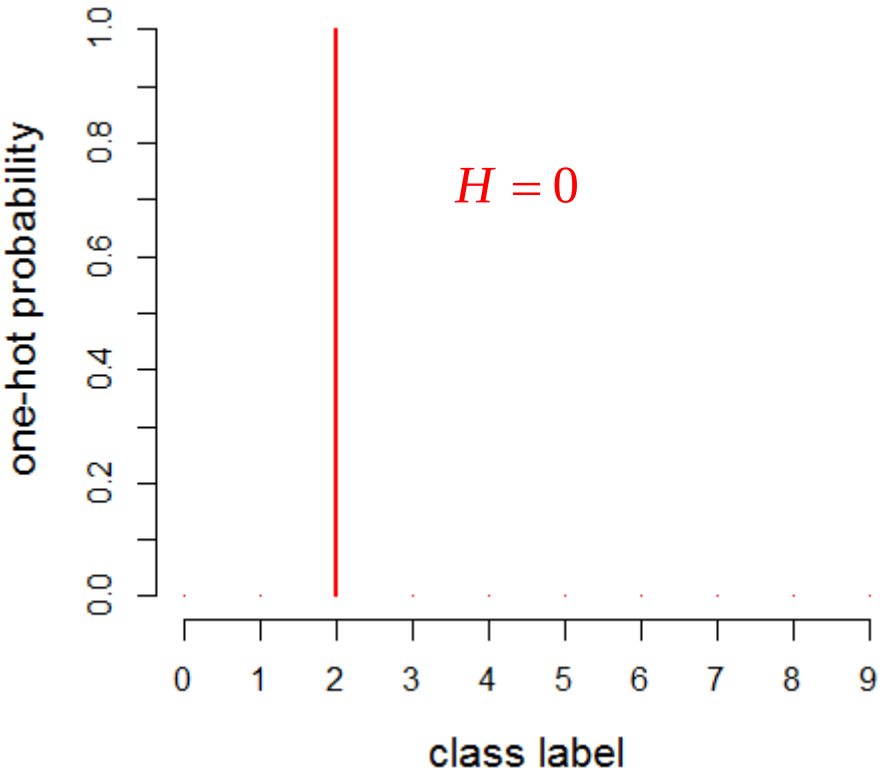
You might expect:

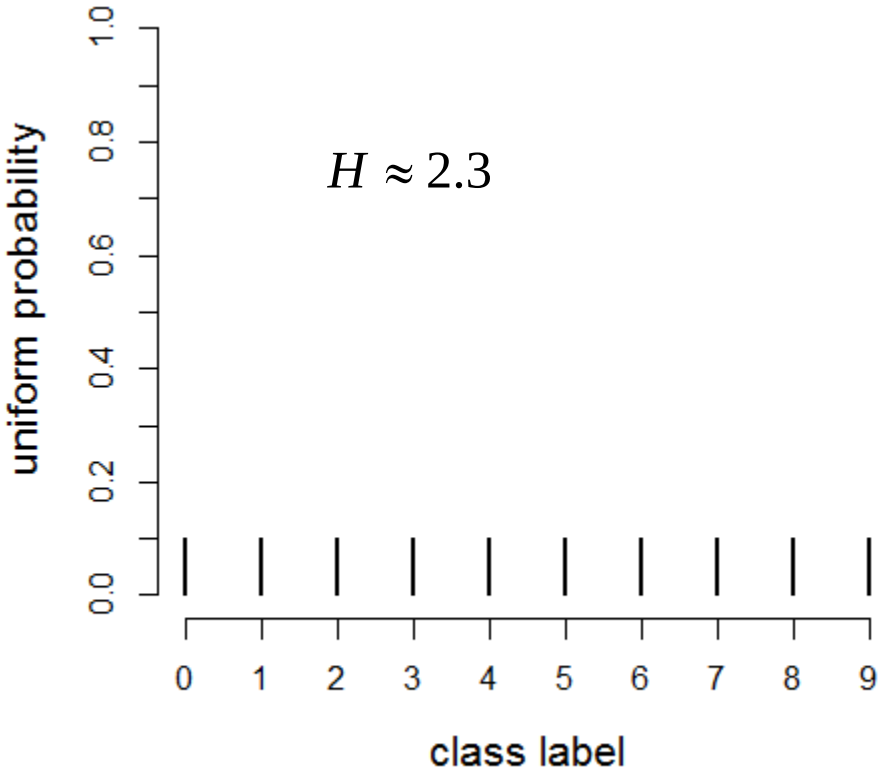


Recall: Entropy as measure for uncertainty

$$H(P) = -\sum_i p_i \cdot \log(p_i)$$

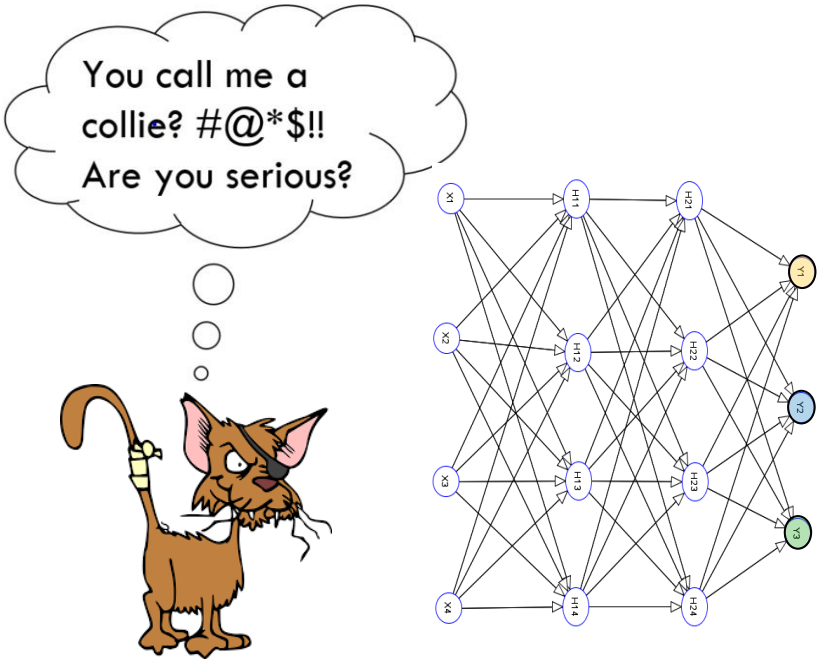
Entropy is a measure for “untidyness” or uncertainty.
If a distribution has only one peak, it is tidy and H=0
If all outcomes are equally probable it is maximal untidy





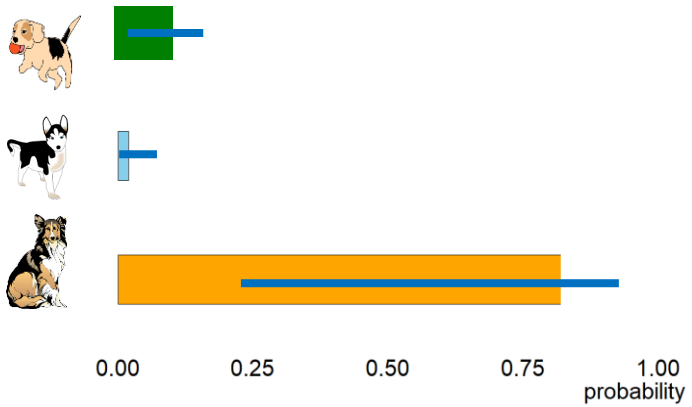
A classical NN cannot ring the alarm in case of out-of-distribution (OOD) examples

What happens if we present a novel class to the CNN?



But you probably get:

Plain wrong !



We need some error bars!
We need epistemic uncertainty!

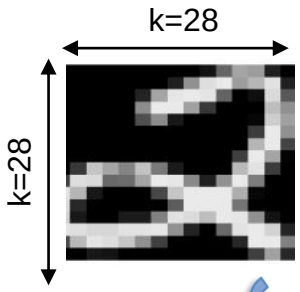
Importance to detect OOD (out of distribution)

- Current DL Systems bad in out of distribution OOD situations
- Application need at least to detect OOD situations

Foto: https://www.crn.de/software-services/halloween-training-fuer-google-autos.108409.html

Deep ensembles

Recall the training of NN models via SGD



Flatten to vector with k^2 elements

input $\mathbf{x}^t$

$$\begin{bmatrix} 24.0 \\ -5.1 \\ 12.2 \\ 89.9 \\ \vdots \\ 3.2 \end{bmatrix}$$

$$\mathbf{x}_{(1,k^2)} \cdot \mathbf{W}_{(k^2,2)} + \mathbf{b}_{(1,2)} = \mathbf{z}_{(1,2)}$$

Score or logit $\mathbf{z}^t$

$$\begin{bmatrix} 2.3 \\ 0.1 \end{bmatrix}$$

$$\hat{p}_k = \frac{e^{z_k}}{\sum_j e^{zj}}$$

Softmax or probability $\mathbf{p}=S(\mathbf{z})$

$$\begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix}$$

1-hot labels $\mathbf{y}$

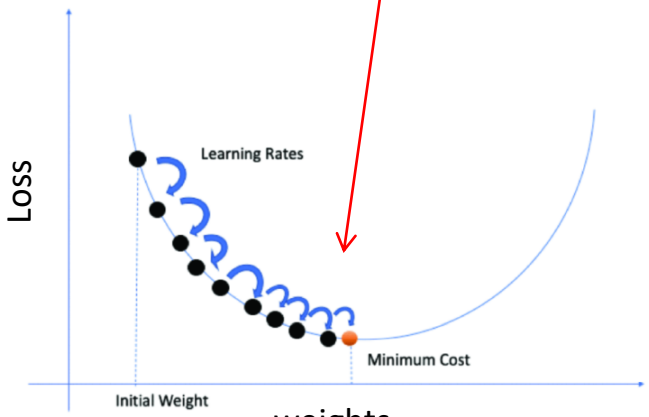
$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$ "2" "1"

Loss: NLL, X-entropy $L(\mathbf{p},\mathbf{y})$

$$= -\sum_{k=1}^{2} y_k \cdot \log(p_k)$$

Take step in direction of descent gradient:
(the gradient is oriented orthogonal to contour lines)

$$w_i^{(t)} = w_i^{(t-1)} - \varepsilon^{(t)} \frac{\partial L(\mathbf{w})}{\partial w_i}\Big|_{w_i=w_i^{(t-1)}}$$



Loss = NLL averaged over all images in mini-batch

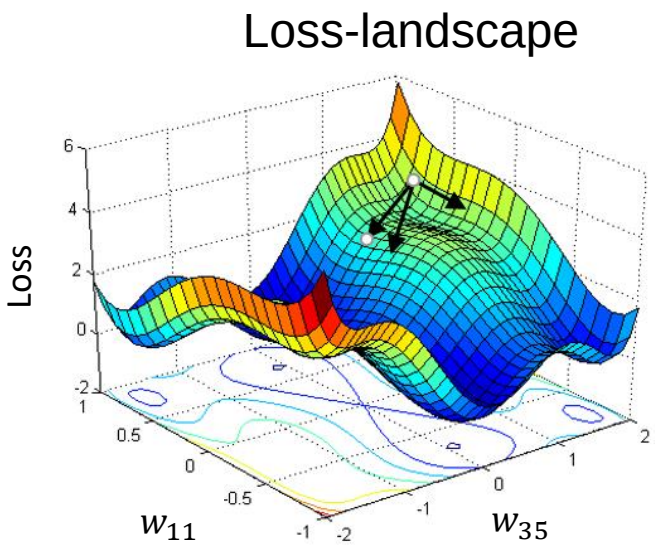
$$\text{NLL} = \frac{1}{N}\sum_i L(\mathbf{p}_i, \mathbf{y}_i)$$

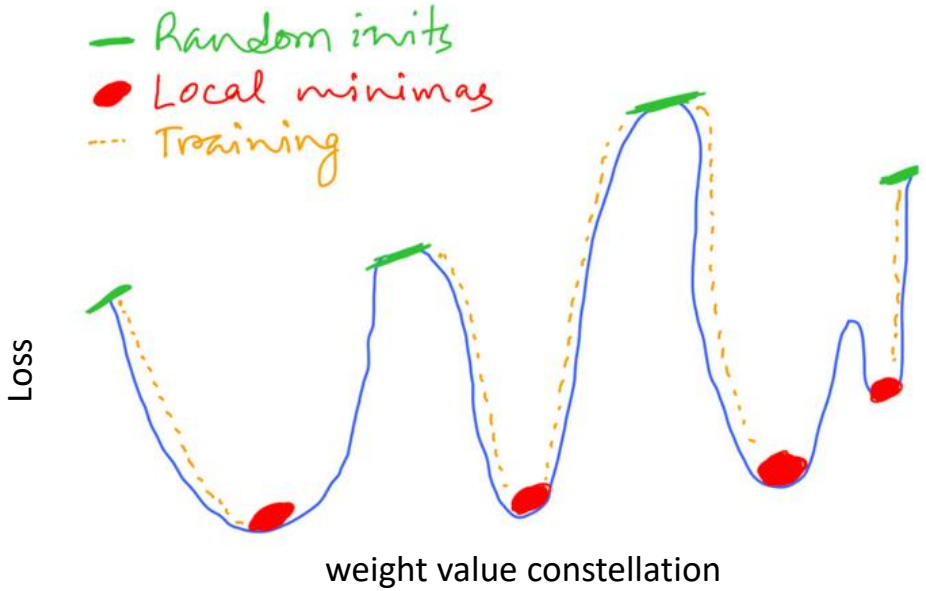
Loss

$$C(w_1, w_2)$$

Image credicts

The loss-landscape in DL is usually not convex



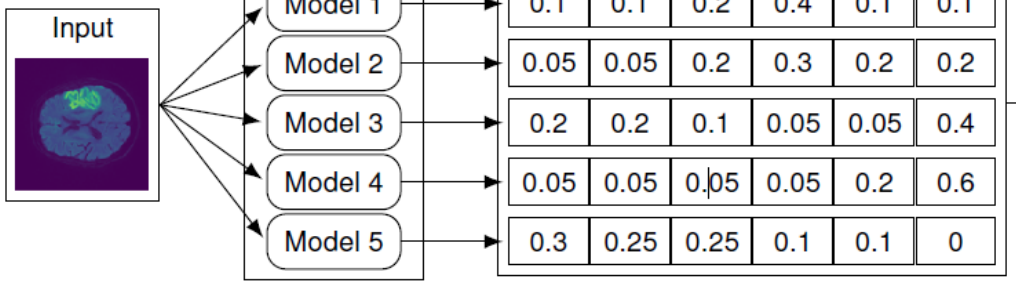


The loss-landscape of DL models has many local minima with similar depth.

Training is started with a random weight value initialization → training the NN with the same data several times is usually ending in different local minima.

Image credit: https://wandb.ai/authors/loss-landscape/reports/Understanding-the-Effectivity-of-Ensembles-in-Deep-Learning--VmlldzoxODAxNjA

Deep ensembling:
Train several NN models and average their predicitions

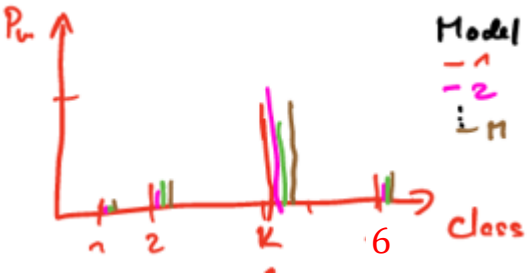


Each model predictis probabilities for the 6 classes

p_1	p_2	p_3	p_4	p_5	p_6
0.1	0.1	0.2	0.4	0.1	0.1
0.05	0.05	0.2	0.3	0.2	0.2
0.2	0.2	0.1	0.05	0.05	0.4
0.05	0.05	0.05	0.05	0.2	0.6
0.3	0.25	0.25	0.1	0.1	0

We get the ensemble predictions by averaging the probabilities that were predicted by the different models for each class

p_1^E	p_2^E	p_3^E	p_4^E	p_5^E	p_6^E
0.14	0.13	0.16	0.18	0.13	0.26



We can get the aleatoric ensemble uncertainties by computing the entropy of the ensemble prediction $H(P^E) = -\sum_i p_i^E \cdot \log(p_i^E)$.
We can get the epistemic ensemble uncertainties by e.g. computing the standard deviation of the probabilities that were predicted by the different models for each class

sd_1^E sd_2^E sd_3^E sd_4^E sd_5^E sd_6^E

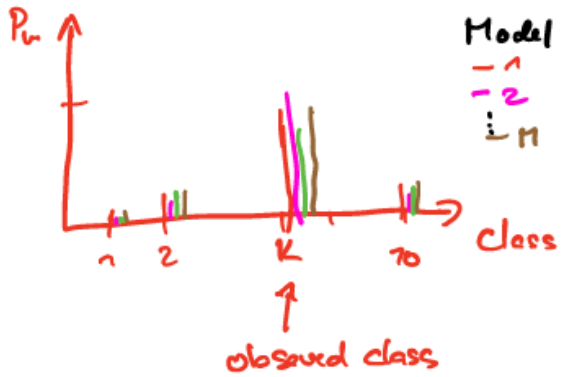
Nice:
For the convex NLL loss, it is guaranteed, that the NLL of the ensemble prediction is better (smaller or equal) than the average NLL of the individual models.

Original paper by Lakshminarayanan et al.: https://arxiv.org/pdf/1612.01474.pdf

Ensembling improves the NLL performance

Ensemble prediction for an observation with observed class = k, based on M models:

$$p_k^E = \frac{1}{M} \sum_{m=1}^{M} p_k \qquad k = 1 \dots 10 \text{ for 10 classes}$$



associated NLL contribution ℓ:

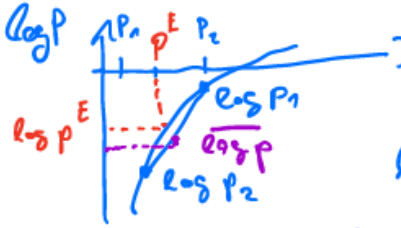
Model m: $\ell_m = -\log p_{km}$; Ensemble: $\ell^E = -\log p_k^E$

to show $\bar{\ell} \geq \ell^E$ with $\bar{\ell} = \frac{1}{M} \sum_{m=1}^{M} \ell_m = \frac{1}{M} \sum_m -\log p_{km}$

$$= -\underbrace{\frac{1}{M} \sum_m \log p_{km}}_{\leq \log \frac{1}{M} \sum_m p_{km}}$$

$$\geq -\log p_k^E = \ell^E \qquad \square$$

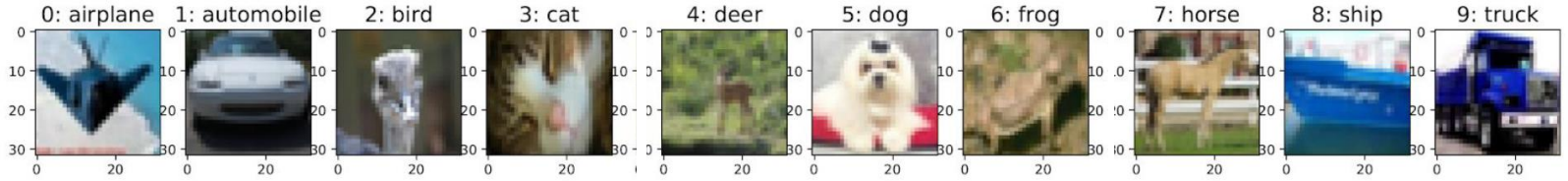
use Jensen inequality

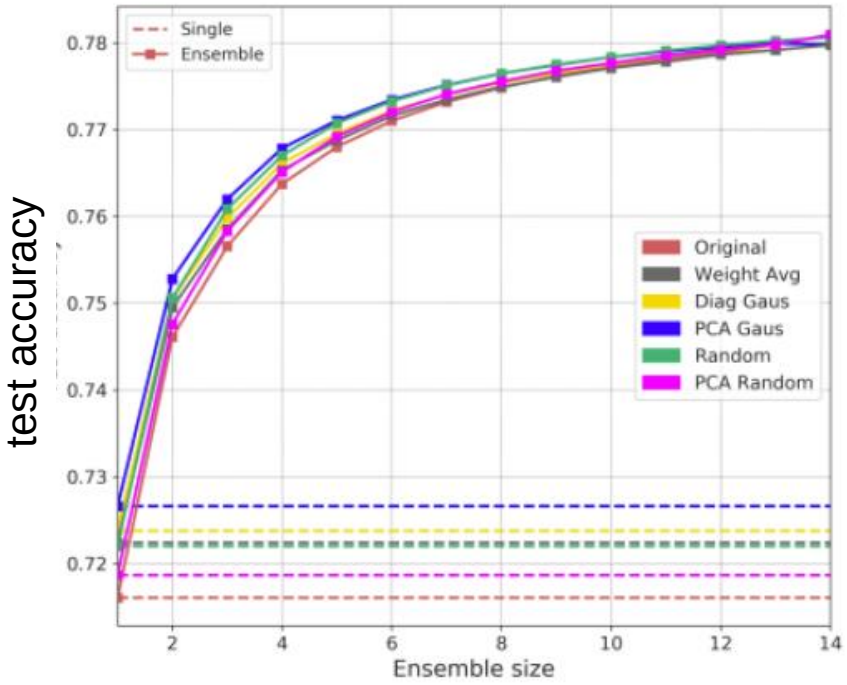


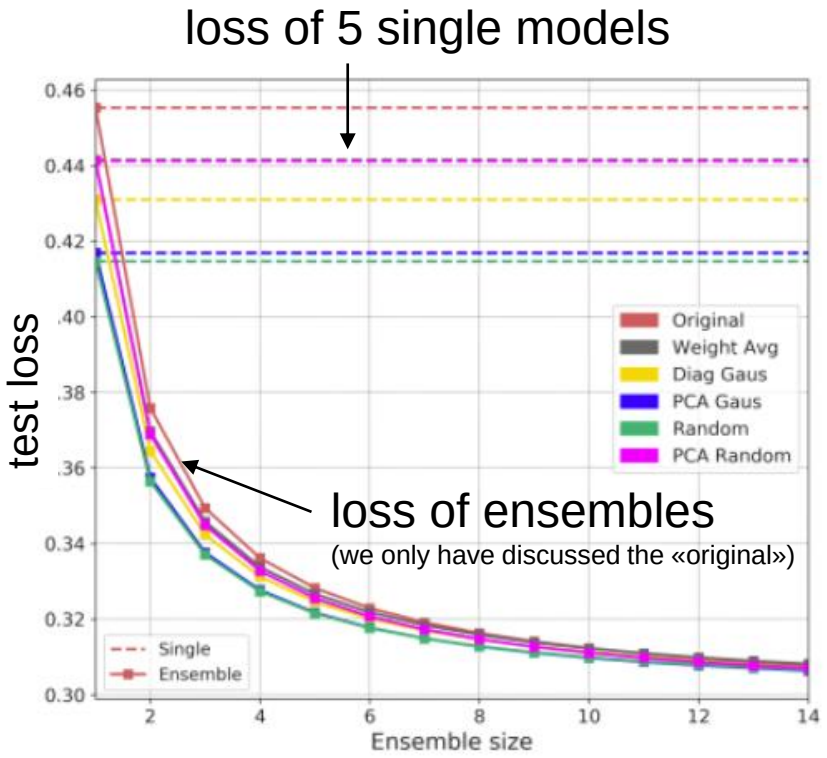
$$\log \bar{p} \geq \overline{\log p}$$

$$\log \frac{1}{M} \sum_m p_m \geq \frac{1}{M} \sum_m \log p_m$$

Deep ensembling improves prediction power







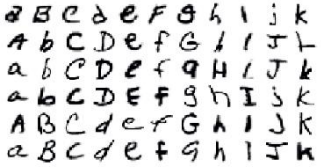
Ensembes with as few as 3 or 5 members are typically enough to achieve a perfromance gain.

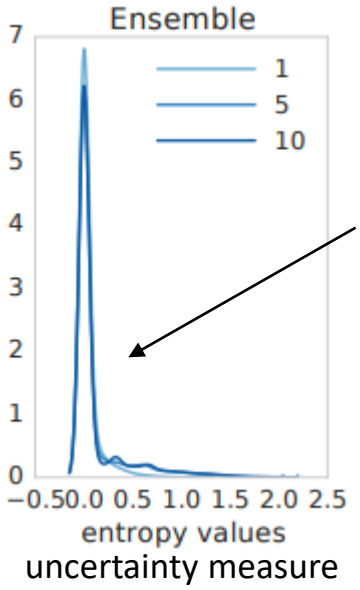
Image credit: https://wandb.ai/authors/loss-landscape/reports/Understanding-the-Effectivity-of-Ensembles-in-Deep-Learning--VmlldzoxODAxNjA

Deep ensembles improve uncertainty measures

We want that a model, that is trained on normal MNIST letter data, should provide large uncertainties when applied on novel (NOT-MNIST) letter images.

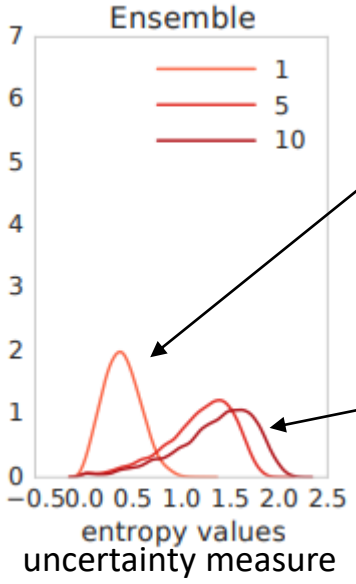
Uncertainties for prediction of normal MNIST letter images





Ensemble uncertainty for known class stay small for all sizes of the ensemble

Uncertainties for prediction of NOT-MNIST images



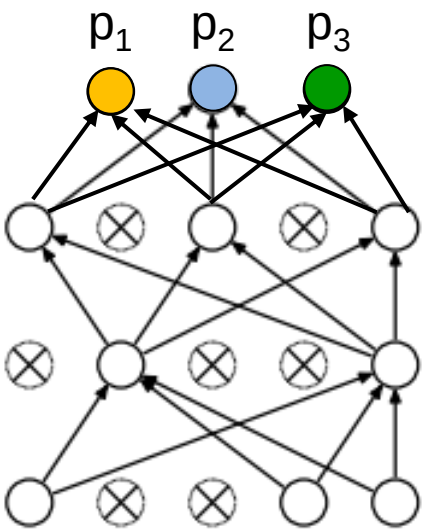
A single models shows less uncertainty for a novel class than an ensemble model

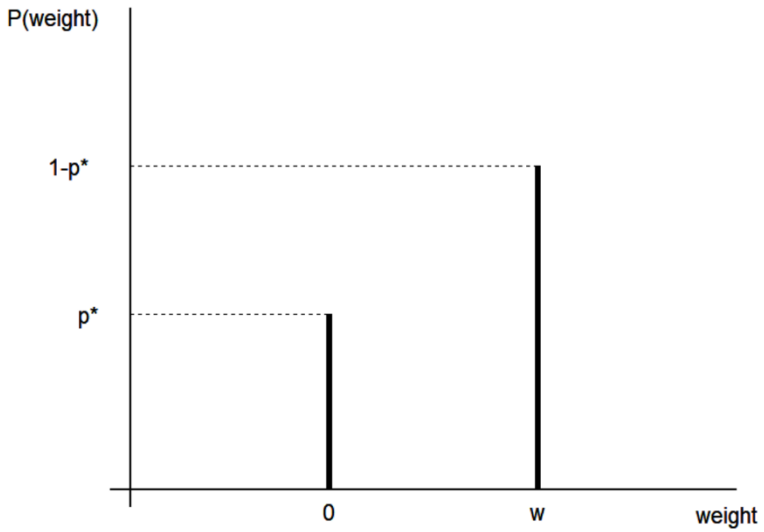
Ensemble uncertainty measures for novel classes improve with size of ensemblev

Image credits Lakshminarayanan et al.: https://arxiv.org/pdf/1612.01474.pdf

Dropout

Recall: Classical Dropout only during training

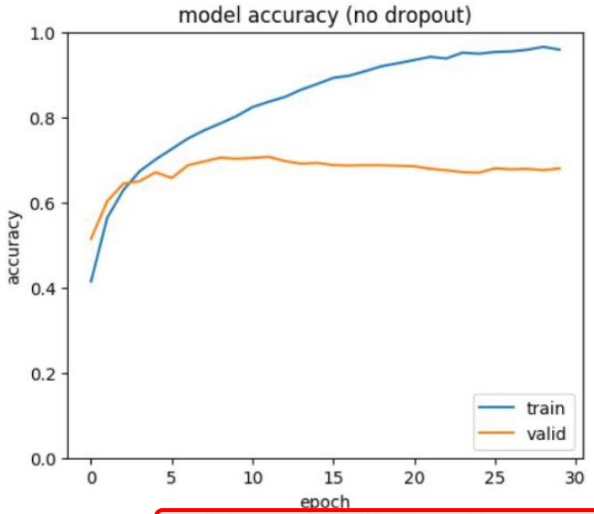


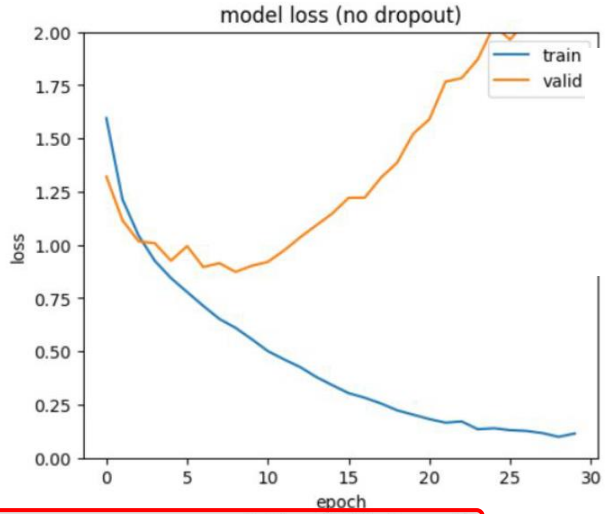


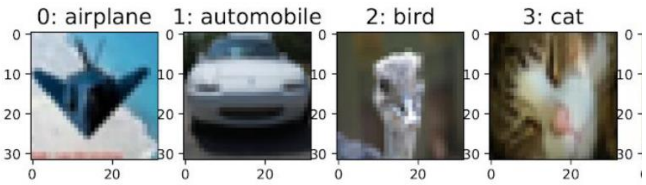
Using dropout during training implies:

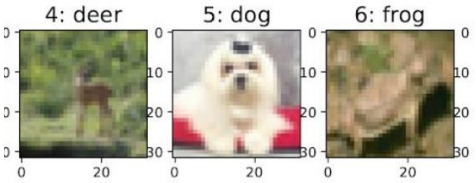
- In each training step only weights to not-dropped units are updated → we train a sparse sub-model NN
- For non-Bayesian NN we freeze the weights after training to a value $w \cdot p^*$

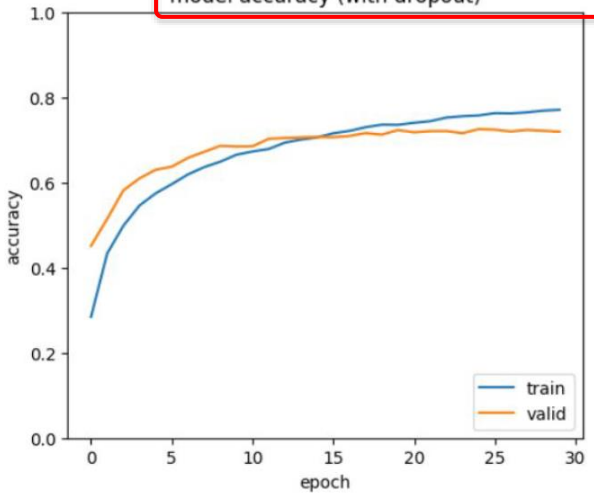
Recall: Dropout fights overfitting in a CIFAR10 CNN

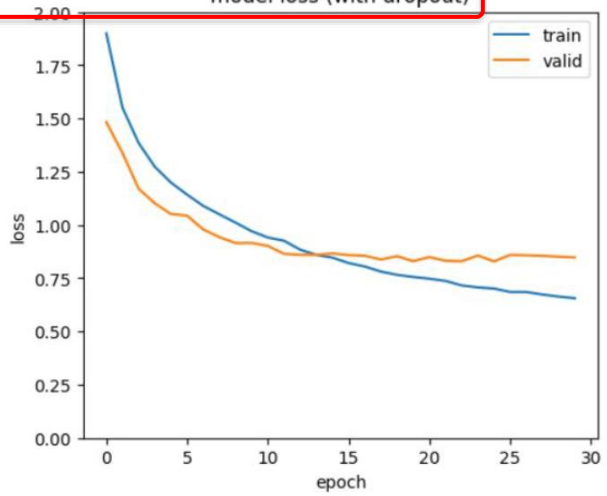










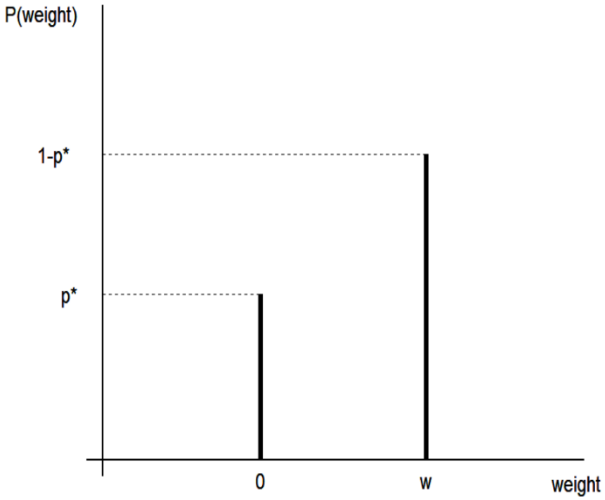


From Dropout during training to MC Dropout during test time

Bayesian NN via MC Dropout

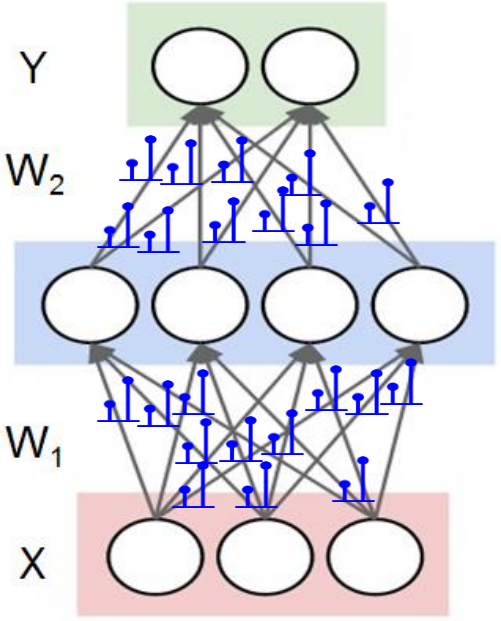
Yarin Gal et al. (2015):
Via Dropout training we learned a whole weight distribution for each connection.
We can sample from this Bernoulli-kind weight distribution by performing dropout during test time and use the dropout-trained NN as Bayesian NN.
Gal showed that doing dropout approximates VI with a Bernoulli-kind variational distribution q_θ (instead of a Gaussian).

Learned
Bernoulli-kind distribution



Dropout in
test time

MC dropout NN



Weights have
Bernoulli-kind distribution

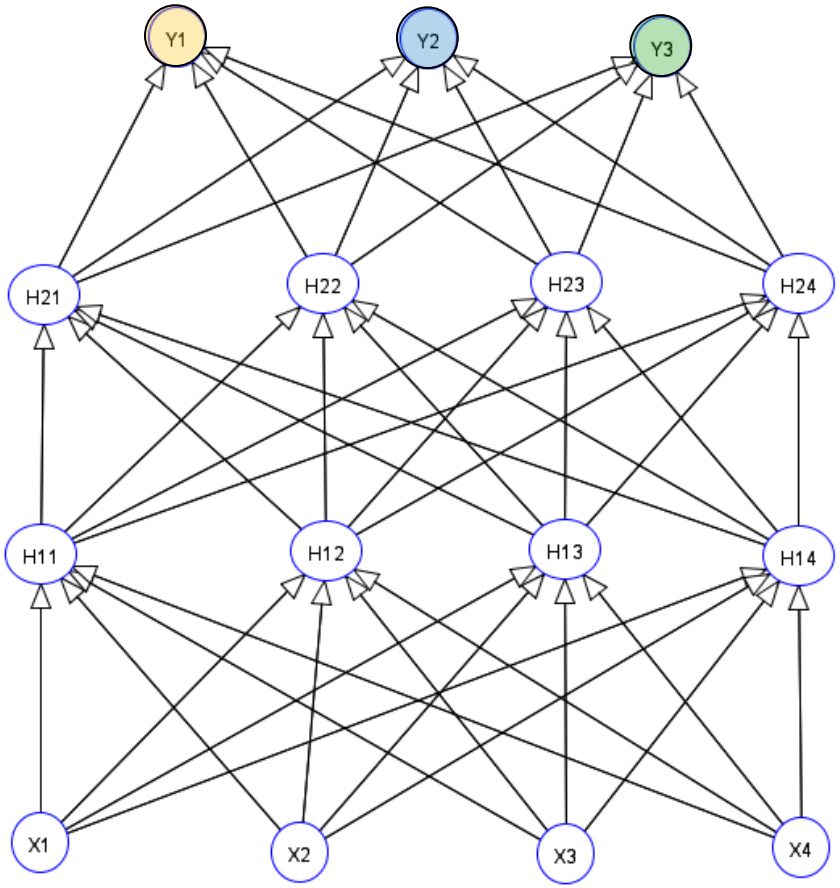
Which parameter has this q_θ?

The value w.

When using Dropout only during training

For non-Bayesian NN we freeze the weights after training to a value $w \cdot p^*$ and use then the trained NN for prediction:

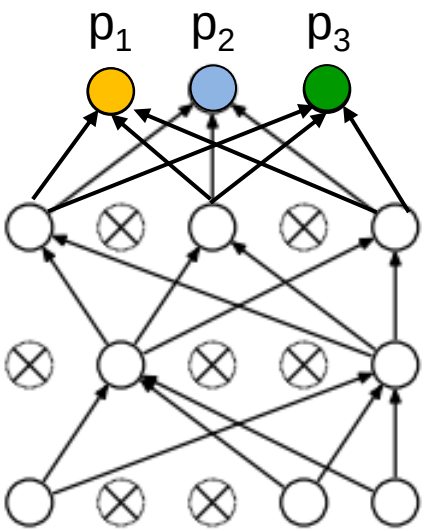
p_1=0.07 **p_2=0.91** p_3=0.02

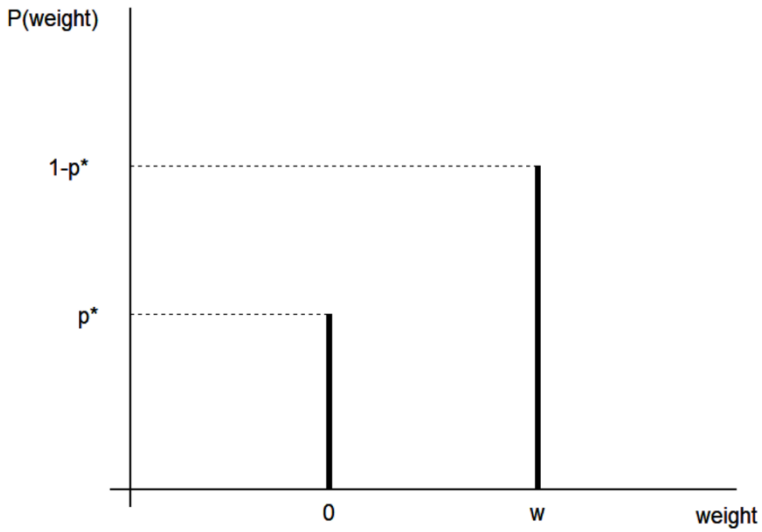


Probability of predicted class: **p_{max}**

Input: image pixel values

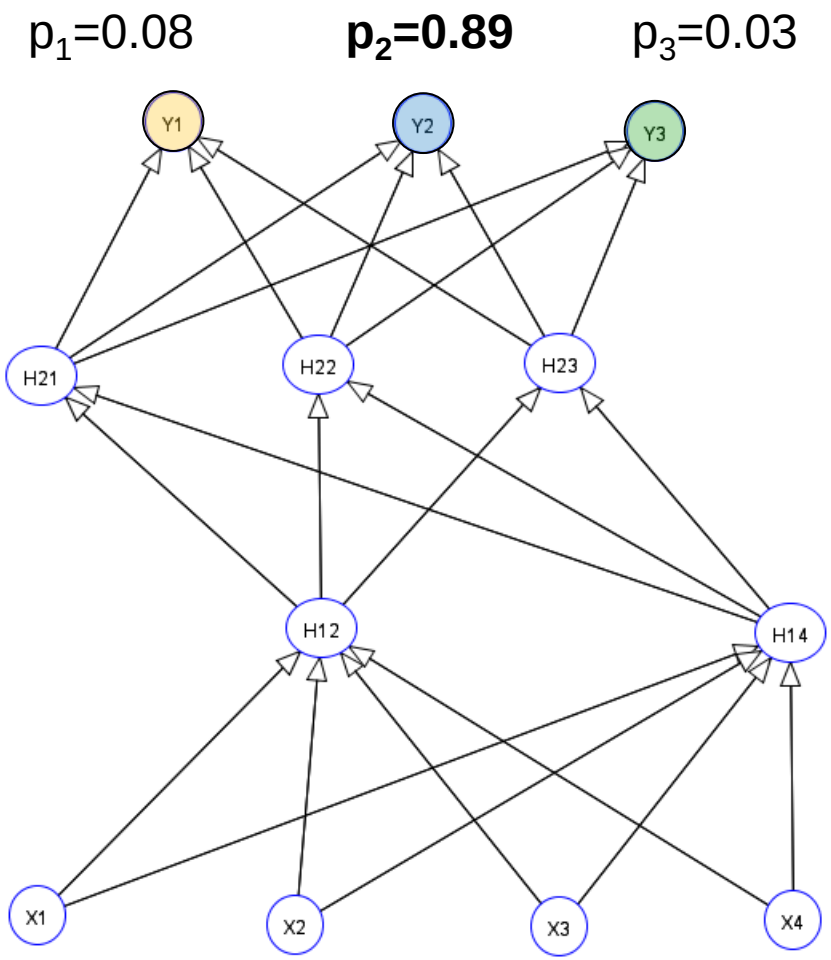
MC Dropout: we also perform dropout during test time

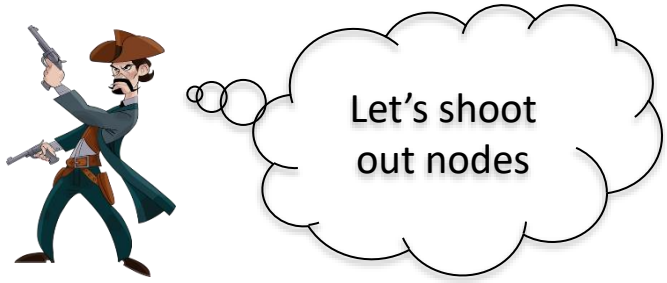




In each prediction instance we dropout a random subset of nodes, which corresponds to setting all weights starting from these nodes to zero.

MC Dropout during test time: Run 1



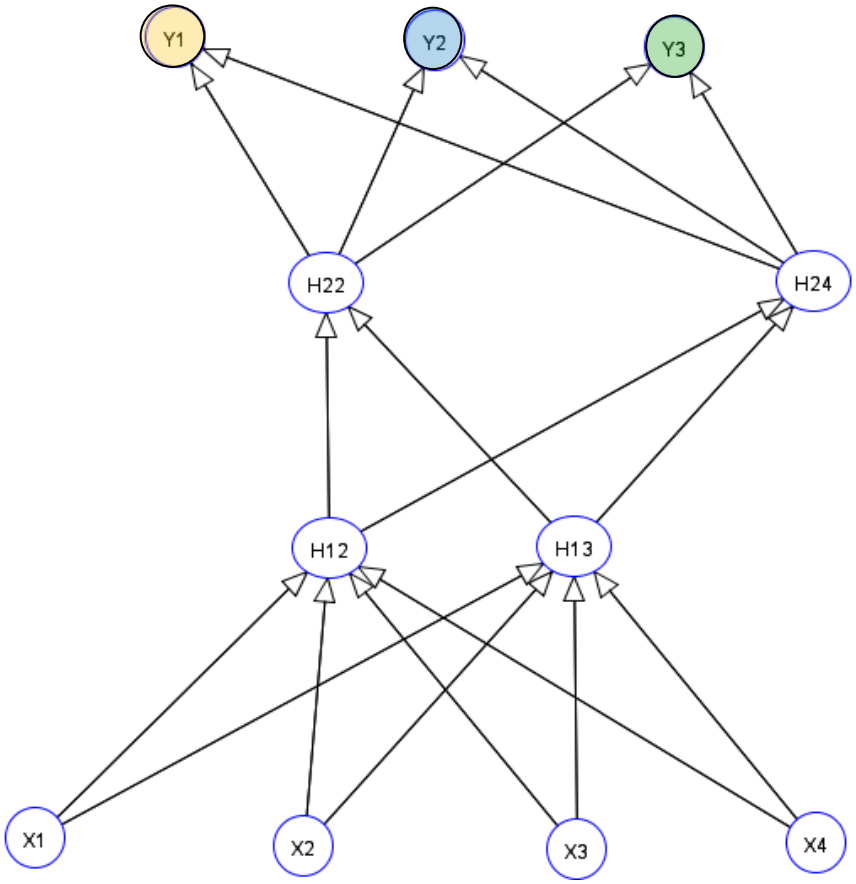


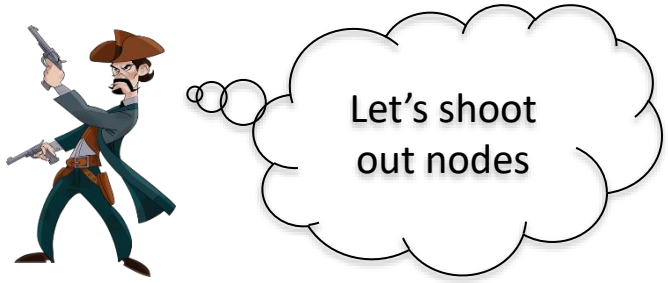
Stochastic dropout of units

Same input image

MC Dropout during test time: Run 2

p_1=0.11 **p_2=0.81** p_3=0.08

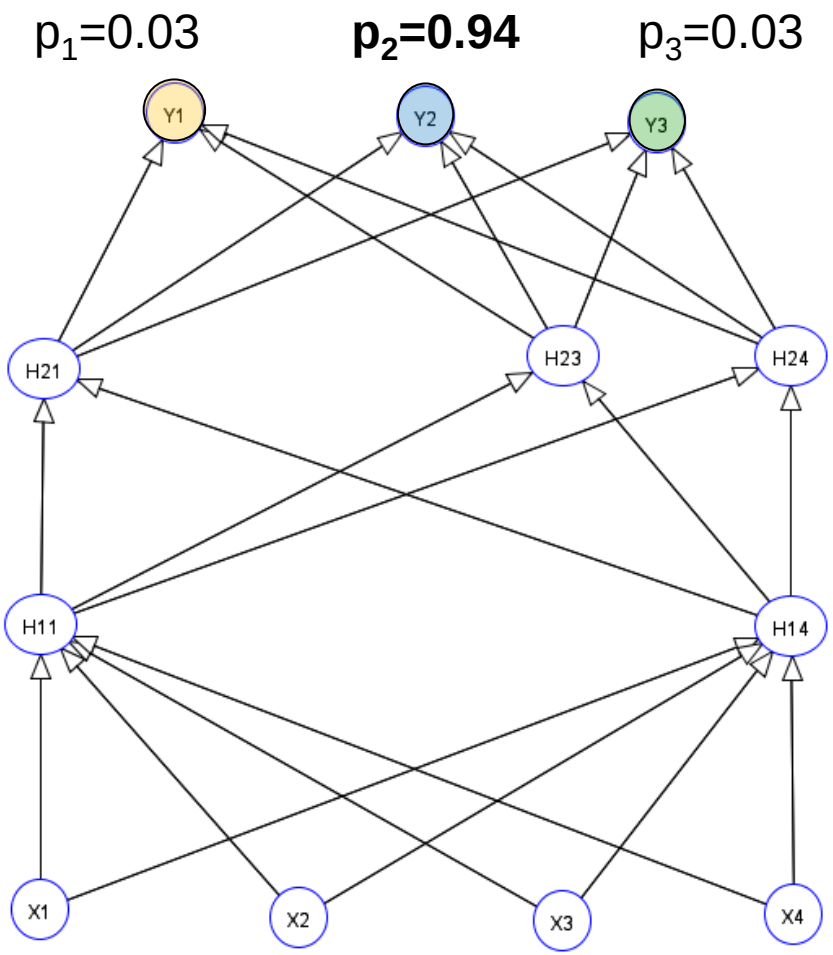


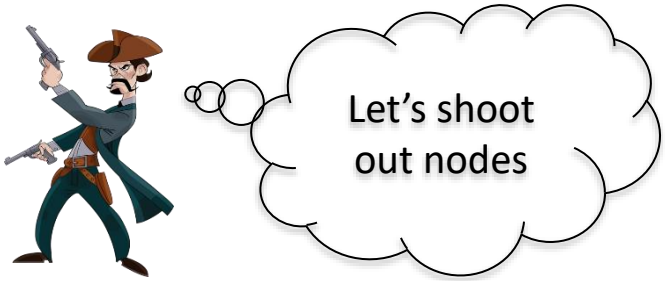


Stochastic dropout of units

Same input image

MC Dropout during test time: Run 3



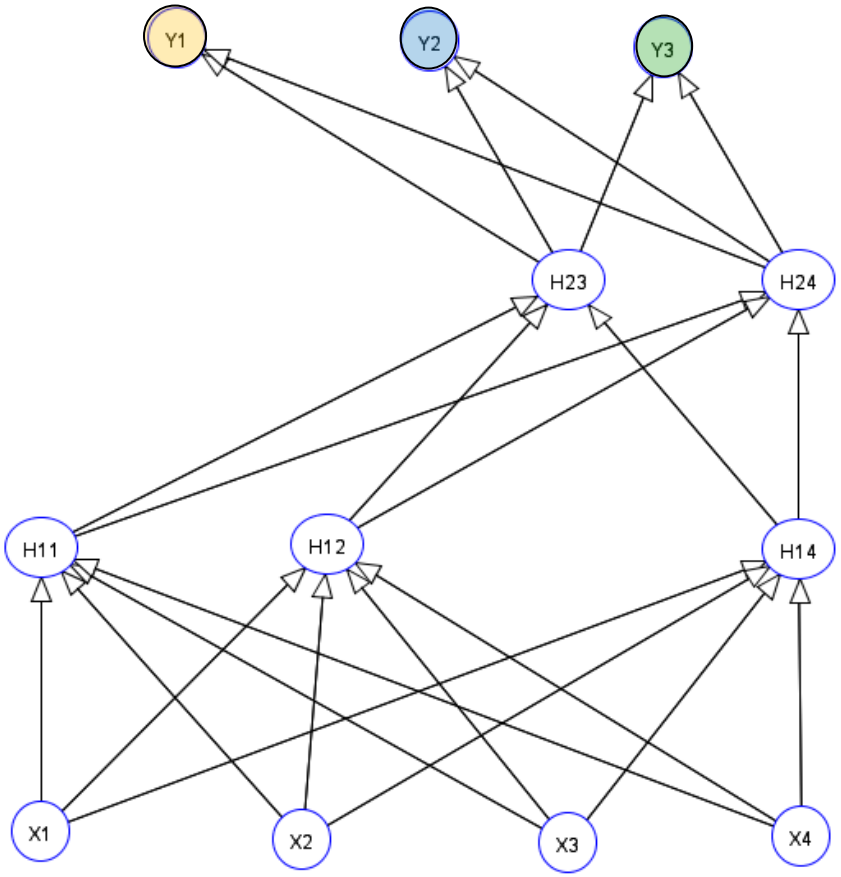


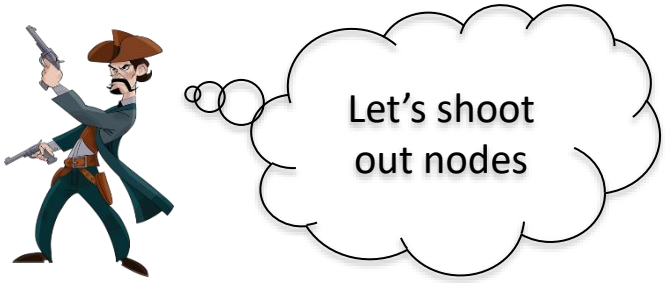
Stochastic dropout of units

Same input image

MC Dropout during test time: Run 4

p_1=0.16 **p_2=0.78** p_3=0.06



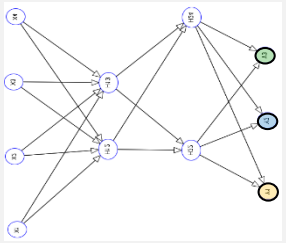


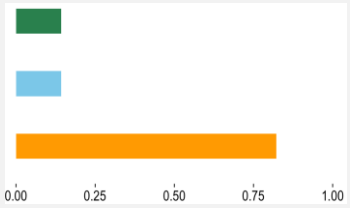
Stochastic dropout of units

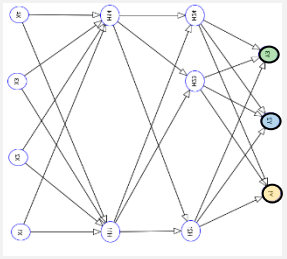
Same input image

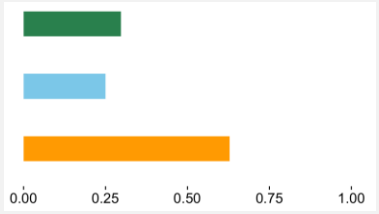
MC Dropout during test time yields a multivariate predictive distribution for the parameters

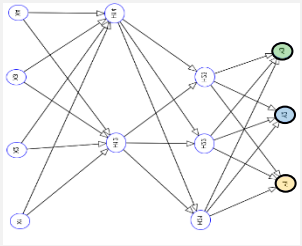
Many Dropout Runs in forward pass

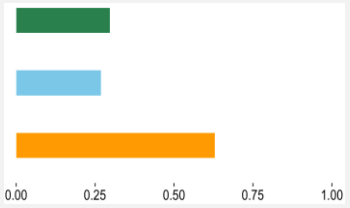




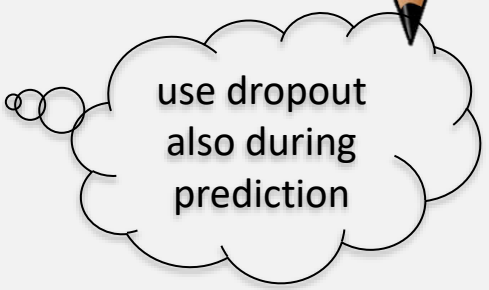


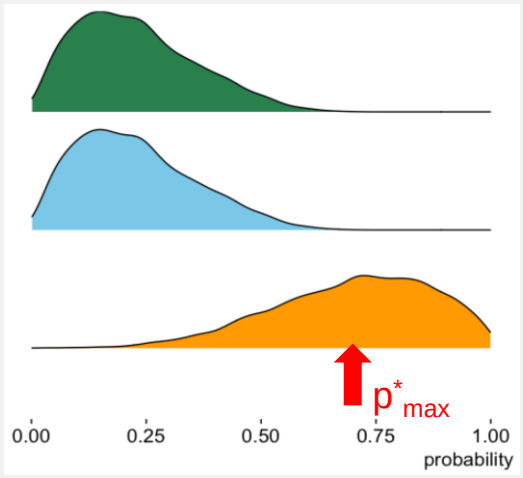






...

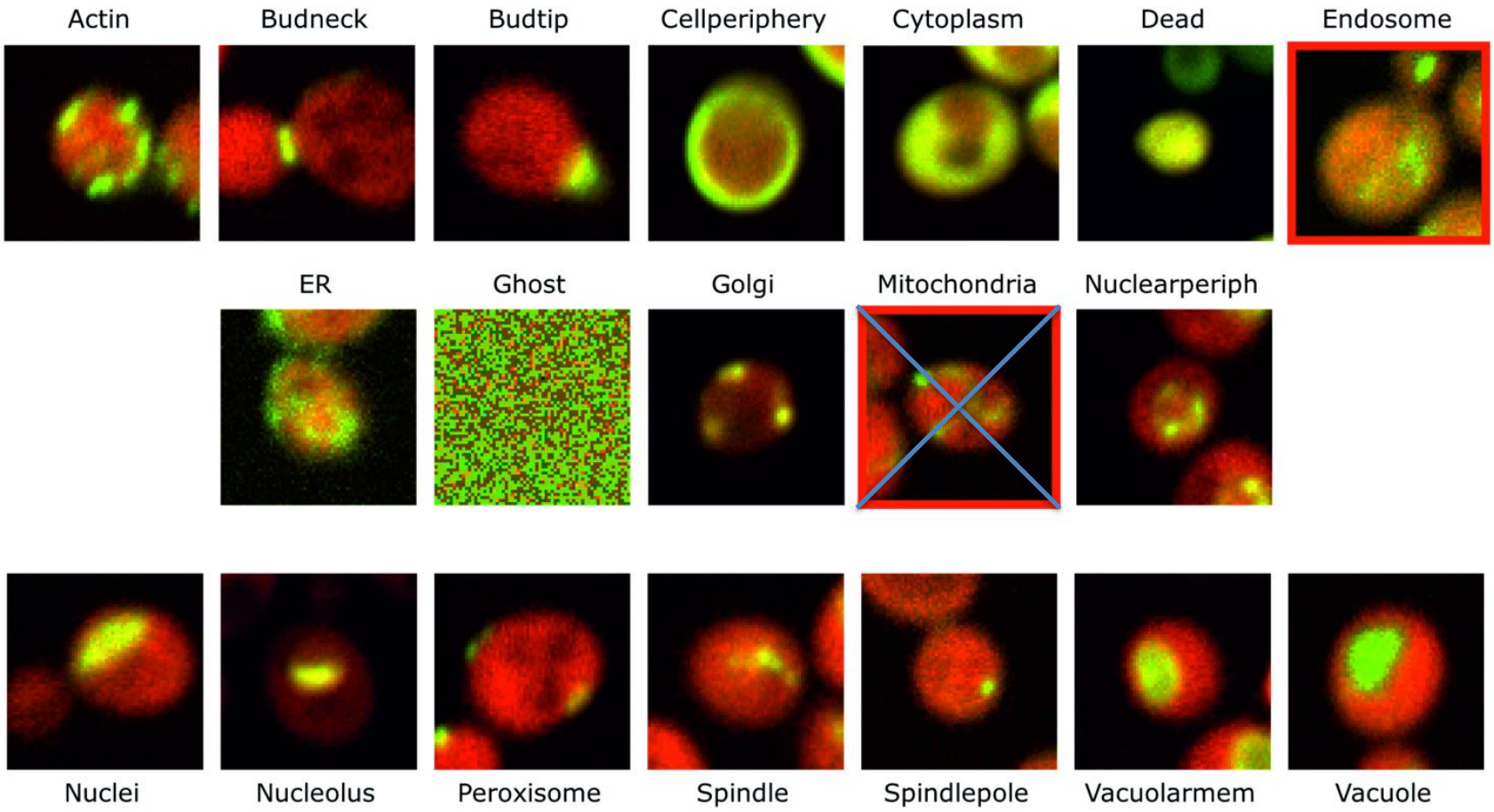




CNN predicts class “collie”
but with high uncertainty

Remark: Mean of marginal give components of mean in multivariate distribution.

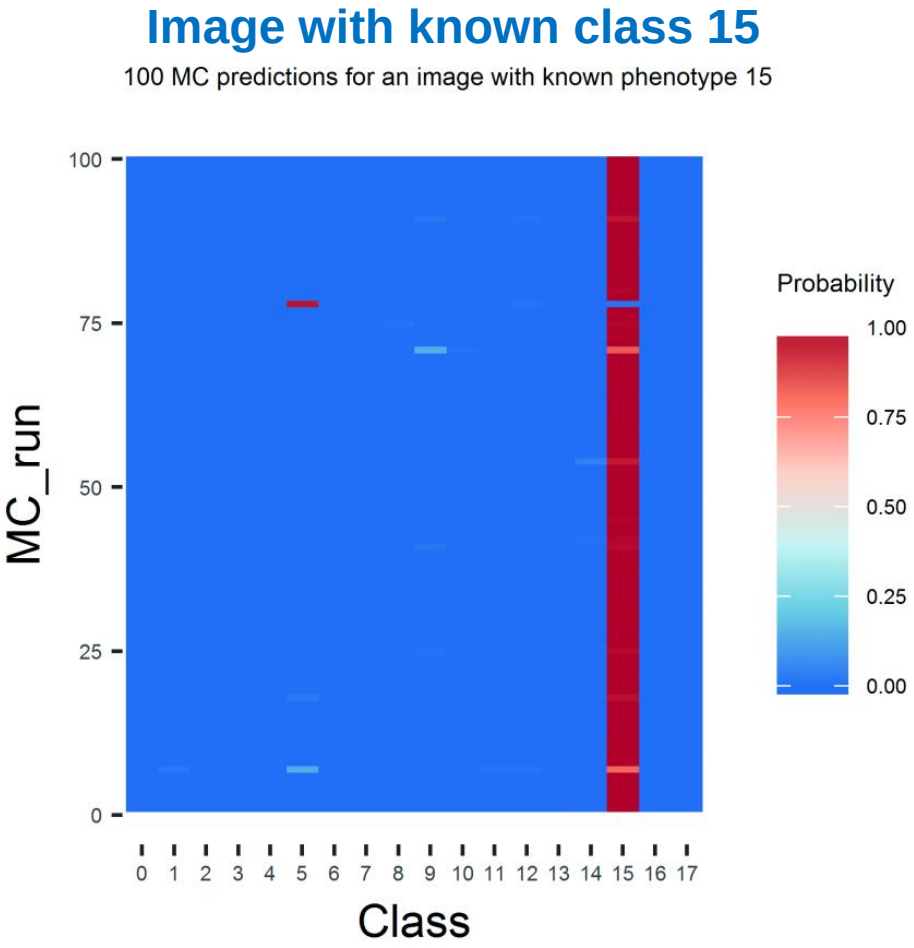
Experiment with unknown phenotype



Dürr O, Murina E, Siegismund D, Tolkachev V, Steigele S, Sick B. Know when you don't know, Assay Drug Dev Technol. 2018

Probability distribution from MC dropout runs

Image with known class 15



Probability distribution from MC dropout runs

Image with known class 15

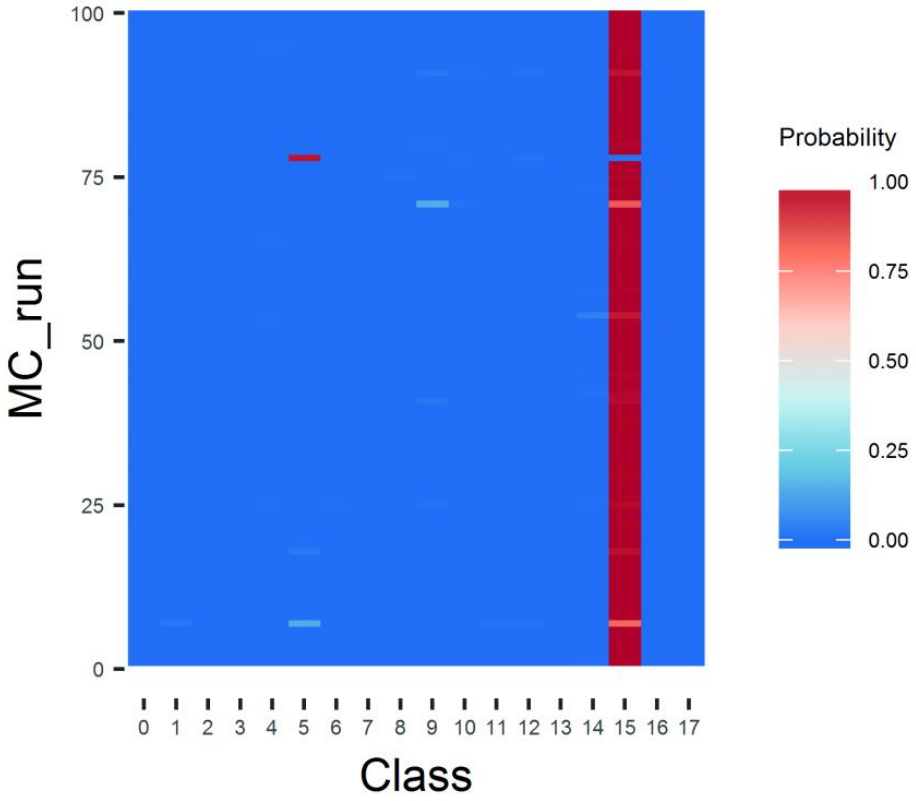
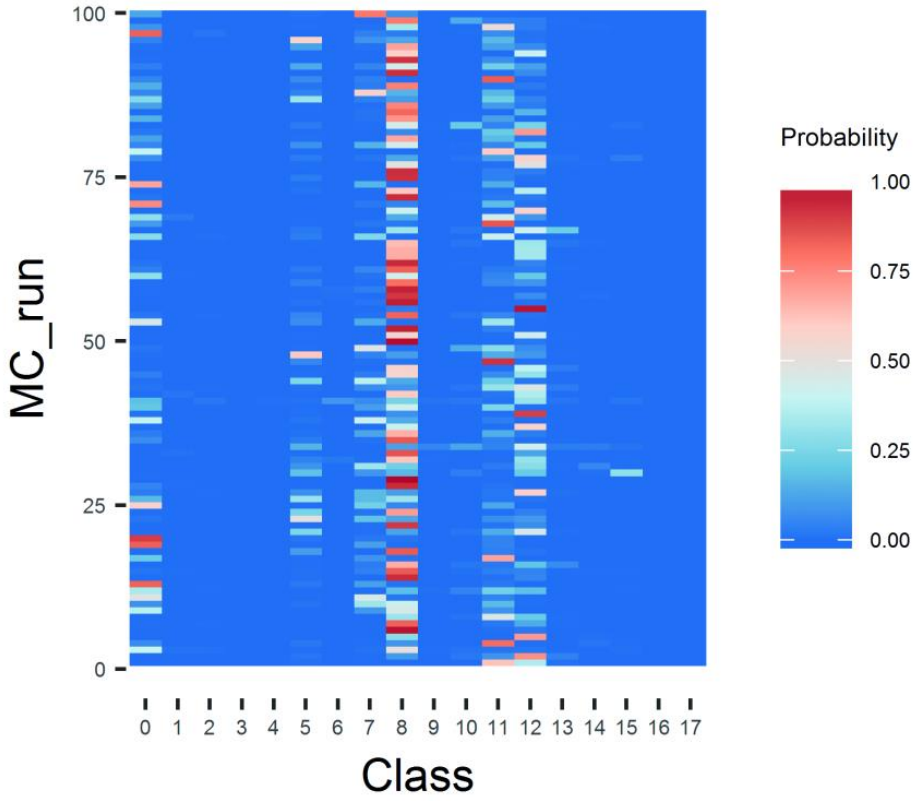
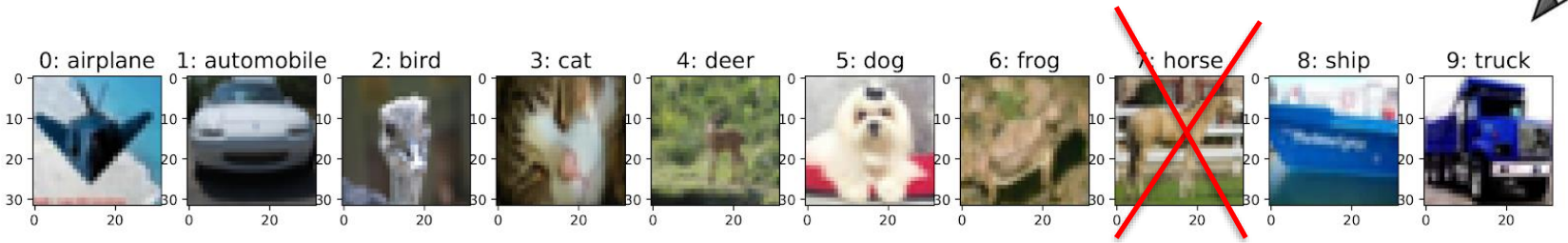


Image with unknown class



Hands-on Time

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.



Cifar10 classification case study with novel class

- Imports
- Loading and preparation of the dataset
- Non-Bayesian CNN
- **Variational Inference**
- MC Dropout
  - Getting mc dropout predictions
- Accuracy on the the known lables in the train set for all three models
  - Non-Bayesian prediction
  - Bayesian VI prediction
  - Bayesian MC prediction
  - Predicted classes for the unknown class
  - Compare the predictions for a known and unknown image

Please focus on MC Dropout

https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/20_cifar10_classification_mc_and_vi.ipynb

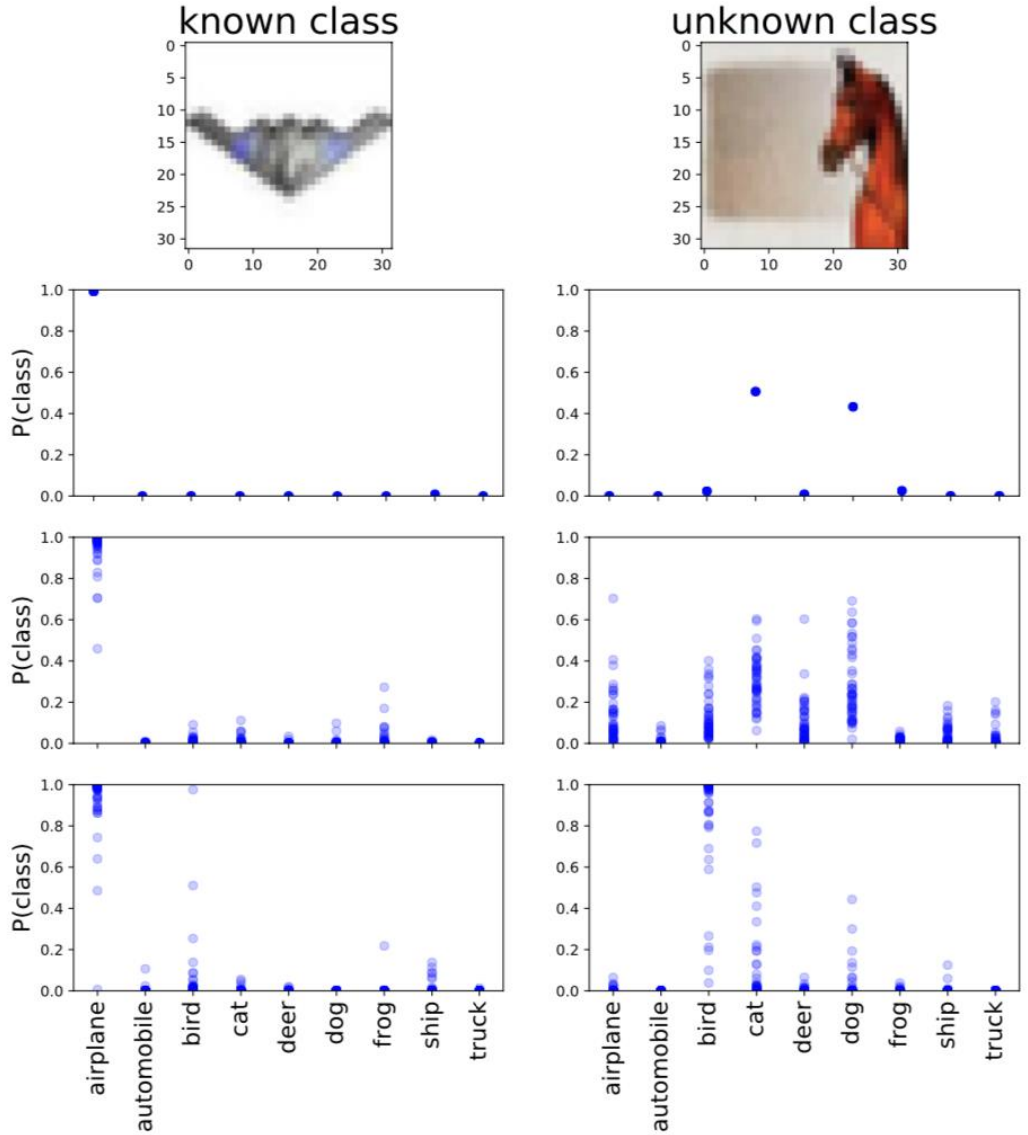
Run the Variational Inference without trying to understand it – we will care later!

Looking at the predictive distribution!

Classical CNN (no Bayes)

Gaussian VI

MC dropout



MC Dropout increases prediction performance

MC dropout is equivalent to performing *several* stochastic forward passes through the network and averaging the results. By doing so Yarin Gal was able to outperform state of the art error rates on MNIST and CIFAR-10 without changing the architecture of the used CNNs or anything else beside dropout.

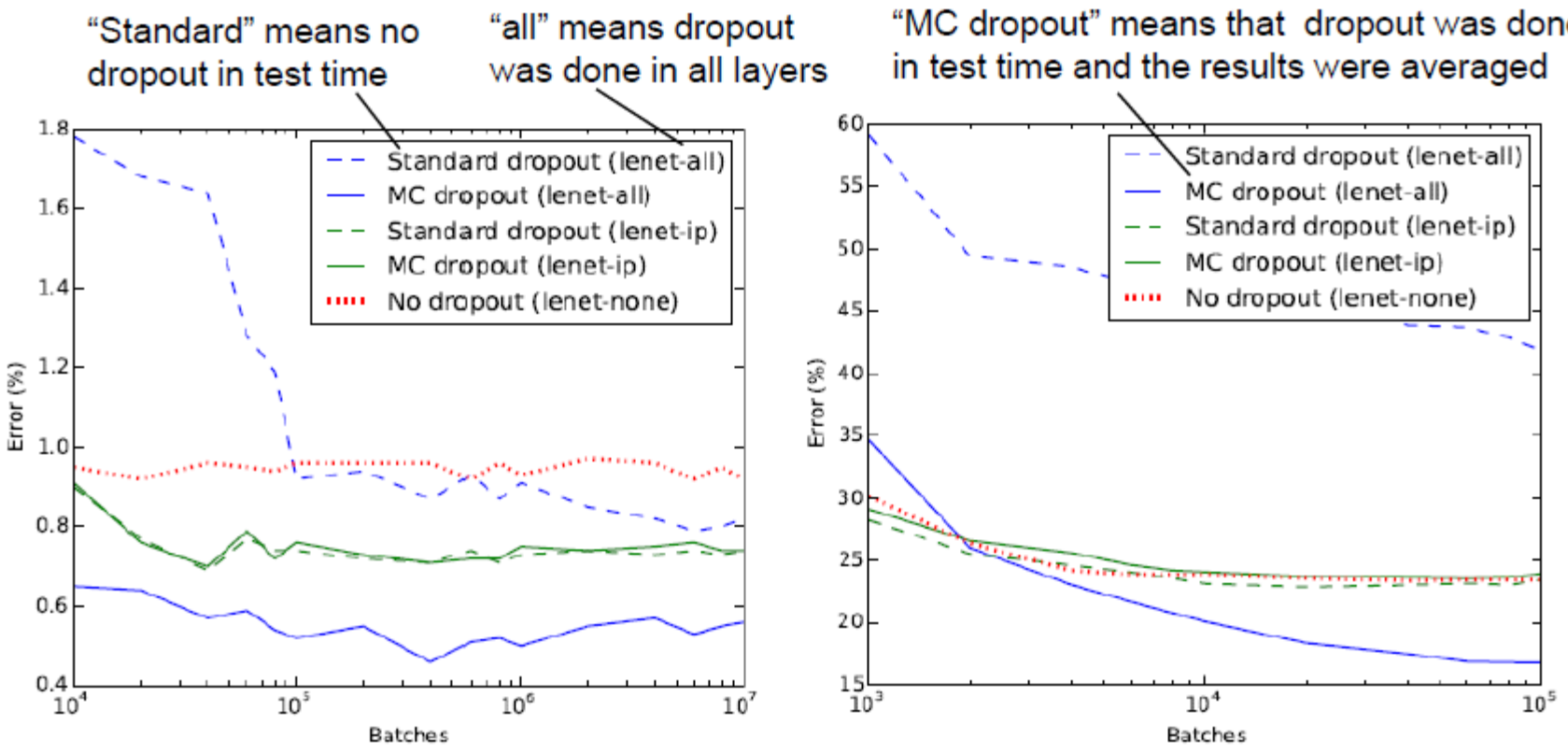


Image credits: PhD thesis of Yarin Gal: http://mlg.eng.cam.ac.uk/yarin/thesis/thesis.pdf

How does MC dropout compare with deep ensembles?

- For MC dropout we only need to compare one NN with as many parameters as a classical NN. We then average different MC dropout predictions

- For deep ensembles we need to train several NNs (typical 3 to 5) with different random initialization. We then average the predictions of these NNs

- Deep ensembles are computationally more costly but provide typically better prediction performance (and also better uncertainty measures) than MC dropout

MNIST classification

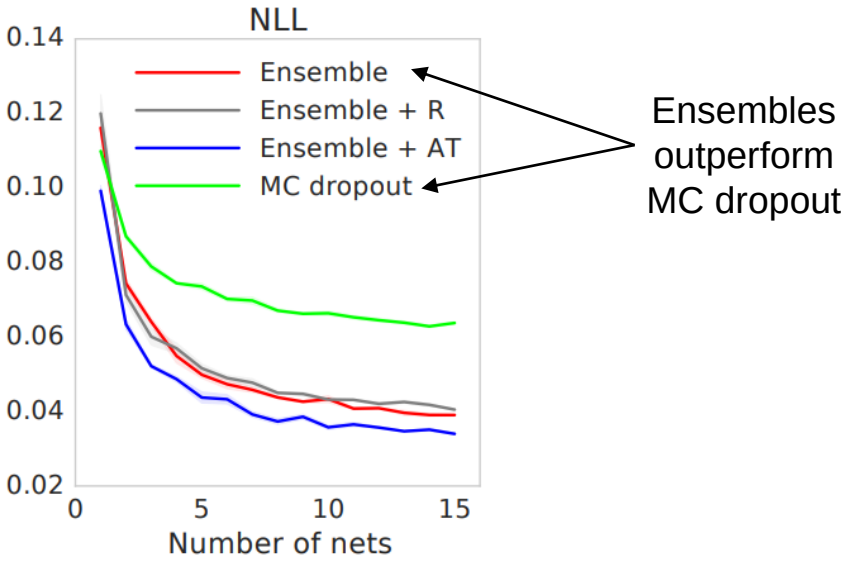
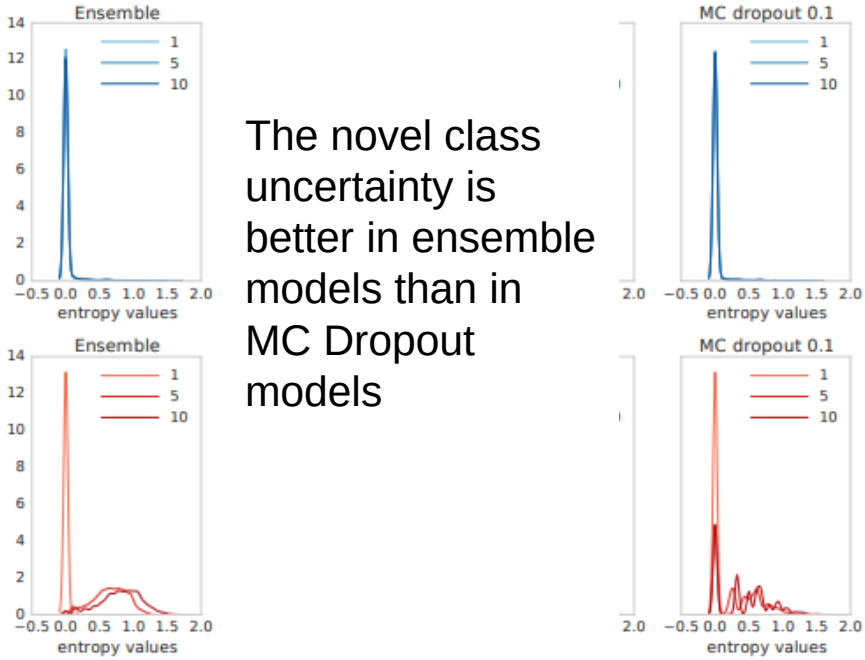
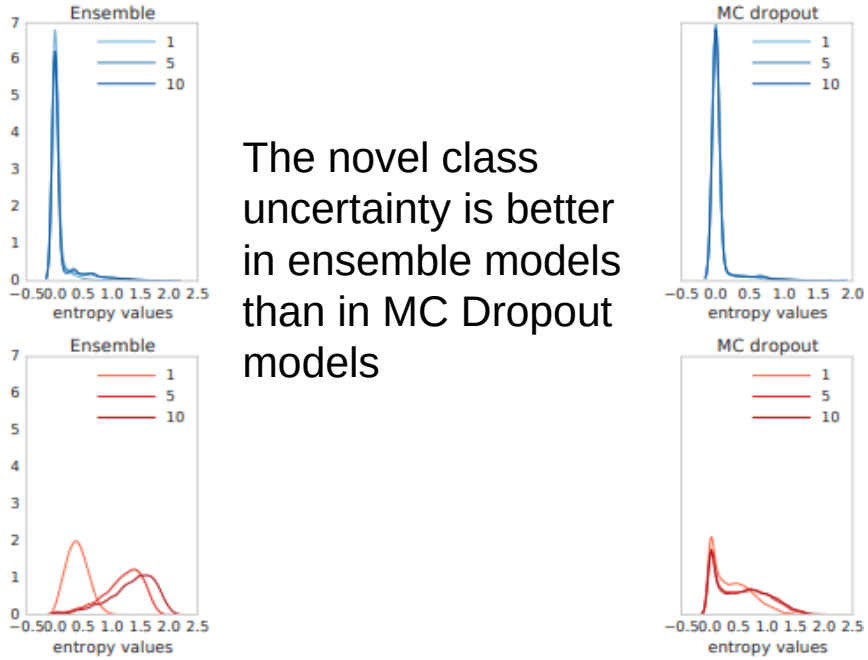


Image credits Lakshminarayanan et al.: https://arxiv.org/pdf/1612.01474.pdf

Compare MC dropout and deep ensembles uncertainties



The novel class uncertainty is better in ensemble models than in MC Dropout models



The novel class uncertainty is better in ensemble models than in MC Dropout models

(a) MNIST-NotMNIST (b) SVHN-CIFAR10

Figure 3: : Histogram of the predictive entropy on test examples from known classes (top row) and unknown classes (bottom row), as we vary ensemble size M.

Image credits Lakshminarayanan et al.: https://arxiv.org/pdf/1612.01474.pdf

Bayesian Deep Learning

Extrapolation

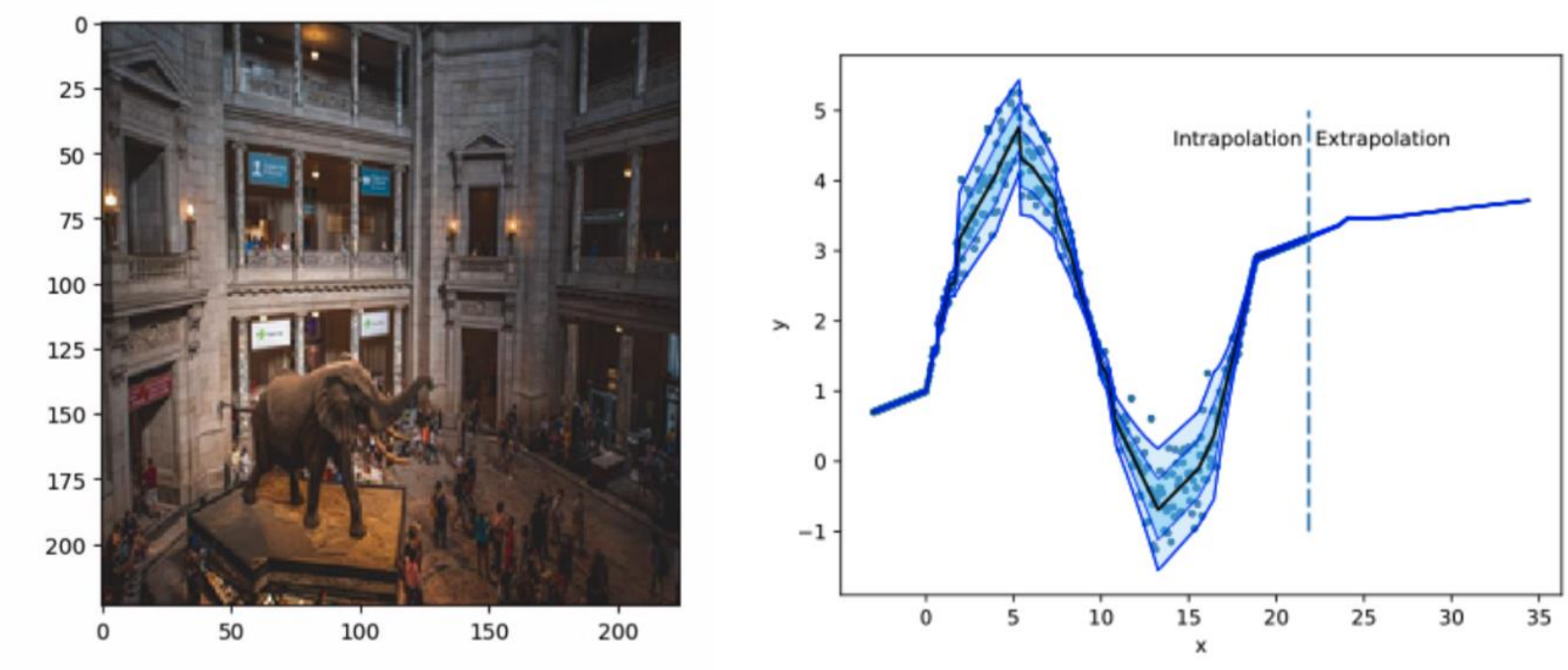
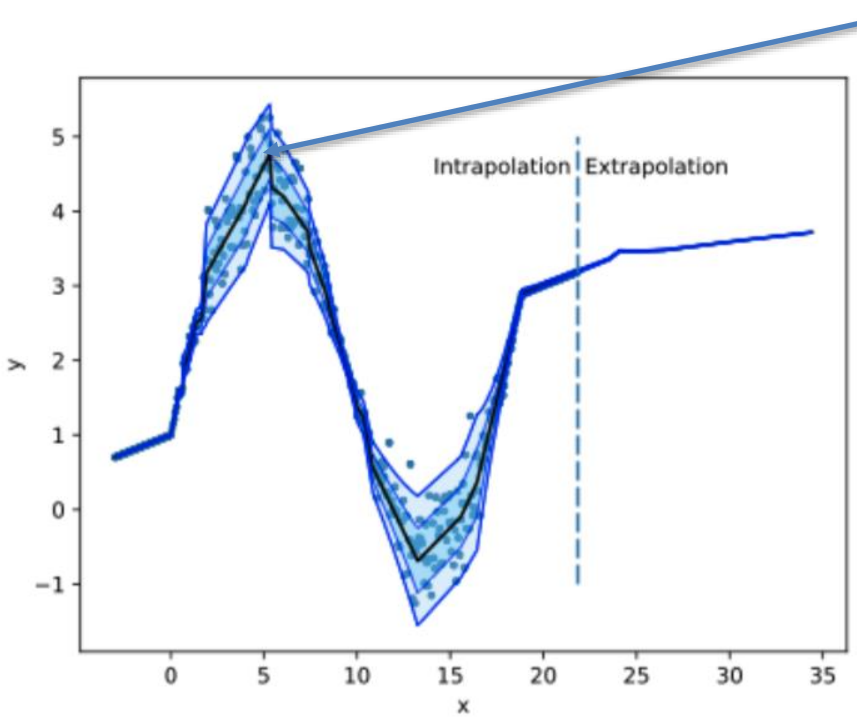


Figure 7.2 Bad case of DL. The high performant VGG16-CNN trained on ImageNet data fails to see the elephant in the room. The five highest-ranked class predictions of the objects in the image are horse_cart, shopping_cart, palace, streetcar, gondola; the elephant is not found! This image is an extrapolation of the training set. In the regression problem on the right side of the dashed vertical line, there's zero uncertainty in the regions where there's no data (extrapolation).

Aleatroic vs. Epistemic Uncertainty

Much spread in data, aleatroic (from "Alea Acta est"))



Die Würfel sind gefallen!

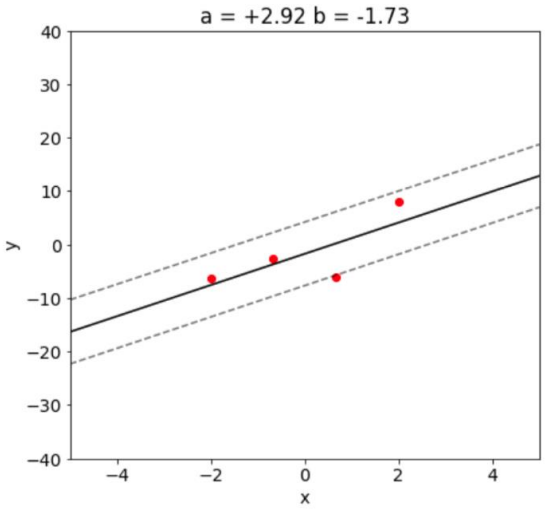
We understand a dice perfectly (no epistemic uncertainty), but still there is a aleatoric uncertainty about the the number that will show up next when rolling the dice.

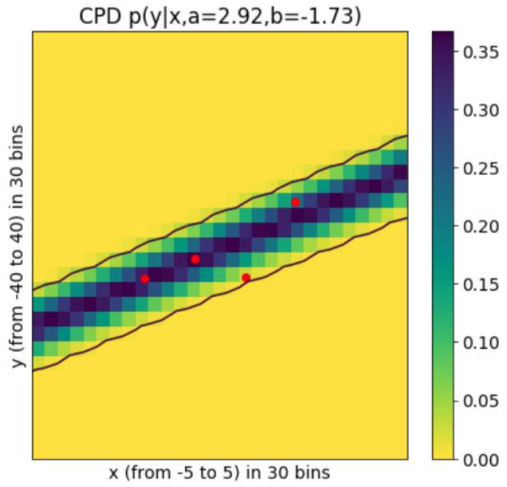
- *Aleatoric* uncertainty is due to the uncertainty, that is inherent in the data.
- The uncertainty when leaving the 'known ground' is called *epistemic* uncertainty.

We use Bayesian methods to model the epistemic model uncertainty, that is due to the uncertainty of the estimated model parameters, called weights in NN models.

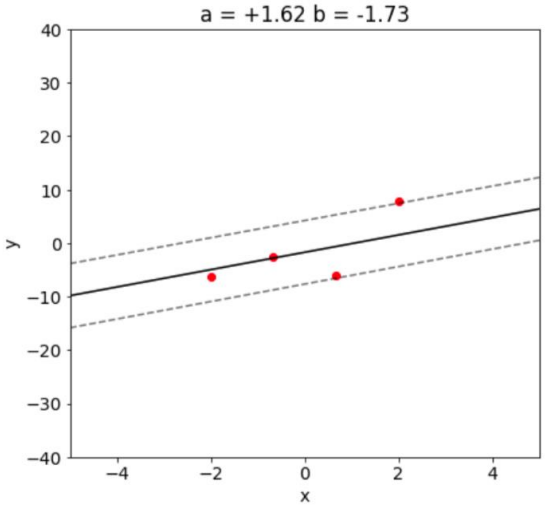
Bayes the hackers' way

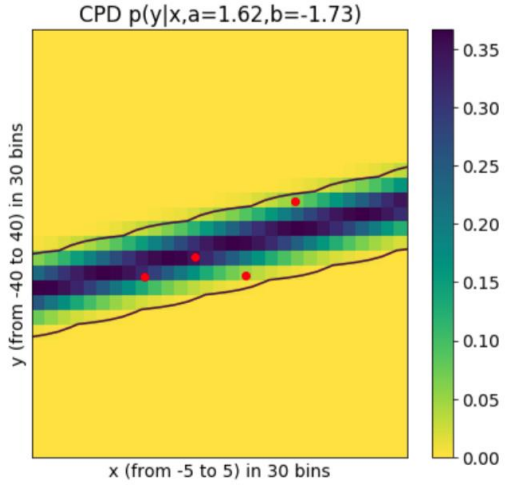
Let's look at good old linear regression to understand the gist of the Bayes idea. Assume $\sigma = 3$ to be known.





MaxLike Solution

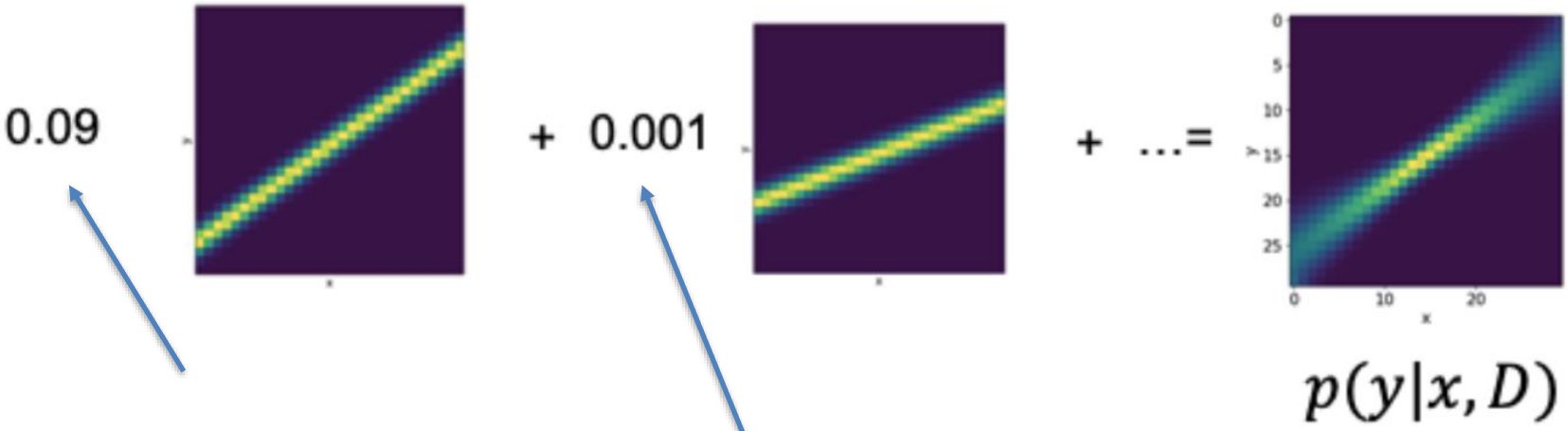




A bit off the MaxLike Solution

Combining different fits

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



Hightest weight for MaxLike

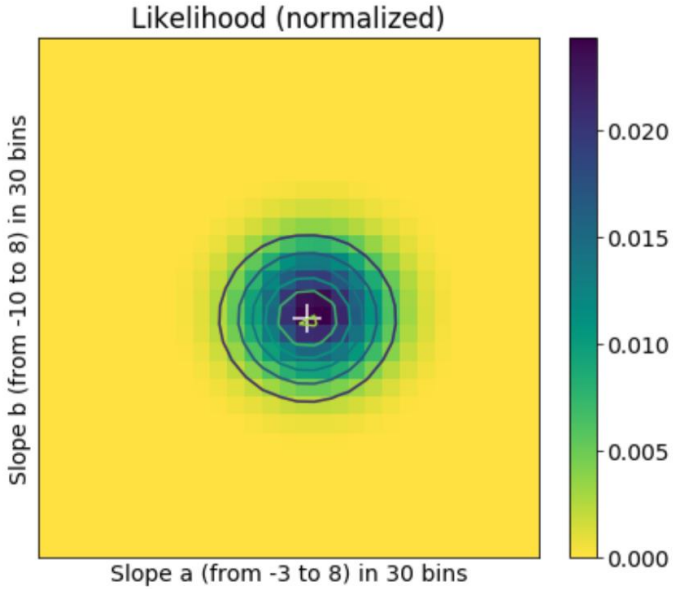
Less likely solutions get a smaller weight

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

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

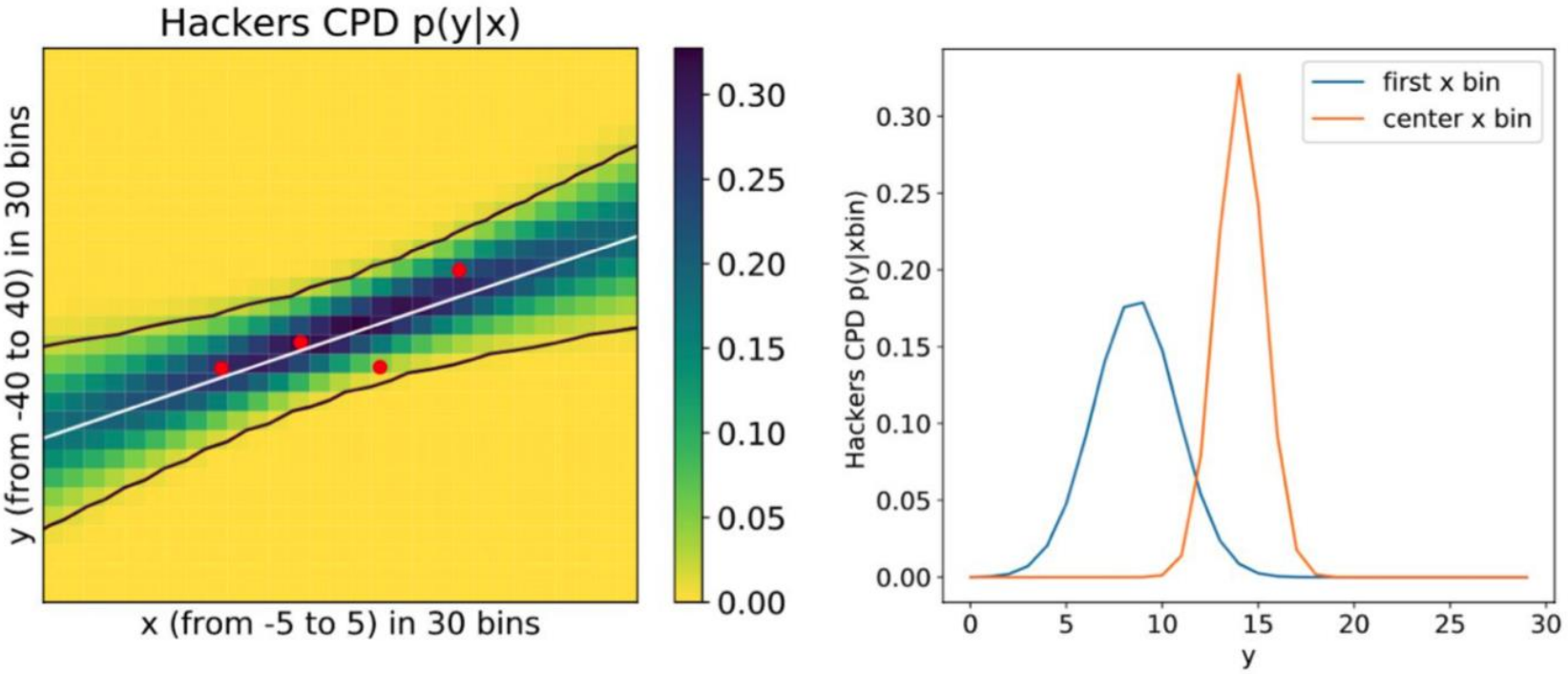


Figure 7.6 The predictive distribution for the Bayesian linear regression model, trained with the four data points shown on the left side by the color-code and on the right as conditional distribution at two different *x* positions. You can clearly see that the uncertainty gets larger when leaving the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

What have we done? (Comparison with MaxLike)

- Bayes the hacker's way (averaging over many)

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

  - For normalization, we calculated likelihood

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

  - at nbins^2 positions (here 900)

- Maximum likelihood (just take the single best)
  - Instead of just using the max likelihood method

$$\hat{a},\hat{b} = \mathrm{argmax}(p_{\mathrm{norm}}(D|(a,b))) = \mathrm{argmax}(p\,(D|(a,b))$$

  - The sum reduces to a single term

$$\mathrm{p}(y|x,D) = p\left(y\middle|x,\left(\hat{a},\hat{b}\right)\right)$$

What have we done? (Abstraction with (a,b)→ w)

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

- Or with the weights *w*
  - $\mathrm{p}(y|x,D) = \sum_w p(y|x,w) \cdot p_{\mathrm{norm}}(D|w)$)
  - $\mathrm{p}(y|x,D) = \int_w p(y|x,w) \cdot p_{\mathrm{norm}}(D|w)\ dw$ for continuous weights

- The weights now have a distribution, they are random variables as well.

- We choose
  - $p_{\mathrm{norm}}(D|w) = \frac{p(D|w)}{\sum_w p(D|w)}$
  - We treated the weights as random variables
  - It turns out that we where not the first to do so…

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*

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

  - $p(\theta|D)$ 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

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.

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

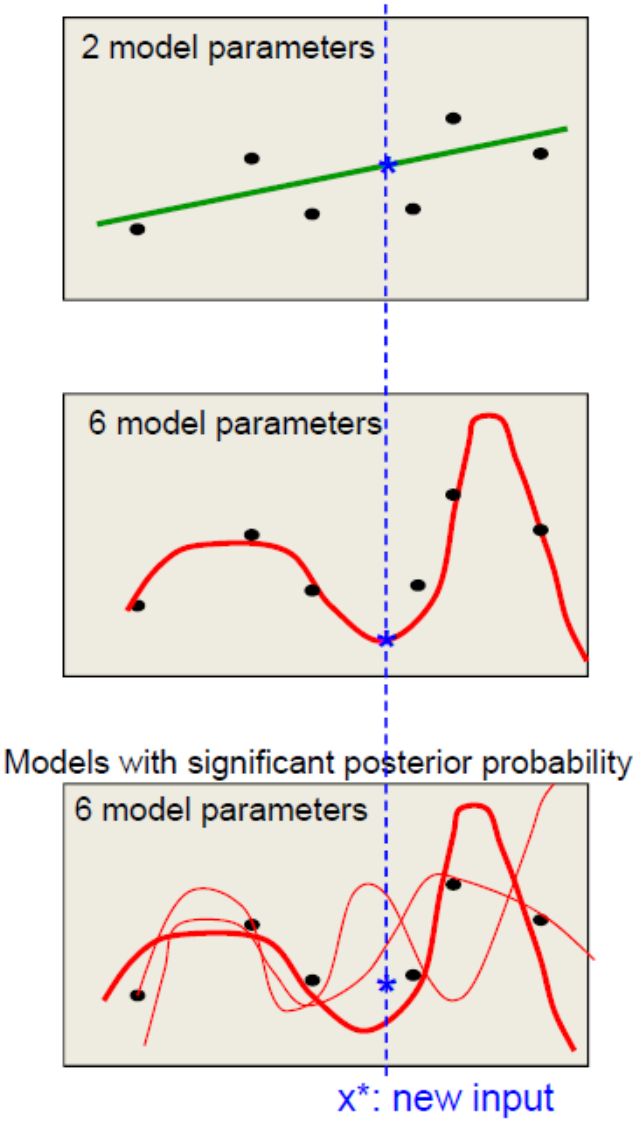


Image credits: Hinton coursera course

The Bayesian Mantra (say it loud)

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

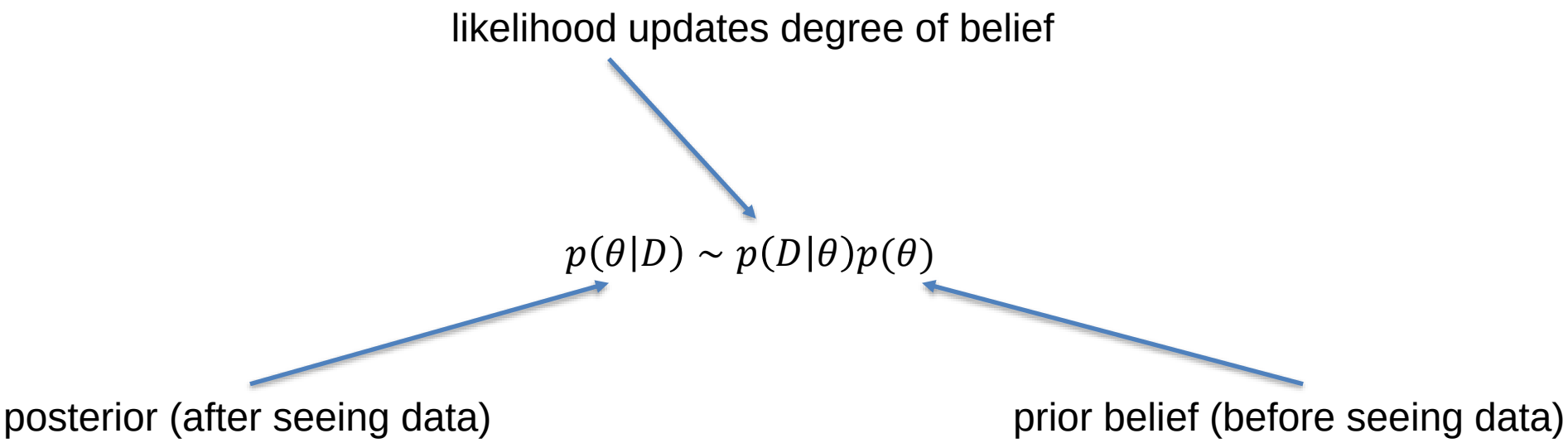
"The posterior is proportional to the likelihood, times the prior"

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

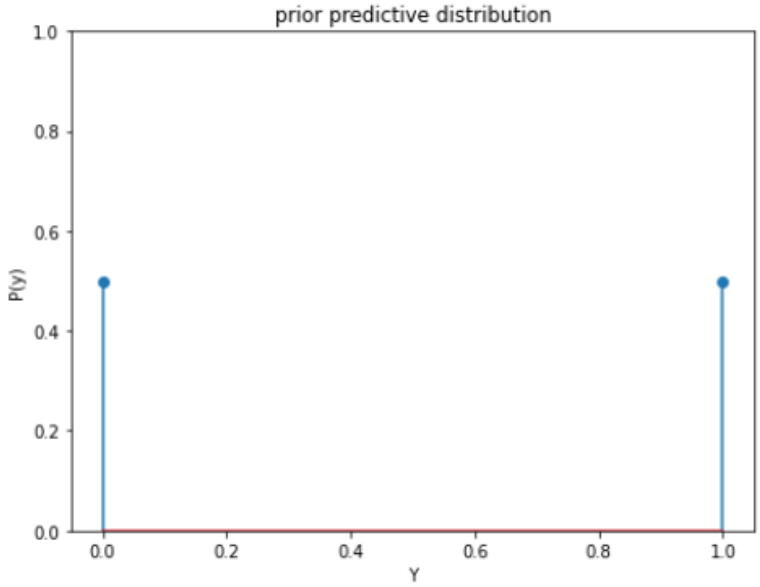


Example Coin Toss

Head or tail?

$Y_i \in \{0, 1\}$, here 0 stands for tail and 1 for head
$Y \sim \text{Ber}(\theta)$, with 1 parameter $\theta = P(Y = 1)$

For a fair coin $\theta = 0.5$



But how to know if a coin is fair?

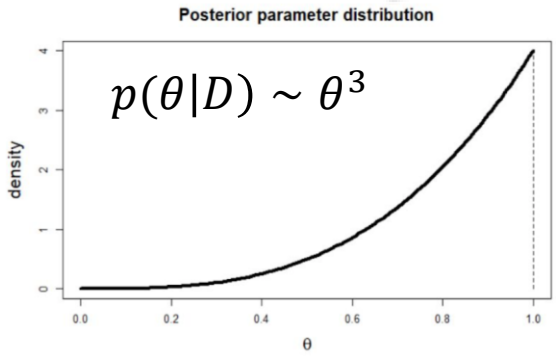
Analyzing a Coin Toss Experiment

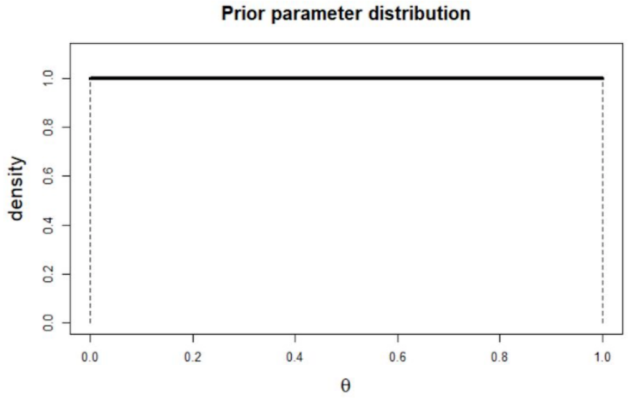
- We do an experiment and observe 3 times head → D='3 heads'
- θ parameter for the Bernoulli-distribution (probability of head)
- Before the experiment we assume all value of θ are equally likely $p(\theta) = \text{const}$
- Calculate likelihood $p(D|\theta) = p(y=1) \cdot p(y=1) \cdot p(y=1) = \theta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

After **Before**

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

θ^3





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

Posterior Predictive Distribution

Posterior predictive distribution via marginalization over θ

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

For unconditional (no x)

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

Prob. of certain result given θ

Posterior prob. of certain value θ

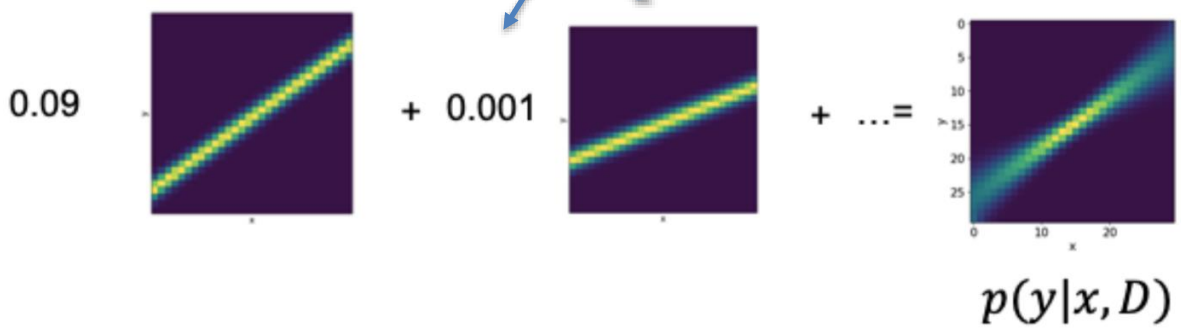
For coin interested in $\mathrm{p}(y = 1|D)$ prob. for head

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

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

Summary

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

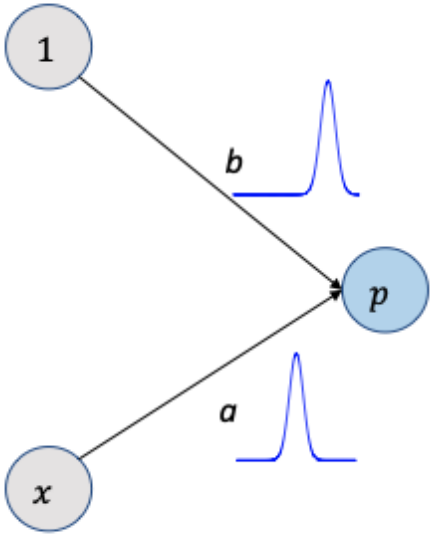


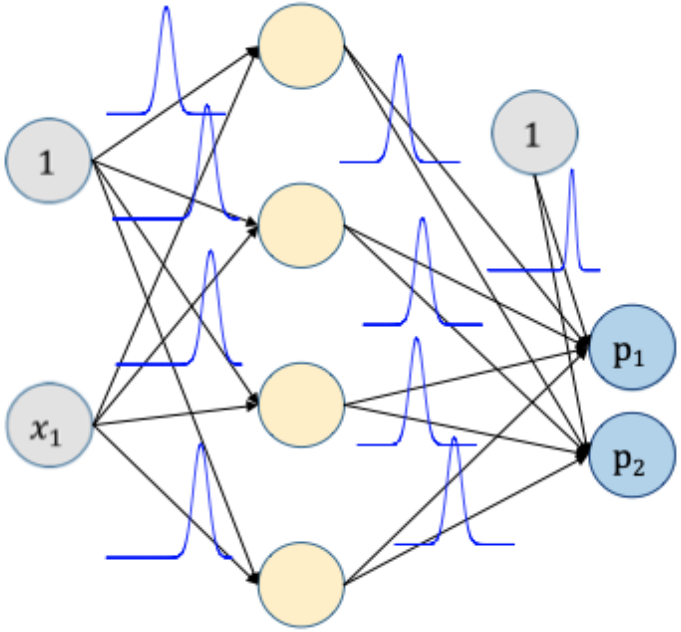
- Not just a single solution
  - "Marginalizing instead of optimization"
- Bayes usually achieves
  - Better uncertainty estimates
  - Better predictive performance

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)
- It's possible to calculate $p(D|\theta)p(\theta)$ for a few values fast
- But calculate $\sum_\theta p(D|\theta)p(\theta)$ is impossible for high-dimension θ
  - We need to calculate 10 values per dimension, then 10^dimensions evals

Approximations

- MCMC (only for very small NN)
  - Sample from $p(\theta|D)$ with knowledge of $p(D|\theta')p(\theta')$ / $p(D|\theta'')p(\theta'')$
- Variational Inference VI
  - Replace $p(w|D)$ with an approximation e.g. $N(\mu, \sigma)$ and determine μ, σ
- MC-Dropout
  - Dropout also during predictions (magically) samples from a posterior

Variational Inference

The principle of VI

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

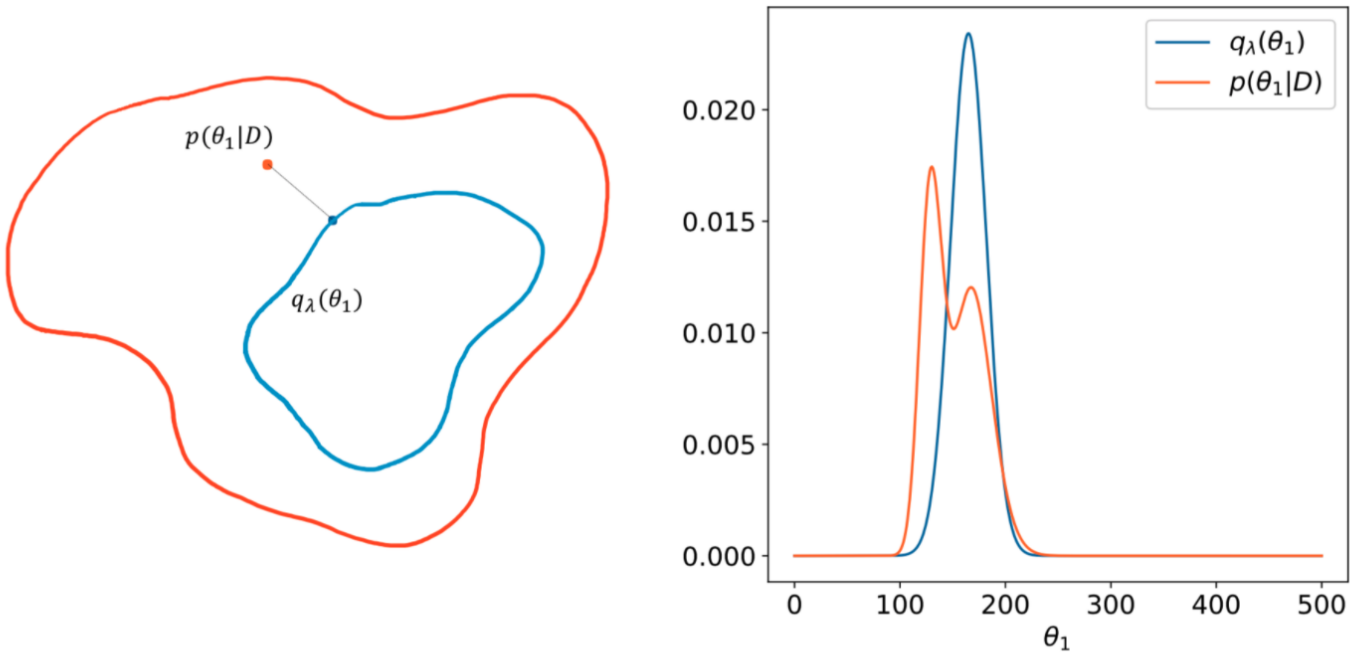


Figure 8.3 The principle idea of variational inference (VI). The larger region on the left depicts the space of all possible distributions, and the dot in the upper left represents the posterior $p(\theta_1|D)$ (corresponding to the dotted density on the right panel). The inner region depicts the space 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`

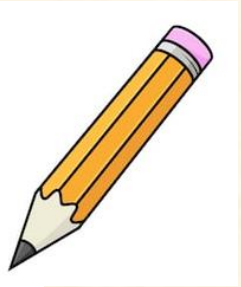
From documentation (Convolution2DFlipout)

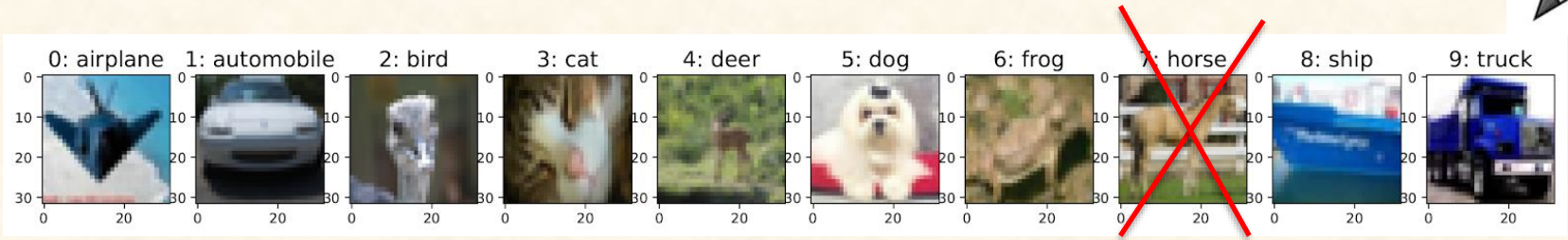
When doing minibatch stochastic optimization, make sure to scale this loss such that it is applied just once per epoch (e.g. if kl is the sum of losses for each element of the batch, you should pass kl / num_examples_per_epoch to your optimizer)

```
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





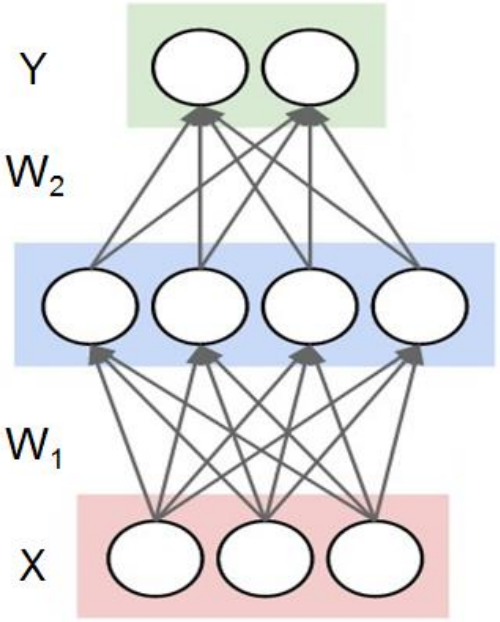
Train a CNN with only 9 of the 10 classes and investigate if the uncertainties are different when predicting images from known or unknown classes.

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

Comparing non-Bayesian with Bayesian NN

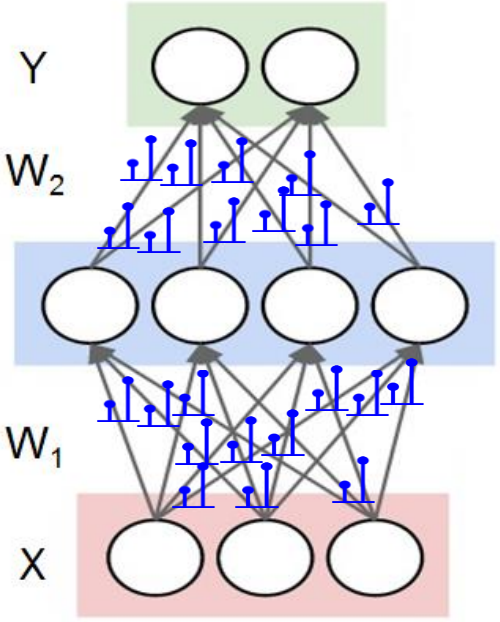
Non-Bayesian and Bayesian NNs

Non-Bayesian NN



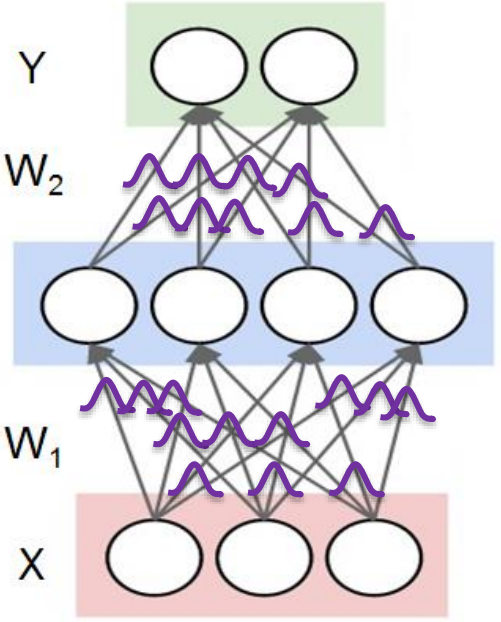
Weights are fixed

MC dropout Bayesian NN



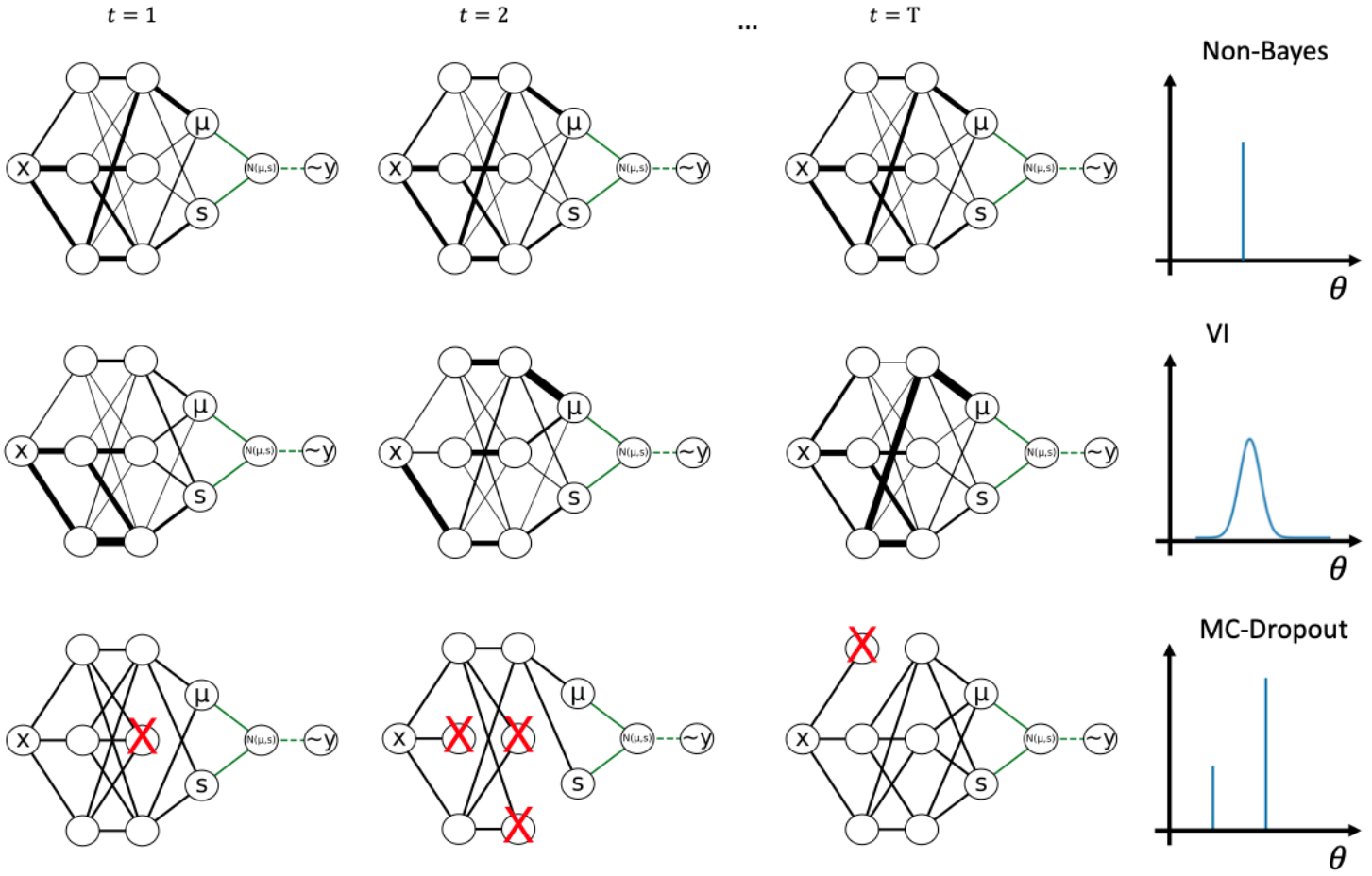
Weights have Bernoulli-kind distribution

VI Bayesian NN



Weights have Gaussian distribution

Comparing different Network types



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

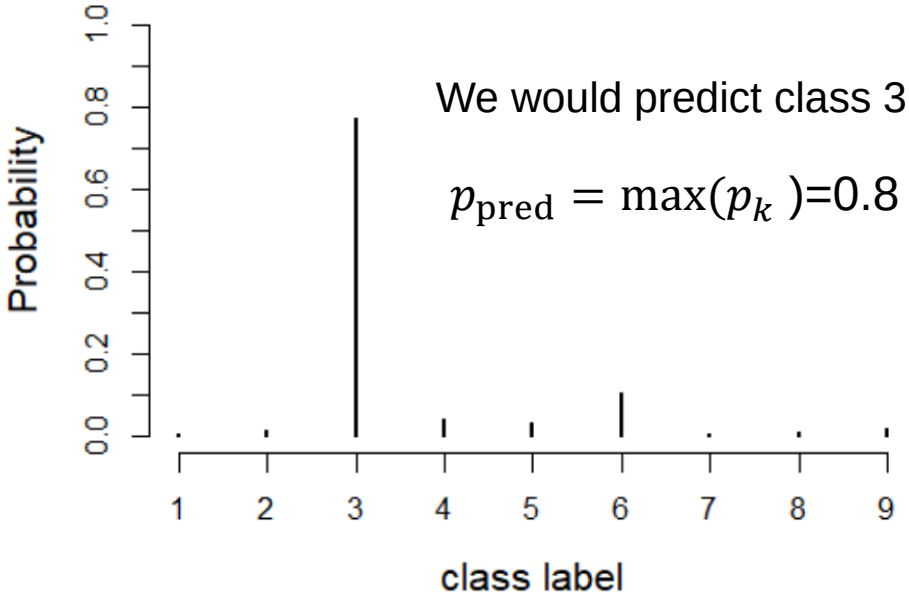
A Non-Baysian NN learns one set of weights: the same input same output
A Bayesian NN learns distribution of weights: same input different outputs

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

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

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: $NLL = -\log(p_{\mathrm{pred}})$

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 ($k \in \{1,2,\ldots,9\}$) we determine the mean probability: $p_k^* = \frac{1}{T}\sum_{i=1}^{T} p_{ki}$

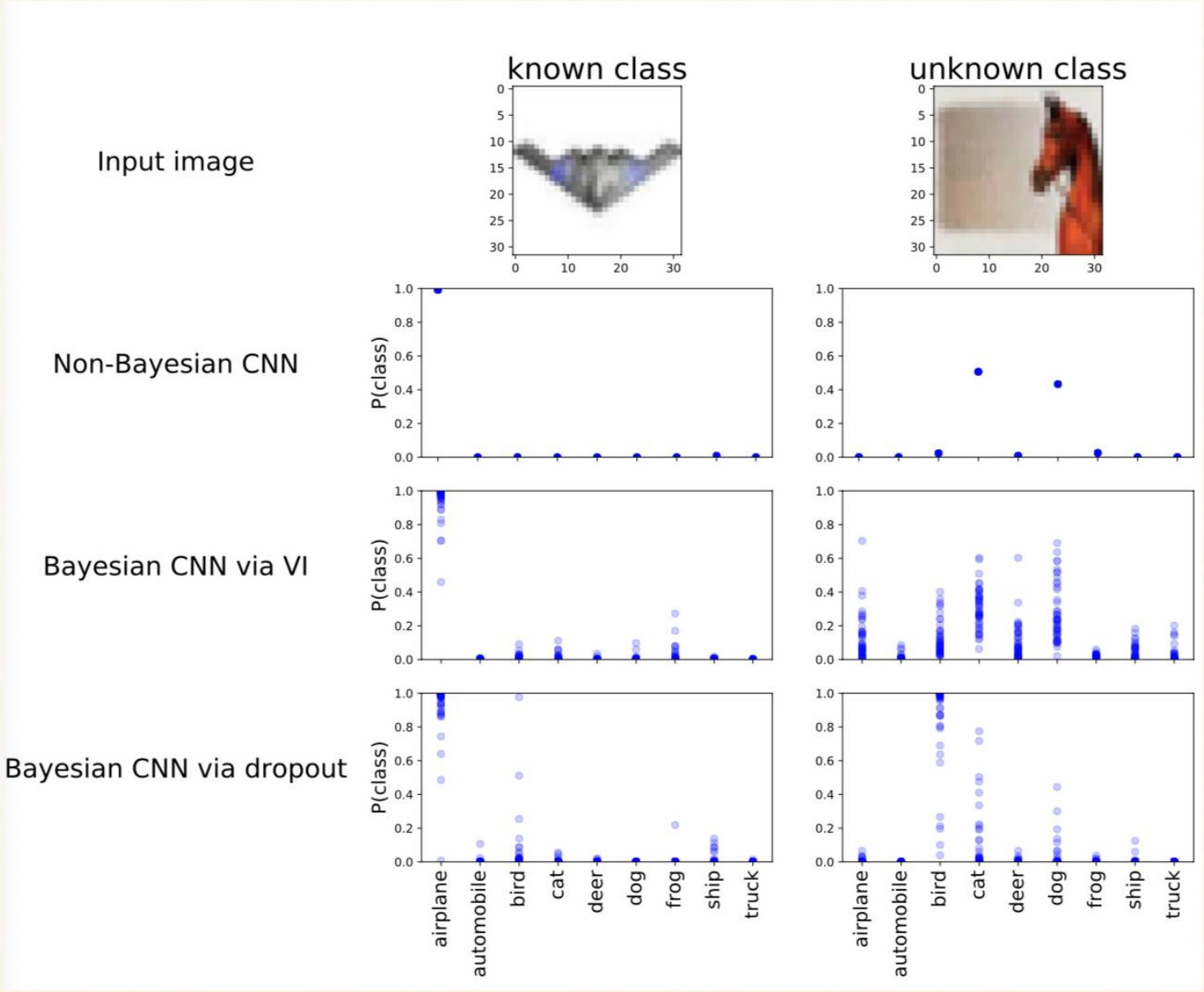
We predict the class with the highest mean probability: $p_{pred}^* = \max(p_k^*)$

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

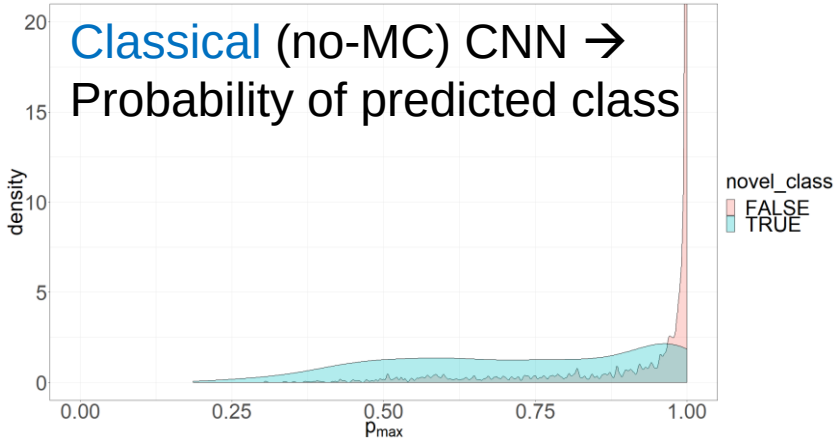
Entropy: $H^* = -\sum_{k=1}^{9} p_k^* \cdot \log(p_k^*)$

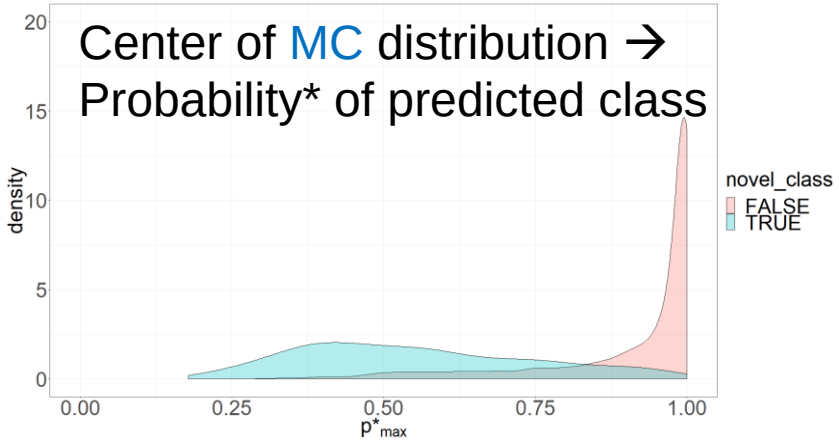
Total variance: $V_{tot}^* = \sum_{k=1}^{9} \mathrm{var}(p_k) = \sum_{k=1}^{9}\sum_{i=1}^{T}(p_{kt} - p_k^*)^2$

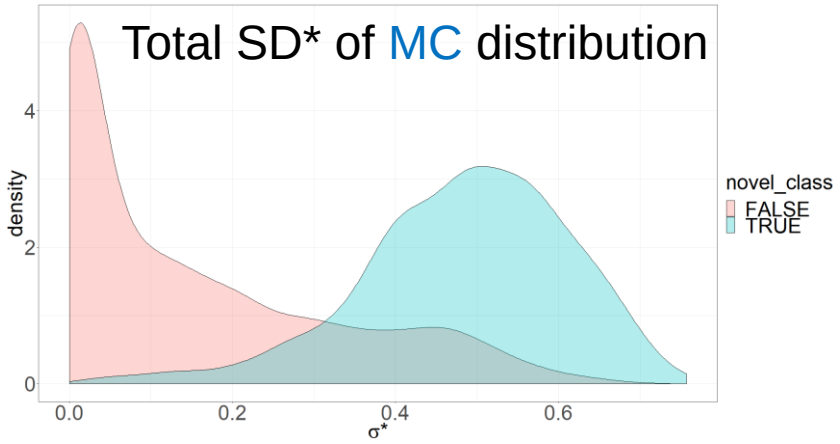
Looking at the predictive distribution!

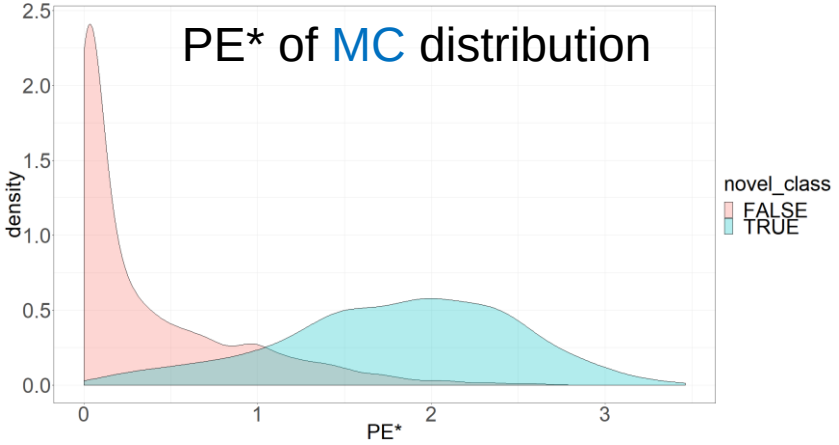


Do known/novel classes yield different values for probability and uncertainty measures?







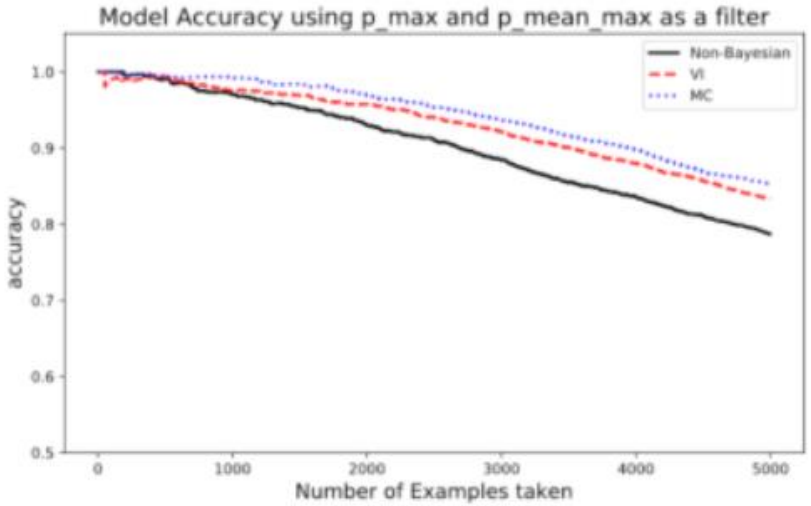


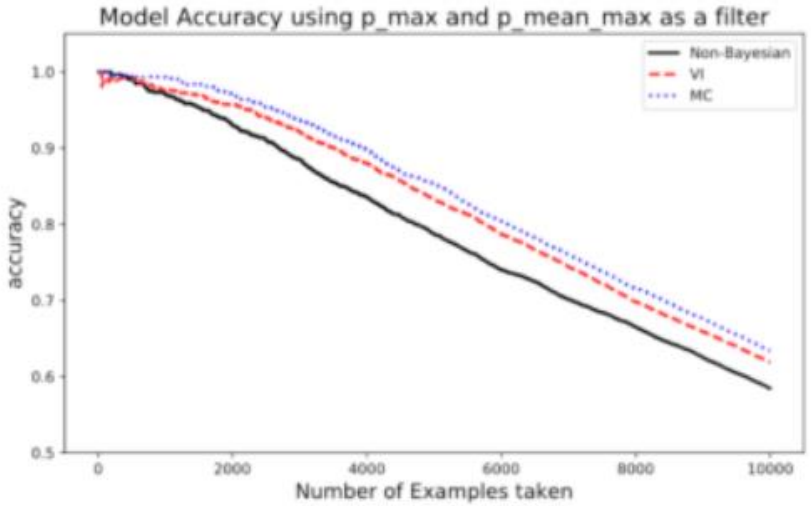
Filtering experiment based on uncertainty

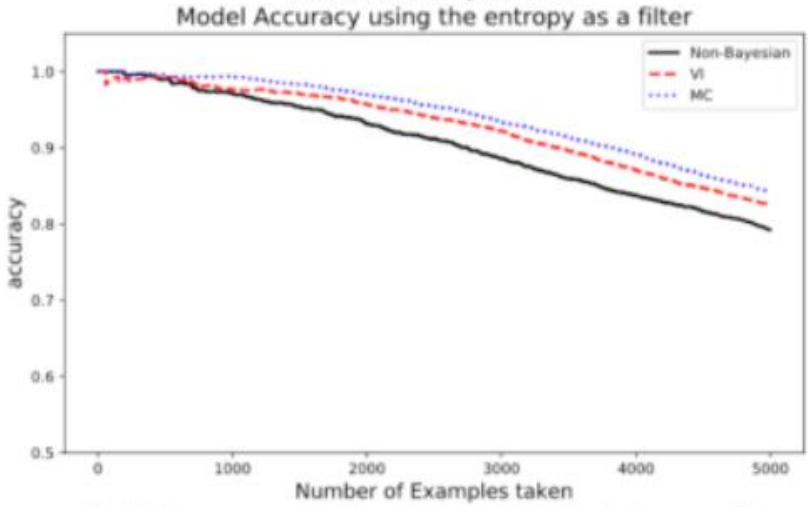
Goal: Get higher accuracy by filter only predictions which are quite certainly correct

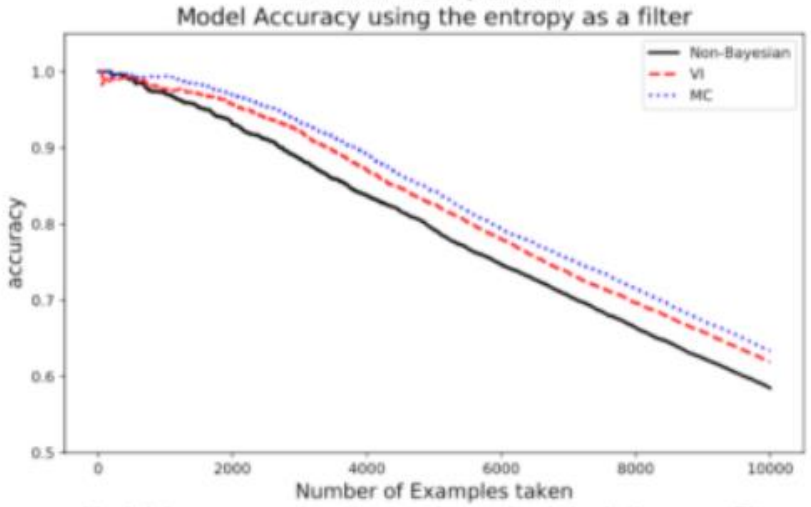
- Each prediction has an attached uncertainty measure
- Sort predictions according to the uncertainty measures
- A set of predictions with very low uncertainties should achieve a high accuracy
- By successively adding predictions with increasing uncertainties should yield predictions sets with decreasing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

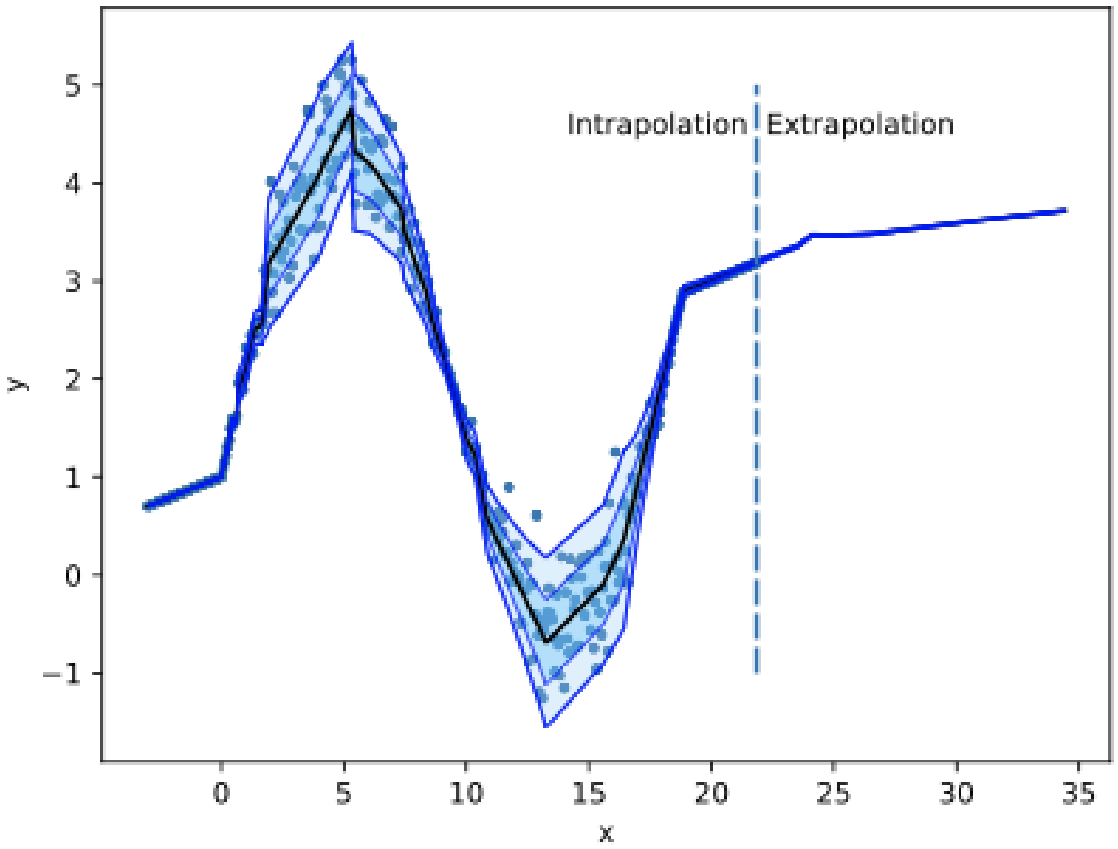
Standard deviation: σ_x

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)

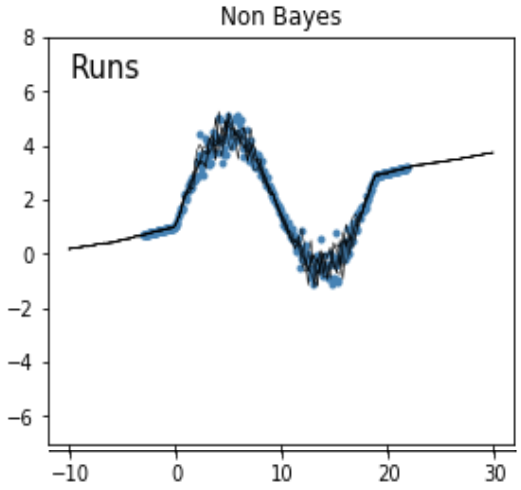
Uncertainty measures including **aleatoric and epistemic** contributions:

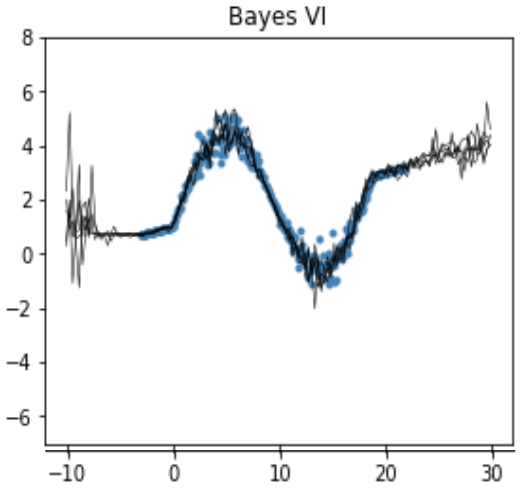
To estimate the 95% PI at position x we sample y-values from each of the T CPDs and determine from the samples the 0.025 and 0.975 quantiles, yielding :

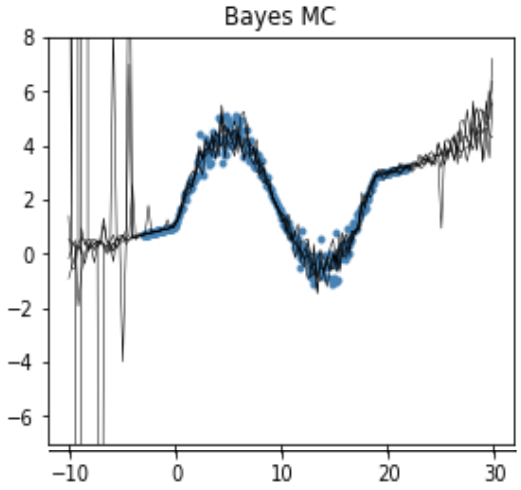
95% PI: $[q_{0.025}; q_{0.975}]$

Can we see enhanced uncertainty in extrapolation

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



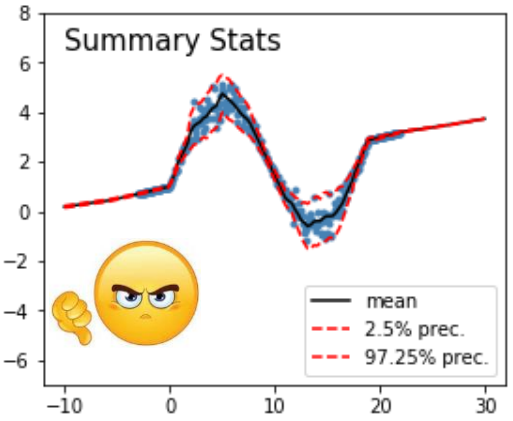


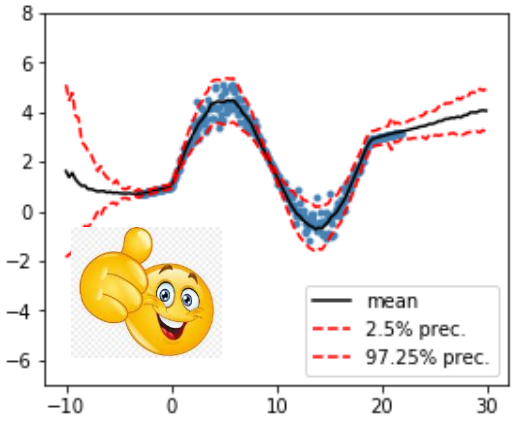


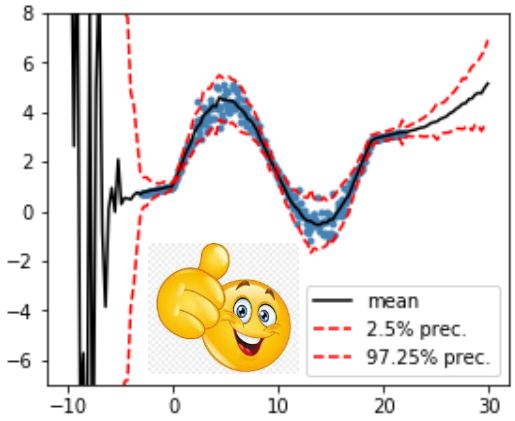
https://youtu.be/FO5avm3XT4g

https://youtu.be/mQrUcUoT2k4

https://youtu.be/0-oyDeR9HrE







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
_________________________________________________________________
```