Machine Intelligence:: Deep Learning
Week 7

Beate Sick, Jonas Brändli, Oliver Dürr

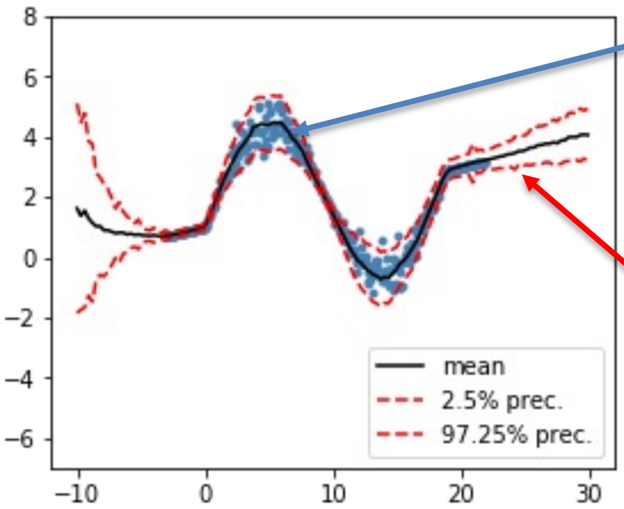
Ensembling approaches for improving the performance and uncertainty estimates of NN models by taking into account the epistemic uncertainty.

Outline:

- Uncertainty in DL models
  - Algorithmic uncertainty
  - Epistemic uncertainty
  - Aleatoric uncertainty

- Approaches to take algorithmic and/or epistemic uncertainty into account:
  - Deep Ensembling
  - MC Dropout
  - Bayes via Variational Inference

Aleatroic vs. Epistemic Uncertainty

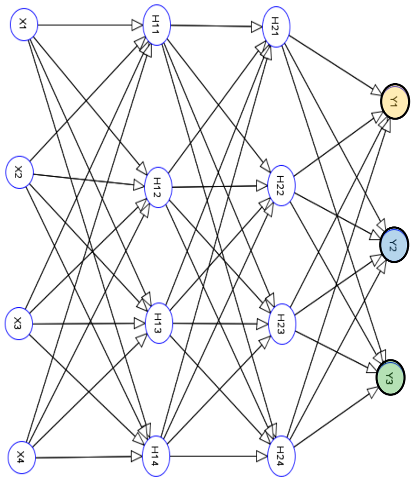


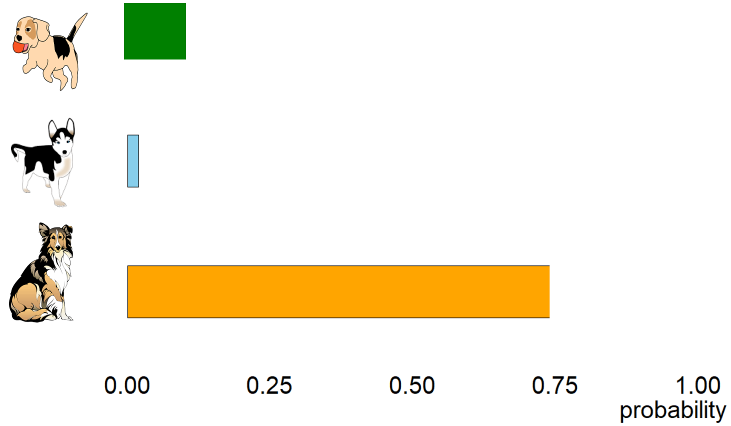
- *Aleatoric* uncertainty is due to the uncertainty, that is inherent in the data
- Algorithmic uncertainty is due to uncertainty inherent in the algorithmus (DL using **S**GD, **random** weight initialization)
- Epistemic uncertainty when leaving the 'known ground' is called *epistemic* uncertainty → only few or no data is available to learn

Infer if a die is fair with
no (or few) train data
→ no (or few) "knowledge"
→ epistemic uncertainty
(from Ancient Greek ἐπιστήμη *(epistḗmē)* 'knowledge')

Predict the number of spots
when rolling a fair die
→ Aleatoric uncertainty
(from latin "Alea Acta est")

Probabilistic CNNs as we know them so far



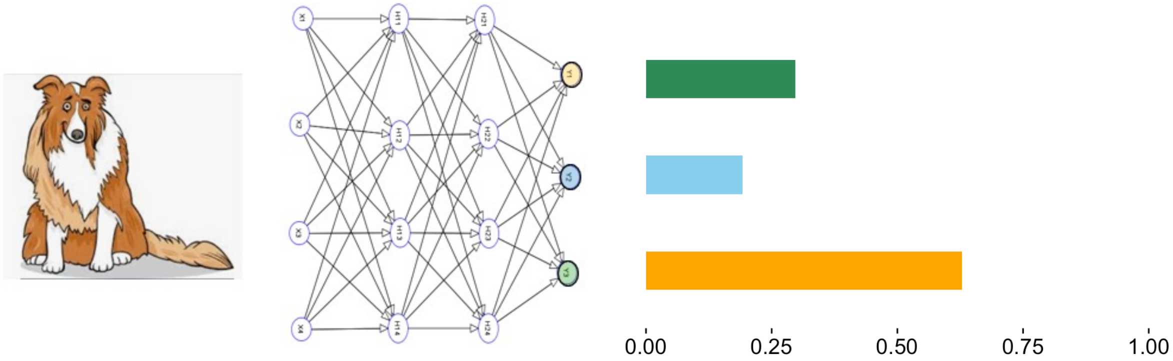


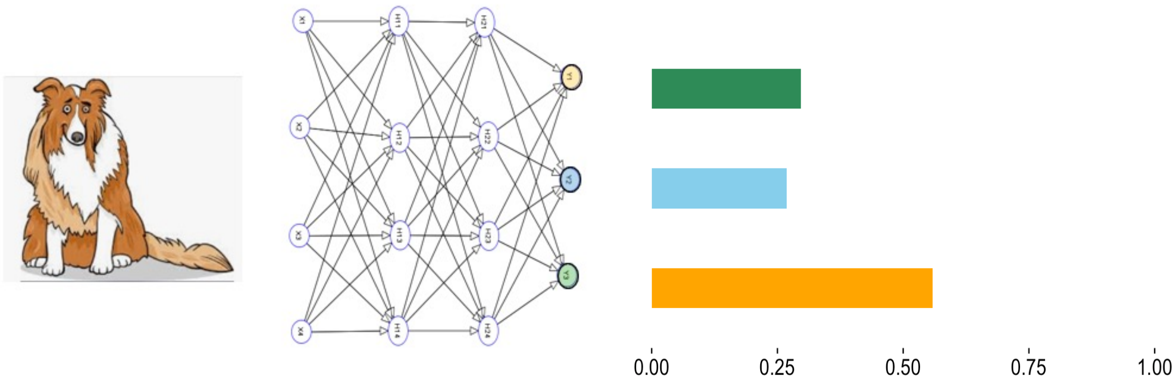
The aleatoric uncertainty is captured by the CPD (conditional probability distribution)

CNNs yield high accuracy and calibrated (=unbiased) probabilities, but…

How good do we know probabilistic CNNs?

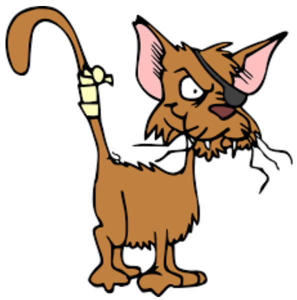
What happens if we train the same CNN twice with the same data?

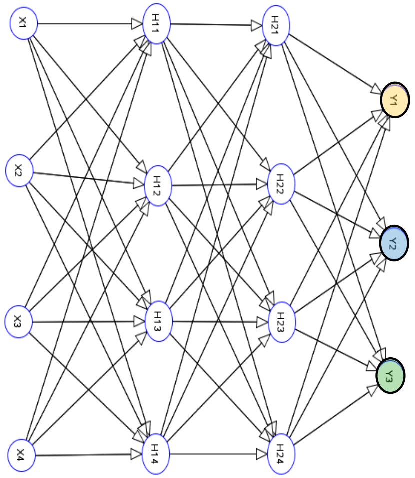




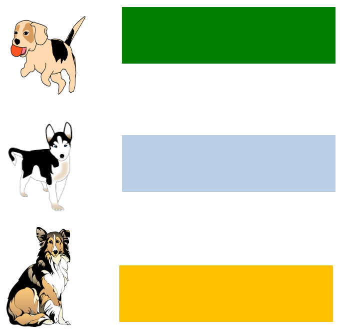
How good do we know probabilistic CNNs?

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



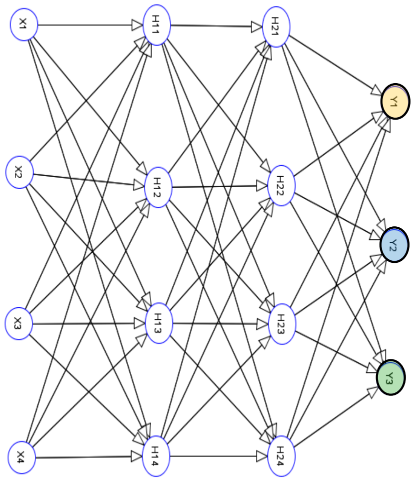


You might expect:



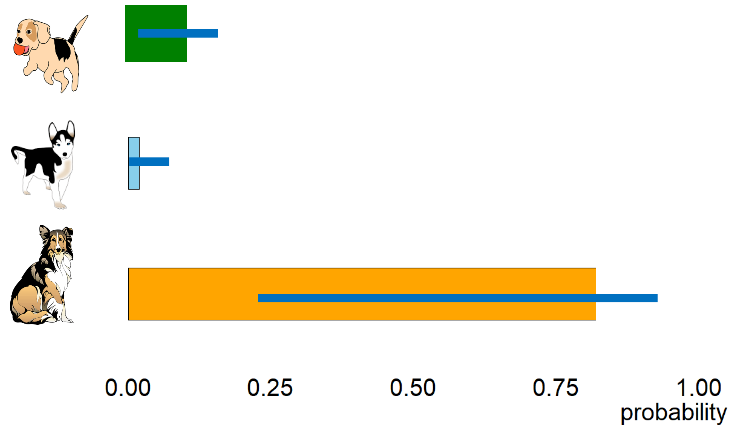
A classical NN cannot ring the alarm in case of out-of-distribution (OOD) examples

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



But you probably get:

Plain wrong !



We need some error bars!
We need algorithmic and epistemic uncertainty!

Importance to detect OOD (out of distribution)

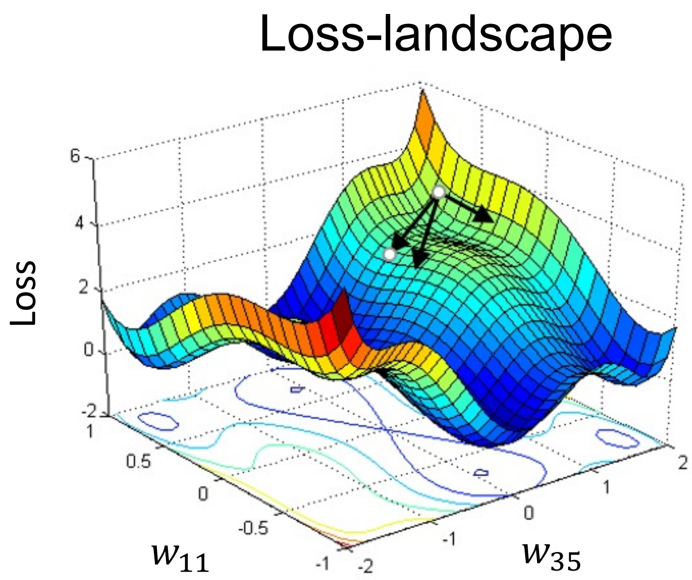
- Current DL Systems bad in out of distribution OOD situations
- Application need at least to detect OOD situations

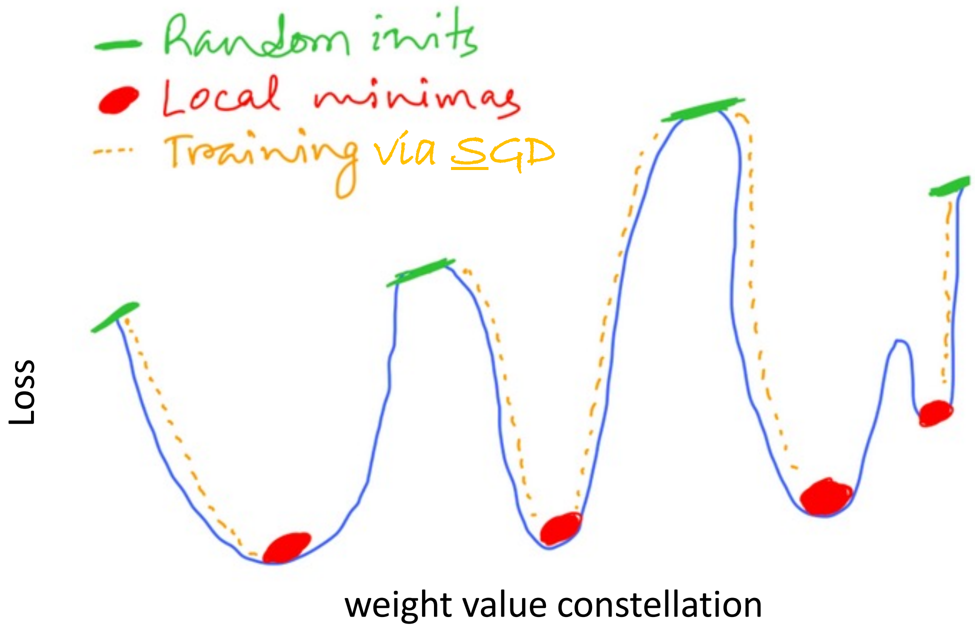
Foto: https://www.crn.de/software-services/halloween-training-fuer-google-autos.108409.html

Algorithmic uncertainty

→ deep ensembling

The loss-landscape in DL is usually not convex



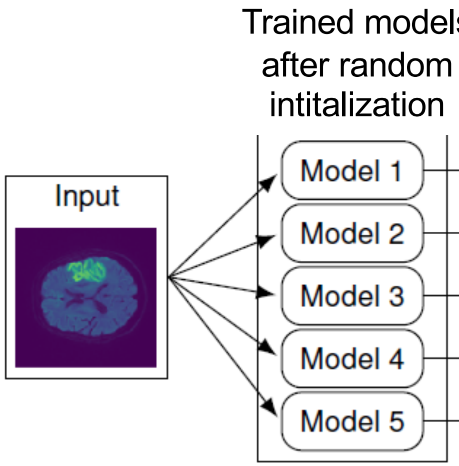


The loss-landscape of DL models has many local minima with similar depth.

Training is started with a **random** weight value initialization → training the NN with **S**GD and 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

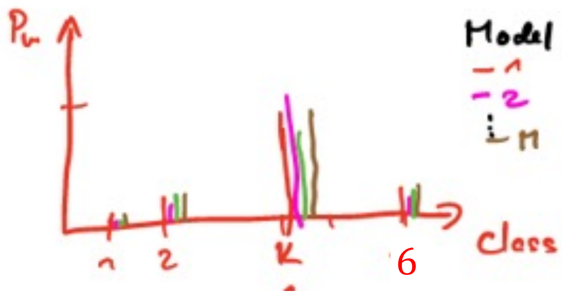


Each model predictis probabilities for the 6 classes

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

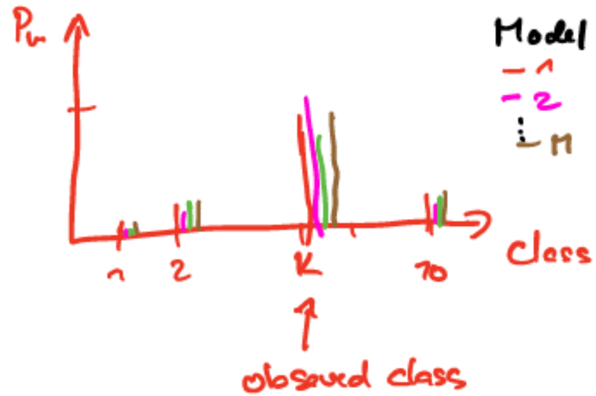
$sd_1^E \quad sd_2^E \quad sd_3^E \quad sd_4^E \quad sd_5^E \quad sd_6^E$

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:



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

associated NLL contribution ℓ:

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

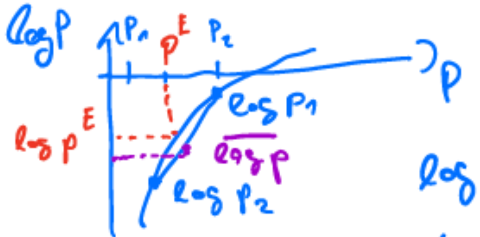
to show $\bar{\ell} \geq \ell^E$ with

$$\bar{\ell} = \frac{1}{M}\sum_{m=1}^{M} \ell_m = \frac{1}{M}\sum_m -\log p_{km}$$

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

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

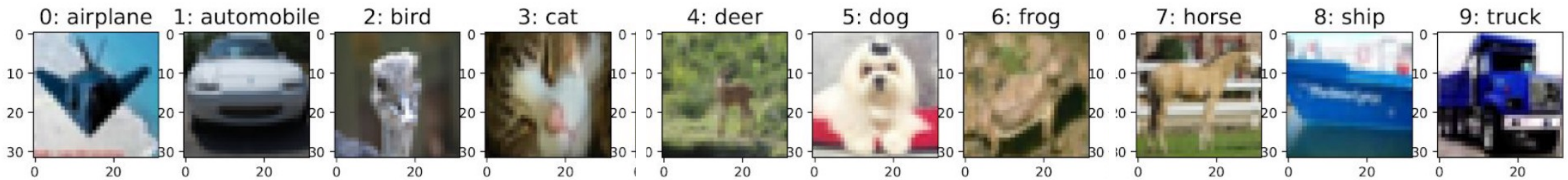
use Jensen inequality

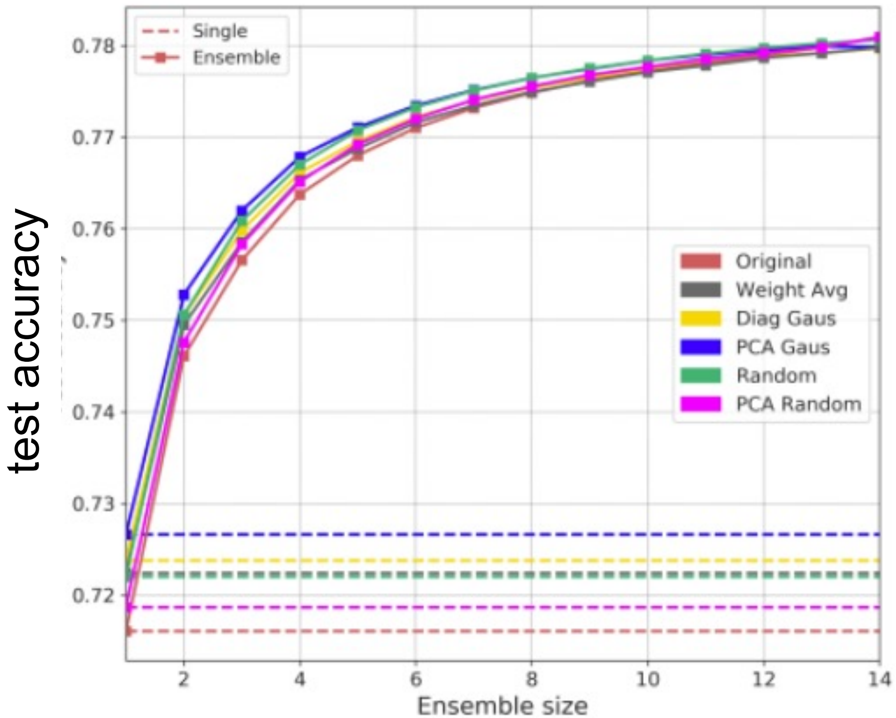


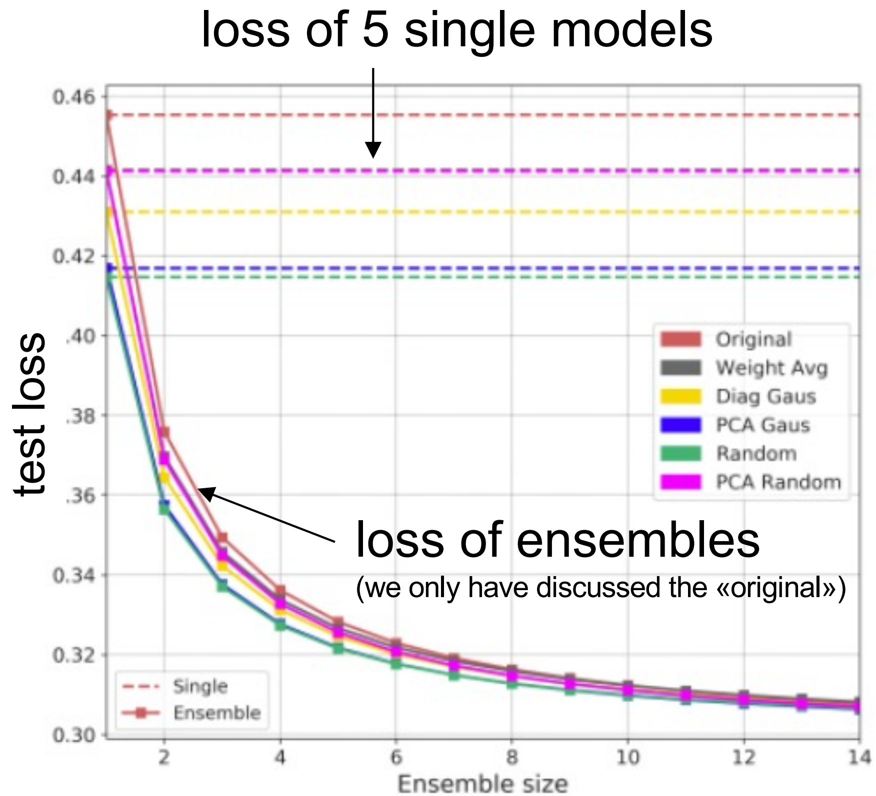
$$\log \bar{p} \geq \overline{\log p}$$

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

Deep ensembling improves prediction power







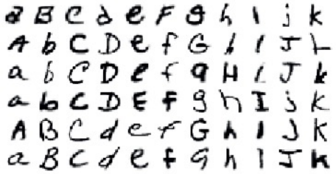
Ensembes with as few as 3 or 5 members are typically enough to achieve a perfromance gain.

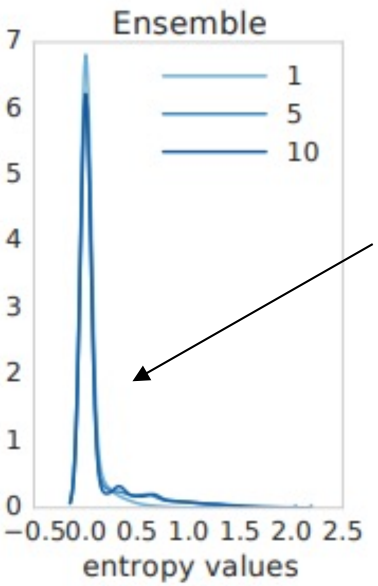
Image credit: https://wandb.ai/authors/loss-landscape/reports/Understanding-the-Effectivity-of-Ensembles-in-Deep-Learning--VmlldzoxODAxNjA

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

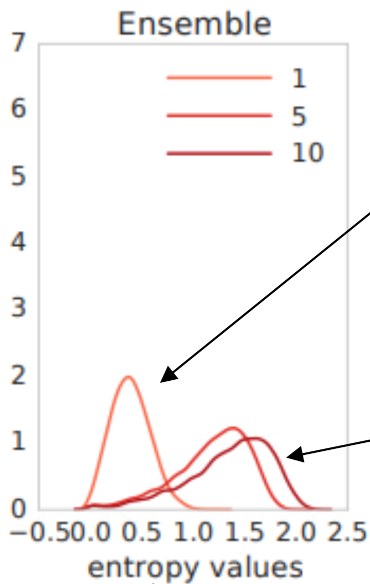




uncertainty measure

Ensemble uncertainty for a known class stays small for all sizes of the ensemble

Uncertainties for prediction of **NOT-MNIST** images



uncertainty measure

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

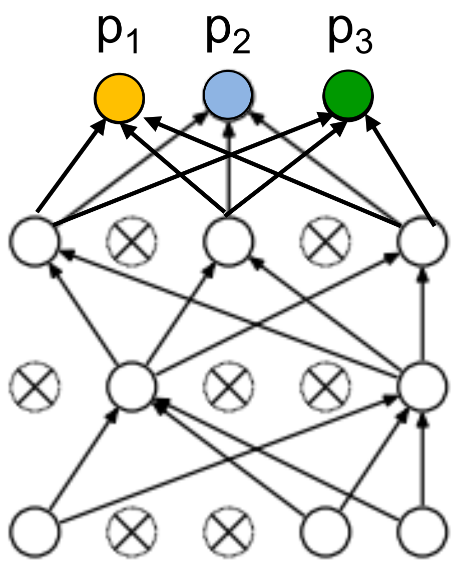
Ensemble uncertainty measures for novel classes improve with #members per ensembles

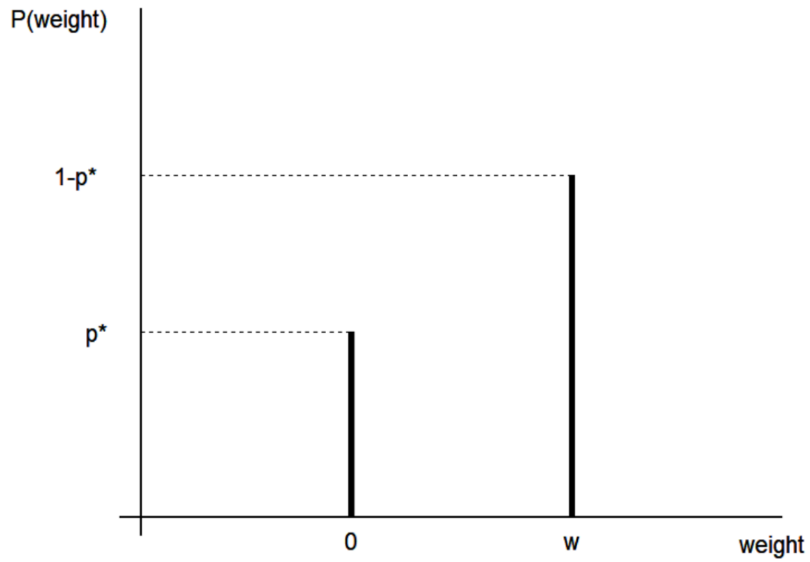
Image credits Lakshminarayanan et al.: https://arxiv.org/pdf/1612.01474.pdf

Epistemic uncertainty (partly)

→ Dropout during test time

Recall: Classical Dropout only during training

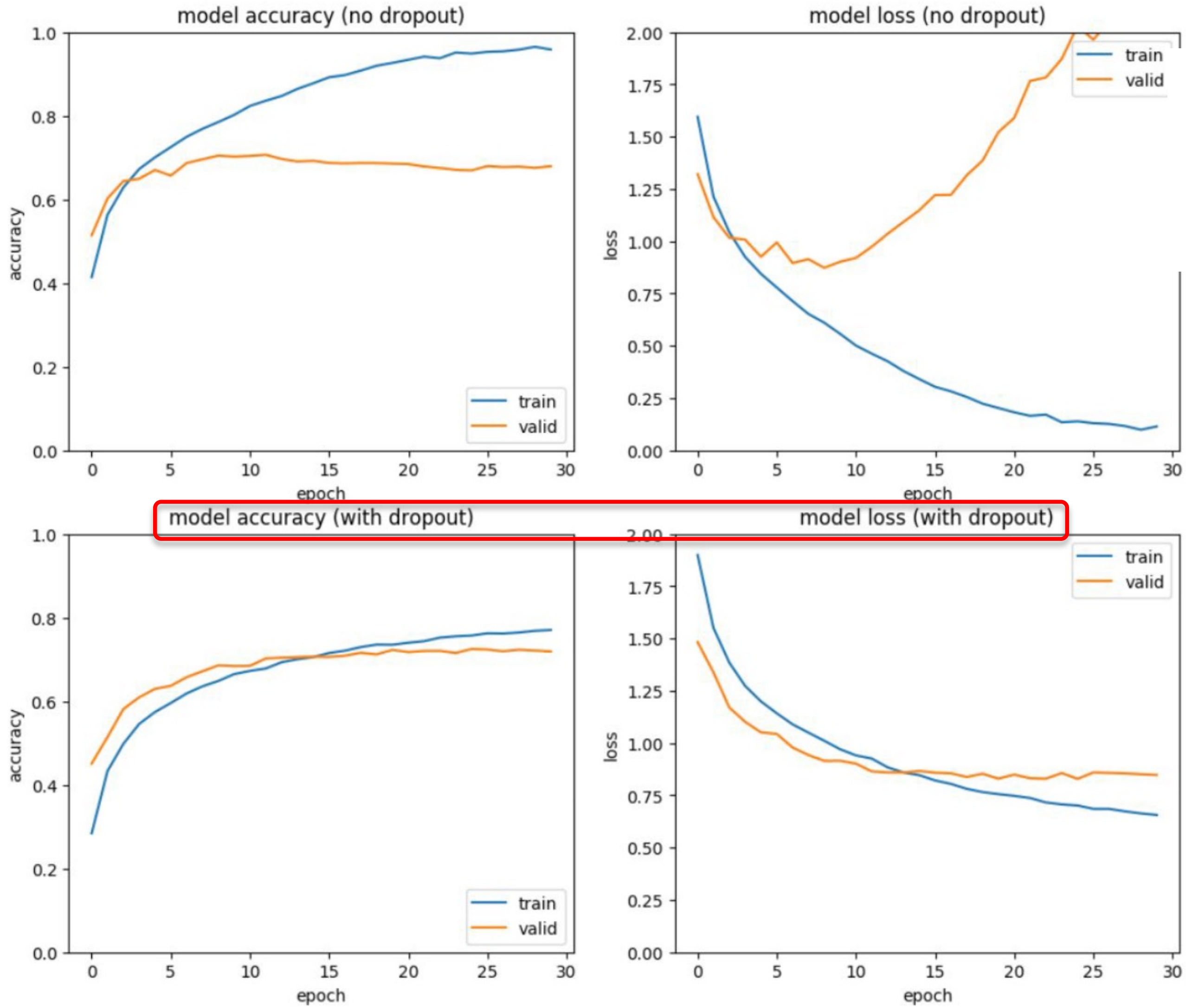


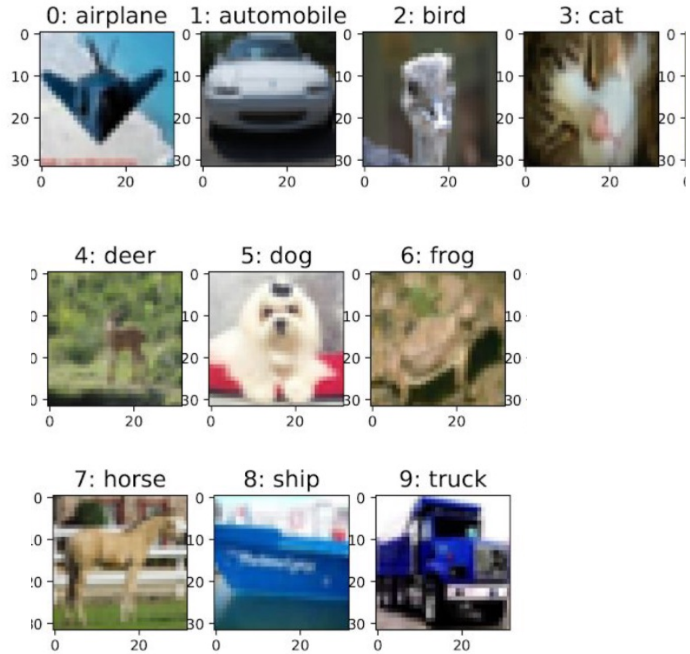


Using dropout during training implies:

- 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



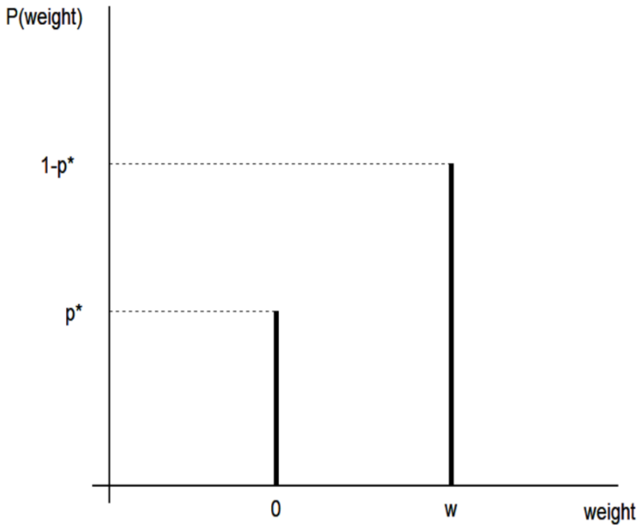


From Dropout during training to MC Dropout during test time

Bayesian NN via MC Dropout

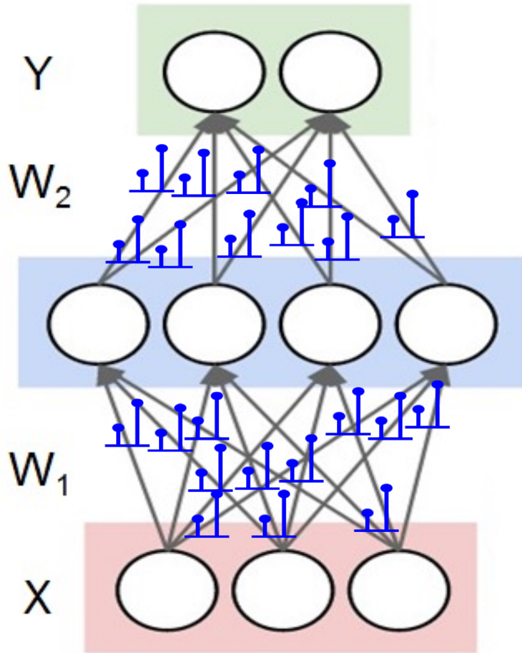
Yarin Gal et al. (2015):
Via Dropout training we learned a whole weight distribution for each connection.
We can sample from this Bernoulli-kind weight distribution by performing dropout during test time and use the dropout-trained NN as Bayesian NN.
Gal showed that doing dropout approximates VI with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

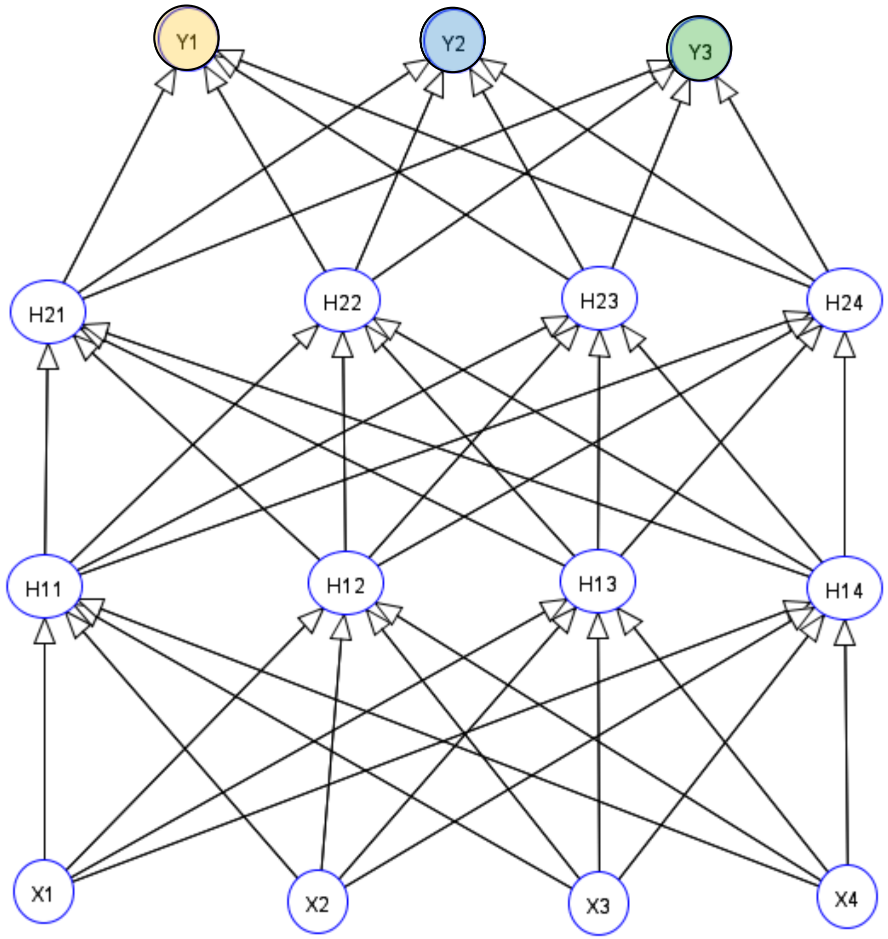
Which parameter has this q_θ?

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:

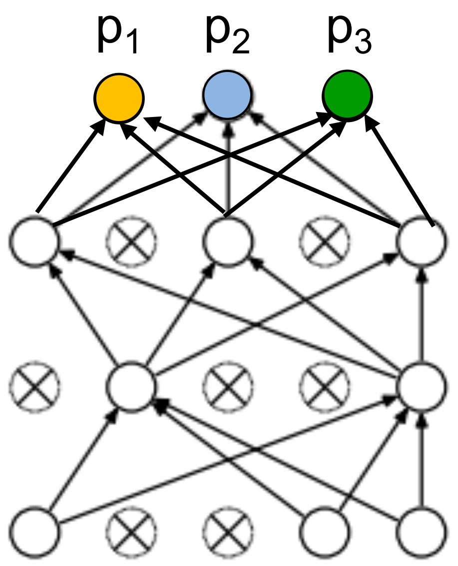
p_1=0.07 **p_2=0.91** p_3=0.02

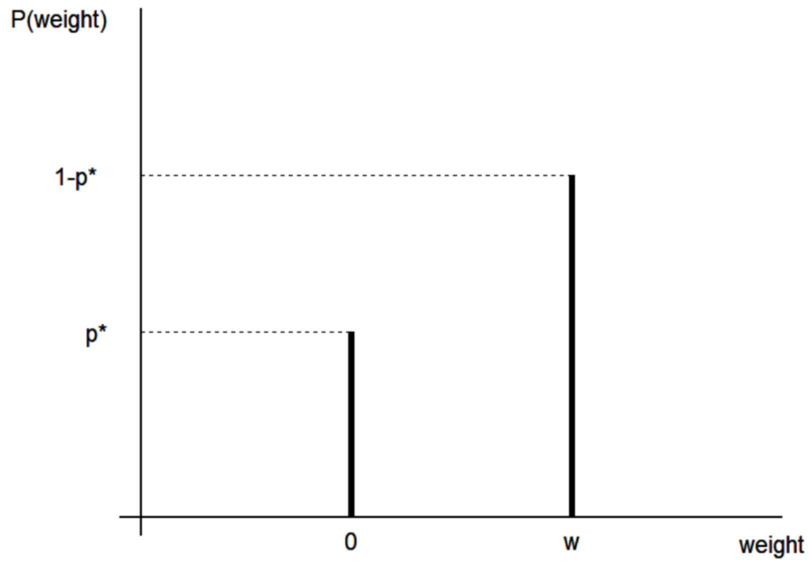


Probability of predicted class: **p_{max}**

Input: image pixel values

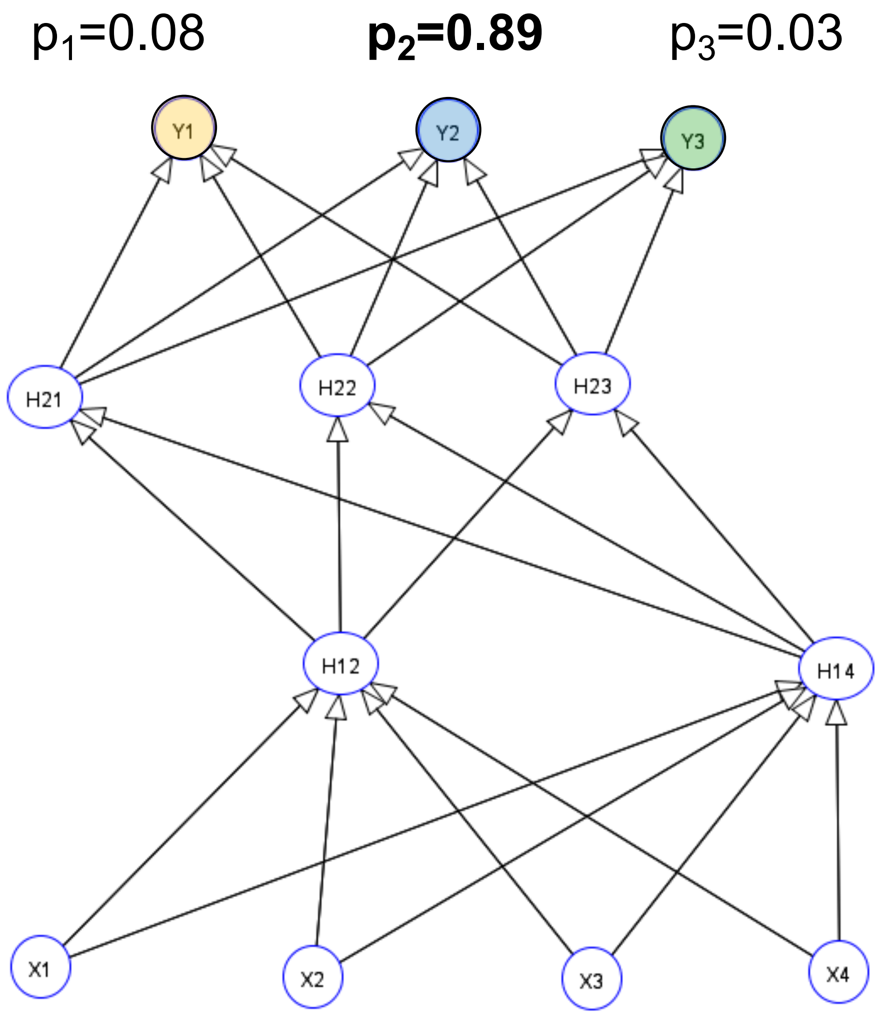
MC Dropout: we also perform dropout during test time

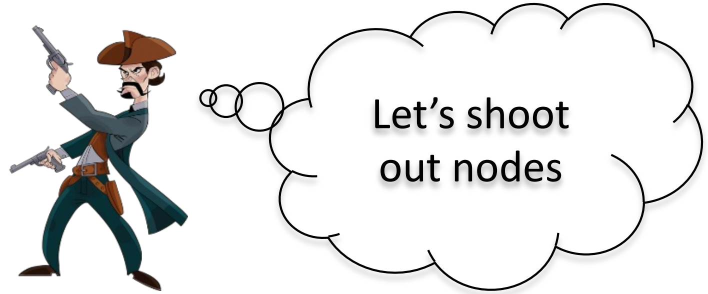




In each prediction instance we dropout a random subset of nodes, which corresponds to setting all weights starting from these nodes to zero.

MC Dropout during test time: Run 1

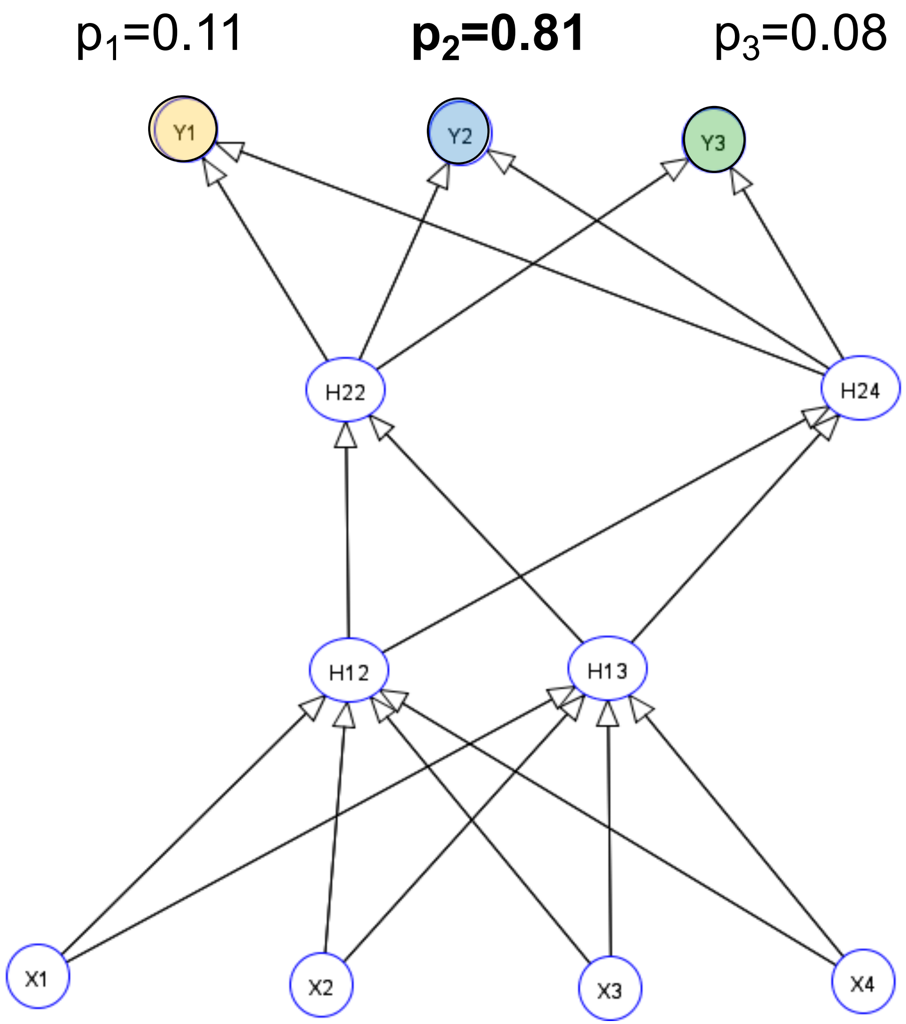


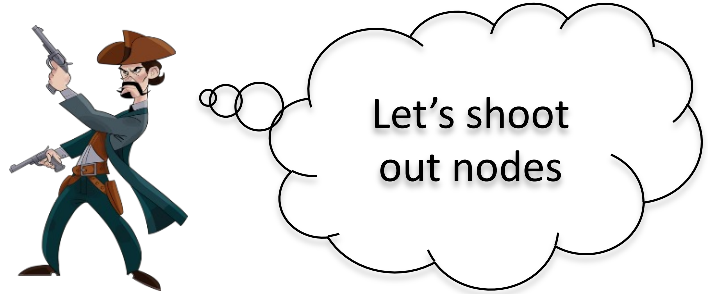


Stochastic dropout of units

Same input image

MC Dropout during test time: Run 2

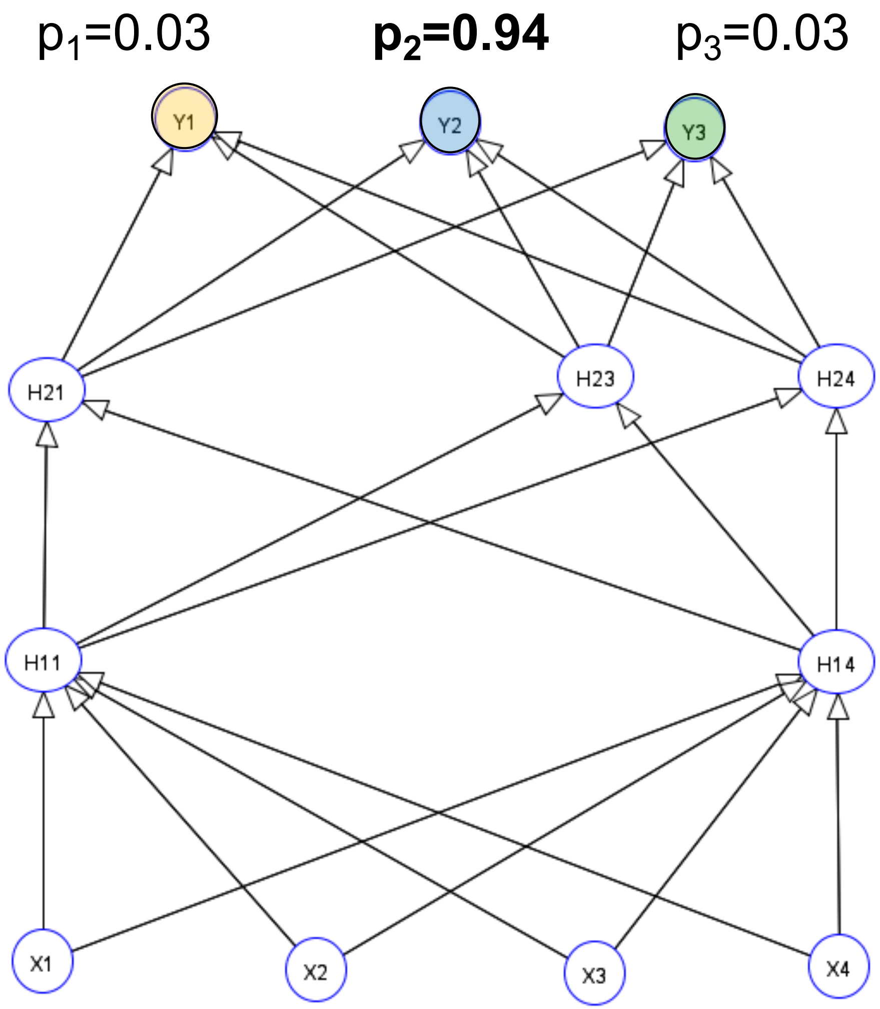


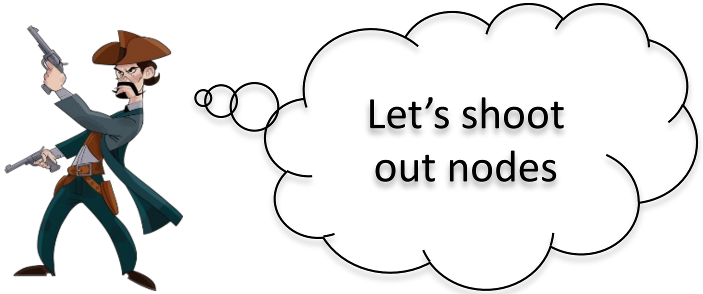


Stochastic dropout of units

Same input image

MC Dropout during test time: Run 3

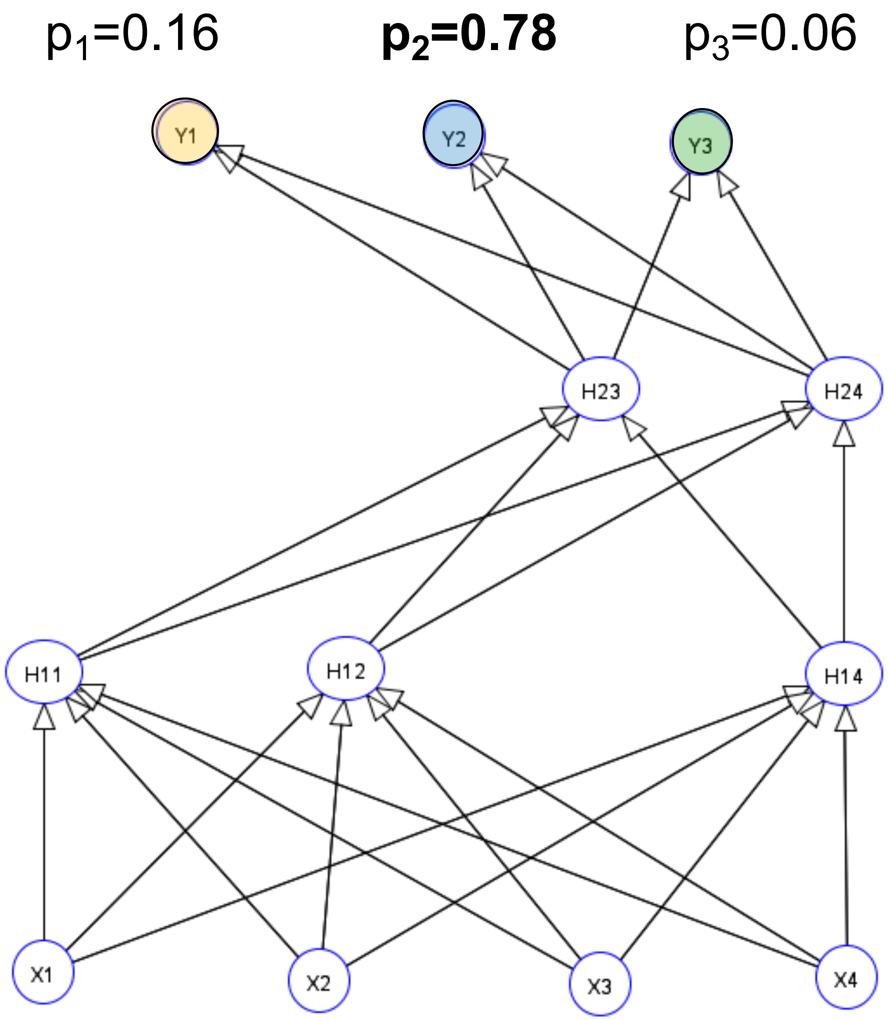


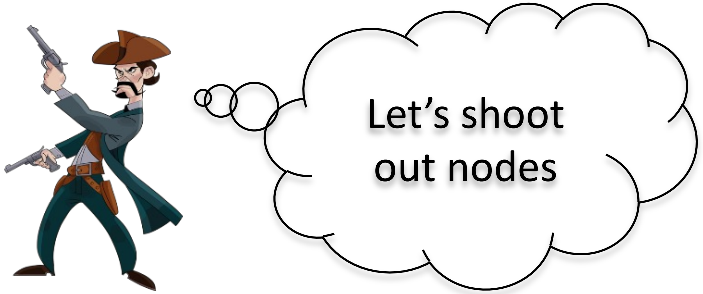


Stochastic dropout of units

Same input image

MC Dropout during test time: Run 4

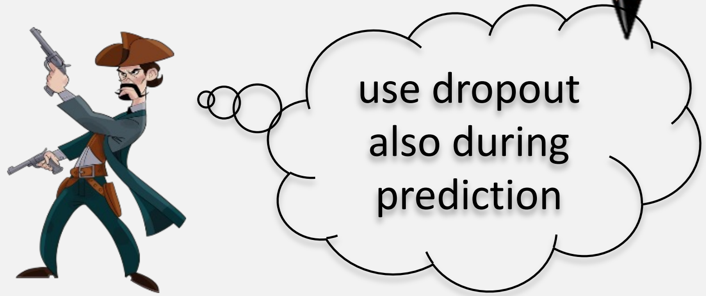




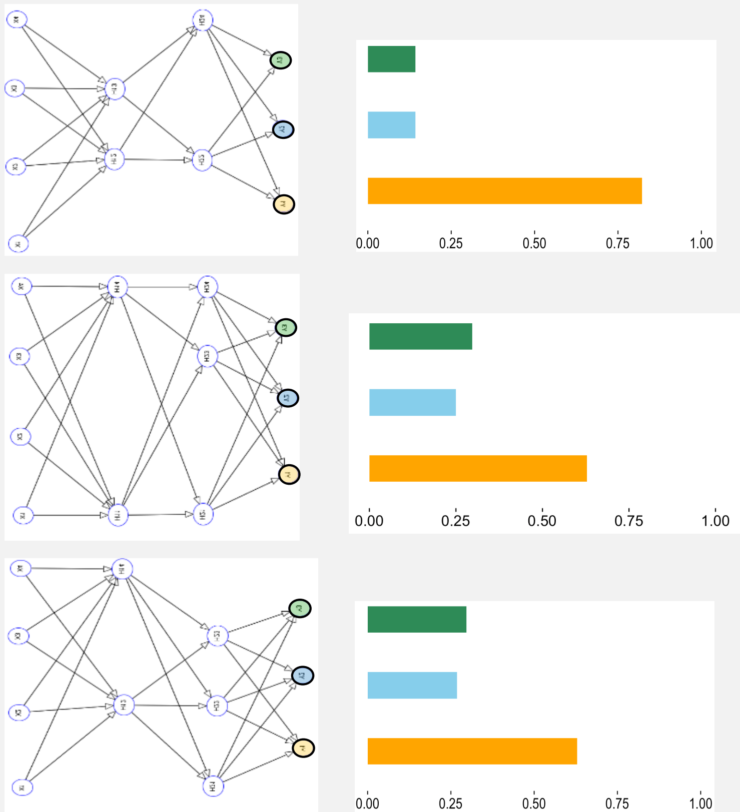
Stochastic dropout of units

Same input image

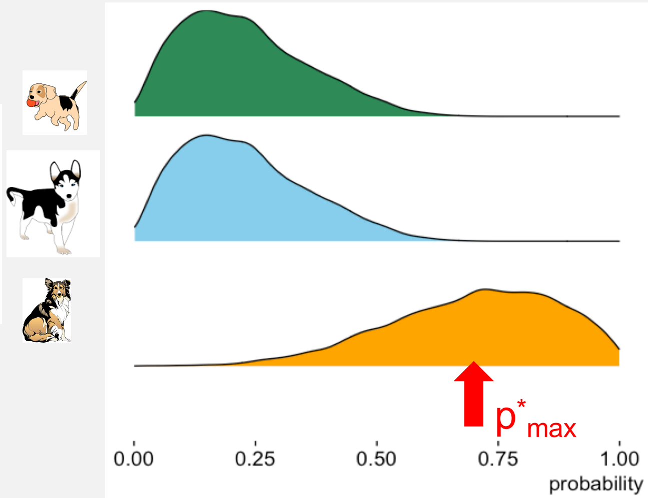
MC Dropout during test time yields a multivariate predictive distribution for the parameters



Many Dropout Runs in forward pass



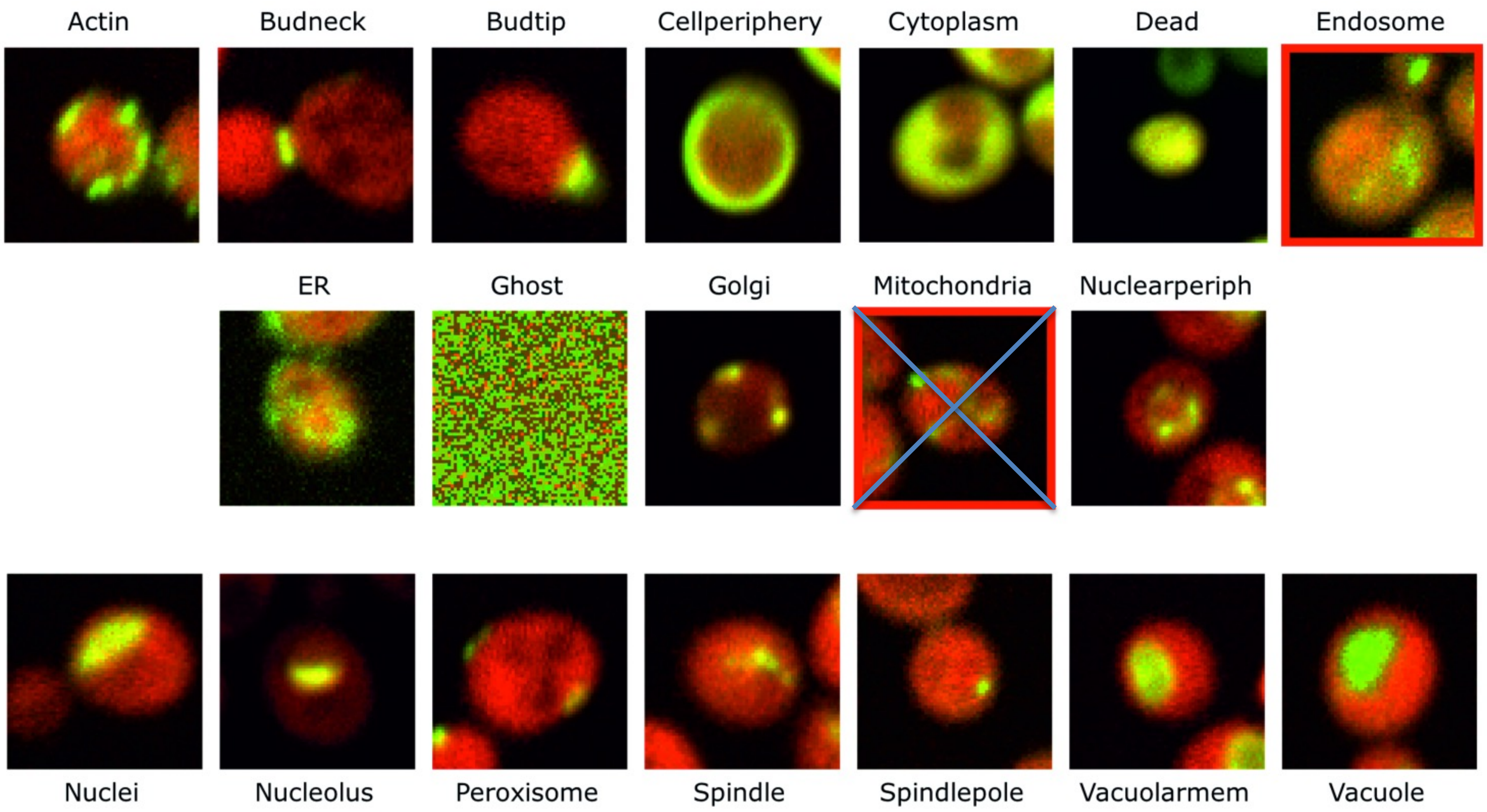
...



CNN predicts class “collie” but with high uncertainty

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

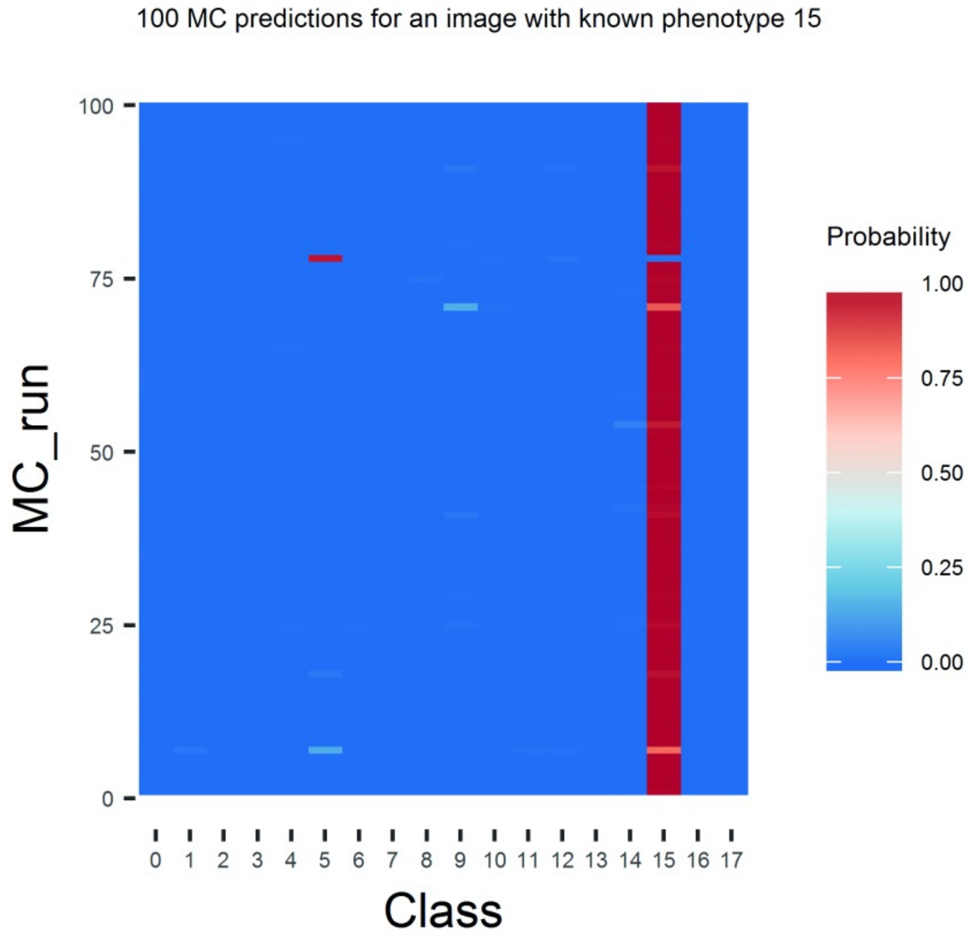
Experiment with unknown phenotype



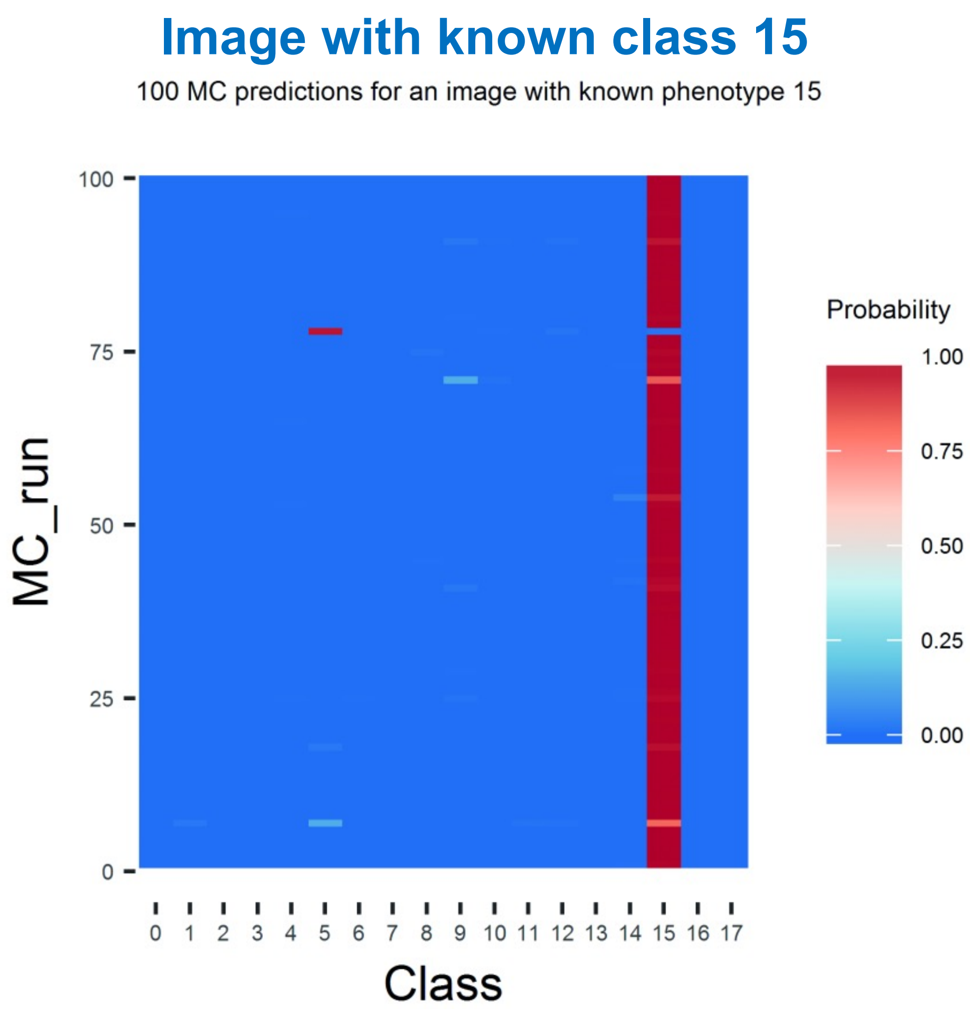
Dürr O, Murina E, Siegismund D, Tolkachev V, Steigele S, Sick B. Know when you don't know, Assay Drug Dev Technol. 2018

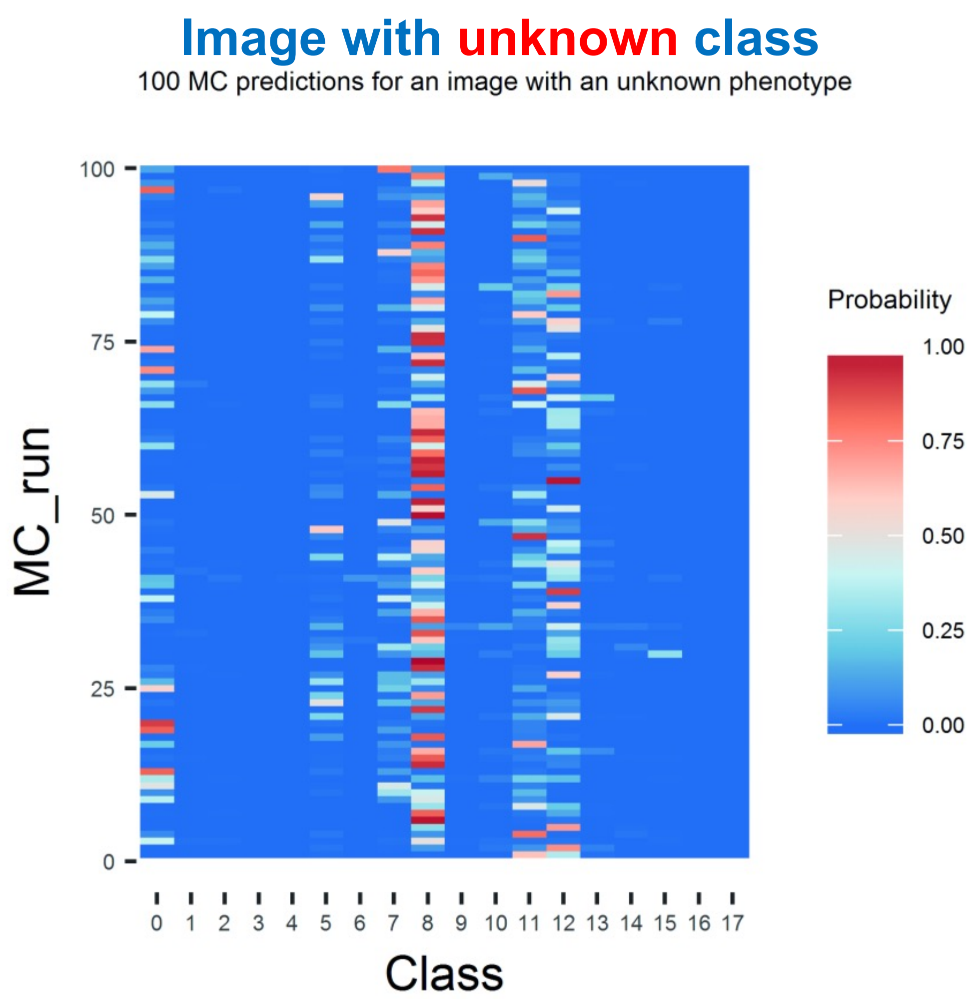
Probability distribution from MC dropout runs

Image with known class 15



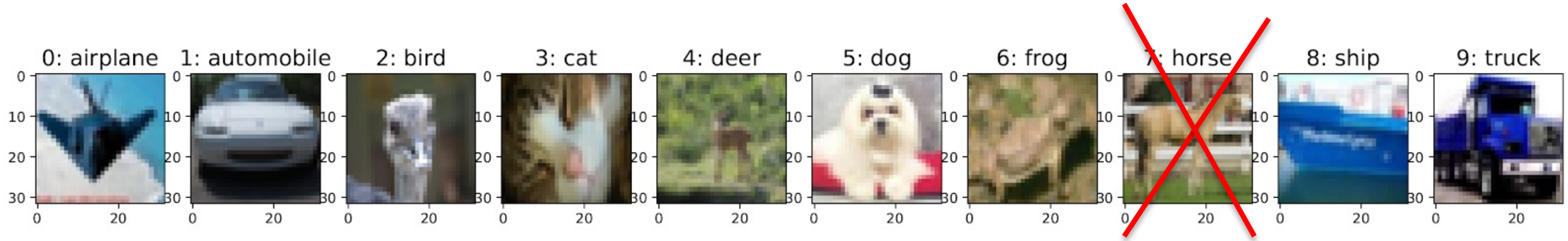
Probability distribution from MC dropout runs





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.

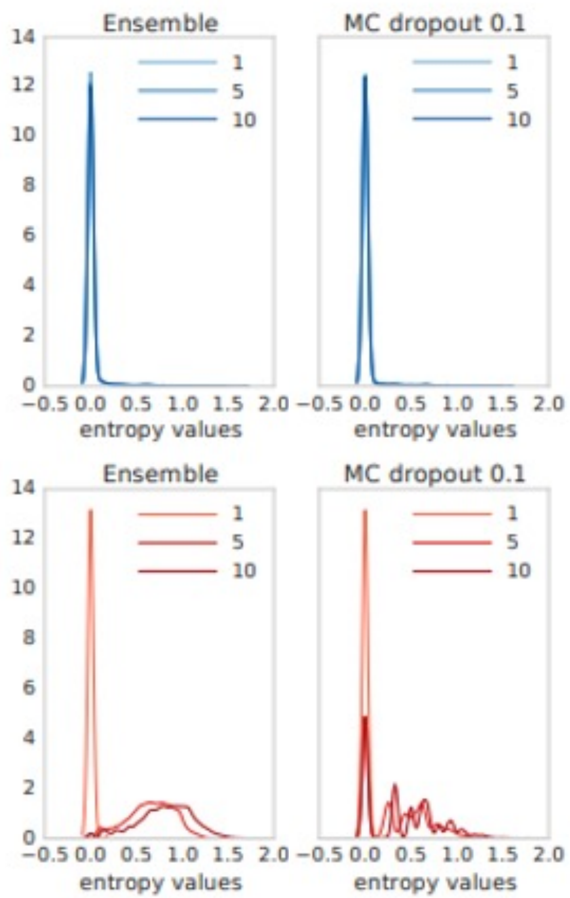


Compare MC dropout and deep ensembles uncertainties

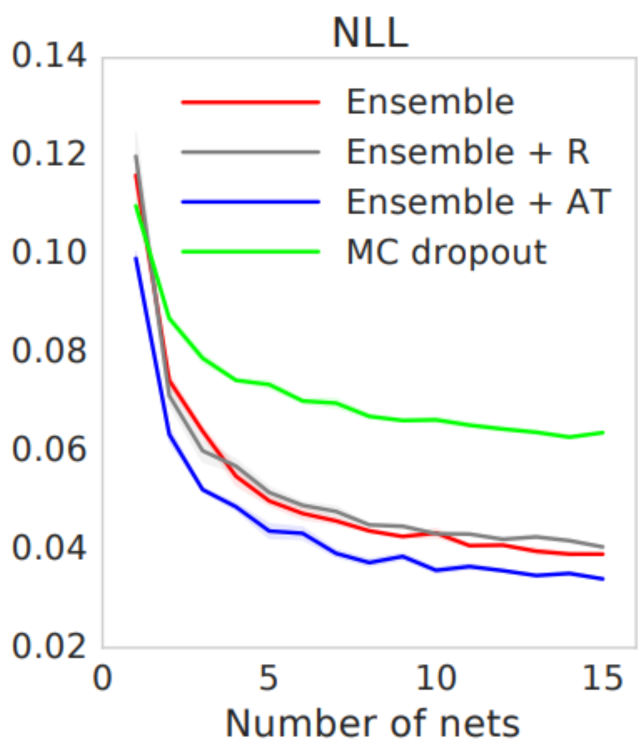
Uncertainty on novel classes

Classes from train

Novel Classes



Prediction performance



Deep ensembles outperform MC Dropout with respect to uncertainty quantification and prediction performance!

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

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

Bayesian Neural Networks

Bayesian NN

Ensembling and MCMC Dropout work

But why?

Bayesian Statistics is the language to formulate epistemic uncertainty

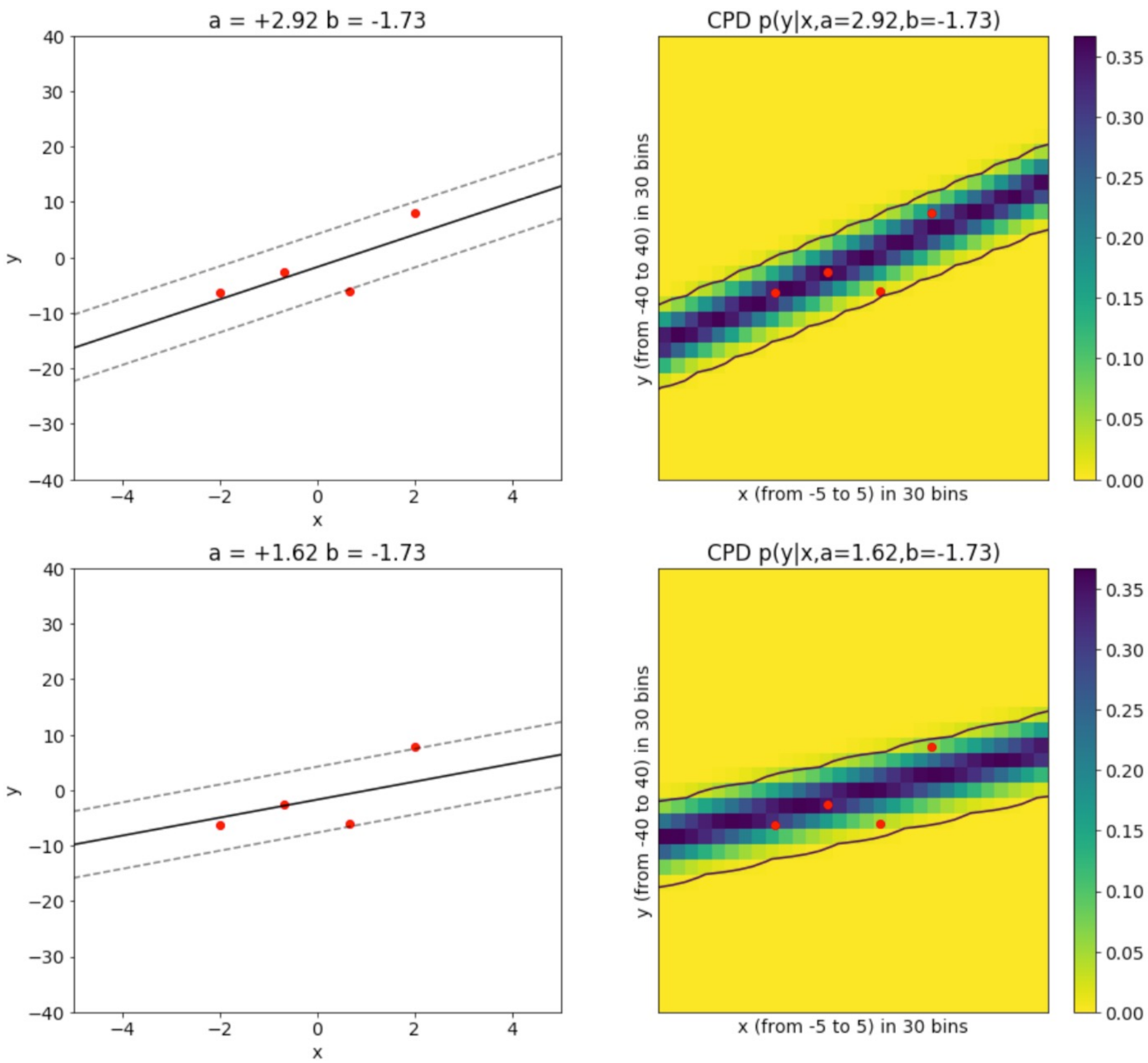
In the following a sneak preview into Bayesian Statistics

For NN we will introduce a direct approximation. Variational Inference

MC Dropout and Ensembling can be also seen a Bayesian Approximations

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.

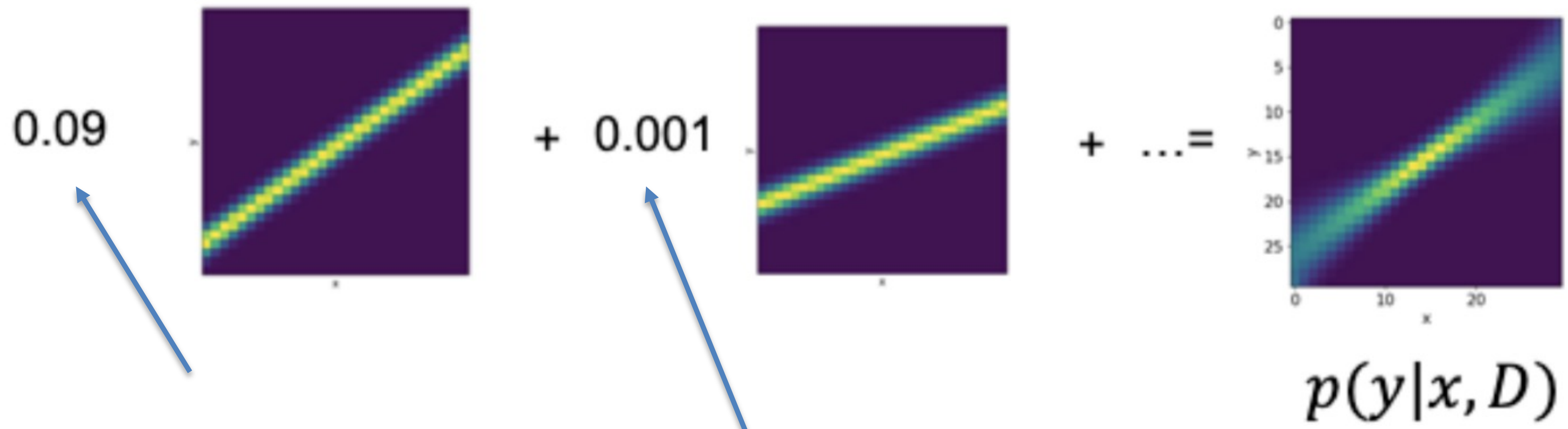


MaxLike Solution

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

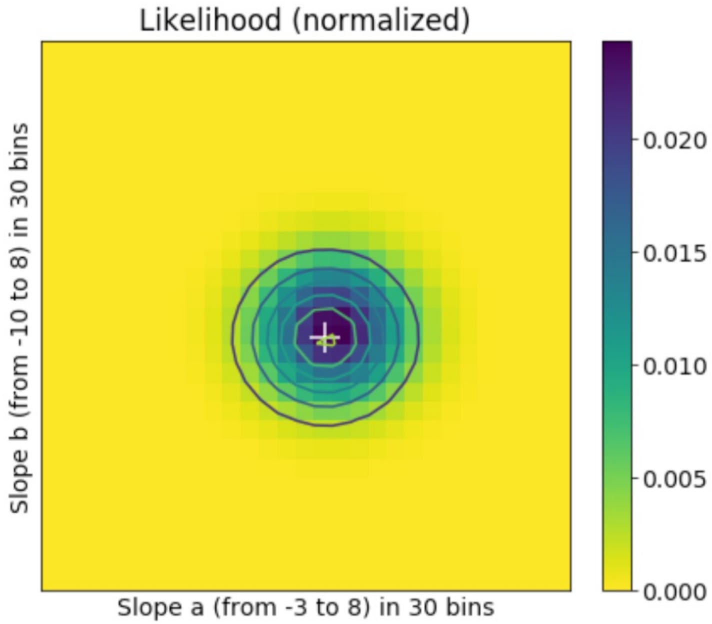
Less likely solutions get
a smaller weight

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



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

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

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

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

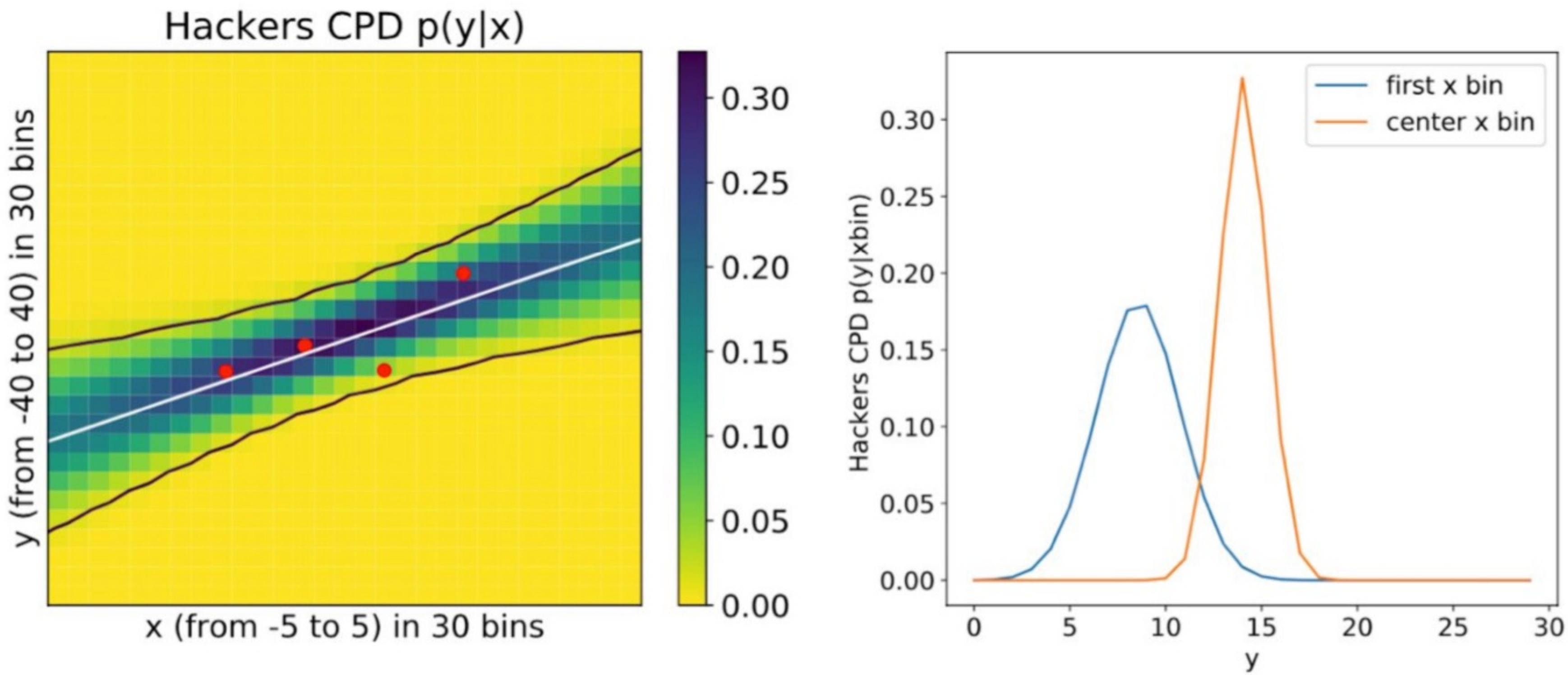


Figure 7.6 The predictive distribution for the Bayesian linear regression model, trained with the four data points shown on the left side by the color-code and on the right as conditional distribution at two different *x* positions. You can clearly see that the uncertainty gets larger when leaving the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

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

$$p(w|D) = \frac{p(D|w)\boldsymbol{p}(\boldsymbol{w})}{p(D)} = \frac{p(D|w)p(w)}{\sum_w p(D|w)p(w)}$$

→ 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.

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

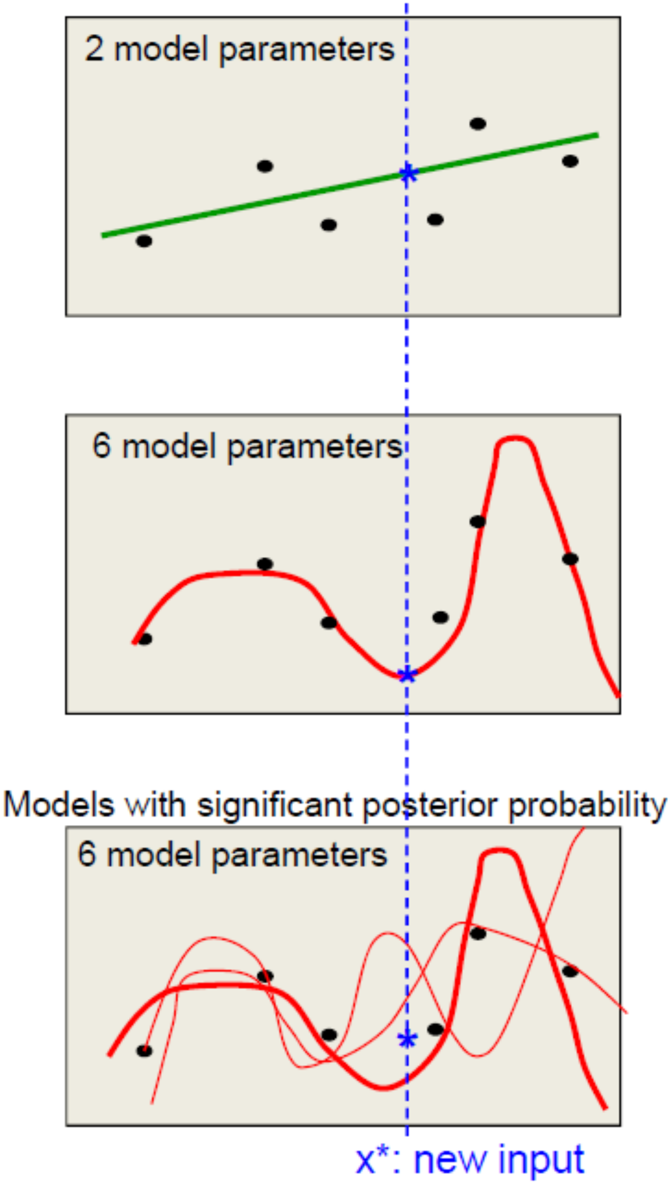


Image credits: Hinton coursera course

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

$p(D|\theta)$ likelihood

$p(\theta)$ prior

$p(D)$ 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

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)

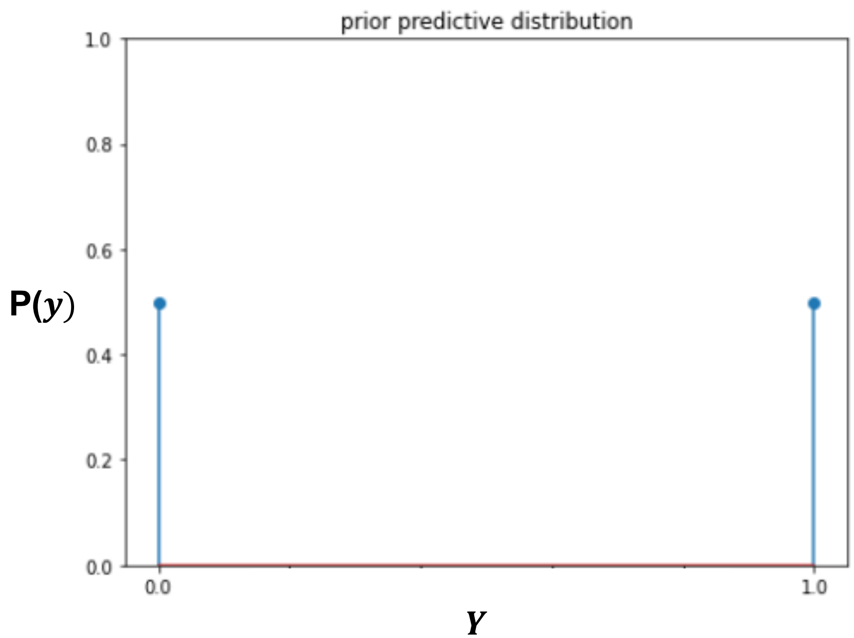
prior belief (before seeing data)

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 $p(\theta|D) \sim p(D|\theta)p(\theta) = \theta^3$ beliefs more in head

Posterior (after)

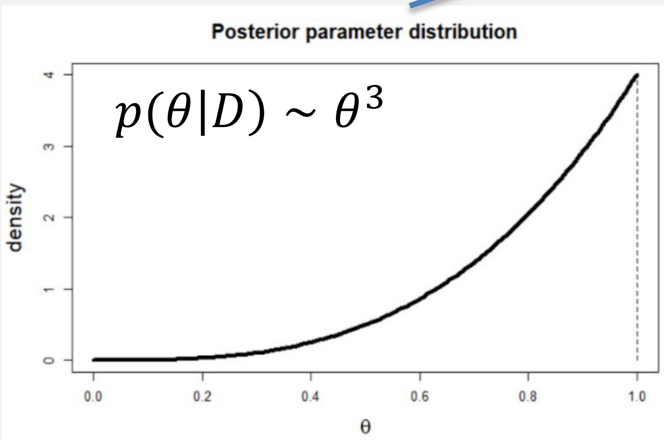
Prior (before)

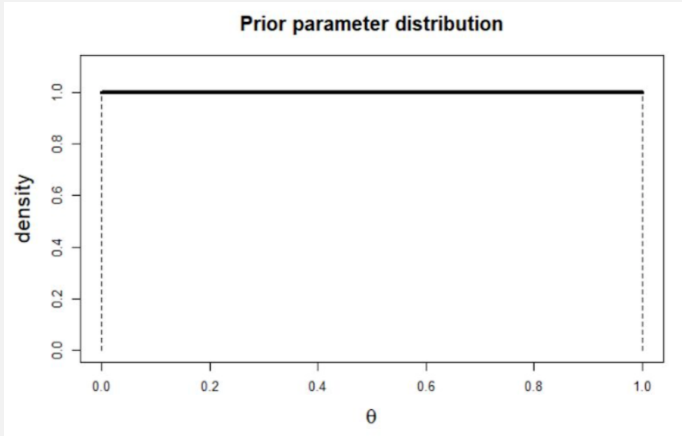
$$p(\theta|D) \sim p(D|\theta)p(\theta)$$

$p(\theta|D)$

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)

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

To compute the posterior predictive probability for head in the coin example, we need:

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

Posterior predictive distribution

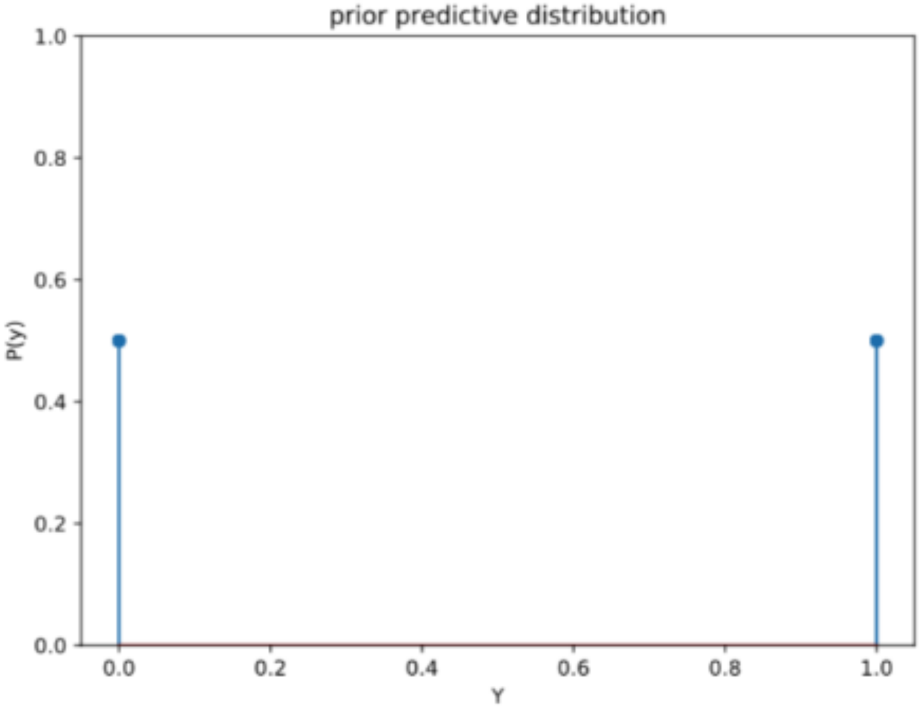
$$\mathrm{p}(y=1|D) = \int_\theta \theta\ \cdot 4\cdot\theta^3 d\theta = 0.8$$

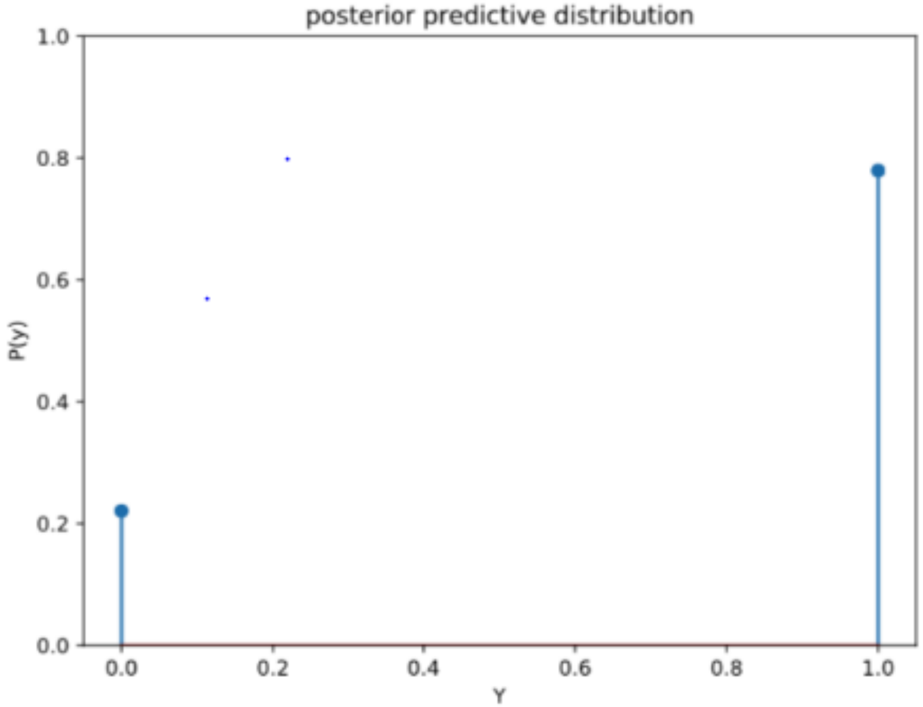
$$P(Y=1|D) = \int_0^1 \theta\cdot 4\cdot\theta^3 d\theta = \frac{4}{5}\cdot\theta^5\ |_0^1$$

$$P(Y=1|D) = \frac{4}{5}\cdot 1^5 - \frac{4}{5}\cdot 0^5 = 0.8$$

$$P(Y=0) = 1 - P(Y=1) = 0.2$$

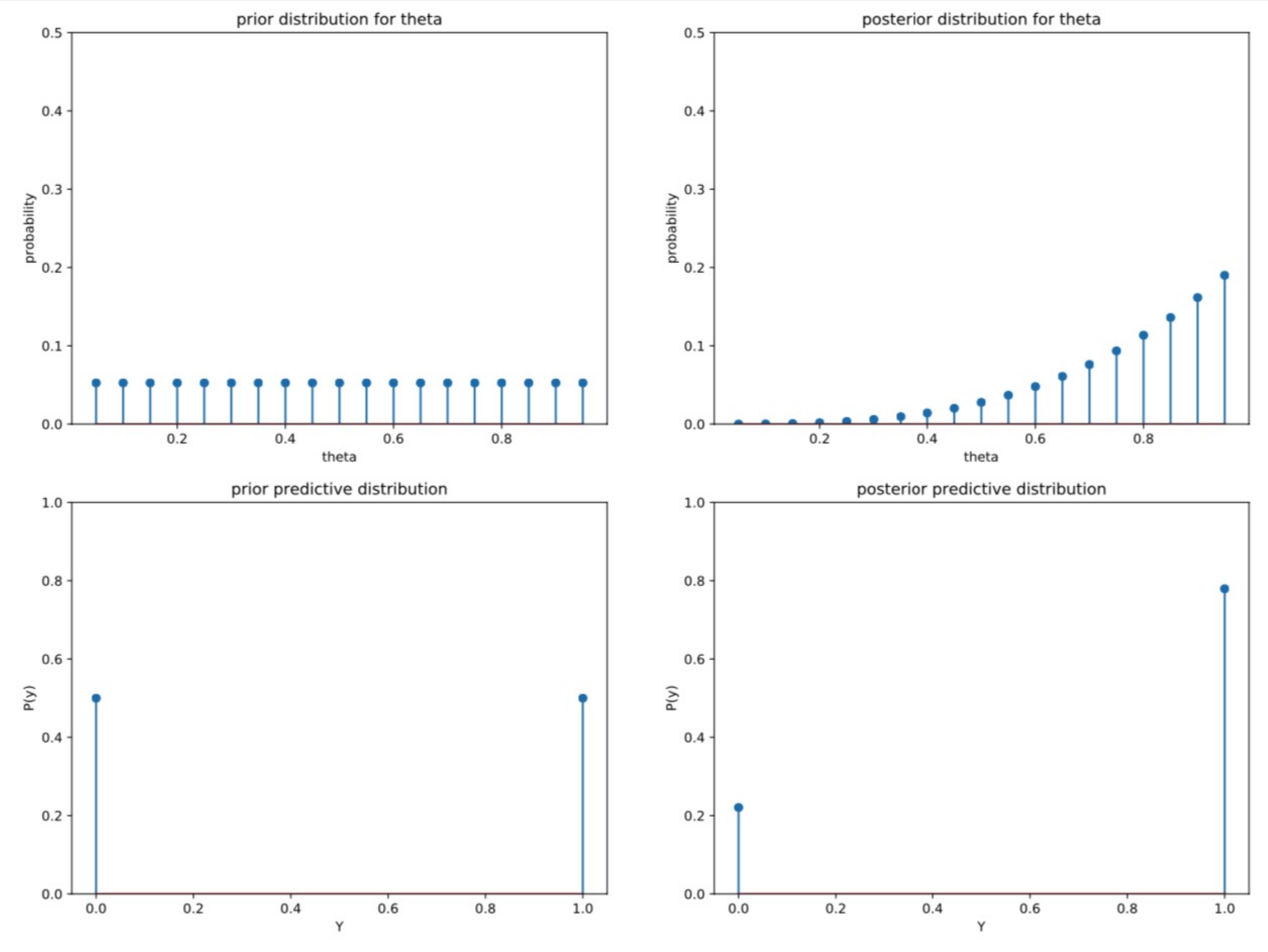
Prior and Posterior Predictive distribution





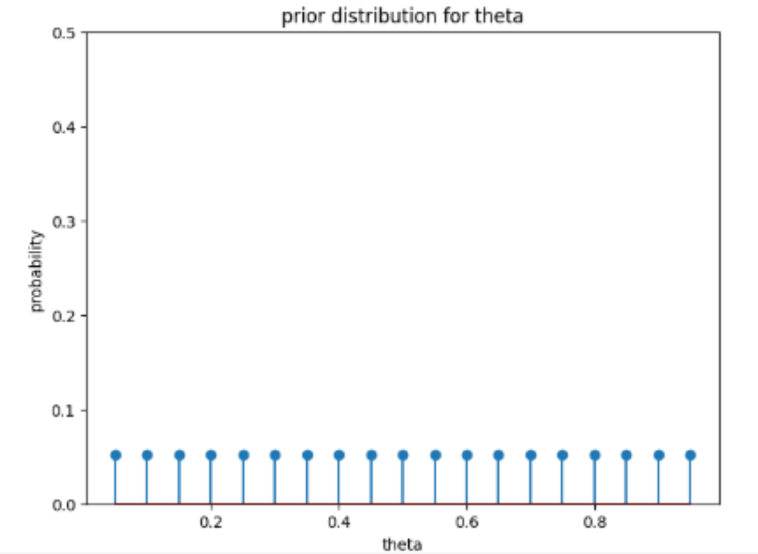
Coin example «the hacker's way»

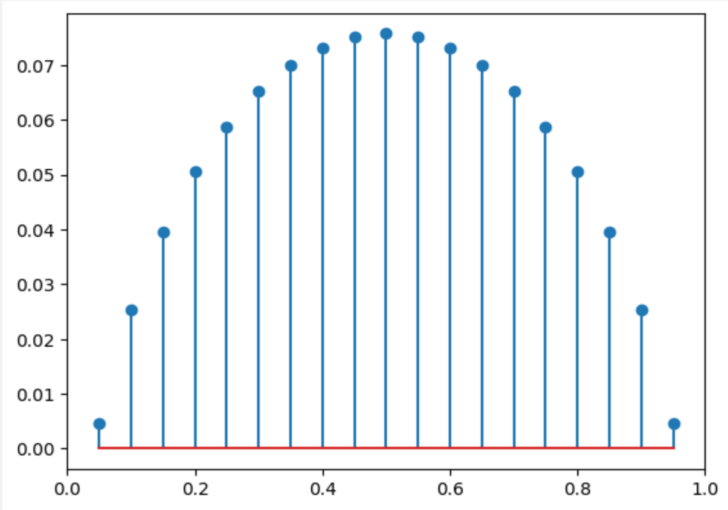
Work through the notebook NB21 and do the excerise therein.



Result 40+11 times head and 9 tails

Prior





Posterior