Machine Intelligence:: Deep Learning
Week 7

Beate Sick, Pascal Bühler, Oliver Dürr

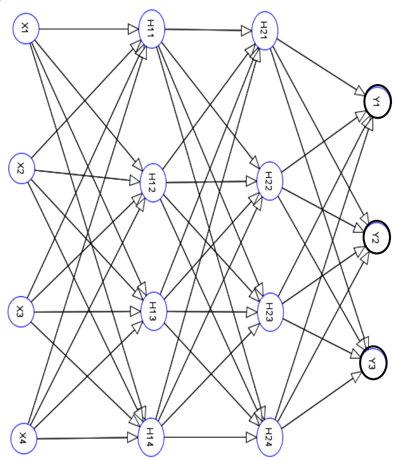
Ensembling approaches for improving the performance and uncertainty estimates of NN models by taking into account the algorithmic epistemic uncertainty.

Outline:

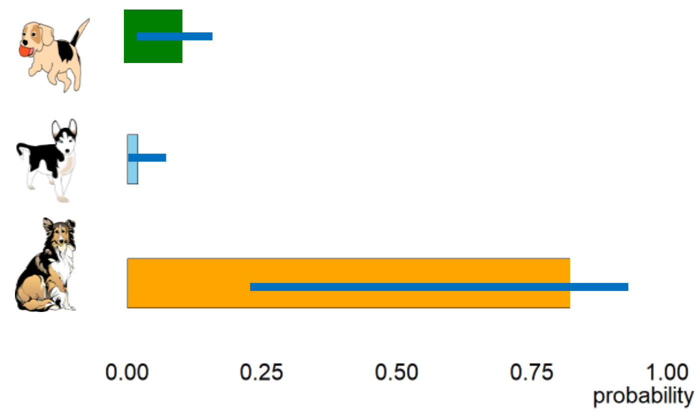
- Bayes
  - Motivation of Bayesian Models
  - Bayes the Hackers way
  - Approximative Bayes via Variational Inference
  - Pseudo approximative Bayes via MC Dropout

A non-Bayesian NN cannot ring the alarm

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



Plain wrong !



We need some error bars!

Recall some points on uncertainty from last lecture

- We have different uncertainty components when working with NN models
  - Model choice uncertainty
    - which model/architecture should we use?
  - Aleatoric uncertainty = data inherent variablilty
    - We capture aleatoric uncertainty by the spread of the predicted conditional probability distribution
  - Algorithmic uncertainty
    - Training twice the same NN-architecture with the same data does not yield the same trained model
    - Random intialization, random mini-batch splits, random augmentation
  - Epistemic uncertainty
    - The lack of knowledge due to a lack of information, such as too few data or a lack of understanding leading to the wrong model choice

- Deep Ensembling is always good to get a better model
    - Better prediction performance: The NLL of an ensemble is better or equal than the mean NLL of the members
    - We can quantify the algorithmic uncertainty of the model

Bayesian Approach

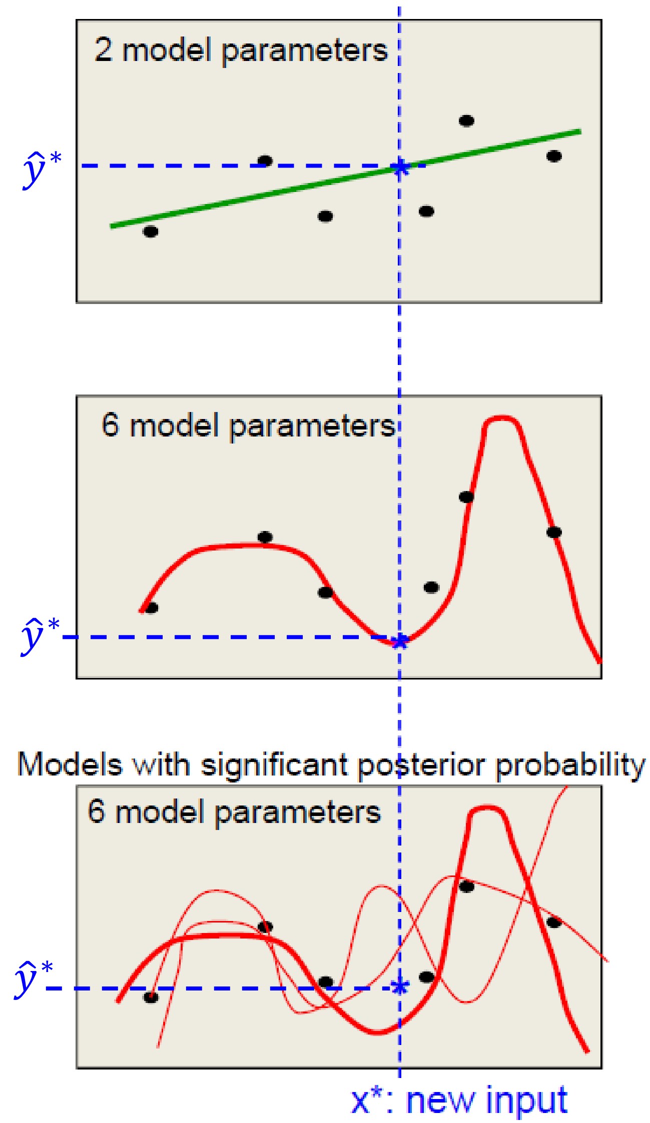
Bayesian prediction models do also kind of ensembling

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.



Conditional outcome distribution for each θ another distribution

$$p(Y|x^*, D) = \int p(y|x^*, \theta) \cdot w(\theta) \, d\theta$$

$\hat{y}^*$=E(Y|x*,D)

Weighted average of different predictions
the weights are the «posteriors of the the parameters» tuned w.r.t. the train 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*

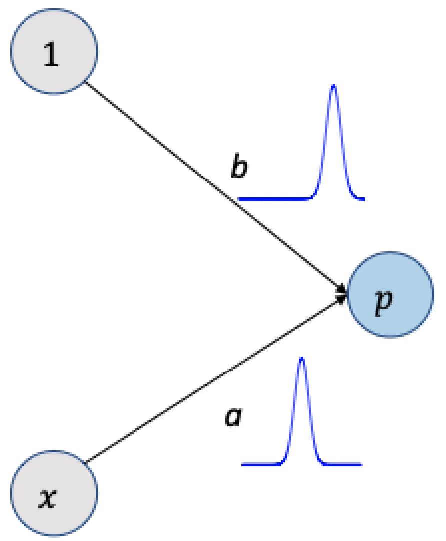
$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}$$

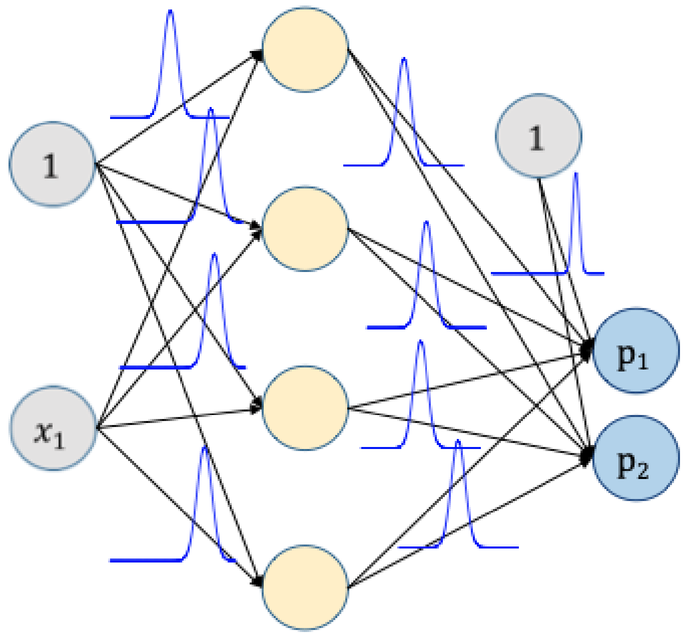
- $p(\theta|D)$ **posterior distribution** or the model parameters θ after seeing the train data D
- $p(D|\theta)$ likelihood of the train data D
- $p(\theta)$ prior or the model parameters θ before seeing any data
- $p(D)$ evidence (just a normalization constant)

Bayesian NN

Bayesian Neural Networks * (BNN)

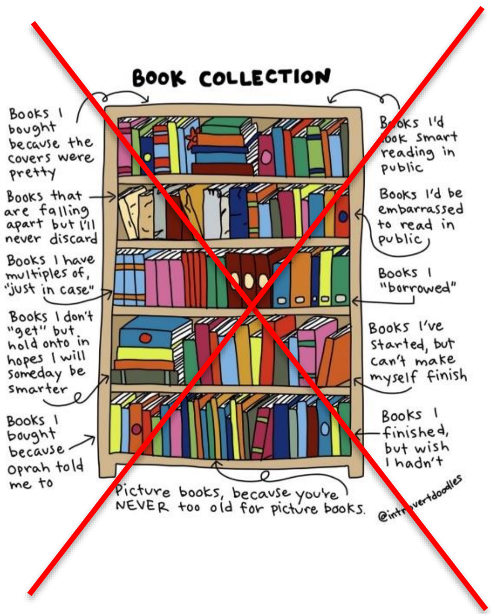
- In a fitted Bayesian Model, such as a Bayesian NN, the parameters (weights) are not fixed numbers but follow a posterior distribution





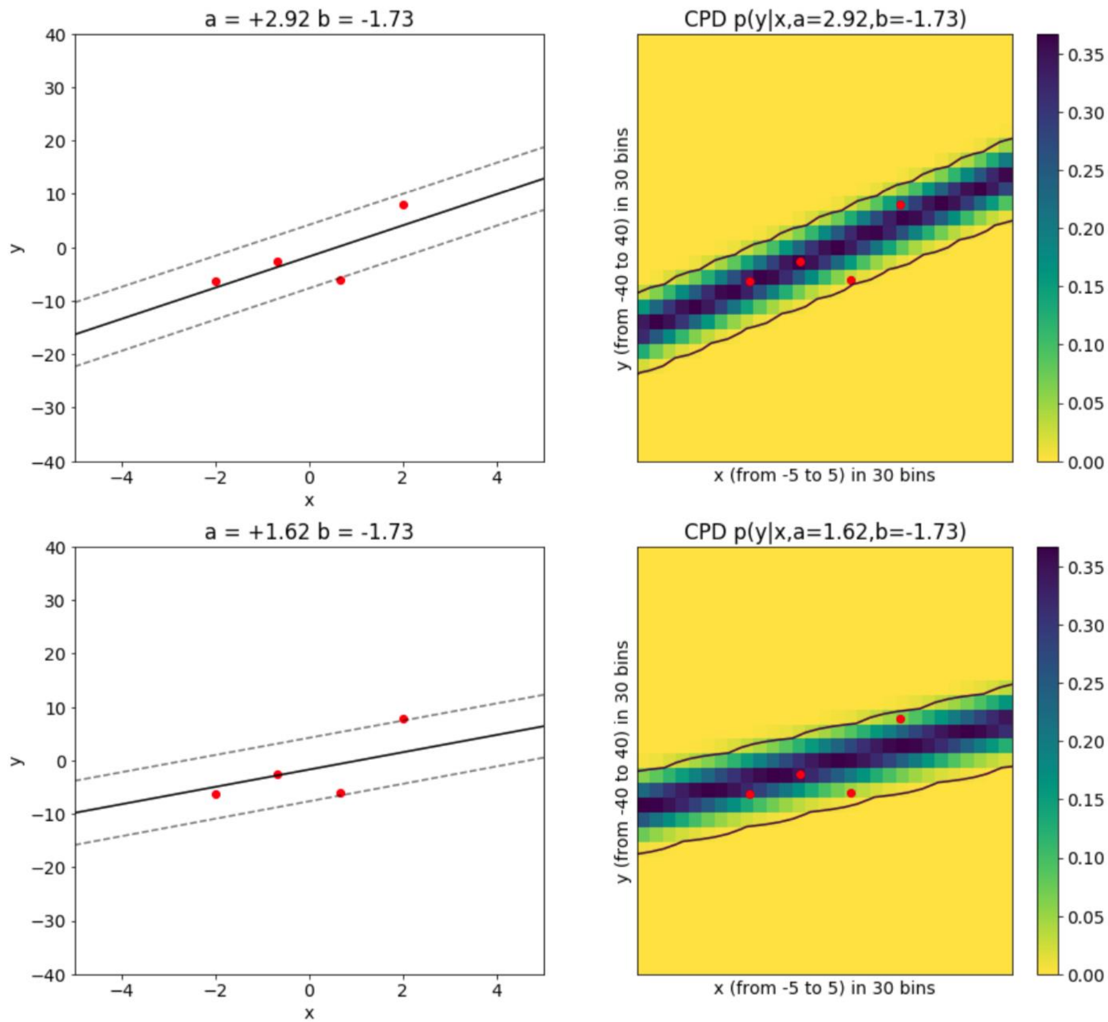
*Don't confuse Bayesian Networks (DAGs) with Bayesian Neural Networks

Bayes the hacker's way



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.



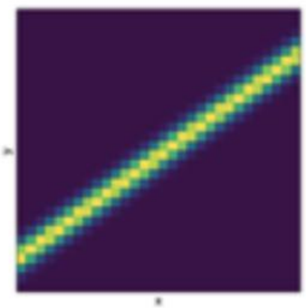
Model with parameter values (a_{ML}, b_{ML}) yielding the maximial likelihood for train data (usual way)

Model with parameter values (a, b) yielding a slightly smaller likelihood for train data

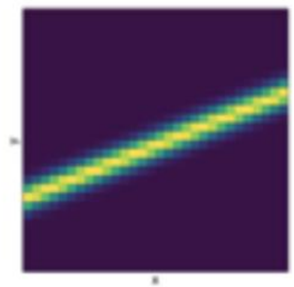
Bayes the hackers way: Get predictive distribution by averaging models based on different parameter values

Take several models with different parameters into account and weight them

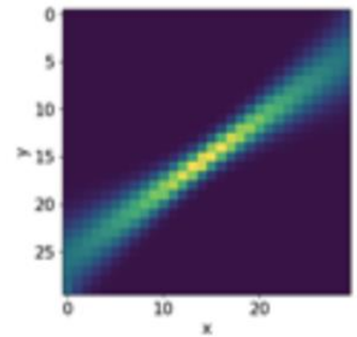
0.09



+ 0.001



+ ...=



$p(y|x, D)$

Hightest weight for MaxLike

Less likely solutions get
a smaller weight

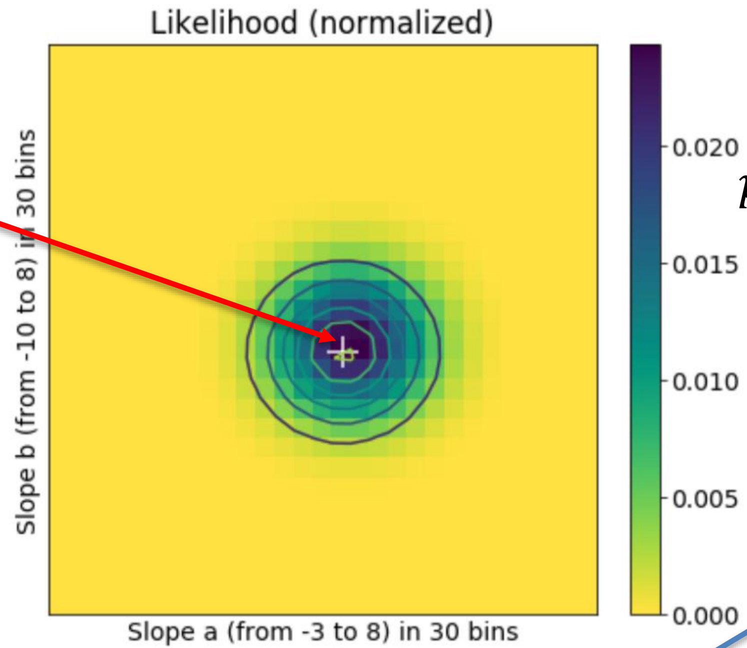
Question: How to get the weight?

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

General idea: Don't put all egg's in one Basket

- Also take other parameters a, b into account, even if the corresponding likelihood is slightly smaller as the ML solution

ML solution



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

Normalized likelihood
$p_{\text{norm}}(D|(a,b))$
serve as weights

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

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

Resulting condtional posterior predictive distribution

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

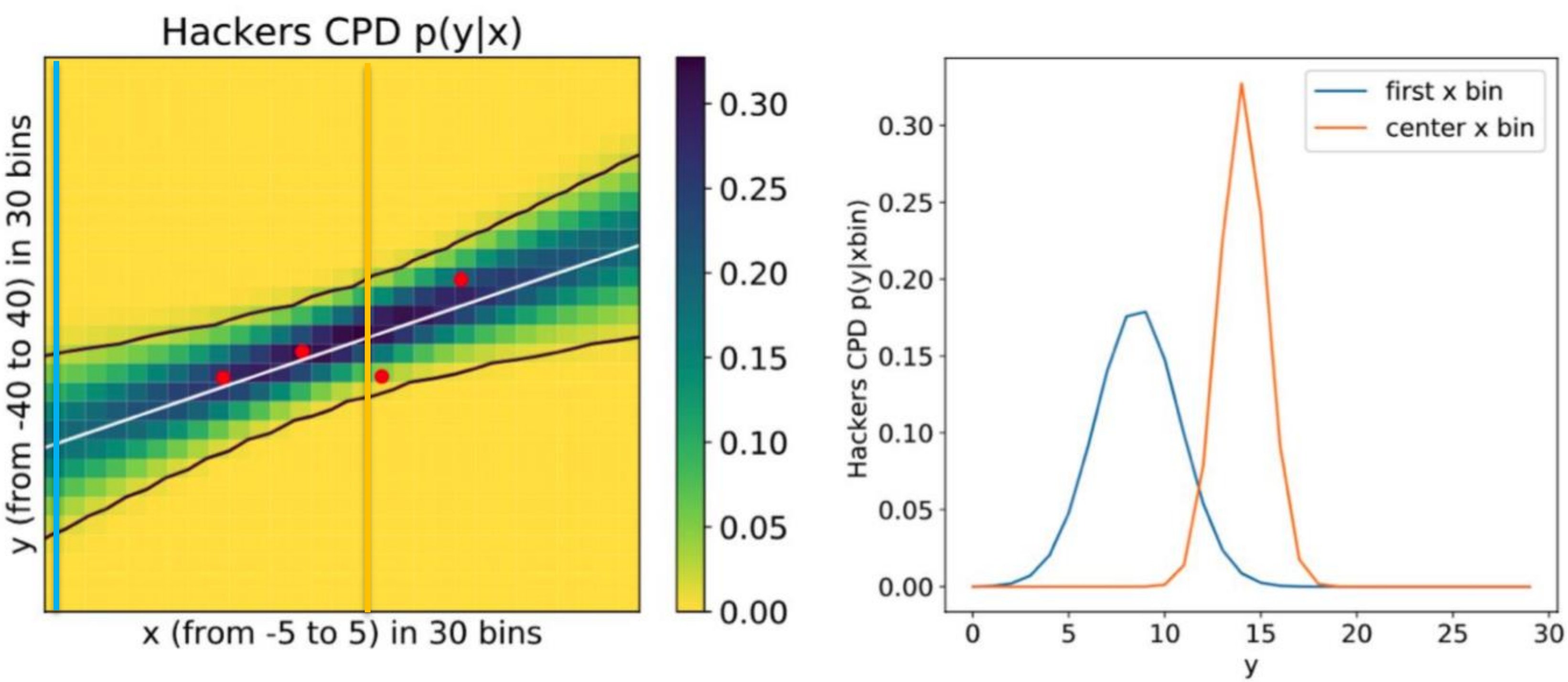
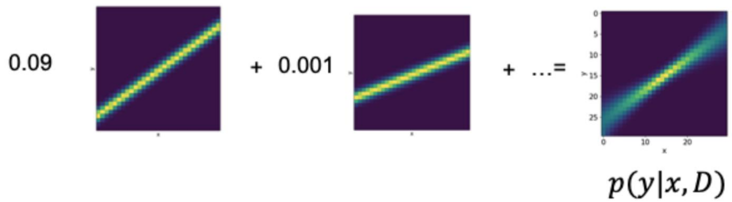


Figure 7.6 The predictive distribution for the Bayesian linear regression model, trained with the four data points shown on the left side by the color-code and on the right as conditional distribution at two different *x* positions. You can clearly see that the uncertainty gets larger when leaving the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

Revisiting the Hackers' example: Get Bayesian predictive posterior as ensemble by weighted averaging different models



- Hacker's way
  - weight / Normalized likelihood

    $$p(y|x,D) = \sum p(y|x,w) \cdot p_{\text{norm}}(D|w) = \sum p(y|x,w) \cdot \frac{p(D|(a,b))}{\sum_w p(D|(a,b))}$$

- Bayes
  - weight / posterior

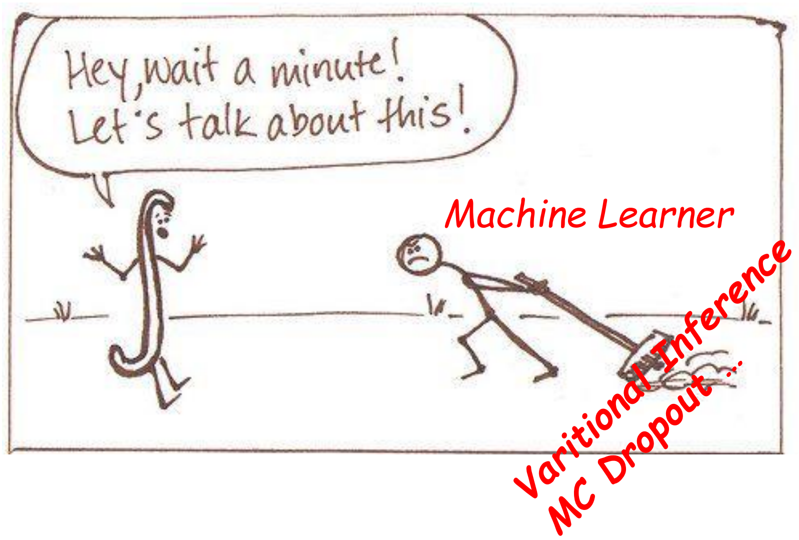
    $$p(y|x,D) = \sum_w p(y|x,w) \cdot p(w|D) = \sum_w p(y|x,w) \cdot \frac{p(D|w)p(w)}{\sum_w p(D|W)p(w)}$$

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

Computing the posterior can get difficult due to $\int_\theta$

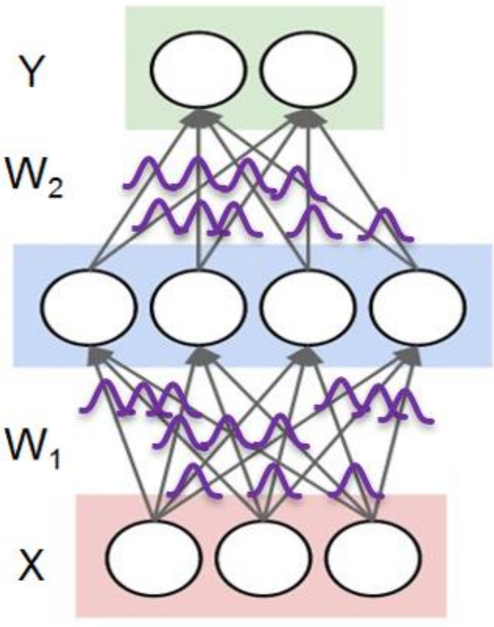
Posterior : $$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\int_\theta p(D|\theta)p(\theta)d\theta} \sim p(D|\theta)p(\theta)$$

- $p(D|\theta)$ likelihood - easy ☺
- $p(\theta)$ prior – easy ☺
- $\int_\theta p(D|\theta)p(\theta)d\theta$ - hard ☹

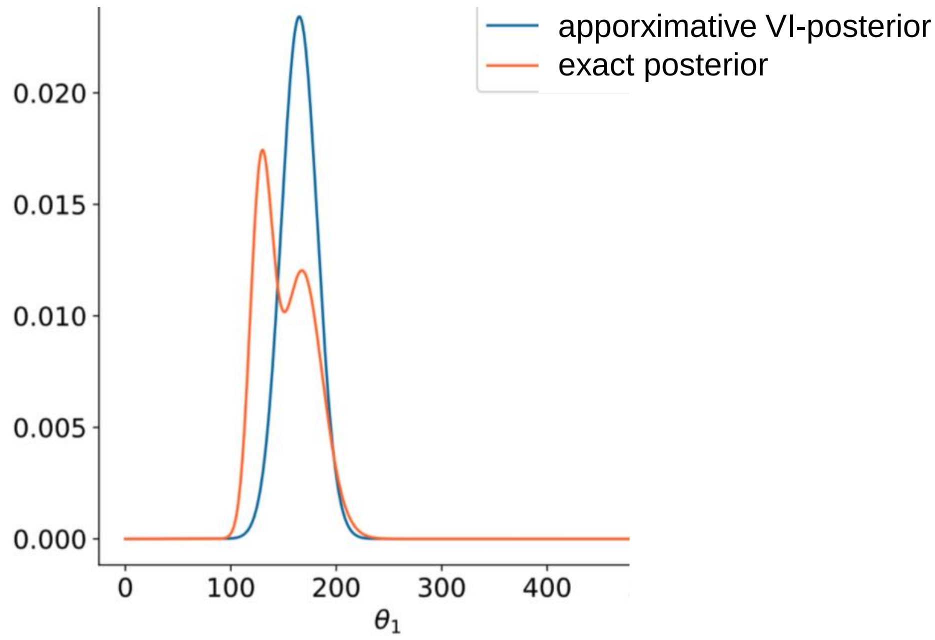


- Bayesian Mantra:
  The posterior is proportional to the likelihood times the prior

Variational Inference to get an approximate posterior



Approximative Bayes via Variational Inference (VI)



Approximate exact posteior by a variational distribution, often a Gausian.
→ Instead of solving an integral, in VI we solve an optimization problem, i.e. we optimize the parameters of the variational distribution, $\mu,\ \sigma$, to be close to the posterior by minimizing a special loss function.

Theory: see chapter 7 in our Book Probabilistic Deep Learning

Approximative Bayes via Variational Inference

- Layers for Variational Inferene :
  - Use VI Flipout layer instead normal Conv layer

```
model_vi.add(tfp.layers.Convolution2DFlipout(16,kernel_size=(3,3),
                    padding="same", activation = 'relu',
                   kernel_divergence_fn=kernel_divergence_fn))
```

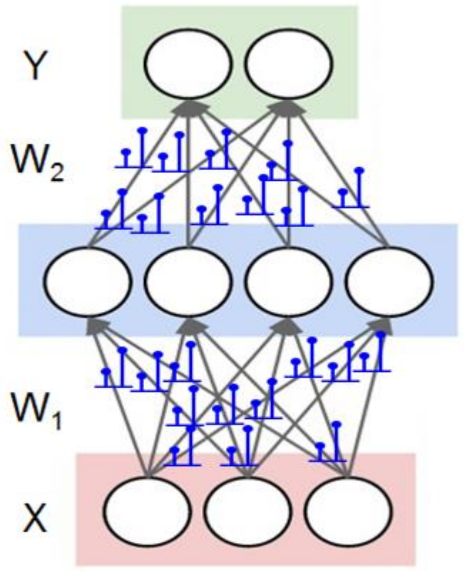
Instead of

```
model.add(Convolution2D(16,kernel_size=(3,3),padding="same", activation = 'relu'))
```

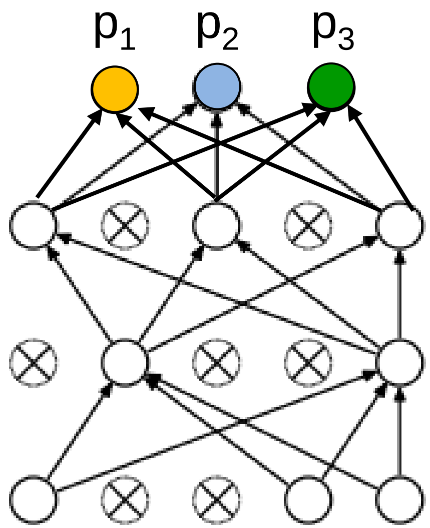
Remark: we skip the theory of Variational Inference

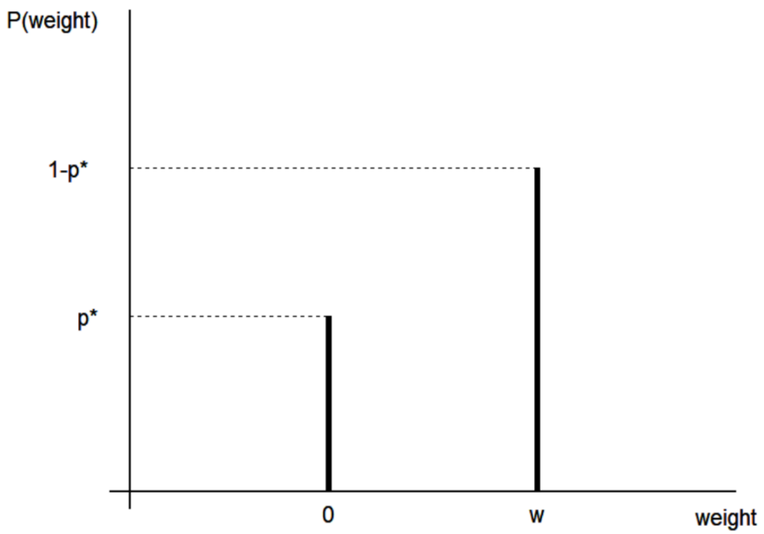
If you are interested in the theory, see chapter 7 in our Book Probabilistic Deep Learning

MC Dropout
as approximate Bayes



Recall: Classical Dropout only during training

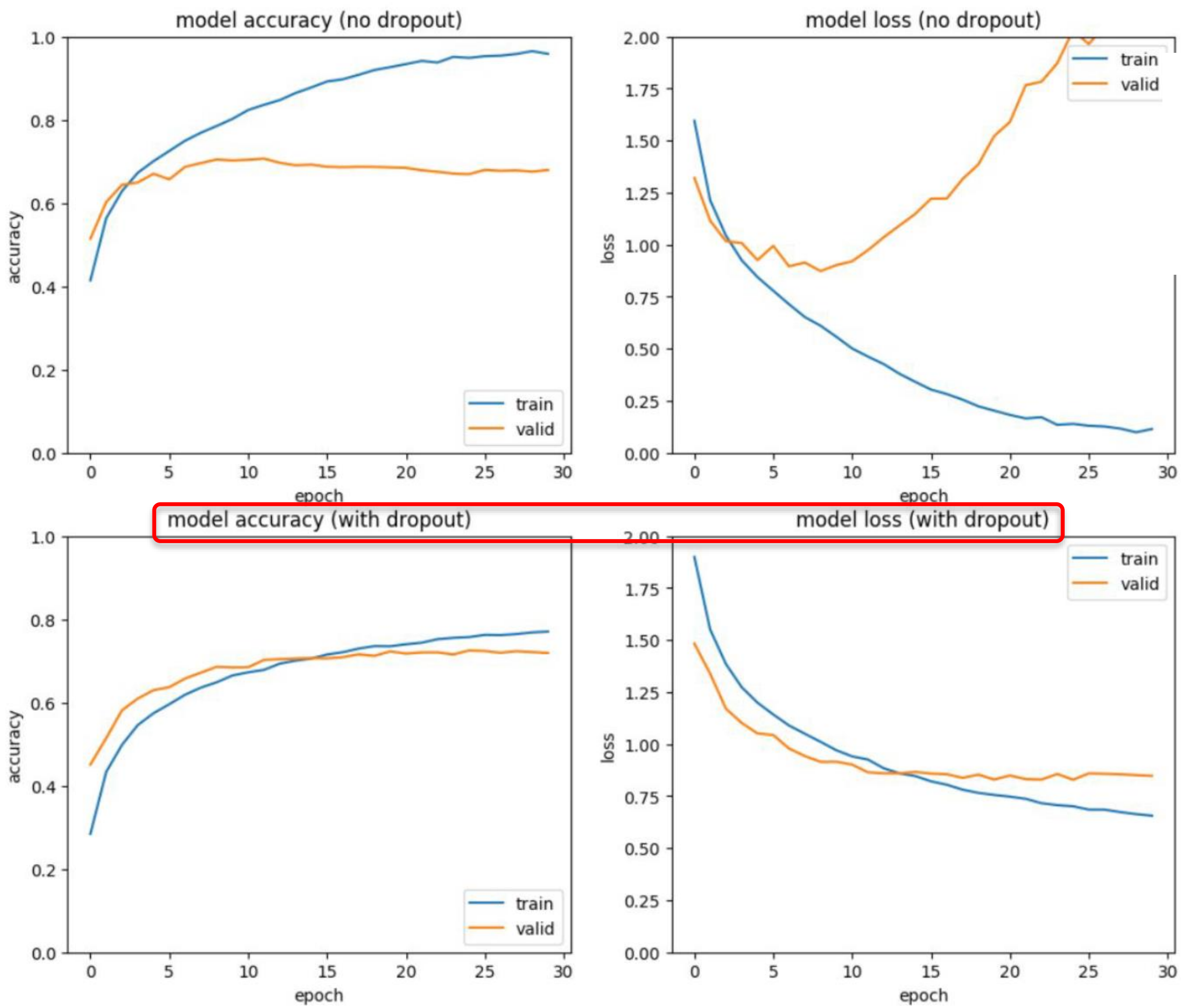


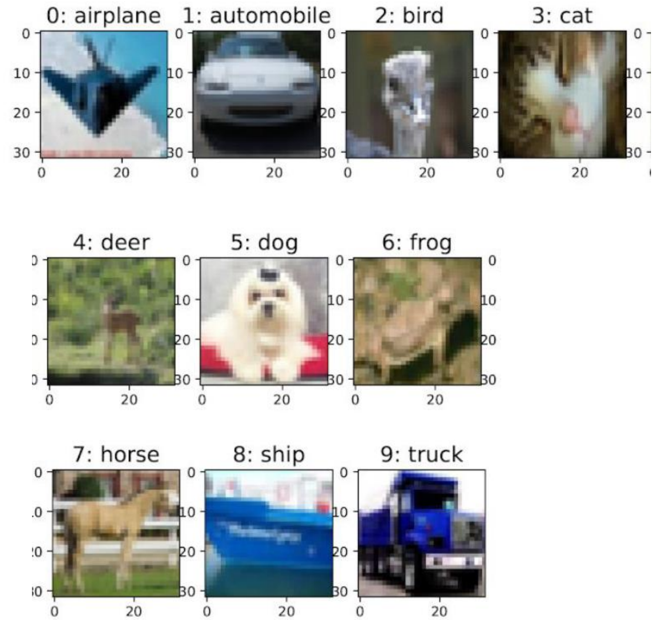


Using dropout during training implies:

- 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





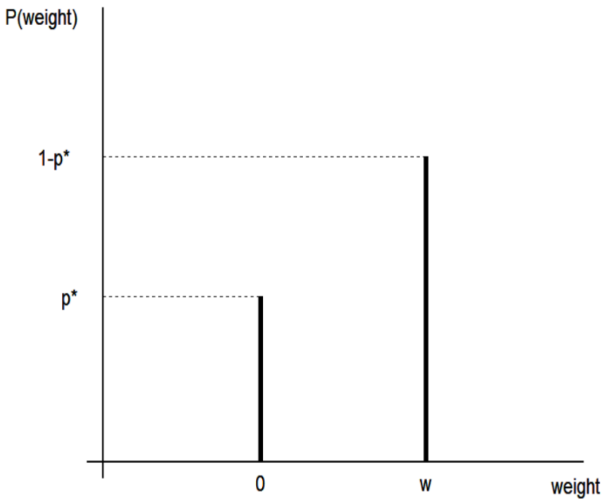
From Dropout during training

to MC Dropout during test time

Bayesian NN via MC Dropout

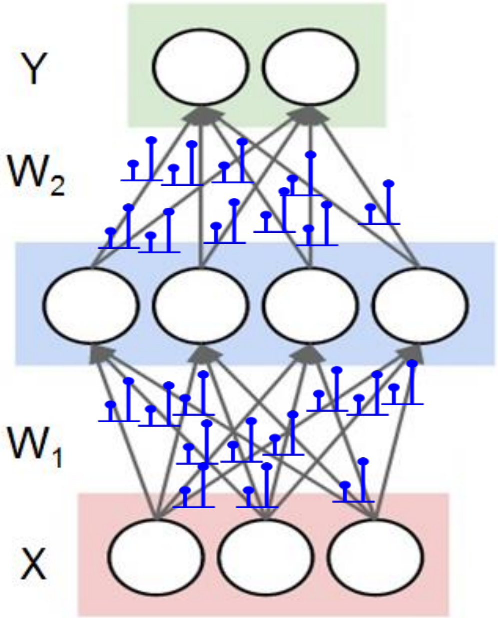
Yarin Gal et al. (2015):
Via Dropout training we learned a whole weight distribution for each connection.
We can sample from this Bernoulli-kind weight distribution by performing dropout during test time and use the dropout-trained NN as Bayesian NN.
Gal showed that doing dropout approximates VI with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

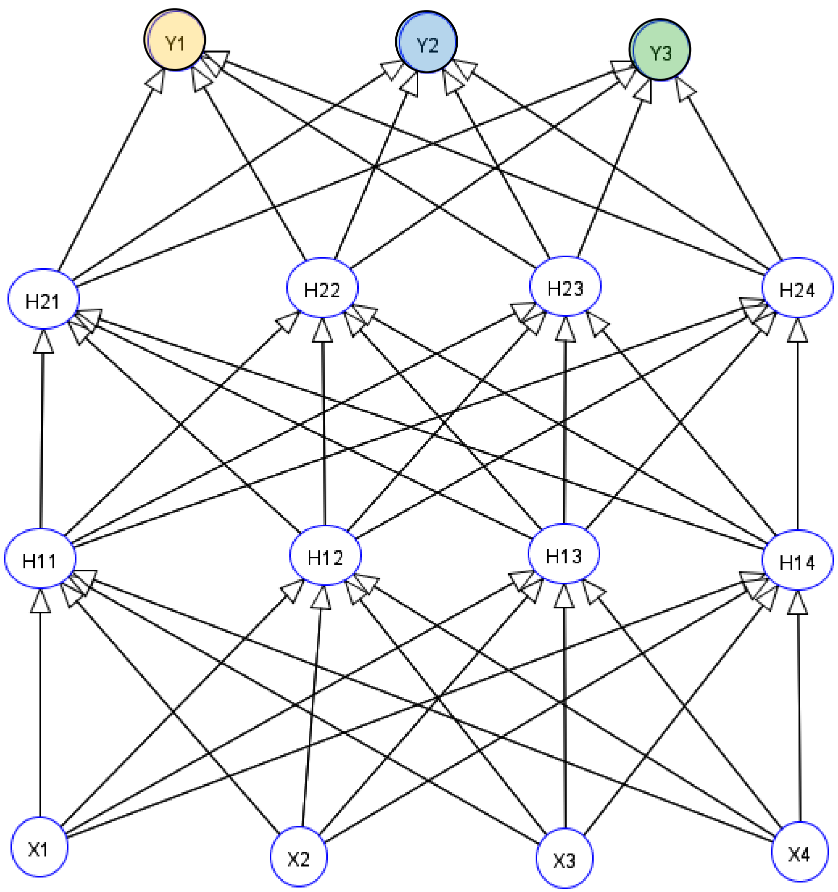
Which parameter has this q_θ?

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

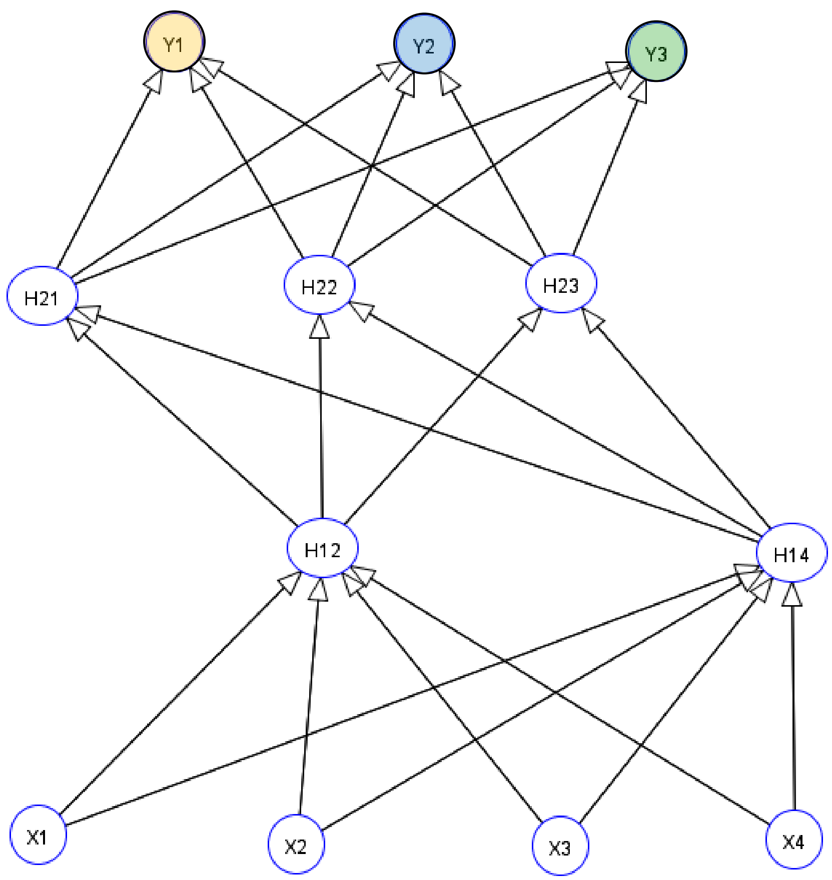


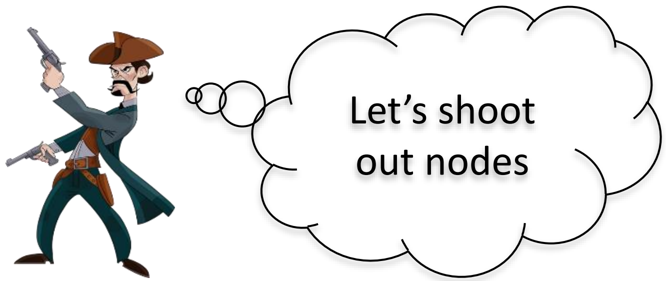
Probability of predicted class: $\mathbf{p_{max}}$

Input: image pixel values

MC Dropout during test time: Run 1

p_1=0.08 **p_2=0.89** p_3=0.03



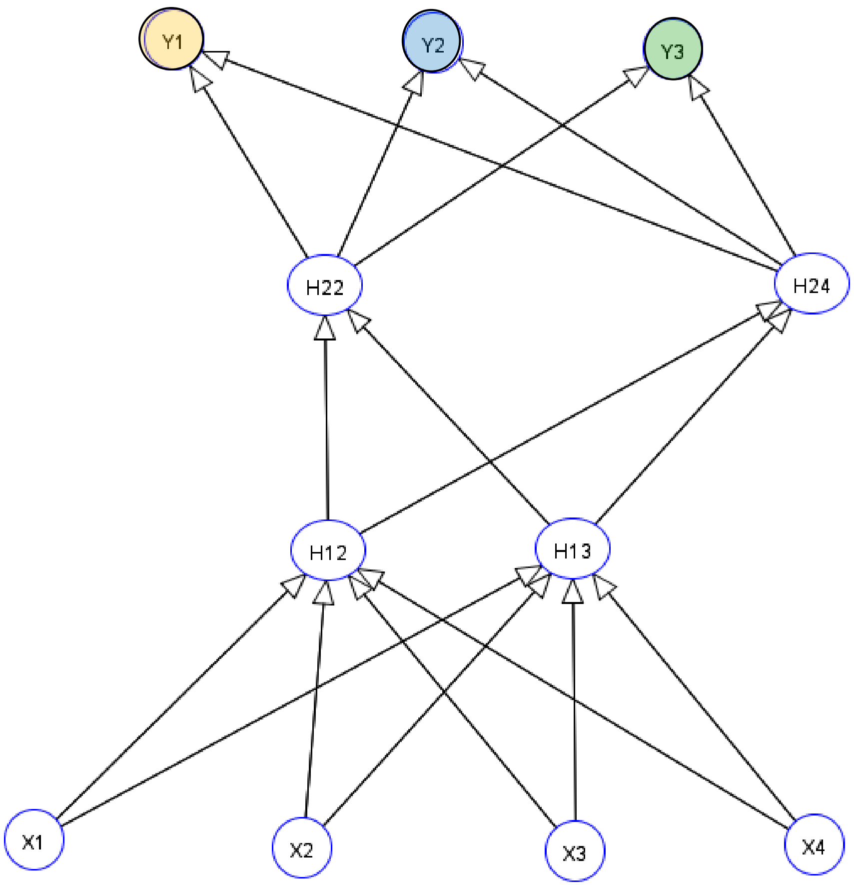


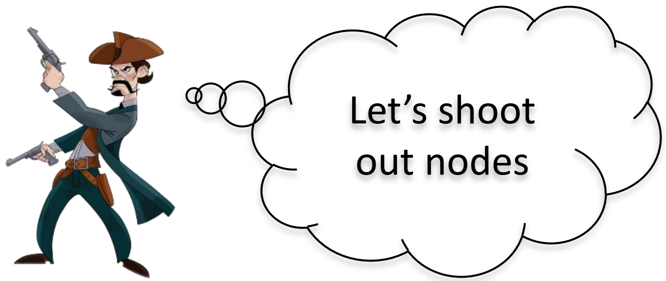
Stochastic dropout of units

Same input image

MC Dropout during test time: Run 2

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

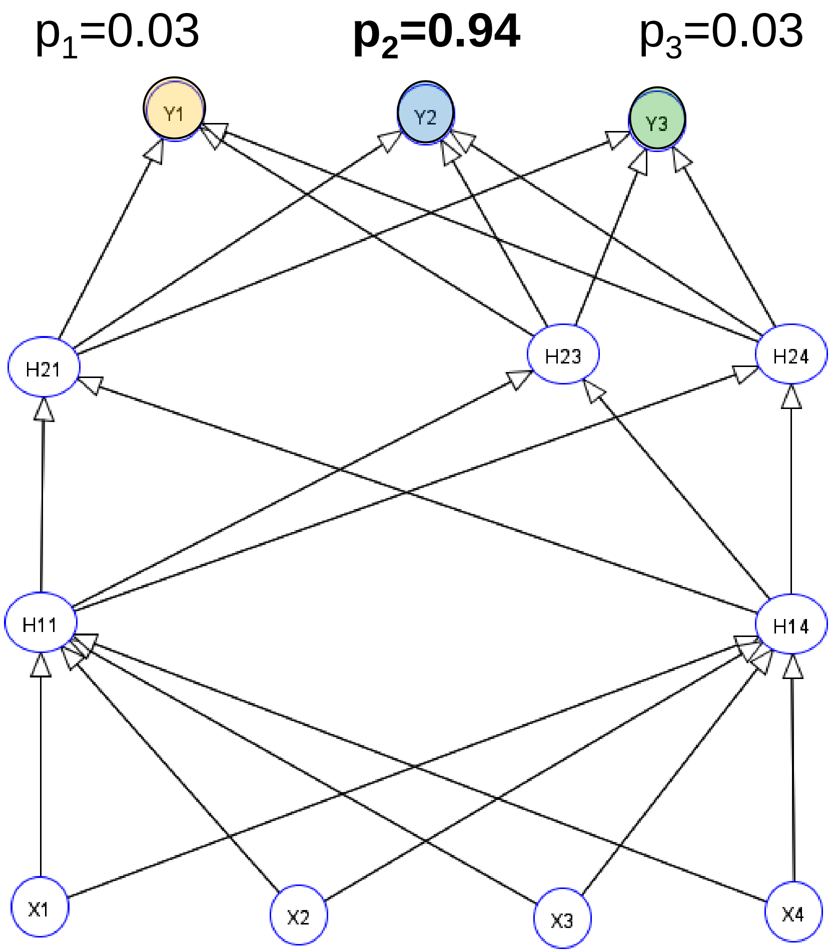


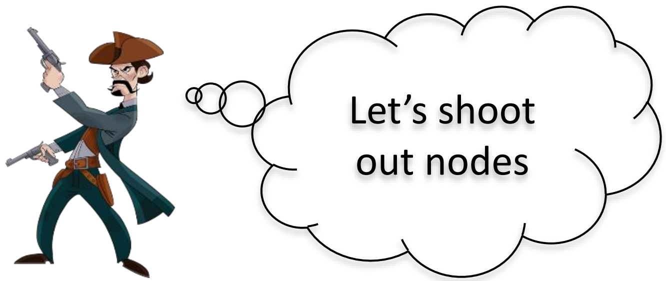


Stochastic dropout of units

Same input image

MC Dropout during test time: Run 3



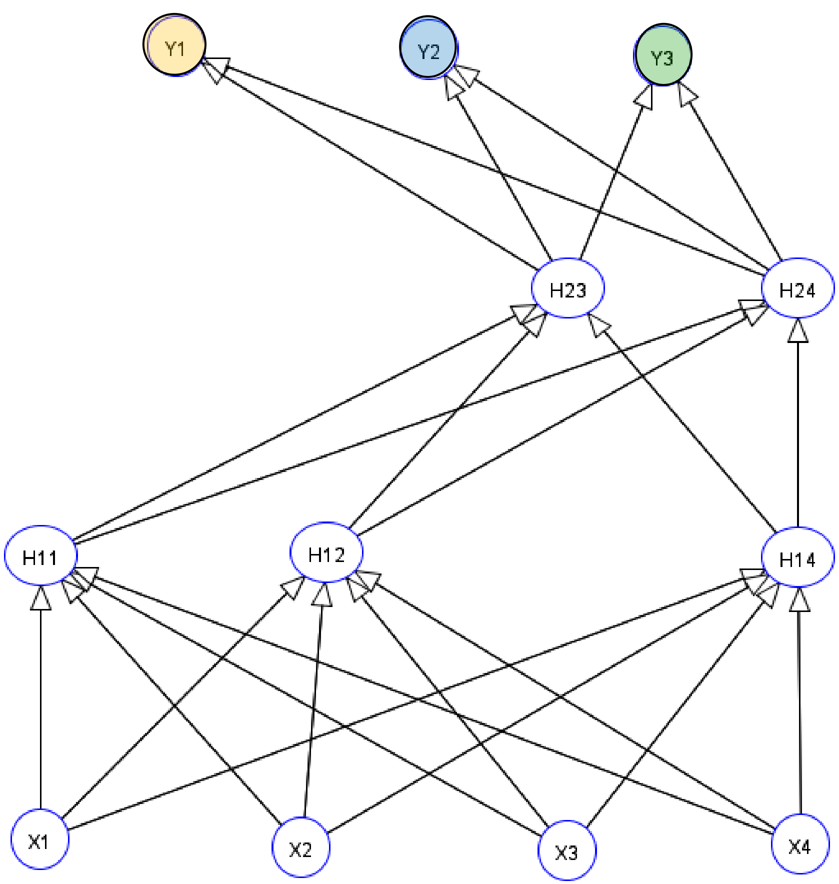


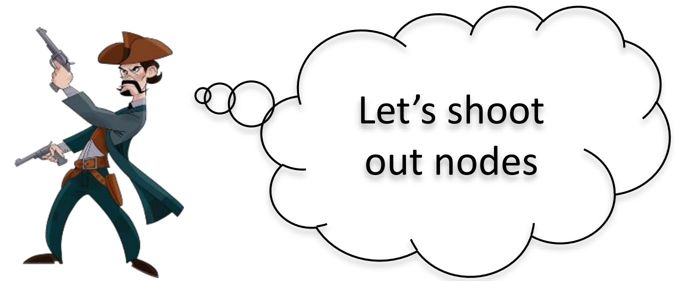
Stochastic dropout of units

Same input image

MC Dropout during test time: Run 4

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

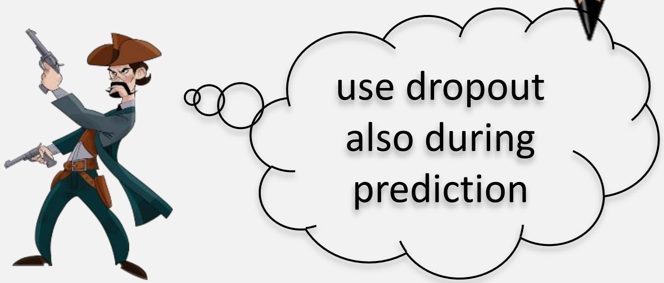




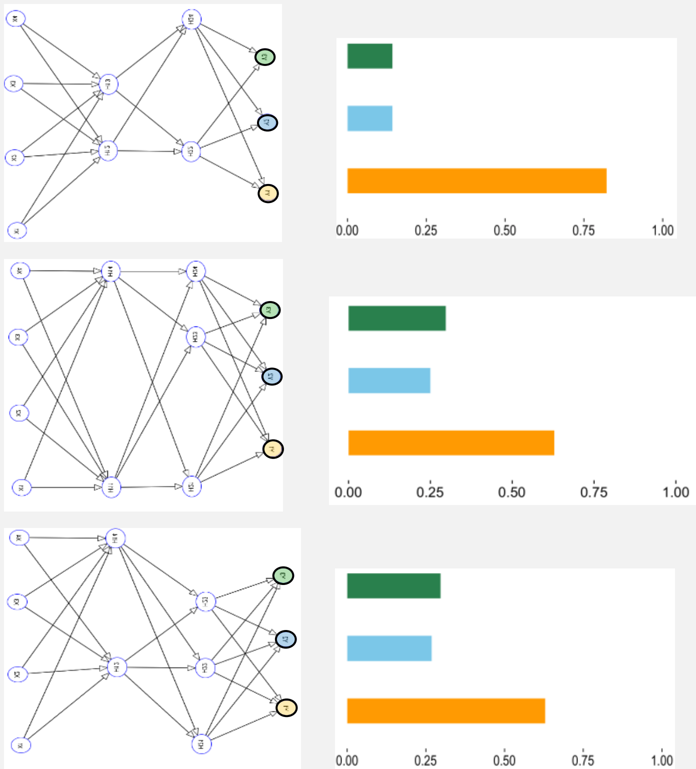
Stochastic dropout of units

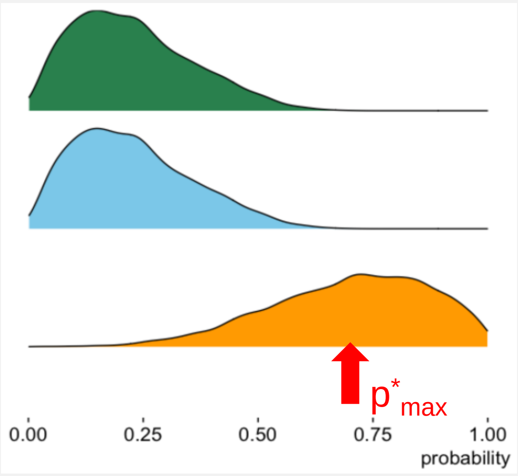
Same input image

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



Many Dropout Runs in forward pass



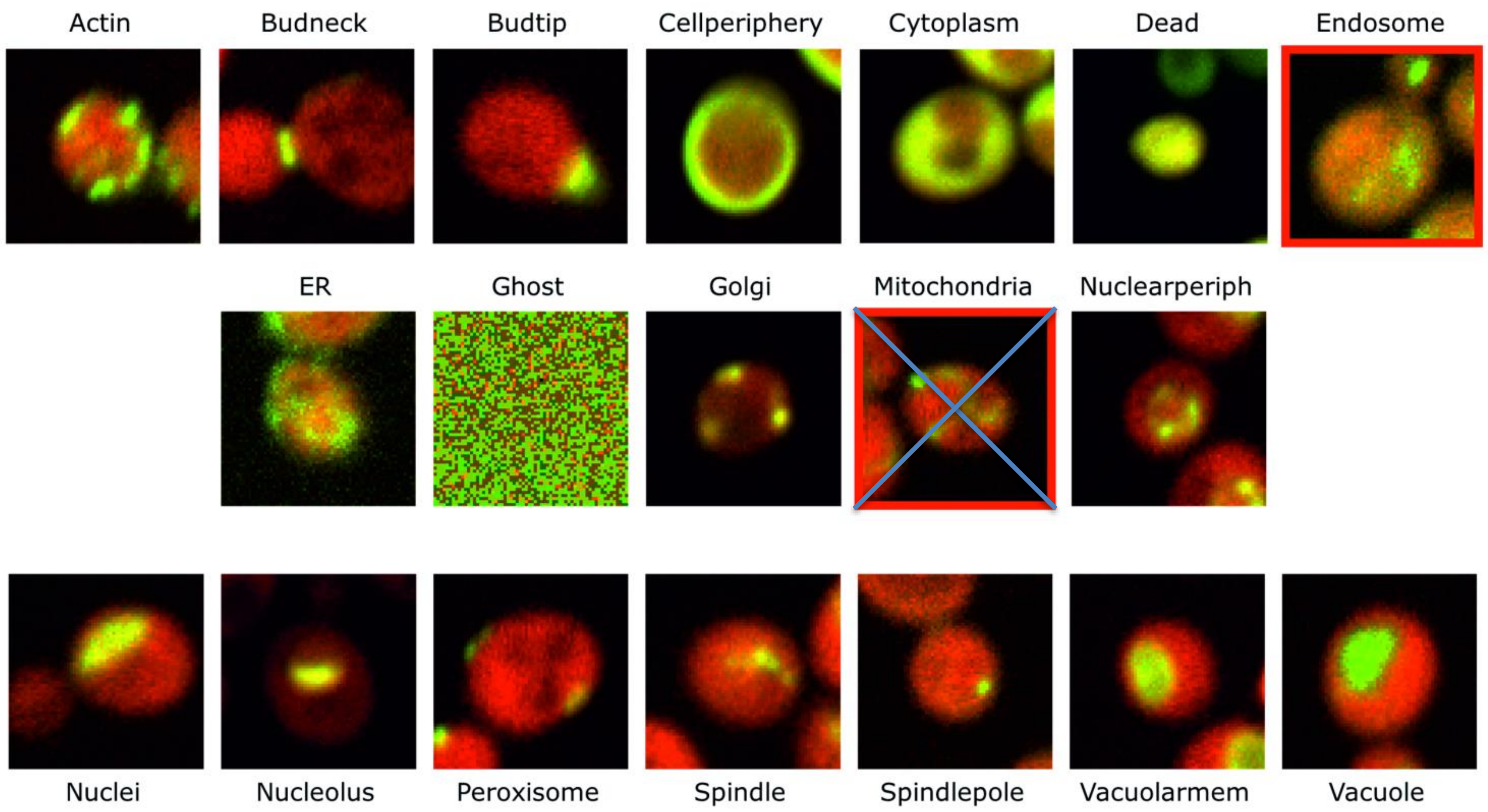


...

CNN predicts class “collie” but with high uncertainty

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

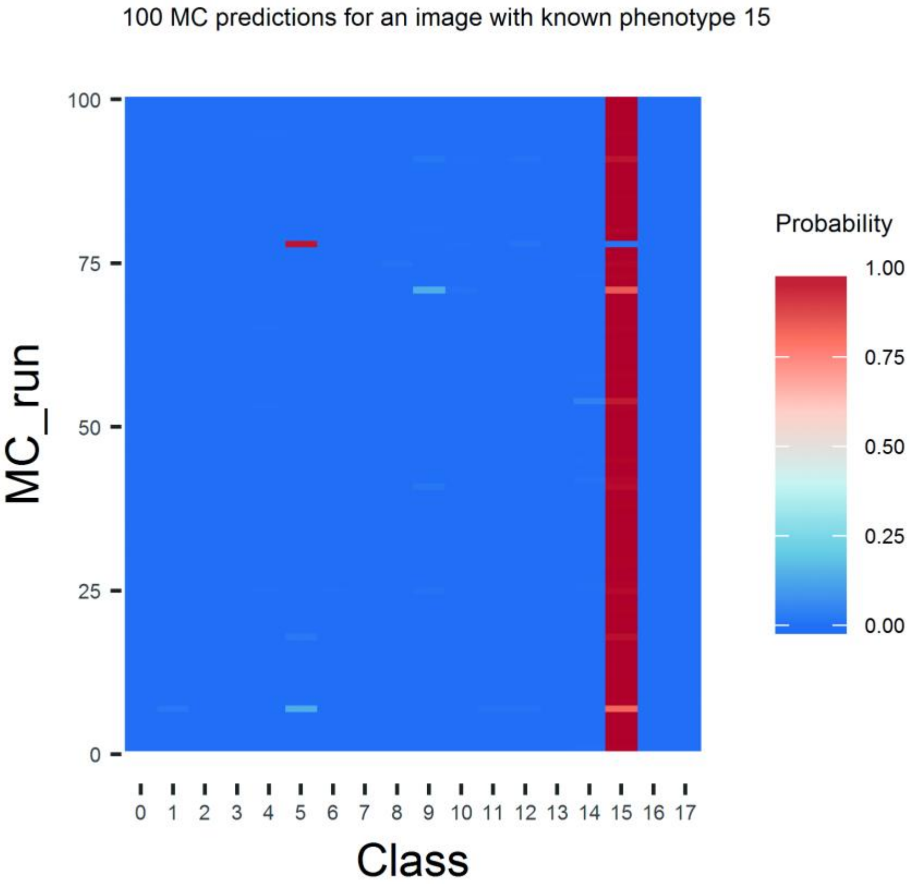
Experiment with unknown phenotype



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

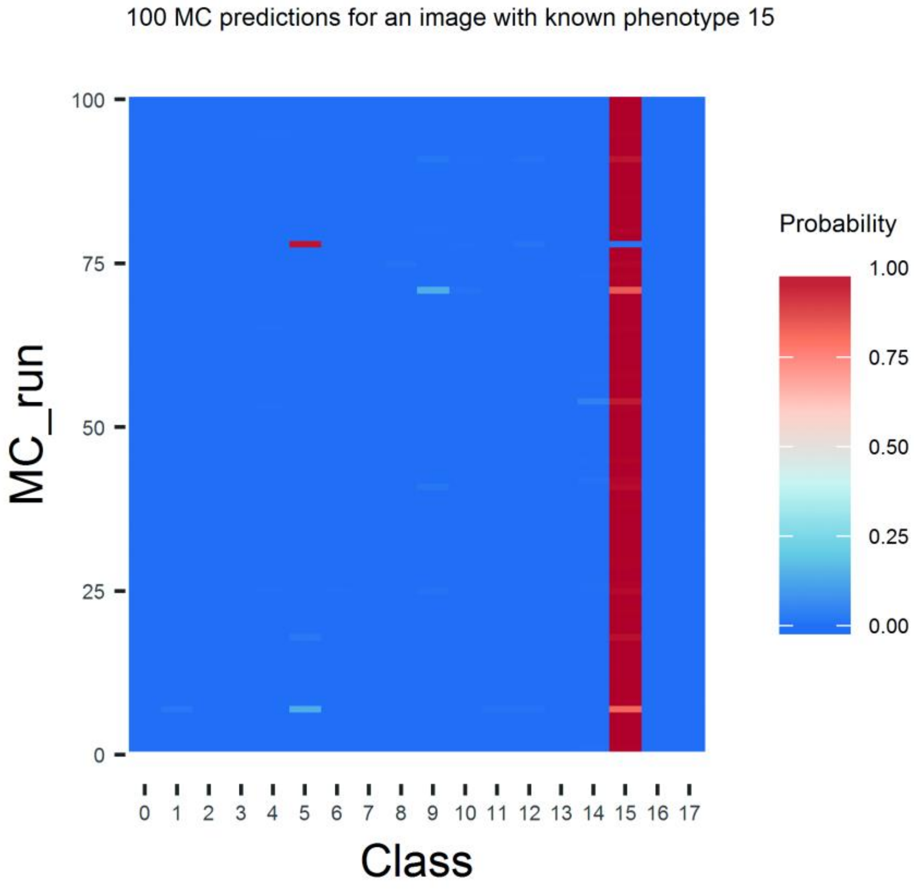
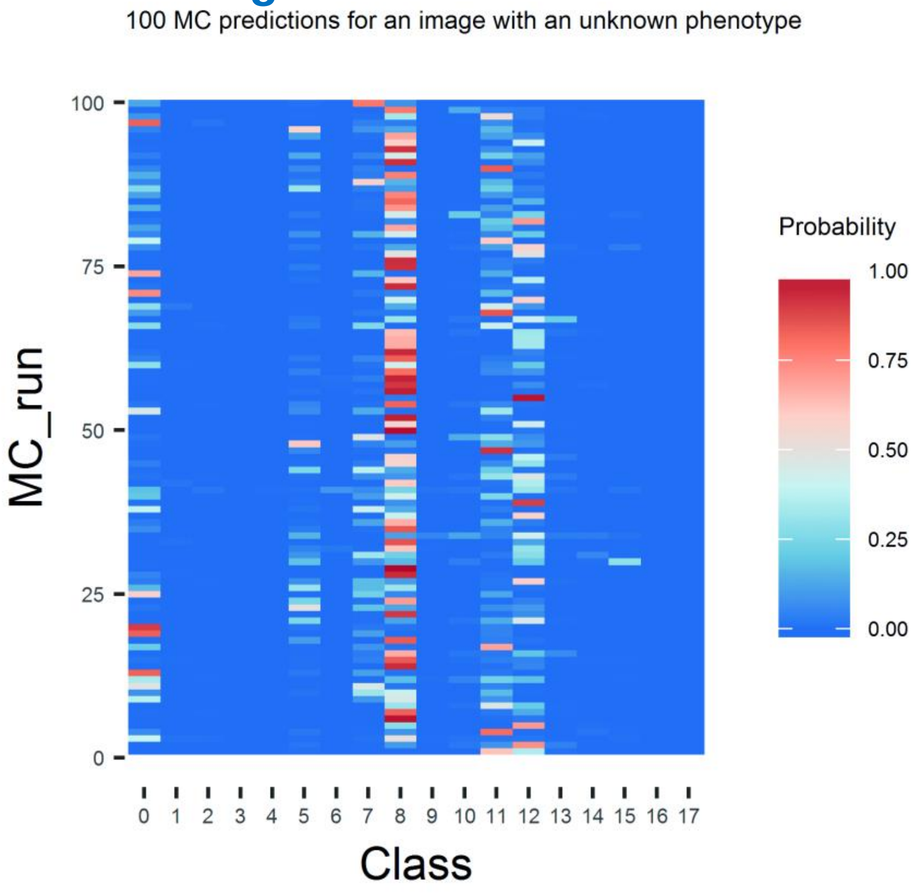


Image with unknown class

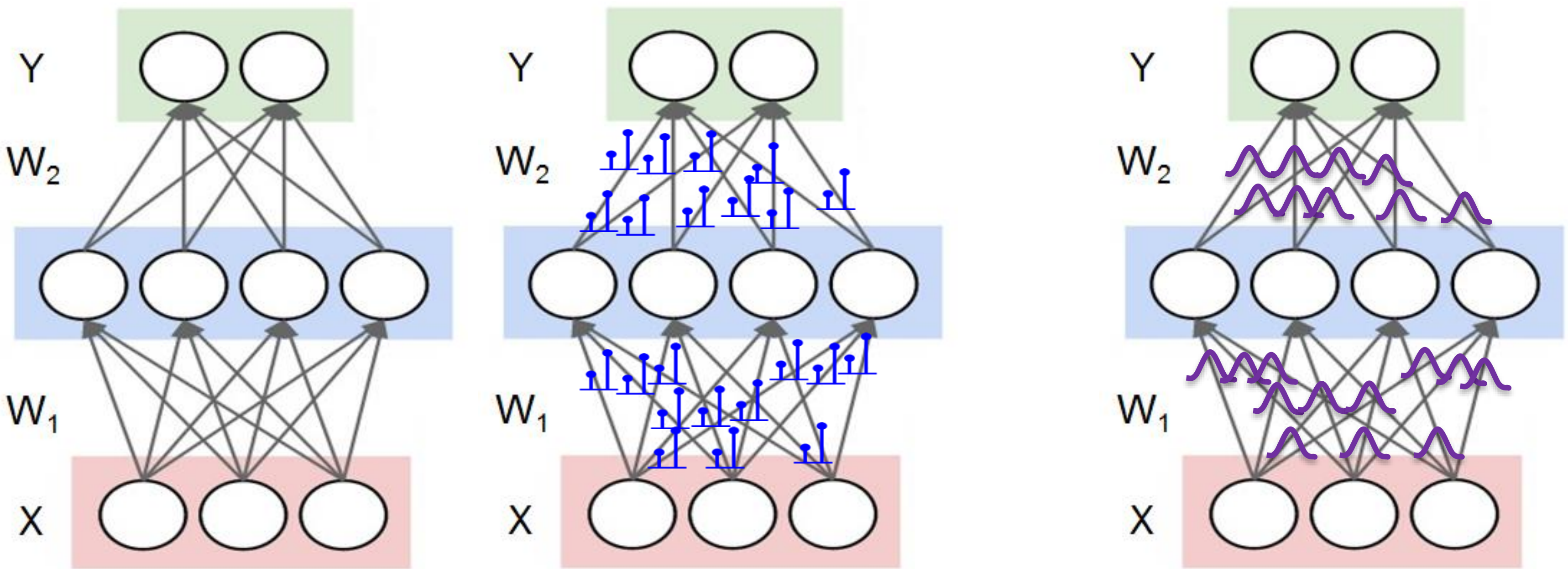


Non-Bayesian and Bayesian NNs

Non-Bayesian NN

MC dropout Bayesian NN

VI Bayesian NN



Weights are fixed

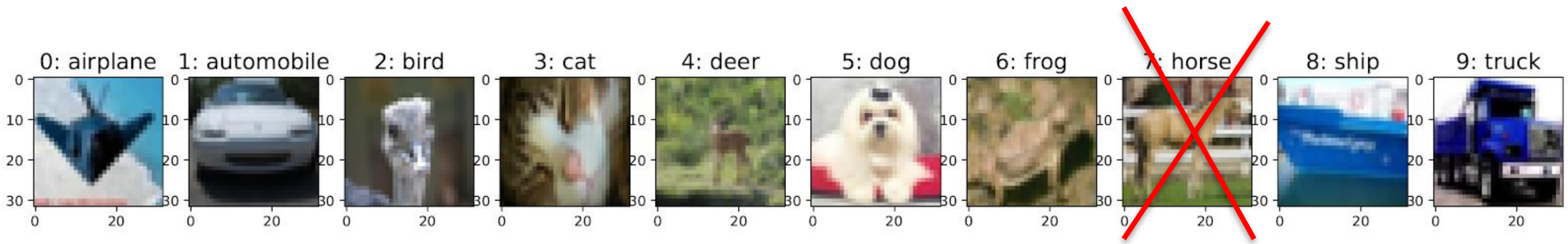
Weights have Bernoulli-kind distribution

Weights have Gaussian distribution

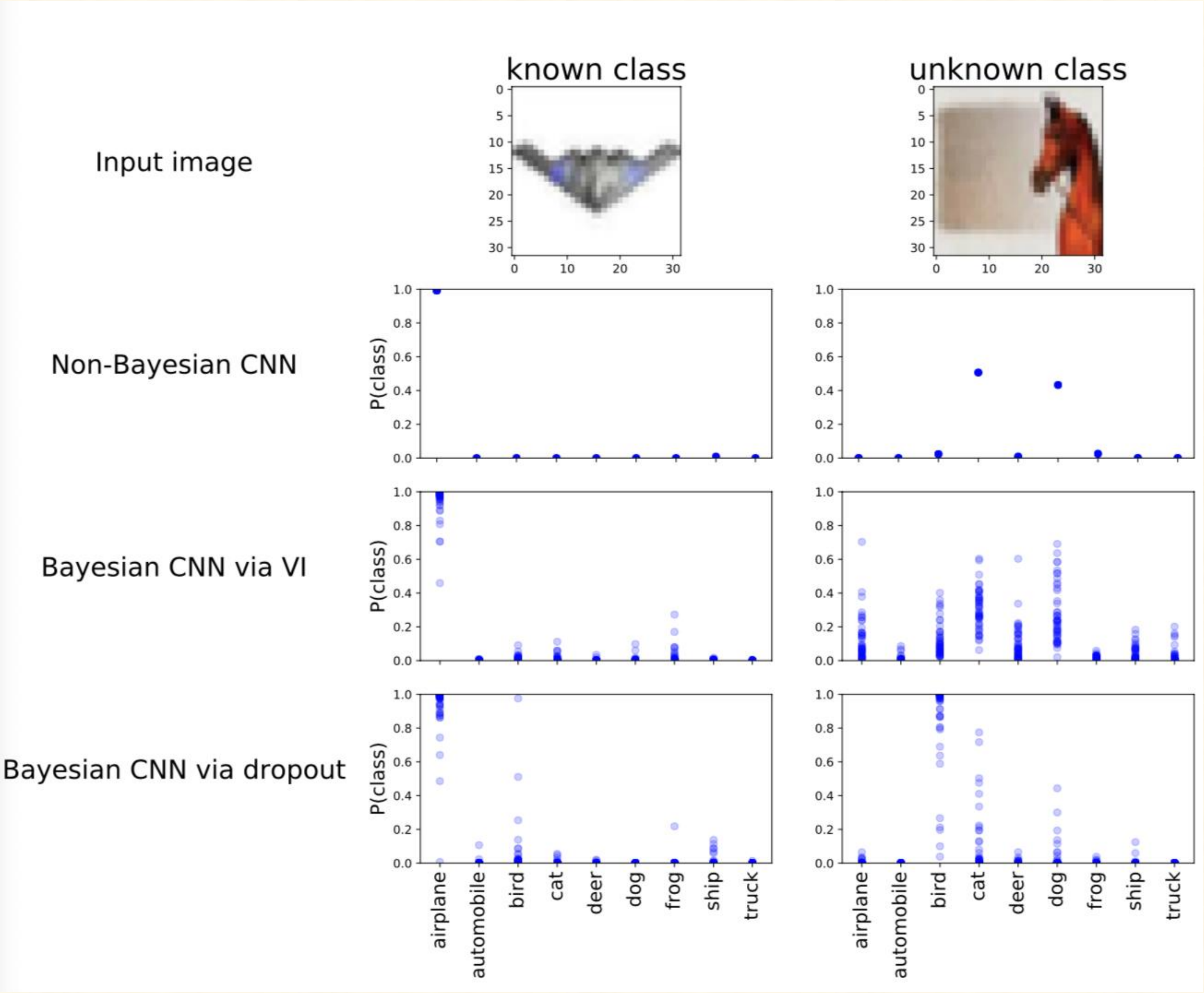
Hands-on Time

[Notebook 17]()

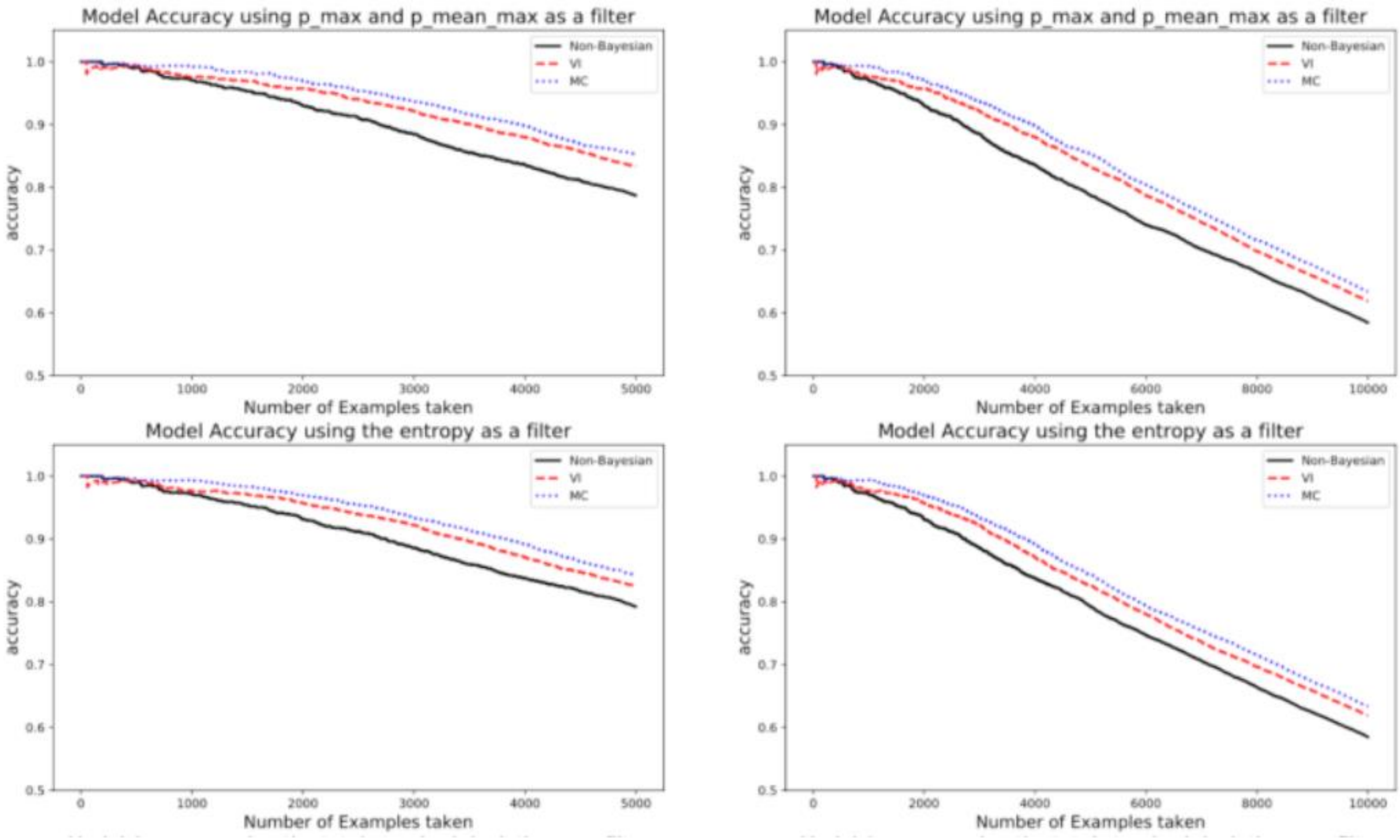
Train a CNN with only 9 of the 10 classes and investigate if the uncertainties are different when predicting images from known or unknown classes.



Looking at the predictive distribution!



Trusting % most «certain» predictions improves performance



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

Take home message

- Using some kind of ensembling or Bayes improve almost all prediction models

- To get a better prediction performance and uncertainty measuers we need to take into account different uncertainty contributions in the model fitting process
  - Work with probabilistic models and fit conditional outcome distributions (capture aleatoric uncertainty)
  - Ensembling of probabilistic models (add algorithmic uncertainty contribution)
  - Bayes (add epistemic uncertainty contribution)
    - Approximative Bayesian NN via Variational Inference (VI)
    - Pseudo apprixmative Bayesian NN via MC Dropout

- For MC dropout we need to fit a NN with dropout which has as many parameters as a classical NN. We also turn on dropout during prediction and 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.