Machine Intelligence:: Deep Learning
Week 7

Beate Sick, Elvis Murina, Oliver Dürr

Institut für Datenanalyse und Prozessdesign
Zürcher Hochschule für Angewandte Wissenschaften

Part I: Preliminary, might change slightly before lecture

Winterthur, 7. April. 2020

Bayesian Deep Learning

Outline: Bayesian Deep Learning

- Issues with current DL approach
  - No extrapolating / no epistemic uncertainty

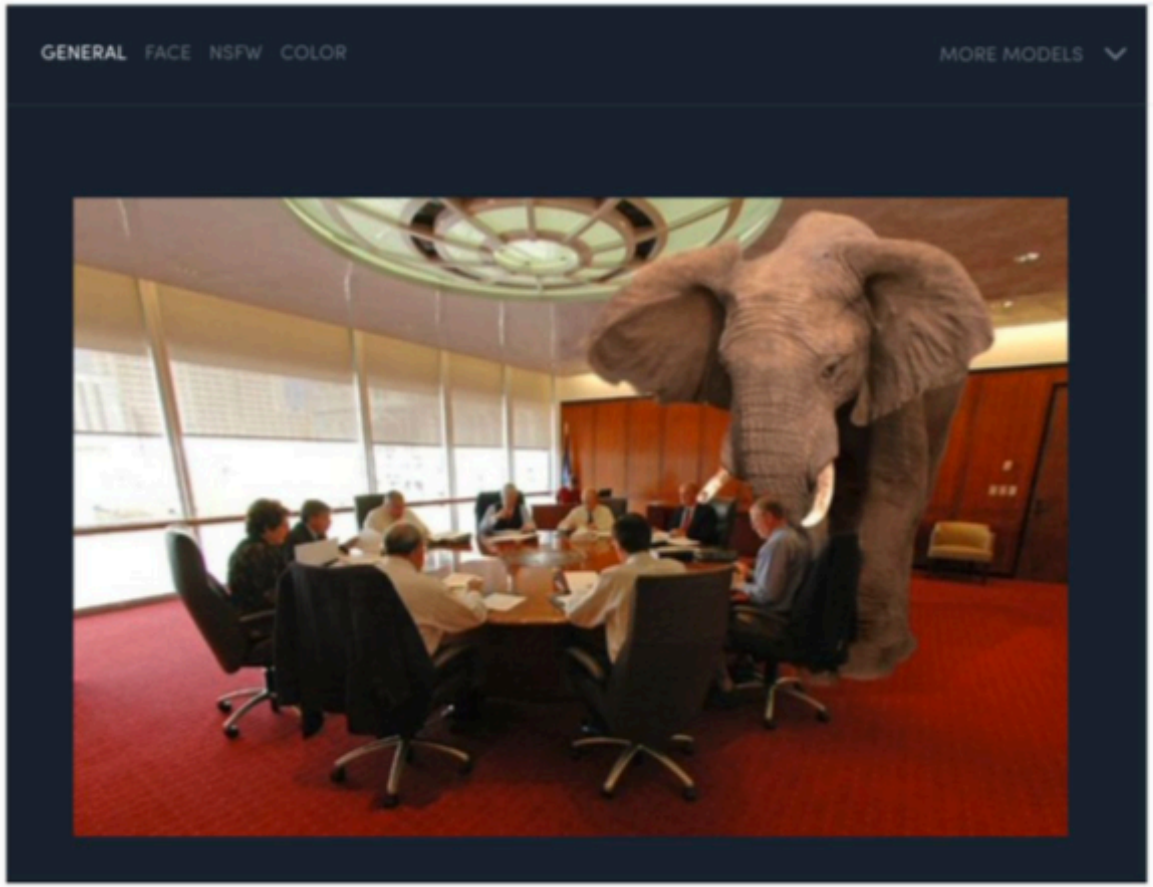
Last week

- Bayesian Statistics
  - A simple example (Bayes the Hacker's way)
  - Introduction to Bayesian Statistics

- Bayesian Neural Networks
  - Variational Approximation
  - MC-Dropout

The elephant in the room

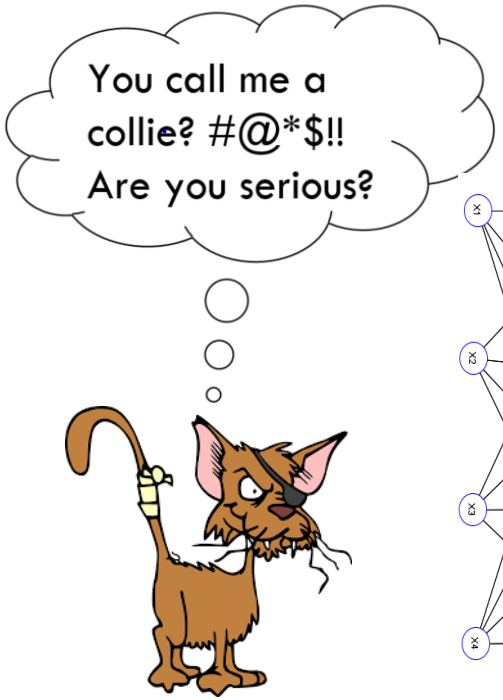
Neuronales Netz erkennt keinen Elefanten!

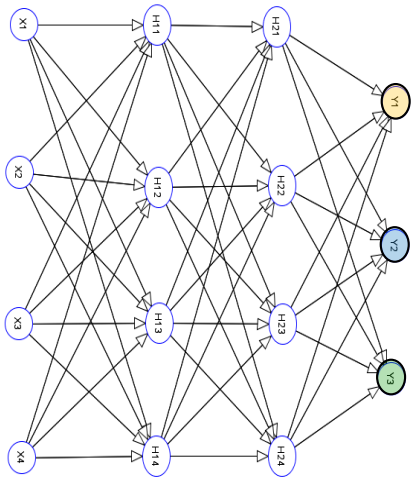


General	VIEW DOCS
PREDICTED CONCEPT	PROBABILITY
group	0.979
adult	0.977
people	0.976
furniture	0.960
room	0.957
business	0.903
indoors	0.901
man	0.896
seat	0.895

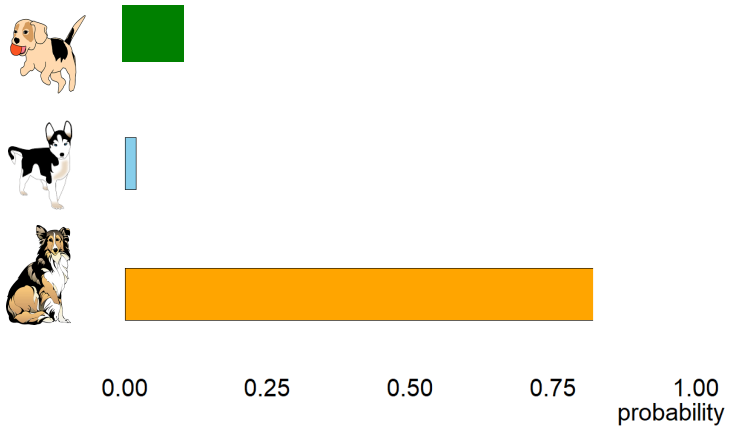
[NeurIPS 2018 Workshop on Causal Learning]

What happens if a novel class is presented to the CNN?





Plain wrong !



The reported probability is only valid if, we have the P(Train)=P(Test).
That's not the case. "The big lie deep learning is based on".
This is more evident, if we have a look at regression problems.

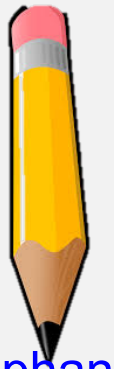
- ERROR Bars

Importance to detect OOD

- 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

Elephant in the room



Aufgabe:

https://github.com/tensorchiefs/dl_course_2020/blob/master/notebooks/18_elephant_in_the_room.ipynb

Extrapolation

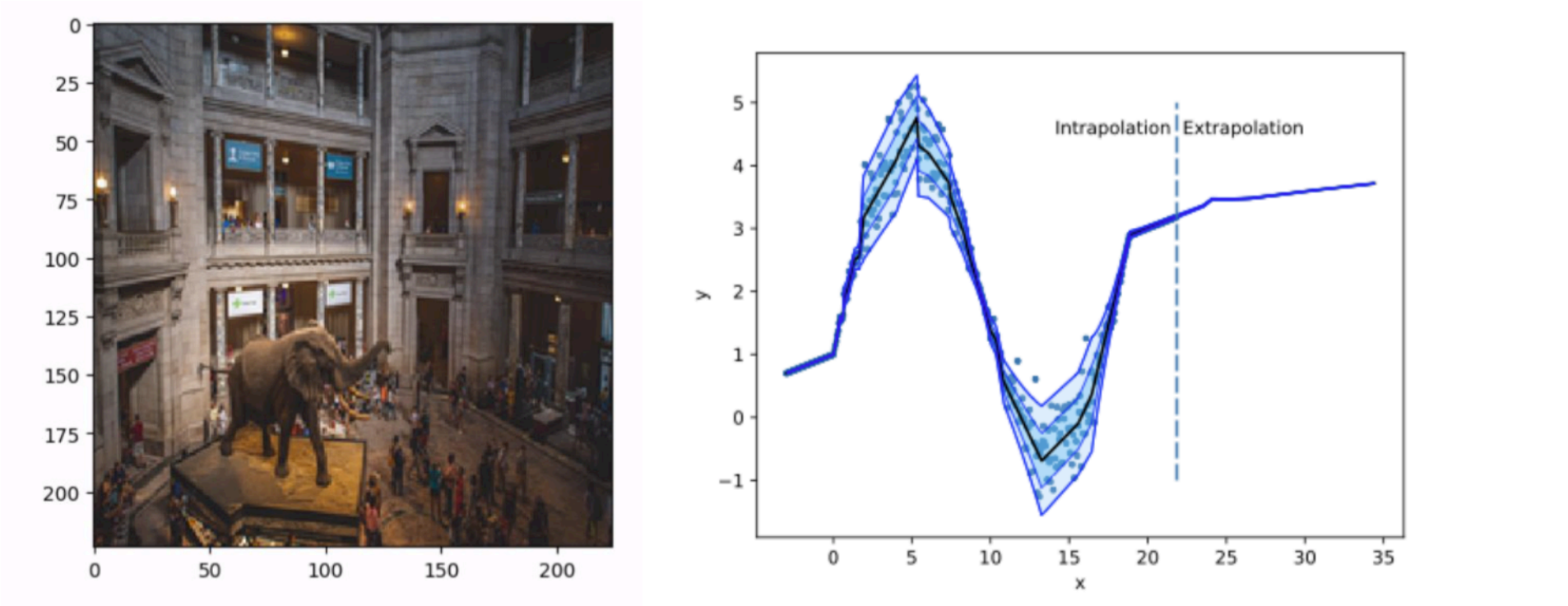
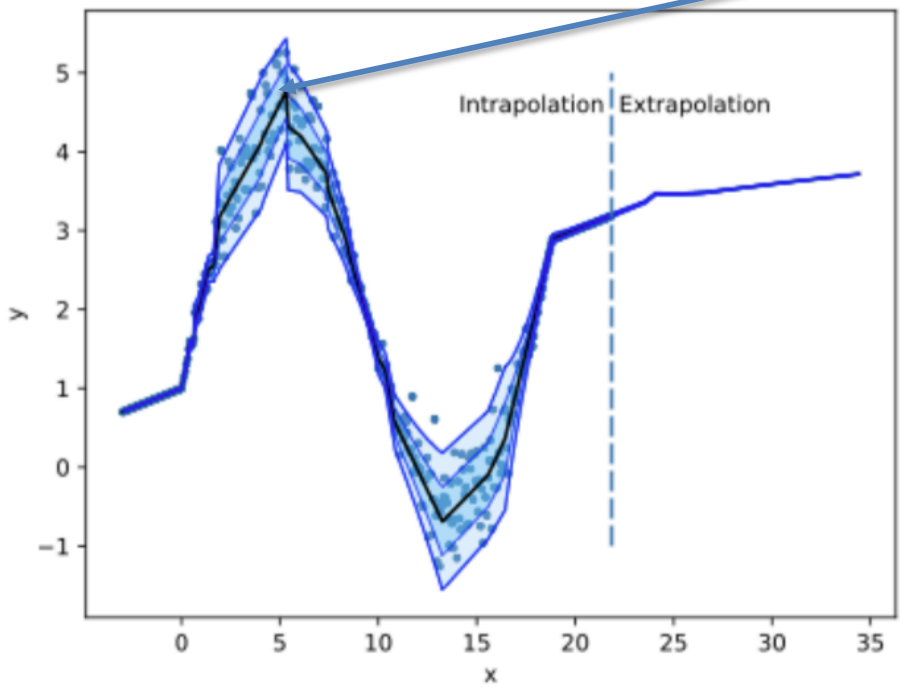


Figure 7.2 Bad case of DL. The high performant VGG16-CNN trained on ImageNet data fails to see the elephant in the room. The five highest-ranked class predictions of the objects in the image are horse_cart, shopping_cart, palace, streetcar, gondola; the elephant is not found! This image is an extrapolation 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”))

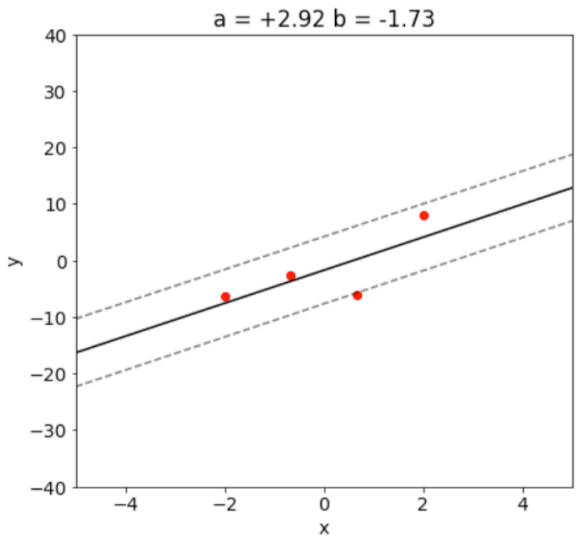


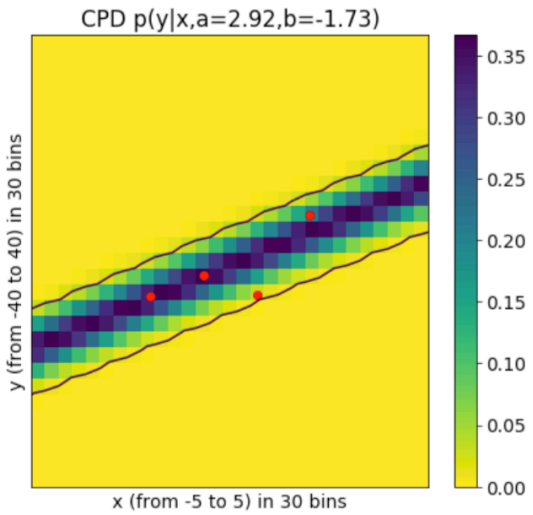
- *Aleatoric* uncertainty is due to the uncertainty in the data.
- The uncertainty when leaving the ‘known ground’ is called *epistemic* uncertainty.

We can model this uncertainty when we take the uncertainty with which we know the weights (called parameter uncertainty) into account. This can be done with Bayesian reasoning.

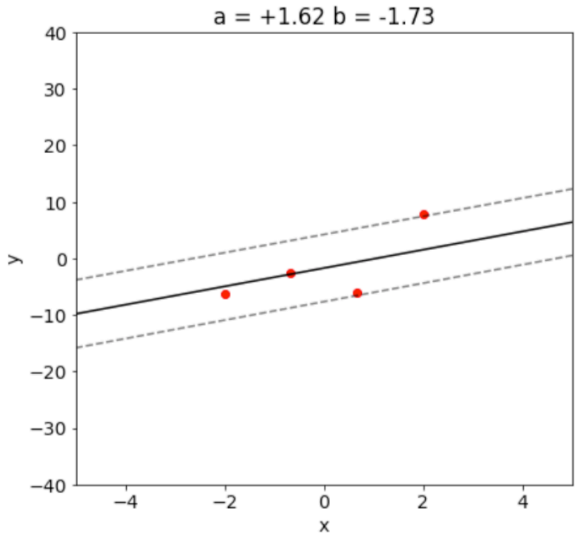
Bayes the hackers' way

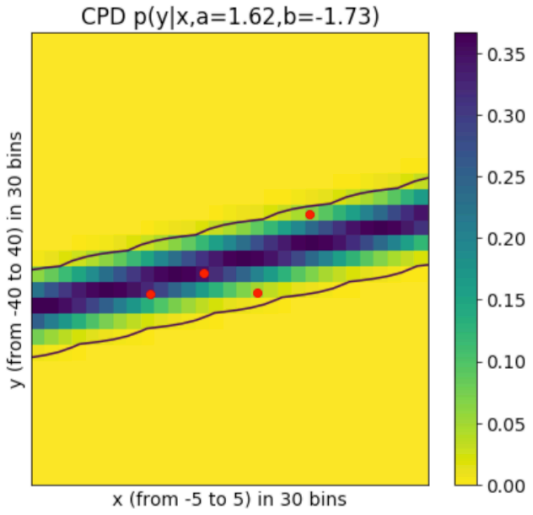
Let's look at good old linear regression to understand the gist of the Bayes idea. Assume $\sigma = 3$ to be known.





MaxLike Solution

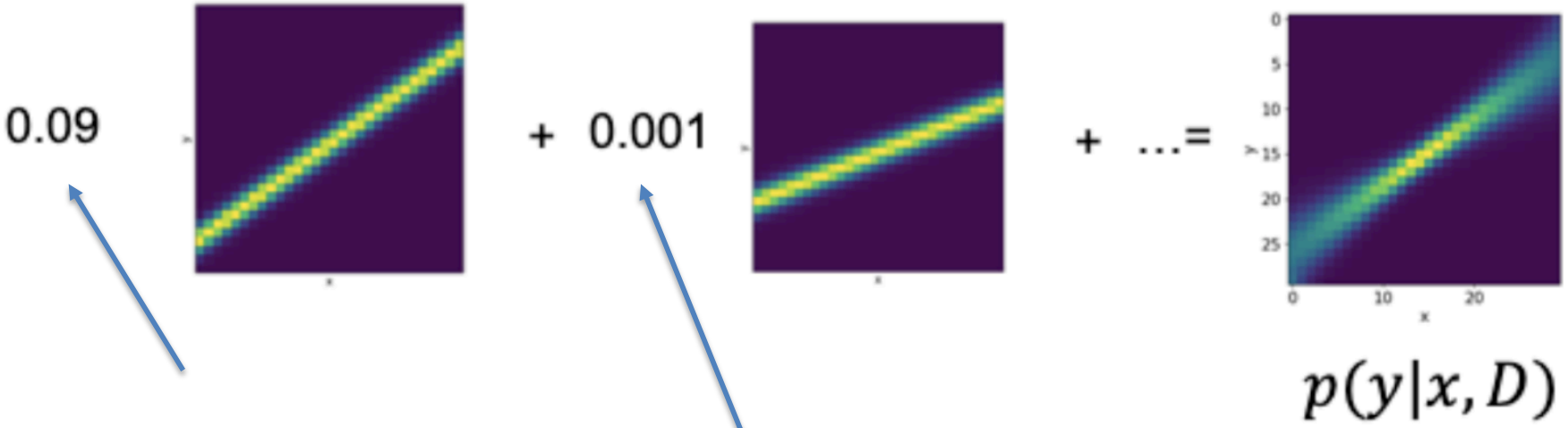




A bit off the MaxLike Solution

Combining different fits

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



Hightest weight for MaxLike

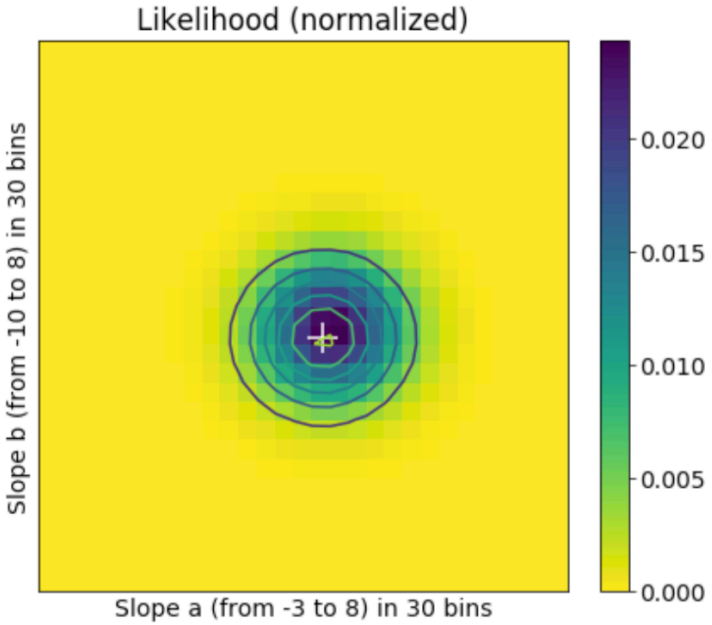
Other other less like solutions, should have a vote.

Question: How to get the weight?

Idea: use the (normalized) likelihood! $p_{norm}((a,b)|D)$ D is training data

Don't put all egg's in one Basket

- Also take other solutions for a,b into account



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

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

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

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

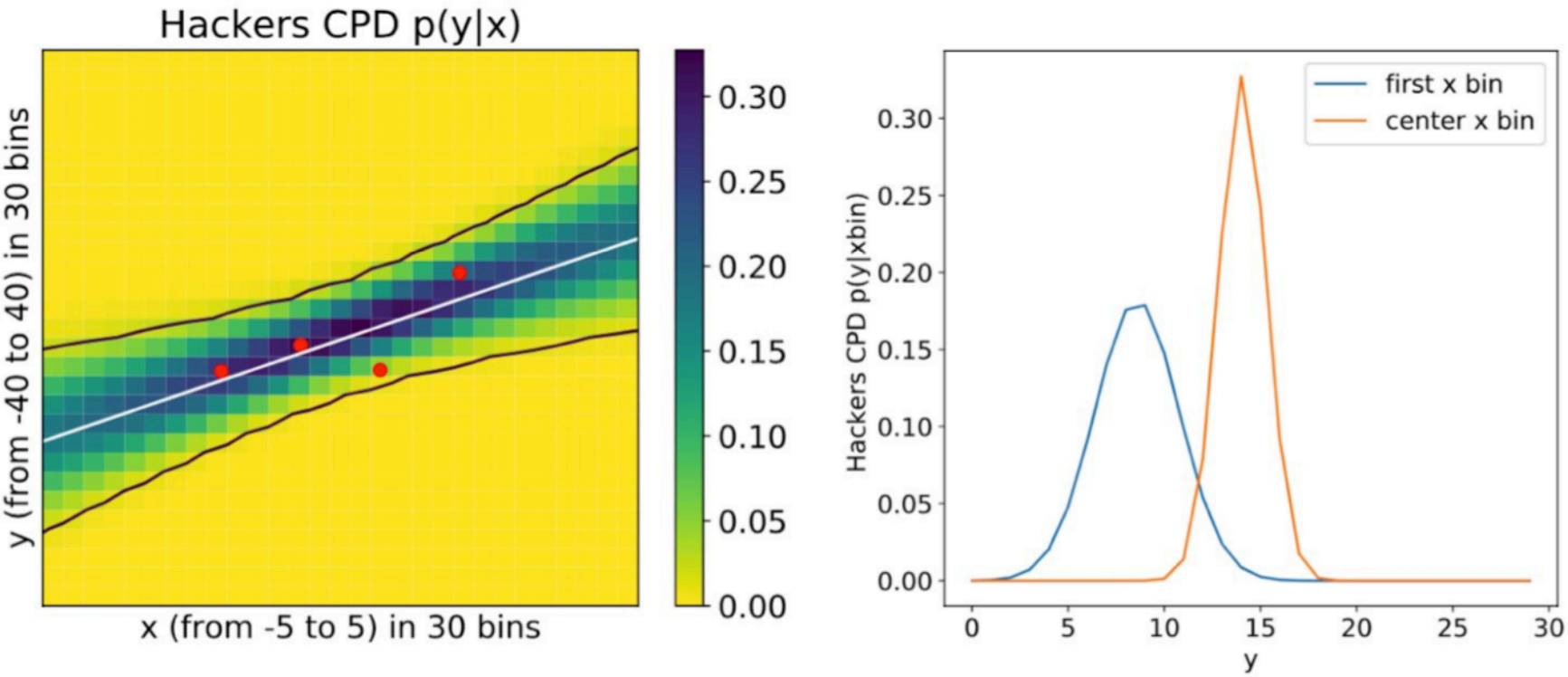


Figure 7.6 The predictive distribution for the Bayesian linear regression model, trained with the four data points shown on the left side by the color-code and on the right as conditional distribution at two different *x* positions. You can clearly see that the uncertainty gets larger when leaving the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

What have we done? (Comparison with MaxLike)

- Bayes (averaging over many)

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

  - For normalization, we calculated likelihood

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

  - at nbins^2 positions (here 900)

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

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

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

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

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

- We choose
  - $p_{norm}(D|w) = \frac{p(D|w)}{\sum_w p(D|w)}$
  - 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

The Bayesian Mantra (say it loud)

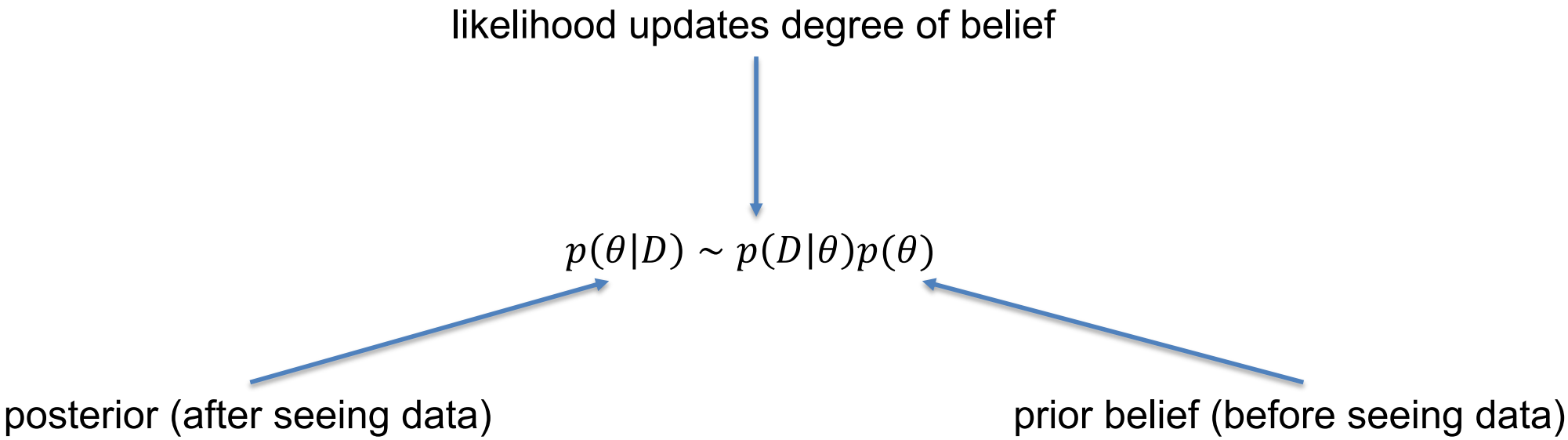
$$\boldsymbol{p(\theta|D) \sim p(D|\theta)p(\theta)}$$

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

Interpretation updating the degree of belief

- Parameters θ are random variables, with distribution
  - We interpret the distribution as our degree of belief in a certain value

- The Bayes formula is seen as an update of our belief in the light of data



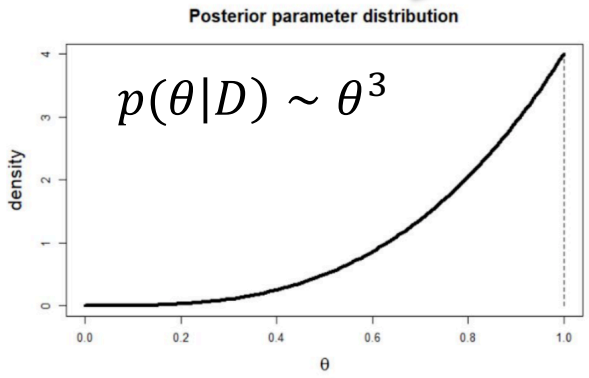
Example Coin Toss

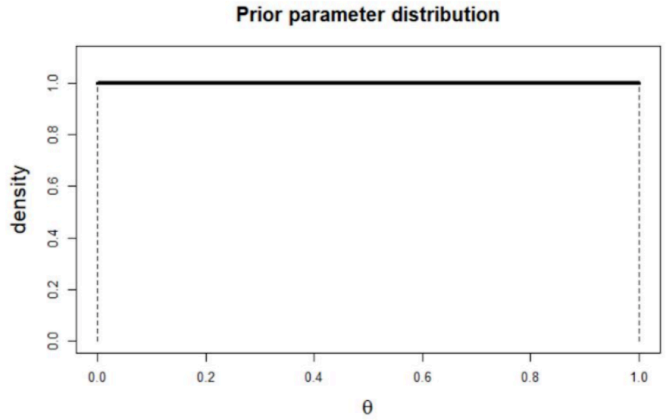
- θ parameter for the Bernoulli-distribution (probability of head)
- Before the experiment we assume all value of θ are equally likely $p(\theta) = const$
- Do the experiment, observe 3 times head D='3 heads'
- Calculate likelihood $p(D|\theta) = p(y = 1) \cdot p(y = 1) \cdot p(y = 1) = \theta \cdot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

After **Before**

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

θ^3





$p(\theta|D) = 4 \cdot \theta^3$ (normalized so that it integrates to 1)

Posterior Predictive Distribution

From the Hacker's experiment

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

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

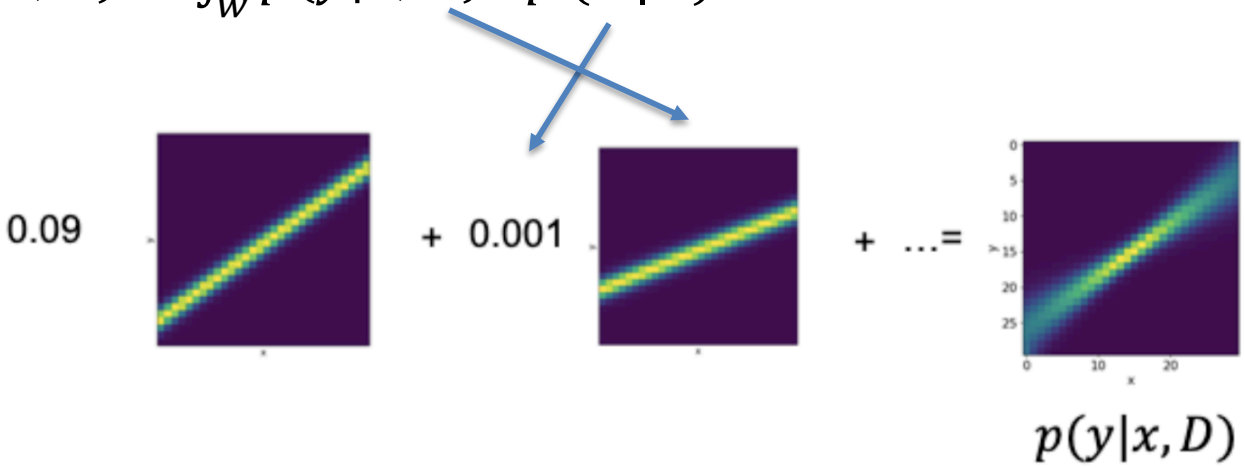
For coin interested in $\mathrm{p}(y=1|D)$ prob. for head

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

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

Summary

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

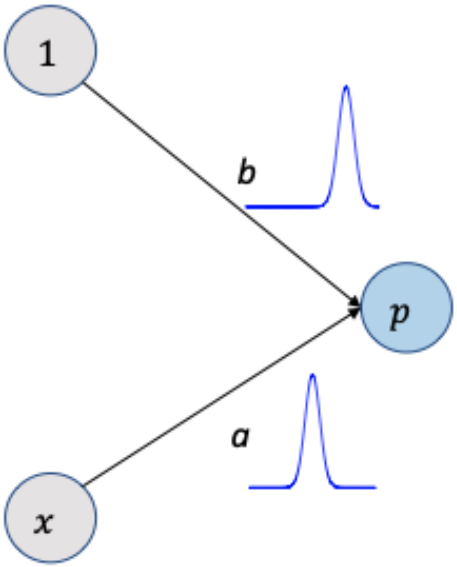


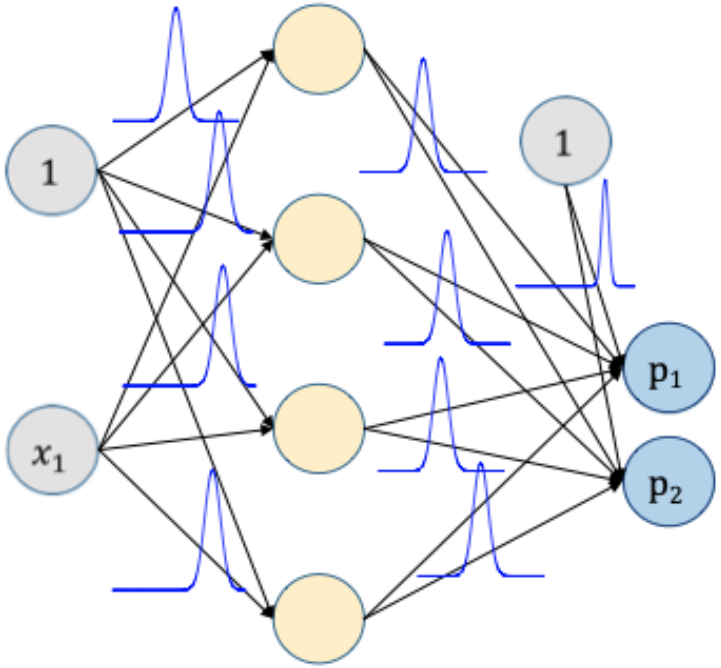
- Not just a single solution
  - "Marginalizing instead of optimization"
- Bayes usually achieves
  - Better uncertainty estimates
  - Better predictive performance

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 get confused Bayesian Network is something completely different

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 $p(a|D)$ with an approximation e.g. $N(\mu_a, \sigma_b)$ and determine μ_a,σ_a
- MC-Dropout
  - Dropout also during predictions (magically) samples from a posteror

Variational Inference

The principle of VI

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

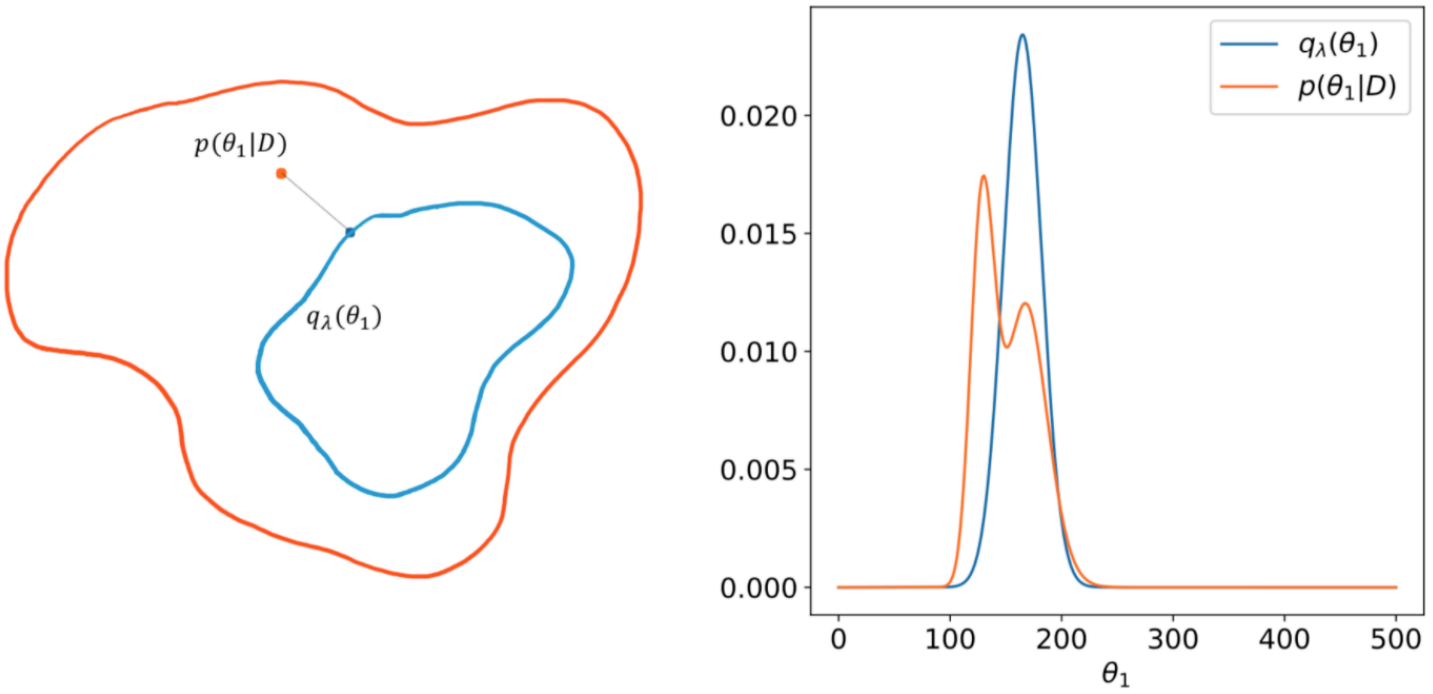


Figure 8.3 The principle idea of variational inference (VI). The larger region on the left depicts the space of all possible distributions, and the dot in the upper left represents the posterior $p(\theta_1|D)$ (corresponding to the dotted density on the right panel). The inner region depicts the space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

Distance between two distributions

- To get $q_\lambda(\theta)$ close to $p(\theta|D)$ we need a distance
- Typical "Distance" is KL-Divergence
- "Distance" between two distributions $f(x)$ and $g(x)$

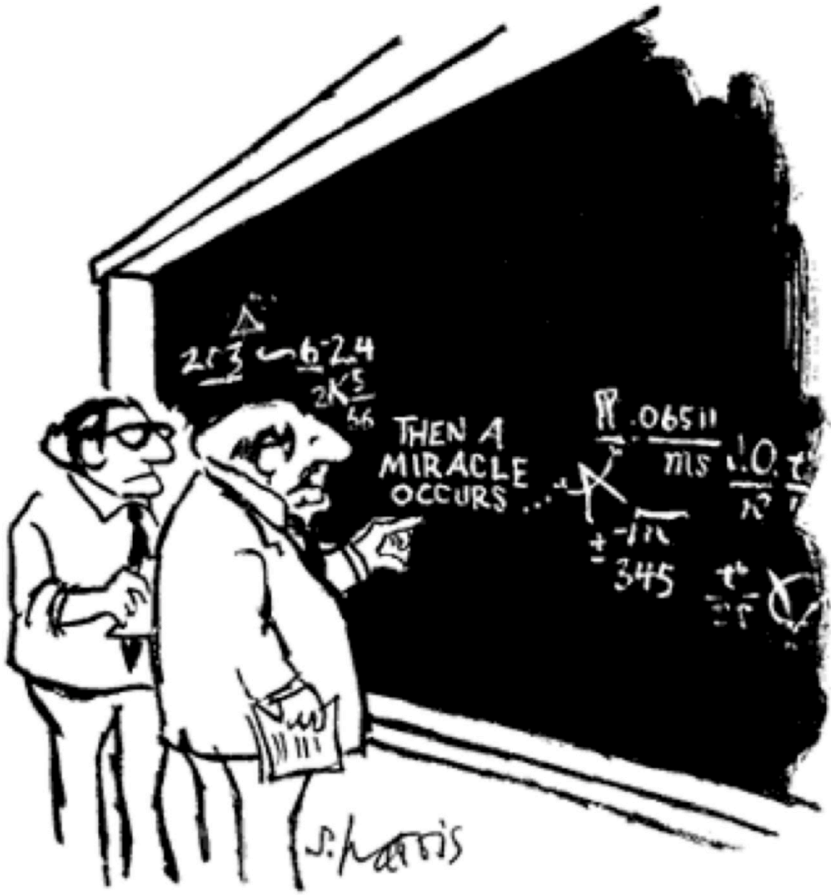
$$KL(f(x)||g(x)) = \int \log\left(\frac{f(x)}{g(x)}\right) f(x)dx = E_{x\sim f(x)}[\log\left(\frac{f(x)}{g(x)}\right)]$$

- Properties of KL-Divergence
  - KL ≥ 0
  - KL = 0 if $f(x) = g(x)$
  - $KL(f(x)||g(x)) \neq KL(g(x)||f(x))$ Not symmetrical not a real distance

Calculating distance to an unknown function

- We need to calculate KL-Divergence between
  - $p(\theta|D)$ the **unknown posterior**
  - $q_\lambda(\theta)$ and the variational approximation

- Is this possible?



"I think you should be more explicit here in step two."

Be more explicit about step two

$$KL[q_\lambda(\theta)\|p(\theta|D)] = \int q_\lambda(\theta)\, log\, \frac{q_\lambda(\theta)}{p(\theta|D)} d\theta$$

We have to start with way, q first

$$p(\theta|D) = p(\theta, D)/p(D)$$

$$KL[q_\lambda(\theta)\|p(\theta|D)] = \int q_\lambda(\theta)\, log\, \frac{q_\lambda(\theta)}{p(\theta, D)/p(D)} d\theta$$

$$log(A \cdot B) = log(A) + log(B)$$

$$log(B/A) = -log(A/B)$$

$$KL[q_\lambda(\theta)\|p(\theta|D)] = \int q_\lambda(\theta)\, log\, p(D)\, d\theta - \int q_\lambda(\theta)\, log \frac{p(\theta, D)}{q_\lambda(\theta)} d\theta$$

no dependence on θ and $\int q_\lambda(\theta) d\theta = 1$

$$KL[q_\lambda(\theta)\|p(\theta|D)] = log\, p(D) - \int q_\lambda(\theta)\, log\, \frac{p(\theta, D)}{q_\lambda(\theta)} d\theta$$

We need to minimize

Be more explicit about step two (cont'd)

$$\lambda^* = argmin\{- \int q_\lambda(\theta)\, log\, \frac{p(\theta,D)}{q_\lambda(\theta)} d\theta\}$$

$$p(\theta,D) = p(D|\theta) \cdot p(\theta)$$

$$\lambda^* = argmin\{- \int q_\lambda(\theta)\, log\, \frac{p(D|\theta)\cdot p(\theta)}{q_\lambda(\theta)} d\theta\}$$

$$\lambda^* = argmin\{\int q_\lambda(\theta)\, log\, \frac{q_\lambda(\theta)}{p(\theta)} d\theta - \int q_\lambda(\theta) \cdot log\, p(D|\theta)\, d\theta\}$$

$$\lambda^* = argmin\{KL[q_\lambda(\theta)\|p(\theta)] - E_{\theta\sim q_\lambda}[log(p(D|\theta)]\}$$

A miracle the unknown posterior $p(\theta|D)$ is gone.

Intuition of the optimization

- Distance of prior to variational approximation (regularization)

$$\lambda^* = argmin\{KL[q_\lambda(\theta)||p(\theta)] - E_{\theta\sim q_\lambda}[log(p(D|\theta)]\}$$

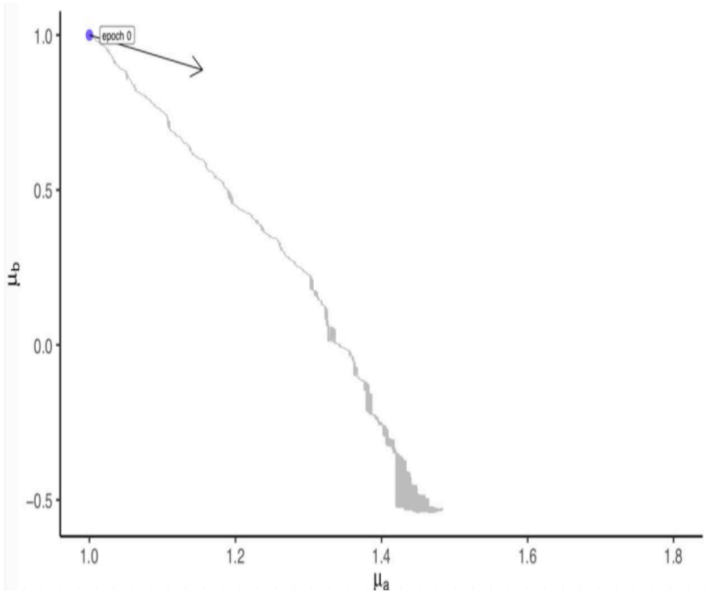
- NLL of trainings data D, now averaged over different weights

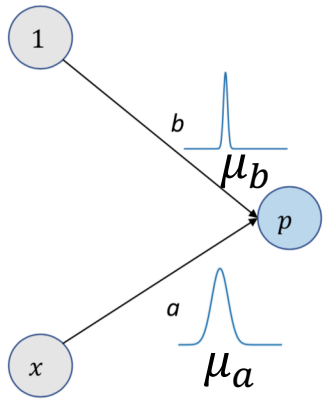
Tradeoff of good fit (low NLL) and regularization small KL to prior

Fitting the variational approximation

$$\lambda^* = argmin\{KL[q_\lambda(\theta)||p(\theta)] - E_{\theta\sim q_\lambda}[log(p(D|\theta)]\}$$

- Same tricks as in the VAE
  - KL Divergence can usually be calculated analytically
  - Instead of calculating $E_{\theta\sim q\_\lambda}$ for many sample use one (unbiased estimate)
  - Reparameterization trick to backprop through sampling

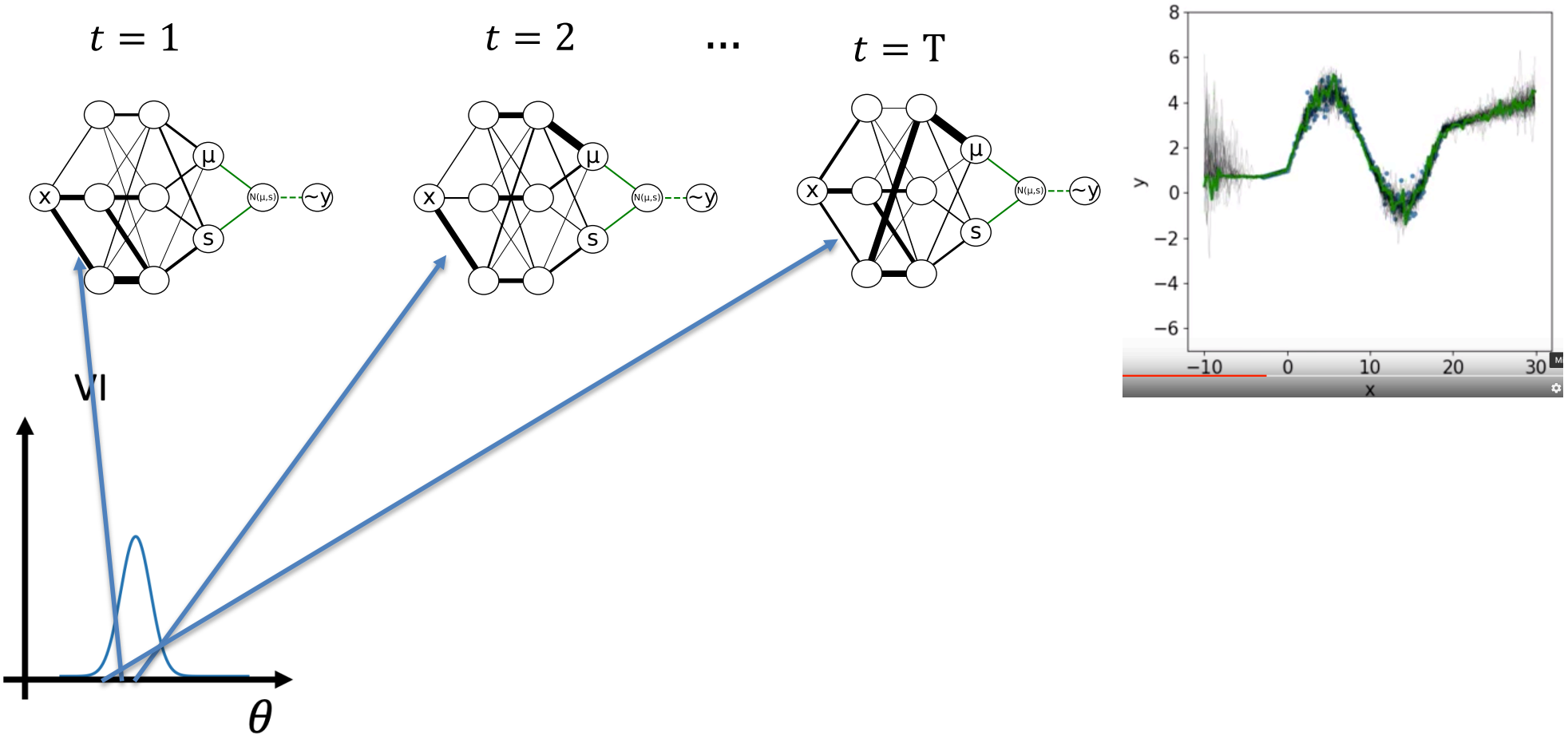




https://www.youtube.com/watch?v=MC_5Ne3Dj6g&feature=youtu.be

For practical reasons there are Keras layers DenseReparametrization

Sampling from posteriori predictive distribution



https://www.youtube.com/watch?v=mQrUcUoT2k4&feature=youtu.be

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
_________________________________________________________________
```

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)
```