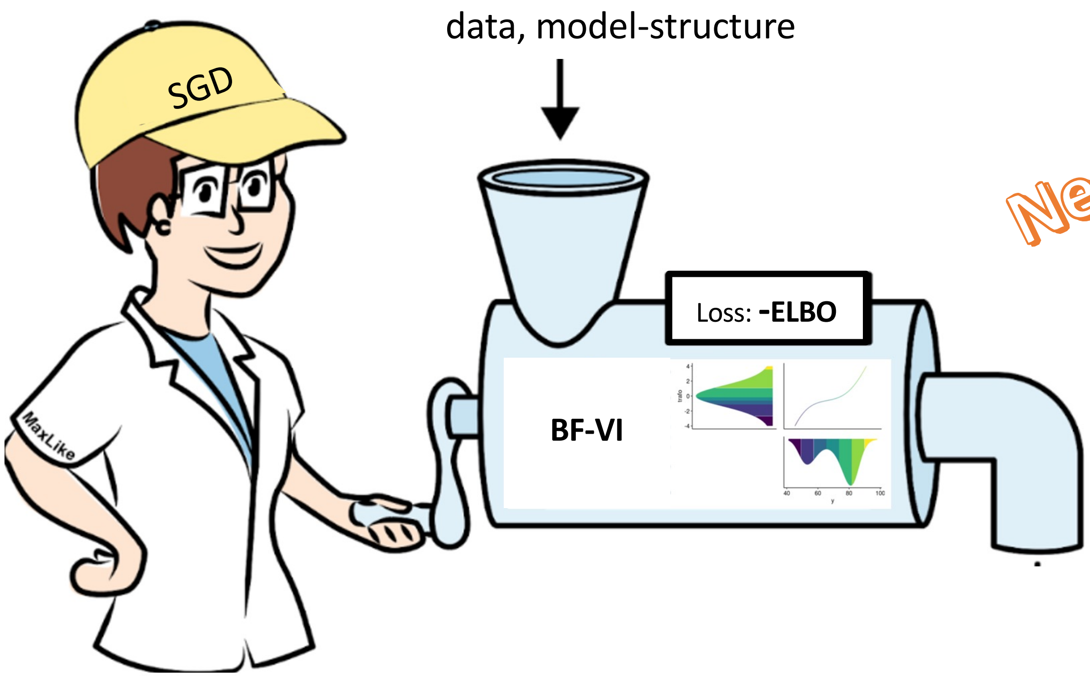
Bayes for dummies 2

Transformation Models for Flexible Posteriors in Variational Bayes

New Works for multivariate posteriors!



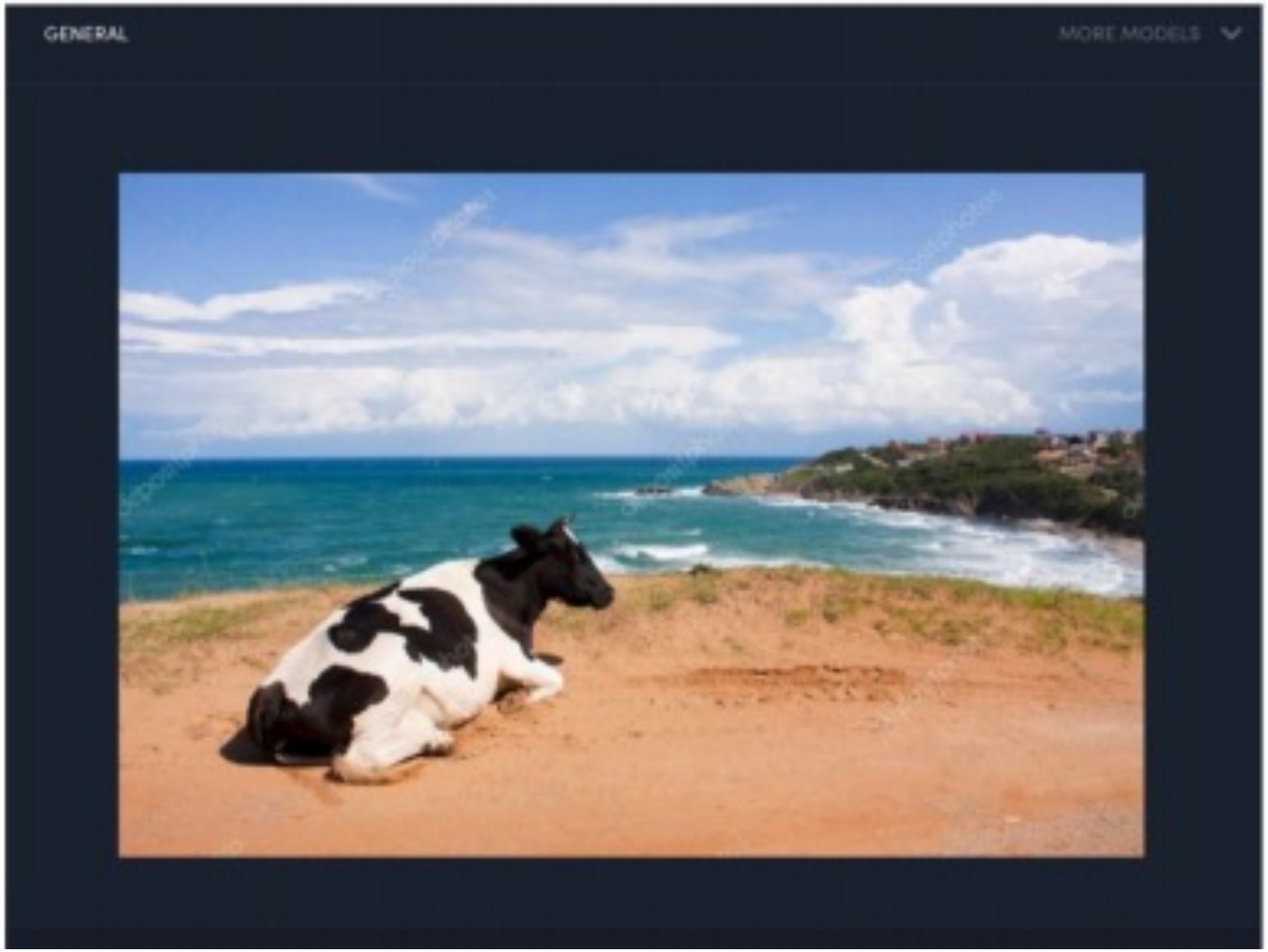
Goal:
Get a fitted Bayesian model with flexible posteriors for the model parameters

Recent work:

- arXiv:2106.00528 (1-D and Mean Field Version)
- https://opus.htwg-konstanz.de/frontdoor/index/index/docId/2974 (Semi Structured)
- Manuscript in preparation (arXiv next few days)

Oliver Dürr, Stefan Hörtling, Ivonne Kovylov, Daniel Dold, Beate Sick

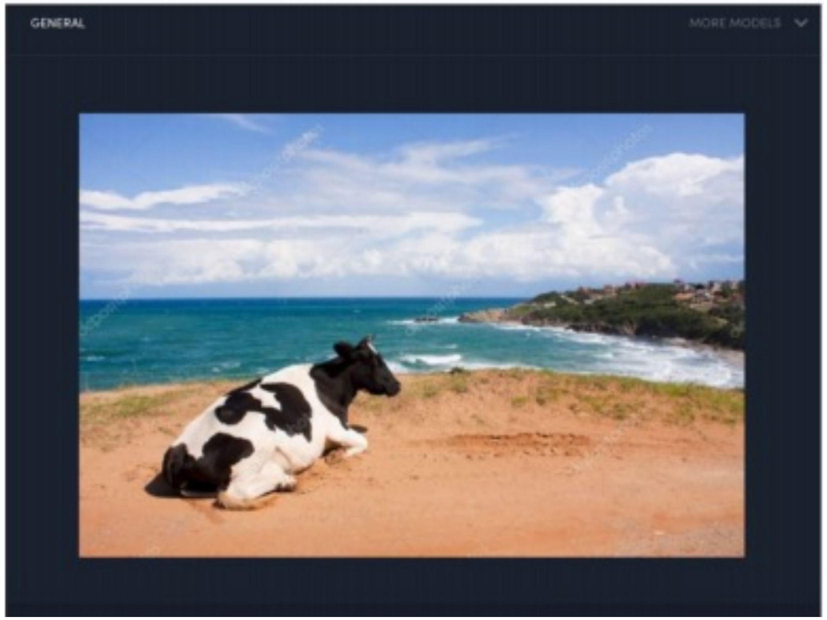
Motivation: OOD Uncertainty / Extrapolation

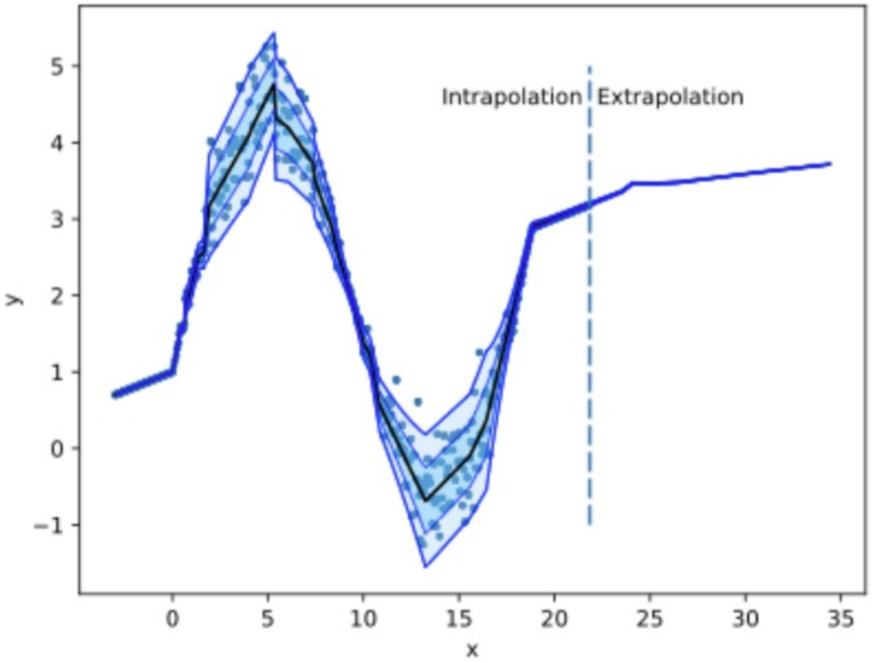


Does DL see the Cow?

Image Credit NeurIPS 2018 Workshop on Causal Learning

Motivation: OOD Uncertainty / Extrapolation





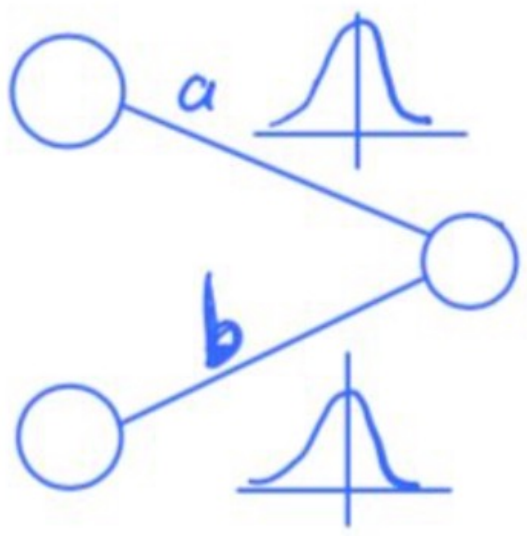
General	VIEW DOCS
no person	0.991
beach	0.990
water	0.985
sand	0.981

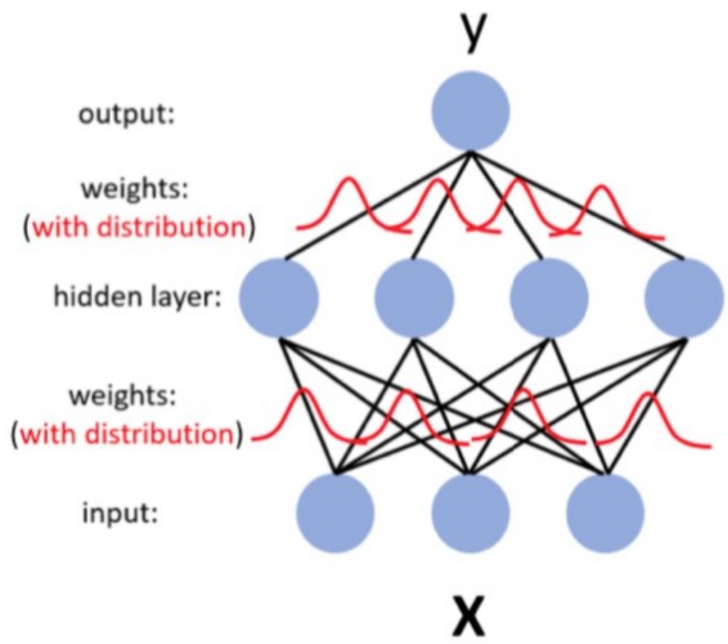
DL does not even say “I don’t know”

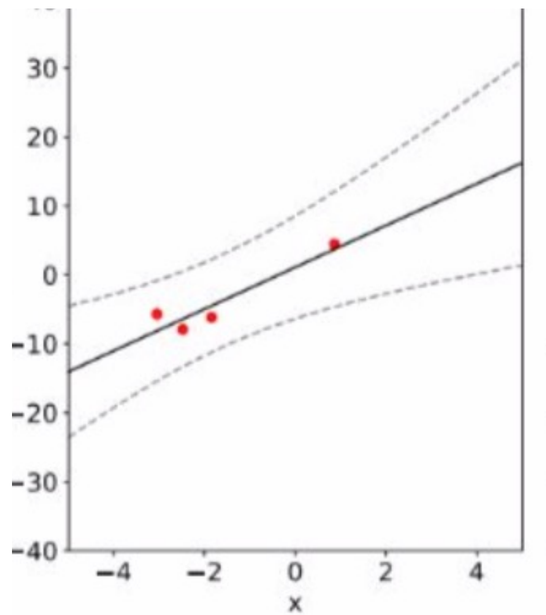
Deep **Learning does not state uncertainty** in OOD Situations
Bayesian Deep Learning allows to model (epistemic) uncertainty

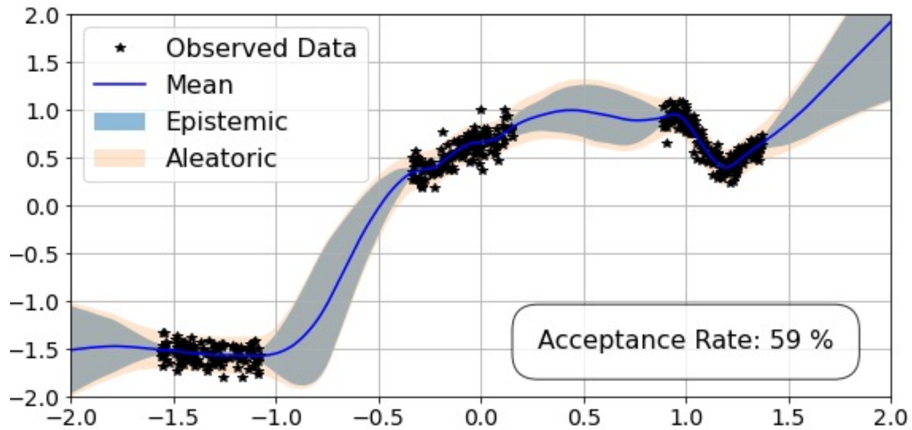
Image Credit NeurIPS 2018 Workshop on Causal Learning

Bayesian Neural Networks to the rescue





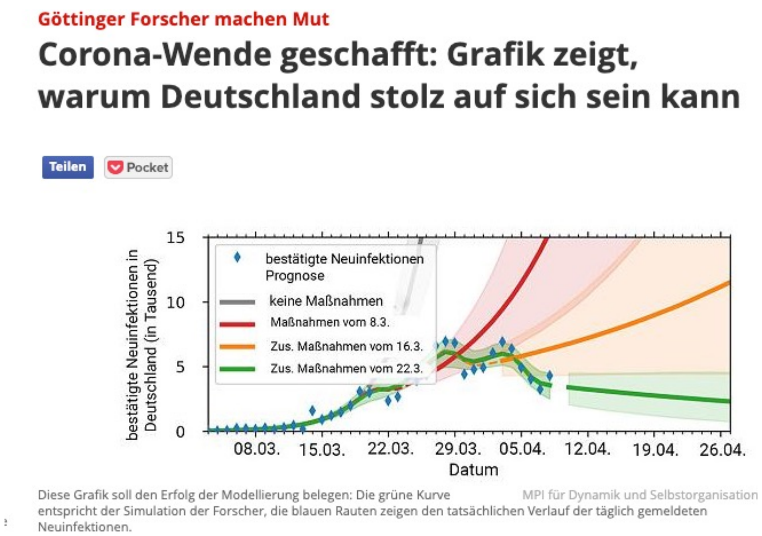


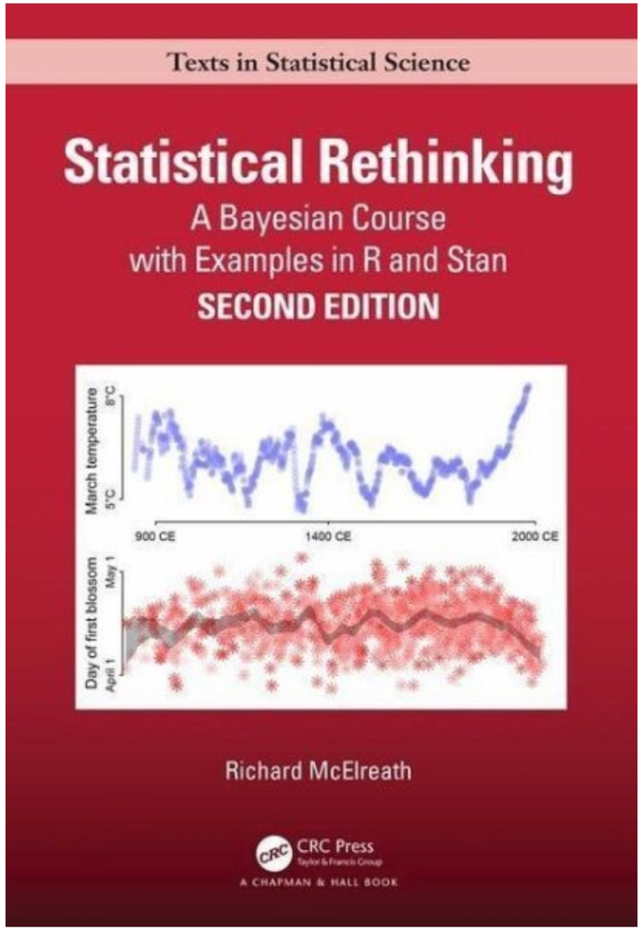


https://adamcobb.github.io/journal/bnn.html

Bayesian models (Besides Bayesian NN)

- Include prior knowledge
- Example Corona Modeling
  - https://arxiv.org/abs/2004.01105 (Science) Priseman Group





Bayesian Model Definition

likelihood

posterior

pior

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)} = \frac{p(D|w)p(w)}{\int p(D|w)p(w)dw} \sim p(D|w)p(w)$$

normalizing denominator

this is the most difficult part!

$p(D|w)$ #D=(x,y), T is Batch

```
get_LL <- function(w, x, y, T) {
  #Determination of the likelihood
  w0 = tf$slice(w,c(0L,0L),c(T,1L))  #First is intercept
  wx = tf$slice(w,c(0L,1L),c(T,P))   #P Slopes
  ws = tf$slice(w,c(0L,P+1L),c(T,1L))   #for SD
  mu = tf$matmul(wx, tf$transpose(x)) + w0 #T,X
  sigma = tf$math$softplus(ws)
  y_rep = tf$transpose(tf$tile(y, c(1L,T)))
  return (tfp$distributions$Normal(loc=mu, scale=sigma)$log_prob(y_rep))
}
```

$p(w)$

```
get_log_prior = function(w){
  w1 = tf$slice(w,c(0L,0L),c(T,1L))
  w2 = tf$slice(w,c(0L,1L),c(T,1L))
  w3 = tf$slice(w,c(0L,2L),c(T,1L))
  sigma = tf$math$softplus(w3)
  return(
    tfd_normal(0,10.)$log_prob(w1) +
      tfd_normal(0,10.)$log_prob(w2) +
      tfd_log_normal(0., 1.)$log_prob(sigma)
  )
```

Many probabilistic programming languages to define model: Stan, pyro, numpyro, TF-probability,…

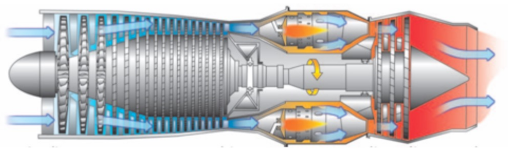
Model Definition, examples

Tell the story, how the data is generated.

Stan model

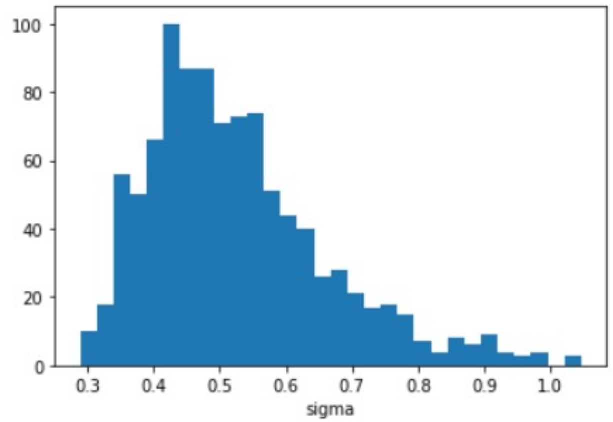
```
data {
  int<lower=0> N;
  int<lower=1> P;
  vector[N] y;
  matrix[N,P] x;
}

parameters {
  vector[P] w;
  real b;
  real<lower=0> sigma;
}

model {
  y ~ normal(x * w + b, sigma);
  b ~ normal(0,10);
  w ~ normal(0, 10);
  sigma ~ lognormal(0.5,1);
}
```

Engines takes Model



and produces Samples from posterior

Samples from posterior for sigma

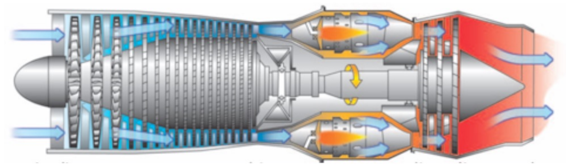


Computing the posterior

- Analytical
  - Impossible for interesting problems

No Blackbox
(i.e. does not work with code)

- MCMC-Sampling (gold standard)
  - Draw samples from the posterior
  - Limitations
    - Larger number of Data Points (no mini-batch) $p(\mathrm{w} \mid D)$ has the “Daten an der Backe”
    - Large models sizes (Deep Learning is out of scope)



- Approximations
  - Laplace
  - Deep Learning Hacks*
    - MC-Dropout
    - (Multi-)SWAG
    - Ensembling
  - Variational Inference ← Focus here
    - There is an automatic version, Black Box Variation Inference

*These are non-Bayesian but still works for the output “function space”

Background: Variational Inference

The principle of VI

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

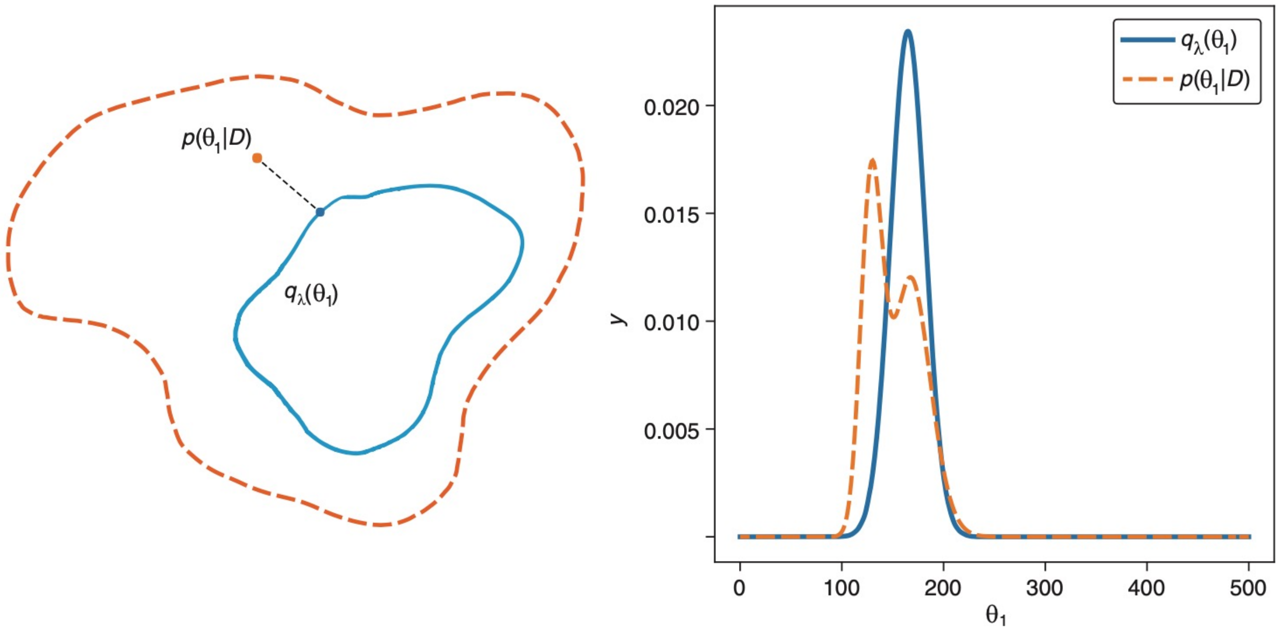


Figure 8.3 The principle idea of variational inference (VI). The larger region on the left depicts the space of all possible distributions, and the dot in the upper left represents the posterior $p(\theta_1|D)$, corresponding to the dotted density in the right panel. In the left panel, the inner region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

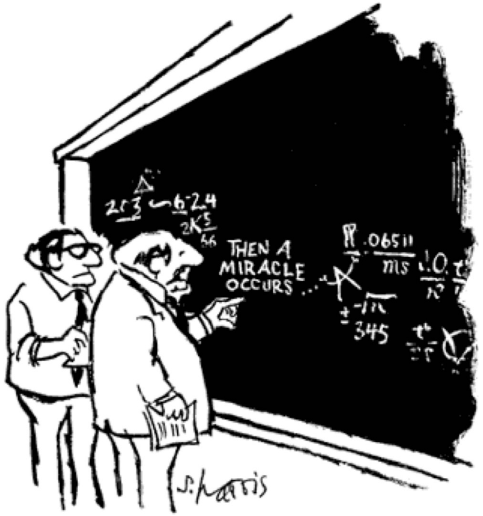
Loss in VI: Minimizing KL-Divergence

- We want $q_\lambda(\theta)$ close to the ***unkown*** $p(\theta|D)$
- We start with KL(q||p) (direction chosen to make life easy)

$$\mathrm{KL}(q_\lambda(\theta) \parallel p(\theta|D)) = \int q_\lambda(\theta) \log\left(\frac{q_\lambda(\theta)}{p(\theta|D)}\right) d\theta$$

$$= \log(p(D)) - \underbrace{(\mathbb{E}_{\theta\sim q_\lambda}(\log(p(D|\theta))) - \mathrm{KL}(q_\lambda(\theta) \parallel p(\theta)))}_{\mathrm{ELBO}(\lambda)}$$

- A bit of math does the magic
- ELBO Evidence Lower Bound is maximized (minimize –ELBO)



"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 $\int q_\lambda(\theta) d\theta = 1$

$$KL[q_\lambda(\theta)||p(\theta|D)] = log\ p(D) - \underbrace{\int q_\lambda(\theta)\ log\ \frac{p(\theta,D)}{q_\lambda(\theta)} d\theta}_{\text{We need to minimize}}$$

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.

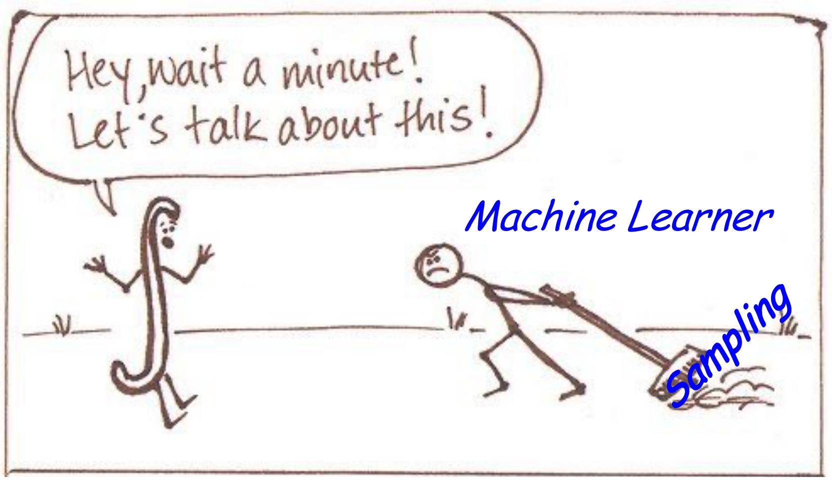
* The KL is ∝ Number of weights
* The NLL terms is ∝ number of datapoints
The more Data the less important the priors.

Fitting the variational approximation

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

$$KL[q_\lambda(\theta)||p(\theta)] = \int q_\lambda(\theta)\log\left(\frac{q_\lambda(\theta)}{p(\theta)}\right)d\theta = E_{\theta\sim q_\lambda}[\log(\frac{q_\lambda(\theta)}{p(\theta)})]$$

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



$$E_{\theta\sim q_\lambda}[f(\theta)] \approx \frac{1}{S}\sum_{\theta_s\sim q_\lambda} f(\theta_s)$$

- Same tricks as in the VAE
  - KL Divergence can often/sometime be calculated analytically
  - Instead of calculating $E_{\theta\sim q\_\lambda}$ for many sample use one (unbiased estimate)

Black Box VI (https://arxiv.org/abs/1401.0118)

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

$$KL = E_{\theta \sim q_\lambda}[\log(\frac{q_\lambda(\theta)}{p(\theta)})]$$

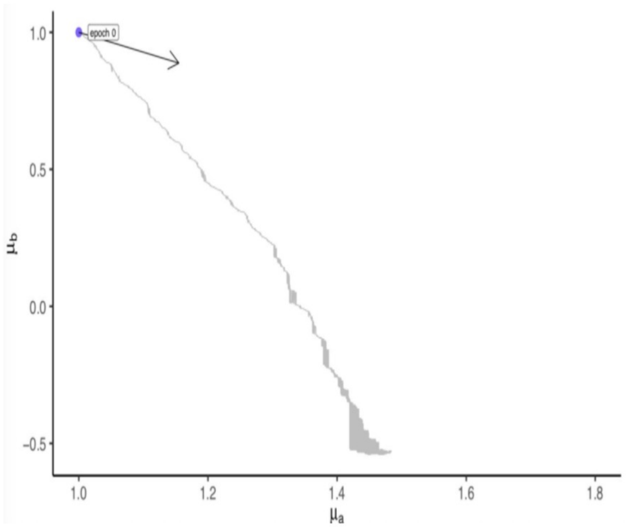
NLL

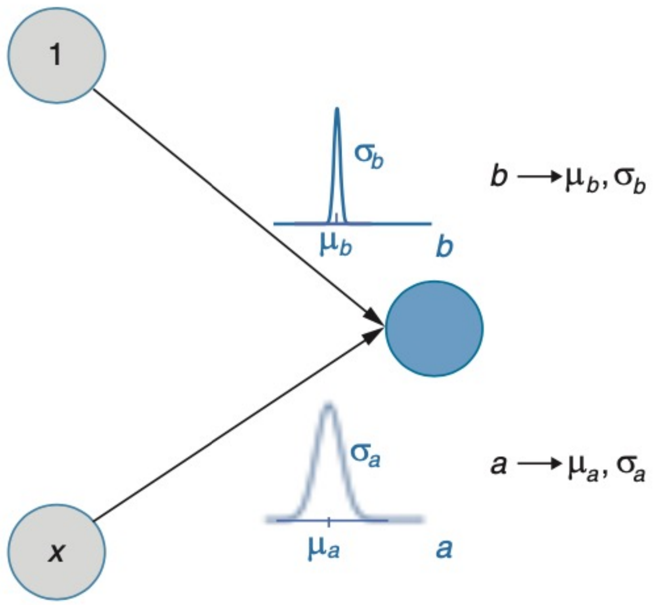
```
def nelbo(λ):
    θ_i <- Samples i = 1,...S from variational distribution
    NLL <- mean(-log(p(D|θ_i))
    KL  <- mean(log(q_λ(θ_i)) - mean(log(p(θ_i))

#SGD (Adam and RMSProp… also possible)
λ ← λ − η · grad(nelbo) #Use autograd to calculate gradient
```

Just provide the functions p(D|θ_i), p(θ_i), and q($\lambda|\theta_i$) and autograd does the rest

Fitting the variational approximation

- VI at work





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

For practical reasons there are Keras layers DenseReparametrization

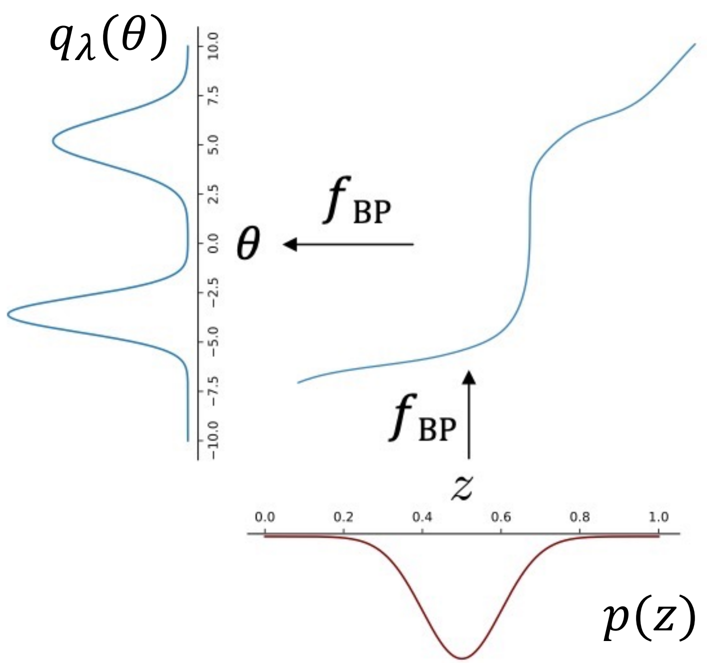
Bernstein VI

Current Limitations of VI

1. Better / other divergences
2. Optimization procedure
   - E.g. Less Noisy Gradient estimator
   - …
3. Flexible variational distributions
   - Mixtures (of Gaussian)
   - Deep Normalizing Flows
   - **Use of Transformation Models**

The idea of transformation models (TM)

The heart of a TM is a **parameterized bijective transformation function** $\mathrm{f}_{\mathrm{B}P}(\mathrm{z}) = h_\lambda(z)$ that transforms between a simple distribution $p(z)$ and a potentially complex distribution $q_\lambda(w)$



$\theta = f_{BP}(\mathrm{z})$

Be careful "change of variable" formula

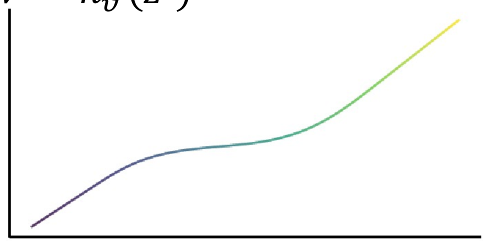
$$q_\lambda(\theta) = p(z) \cdot \left| \frac{\partial f_{BP}(z)}{\partial z} \right|^{-1}$$

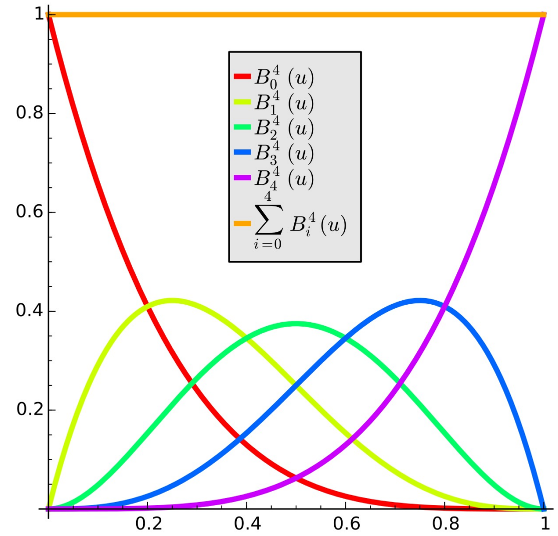
Using Bernstein-polynomial for $h_\lambda(z)$

$$w' = h_\vartheta\,(z') = \sum_{k=1}^{M} \frac{\vartheta_k}{M+1} \mathrm{B}_k^M\,(z')$$

$z' \in [0,1]$

$w' = h_\vartheta\,(z')$





A Bernstein polynomial has nice properties:

- It can approximate every function on the support $[0,1]$ (Bernstein 1906)
- Its flexibility can be controlled by the order M
- It is bijective, i.e. monotone increasing, if parameters $\vartheta_1 \leq \vartheta_2 \leq \cdots \leq \vartheta_M$

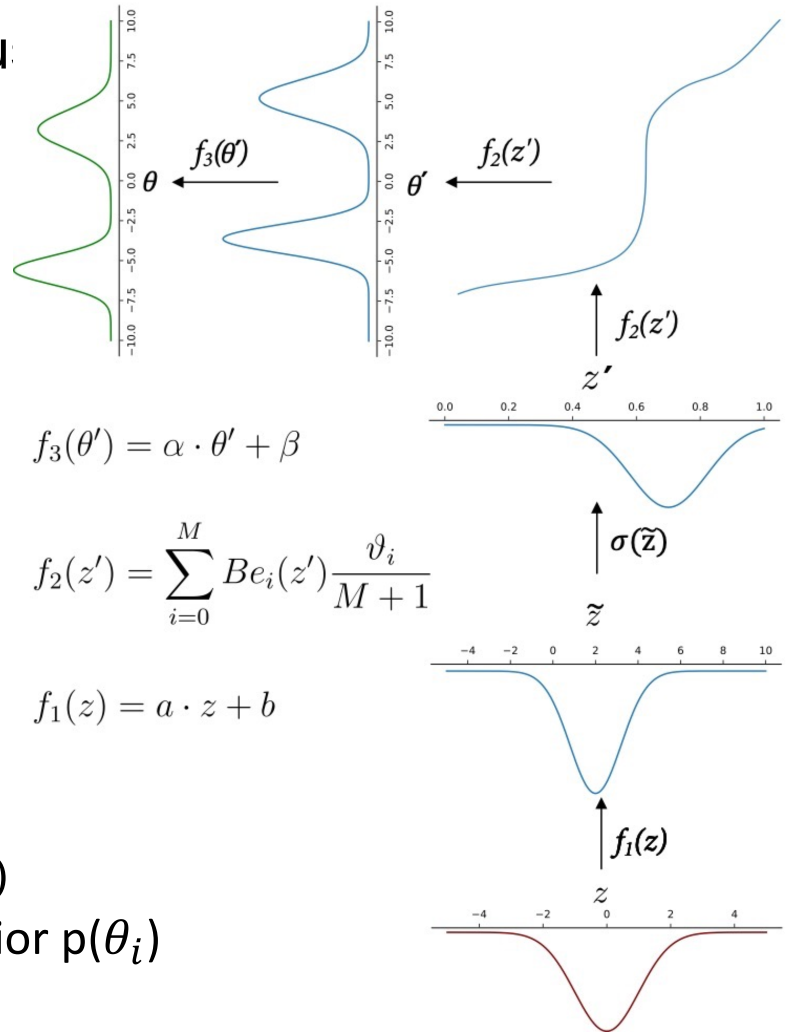
Most Likely Transformation (MLT) 2017 by T.Hothorn, L.Möst, P.Bühlmann https://onlinelibrary.wiley.com/doi/full/10.1111/sjos.12291
Use of Bernstein Polynomials to model complex predictive distribution $p(y|x)$ using NN to control ϑ_k's

- B. Sick, T. Hothorn, O. Dürr (2021) ICPR, Deep Transformation Models introduction of method
- M. Arpogauss et al. (2021) Probabilistic Short-Term Low-Voltage Load Forecasting using Bernstein-Polynomial Normalizing Flows

Single parameter models

Single Parameter Models

- Fit the parameters of the transformation u
Black Box Variational Inference
  - Done using TensorFlow
  - https://github.com/stefan1893/TM-VI
  - Also for mean field
- Sandwich Bernstein With Linear Shift



def nelbo(λ):

θ_i <- Samples $i = 1, \ldots S$ from variational

NLL <- mean(-log(p(D|θ_i))

KL <- mean(log(q$_\lambda(\theta_i)$)- mean(log(p(θ_i))

In transformation models $\lambda = (a, b, \vartheta_0, \ldots, \vartheta_M, \alpha, \beta)$

Sample z_i --> θ_i with that likelihood p(D|θ_i) and prior p(θ_i)

Little trick q$_\lambda(\theta_i) = p(z_i)$

Bernoulli experiment as one-parameter-model

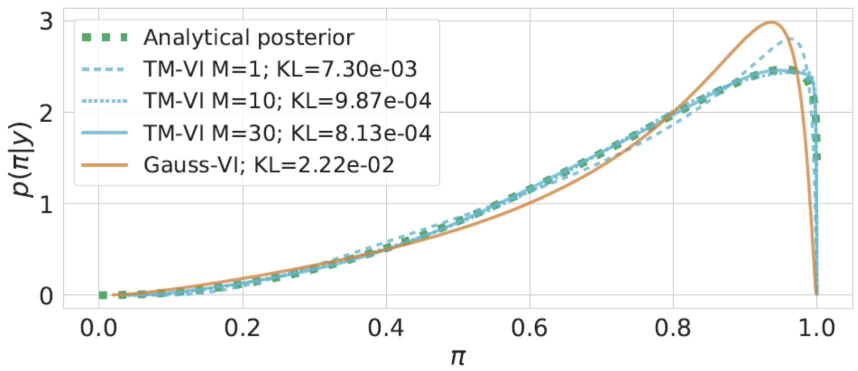
Bernoulli model $y \sim \mathrm{Ber}(\pi)$;   two observations $\mathrm{D} = (y_1 = 1, y_2 = 1)$.

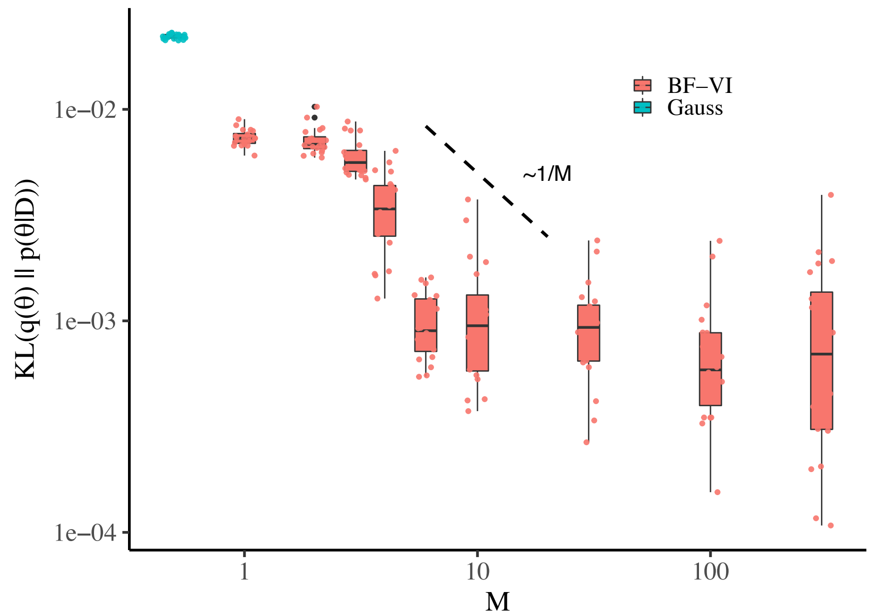
Exact analytical posterior:

Prior: $p(\pi) = \mathrm{Beta}(\alpha = 1.1, \beta = 1.1)$

Likelihood: $\mathrm{p}(D|\pi) = \pi \cdot \pi = \pi^2$

Posterior: $p(\pi|D) = \mathrm{Beta}(\alpha + \sum y_i, \beta + n - \sum y_i) = \mathrm{Beta}(3.3,1.1)$



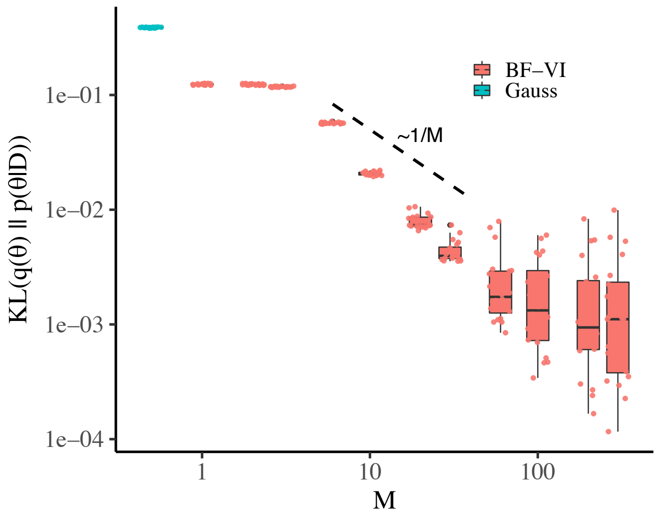


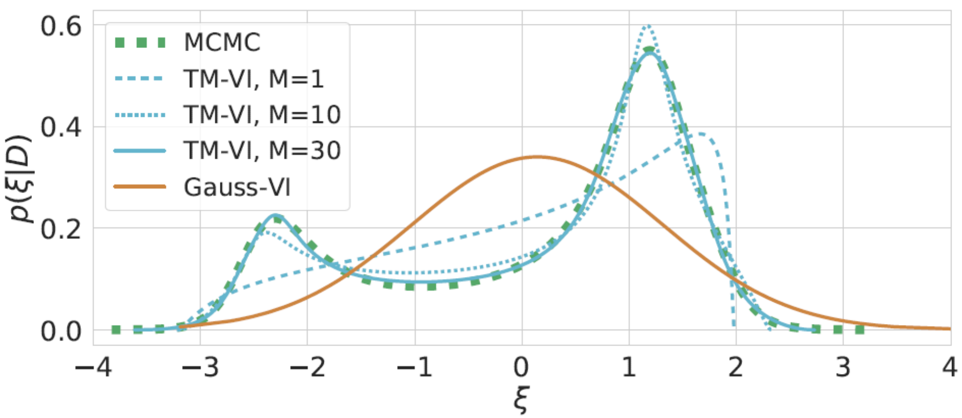
Cauchy experiment as one-parameter-model

Cauchy model $y \sim \text{Cauchy}(\xi; \gamma)$; 6 observations sampled from a mixture-Cauchy*

Exact posterior via MCMC (Stan):

```
data{
  int<lower=0> N;
  real<lower=0> gamma;
  vector[N] y;
}
parameters{
  real xi;
}
model{
  y ~ cauchy(xi, gamma);  // likelihood
  xi ~ normal(0, 1);          // prior
}
```





*Yao, Y., Vehtari, A., and Gelman, A. The curse and blessing of multimodal posteriors. https://arxiv.org/abs/2006.12335

Multi-parameter models

Mean-field approximation for multi-parameter-models

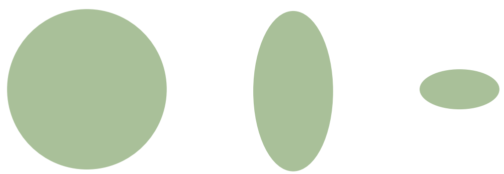
In **mean-field VI** we assume that we can model all variational distributions independently.

Hence the joint variational distribution is given by a product of marginal distributions:

$$q_\lambda(\boldsymbol{w}) = \prod_{k=1}^{p} q_{\lambda_k}(w_k)$$

Pros: no need to model dependencies
→ less parameters are needed

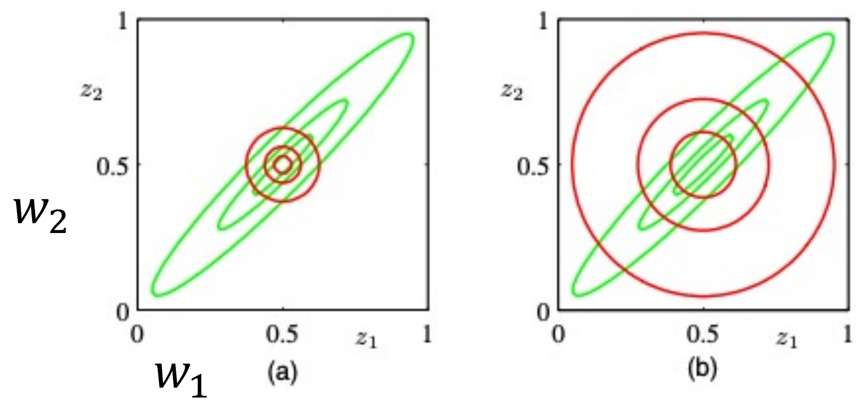
Possible bivariate Gaussians w/o dependencies



Impossible bivariate Gaussians with dependencies

Cons: dependencies are ignored

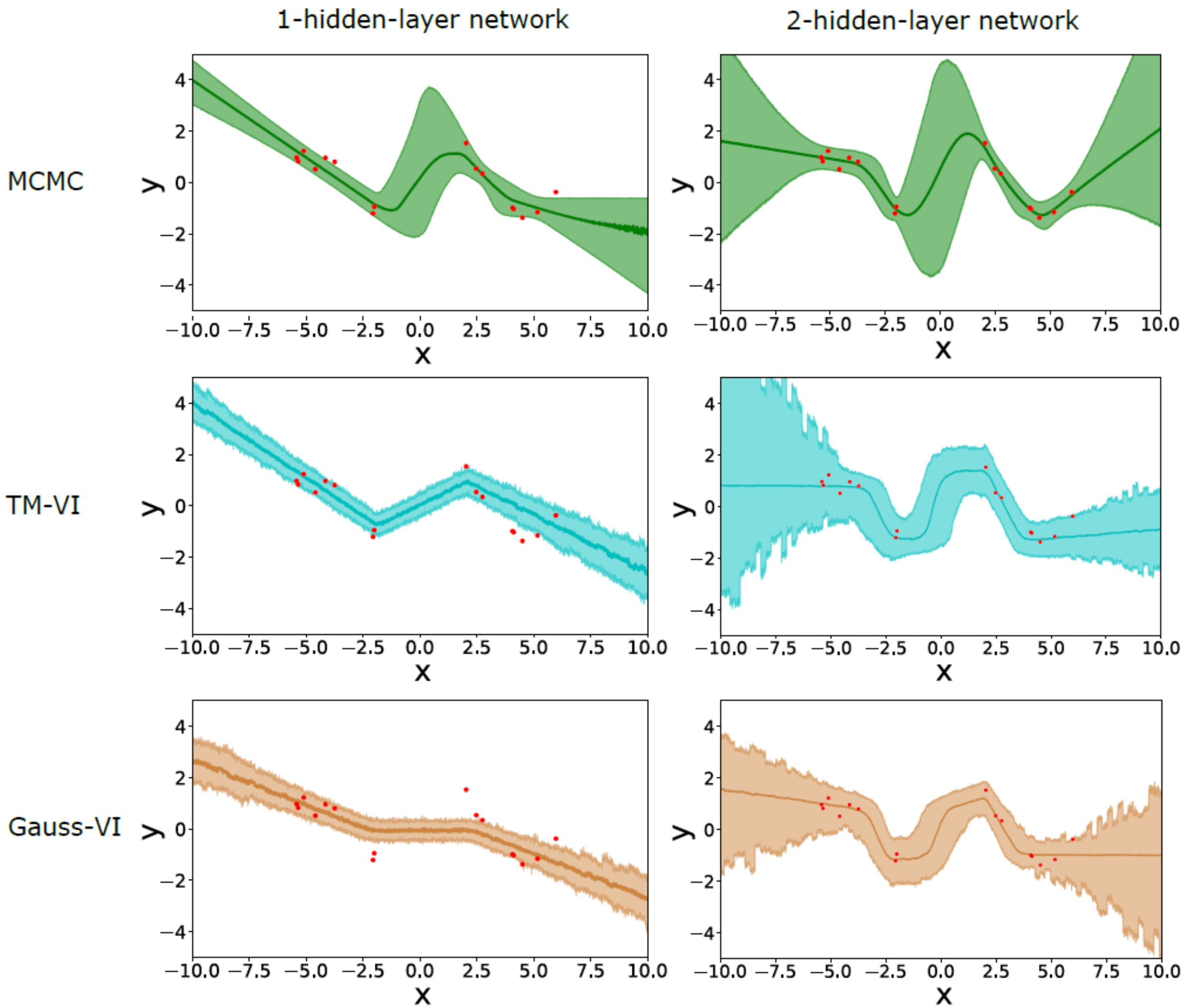
Our $KL(q||p)$ Reveres $KL(p||q)$



The famous figure 10.2 from Bishop

Mean-field VI for multi-parameter NN

We use Bayesian NNs to estimate the conditional mean $\mu(x)$ of $(y|x) \sim N(\mu(x), \sigma)$



Both VI-approaches underestimate the uncertainty. TM-VI can't leverage in mean-field.

Note: For Gaussian-VI ist known that mean-field does not hurt in deep NN https://arxiv.org/abs/2002.03704

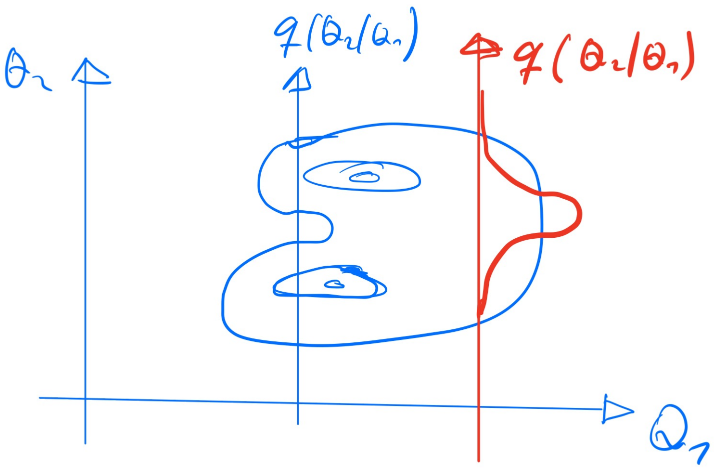
Modeling Dependencies with a NN

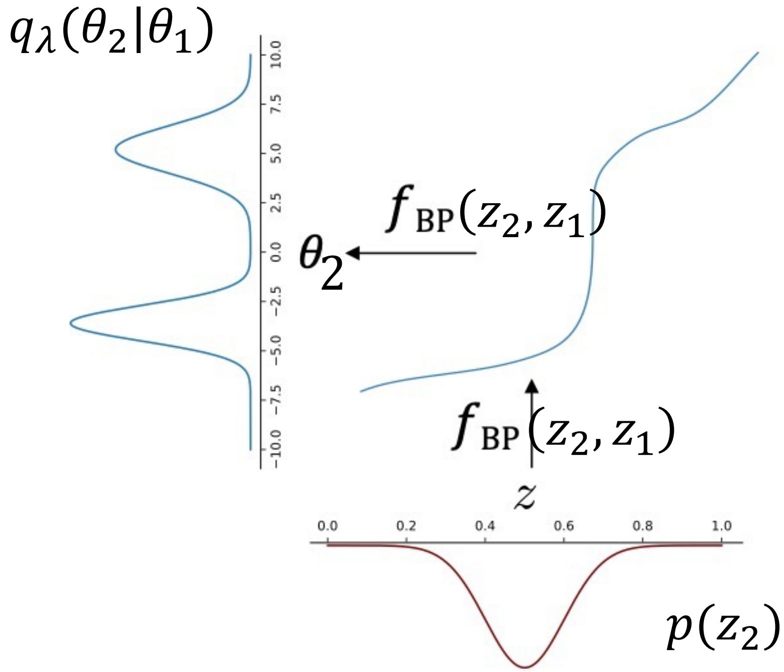
M+1 constants

- Use $q(\theta_1, \theta_2) = q(\theta_2|\theta_1)q(\theta_1)$
- $\theta_1 = f_{BP}(z_1) = \frac{1}{M+1} \sum \vartheta_i^1 \cdot Be_i(z_1)$

This is controlled by NN

- $\theta_2|\theta_1 = f_{BP}(z_1, z_2) = \frac{1}{M+1} \sum \vartheta_i^2(z_1) \cdot Be_i(z_2)$





Modeling Dependencies with a NN cnt'd

Transformation depends on smaller components (Triangular Map)

$$\theta_j = f_{\mathrm{BP}j}(z_{1:k\leq j}) = \frac{1}{M+1}\sum_{i=0}^{M} \vartheta_i^j(z_1,\ldots,z_{j-1})\,\mathrm{Be}_i(z_j)$$

This is a NN for $j \geq 2$

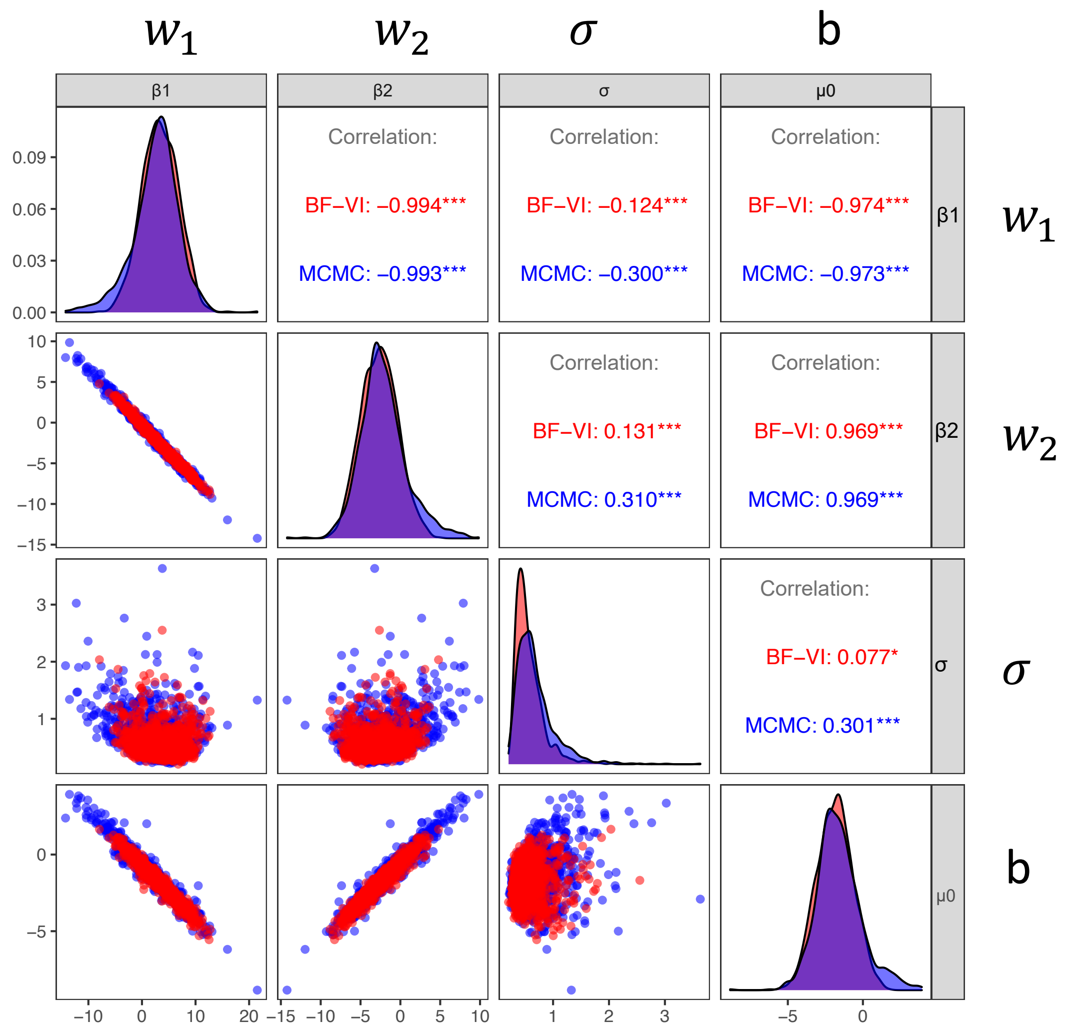
$$q_{\boldsymbol{\lambda}}(\boldsymbol{\theta}_s) = p(z_s)\cdot|\det\nabla \boldsymbol{f}_{\mathrm{BP}}(\boldsymbol{z}_s))|^{-1}$$

Easy to calculate (Triangular Matrix)

For 2D, we need constants coefficients $\vartheta_0^1, \vartheta_1^1, \ldots \vartheta_M^1$ and a network
For 3D, we would need 2 networks…
Efficient way, using Masked Autoregressive Flow (MAF) Networks which do not depend on earlier coordinates.

Results: Simple 1-D Regression Experiment

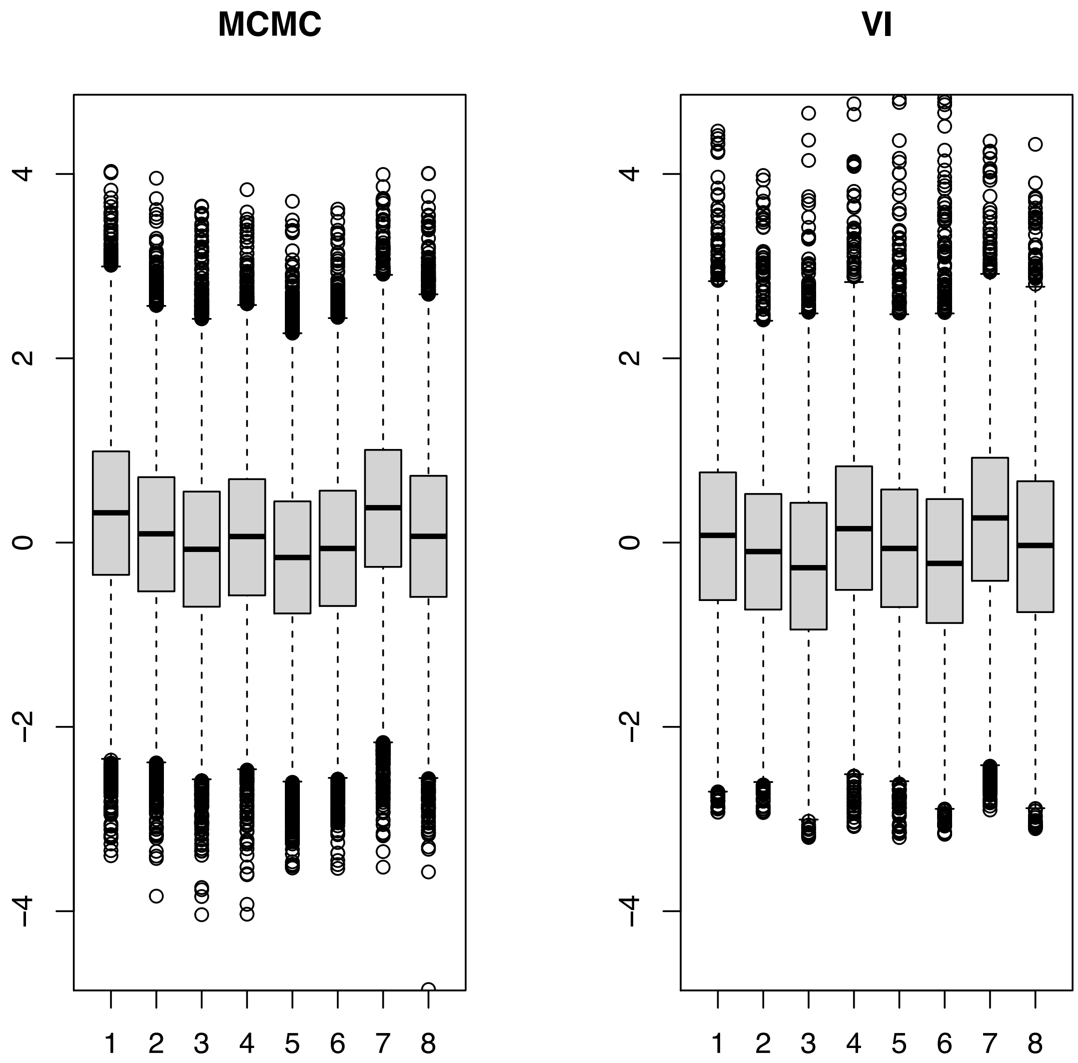
```
data {
  int<lower=0> N;
  int<lower=1> P;
  vector[N] y;
  matrix[N,P] x;
}

parameters {
  vector[P] w;
  real b;
  real<lower=0> sigma;
}

model {
  y ~ normal(x * w + b, sigma);
  b ~ normal(0,10);
  w ~ normal(0, 10);
  sigma ~ lognormal(0.5,1);
}
```



Results 8 Schools NCP (Standard Bayesian Benchmark)

Model Definition in Stan

```
data {
  int<lower=0> J;
  real y[J];
  real<lower=0> sigma[J];
}

parameters {
  real mu;
  real<lower=0> tau;
  real theta_tilde[J];
}

transformed parameters {
  real theta[J];
  for (j in 1:J)
    theta[j] = mu + tau * theta_tilde[j];
}

model {
  mu ~ normal(0, 5);
  tau ~ cauchy(0, 5);
  theta_tilde ~ normal(0, 1);
  y ~ normal(theta, sigma);
}
```



Comparison with other methods (k-hat estimator)

$\hat{k} \in [0, \infty]$ is measure of quality for VI samples**

Model	BF-VI	NF-Planar*	NF-NVP*	Gaussian MF	
8 Schools CP	**0.61**	1.3	1.1	0.9	
8 Schools NCP	**0.35**	1.2	0.7	0.7	
Diamond	30.23	∞	∞	**1.2**	Larger Dataset → Posteriors spiked Gaussians.

**Y. Yao, A. Vehtari, D. Simpson, and A. Gelman, "Yes, but did it work?: Evaluating variational inference," in International Conference on Machine Learning. PMLR, 2018, pp. 5581–5590.

*A. K. Dhaka, A. Catalina, M. Welandawe, M. R. Andersen, J. Huggins, and A. Vehtari, "Challenges and opportunities in high-dimensional variational inference," *arXiv preprint arXiv:2103.01085*, 2021.

Semi Structured Models

Goal

Example: Images from Melanoma and clinical data (e.g. age of patient)

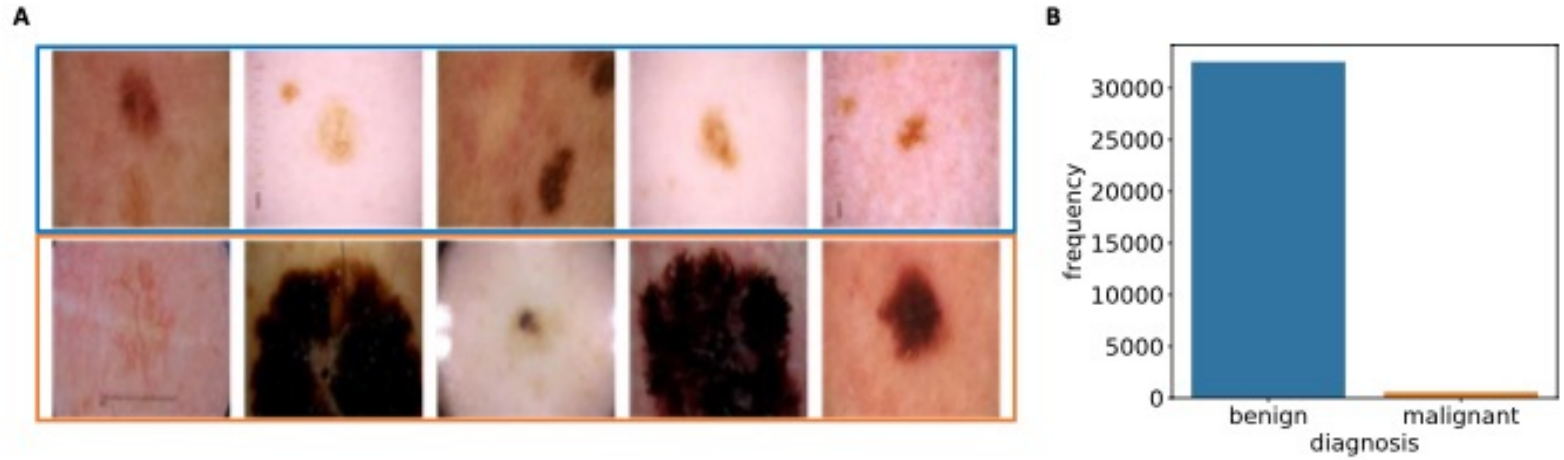
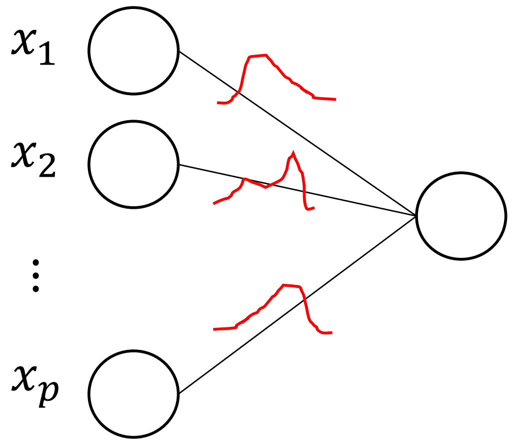


Figure 2.1: A: Example skin lesions from the ISIC 2020 Challenge Dataset with diagnosis 'benign' (top row) and 'malignant' (bottom row). **B:** Distribution of binary outcome variable of ISIC 2020 Challange Dataset. The outcome is quite imbalanced with $\approx 98\%$ 'benign' ($y = 0$) and $\approx 2\%$ 'malignant' ($y = 1$) diagnosis.

- Modeling complex posteriors for statistical (Bayesian) Models
- Modeling combination of DL models and interpretable statistical models

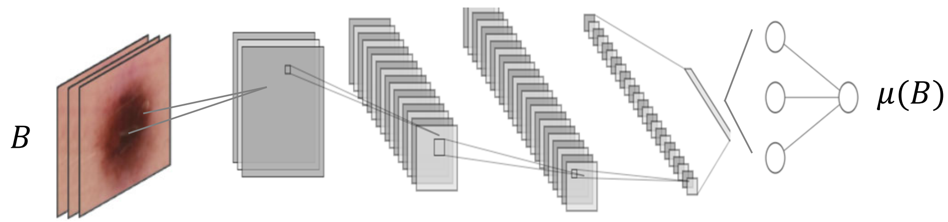
Background: Bayesian Statistics vs. Bayesian Deep Learning

Complex posteriors $p(\theta|D)$
"Classical Bayesian Statistics"



- Parameters θ have **interpretation**
- Gold Standard MCMC Simulations works
  - Breaks down in Big Data regime
- VI works
  - Need for complex distributions

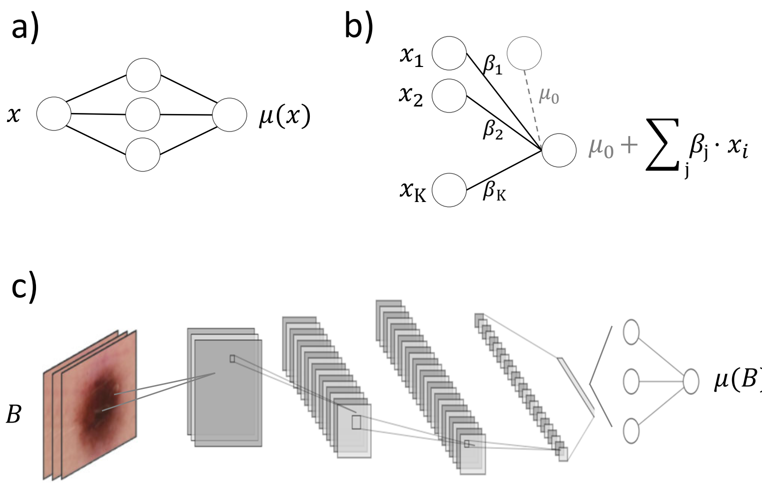
Deep Learning

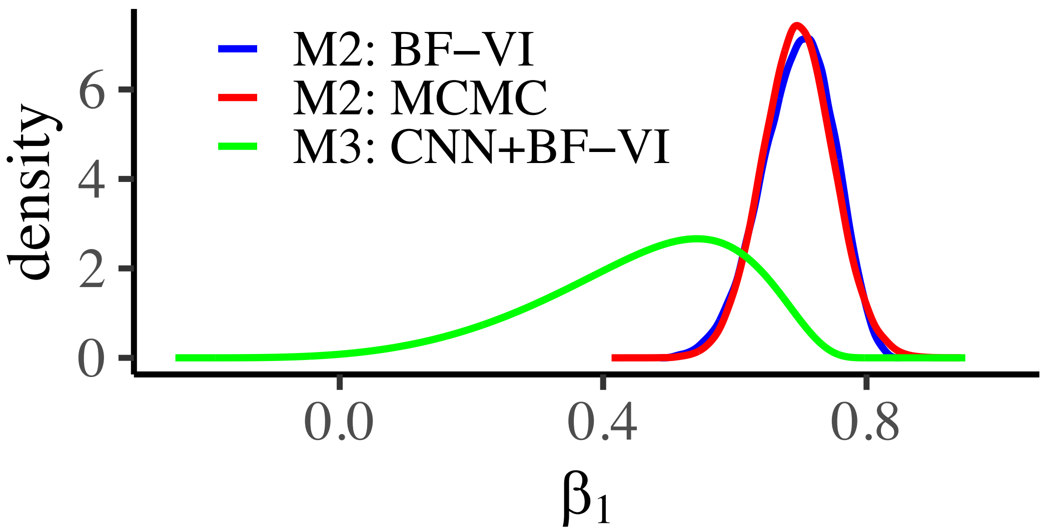


- Parameters θ have no interpretation (**weight space**)
- Important is outcome (**function space**)
- Simple Approximations also OK
  - Liberty or depth paper (Gal 2021)
- MCMC Simulations are not possible
- VI works with simple MF distributions
  - No urgent need for complex distributions

Sometimes we need complex distributions and don't know the distribution family.

First Result for Semi Structured Models





M2: $h = \beta_0 + \beta_1 \cdot x$

M3: $h = \mu_0(B) + \beta_1 \cdot x$

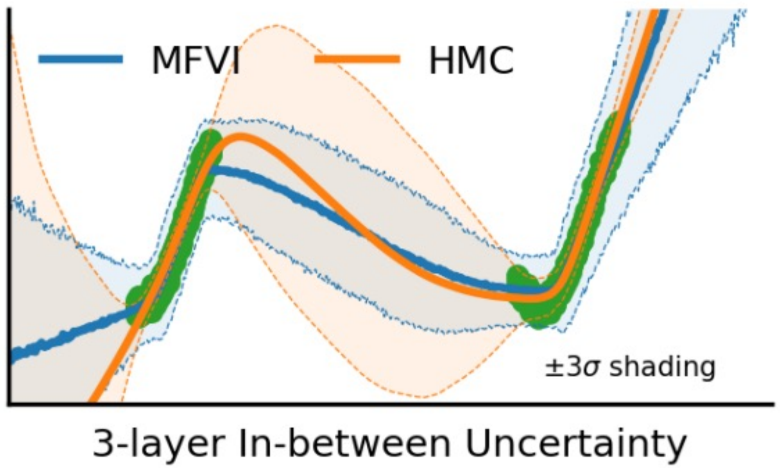
Conclusion and outlook

- VI allows for approximating posterior by an optimization process. Can use DL Toolbox
- VI special important for semi-structured Bayesian models inaccessible so far (in our opinion)
- Current Challenges of VI
    1. Constructing variational distributions that are flexible enough to match the true posterior distribution
    2. Defining suited variational objective for tuning the variational distribution, which boils down to finding the most suited divergence measure
    3. Developing robust and accurate stochastic optimization frameworks for the variational objective
- Contributed to 1, will work on 2 and 3.

Thanks

Attik

Bayes to the rescue: Results for Bayesian NN for 1-D



(b) 3-layer BNN

https://arxiv.org/pdf/2002.03704.pdf
"Liberty or Depth"

MCMC methods such as HMC are the gold standard for Bayes. MCMC is sampling not fitting.
MCMC has Problems in Scaling

- Larger number of Data Points (no mini-batch) $p(\theta|D)$ has the "Daten an der Backe"
- Large models sizes (Deep Learning is out of scope)