02477 – Bayesian Machine Learning: Lecture 13

Michael Riis Andersen

Technical University of Denmark,
DTU Compute, Department of Applied Math and Computer Science

Outline

1. Wrapping up VI
   - Variational autoencoders
   - Summary

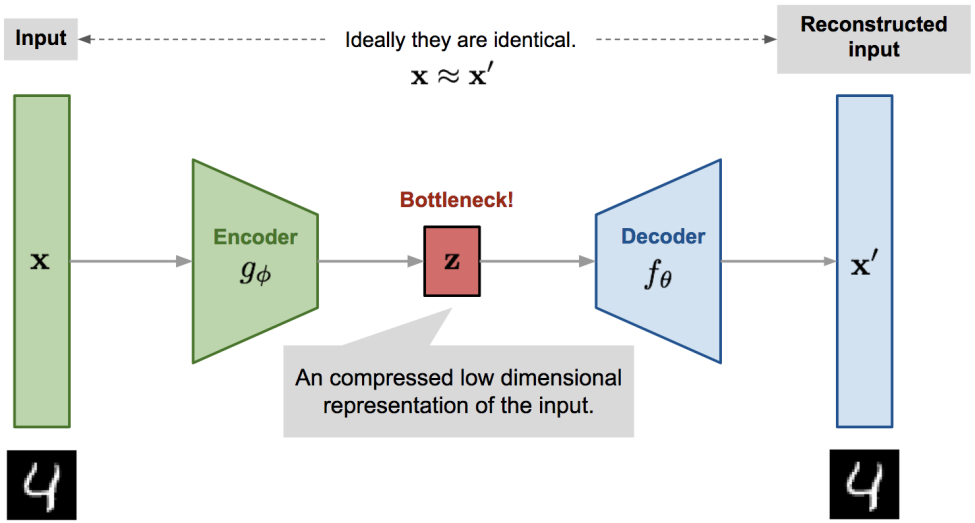
2. Bayesian Neural Networks
   - Motivation and challenges
   - Approximating the posterior for neural networks
   - Monte Carlo Dropout
   - Mean-field
   - Laplace approximations
   - Deep Ensembles

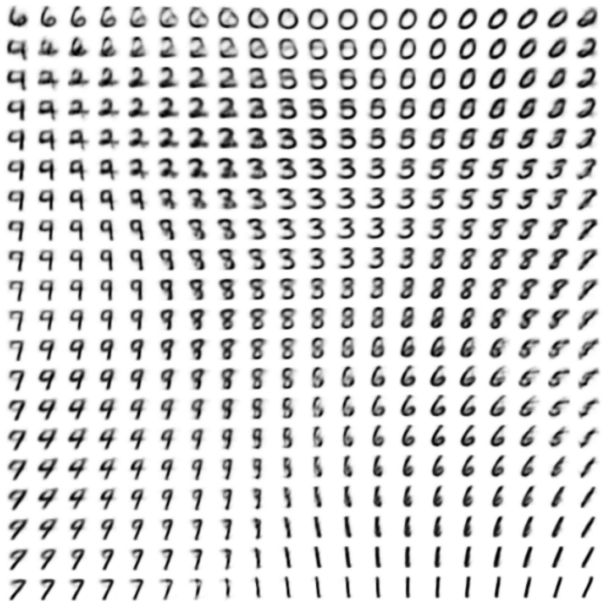
Wrapping up VI

Wrapping up VI: Variational autoencoders

Variational autoencoders (VAEs)

How does VAEs related to all this?





Variational auto-encoders

- Unsupervised learning
- Finds low-dimensional latent representation $\boldsymbol{z} \in \mathbb{R}^K$ for data $\boldsymbol{x} \in \mathbb{R}^D$ (encoder)
- Generative model: Can sample $\boldsymbol{z}$ and transform back to data-space (decoder)

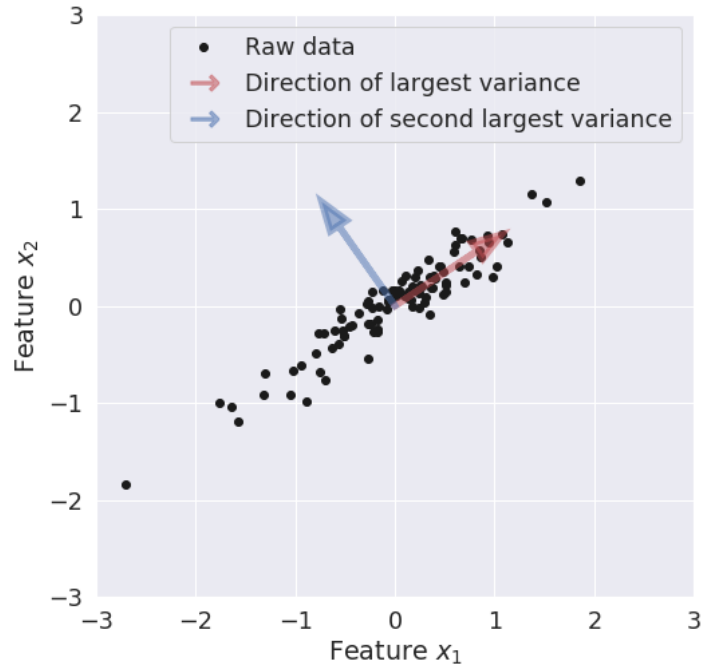
https://lilianweng.github.io/posts/2018-08-12-vae/

Kingma & Welling, 2014: Auto-encoding variational Bayes

Factor models: PCA and pPCA

- Principal component analysis

$$\boldsymbol{x}_n \approx \mathbf{W}\boldsymbol{z}_n$$



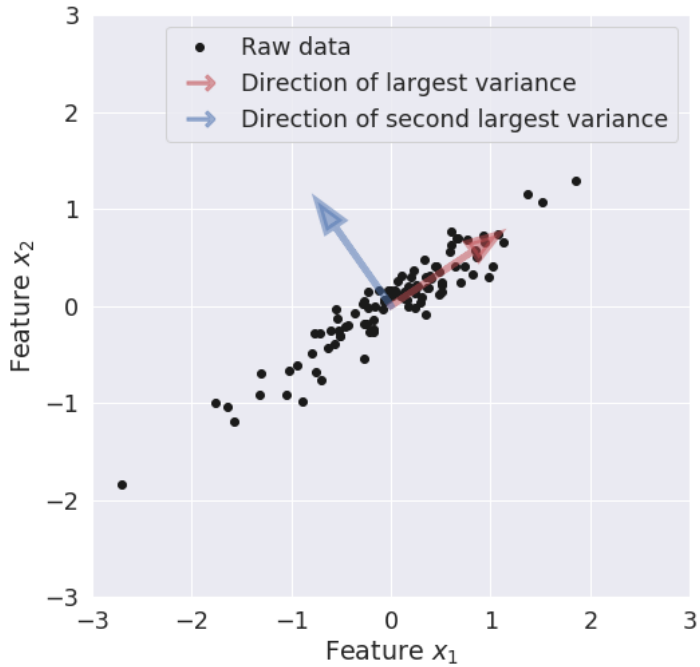
Factor models: PCA and pPCA

- Principal component analysis

$$\boldsymbol{x}_n \approx \mathbf{W}\boldsymbol{z}_n$$

- Probabilistic PCA (PPCA)

$$\boldsymbol{x}_n|\boldsymbol{z}_n \sim \mathcal{N}(\mathbf{W}\boldsymbol{z}_n, \sigma^2\boldsymbol{I})$$
$$\boldsymbol{z}_n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$



Factor models: PCA and pPCA

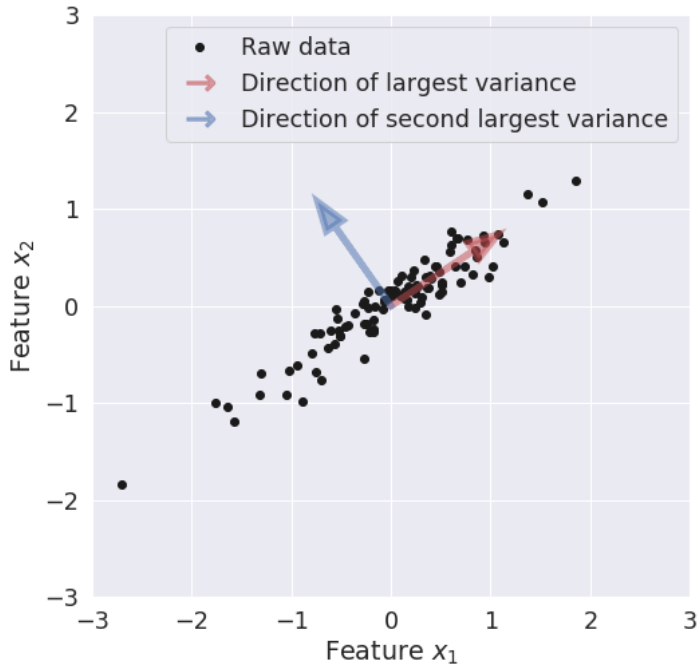
- Principal component analysis

$$\boldsymbol{x}_n \approx \mathbf{W}\boldsymbol{z}_n$$

- Probabilistic PCA (PPCA)

$$\boldsymbol{x}_n|\boldsymbol{z}_n \sim \mathcal{N}(\mathbf{W}\boldsymbol{z}_n, \sigma^2\boldsymbol{I})$$
$$\boldsymbol{z}_n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

- Properties of PPCA
  1. Linear Gaussian model
  2. Conjugate model: easy inference
  3. Generative model: can sample new data points
  4. Can handle missing data
  5. Enables probabilistic reasoning



Factor models: PCA and pPCA

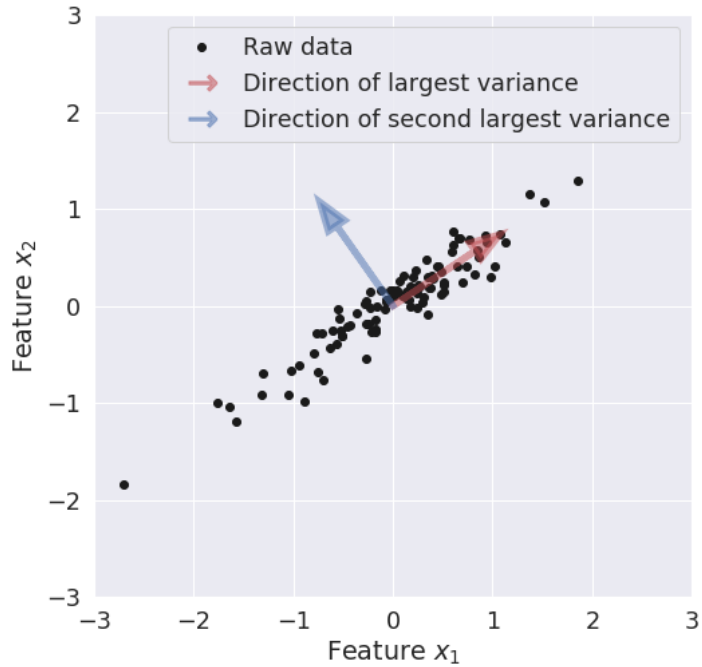
- Principal component analysis

$$\boldsymbol{x}_n \approx \mathbf{W}\boldsymbol{z}_n$$

- Probabilistic PCA (PPCA)

$$\boldsymbol{x}_n|\boldsymbol{z}_n \sim \mathcal{N}(\mathbf{W}\boldsymbol{z}_n, \sigma^2\boldsymbol{I})$$
$$\boldsymbol{z}_n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

- Properties of PPCA
  1. Linear Gaussian model
  2. Conjugate model: easy inference
  3. Generative model: can sample new data points
  4. Can handle missing data
  5. Enables probabilistic reasoning



- Replacing the linear model with a neural network $f_\theta(\boldsymbol{z}) = \text{NN}_{\boldsymbol{\theta}}(\boldsymbol{z})$

$$\boldsymbol{x}_n|\boldsymbol{z}_n \sim \mathcal{N}(f_\theta(\boldsymbol{z}_n), \sigma^2\boldsymbol{I})$$
$$\boldsymbol{z}_n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

Amortized variational inference

- Non-linear factor model

$$\boldsymbol{x}_n|\boldsymbol{z}_n \sim \mathcal{N}(f_\theta(\boldsymbol{z}_n), \sigma^2 \boldsymbol{I})$$
$$\boldsymbol{z}_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

Amortized variational inference

- Non-linear factor model

$$\boldsymbol{x}_n|\boldsymbol{z}_n \sim \mathcal{N}(f_\theta(\boldsymbol{z}_n), \sigma^2 \boldsymbol{I})$$
$$\boldsymbol{z}_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

- No longer conjugate: variational inference to the rescue

$$q(\boldsymbol{Z}) = \prod_{n=1}^{N} q(\boldsymbol{z}_n) \qquad q(\boldsymbol{z}_n) = \prod_{k=1}^{K} \mathcal{N}(z_{nk}|m_{nk}, v_{nk})$$

Amortized variational inference

- Non-linear factor model

$$\boldsymbol{x}_n|\boldsymbol{z}_n \sim \mathcal{N}(f_\theta(\boldsymbol{z}_n), \sigma^2 \boldsymbol{I})$$
$$\boldsymbol{z}_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

- No longer conjugate: variational inference to the rescue

$$q(\boldsymbol{Z}) = \prod_{n=1}^{N} q(\boldsymbol{z}_n) \qquad q(\boldsymbol{z}_n) = \prod_{k=1}^{K} \mathcal{N}(z_{nk}|m_{nk}, v_{nk})$$

- *Amortized inference*: Avoiding the number of variational parameters growing with the number of data points

$$q(\boldsymbol{z}_n) = \prod_{k=1}^{K} \mathcal{N}(z_{nk}|m(\boldsymbol{x}_n), v(\boldsymbol{x}_n))$$

Amortized variational inference

- Non-linear factor model

$$\boldsymbol{x}_n|\boldsymbol{z}_n \sim \mathcal{N}(f_\theta(\boldsymbol{z}_n), \sigma^2 \boldsymbol{I})$$
$$\boldsymbol{z}_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

- No longer conjugate: variational inference to the rescue

$$q(\boldsymbol{Z}) = \prod_{n=1}^{N} q(\boldsymbol{z}_n) \qquad q(\boldsymbol{z}_n) = \prod_{k=1}^{K} \mathcal{N}(z_{nk}|m_{nk}, v_{nk})$$

- *Amortized inference*: Avoiding the number of variational parameters growing with the number of data points

$$q(\boldsymbol{z}_n) = \prod_{k=1}^{K} \mathcal{N}(z_{nk}|m(\boldsymbol{x}_n), v(\boldsymbol{x}_n))$$

where m and v controlled by another neural network g_ϕ with two outputs

$$m(\boldsymbol{x}_n) = g_\phi^1(\boldsymbol{x}_n)$$
$$\ln v(\boldsymbol{x}_n) = g_\phi^2(\boldsymbol{x}_n)$$

Variational autoencoders

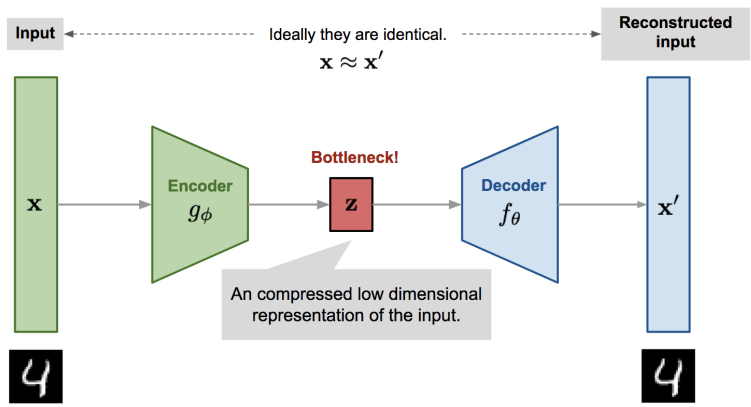
How does VAEs relates to BBVI?

- Probabilistic model (*decoder*)

$$\boldsymbol{x}_n|\boldsymbol{z}_n \sim \mathcal{N}(f_\theta(\boldsymbol{z}_n), \sigma^2 \boldsymbol{I})$$
$$\boldsymbol{z}_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

- Variational approximation (*encoder*)

$$q(\boldsymbol{z}_n) = \prod_{k=1}^{K} \mathcal{N}(z_{nk}|m(\boldsymbol{x}_n), v(\boldsymbol{x}_n))$$
$$m(\boldsymbol{x}_n) = g_\phi^1(\boldsymbol{x}_n)$$
$$\ln v(\boldsymbol{x}_n) = g_\phi^2(\boldsymbol{x}_n)$$



Credit: https://lilianweng.github.io/posts/2018-08-12-vae/

Variational autoencoders

How does VAEs relates to BBVI?

- Probabilistic model (*decoder*)

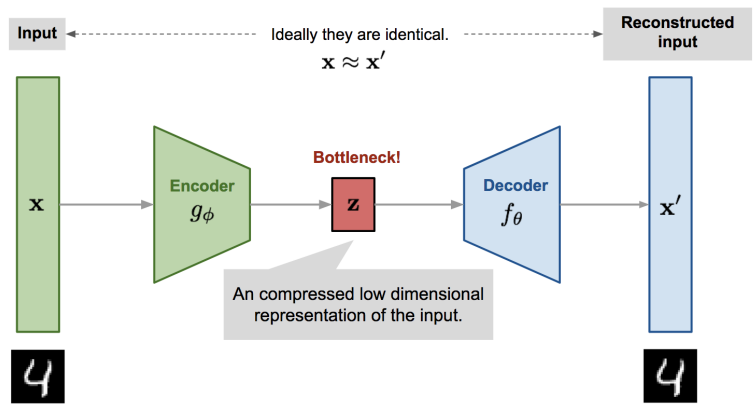
$$\boldsymbol{x}_n|\boldsymbol{z}_n \sim \mathcal{N}(f_\theta(\boldsymbol{z}_n), \sigma^2 \boldsymbol{I})$$
$$\boldsymbol{z}_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

- Variational approximation (*encoder*)

$$q(\boldsymbol{z}_n) = \prod_{k=1}^{K} \mathcal{N}(z_{nk}|m(\boldsymbol{x}_n), v(\boldsymbol{x}_n))$$
$$m(\boldsymbol{x}_n) = g_\phi^1(\boldsymbol{x}_n)$$
$$\ln v(\boldsymbol{x}_n) = g_\phi^2(\boldsymbol{x}_n)$$

- Fit model by optimizing the ELBO wrt. ϕ, θ (NN parameters)

$$\arg\max_{\theta,\phi} \mathcal{L}[q]$$



Credit: https://lilianweng.github.io/posts/2018-08-12-vae/

Variational autoencoders

How does VAEs relates to BBVI?

- Probabilistic model (*decoder*)

$$\boldsymbol{x}_n|\boldsymbol{z}_n \sim \mathcal{N}(f_\theta(\boldsymbol{z}_n), \sigma^2 \boldsymbol{I})$$
$$\boldsymbol{z}_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

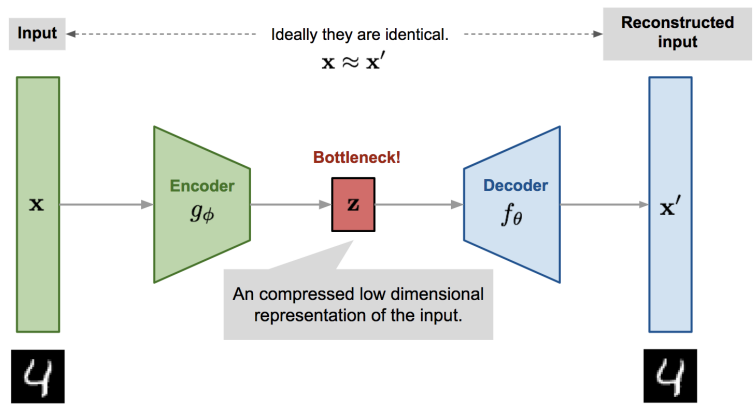
- Variational approximation (*encoder*)

$$q(\boldsymbol{z}_n) = \prod_{k=1}^{K} \mathcal{N}(z_{nk}|m(\boldsymbol{x}_n), v(\boldsymbol{x}_n))$$
$$m(\boldsymbol{x}_n) = g_\phi^1(\boldsymbol{x}_n)$$
$$\ln v(\boldsymbol{x}_n) = g_\phi^2(\boldsymbol{x}_n)$$

- Fit model by optimizing the ELBO wrt. ϕ, θ (NN parameters)

$$\arg\max_{\theta,\phi} \mathcal{L}[q]$$

- Reparametrization trick to estimate gradients



Credit: https://lilianweng.github.io/posts/2018-08-12-vae/

Variational autoencoders

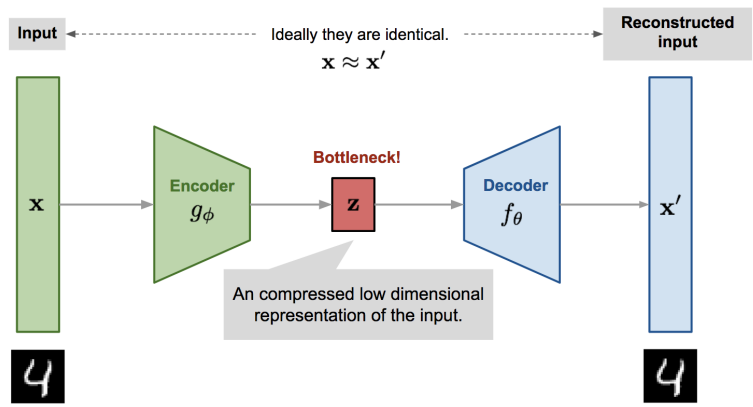
How does VAEs relates to BBVI?

- Probabilistic model (*decoder*)

$$\boldsymbol{x}_n|\boldsymbol{z}_n \sim \mathcal{N}(f_\theta(\boldsymbol{z}_n), \sigma^2 \boldsymbol{I})$$
$$\boldsymbol{z}_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

- Variational approximation (*encoder*)

$$q(\boldsymbol{z}_n) = \prod_{k=1}^{K} \mathcal{N}(z_{nk}|m(\boldsymbol{x}_n), v(\boldsymbol{x}_n))$$
$$m(\boldsymbol{x}_n) = g_\phi^1(\boldsymbol{x}_n)$$
$$\ln v(\boldsymbol{x}_n) = g_\phi^2(\boldsymbol{x}_n)$$



Credit: https://lilianweng.github.io/posts/2018-08-12-vae/

- Fit model by optimizing the ELBO wrt. ϕ, θ (NN parameters)

$$\arg\max_{\theta,\phi} \mathcal{L}[q]$$

- Reparametrization trick to estimate gradients

- VAE = Non-linear factor model + variational inference

Wrapping up VI: Summary

Wrap-up of variational inference I

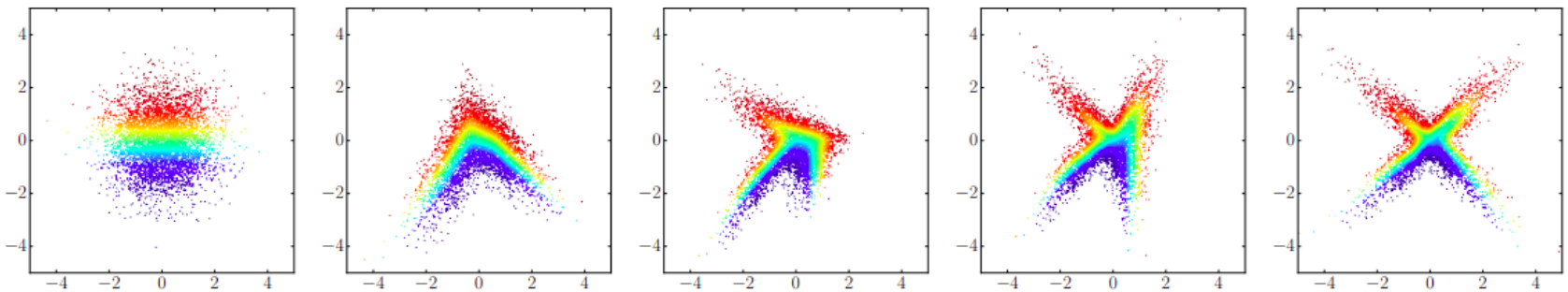
Variational inference

1. Flexible tool for approximating posterior for many models (from linear model to neural networks)
2. Better control of accuracy vs speed trade-off compared to Laplace approx. and MCMC
3. No strong theoretical guarantees as in MCMC
4. Can be scaled to huge datasets using mini batching
5. BBVI is the basis for variational inference in Tensorflow, Pytorch, Pyro, Stan etc.

Active research area: how to improve VI?

Improving variational inference is a very active area of research. Examples

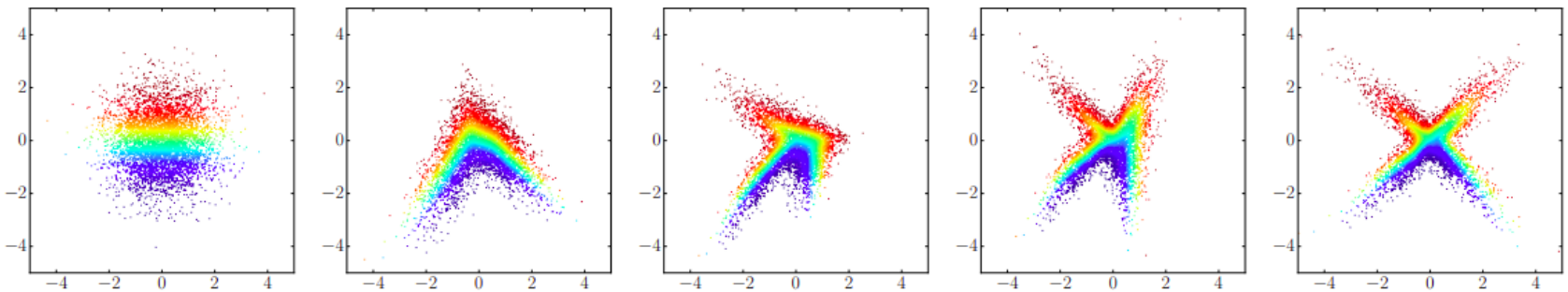
Constructing complex distributions using normalizing flows (Papamakarios et al 2021)



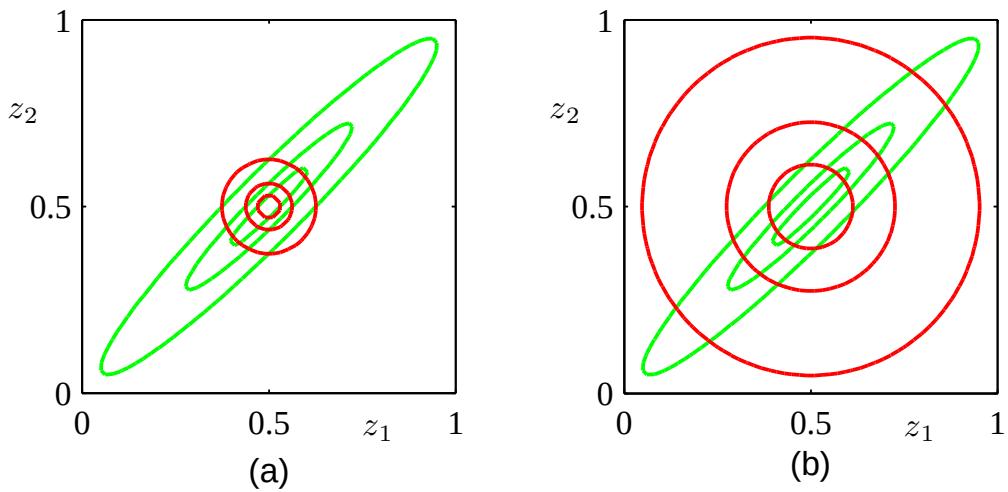
Active research area: how to improve VI?

Improving variational inference is a very active area of research. Examples

Constructing complex distributions using normalizing flows (Papamakarios et al 2021)



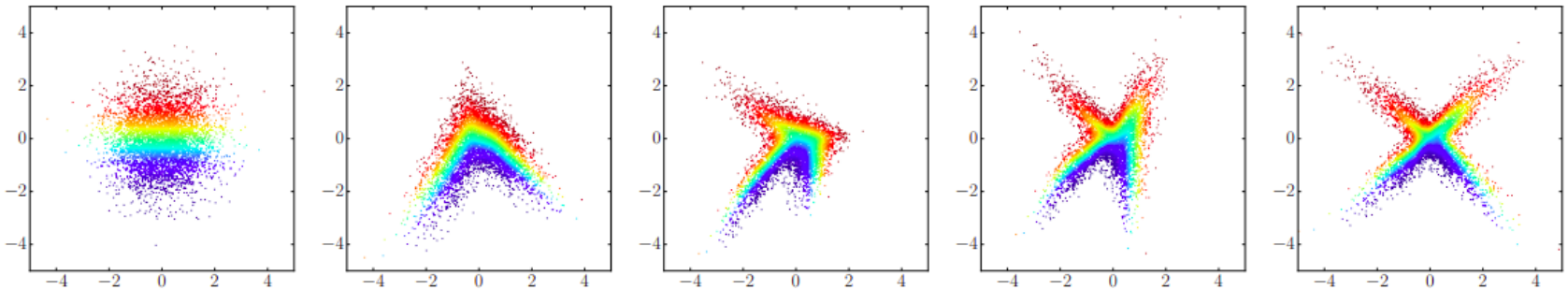
Investigating other divergences (KL$[q||p]$, KL$[p||q]$, α-divergences, χ^2-divergences, f-divergences)



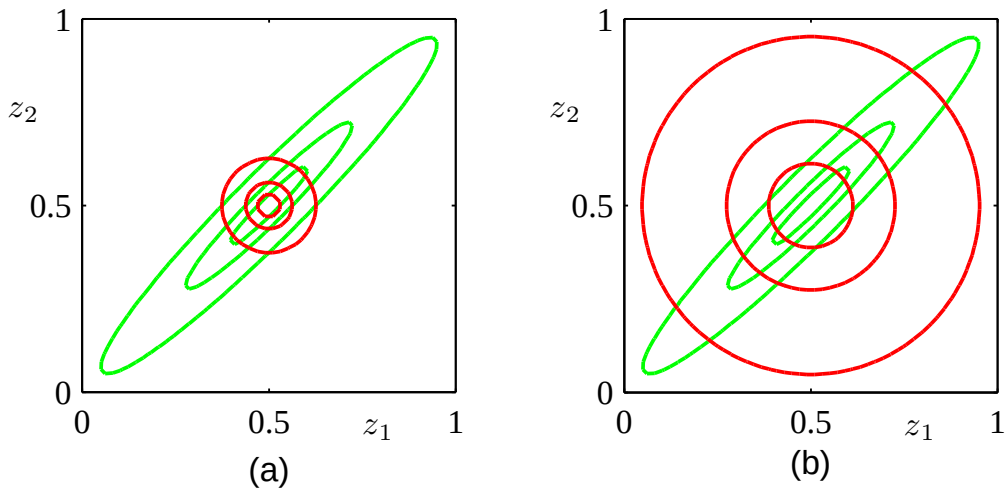
Active research area: how to improve VI?

Improving variational inference is a very active area of research. Examples

Constructing complex distributions using normalizing flows (Papamakarios et al 2021)



Investigating other divergences (KL$[q||p]$, KL$[p||q]$, α-divergences, χ^2-divergences, f-divergences)



Which variational families are most suites for high-dimensional posterior of NNs?

Bayesian Neural Networks

Neural Network notation

- Sequence of linear (affine) and non-linear mappings from input $\mathbf{x}$ to output y

- Two-layer neural network (NN) with single output

$$\begin{aligned} \boldsymbol{z}_1 &= h(\boldsymbol{W}_0 \boldsymbol{x} + \boldsymbol{b}_0) \\ \boldsymbol{z}_2 &= h(\boldsymbol{W}_1 \boldsymbol{z}_1 + \boldsymbol{b}_1) \\ f &= \boldsymbol{W}_2 \boldsymbol{z}_2 + \boldsymbol{b}_2 \end{aligned}$$

- All parameters of a network with L hidden layers

$$\mathbf{w} = \{\mathbf{W}_0, \mathbf{b}_0, \mathbf{W}_1, \mathbf{b}_1, \ldots, \mathbf{W}_L, \mathbf{b}_L\}$$

- From input to output

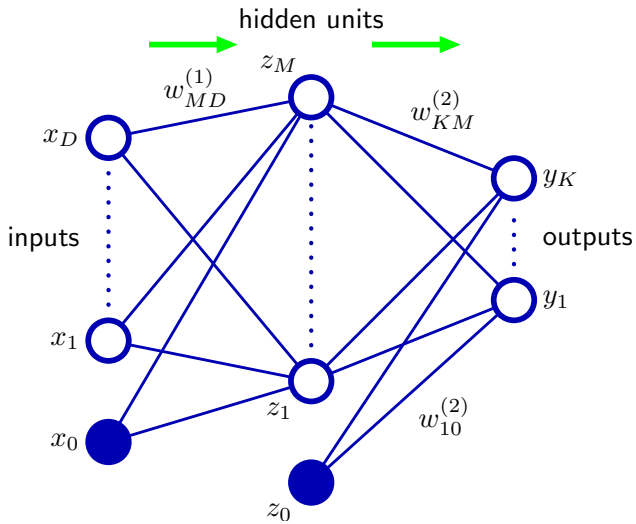
$$f_{\mathbf{w}}(\boldsymbol{x}) = \boldsymbol{W}_2 h(\boldsymbol{W}_1 h(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_0) + \boldsymbol{b}_1) + \boldsymbol{b}_2$$

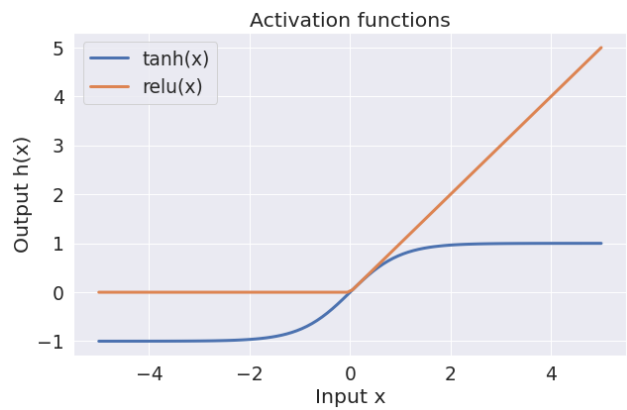
- NNs in probabilistic modelling

$$\begin{aligned} p(y_n|\boldsymbol{w}) &= \mathcal{N}(y_n|f_{\boldsymbol{w}}(\boldsymbol{x}), \sigma^2) \\ p(y_n|\boldsymbol{w}) &= \text{Ber}(y_n|\sigma(f_{\boldsymbol{w}}(\boldsymbol{x}))) \end{aligned}$$

- Completing the model with a prior over the weights, e.g.

$$p(\boldsymbol{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$$



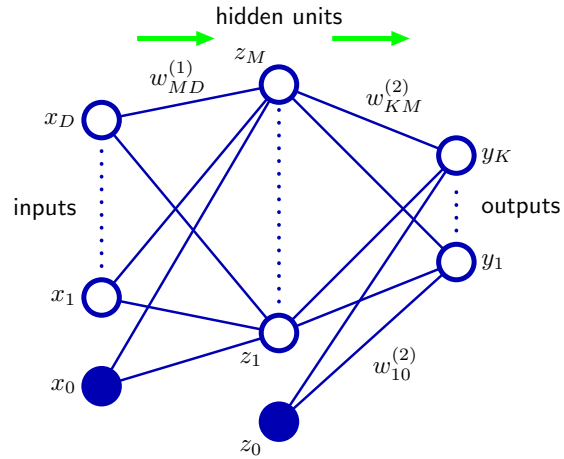


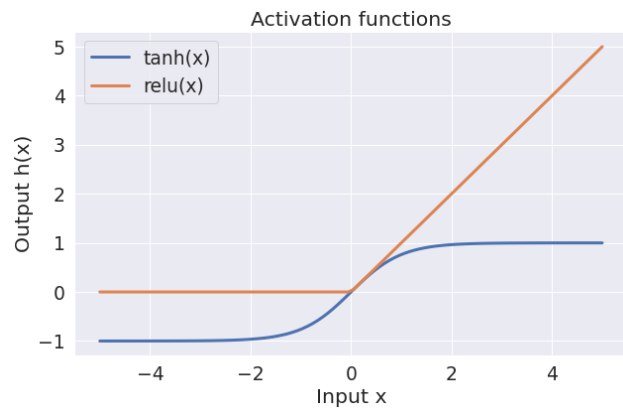
Bayesian Neural Networks: Motivation and challenges

Bayesian Neural Networks: Motivation

- Bayesian deep learning is a very active research area

- Why Bayesian neural networks?
  1. Predictive performance
  2. Epistemic uncertainty quantification
  3. Better calibration
  4. Sequential updating
  5. Less prone to overfitting
  6. Side step pathologies with maximum likelihood learning
  7. Active learning & data efficiency

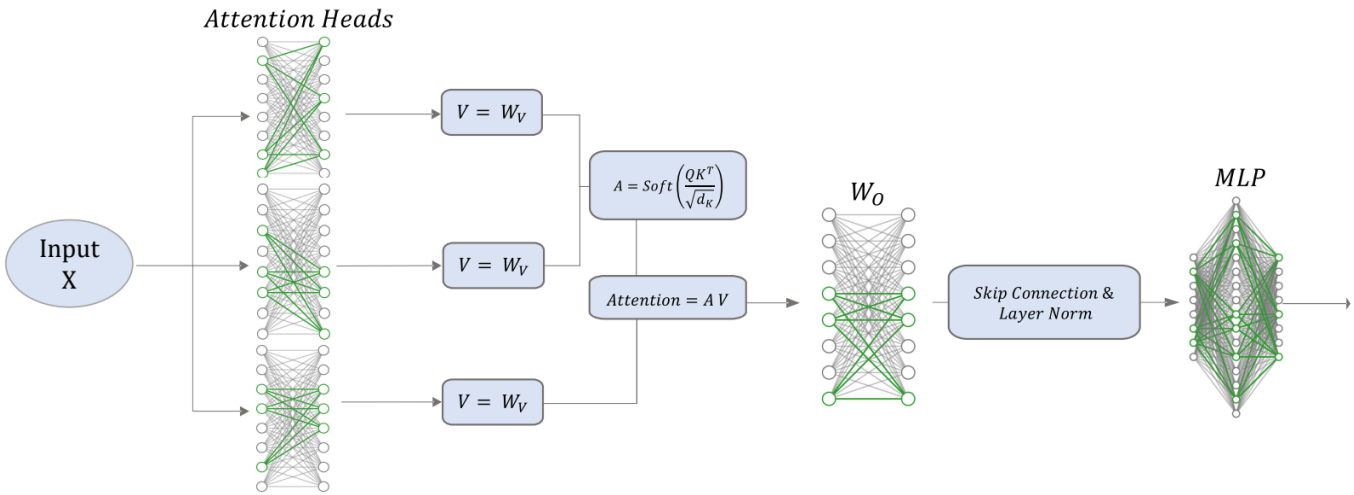




$$\boldsymbol{z}_1 = h(\boldsymbol{W}_0\boldsymbol{x} + \boldsymbol{b}_0)$$
$$\boldsymbol{z}_2 = h(\boldsymbol{W}_1\boldsymbol{z}_1 + \boldsymbol{b}_1)$$
$$f = \boldsymbol{W}_2\boldsymbol{z}_2 + \boldsymbol{b}_2$$

Bayesian Neural Networks: Motivation

- On-going research: Can Bayesian methods to improve Transformers for NLP tasks?
- General Language Understanding Evaluation (GLUE) tasks: Stanford Sentiment Treebank (SST-2), Microsoft Research Paraphrase Corpus (MRPC), Recognizing Textual Entailment (RTE)

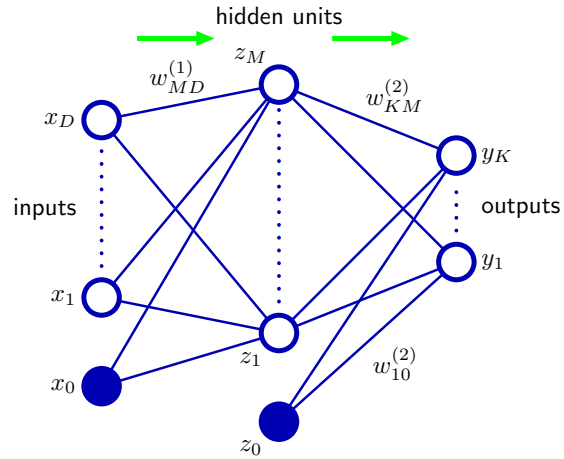


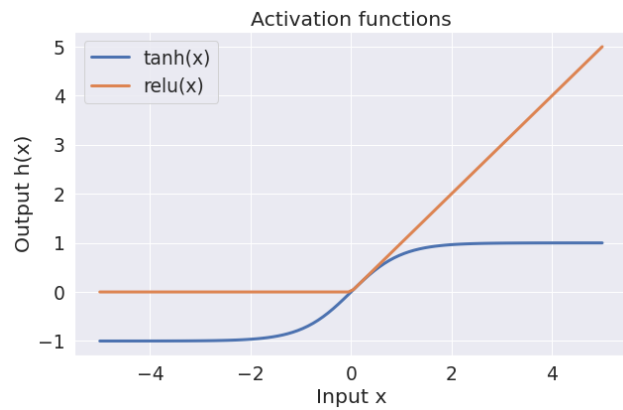
Methods	SST-2		MRPC		RTE	
	NLL↓	ECE↓	NLL↓	ECE↓	NLL↓	ECE↓
MAP	0.74 ± 0.10	0.11 ± 0.01	0.57 ± 0.34	0.14 ± 0.04	1.71 ± 0.23	0.33 ± 0.05
Temp. Scaled	0.36 ± 0.02	0.06 ± 0.00	0.44 ± 0.01	0.05 ± 0.02	0.66 ± 0.01	0.13 ± 0.04
S-KFAC	0.28 ± 0.00	0.04 ± 0.01	**0.37** ± 0.02	**0.04** ± 0.00	**0.65** ± 0.01	0.05 ± 0.01
Last Layer	0.33 ± 0.02	0.06 ± 0.00	0.42 ± 0.01	0.07 ± 0.01	0.67 ± 0.01	0.08 ± 0.03
Fully Stoch. (**LA**)	**0.27** ± 0.01	**0.03** ± 0.00	0.39 ± 0.01	0.06 ± 0.00	0.66 ± 0.01	0.06 ± 0.01
Fully Stoch. (**SWAG**)	**0.27** ± 0.06	**0.03** ± 0.03	0.62 ± 0.11	0.07 ± 0.02	0.69 ± 0.00	0.05 ± 0.02

Peter J. T. Kampen, Gustav R. S. Als, Michael R. Andersen: Towards Scalable Bayesian Transformers: Investigating stochastic subset selection for NLP. *Accepted for publication at Uncertainty in Artificial Intelligence 2024*

Bayesian Neural Networks: Challenges

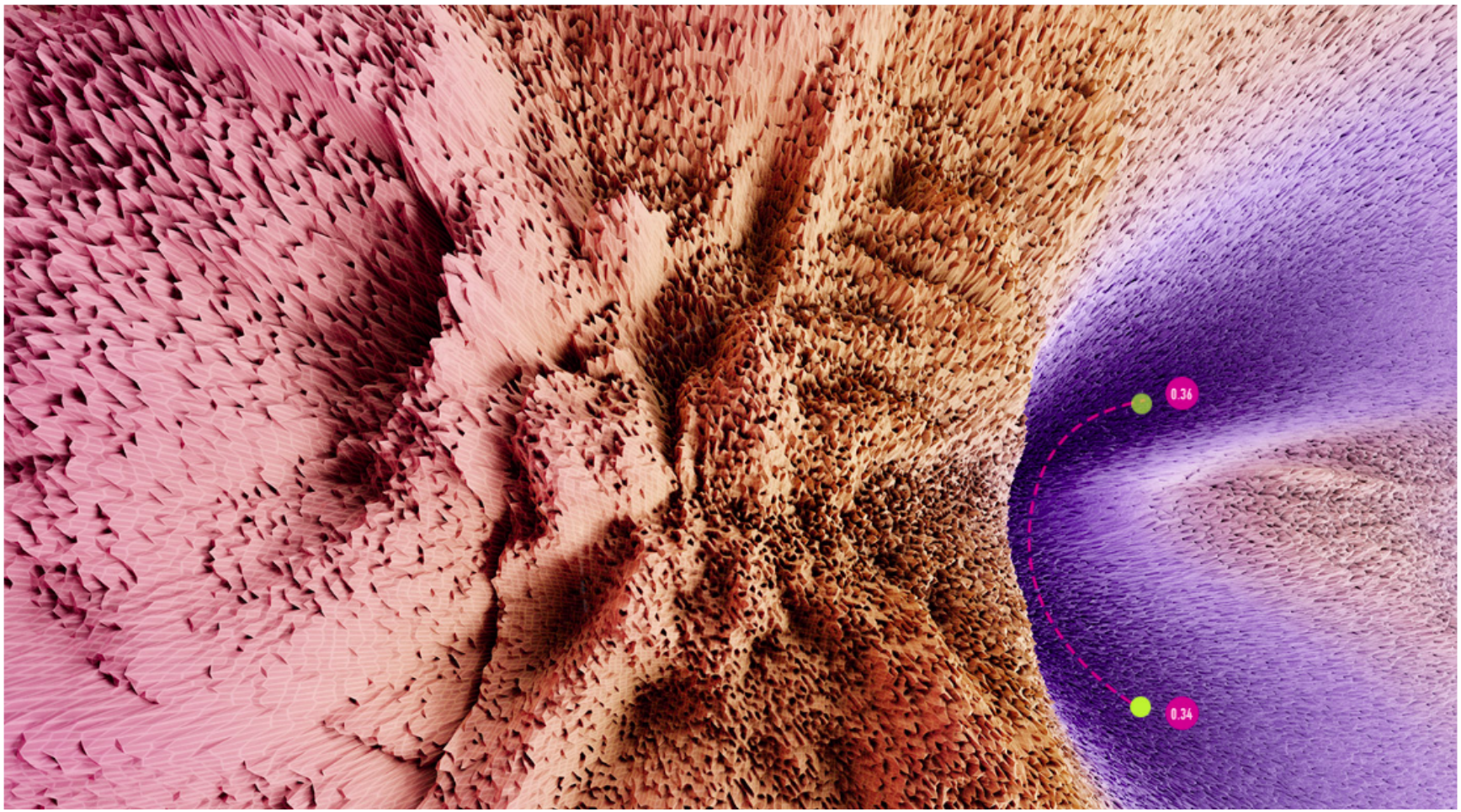
- Posterior geometries of NNs are complicated!
  1. NNs are highly non-linear
  2. Posteriors are highly multi-modal
  3. NNs have weight-space symmetries
  4. Often underdetermined by the data
- Bayesian inference in neural networks is generally a really *difficult open problem*





$$\begin{aligned} \boldsymbol{z}_1 &= h(\boldsymbol{W}_0 \boldsymbol{x} + \boldsymbol{b}_0) \\ \boldsymbol{z}_2 &= h(\boldsymbol{W}_1 \boldsymbol{z}_1 + \boldsymbol{b}_1) \\ f &= \boldsymbol{W}_2 \boldsymbol{z}_2 + \boldsymbol{b}_2 \end{aligned}$$

Posterior geometries of NNs are complicated!



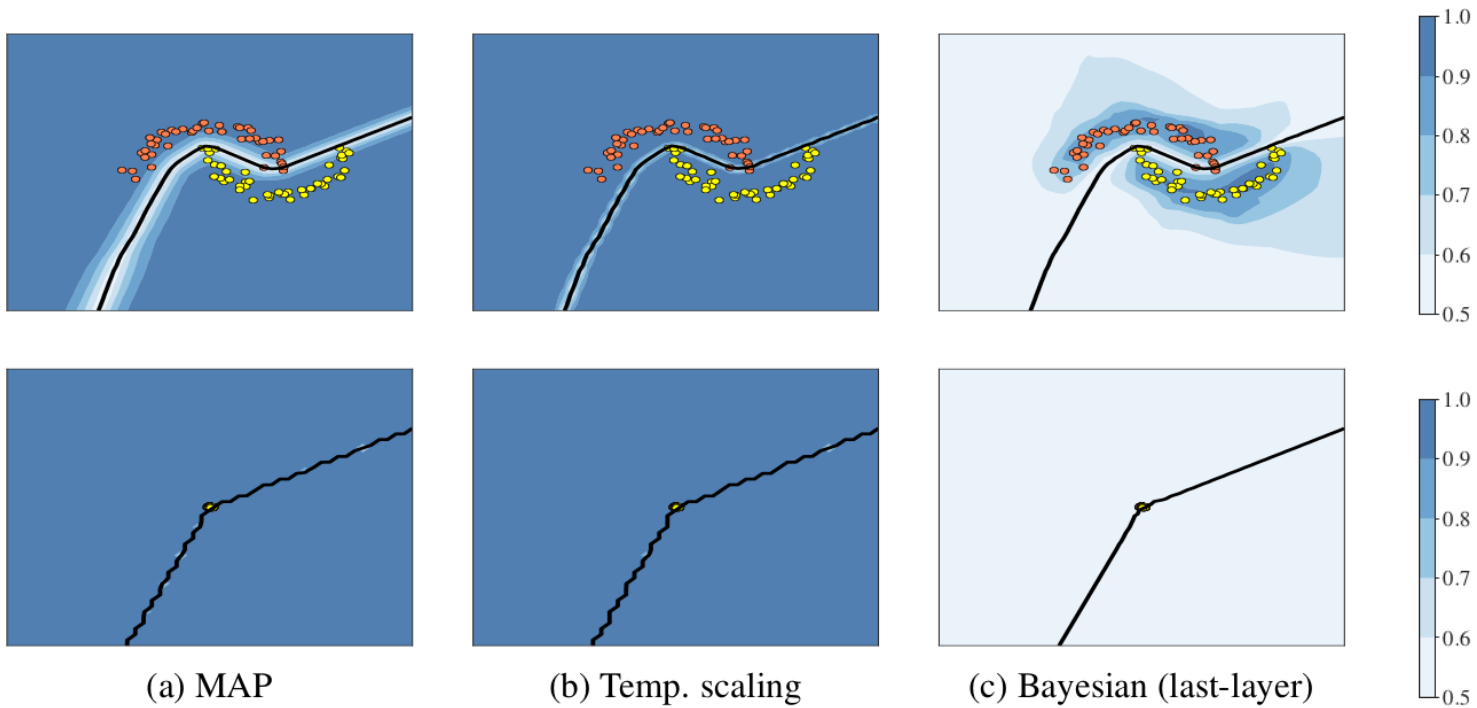
Visualization of loss surface for ResNet20

https://izmailovpavel.github.io/curves_blogpost/ (image credit)

Uncertainty and overconfident neural networks

- If it is so difficult, why do we bother?
- Classical neural networks are very often *overconfident*
- ReLU networks can be made *arbitrary confident* far away from training data, e.g.

$$\lim_{\delta \to \infty} \text{softmax}(y(\delta \boldsymbol{x}))_i = 1 \qquad \text{for some class } i$$



(a) MAP (b) Temp. scaling (c) Bayesian (last-layer)

Guo et al., 2017: On Calibration of Modern Neural Networks

Kristiadi et al., 2020: Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks (image credit)

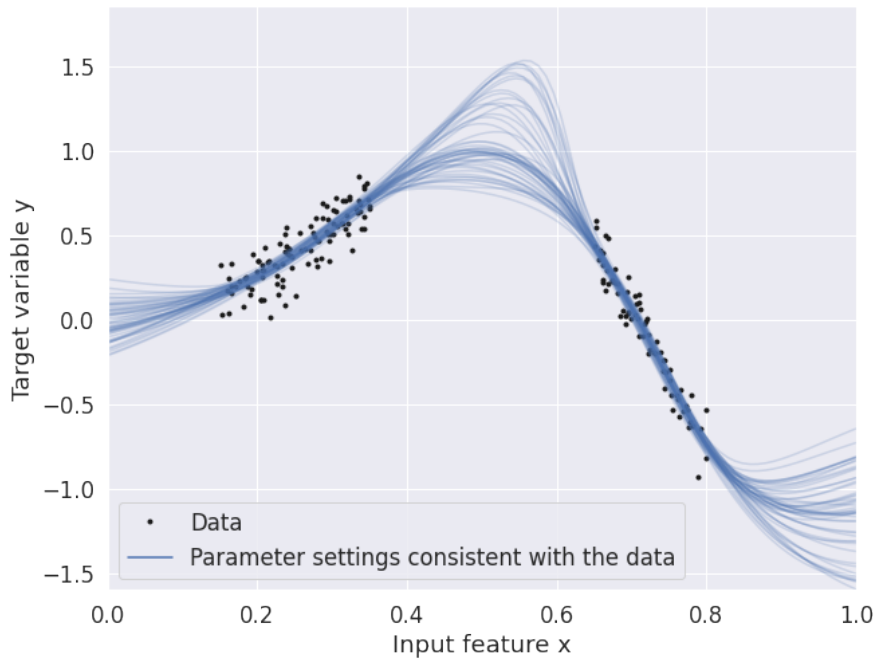
Model uncertainty due to finite data

- There exists many different parameter settings (for the same network), which lead to low training losses, but to very different predictions
- In classical training, we pick one set of weights (and one set of predictions)

$$\boldsymbol{w}_{\text{MAP}} = \arg\max_{\boldsymbol{w}} p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w})$$

- In the Bayesian world, we want to compute a weighted averaged over all of them

$$p(y^*|\boldsymbol{x}^*) = \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{y})\mathrm{d}\boldsymbol{w}$$

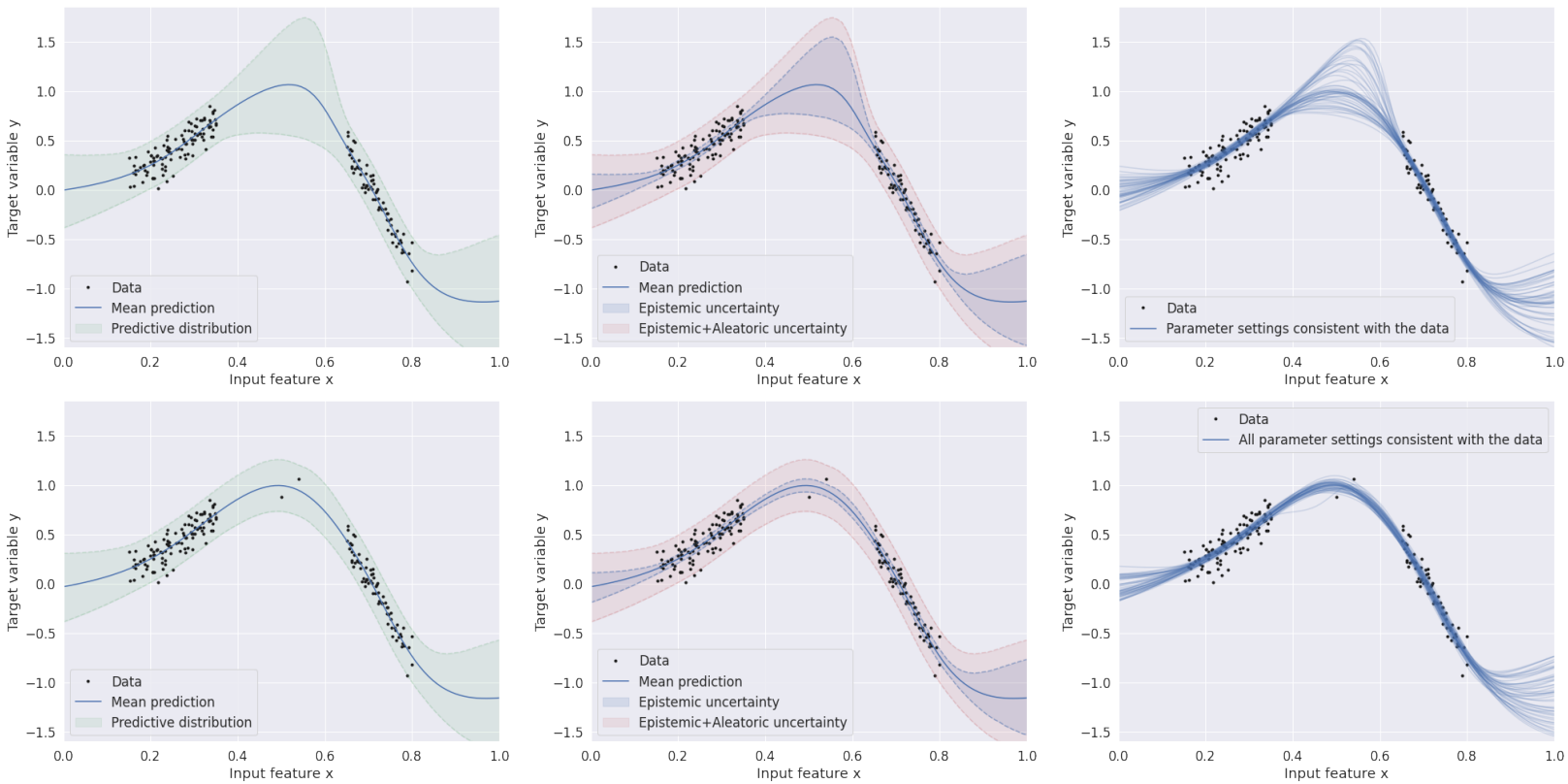


Uncertainty quantification

Two sources of uncertainty

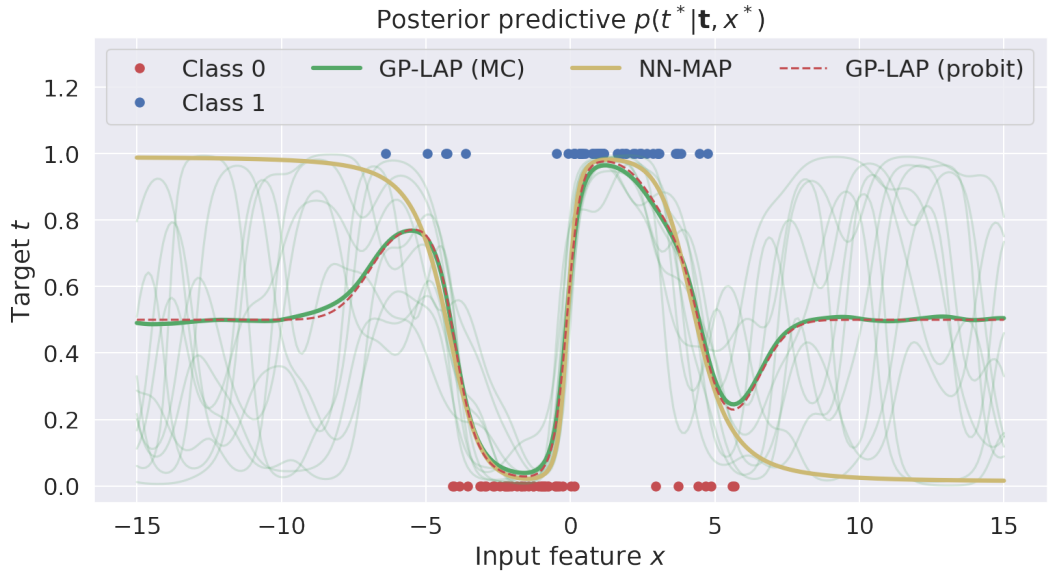
1. *Epistemic uncertainty* is due to lack of knowledge (e.g. often due to a limited data set). Also sometimes called the *reducible* uncertainty.

2. *Aleatoric uncertainty* refers to the inherent randomness (e.g. measurement noise). Also sometimes called the *irreducible* uncertainty.



- Classical training only captures aleatoric uncertainty, *not epistemic uncertainty*

DTU Learn quiz: Epistemic and aleatoric uncertainty for classification



DTU Learn - Quiz - Lecture 13: Aleatoric and epistemic uncertainty

Priors on neural network parameters

- Recall our study of linear models

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + \epsilon$$

- We imposed Gaussian priors on each $w_i \sim \mathcal{N}(w_i|0, \kappa^2)$

- We defined the prior in the *weight-space*, but it's easy to understand the effect in *function-space*

Priors on neural network parameters

- Recall our study of linear models

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + \epsilon$$

- We imposed Gaussian priors on each $w_i \sim \mathcal{N}(w_i|0, \kappa^2)$

- We defined the prior in the *weight-space*, but it's easy to understand the effect in *function-space*

- *Gaussian processes*: a prior directly on function-space – often easy to interpret parameters

Priors on neural network parameters

- Recall our study of linear models

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + \epsilon$$

- We imposed Gaussian priors on each $w_i \sim \mathcal{N}(w_i|0, \kappa^2)$
- We defined the prior in the *weight-space*, but it's easy to understand the effect in *function-space*
- *Gaussian processes*: a prior directly on function-space – often easy to interpret parameters
- It is common to impose Gaussian priors on the parameters of a NN, but it is *hard to interpret* what it means in function space.

Priors on neural network parameters

- Recall our study of linear models

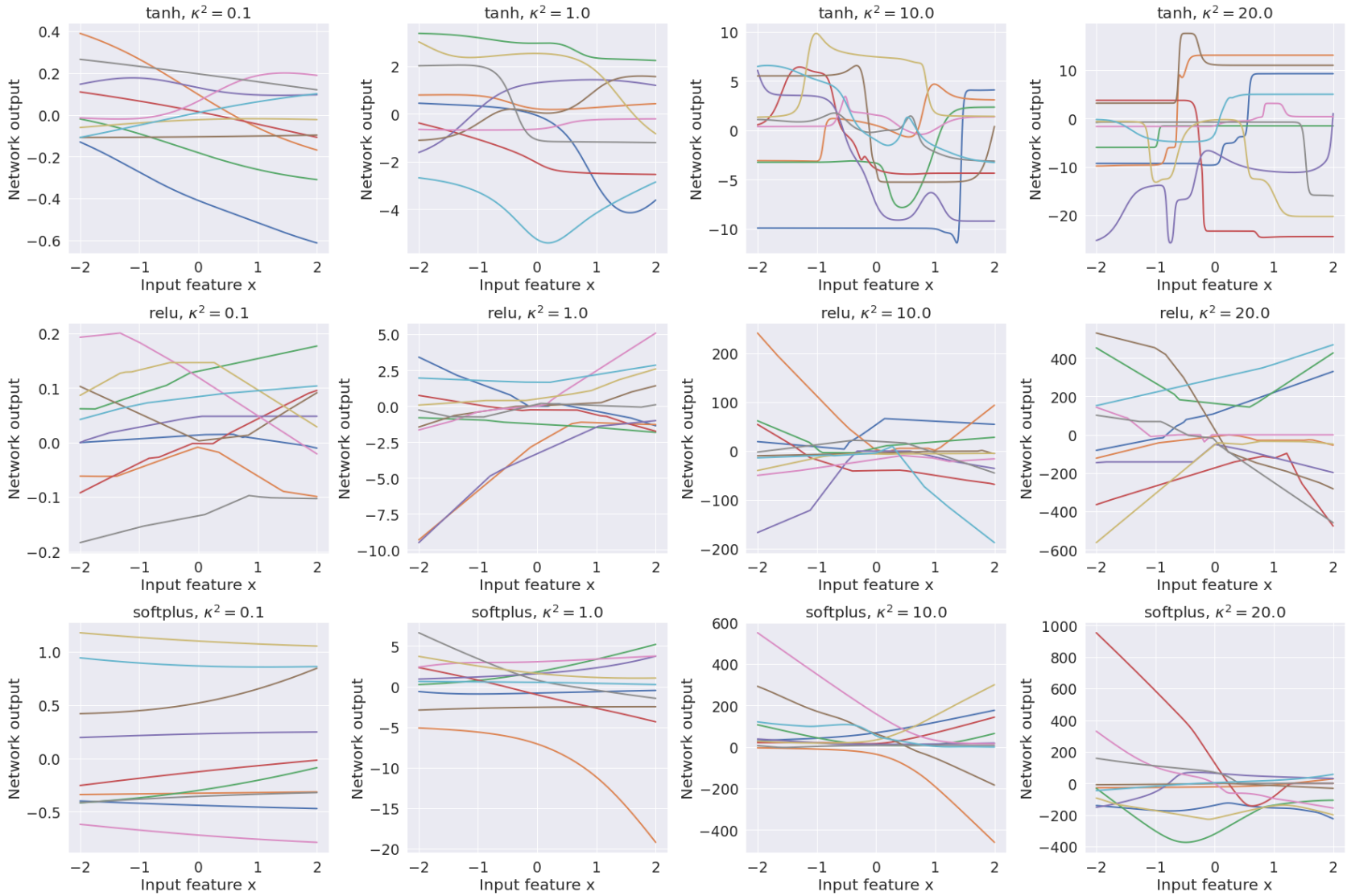
$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + \epsilon$$

- We imposed Gaussian priors on each $w_i \sim \mathcal{N}(w_i|0, \kappa^2)$
- We defined the prior in the *weight-space*, but it's easy to understand the effect in *function-space*
- *Gaussian processes*: a prior directly on function-space – often easy to interpret parameters
- It is common to impose Gaussian priors on the parameters of a NN, but it is *hard to interpret* what it means in function space.
- Let's generate some samples from the prior $\boldsymbol{w} \sim \mathcal{N}(0, \kappa^2 \boldsymbol{I})$ for $\boldsymbol{W} = \{\boldsymbol{W}_0, \boldsymbol{b}_0, \boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{W}_2, \boldsymbol{b}_2\}$

$$\begin{aligned} \boldsymbol{z}_1 &= h(\boldsymbol{W}_0 \boldsymbol{x} + \boldsymbol{b}_0) \\ z_2 &= h(\boldsymbol{W}_1 \boldsymbol{z}_1 + \boldsymbol{b}_1) \\ y &= \boldsymbol{W}_2 \boldsymbol{z}_2 + \boldsymbol{b}_2 \end{aligned}$$

Prior samples from Bayesian neural network

Gaussian priors on all parameters $\boldsymbol{w} \sim \mathcal{N}(0, \kappa^2 \mathbf{I})$, 2 layers with 5 units each



Bayesian Neural Networks: Approximating the posterior for neural networks

Approximating the posterior for neural networks

- All the standard tools have been used
  1. Laplace approximations
  2. Variational inference (BBVI)
  3. MCMC (Hamiltonian Monte Carlo)

- MCMC is the golden standard, but does not scale to interesting datasets

Approximating the posterior for neural networks

- All the standard tools have been used
  1. Laplace approximations
  2. Variational inference (BBVI)
  3. MCMC (Hamiltonian Monte Carlo)
- MCMC is the golden standard, but does not scale to interesting datasets
- Modern networks can have huge parameter spaces
  1. Image: ResNet-50 contains $\approx$ 23 million parameters
  2. Text: GPT-3 has 175 billion parameters

Approximating the posterior for neural networks

- All the standard tools have been used
  1. Laplace approximations
  2. Variational inference (BBVI)
  3. MCMC (Hamiltonian Monte Carlo)
- MCMC is the golden standard, but does not scale to interesting datasets
- Modern networks can have huge parameter spaces
  1. Image: ResNet-50 contains $\approx$ 23 million parameters
  2. Text: GPT-3 has 175 billion parameters
- Even the simplest mean-field approximation would contain twice as many variational parameters

Approximating the posterior for neural networks

- All the standard tools have been used
  1. Laplace approximations
  2. Variational inference (BBVI)
  3. MCMC (Hamiltonian Monte Carlo)
- MCMC is the golden standard, but does not scale to interesting datasets
- Modern networks can have huge parameter spaces
  1. Image: ResNet-50 contains $\approx$ 23 million parameters
  2. Text: GPT-3 has 175 billion parameters
- Even the simplest mean-field approximation would contain twice as many variational parameters
- Many approximate solutions have been proposed
  1. Monte Carlo Drop-out (McD)
  2. Last-layer Laplace approximations (LLLA)
  3. Deep ensembles
  4. Stochastic Weight Averaging Gaussian (SWAG)
  5. Last-layer Laplace + Variational inference with normalizing flows
  6. ...

Approximating the posterior for neural networks

- All the standard tools have been used
  1. Laplace approximations
  2. Variational inference (BBVI)
  3. MCMC (Hamiltonian Monte Carlo)
- MCMC is the golden standard, but does not scale to interesting datasets
- Modern networks can have huge parameter spaces
  1. Image: ResNet-50 contains $\approx$ 23 million parameters
  2. Text: GPT-3 has 175 billion parameters
- Even the simplest mean-field approximation would contain twice as many variational parameters
- Many approximate solutions have been proposed
  1. Monte Carlo Drop-out (McD)
  2. Last-layer Laplace approximations (LLLA)
  3. Deep ensembles
  4. Stochastic Weight Averaging Gaussian (SWAG)
  5. Last-layer Laplace + Variational inference with normalizing flows
  6. ...
- Large emphasis on *post-hoc* Bayesian approximations with little computational overhead *given a pretrained model* $\boldsymbol{w}_{\text{MAP}}$

Approximating the predictive distribution I

- Overall goal is to get the *posterior predictive distribution* by averaging wrt. the posterior distribution

$$p(y^*|\boldsymbol{y}) = \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{y})\mathrm{d}\boldsymbol{w} \approx \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})q(\boldsymbol{w})\mathrm{d}\boldsymbol{w}$$

Approximating the predictive distribution I

- Overall goal is to get the *posterior predictive distribution* by averaging wrt. the posterior distribution

$$p(y^*|\boldsymbol{y}) = \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{y})\mathrm{d}\boldsymbol{w} \approx \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})q(\boldsymbol{w})\mathrm{d}\boldsymbol{w}$$

- Given $q(\boldsymbol{w})$, the *posterior predictive* can be estimated using *Monte Carlo* sampling

$$p(y^*|\boldsymbol{y}) \approx \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})q(\boldsymbol{w})\mathrm{d}\boldsymbol{w} \approx \frac{1}{S}\sum_{i=1}^{S} p(y^*|\boldsymbol{x}^*, \boldsymbol{w}^{(i)}) \quad \text{for} \quad \boldsymbol{w}^{(i)} \sim q(\boldsymbol{w})$$

Approximating the predictive distribution I

- Overall goal is to get the *posterior predictive distribution* by averaging wrt. the posterior distribution

$$p(y^*|\boldsymbol{y}) = \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{y})\mathrm{d}\boldsymbol{w} \approx \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})q(\boldsymbol{w})\mathrm{d}\boldsymbol{w}$$

- Given $q(\boldsymbol{w})$, the *posterior predictive* can be estimated using *Monte Carlo* sampling

$$p(y^*|\boldsymbol{y}) \approx \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})q(\boldsymbol{w})\mathrm{d}\boldsymbol{w} \approx \frac{1}{S}\sum_{i=1}^{S} p(y^*|\boldsymbol{x}^*, \boldsymbol{w}^{(i)}) \quad \text{for} \quad \boldsymbol{w}^{(i)} \sim q(\boldsymbol{w})$$

- There are two common overall strategies to obtain "posterior samples" $\boldsymbol{w}^{(i)}$

  1. We do a proper posterior approximation $q(\boldsymbol{w})$ followed by sampling $\boldsymbol{w}^{(i)} \sim q(\boldsymbol{w})$

Approximating the predictive distribution I

- Overall goal is to get the *posterior predictive distribution* by averaging wrt. the posterior distribution

$$p(y^*|\boldsymbol{y}) = \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{y})\mathrm{d}\boldsymbol{w} \approx \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})q(\boldsymbol{w})\mathrm{d}\boldsymbol{w}$$

- Given $q(\boldsymbol{w})$, the *posterior predictive* can be estimated using *Monte Carlo* sampling

$$p(y^*|\boldsymbol{y}) \approx \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})q(\boldsymbol{w})\mathrm{d}\boldsymbol{w} \approx \frac{1}{S}\sum_{i=1}^{S} p(y^*|\boldsymbol{x}^*, \boldsymbol{w}^{(i)}) \quad \text{for} \quad \boldsymbol{w}^{(i)} \sim q(\boldsymbol{w})$$

- There are two common overall strategies to obtain "posterior samples" $\boldsymbol{w}^{(i)}$

  1. We do a proper posterior approximation $q(\boldsymbol{w})$ followed by sampling $\boldsymbol{w}^{(i)} \sim q(\boldsymbol{w})$

Approximating the predictive distribution I

- Overall goal is to get the *posterior predictive distribution* by averaging wrt. the posterior distribution

$$p(y^*|\boldsymbol{y}) = \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{y})\mathrm{d}\boldsymbol{w} \approx \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})q(\boldsymbol{w})\mathrm{d}\boldsymbol{w}$$

- Given $q(\boldsymbol{w})$, the *posterior predictive* can be estimated using *Monte Carlo* sampling

$$p(y^*|\boldsymbol{y}) \approx \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})q(\boldsymbol{w})\mathrm{d}\boldsymbol{w} \approx \frac{1}{S}\sum_{i=1}^{S} p(y^*|\boldsymbol{x}^*, \boldsymbol{w}^{(i)}) \quad \text{for} \quad \boldsymbol{w}^{(i)} \sim q(\boldsymbol{w})$$

- There are two common overall strategies to obtain "posterior samples" $\boldsymbol{w}^{(i)}$

  1. We do a proper posterior approximation $q(\boldsymbol{w})$ followed by sampling $\boldsymbol{w}^{(i)} \sim q(\boldsymbol{w})$

  2. We collect a set of samples $\left\{\boldsymbol{w}^{(i)}\right\}_{i=1}^{S}$ directly without having an actual distribution $q(\boldsymbol{w})$

Approximating the predictive distribution I

- Overall goal is to get the *posterior predictive distribution* by averaging wrt. the posterior distribution

$$p(y^*|\boldsymbol{y}) = \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{y})\mathrm{d}\boldsymbol{w} \approx \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})q(\boldsymbol{w})\mathrm{d}\boldsymbol{w}$$

- Given $q(\boldsymbol{w})$, the *posterior predictive* can be estimated using *Monte Carlo* sampling

$$p(y^*|\boldsymbol{y}) \approx \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})q(\boldsymbol{w})\mathrm{d}\boldsymbol{w} \approx \frac{1}{S}\sum_{i=1}^{S} p(y^*|\boldsymbol{x}^*, \boldsymbol{w}^{(i)}) \quad \text{for} \quad \boldsymbol{w}^{(i)} \sim q(\boldsymbol{w})$$

- There are two common overall strategies to obtain "posterior samples" $\boldsymbol{w}^{(i)}$

  1. We do a proper posterior approximation $q(\boldsymbol{w})$ followed by sampling $\boldsymbol{w}^{(i)} \sim q(\boldsymbol{w})$

     For instance, variational inference, Laplace approximations, ...

  2. We collect a set of samples $\left\{\boldsymbol{w}^{(i)}\right\}_{i=1}^{S}$ directly without having an actual distribution $q(\boldsymbol{w})$

Approximating the predictive distribution I

- Overall goal is to get the *posterior predictive distribution* by averaging wrt. the posterior distribution

$$p(y^*|\boldsymbol{y}) = \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{y})\mathrm{d}\boldsymbol{w} \approx \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})q(\boldsymbol{w})\mathrm{d}\boldsymbol{w}$$

- Given $q(\boldsymbol{w})$, the *posterior predictive* can be estimated using *Monte Carlo* sampling

$$p(y^*|\boldsymbol{y}) \approx \int p(y^*|\boldsymbol{x}^*, \boldsymbol{w})q(\boldsymbol{w})\mathrm{d}\boldsymbol{w} \approx \frac{1}{S}\sum_{i=1}^{S} p(y^*|\boldsymbol{x}^*, \boldsymbol{w}^{(i)}) \quad \text{for} \quad \boldsymbol{w}^{(i)} \sim q(\boldsymbol{w})$$

- There are two common overall strategies to obtain "posterior samples" $\boldsymbol{w}^{(i)}$

  1. We do a proper posterior approximation $q(\boldsymbol{w})$ followed by sampling $\boldsymbol{w}^{(i)} \sim q(\boldsymbol{w})$

     For instance, variational inference, Laplace approximations, ...

  2. We collect a set of samples $\left\{\boldsymbol{w}^{(i)}\right\}_{i=1}^{S}$ directly without having an actual distribution $q(\boldsymbol{w})$

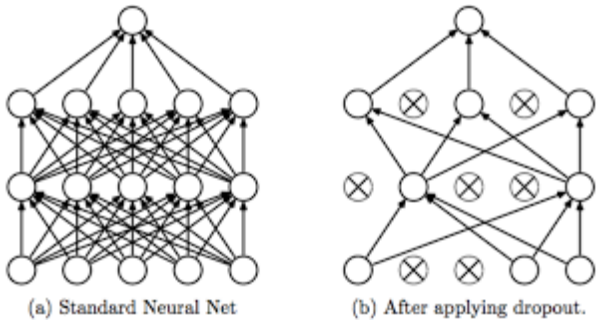
     For instance, SGD as approximate inference, SWAG, deep ensembles, MAP-inference, ...

Bayesian Neural Networks: Monte Carlo Dropout

Monte Carlo Dropout I

- *Dropout is a regularization technique for training NNs*: During training, replace input for each layer with a product of a random variable, e.g. $\boldsymbol{W}_0\boldsymbol{x}$ is replaced with

$$\boldsymbol{W}_0(\boldsymbol{x} \circ \boldsymbol{s}), \qquad \text{where} \quad s \sim p(s)$$



(a) Standard Neural Net (b) After applying dropout.

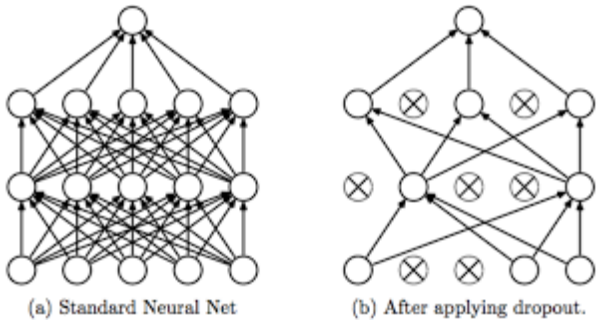
Gal & Ghahramani, 2015: Dropout as a Bayesian Approximation (5k+ citations!)

Monte Carlo Dropout I

- *Dropout is a regularization technique for training NNs*: During training, replace input for each layer with a product of a random variable, e.g. $\boldsymbol{W}_0\boldsymbol{x}$ is replaced with

$$\boldsymbol{W}_0(\boldsymbol{x} \circ \boldsymbol{s}), \qquad \text{where} \quad s \sim p(s)$$

- For example, $s \sim \text{Ber}(p)$, where $1 - p$ is the drop out rate *ignores* a random subset of the input in each iteration



(a) Standard Neural Net (b) After applying dropout.

Gal & Ghahramani, 2015: Dropout as a Bayesian Approximation (5k+ citations!)

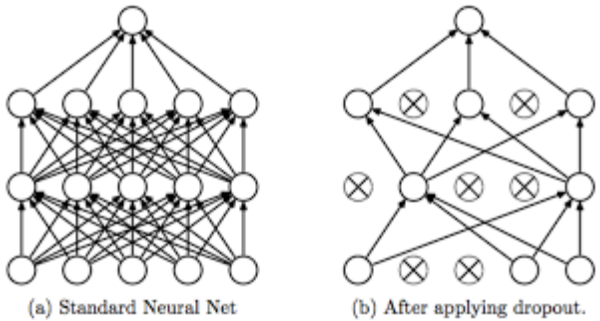
Monte Carlo Dropout I

- *Dropout is a regularization technique for training NNs*: During training, replace input for each layer with a product of a random variable, e.g. $\boldsymbol{W}_0\boldsymbol{x}$ is replaced with

$$\boldsymbol{W}_0(\boldsymbol{x} \circ \boldsymbol{s}), \qquad \text{where} \quad s \sim p(s)$$

- For example, $s \sim \text{Ber}(p)$, where $1 - p$ is the drop out rate *ignores* a random subset of the input in each iteration

- Dropout as regularization can be shown to be equivalent to a special case of ridge regression (ℓ_2-penalty)



(a) Standard Neural Net (b) After applying dropout.

Gal & Ghahramani, 2015: Dropout as a Bayesian Approximation (5k+ citations!)

Monte Carlo Dropout I

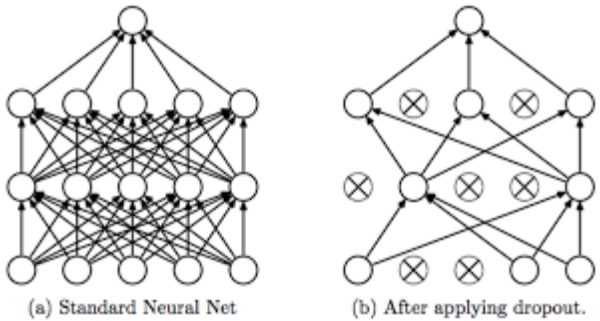
- *Dropout is a regularization technique for training NNs*: During training, replace input for each layer with a product of a random variable, e.g. $\boldsymbol{W}_0\boldsymbol{x}$ is replaced with

$$\boldsymbol{W}_0(\boldsymbol{x} \circ \boldsymbol{s}), \qquad \text{where} \quad s \sim p(s)$$

- For example, $s \sim \text{Ber}(p)$, where $1 - p$ is the drop out rate *ignores* a random subset of the input in each iteration

- Dropout as regularization can be shown to be equivalent to a special case of ridge regression (ℓ_2-penalty)

- *Monte Carlo Dropout*: Perform drop-out at test time and average over results



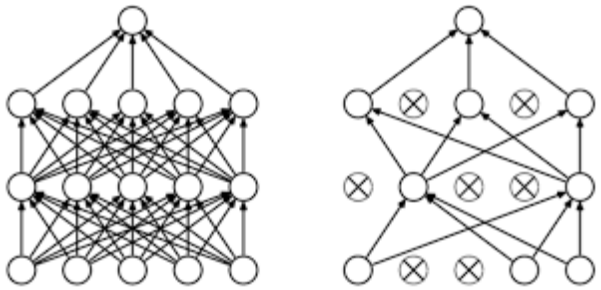
(a) Standard Neural Net (b) After applying dropout.

Gal & Ghahramani, 2015: Dropout as a Bayesian Approximation (5k+ citations!)

Monte Carlo Dropout I

- *Dropout is a regularization technique for training NNs*: During training, replace input for each layer with a product of a random variable, e.g. $\boldsymbol{W}_0\boldsymbol{x}$ is replaced with

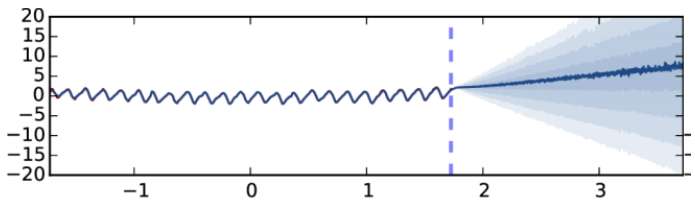
$$\boldsymbol{W}_0(\boldsymbol{x} \circ \boldsymbol{s}), \qquad \text{where} \quad s \sim p(s)$$



(a) Standard Neural Net (b) After applying dropout.

- For example, $s \sim \text{Ber}(p)$, where $1 - p$ is the drop out rate *ignores* a random subset of the input in each iteration

- Dropout as regularization can be shown to be equivalent to a special case of ridge regression (ℓ_2-penalty)



(c) MC dropout with ReLU non-linearities

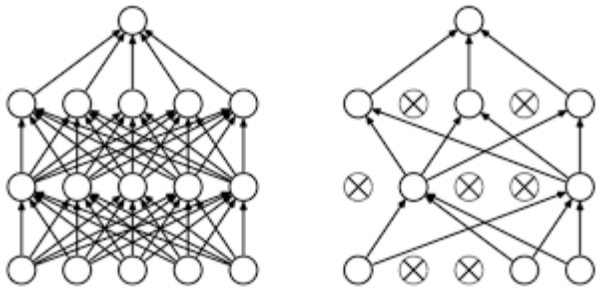
- *Monte Carlo Dropout*: Perform drop-out at test time and average over results

Gal & Ghahramani, 2015: Dropout as a Bayesian Approximation (5k+ citations!)

Monte Carlo Dropout I

- *Dropout is a regularization technique for training NNs*: During training, replace input for each layer with a product of a random variable, e.g. $\boldsymbol{W}_0\boldsymbol{x}$ is replaced with

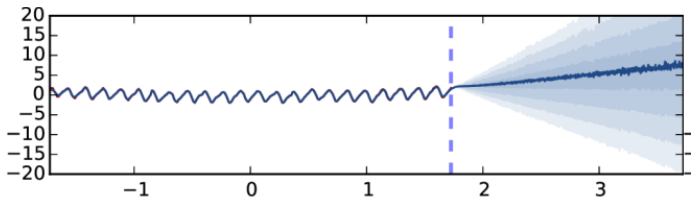
$$\boldsymbol{W}_0(\boldsymbol{x} \circ \boldsymbol{s}), \qquad \text{where} \quad s \sim p(s)$$



(a) Standard Neural Net (b) After applying dropout.

- For example, $s \sim \text{Ber}(p)$, where $1 - p$ is the drop out rate *ignores* a random subset of the input in each iteration

- Dropout as regularization can be shown to be equivalent to a special case of ridge regression (ℓ_2-penalty)



(c) MC dropout with ReLU non-linearities

- *Monte Carlo Dropout*: Perform drop-out at test time and average over results

- Very popular, very easy to implement, but not really justified in anyway

Gal & Ghahramani, 2015: Dropout as a Bayesian Approximation (5k+ citations!)

Monte Carlo Dropout II

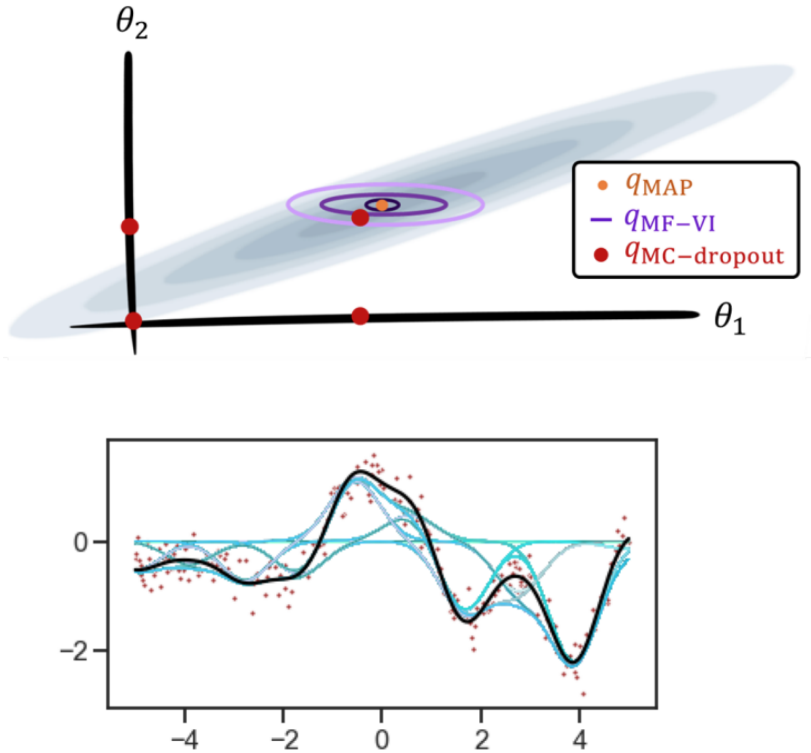


Fig. 3. MC-dropout posterior on 1-layer RBF regression. Black line: true model. Brown dots: observations. Blue lines are not samples from the approximate posterior, but the set of 1024 delta-Dirac functions forming the multimodal MC-dropout posterior. Evidently the location of the modes is unrelated to the model uncertainty and is instead simply an artefact of the MC-dropout approximation.

- Epistemic uncertainty does not decrease as size of dataset increases
- Yet, there are many papers claiming to benefit from MC Dropout.

Folgoc et al, 2021: Is MC Dropout Bayesian?

Bayesian Neural Networks: Mean-field

Mean-field for BNNs I

What about mean-field for approximating $q(\boldsymbol{w})$?

- Mean-field approximations can be tricky for BNNs: $q(\boldsymbol{w}) = \prod_i \mathcal{N}(w_i | m_i, v_i)$

Coker et al., 2021: Wide Mean-Field Variational Bayesian Neural Networks Ignore the Data

Farquhar etal, 2020: Liberty or Depth: Deep Bayesian Neural Nets Do Not Need Complex Weight Posterior Approximations

Mean-field for BNNs I

What about mean-field for approximating $q(\boldsymbol{w})$?

- Mean-field approximations can be tricky for BNNs: $q(\boldsymbol{w}) = \prod_i \mathcal{N}(w_i | m_i, v_i)$
- Helps to initialize m_i using $(\mathbf{w}_{\text{MAP}})_i$ and initailize v_i to small values, i.e. $v_i = 10^{-6}$

Coker et al., 2021: Wide Mean-Field Variational Bayesian Neural Networks Ignore the Data

Farquhar etal, 2020: Liberty or Depth: Deep Bayesian Neural Nets Do Not Need Complex Weight Posterior Approximations

Mean-field for BNNs I

What about mean-field for approximating $q(\boldsymbol{w})$?

- Mean-field approximations can be tricky for BNNs: $q(\boldsymbol{w}) = \prod_i \mathcal{N}(w_i | m_i, v_i)$
- Helps to initialize m_i using $(\mathbf{w}_{\text{MAP}})_i$ and initailize v_i to small values, i.e. $v_i = 10^{-6}$
- New research and theory on MFVI for BNNs show

Coker et al., 2021: Wide Mean-Field Variational Bayesian Neural Networks Ignore the Data

Farquhar etal, 2020: Liberty or Depth: Deep Bayesian Neural Nets Do Not Need Complex Weight Posterior Approximations

Mean-field for BNNs I

What about mean-field for approximating $q(\boldsymbol{w})$?

- Mean-field approximations can be tricky for BNNs: $q(\boldsymbol{w}) = \prod_i \mathcal{N}(w_i | m_i, v_i)$
- Helps to initialize m_i using $(\mathbf{w}_{\text{MAP}})_i$ and initailize v_i to small values, i.e. $v_i = 10^{-6}$
- New research and theory on MFVI for BNNs show
  1. For a BNN with a *single* hidden layer: if we make the network wider and wider, eventually the posterior approximation will converge to the *prior distribution*. That is, the model will eventually *completely ignore the data*.

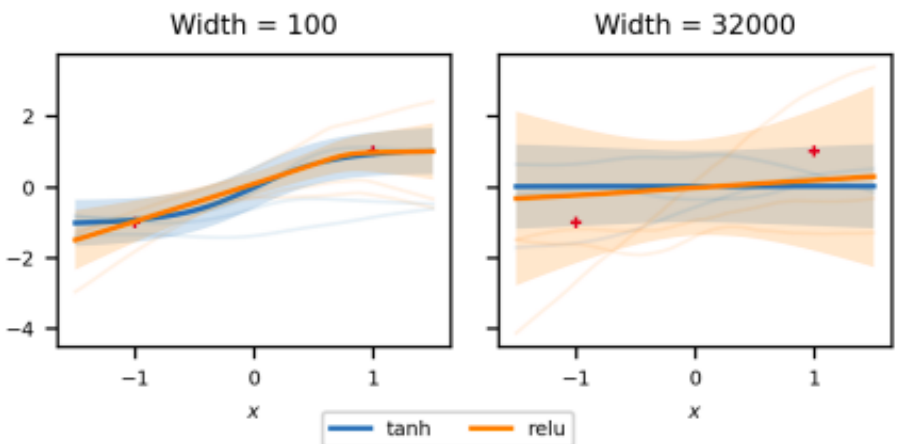
Coker et al., 2021: Wide Mean-Field Variational Bayesian Neural Networks Ignore the Data

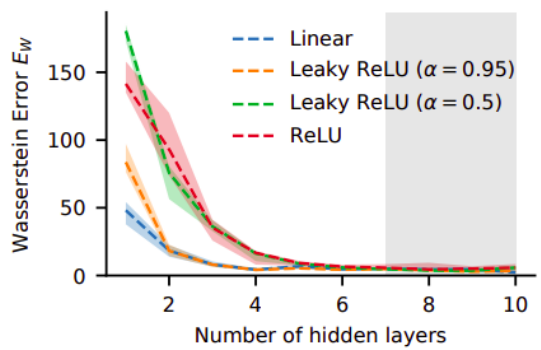
Farquhar etal, 2020: Liberty or Depth: Deep Bayesian Neural Nets Do Not Need Complex Weight Posterior Approximations

Mean-field for BNNs I

What about mean-field for approximating $q(\boldsymbol{w})$?

- Mean-field approximations can be tricky for BNNs: $q(\boldsymbol{w}) = \prod_i \mathcal{N}(w_i | m_i, v_i)$
- Helps to initialize m_i using $(\mathbf{w}_{\text{MAP}})_i$ and initailize v_i to small values, i.e. $v_i = 10^{-6}$
- New research and theory on MFVI for BNNs show
  1. For a BNN with a *single* hidden layer: if we make the network wider and wider, eventually the posterior approximation will converge to the *prior distribution*. That is, the model will eventually *completely ignore the data*.





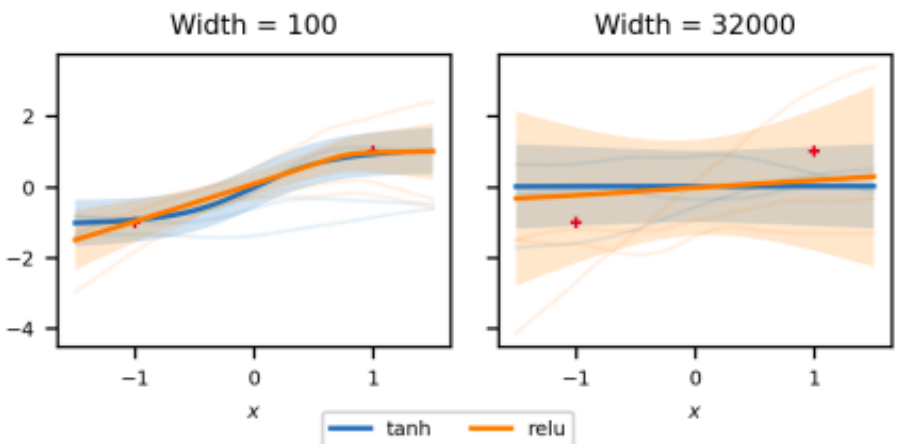
Coker et al., 2021: Wide Mean-Field Variational Bayesian Neural Networks Ignore the Data

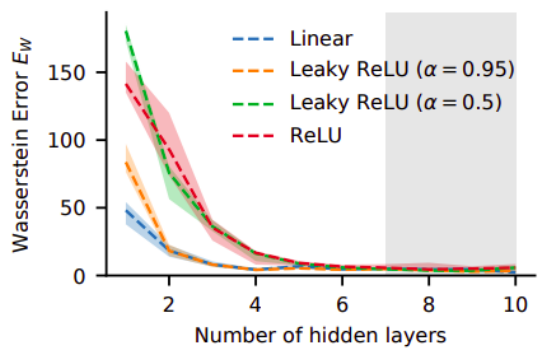
Farquhar etal, 2020: Liberty or Depth: Deep Bayesian Neural Nets Do Not Need Complex Weight Posterior Approximations

Mean-field for BNNs I

What about mean-field for approximating $q(\boldsymbol{w})$?

- Mean-field approximations can be tricky for BNNs: $q(\boldsymbol{w}) = \prod_i \mathcal{N}(w_i | m_i, v_i)$
- Helps to initialize m_i using $(\mathbf{w}_{\text{MAP}})_i$ and initailize v_i to small values, i.e. $v_i = 10^{-6}$
- New research and theory on MFVI for BNNs show
  1. For a BNN with a *single* hidden layer: if we make the network wider and wider, eventually the posterior approximation will converge to the *prior distribution*. That is, the model will eventually *completely ignore the data*.
  2. As networks becomes deeper and deeper, the posterior distribution will become *easier* to approximate with a mean-field approximation





Coker et al., 2021: Wide Mean-Field Variational Bayesian Neural Networks Ignore the Data

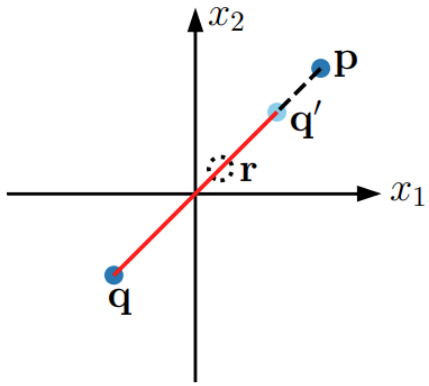
Farquhar etal, 2020: Liberty or Depth: Deep Bayesian Neural Nets Do Not Need Complex Weight Posterior Approximations

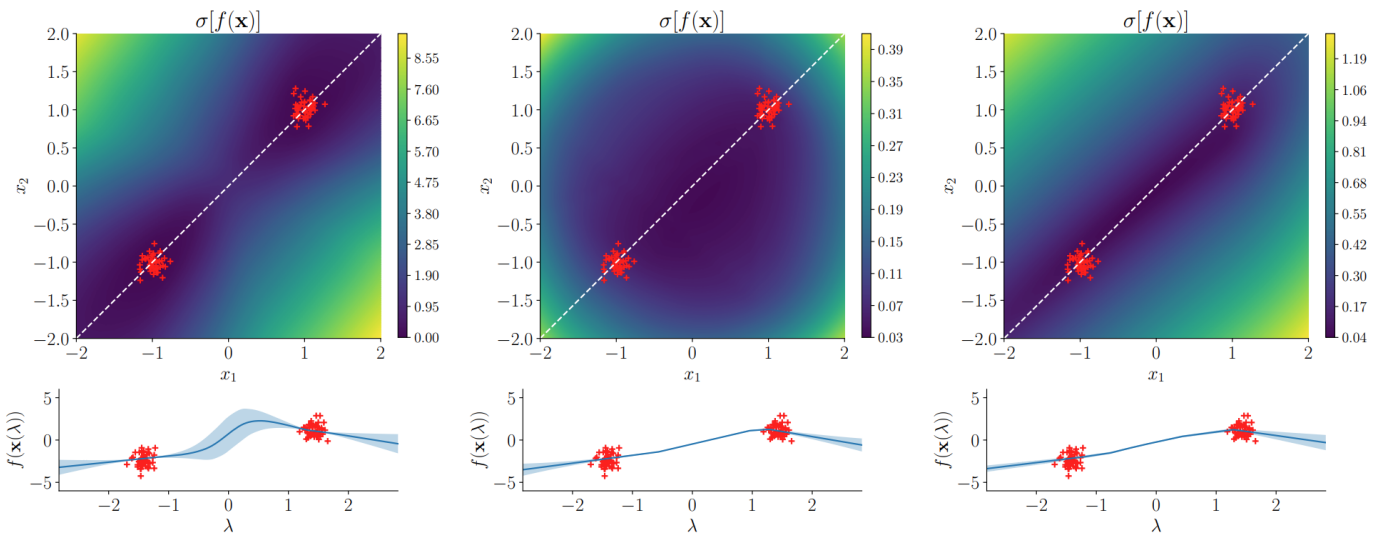
Mean-field for BNNs II

- For single-layer networks: if $\boldsymbol{r} = (1-\lambda)\boldsymbol{p} + \lambda\boldsymbol{q}$ for $\lambda \in [0,1]$, it has been proven that

$$\mathbb{V}\left[y(\boldsymbol{r})\right] \leq \mathbb{V}\left[y(\boldsymbol{p})\right] + \mathbb{V}\left[y(\boldsymbol{q})\right]$$

- Note: Figure from paper represents NN with f rather than y





(b) HMC (c) MFVI (d) MCDO

Foong et al, 2020: On the Expressiveness of Approximate Inference in Bayesian Neural Networks

Bayesian Neural Networks: Laplace approximations

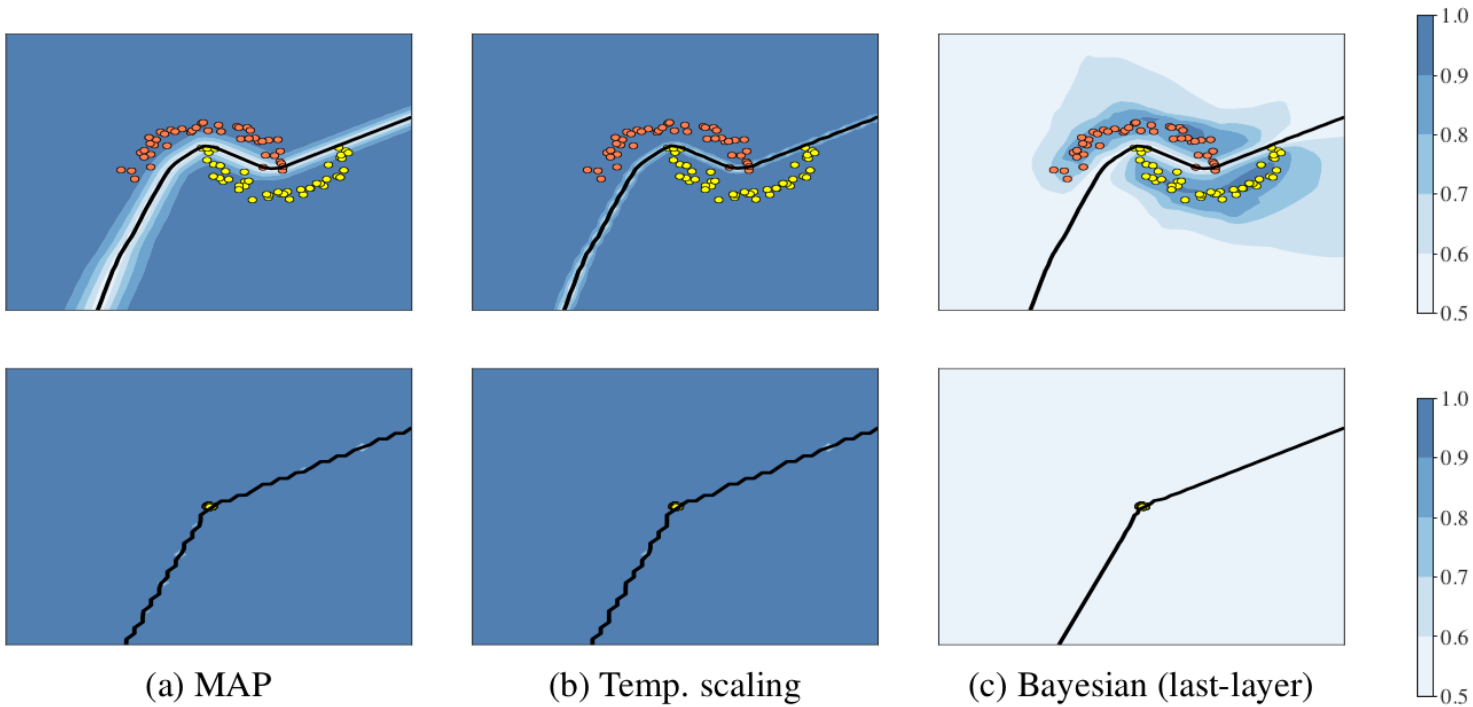
Laplace approximations for BNNs

- Recall the Laplace approximation

$$p(\boldsymbol{w}|\boldsymbol{y}) \approx q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{w}_{\text{MAP}}, -\boldsymbol{A}^{-1})$$

where

$$\boldsymbol{A} = -\nabla\nabla \ln p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w})\big|_{\boldsymbol{w}=\boldsymbol{w}_{\text{MAP}}}$$



(a) MAP (b) Temp. scaling (c) Bayesian (last-layer)

Kristiadi et al., 2020: Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks (image credit)

Laplace approximations for BNNs

- Recall the Laplace approximation

$$p(\boldsymbol{w}|\boldsymbol{y}) \approx q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{w}_{\text{MAP}}, -\boldsymbol{A}^{-1})$$

  where

$$\boldsymbol{A} = -\nabla\nabla \ln p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w})\big|_{\boldsymbol{w}=\boldsymbol{w}_{\text{MAP}}}$$

- Many nice properties
  1. Can be applied post-hoc given a pretrained model $\boldsymbol{w}_{\text{MAP}}$
  2. For many likelihoods (e.g. binary classification), it can be shown to give the same *decisions*, but are often better calibrated
  3. Simple, intuitive and relatively easy to implement

Kristiadi et al., 2020: Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks (image credit)

Laplace approximations for BNNs

- Recall the Laplace approximation

$$p(\boldsymbol{w}|\boldsymbol{y}) \approx q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{w}_{\text{MAP}}, -\boldsymbol{A}^{-1})$$

  where

$$\boldsymbol{A} = -\nabla\nabla \ln p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w})\big|_{\boldsymbol{w}=\boldsymbol{w}_{\text{MAP}}}$$

- Many nice properties
  1. Can be applied post-hoc given a pretrained model $\boldsymbol{w}_{\text{MAP}}$
  2. For many likelihoods (e.g. binary classification), it can be shown to give the same *decisions*, but are often better calibrated
  3. Simple, intuitive and relatively easy to implement
- But what is the size of the Hessian for at neural network with 10 million parameters?

Kristiadi et al., 2020: Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks (image credit)

Laplace approximations for BNNs

- Recall the Laplace approximation

$$p(\boldsymbol{w}|\boldsymbol{y}) \approx q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{w}_{\text{MAP}}, -\boldsymbol{A}^{-1})$$

  where

$$\boldsymbol{A} = -\nabla\nabla \ln p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w})\big|_{\boldsymbol{w}=\boldsymbol{w}_{\text{MAP}}}$$

- Many nice properties
  1. Can be applied post-hoc given a pretrained model $\boldsymbol{w}_{\text{MAP}}$
  2. For many likelihoods (e.g. binary classification), it can be shown to give the same *decisions*, but are often better calibrated
  3. Simple, intuitive and relatively easy to implement
- But what is the size of the Hessian for at neural network with 10 million parameters?

$$\boldsymbol{A} \in \mathbb{R}^{10^7 \times 10^7}$$

Kristiadi et al., 2020: Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks (image credit)

Laplace approximations for BNNs

- Recall the Laplace approximation

$$p(\boldsymbol{w}|\boldsymbol{y}) \approx q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{w}_{\text{MAP}}, -\boldsymbol{A}^{-1})$$

  where

$$\boldsymbol{A} = -\nabla\nabla \ln p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w})\big|_{\boldsymbol{w}=\boldsymbol{w}_{\text{MAP}}}$$

- Many nice properties
  1. Can be applied post-hoc given a pretrained model $\boldsymbol{w}_{\text{MAP}}$
  2. For many likelihoods (e.g. binary classification), it can be shown to give the same *decisions*, but are often better calibrated
  3. Simple, intuitive and relatively easy to implement
- But what is the size of the Hessian for at neural network with 10 million parameters?

$$\boldsymbol{A} \in \mathbb{R}^{10^7 \times 10^7}$$

- It would require approximately 728 TB to store in memory (64 bit precision)

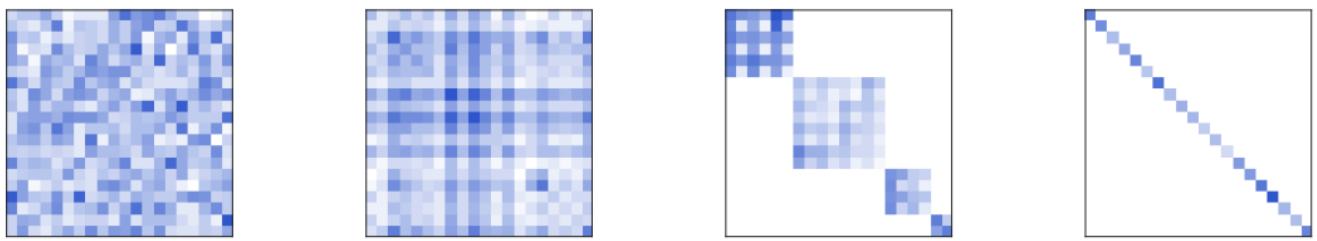
Kristiadi et al., 2020: Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks (image credit)

Approximating the Hessian

- The (negative) Hessian is given by

$$\boldsymbol{A} = -\nabla\nabla \ln p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w})\big|_{\boldsymbol{w}=\boldsymbol{w}_{\text{MAP}}} = -\underbrace{\nabla\nabla \ln p(\boldsymbol{y}|\boldsymbol{w})}_{\text{Hessian of log lik.}}\big|_{\boldsymbol{w}=\boldsymbol{w}_{\text{MAP}}} - \underbrace{\nabla\nabla \ln p(\boldsymbol{w})}_{\text{Hessian of log prior}}\big|_{\boldsymbol{w}=\boldsymbol{w}_{\text{MAP}}}$$

- For a Gaussian prior, the Hessian is easy: $\frac{\alpha}{2}\boldsymbol{I}$
- Several approximation proposed for the Hessian of the log likelihood
  1. Full (no approximation)
  2. Low rank
  3. Kronecker-factored approximate curvature (KFAC)
  4. Diagonal



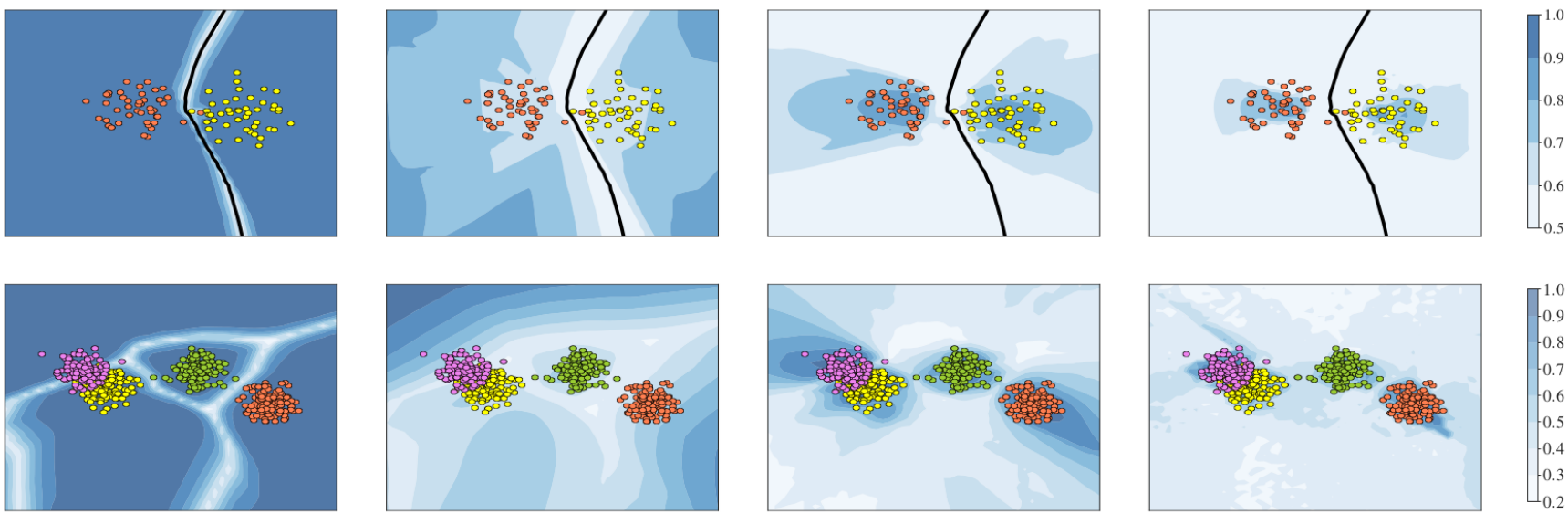
(a) Full **(b)** LRank **(c)** KFAC **(d)** Diag.

- Hessians are not guaranteed so be positive definite, so sometimes the generalized Gauss-Newton matrix is used instead

Daxberger et al, 2022: Laplace Redux – Effortless Bayesian Deep Learning (image credit)

Last-layer Laplace approximations (LLLA) I

- Laplace approximations for the last layer only is often a very good alternative

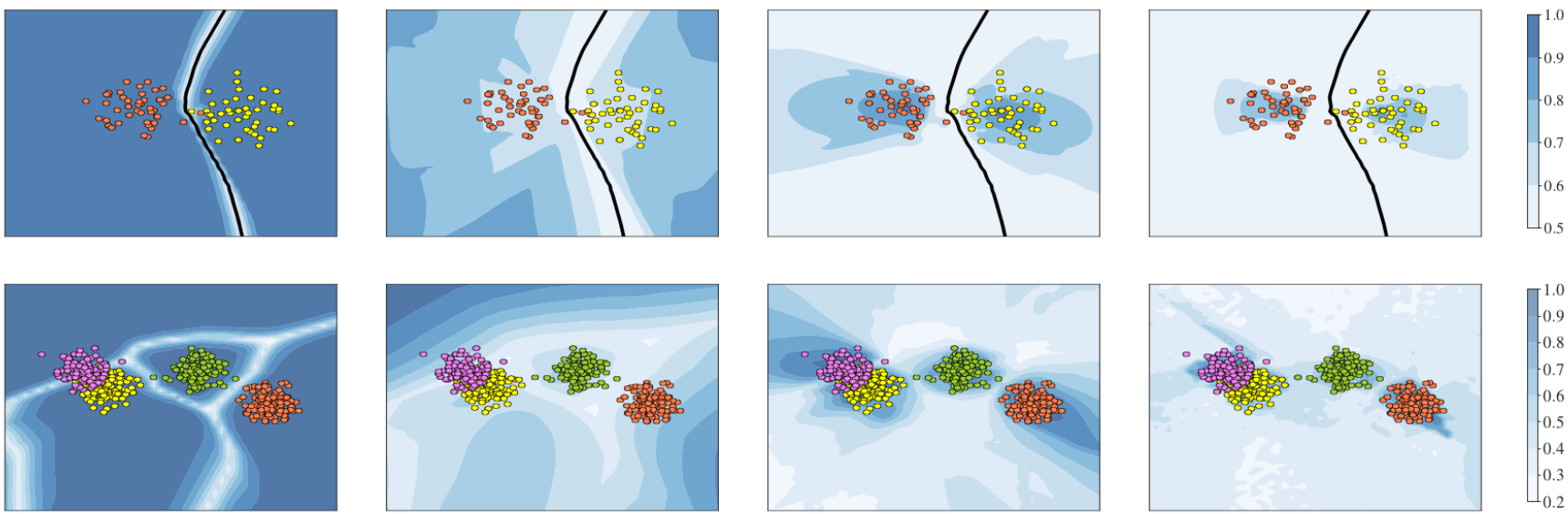


(a) MAP (b) Temp. scaling (c) LLLA (d) Full Laplace

Kristiadi et al., 2020: Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks (image credit)

Last-layer Laplace approximations (LLLA) I

- Laplace approximations for the last layer only is often a very good alternative
- ResNet18 to be fine-tuned to CIFAR10 (image dataset with 10 classes)
  1. Full network: contains $\approx 11 \cdot 10^6$ parameters in total
  2. Last fully connected layer: contains $\approx 5 \cdot 10^3$ parameters

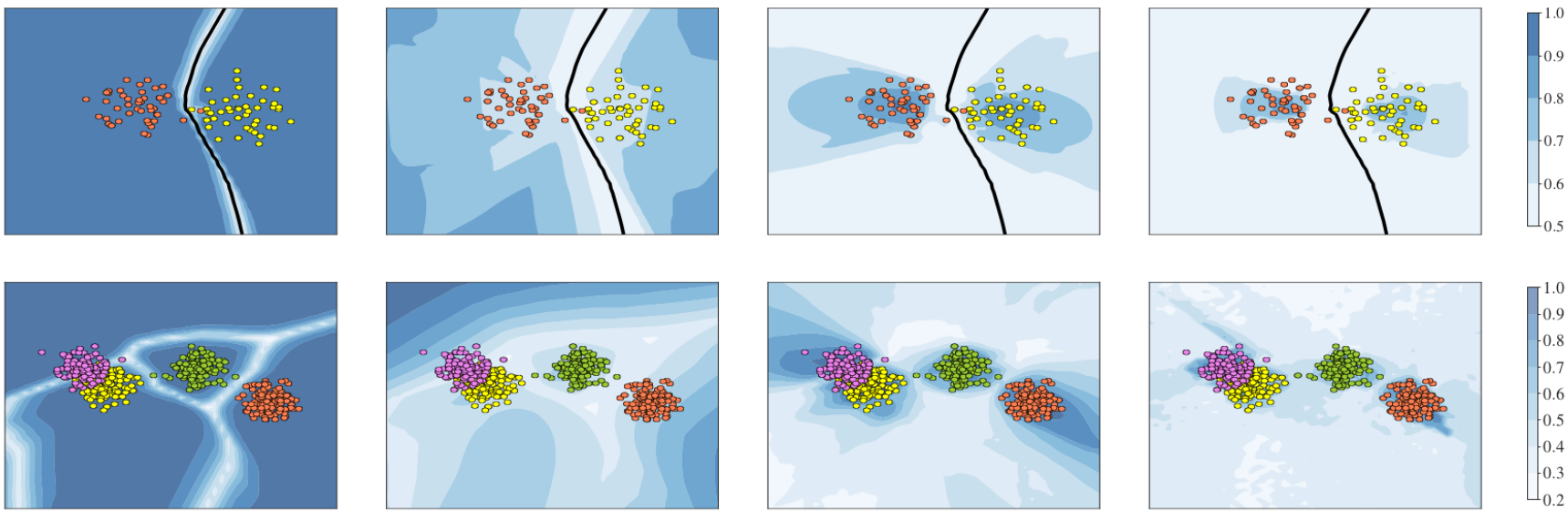


(a) MAP (b) Temp. scaling (c) LLLA (d) Full Laplace

Kristiadi et al., 2020: Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks (image credit)

Last-layer Laplace approximations (LLLA) I

- Laplace approximations for the last layer only is often a very good alternative
- ResNet18 to be fine-tuned to CIFAR10 (image dataset with 10 classes)
  1. Full network: contains $\approx 11 \cdot 10^6$ parameters in total
  2. Last fully connected layer: contains $\approx 5 \cdot 10^3$ parameters
- Often works almost as well as for the full Laplace approximation, but much faster



(a) MAP (b) Temp. scaling (c) LLLA (d) Full Laplace

Kristiadi et al., 2020: Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks (image credit)

Last-layer Laplace approximations (LLLA) II

- Consider our simple network with two hidden layers

$$\begin{aligned} \boldsymbol{z}_1 &= h(\boldsymbol{W}_0 \boldsymbol{x} + \boldsymbol{b}_0) \\ \boldsymbol{z}_2 &= h(\boldsymbol{W}_1 \boldsymbol{z}_1 + \boldsymbol{b}_1) \\ f &= \boldsymbol{W}_2 \boldsymbol{z}_2 + \boldsymbol{b}_2 \end{aligned}$$

- **Recipe for fitting a LLLA**

  Step 1: Compute $\boldsymbol{w}_{\text{MAP}}$ for full network (or use pretrained model)

  Step 2: Estimate (full) Hessian for parameters of the last layer $\boldsymbol{w}_L = \{\boldsymbol{W}_2, \boldsymbol{b}_2\}$

  Step 3: Construct approximation $q_{\text{LLLA}}(\boldsymbol{w}_L) = \mathcal{N}(\boldsymbol{w}_L | \boldsymbol{w}_L, \boldsymbol{A}_L^{-1})$

- **Recipe for making predictions with LLLA for new point $\boldsymbol{x}^*$**

  Step 1: Generate samples $\boldsymbol{w}_L^{(i)} \sim q_{\text{LLLA}}(\boldsymbol{w}_L)$ for $i = 1, \ldots, S$

  Step 2: First feed input $\boldsymbol{x}^*$ through first layers of network to get $\boldsymbol{z}_2^*$

  Step 3: Compute $f^{(i)} = f(\boldsymbol{z}_2^*)$ for each posterior sample of the weights $\boldsymbol{w}_L^{(i)} = \left\{\boldsymbol{W}_2^{(i)}, \boldsymbol{b}_2^{(i)}\right\}$

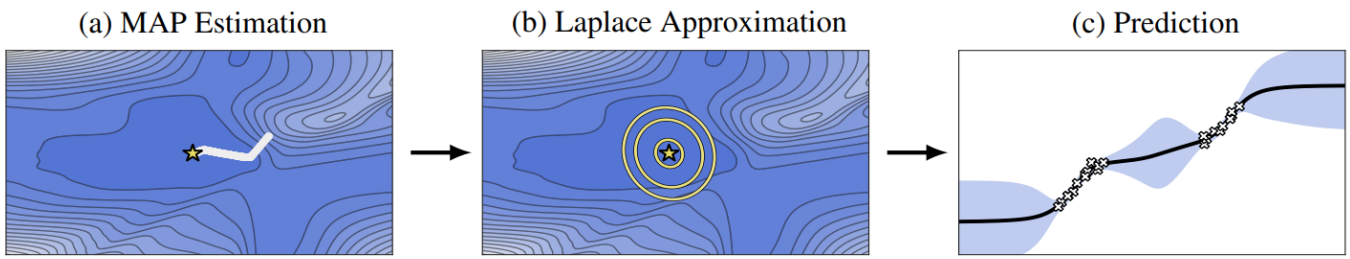
  Step 4: Use $\left\{f^{(i)}\right\}_{i=1}^{S}$ as samples to estimate the posterior predictive

Last-layer Laplace approximations (LLLA) III

- *Laplace Redux*: Very convenient and robust package for Laplace approximations in Pytorch

https://github.com/AlexImmer/Laplace

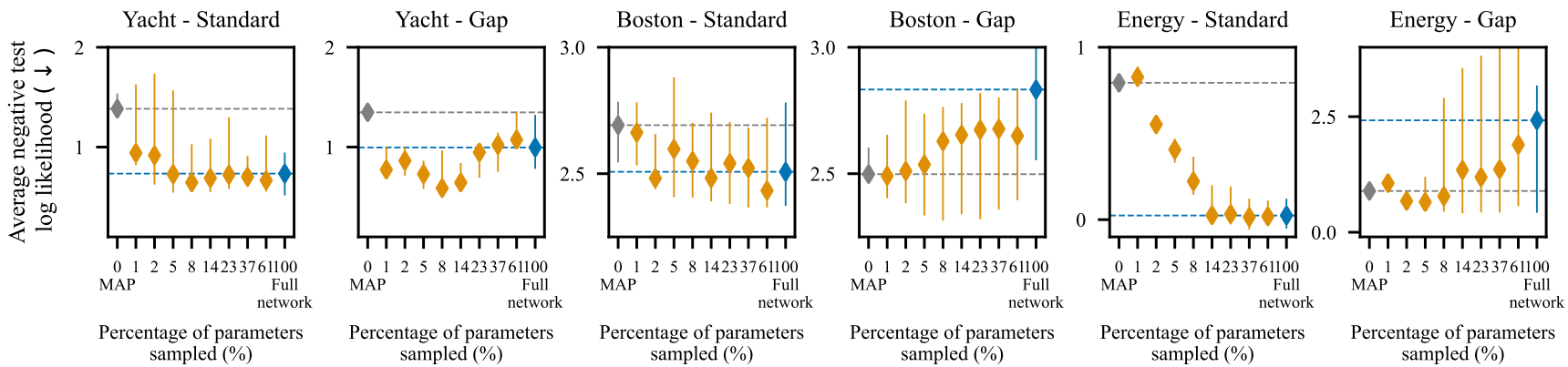
- *Plug and play* for many common Pytorch models
- Very good starting point for a more *Bayesian approach* to deep learning *in practice*
- Supports both *full Laplace* (with various approximations) and *last-layer Laplace*



Daxberger et al, 2022: Laplace Redux – Effortless Bayesian Deep Learning (image credit)

Is a stochastic last layer really the best?

- Recent work has shown that networks with a small amount of (input-independent) stochasticity *before* the last layer are *universal conditional distribution approximators*.
- This is verified experimentally: partially stochastic networks consistently either match or outperform their fully-stochastic variants.
- However, it is unclear if the sampling for the fully stochastic networks has adequately captured the posterior.

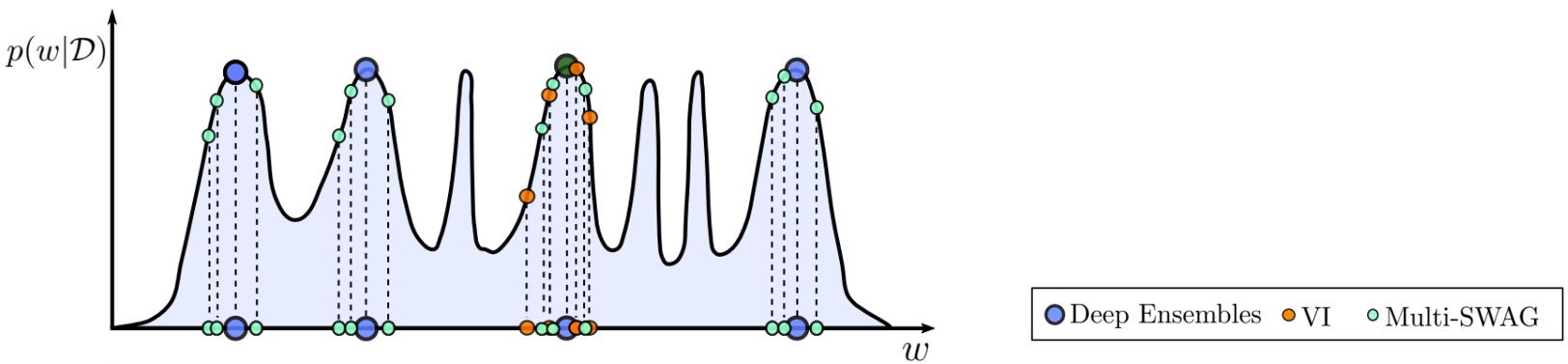


Sharma et al. (2023), *Do Bayesian Neural Networks Need To Be Fully Stochastic?* (image credit)

Bayesian Neural Networks: Deep Ensembles

Deep Ensembles I

- *Deep ensembling* is often an easy way to improve generalization and calibration
- Super simple idea
  1. re-train your model S times from different initial parameter values (e.g. different seeds)
  2. compute predictions for the S different model parameters and average the results



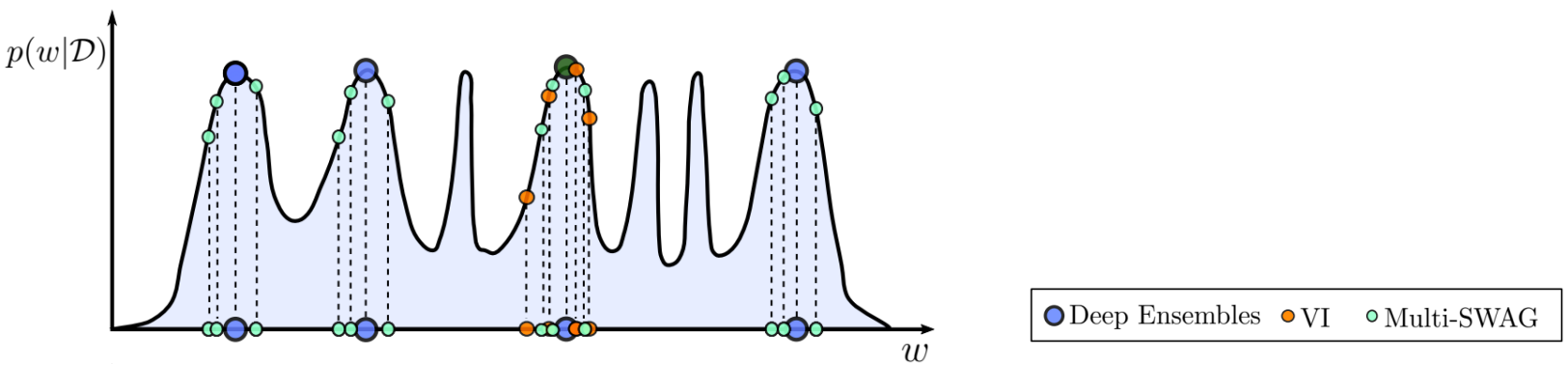
Lakshminarayanan et al, 2017: Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles

Wilson & Izmailov, 2021: Bayesian Deep Learning and a Probabilistic Perspective of Generalization (image credit)

Deep Ensembles I

- *Deep ensembling* is often an easy way to improve generalization and calibration
- Super simple idea
  1. re-train your model S times from different initial parameter values (e.g. different seeds)
  2. compute predictions for the S different model parameters and average the results
- More formally, deep ensembles approximates the predictive posterior as follows

$$p(y^*|\mathbf{y}, \mathbf{x}^*) \approx \int p(y^*|\mathbf{w}, \mathbf{x}^*) q_{\text{DE}}(\mathbf{w}) \mathrm{d}\mathbf{w} = \frac{1}{S} \sum_{i=1}^{S} p(y^*|\mathbf{w}^{(i)}, \mathbf{x}^*).$$



Lakshminarayanan et al, 2017: Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles

Wilson & Izmailov, 2021: Bayesian Deep Learning and a Probabilistic Perspective of Generalization (image credit)

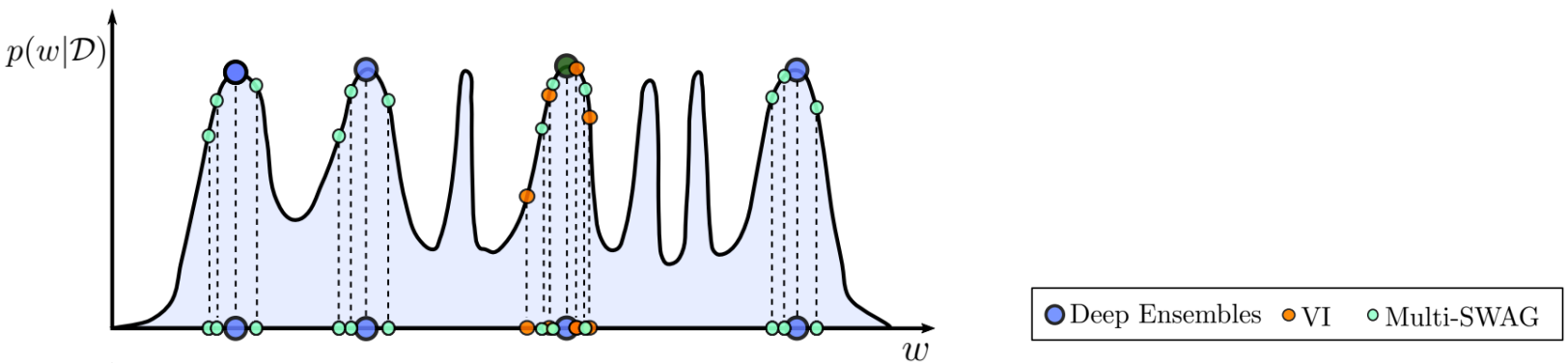
Deep Ensembles I

- *Deep ensembling* is often an easy way to improve generalization and calibration
- Super simple idea
  1. re-train your model S times from different initial parameter values (e.g. different seeds)
  2. compute predictions for the S different model parameters and average the results
- More formally, deep ensembles approximates the predictive posterior as follows

$$p(y^*|\mathbf{y}, \mathbf{x}^*) \approx \int p(y^*|\mathbf{w}, \mathbf{x}^*) q_{\text{DE}}(\mathbf{w}) \mathrm{d}\mathbf{w} = \frac{1}{S} \sum_{i=1}^{S} p(y^*|\mathbf{w}^{(i)}, \mathbf{x}^*).$$

- ... corresponding to a posterior approximation

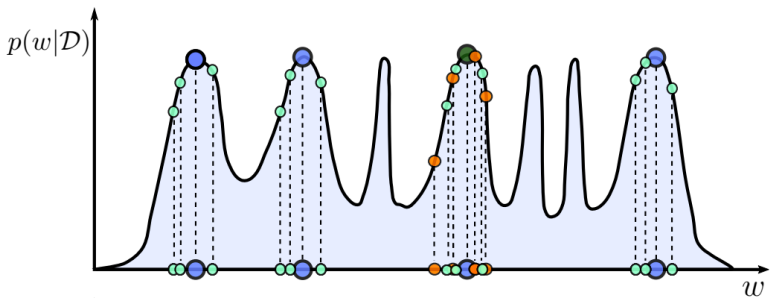
$$q_{\text{DE}}(\mathbf{w}) = \frac{1}{S} \sum_{i=1}^{S} \delta(\mathbf{w} - \mathbf{w}^{(i)})$$



Lakshminarayanan et al, 2017: Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles

Wilson & Izmailov, 2021: Bayesian Deep Learning and a Probabilistic Perspective of Generalization (image credit)

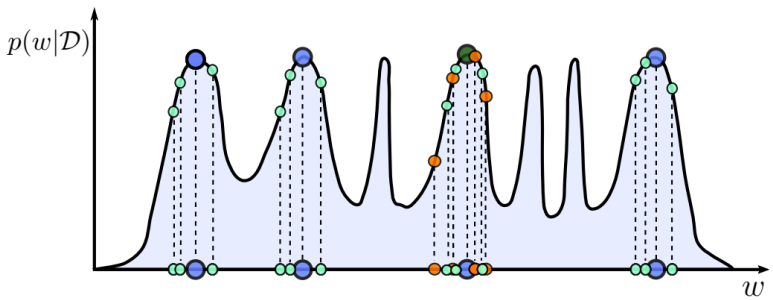
Deep Ensembles II



Lakshminarayanan et al, 2017: Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles

Wilson & Izmailov, 2021: Bayesian Deep Learning and a Probabilistic Perspective of Generalization (image credit)

Deep Ensembles II



Pros

1. Simple and very easy to implement
2. Training each model can easily be parallelized
3. Often works very well in practice

Cons

1. Requires training S copies of the model
2. Predictions become more expensive (S forward passes needed)
3. All models in the ensemble are given the same importance

Lakshminarayanan et al, 2017: Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles

Wilson & Izmailov, 2021: Bayesian Deep Learning and a Probabilistic Perspective of Generalization (image credit)

Summary for BNNs

- Motivation
  1. Classical neural networks are often overconfident
  2. We want both epistemic and aleatoric uncertainty
- Very challenging to do Bayesian inference for NNs
  1. Complicated posterior geometries
  2. Very high-dimensional parameter spaces
  3. Hard to assign informative priors
  4. Must scale to large datasets
- We discussed several options for approximate inference
  1. MC Dropout
  2. Mean-field
  3. Laplace approximations
  4. Deep ensembles

The end

- Thank you for participating in the course and for being patient with all the small issues: Typos in materials etc.
- Thank you for taking the time to provide valuable feedback and improving the course and the material
- Remember: Q&A session on Wednesday the 14th of May from 10-12 in building 321, room 033