

Learnable Inductive Biases in Neural Networks

Mark van der Wilk

Department of Computing
Imperial College London
<https://mvdw.uk>



@markvanderwilk
m.vdwilk@imperial.ac.uk

Jul 11, 2022

About our research group

Growing research group, with focus:

- ▶ Gaussian process inference, backed by theory to make **reliable decision making systems**.
 - ▶ Automatic learning of inductive bias in neural networks.
- Central question: **When should neurons be connected?**



Anish Dhir



Artem Artemev



Jose Pablo Folch



Ruby Sedgwick



Seth Nabarro



Tycho van der Ouderaa

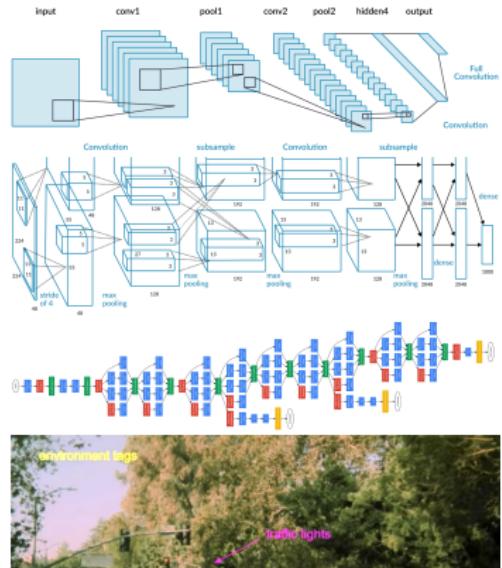
Hyperparameter Selection & Architecture Design

Every time we train a NN we need to decide on hyperparameters:

- ▶ How many layers? How many units in a layer?
- ▶ What layer structure? Convolutional? Skip connections?
- ▶ Data augmentation parameters?

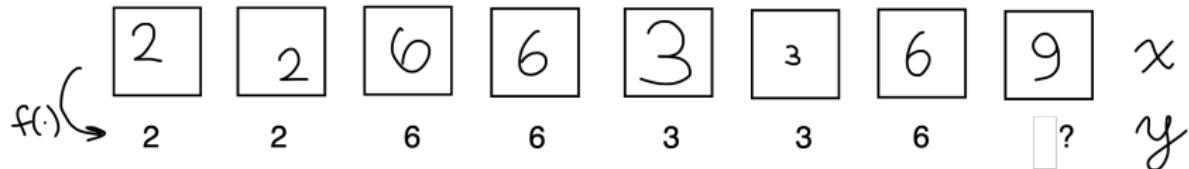
As architectures get more complex,
so does design! E.g. multitask.

- ▶ Which layers to share?
- ▶ What kind of task-specific layers?
- ▶ How much capacity to assign to each task?



Invariances

Every prediction problem needs an **inductive bias**:



Architecture determines inductive bias through
e.g. **equivariance**:

- ▶ Convolutions are a common solution
- ▶ Can also convolve according to other transformations (e.g. rotations)

Can we automatically adjust invariance properties in layers?



Summary

Goal:

Given a dataset,
adapt the inductive bias to it.

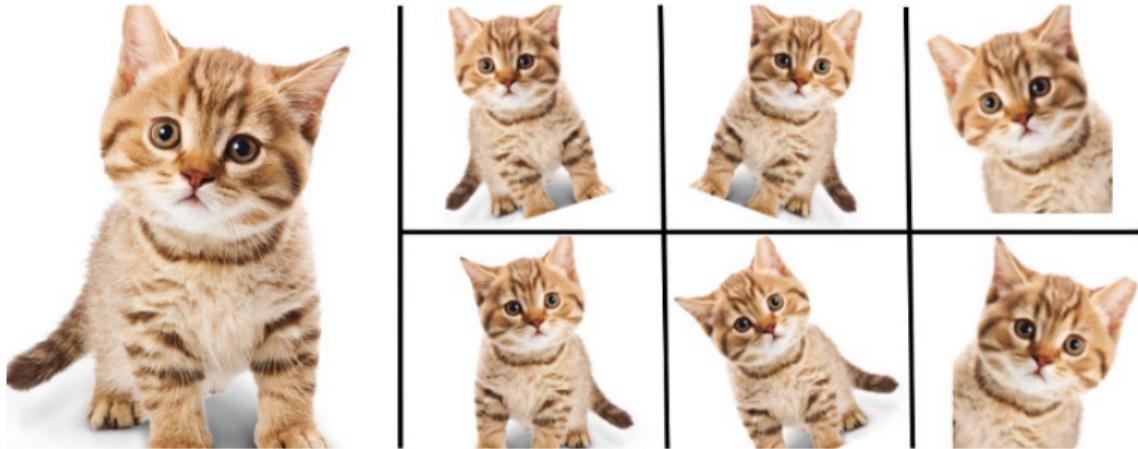
Key requirements:

1. Find a **parameterisation** for different inductive biases
2. Find a **learning objective** that works for inductive biases
 - ▶ We want to optimise it through **backprop** (so it's easy!)

We will look at:

- ▶ Invariances / equivariances parameterised though:
 - ▶ transformations on the input (data augmentation)
 - ▶ transformations on the filter (convolutions)
- ▶ How Bayes helps with finding a learning objective
- ▶ Single-layer and deep models

Data Augmentation

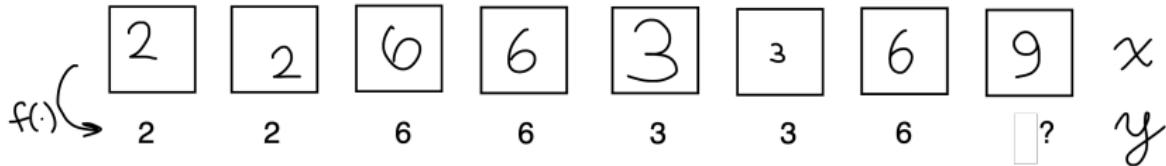


1. Take a dataset $\mathbf{y} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$
2. Create larger dataset $\mathbf{y}' = \{t(\mathbf{x}), y \mid t \in \mathcal{T}, (\mathbf{x}, y) \in \mathbf{y}\}$
3. Train predictor f on \mathbf{y}'

Making Models Invariant

Data augmentation:

- ▶ Imagine knowing that $f(\mathbf{x}) = f(t_i(\mathbf{x}))$ for transformations $\{t_i\}$
- ▶ E.g., translation, small rotations, scale...



Procedure:

- ▶ Apply random transformations to training data using $p(t)$
- ▶ Minimise new average training loss on new dataset:

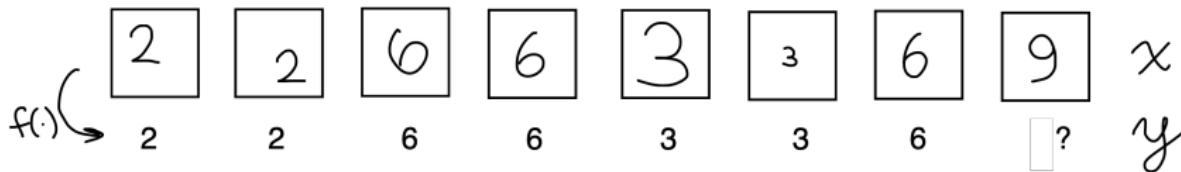
$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} \sum_{n=1}^N \mathbb{E}_{p(t)} [\text{loss}(f_{\mathbf{w}}(t(\mathbf{x})), y_n)] \quad (1)$$

Parameterising Invariance

What if we don't know the transformations for which $f(\mathbf{x}) = f(t_i(\mathbf{x}))$?

Follow the usual Machine Learning procedure!

1. Parameterise your unknowns!
2. Add parameters to your augmentation distribution: $p(t|\theta)$
3. Try to learn the unknown parameters θ !
 - ▶ You could think of θ as containing the *amount* of different transformations to apply.
 - ▶ In running example: Want small rotation invariance, large translation etc...



Training Loss

How do we find the right parameters θ ?

- ▶ Can we just minimise the training loss over θ ?

$$\mathbf{w}^*, \theta^* = \underset{\mathbf{w}, \theta}{\operatorname{argmin}} \sum_{n=1}^N \mathbb{E}_{p(t|\theta)}[\text{loss}(f_{\mathbf{w}}(t(\mathbf{x})), y_n)] \quad (2)$$

- ▶ It successfully learns \mathbf{w} , after all..!

No ☹

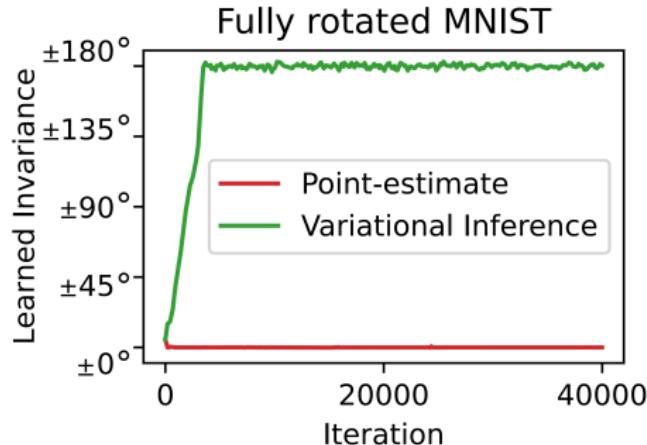
- ▶ Invariances (and data aug) try to **constrain** our model $f(\cdot)$ so that we have $f(\mathbf{x}) = f(t_i(\mathbf{x}))$
- ▶ The easiest way to minimise the loss is to make $\mathbf{x}_a = \mathbf{x}_n$,
... so no augmentation at all!

Training Loss: Example

- Rotated MNIST dataset:

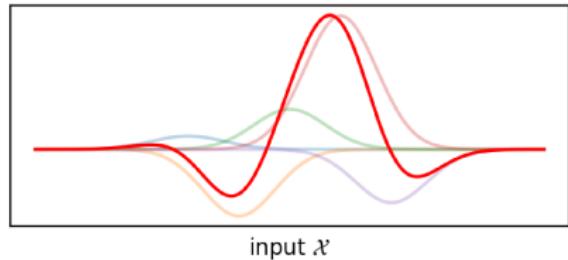
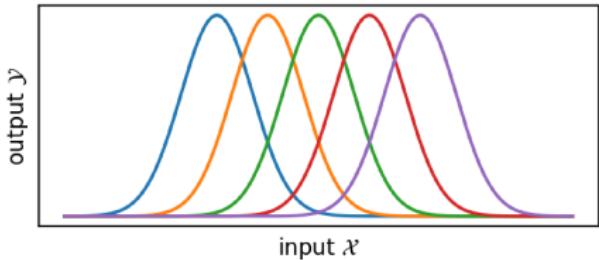


- The augmentation distribution $p(t|\theta)$ randomly rotates training input images, with θ controlling by **how much**.
- “Point estimate” minimises the training loss:



- This is why we use trial-and-error / cross-validation

Hyperparameter Selection Example



$$f_{\mathbf{w}, \theta}(x) = \boldsymbol{\phi}_{\theta}(x)^T \mathbf{w} = \sum_{i=1}^K \phi_{\theta}^{(i)}(x) w_i \quad (3)$$

$$\mathcal{L}_{\text{train}} = \sum_{n=1}^{N_{\text{train}}} (f_{\mathbf{w}, \theta}(x_n) - y_n)^2 + \lambda \|\mathbf{w}\|^2 \quad (4)$$

- ▶ Sum basis functions with weights \mathbf{w} , hyperparameters θ control “wigglyness”.
- ▶ Normally, we minimize $\mathcal{L}_{\text{train}}$ w.r.t. \mathbf{w} , while keeping hyperparameters θ fixed.
- ▶ Architectural choices change the **inductive bias**, like hyperparameters θ here change the width of basis functions.

Why do we need cross-validation?

What happens if we minimise $\mathcal{L}_{\text{train}}$ w.r.t. both w **and** θ ?

- ▶ Training loss learns weights, only with **fixed hyperparameters**.
- ▶ Why? Inductive bias is a **restriction** on functions.
Least restriction is best for training loss.
- ▶ Cross-validation: Try different values of θ ,
measure performance on separate **validation set**.
- ▶ Goal: Find **objective function** for hyperparameters
that we can optimise with **gradients**.

Bayesian Model Selection

Bayes tells us: Just find the posterior over all your unknowns!

$$p(f, \theta | \mathbf{y}) = \frac{p(\mathbf{y}|f)p(f|\theta)p(\theta)}{p(\mathbf{y})} = \underbrace{\frac{p(\mathbf{y}|f)p(f|\theta)}{p(\mathbf{y}|\theta)}}_{\text{usual posterior}} \underbrace{\frac{p(\mathbf{y}|\theta)p(\theta)}{p(\mathbf{y})}}_{\text{hyper posterior}} \quad (5)$$

- ▶ Posterior over functions is unchanged!
- ▶ Posterior over hyperparams requires **marginal likelihood**:

$$p(\mathbf{y}|\theta) = \int p(\mathbf{y}|f)p(f|\theta)d\theta \quad (6)$$

Bayesian model selection is commonly done by ML-II (Berger, 1985):

$$\theta^* = \operatorname{argmax}_{\theta} \log p(\mathbf{y} | \theta), \quad \text{predict using } p(f|\mathbf{y}, \theta^*) \quad (7)$$

Gradient-based optimisation is **super convenient!**
... if we can compute $p(\mathbf{y} | \theta)$

Bayesian Model Selection: Example

- ▶ Here, we optimise **marginal likelihood** instead of **training loss**.
- ▶ Can still be computed on **training data only**.
- ▶ And, we can **compute gradients!**
- ▶ No more trial-and-error, here hyperparameter selection is **just as easy as learning weights!**

Idea:

Learn NN architectural parameters in this way!

Learning Data Augmentation

Can we formulate learning data augmentation as Bayesian hyperparameter learning? (van der Wilk et al., 2018)

- ▶ Incorporate the invariance of data augmentation into the **function**, rather than the loss.

$$f_{\mathbf{w}, \theta}(\mathbf{x}) = \mathbb{E}_{p(t|\theta)} [\boldsymbol{\phi}_\theta(t(\mathbf{x}))^\top \mathbf{w}] \quad (8)$$

- ▶ This now gives a proper Bayesian model:

$$p(y_n | \mathbf{w}, \theta) = \mathcal{N}(y_n; f_{\mathbf{w}, \theta}(\mathbf{x}), \sigma^2), \quad p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; 0, I). \quad (9)$$

- ▶ The model has a well-defined marginal likelihood $p(\mathbf{y}|\theta)$, although it is intractable!

Variational Inference

We want a tractable approximation to the marginal likelihood $p(\mathbf{y}|\boldsymbol{\theta})$.

$$\log p(\mathbf{y}|\boldsymbol{\theta}) \geq \mathcal{L} = \sum_n \mathbb{E}_{q(\mathbf{w})} [\log p(y_n | \mathbb{E}_{p(t|\boldsymbol{\theta})} [\boldsymbol{\phi}_{\boldsymbol{\theta}}(t(\mathbf{x}))^\top \mathbf{w}])] - \text{KL}[q(\mathbf{w}) || p(\mathbf{w})] \quad (10)$$

Apply a **multi-sample** Jensen's inequality to $\mathbb{E}_{p(t|\boldsymbol{\theta})}$:

$$\mathcal{L} \geq \sum_n \mathbb{E}_{q(\mathbf{w}) \prod_s p(t_s|\boldsymbol{\theta})} \left[\log p(y_n | \frac{1}{S} \sum_s \boldsymbol{\phi}_{\boldsymbol{\theta}}(t_s(\mathbf{x}))^\top \mathbf{w}) \right] - \text{KL}[q(\mathbf{w}) || p(\mathbf{w})]$$

(Nabarro et al., 2022; Schwöbel et al., 2022; van der Ouderaa and van der Wilk, 2022)

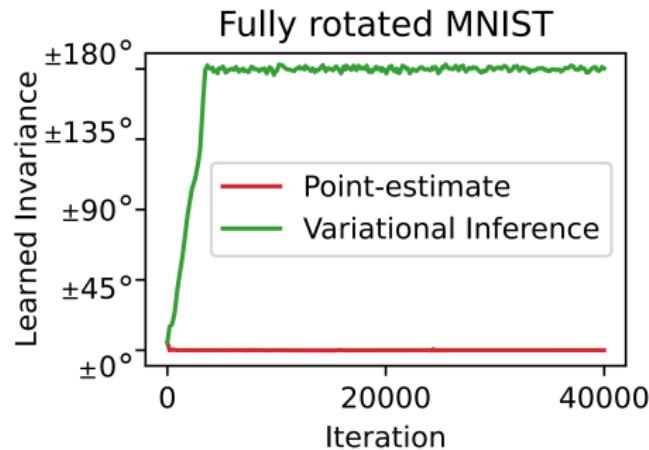


Results

- Rotated MNIST dataset:



- We now optimise $\mathcal{L}(q(\mathbf{w}), \theta)$, which avoids the collapse!

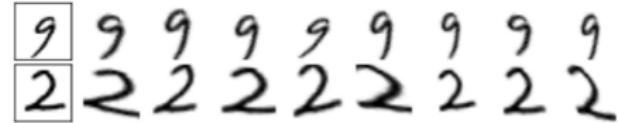
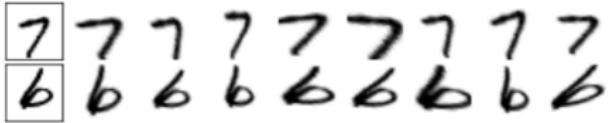
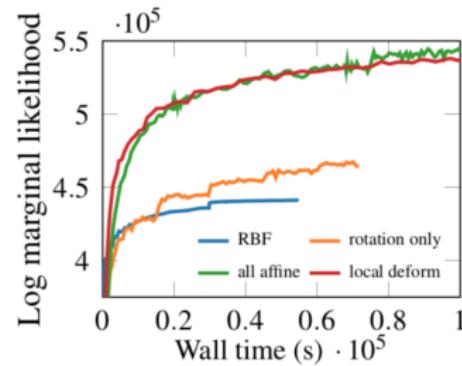
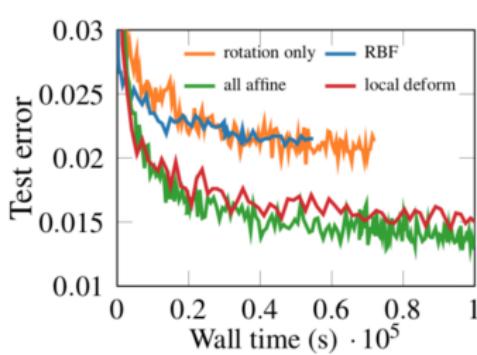


- Can use gradient-based optimisation,
instead of trial-and-error / cross-validation!

Results

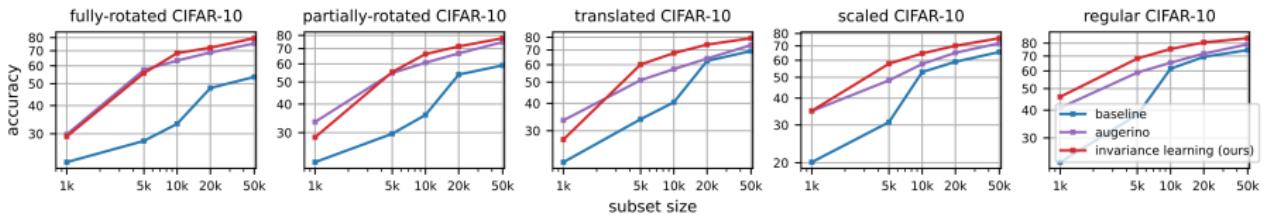
We used various $p(t|\theta)$:

- ▶ Affine transformations (parameters: how much rotation / skew / scale to apply)
- ▶ Local deformations (parameters: how much deformation, how much to smooth deformations etc)



Deep Neural Networks

- ▶ In Immer et al. (2022), we apply the same principle to learn invariance parameters for **deep** neural networks.
- ▶ We use a Kronecker Factorised (K-FAC) Laplace approximation.
- ▶ Can be justified as a Bayesian approximation, or as searching for flat minima.
- ▶ Improves performance for small subsets of data, and full dataset:



First authors Alexander Immer and Tycho van der Ouderaa



Learning Equivariances

- ▶ Previously, we added invariance by transforming the input image
- ▶ Can we learn equivariance by transforming weights, like (group) convolutions?
- ▶ For a single layer, they are actually (nearly) equivalent!

Our construction for Data Augmentation was:

$$f_{\mathbf{w}, \theta}(\mathbf{x}) = \mathbb{E}_{p(t|\theta)} [\boldsymbol{\phi}_\theta(t(\mathbf{x}))^\top \mathbf{w}] \quad (11)$$

- ▶ For neural networks, we have $\boldsymbol{\phi}_\theta(t(\mathbf{x})) = \sigma(W \circ t \circ \mathbf{x})$
- ▶ We can choose to apply t to W or \mathbf{x} !
- ▶ Transforming the filters allows us to learn convolutional structure (van der Ouderaa and van der Wilk, 2022)

Learning Equivariances

Rotated MNIST dataset: 

Learned filters:

Filters for other transformed MNIST variants:



(a) Sampled filters of affine model trained on regular mnist.



(b) Sampled filters of affine model trained on rotated mnist.



(c) Sampled filters of affine model trained on scaled mnist.



(d) Sampled filters of affine model trained on translated mnist.

Learning Invariances: Papers

- ▶ **Learning Invariances using the Marginal Likelihood**
(van der Wilk et al., 2018)
Learning invariance by backprop, but Gaussian processes only.
- ▶ **Data augmentation in BNNs and the cold posterior effect**
(Nabarro et al., 2022)
We investigated whether a principled approach to DA influences the cold posterior.
- ▶ **Learning Invariant Weights in Neural Networks**
(van der Ouderaa and van der Wilk, 2021)
Show how filter banks can be learned, but shallow NNs only.
- ▶ **Invariance Learning in Deep Neural Networks with Differentiable Laplace Approximations**
(Immer et al., 2022)
We show that the marginal likelihood works in deep NNs, and is competitive.

Summary

Goal:

Given a dataset,
adapt the inductive bias to it.

Key requirements:

1. Find a **parameterisation** for different inductive biases
2. Find a **learning objective** that works for inductive biases
 - ▶ We want to optimise it through **backprop** (so it's easy!)

We looked at:

- ▶ Invariances / equivariances parameterised though:
 - ▶ transformations on the input (data augmentation)
 - ▶ transformations on the filter (convolutions)
- ▶ How Bayes helps with finding a learning objective
- ▶ Single-layer and deep models

Outlook

- ▶ We want something better than trial-and-error to design NNs.
- ▶ Bayesian methods are helping the automation of selecting **invariances**, and making it as easy as **backprop**!
- ▶ Can help make NNs **1)** more accurate, **2)** easier to use, **3)** more energy-efficient.
- ▶ A lot more to do to get to **the smarter neuron!**
Meta-learning? More Bayes? Causality? Cellular automata?

Hard to say, but it'll be fun to find out!

Join us!



Anish Dhir



Artem Artemev



Jose Pablo Folch



Ruby Sedgwick



Seth Nabarro



Tycho van der Ouderaa

- ▶ There will be an open PhD position for Apr/Oct 2023.
- ▶ Check my website (<https://mvdw.uk/>) for tips on applying, and how to get in touch.
- ▶ Topics: Invariance, Bayes, Gaussian processes, BayesOpt, PAC-Bayes, causality, meta-learning, model-based RL.

Variational Bayesian Model Selection

Bayes tells us what to do, but not how to do it. Variational inference actually does it, and gives us

- ▶ An approximate posterior
- ▶ An estimate of the marginal likelihood! (lower bound)

$$\begin{aligned}\log p(\mathbf{y} | \theta) &= \mathcal{L}(\phi, \theta) + \text{KL}[q_\phi(f) || p(f|\mathbf{y}, \theta)] \\ &\geq \mathcal{L} = \sum_{n=1}^N \mathbb{E}_{q_\phi(f(\mathbf{x}_n))} [\log p(y_n | f(\mathbf{x}_n))] - \text{KL}[q_\phi(f) || p(f|\theta)]\end{aligned}$$

- ▶ Find posterior and hyperparameters simultaneously by

$$\underset{\phi, \theta}{\operatorname{argmax}} \mathcal{L}(\phi, \theta) \tag{12}$$

- ▶ Quality of posterior is linked to the accuracy of lower bound!

References I

- Berger, J. O. (1985). Statistical decision theory and Bayesian analysis. Springer.
- Immer, A., van der Ouderaa, T. F. A., Fortuin, V., Rätsch, G., and van der Wilk, M. (2022). Invariance learning in deep neural networks with differentiable laplace approximations.
- Nabarro, S., Ganey, S. K., Garriga-Alonso, A., Fortuin, V., van der Wilk, M., and Aitchison, L. (2022). Data augmentation in bayesian neural networks and the cold posterior effect. In The 38th Conference on Uncertainty in Artificial Intelligence.
- Ru, B., Lyle, C., Schut, L., Fil, M., van der Wilk, M., and Gal, Y. (2021). Speedy performance estimation for neural architecture search. In Advances in Neural Information Processing Systems (NeurIPS), volume 34.

References II

- Schwöbel, P., Jørgensen, M., Ober, S. W., and van der Wilk, M. (2022). Last layer marginal likelihood for invariance learning. In Proceedings of the Twenty Fifth International Conference on Artificial Intelligence and Statistics (AISTATS).
- van der Ouderaa, T. and van der Wilk, M. (2021). Learning invariant weights in neural networks. In ICML 2021 Workshop on Uncertainty & Robustness in Deep Learning.
- van der Ouderaa, T. F. and van der Wilk, M. (2022). Learning invariant weights in neural networks. In The 38th Conference on Uncertainty in Artificial Intelligence.
- van der Wilk, M., Bauer, M., John, S., and Hensman, J. (2018). Learning invariances using the marginal likelihood. In Advances in Neural Information Processing Systems 31.