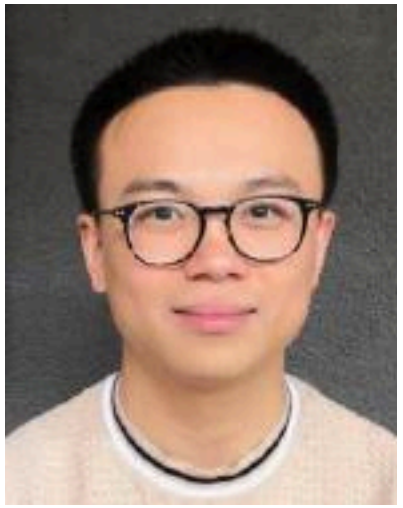
Fast Computation with Linearized Neural Networks for Domain Adaptation

Andreas Damianou - **Alexa Shopping Science**

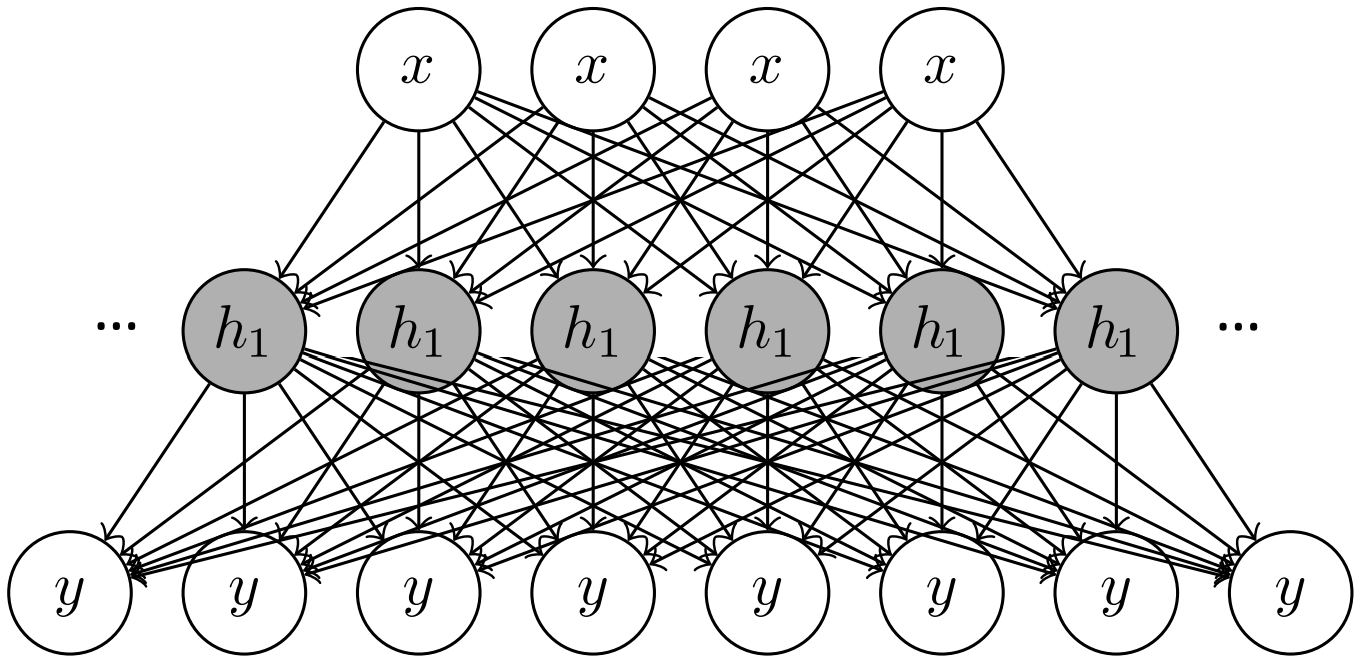
amazon

W. Maddox, S. Tang, P. G. Moreno, A. G. Wilson, A. Damianou:
Fast Adaptation with Linearized Neural Networks. 2019

Infinite width NN → GP

Also assuming Gaussian *i.i.d* noise for weights and biases

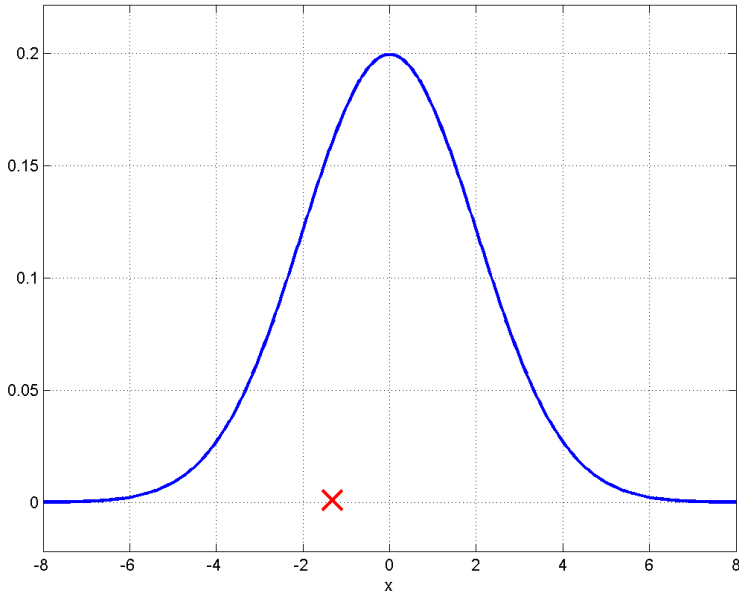


GP

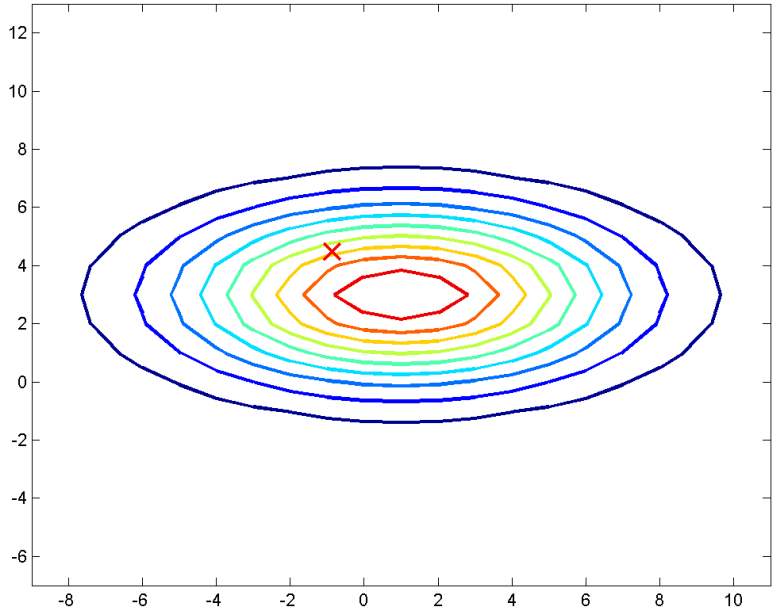
Neal 1994; Williams 1997

amazon

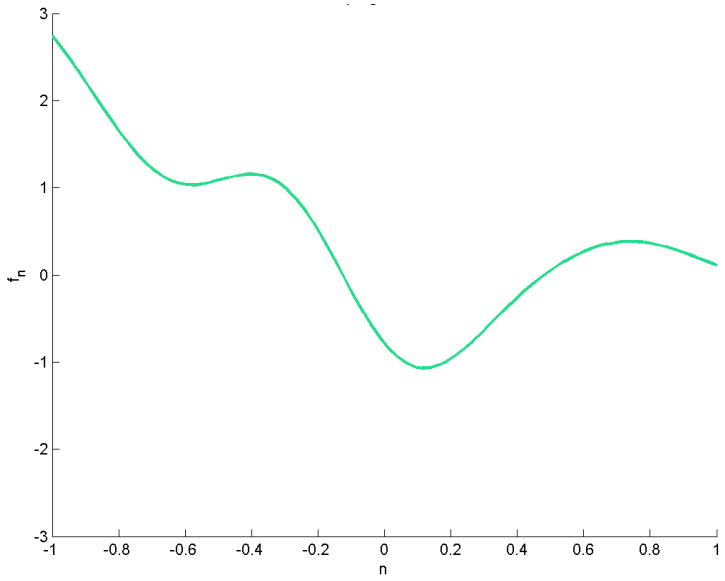
GP: Infinite dimensional Gaussian distribution



1-dim



2-dim



∞-dim

A GP is a distribution over functions.

amazon

Gaussian process

- A Gaussian **distribution** depends on a covariance **matrix**.
- A Gaussian **process** depends on a covariance **function**.

$$p(f_1, f_2, \cdots, f_n) \sim \mathcal{N}(0, \mathbf{K})$$

$$p(f_1, f_2, \cdots, f_n, \cdots, f_\infty) \sim \mathcal{GP}(0, \underbrace{k(x, x')})$$

covariance function

amazon

Neural networks:

- Require a large number of data and, even then, are fooled by out-of-distribution examples
- Do not have good uncertainty calibration $p(y|f(\hat{w}_D))$ vs $\int_f p(y|f)p(f|D)$
- Are not easy to compose together (black-boxes) $f([w_1, w_2])$ vs $f_1 \circ f_2$

But they are easy to learn and work well in practice!

amazon

Defining the model != learning the model

ML in a nutshell:

STEP 1: Define the model
STEP 2: Learn the model

Defining the model != learning the model

ML in a nutshell:

STEP 1: Define the model
STEP 2: **Learn** the model

"Did we throw the baby out with the bathwater? (MacKay 2002)"

amazon

Our Motivation

- Use neural networks in the situations where they work well.
- Encode them in function space through a GP with Neural Tangent Kernel (NTK).
- Manipulate the resulting function e.g. for domain adaptation.

amazon

Our Motivation

- Use neural networks in the situations where they work well.
- Encode them in function space through a GP with Neural Tangent Kernel (NTK).
- Manipulate the resulting function e.g. for domain adaptation.

*We encode inductive biases from the **trained** DNN (leveraging learnability), **not** from the DNN architecture (which would defeat the purpose).*

W. Maddox, S. Tang, P. Moreno, A. Wilson, A. Damianou: *Fast Adaptation with Linearized Neural Networks*. 2019

amazon

Adaptation in parameter vs function space

If $f_1(w_1)$ solves task 1 and $f_2(w_2)$ (unknown) solves task 2:

- Fine-tuning: Take an arbitrary subset of (the vast) w_1, move it for an arbitrary number of steps towards (hopefully) w_2.
- Our method: Take the whole f_1 and move it towards f_2 in the function space - closed-form (for regression) with accompanying uncertainty.

A GP encodes smoothness assumptions: task 1 close to task $2 \Rightarrow f_1$ close to f_2.

amazon

GP with NN inductive biases

- Fit DNN's parameters $\mathbf{w}$ on data $\mathbf{x}$.
- Consider a GP f with kernel: $k(x, x') = J(x; \mathbf{w})^\top J(x'; \mathbf{w})$.
- Adaptation: $p(f_*|x_*, \mathbf{x}, \mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}_*, k_{**} - \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{k}_{\mathbf{x}*})$

  where e.g. $\mathbf{k}_{*\mathbf{x}} = k(x_*, \mathbf{x}) = J(x_*; \mathbf{w})^T J(\mathbf{x}; \mathbf{w})$.
- Equivalent to Bayesian generalized linear model with Jacobians as features.

Efficient computations

$$\boldsymbol{\mu}_* = \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{f}$$
$$k_* = k_{**} - \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{k}_{\mathbf{x}*}$$

- Observation 1: Difficult parts can be cached.
- Observation 2: We can turn inversion into optimization:

$g(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top\mathbf{K}\mathbf{a} - \mathbf{a}^\top\mathbf{f}$ has maximum at $\hat{\mathbf{a}} = \mathbf{K}^{-1}\mathbf{f}$

- For t iterations the covariance expressed in terms of matrices with t columns.
- Allows to access $\mathbf{K}^{-1}\mathbf{f}$ lazily using fast MVP.

Efficient computations

$$\boldsymbol{\mu}_* = \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{f}$$
$$k_* = k_{**} - \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{k}_{\mathbf{x}*}$$

- Observation 1: Difficult parts can be cached.
- Observation 2: We can turn inversion into optimization:

$$g(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top \mathbf{K}\mathbf{a} - \mathbf{a}^\top \mathbf{f} \text{ has maximum at } \hat{\mathbf{a}} = \mathbf{K}^{-1}\mathbf{f}$$

- For t iterations the covariance expressed in terms of matrices with t columns.
- Allows to access $\mathbf{K}^{-1}\mathbf{f}$ lazily using fast MVP.

Efficient computations

$$\boldsymbol{\mu}_* = \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{f}$$
$$k_* = k_{**} - \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{k}_{\mathbf{x}*}$$

- Observation 1: Difficult parts can be cached.
- Observation 2: We can turn inversion into optimization:

$g(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top\mathbf{K}\mathbf{a} - \mathbf{a}^\top\mathbf{f}$ has maximum at $\hat{\mathbf{a}} = \mathbf{K}^{-1}\mathbf{f}$ *[Pleiss et al. 2018]*

- For t iterations the covariance expressed in terms of matrices with t columns.
- Allows to access $\mathbf{K}^{-1}\mathbf{f}$ lazily using fast MVP.

Efficient computations

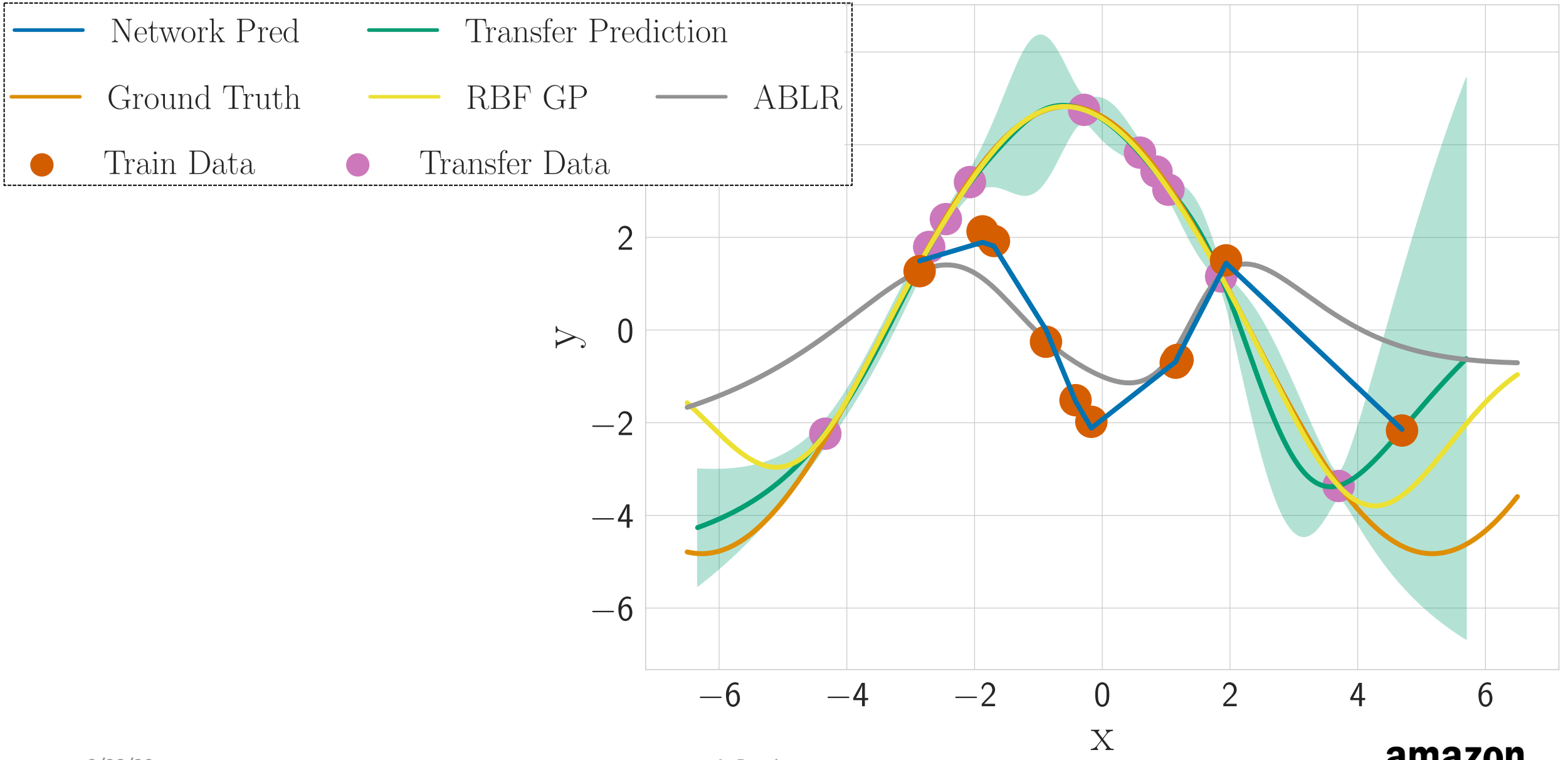
$$\boldsymbol{\mu}_* = \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{f}$$
$$k_* = k_{**} - \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{k}_{\mathbf{x}*}$$

- Observation 1: Difficult parts can be cached.
- Observation 2: We can turn inversion into optimization:

$g(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top\mathbf{K}\mathbf{a} - \mathbf{a}^\top\mathbf{f}$ has maximum at $\hat{\mathbf{a}} = \mathbf{K}^{-1}\mathbf{f}$ *[Pleiss et al. 2018]*

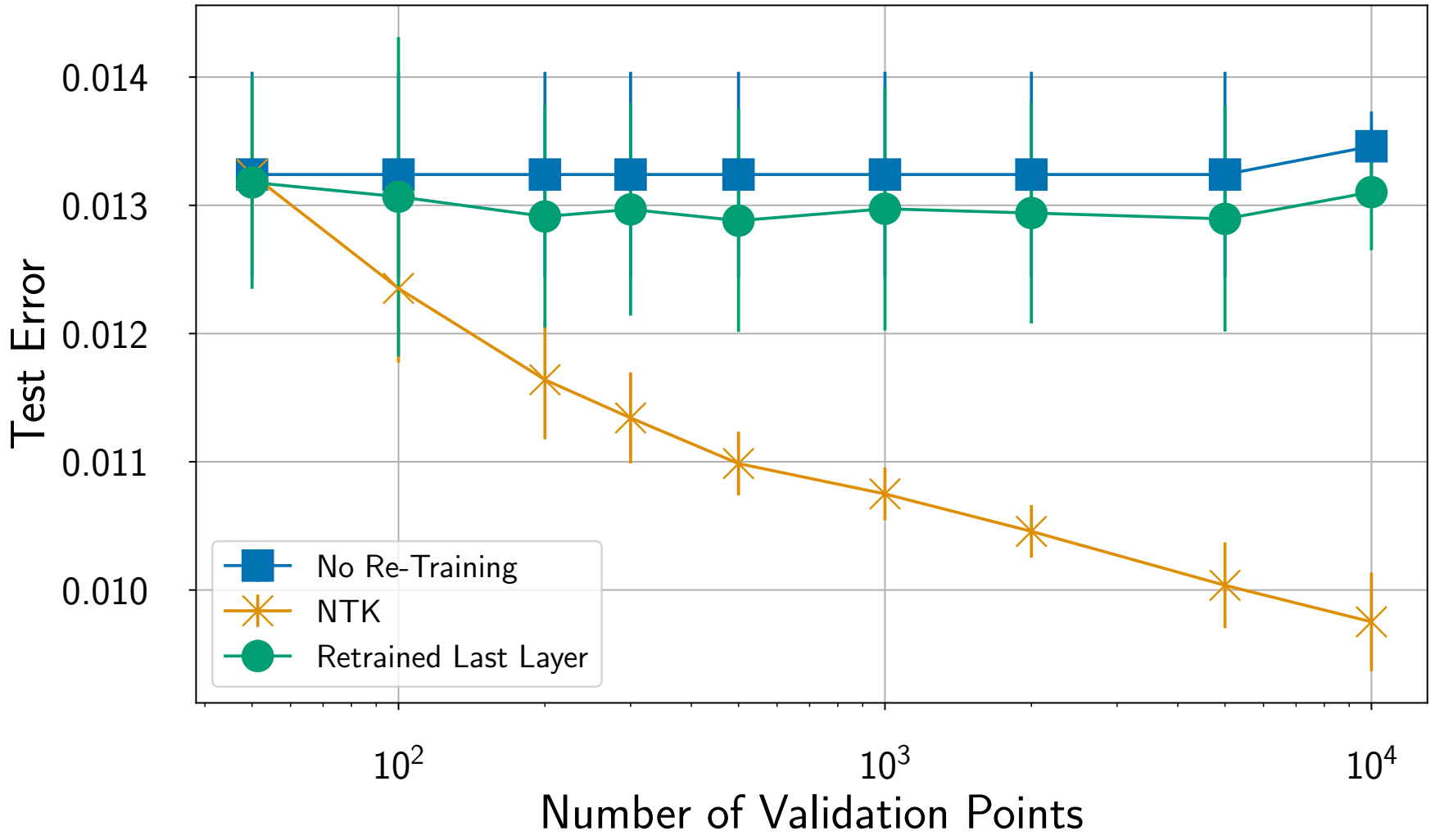
- For t iterations the covariance expressed in terms of matrices with t columns.
- Allows to access $\mathbf{K}^{-1}\mathbf{f}$ lazily using fast MVP.

Transfer learning (toy regression data)



amazon

Malaria spread data



amazon

What I don’t have time to show

- Theory from DNN training dynamics across GD justifies the approach.
- Justifying the assumption about Jacobian similarity across tasks.
- Framework for working in the parameter space (Bayesian GLM) or the function space (GP).
- Fast computations through MVP.

amazon

Thanks!

Questions?

Appendix

DNN training dynamics in GD

$$\frac{\mathrm{d}f(x)}{\mathrm{d}t} = -\eta \mathbf{J}_{\mathbf{w}}(x)^{\top} \mathbf{J}_{\mathbf{w}}(x) \nabla_f \log p(y|f,x)$$

Neural Tangent Kernel (NTK)

- The NTK governs the dynamics of f throughout the GD training of $\mathbf{w}$.
- Taylor expand: $f(x, \mathbf{w}) \approx f(x, \mathbf{w}_0) + J_{\mathbf{w}}(x, \mathbf{w}_0)^T (\mathbf{w} - \mathbf{w}_0)$
- Linear in $\mathbf{w}$, non-linear in inputs (because of J)
- Linear model using feature map (kernel) $J_{\mathbf{w}}(x, \mathbf{w}_0)^{\top} J_{\mathbf{w}}(x, \mathbf{w}_0)$

Lee et al. *arXiv:1902.06720*

amazon

DNN training dynamics in GD

$$\frac{\mathrm{d}f(x)}{\mathrm{d}t} = -\eta \underbrace{\mathbf{J}_{\mathbf{w}}(x)^{\top}\mathbf{J}_{\mathbf{w}}(x)}_{\text{Neural Tangent Kernel (NTK)}} \nabla_f \log p(y|f,x)$$

- The NTK governs the dynamics of f throughout the GD training of $\mathbf{w}$.
- Taylor expand: $f(x,\mathbf{w}) \approx f(x,\mathbf{w}_0) + J_{\mathbf{w}}(x,\mathbf{w}_0)^T(\mathbf{w}-\mathbf{w}_0)$
- Linear in $\mathbf{w}$, non-linear in inputs (because of J)
- Linear model using feature map (kernel) $J_{\mathbf{w}}(x,\mathbf{w}_0)^{\top} J_{\mathbf{w}}(x,\mathbf{w}_0)$

Lee et al. *arXiv:1902.06720*

amazon

DNN -> GP through NTK

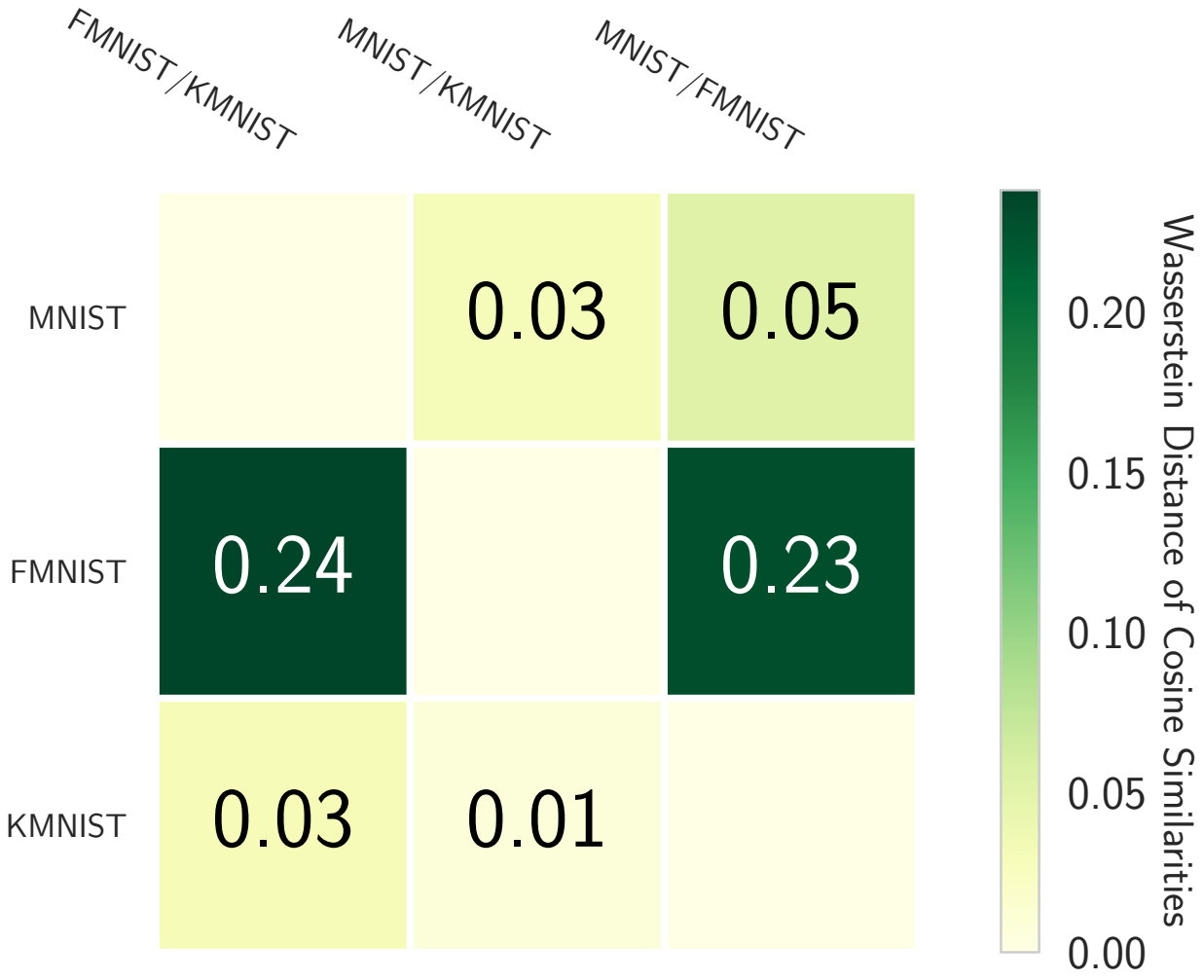
- For linearized networks around w0, the network output becomes a linear model with NTK.
- For non-linearized networks and small learning rate, same behavior arises with GD.

DNN -> GP through NTK

- For linearized networks around *w0*, the network output becomes a linear model with NTK.
- For non-linearized networks and small learning rate, same behavior arises with GD.
- We leverage this to linearize the network at convergence, *w_final*, and use it within a GP with the NTK.
- This also allows us to do DNN transfer learning analytically with GPs: we transfer neural network parameters (= kernel parameters) across tasks.

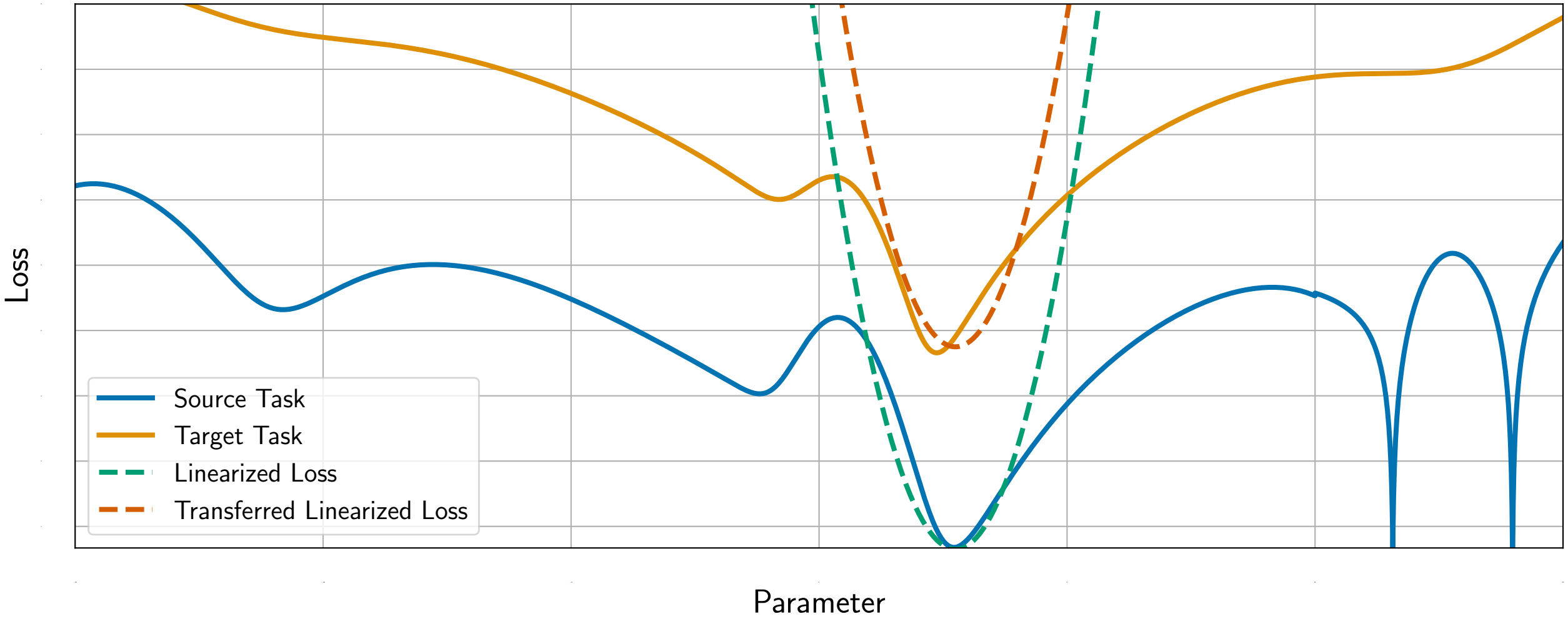
amazon

Similar tasks have similar Jacobians



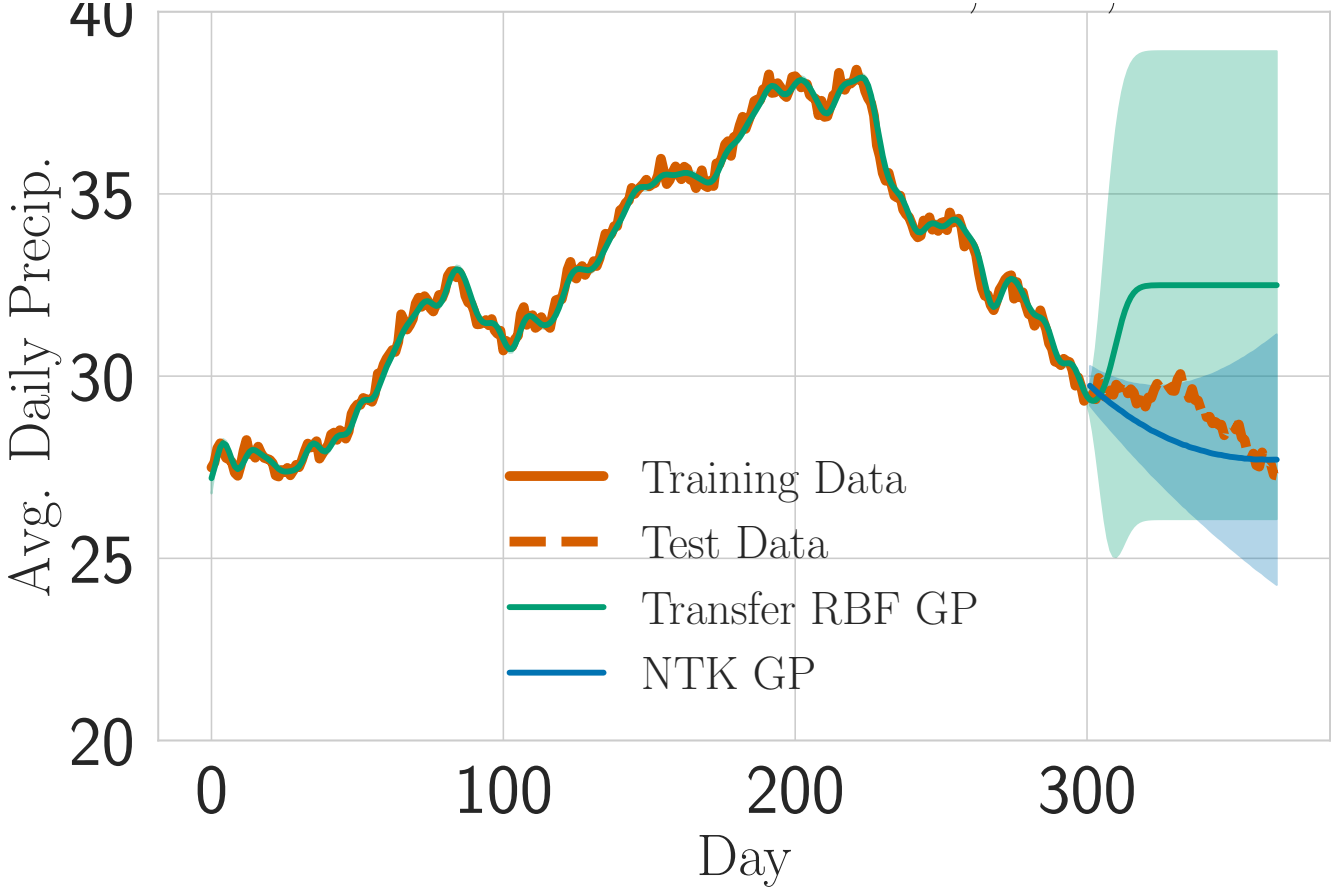
See also Fisher Matrix similarity assumptions by Lian et al. '17, Achille et al. '19

amazon

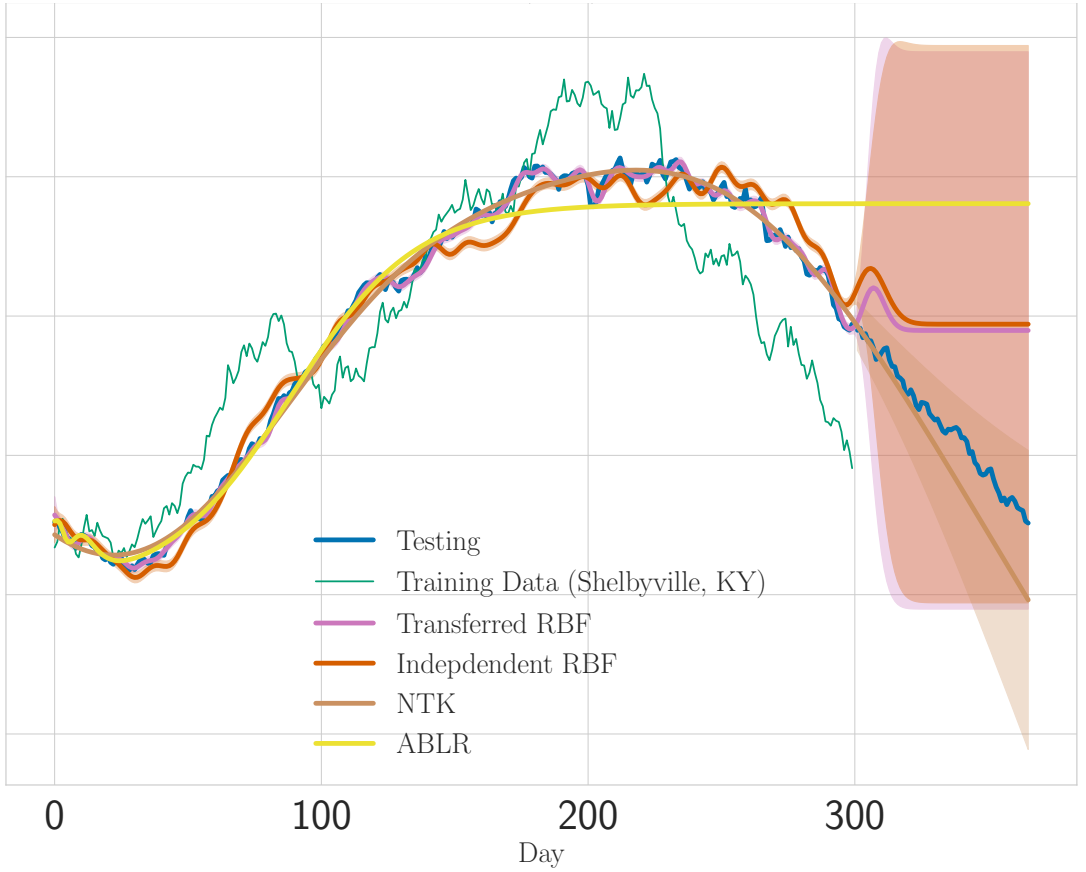

Loss
Parameter
Source Task
Target Task
Linearized Loss
Transferred Linearized Loss

Transfer learning for precipitation data

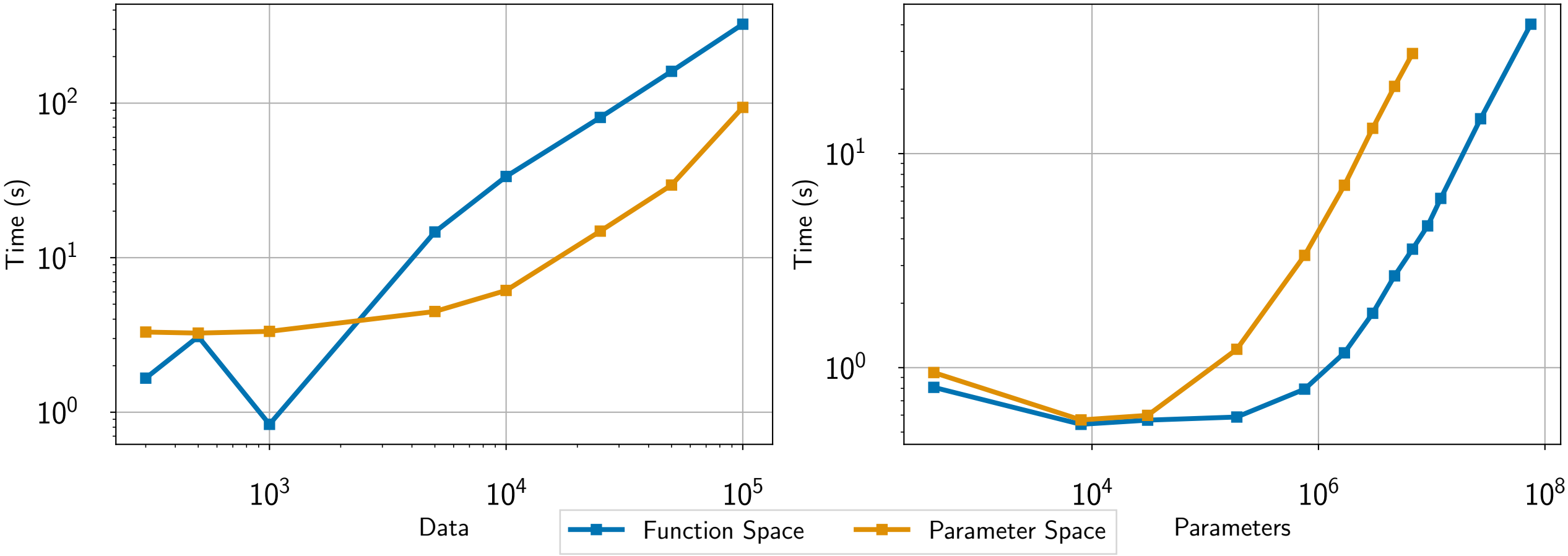
Source task



Target task



Scaling


Time (s)
10^2
10^1
10^0
10^3
10^4
10^5
Data
Function Space
Parameter Space
Time (s)
10^1
10^0
10^4
10^6
10^8
Parameters