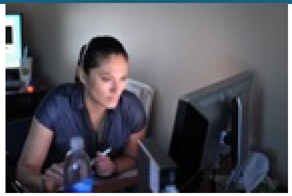
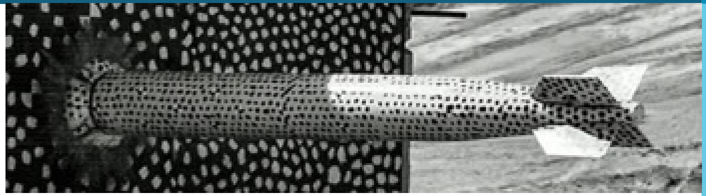
SAND2022-9064C

Sandia National Laboratories

Training and Generalization of Residual Neural Networks as Discrete Analogues of Neural ODEs





Khachik Sargsyan

SNL : Joshua Hudson, Oscar Diaz-Ibarra, Marta D'Elia, Habib Najm
Emory U : Lars Ruthotto, Haley Rosso

July 26, 2022

MLDL Workshop

U.S. DEPARTMENT OF ENERGY NNSA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Despite all the success, there are many recognized challenges and unknowns in neural network behavior

- **Generalization / predictability**
- **Physical insight / interpretation**
- **Confidence assessment**
- **Robustness to adversarial attacks**

Take advantage of 'continuous' limit of residual neural networks.

Neural ODEs

Probabilistic viewpoint opens even more opportunities

Probabilistic NN

Road to Trusted AI

Arguably, the two most important hurdles along the way

Goals: Generalization / predictability | Confidence assessment

Tools: Neural ODEs / ResNets — Probabilistic NN

Take advantage of legacy knowledge in ODEs and UQ to achieve

- Improved architectures
- Generalizable models

- Confidence assessment
- Robustness to noise

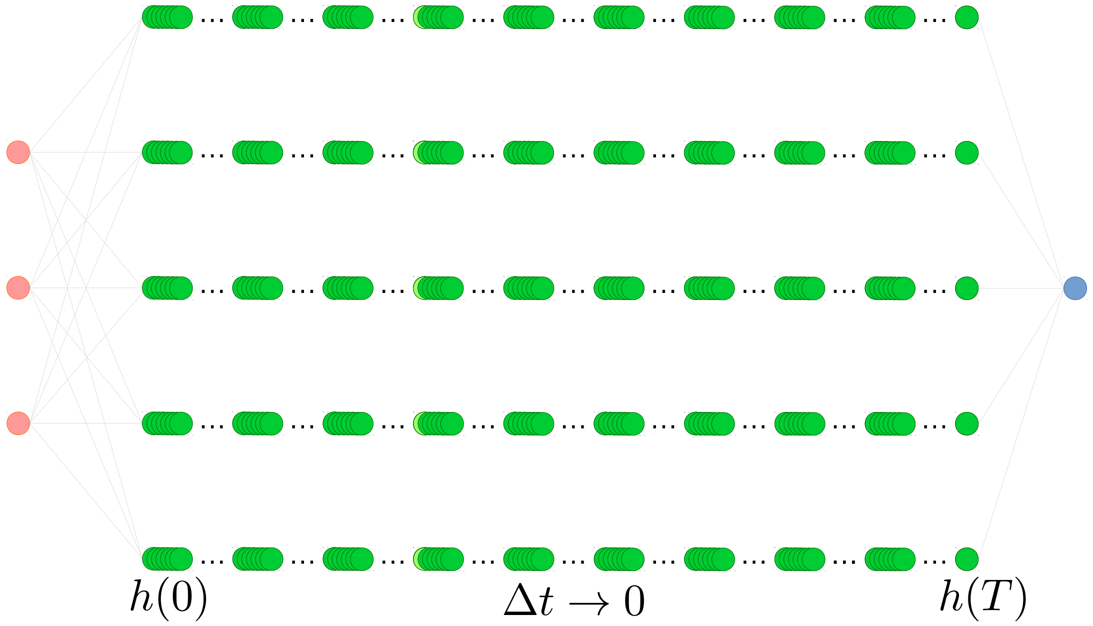
Main building block: ResNets

Neural Networks (NNs) layer-to-layer function $\overset{\text{state}}{h_{t+1}} = F(h_t, \theta)$ (state: h_t; weights: θ)

Residual NN: learn the residual, not the state $h_{t+1} = h_t + F(h_t, \theta)$

Now, take the limit of infinite layers

$$\frac{dh(t)}{dt} = F(h(t), \theta)$$



Focus on: ResNet and NODE in a regression setting (supervised ML)

ResNet (discrete)

$$\begin{cases} \boldsymbol{x}_1 = \boldsymbol{x} + \boldsymbol{\alpha}_0 \boldsymbol{\sigma}(\boldsymbol{W}_0 \boldsymbol{x}_0 + \boldsymbol{b}_0) \\ \vdots \\ \boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{\alpha}_n \boldsymbol{\sigma}(\boldsymbol{W}_n \boldsymbol{x}_n + \boldsymbol{b}_n) \\ \vdots \\ \boldsymbol{y} = \boldsymbol{x}_{L-1} + \boldsymbol{\alpha}_{L-1} \boldsymbol{\sigma}(\boldsymbol{W}_{L-1} \boldsymbol{x}_{L-1} + \boldsymbol{b}_{L-1}) \end{cases}$$

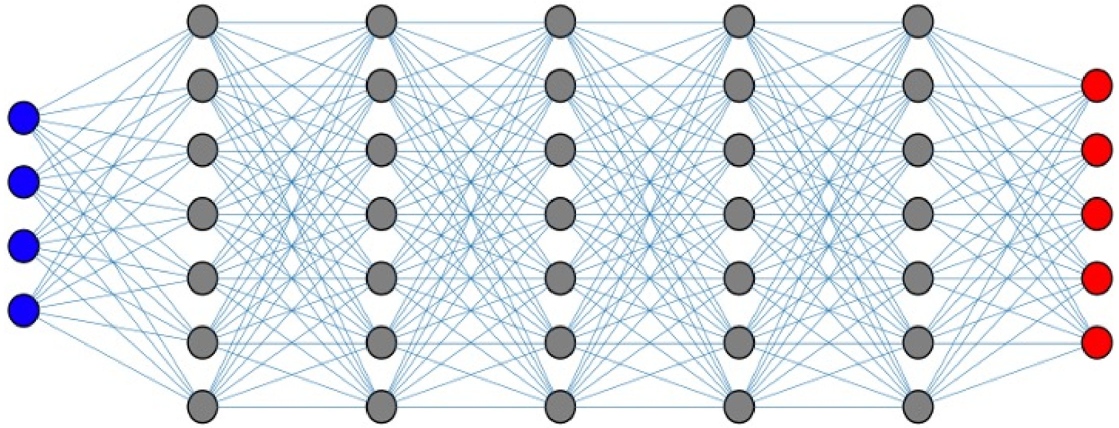
Neural ODE (continuous)

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{\sigma}(\boldsymbol{W}(t)\boldsymbol{x} + \boldsymbol{b}(t))$$

$$\boldsymbol{x}(0) = \boldsymbol{x} \qquad \boldsymbol{x}(T) = \boldsymbol{y}$$

$\boldsymbol{x} = \boldsymbol{x}_0$

Input



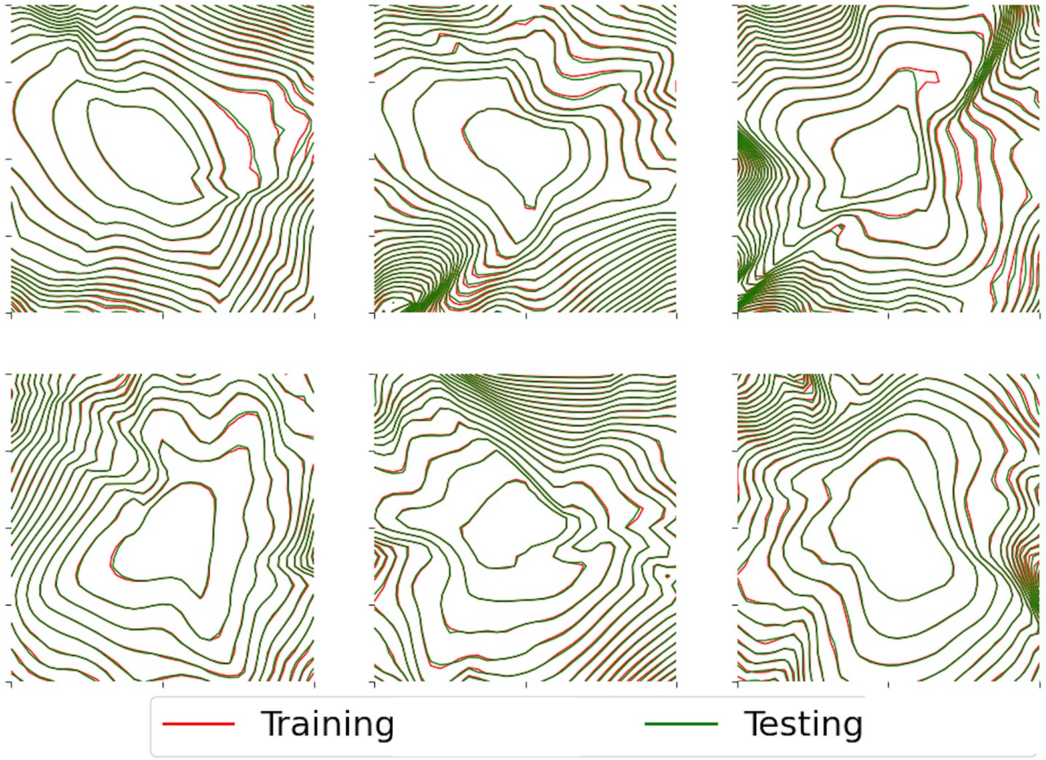
$\boldsymbol{y} = \boldsymbol{x}_T$

Output

ResNets regularize loss landscape compared to MLPs

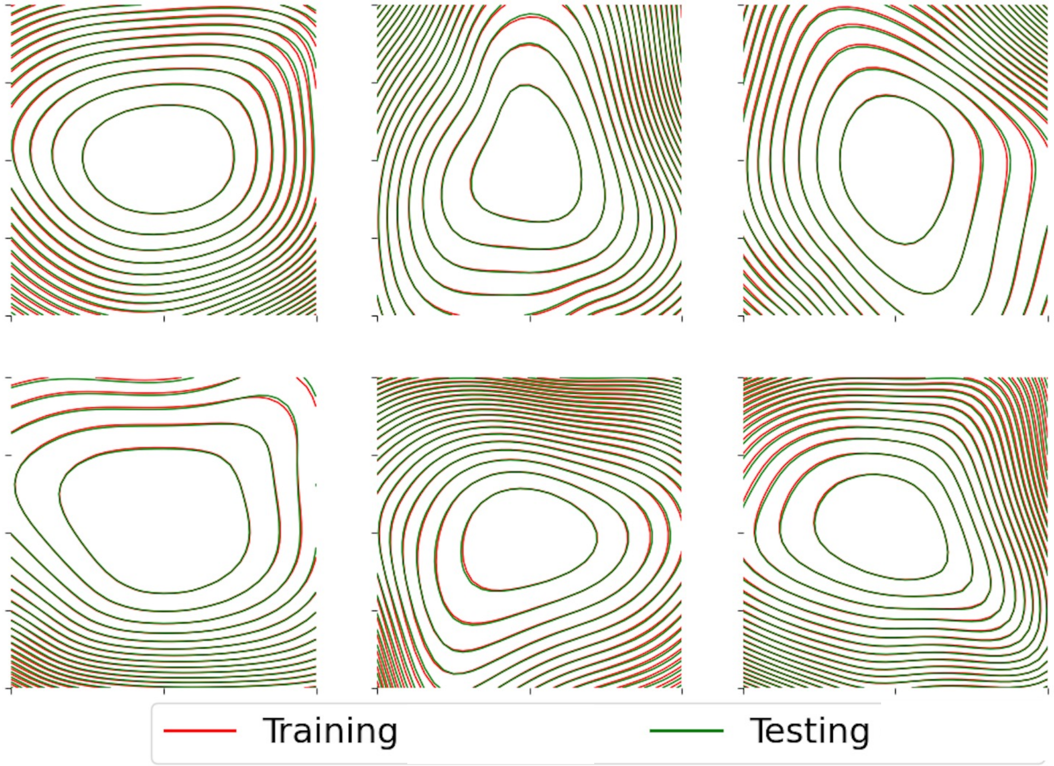
MLP NN: $x_{n+1} = \sigma(W_n x_n + b_n)$

Multilayer Perceptron (learning the layer)



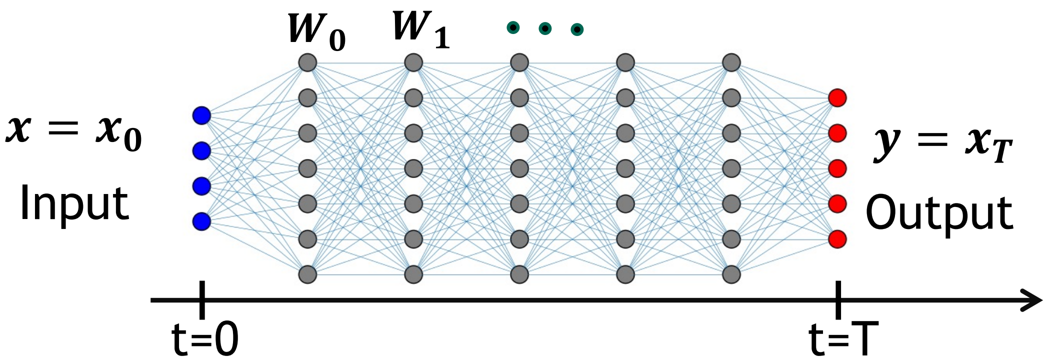
ResNet: $x_{n+1} = x_n + \alpha_n \sigma(W_n x_n + b_n)$

ResNets (learning the layer diff.)



Weight parameterization as a regularization tool, inspired by ODEs

ResNet: $x_{n+1} = x_n + \alpha_n \sigma(W_n x_n + b_n)$

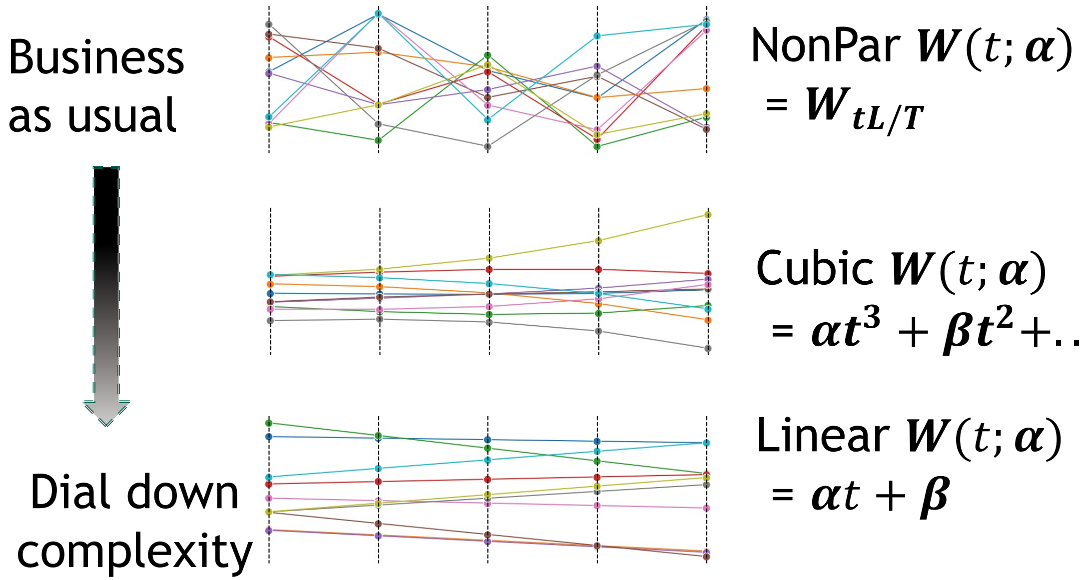


Training for weight matrices $W_0, W_1, \ldots$

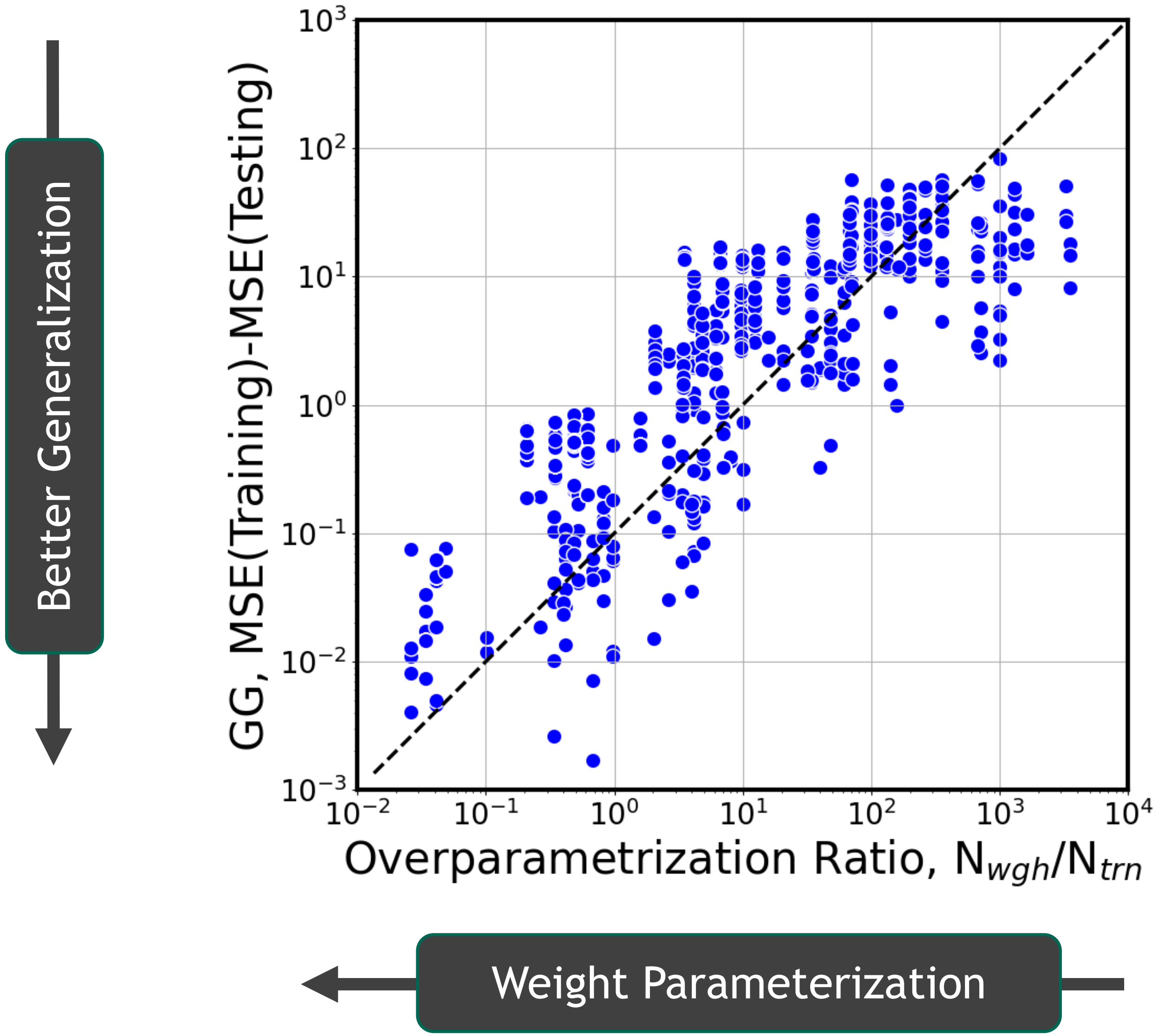
Heavily overparameterized, does not generalize well

Parameterize $W(t;\alpha)$ and train for α's.

Parameterization of weight functions reduces capacity and improves generalization



Weight parameterization improves generalization



- Generalization Gap correlates with overparameterization

- Weight-parameterized ResNets reduce Generalization Gap

Each dot is a training run with varying weight parameterization functions

Probabilistic Learning: Bayesian NN

- Conventional NN: training for <u>*deterministic*</u> weight matrices $\boldsymbol{W_0}, \boldsymbol{W_1}, \ldots$
- <u>*Probabilistic*</u> approach: training for probability distributions $\boldsymbol{p}(\boldsymbol{W_0}), \boldsymbol{p}(\boldsymbol{W_1}), \ldots$
- Three classes of options:

Full Bayesian → Approximate Bayesian → Ensemble methods

Full Bayesian

- Markov chain Monte Carlo (MCMC)
- Typically, infeasible for overparameterized NNs
- With weight parameterization loss functions are better behaved (lower-dimensional, fewer symmetries), hence MCMC path more feasible

Approximate Bayesian

- Variational methods
- Practically feasible, but many hyperparameters to tune
- Typically underestimates extrapolative predictions

Ensemble methods

- Heuristic, but works
- … but works best for complex models
- Deep ensembles, QBC…
- Many recent papers viewing deep ensembles as Bayesian approximation

QUiNN: Quantifying Uncertainties in NN

Deterministic → **Probabilistic**

torch.nn.module → **wrapper(torch.nn.module)**

Usage: →

```
uqnet = MCMC_NN(nnet)
```

```
class MCMC_NN(QUiNNBase):
    def __init__(self, nnmodule, verbose=True):
        super(MCMC_NN, self).__init__(nnmodule)
        self.verbose = verbose
```

Option 1: MCMC

```
uqnet = VI_NN(nnet)
```

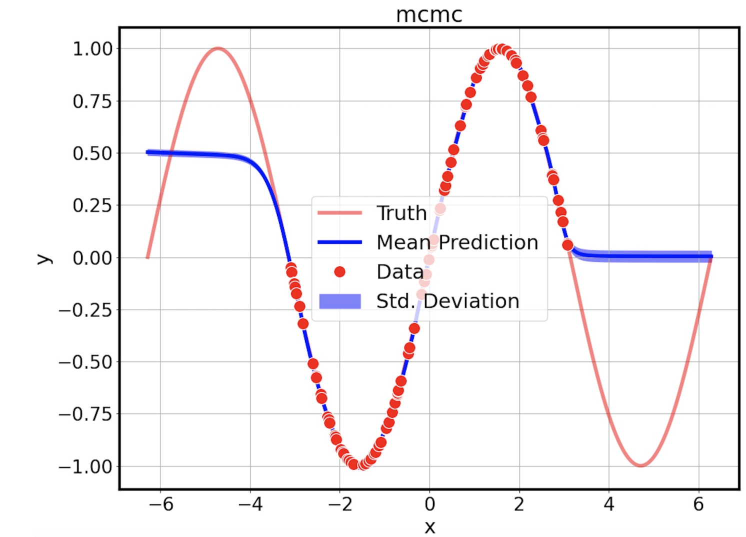
```
class VI_NN(QUiNNBase):
    def __init__(self, nnmodule, verbose=False):
        super(VI_NN, self).__init__(nnmodule)
        self.bmodel = BNet(nnmodule)
        self.verbose = verbose
```

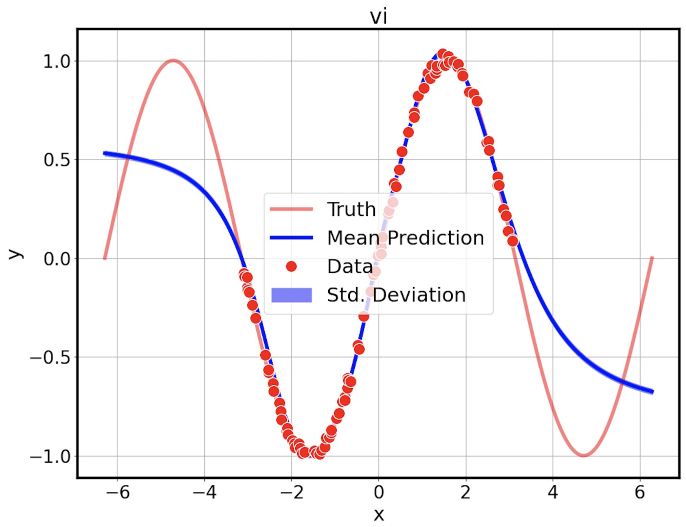
Option 2: Variational Inference

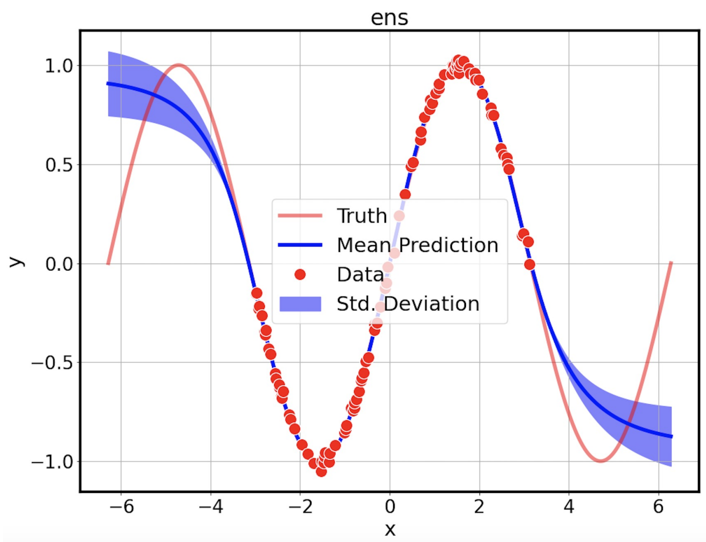
```
uqnet = Ens_NN(nnet, nens=nmc)
```

```
class Ens_NN(QUiNNBase):
    def __init__(self, nnmodule, nens=1, verbose=False):
        super(Ens_NN, self).__init__(nnmodule)
        self.verbose = verbose
        self.nens = nens
```

Option 3: Ensembling





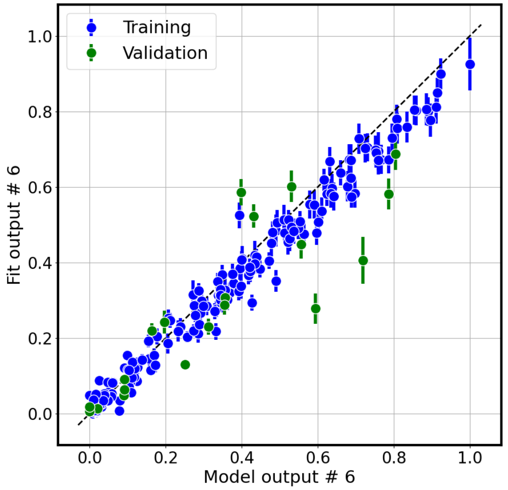


Apps:

- Multiple applications are informing the development of foundational research
- None of these applications have been previously exposed to NN prediction uncertainties, particularly in the context of ResNets and weight parameterization

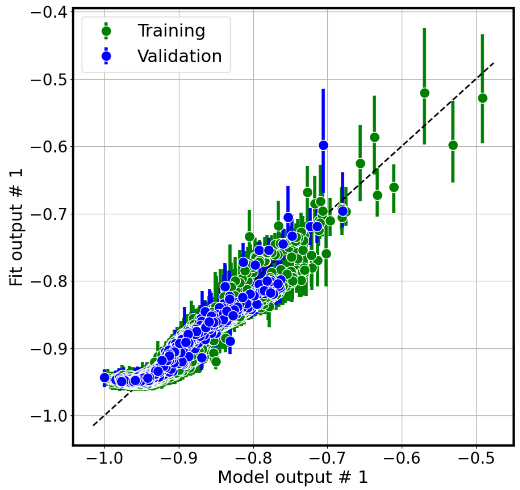
E3SM Vegetation Dynamics

- 15 input parameters
- 10 static output QoIs
- 2K training simulations



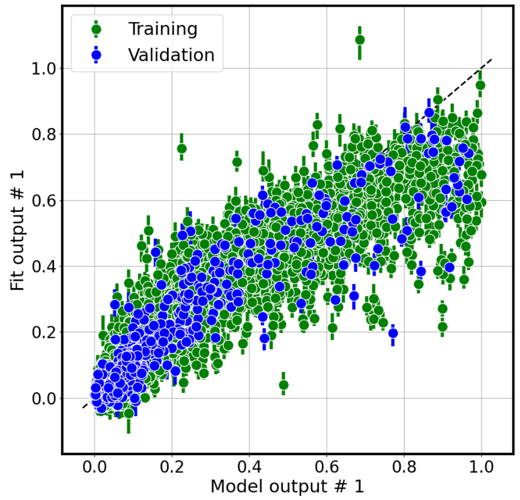
FitSNAP Entropy Dataset

- 30 input bases
- 1 output (Energy/Force/Stress)
- 20K training DFT simulations



CO-on-Pt(111) Adsorbate

- 6 input d.o.f.
- 1 output (Energy)
- 10K training DFT simulations



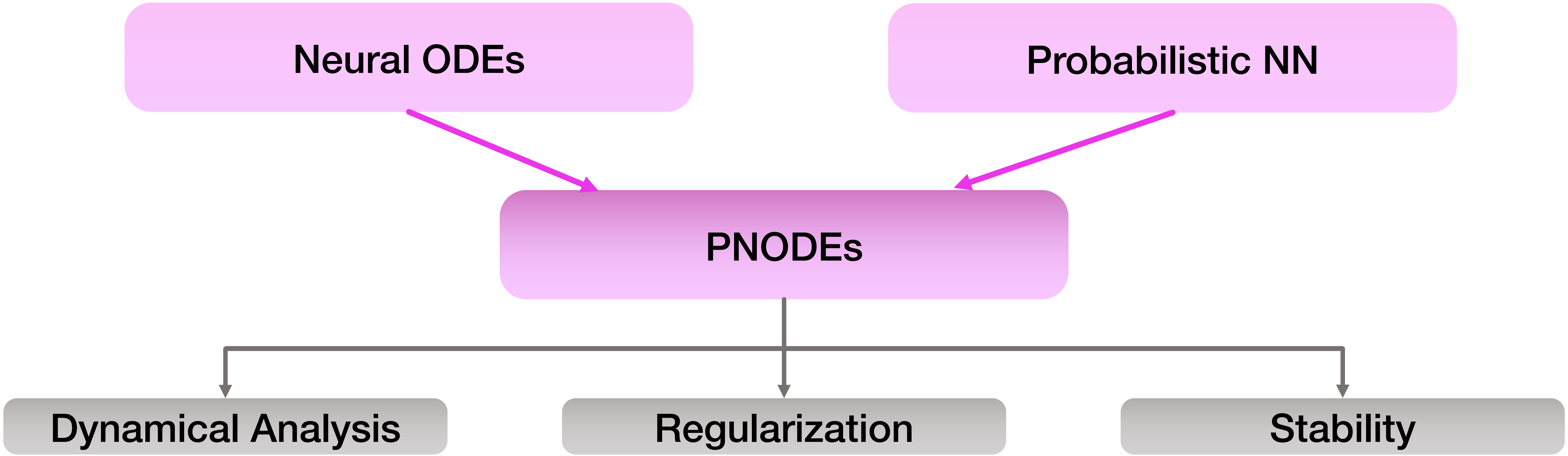
Summary

- Focus on ResNets and draw inspiration from ODEs
- ResNets regularize the learning problem, smoother loss surface
- **Weight parameterization** allows further regularization

- **Probabilistic approaches** more feasible with weight-parameterized ResNets
- Need to find sweet spot between empirical to fully Bayesian, and ….
- … conventional mean-field variation inference isn’t that.

Extra Materials

Analysis of Neural Networks as Random Dynamical Systems



Dynamical Analysis

- Singular perturbation
- Stiffness
- Model reduction
- Non-local interactions

Regularization

- Random field parameterization of weights
- Enforce structure: smoothness, sparsity, low-rank

Stability

- Robustness with noise
- Eigenvalue structure under uncertainty

Generalizable model; improved architecture; confidence assessment; robustness to noise

Foundational capabilities impacting multiple applications

Predictive capability of Neural Networks (NNs) hinges on **generalization** (ability to predict well outside training data).

Regularization of NNs as a way to achieve generalization.

- ✓ Stiffness Penalization
- ✓ Weight Parameterization
- ✓ Probabilistic Weights

Methods

- ✓ Climate Land Modeling
- ✓ Catalytic Chemistry
- ✓ Materials Science

Applications

DTO vs OTD

ResNet

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{\alpha}_n \boldsymbol{\sigma}(\boldsymbol{W}_n \boldsymbol{x}_n + \boldsymbol{b}_n)$$

NODE

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{\sigma}(\boldsymbol{W}(t)\boldsymbol{x} + \boldsymbol{b}(t))$$

Forward equivalence:

Neural ODE discretized using explicit Euler and ResNet produce identical outputs choosing time step: $\Delta t = \frac{T}{L}$, $\alpha_n := \Delta t, \boldsymbol{W}_n := \boldsymbol{W}(n\Delta t)$ and $\boldsymbol{b}_n := \boldsymbol{b}(n\Delta t)$ for all n.

Backward not so much:

Gradient computations differ!

Consider $W(t) \equiv W$ and $b \equiv 0$:

Discretized Neural ODE with adjoint method:

$$\nabla loss = 2\big((1 + \delta t\, W)^L x - y\big)(1 + \delta t\, W)^{L} x$$

ResNet with backpropagation:

$$\nabla loss = 2\big((1 + \delta t\, W)^L x - y\big)(1 + \delta t\, W)^{L-1} x$$

- Gradients converge as $L \to \infty$ but differences can be large for small L,
- Optimize then discretize (adjoint method) $\neq$ discretize then optimize (backpropagation).

Prior Work on Probabilistic NN

- **Probabilistic NN** have been around since 90s [*MacKay, 1992; Neal, 1997*]
  - Full probabilistic treatment was infeasible back then (and still is, generally)
  - Recent work showed avenues via variational methods with clever tricks:

    Bayes by Backprop [*Blundell, 2015*]; Probabilistic backprop [*Hernandez-Lobato 2015*]
  - *Ghahramani, "Probabilistic Machine Learning and Artificial Intelligence". Nature, 2015*
    - *"Nearly all approaches to probabilistic programming are Bayesian since*

      it is hard to create other coherent frameworks for automated reasoning about uncertainty"
  - Industry ***is*** catching up: Bayesflow at Google, infer.NET at Microsoft, Uber has shown interest
  - Still not industry-standard: expensive, not well understood.