Neural networks basics

by C. Yao Lai for 2025 Glamacles summer school

List compiled by Josh and Yao

ML for cryosphere observations

We would love to know what you're working on that can be added to the list!

40+ papers since 2019!

Data creation (continent-wide dataset marked by *):

- Icebergs (Barbat et al. 2019*, Rezvanbehbahani, Stearns et al. 2020)
- Sea ice (Cooke et al. 2019, Landy et al. 2022*)
- Calving front /termini (Baumhoer et al. 2019, Mohajerani et al. 2019, Zhang et al. 2019, CALFIN: Cheng et al. 2021, AutoTerm: Zhang et al. 2023*…etc)
- Fractures (Ice-shelf: Lai et al. 2020*, Ice sheet: Chudley et al. 2021, Ice shelf + sheet: Surawy-Stepney et al. 2023…etc)
- Bed topography (DeepBedMap: Leong & Horgan 2020*)
- Supraglacial water – Antarctica (Dirscherl et al. 2020, Dell et al. 2021, Dell et al. 2024)
- Supraglacial water – Greenland (Dunmire et al. 2021*, Dunmire et al. 2025*)
- Internal ice layers using radar data (Rahnemoonfar et al. 2021)
- Grounding lines using InSAR data (Mohajerani et al. 2021)
- Greenland firn aquifers (Shang et al. 2022*)
- Blue ice- Antarctica (Tollenaar et al. 2024*)

List compiled by Yao

We would love to know what you're working on that can be added to the list!

ML for cryosphere modeling

30+ papers since 2021!

- **Surrogate/emulation model (forward modeling):**
  GP (Edwards et al. 2021, Hill et al. 2024)
  CNN (IGM: Jouvet et al. 2021, 2023a,b, Verjans & Robel 2024)
  DeepONet (Howard, Perego et al. 2023, He, Perego et al. 2023)
  LSTM (Katwyk et al. 2023)
  GNN (Rahnemoonfar and Koo 2025)
- **Data assimilation (inverse modeling):**
  Bayesian inference + NN (Brinkerhoff et al. 2021)
  PINN (Riel, Michew et al. 2021, Diffice: Wang, Lai et al. 2022, 2025, Cheng, Morlighem et al. 2024, Steidl, Bamber et al. 2025)
  GP (Downs et al. 2022, Brinkerhoff et al. 2024)
  UDE (ODINN: Bolibar, Sapienza et al. 2023)
  CNN (DeepBedMap: Leong & Horgan 2020, Jouvet 2023a,b, DeepTopoNet: Tama et al. 2025)

We're bad at predicting the future

The 7 Worst Tech Predictions of All Time, By Robert Strohmeyer
https://www.pcworld.com/article/532605/worst_tech_predictions.html

1. "I think there is a world market for maybe five computers." Thomas Watson, president, IBM, 1943
2. "Television won't be able to hold on to any market it captures after the first six months. People will soon get tired of staring at a plywood box every night." Darryl Zanuck, executive at 20th Century Fox, 1946
3. "Nuclear-powered vacuum cleaners will probably be a reality within ten years." Alex Lewyt, president of Lewyt vacuum company, 1955
4. "There is no reason anyone would want a computer in their home." Ken Olsen, founder of Digital Equipment Corporation, 1977
5. "Almost all of the many predictions now being made about 1996 hinge on the Internet's continuing exponential growth. But I predict the Internet will soon go spectacularly supernova and in 1996 catastrophically collapse." Robert Metcalfe, founder, 3Com, 1995
6. "Apple is already dead." Nathan Myhrvold, former Microsoft CTO, 1997

Outline

- Why neural networks?
- NN basics:
  Universal function approximation
  Gradient descent
  Back propagation
- Advanced topics: challenges of using NN in scientific problems
- Coding exercise

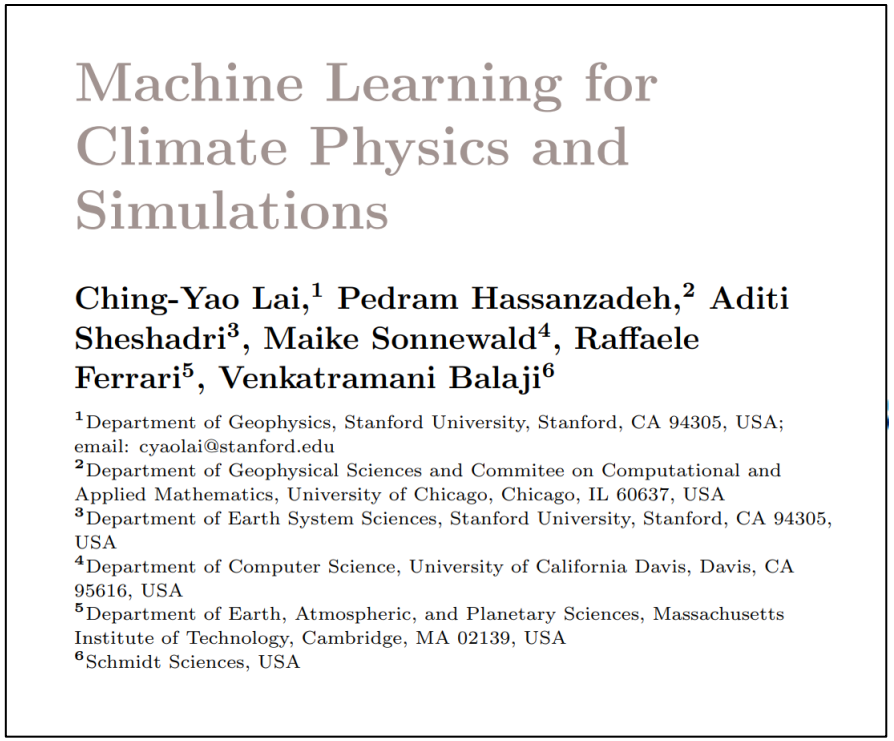
So…why neural networks?

- Very often in science we want to know A as a function of B. NN has the flexibility to fit that function without prior assumptions about the functional form, particularly useful when the function is high dimensional (it is a universal function approximator).

- Leverages GPU (hardware) and open-source software (e.g. tensoflow, Pytorch, JAX). The optimization method is well-developed and user friendly. It can usually be easily adopted for different problems.

- We will talk about the challenges after learning about the basics of NN!

Definitions that may be useful



Machine Learning for Climate Physics and Simulations

Ching-Yao Lai,[1] Pedram Hassanzadeh,[2] Aditi Sheshadri[3], Maike Sonnewald[4], Raffaele Ferrari[5], Venkatramani Balaji[6]

[1]Department of Geophysics, Stanford University, Stanford, CA 94305, USA; email: cyaolai@stanford.edu
[2]Department of Geophysical Sciences and Commitee on Computational and Applied Mathematics, University of Chicago, Chicago, IL 60637, USA
[3]Department of Earth System Sciences, Stanford University, Stanford, CA 94305, USA
[4]Department of Computer Science, University of California Davis, Davis, CA 95616, USA
[5]Department of Earth, Atmospheric, and Planetary Sciences, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
[6]Schmidt Sciences, USA

Lai et al., Annu. Rev. Condens. Matter Phys. (2025). doi.org/10.1146/annurev-conmatphys-043024-114758

Simulation: Physics-based models solved numerically. For example the General Circulation Model is a climate simulation that solves the Navier-Stokes equation…etc.

Emulator: In this article emulator refers to a subset of models that fit the data, bypassing solving physics-based equations.

Definitions that may be useful

A nonlinear partial differential equation:

$$\frac{\partial u(x,t)}{\partial t} = N\big(u(x,t), \theta(x,t)\big)$$

Forward problem:
Given initial/boundary conditions, model parameter $\theta(x,t)$, find $u(x,t)$

Inverse problem:
Given $u(x,t)$, find model parameter $\theta(x,t)$

Definitions that may be useful

A nonlinear partial differential equation:

$$\frac{\partial u(x,t)}{\partial t} = \nabla \cdot \left(D(u,x) \nabla u(x,t) \right)$$

Forward problem:
Given initial/boundary conditions, model parameter $D(u,x)$, find $u(x,t)$

Inverse problem:
Given $u(x,t)$, find model parameter $D(u,x)$

Surrogate/emulation (forward in time)

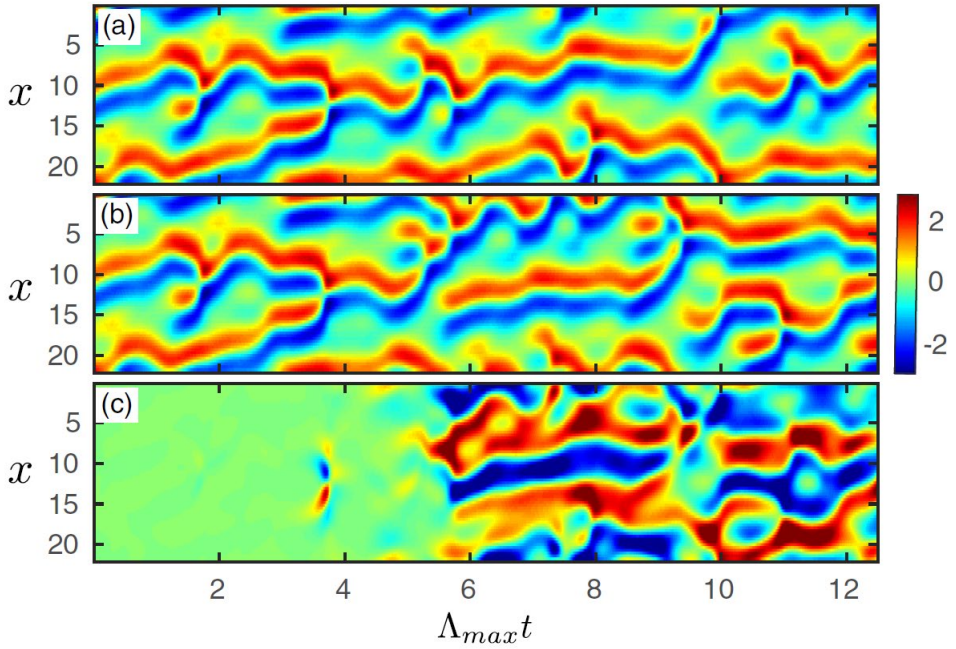
Kuramoto-Sivashinsky (KS)

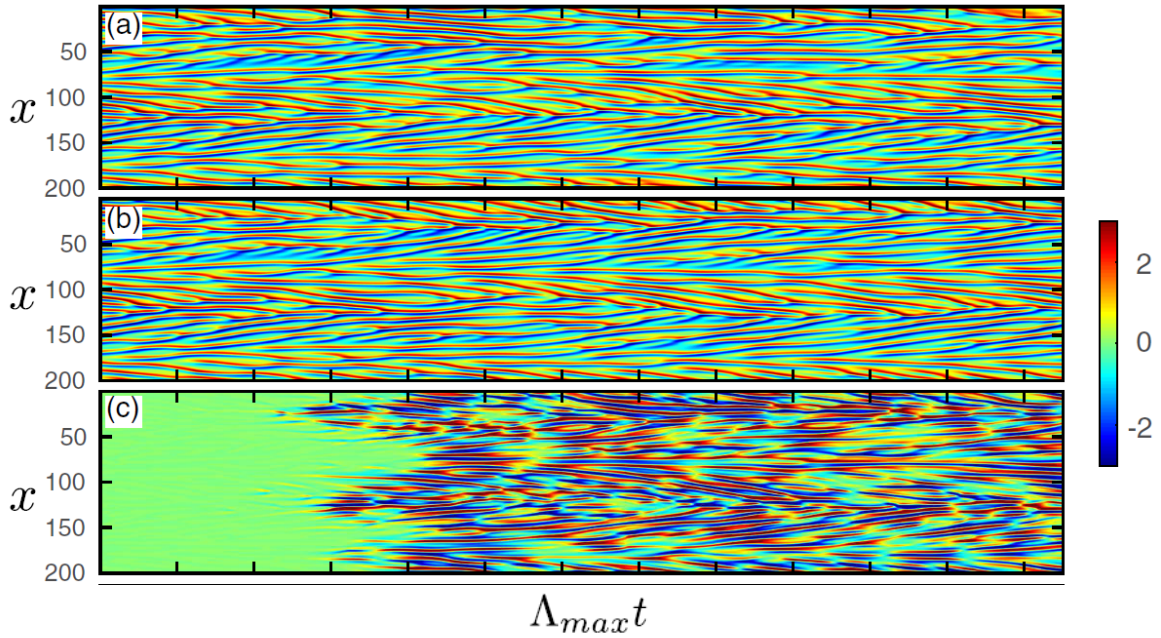
$$y_t = -yy_x - y_{xx} - y_{xxxx} + \mu \cos\left(\frac{2\pi x}{\lambda}\right)$$

PHYSICAL REVIEW LETTERS **120,** 024102 (2018)

Model-Free Prediction of Large Spatiotemporally Chaotic Systems from Data: A Reservoir Computing Approach

Jaideep Pathak,[1,2,*] Brian Hunt,[3,4] Michelle Girvan,[1,3,2] Zhixin Lu,[1,3] and Edward Ott[1,2,5]
[1]*Institute for Research in Electronics and Applied Physics, University of Maryland, College Park, Maryland 20742, USA*





Physics-based simulations of weather

Model of weather:

$$\frac{\partial \boldsymbol{z}(x,t)}{\partial t} = f(x,t), \quad \text{where} \quad x \in \mathbb{R}^d \quad \text{and} \quad z, f \in \mathbb{R}^n,$$

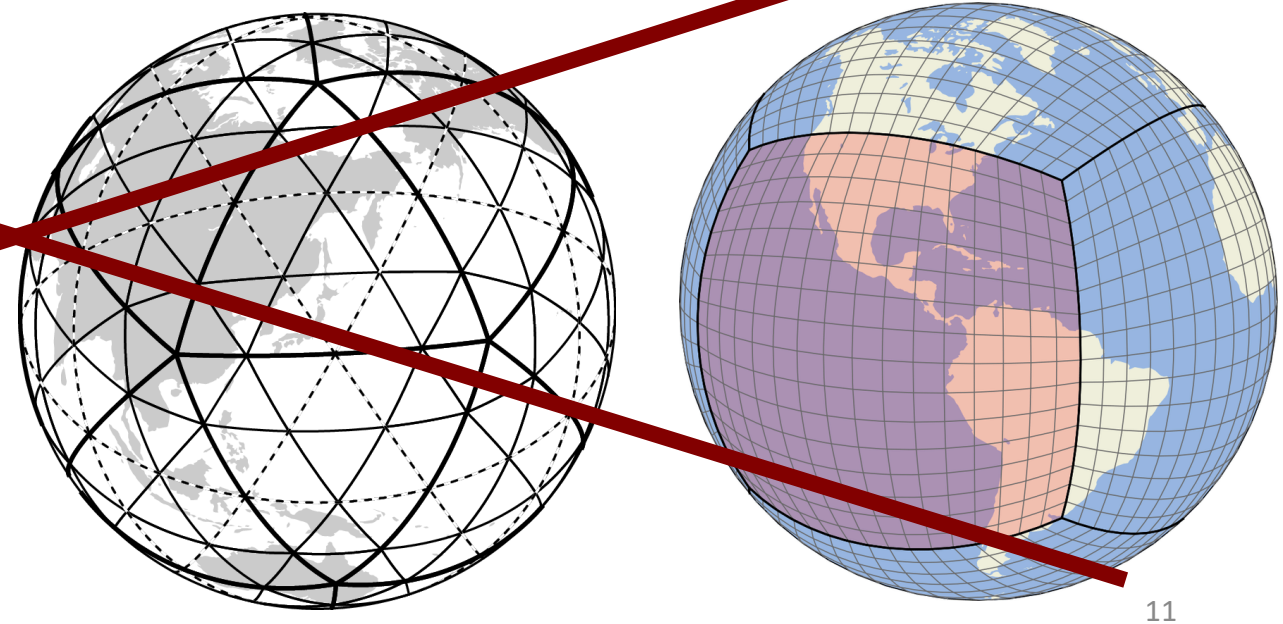
Governing Equations in physics-based simulations of weather prediction model

$$\frac{D\vec{u}}{Dt} + 2\Omega \times \vec{u} = -\frac{1}{\rho}\nabla p - \nabla\Phi + \vec{F} \quad \text{(Momentum)}$$

$$\frac{\partial \rho}{\partial t} + \nabla.(\rho\vec{u}) = 0 \quad \text{(Mass)}$$

$$p = \rho RT \quad \text{(Ideal Gas Law)}$$

$$c_p \frac{DT}{Dt} - \frac{1}{\rho}\frac{Dp}{Dt} = \dot{Q} \quad \text{(Energy)}$$



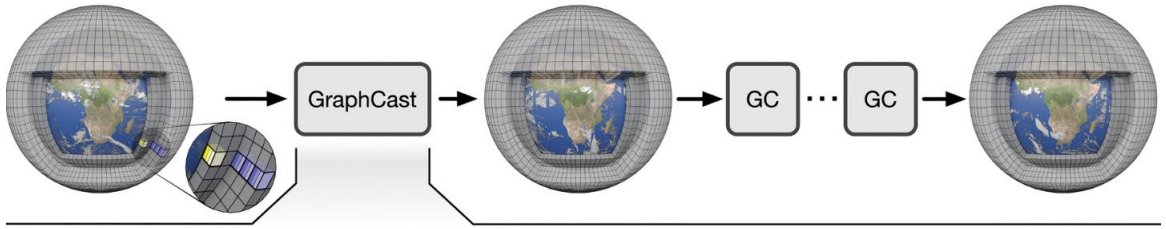
Bauer, P., Thorpe, A. & Brunet, G. The quiet revolution of numerical weather prediction. *Nature* **525**, 47–55 (2015).

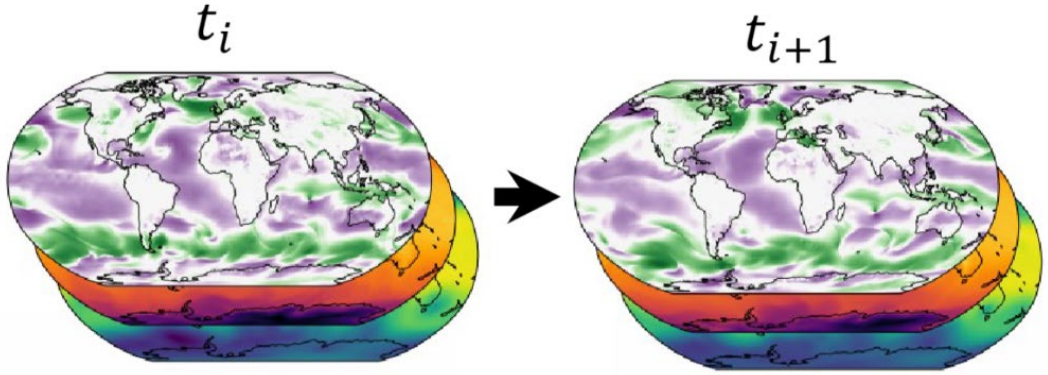
Surrogate/emulation (forward in time)

Model of weather:

$$\frac{\partial \boldsymbol{z}(x,t)}{\partial t} = f(x,t), \quad \text{where} \quad x \in \mathbb{R}^d \quad \text{and} \quad z, f \in \mathbb{R}^n,$$

Find θ such that $\mathrm{ML}(\theta) = f(x,t)$, mapping the rate of change of states $\boldsymbol{z}(x,t)$ (ML: $z(x,t_i) \mapsto z(x,t_{i+1})$)





Deepmind GraphCast model (Lam et al, Science 2023) trained a graph neural network on physics-based simulations of weather prediction model to learn the mapping of states of the global atmosphere from t_i to t_{i+1}, then from t_{i+1} to t_{i+2}, and so on.

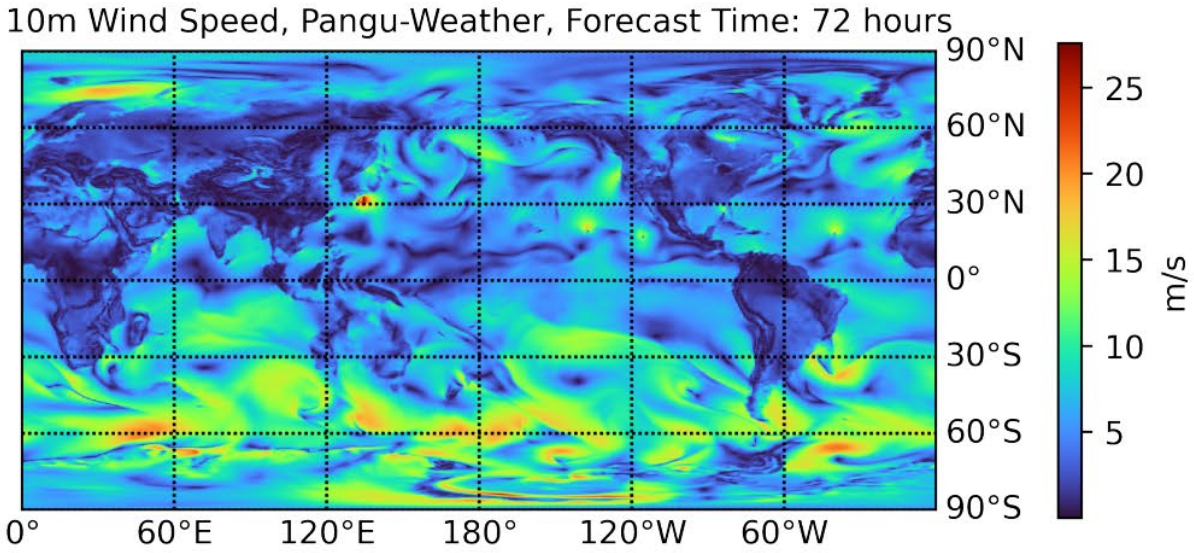
ML weather forecast models

ECMWF is now running a series of data-driven forecasts as part of its experimental suite. These machine-learning based models are very fast, and they produce a 10-day forecast with 6-hourly time steps in approximately one minute. The outputs are available in graphical form.

Neural networks are used to emulate the weather (Pangu-Weather: Bi et al. (2023), Nature)

Which one is real?

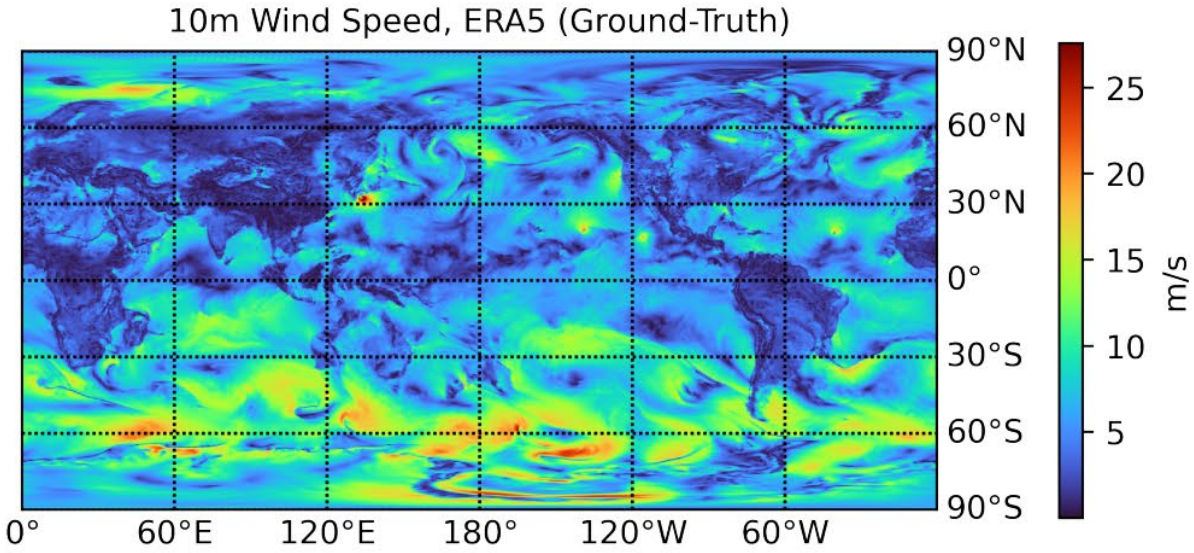
A



AI weather forecast

Bi et al (2023) Nature

B



Training data

(Numerical weather prediction simulation)

ML for cryosphere modeling

30+ papers since 2021!

- **Surrogate/emulation model (forward modeling):**
  GP (Edwards et al. 2021, Hill et al. 2024)
  CNN (IGM: Jouvet et al. 2021, 2023a,b, Verjans & Robel 2024)
  DeepONet (Howard, Perego et al. 2023, He, Perego et al. 2023)
  LSTM (Katwyk et al. 2023)
  GNN (Rahnemoonfar and Koo 2025)
- **Data assimilation (inverse modeling):**
  Bayesian inference + NN (Brinkerhoff et al. 2021)
  PINN (Riel, Michew et al. 2021, Diffice: Wang, Lai et al. 2022, 2025, Cheng, Morlighem et al. 2024, Steidl, Bamber et al. 2025)
  GP (Downs et al. 2022, Brinkerhoff et al. 2024)
  UDE (ODINN: Bolibar, Sapienza et al. 2023)
  CNN (DeepBedMap: Leong & Horgan 2020, Jouvet 2023a,b, DeepTopoNet: Tama et al. 2025)

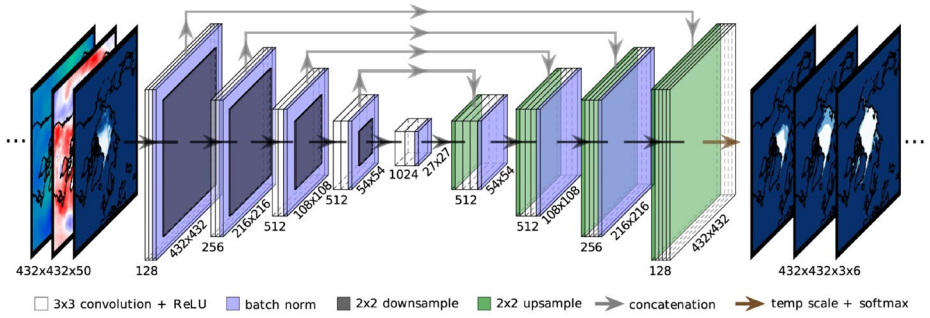
Surrogate/emulation (forward in time)

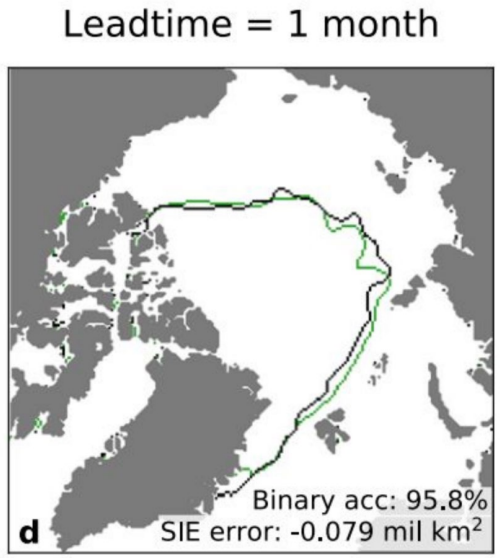
- CNN learns the input-output relationship from lots of training data.
- Input = ? Output = ?

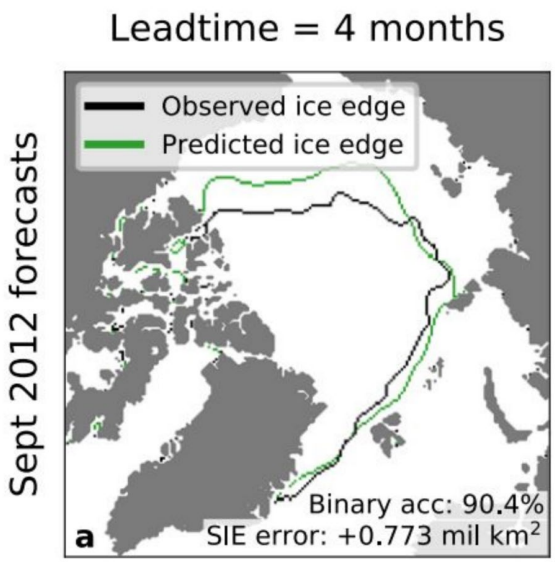
Seasonal Arctic sea ice forecasting with probabilistic deep learning

Tom R. Andersson, J. Scott Hosking, María Pérez-Ortiz, Brooks Paige, Andrew Elliott, Chris Russell, Stephen Law, Daniel C. Jones, Jeremy Wilkinson, Tony Phillips, James Byrne, Steffen Tietsche, Beena Balan Sarojini, Eduardo Blanchard-Wrigglesworth, Yevgeny Aksenov, Rod Downie & Emily Shuckburgh

Nature Communications **12**, Article number: 5124 (2021) | Cite this article







Input: 50 different climate variables (e.g. temperature, wind speed…etc)
Output: 3 sea ice concentration classes in 6 forecast months

Surrogate/emulation (between variables)

Model of glaciers:

Mass balance: $$\frac{\partial h}{\partial t} + \nabla \cdot (\bar{\mathbf{u}} h) = \mathrm{SMB}$$

Momentum balance:

$$\partial_x(4h\eta\partial_x u + 2h\eta\partial_y v) + \partial_y(h\eta(\partial_x v + \partial_y u)) - \beta^2 u = \rho g h(\partial_x s \cos\alpha - \sin\alpha),$$
$$\partial_y(4h\eta\partial_y v + 2h\eta\partial_x u) + \partial_x(h\eta(\partial_y u + \partial_x v)) - \beta^2 v = \rho g h \partial_y s \cos\alpha.$$

$$\beta = c^{-1/2m} |\boldsymbol{v}_b|^{\frac{1-m}{2m}}$$

Expensive!

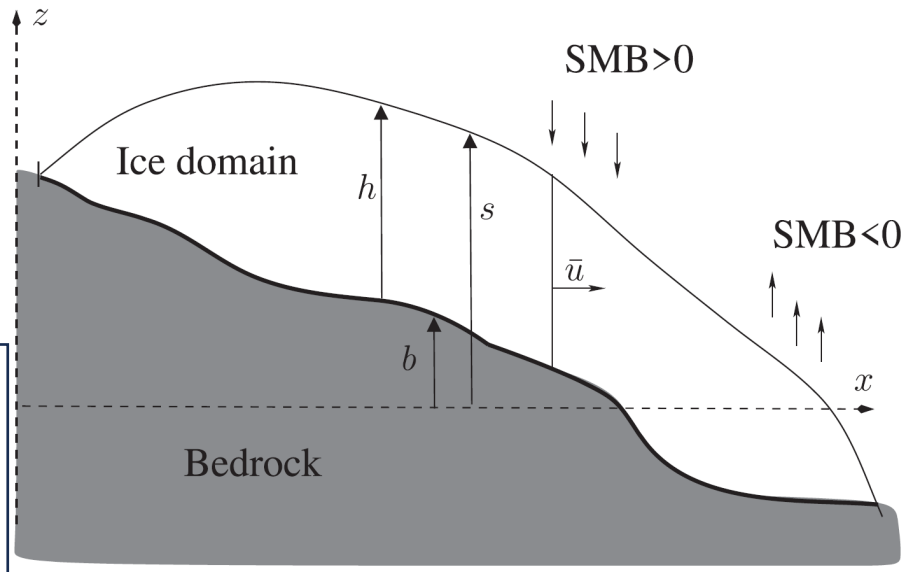


Fig. 2. Cross-section of a glacier with notations.

$$\mathcal{M}: \quad \left\{h, \frac{\partial s}{\partial x}, \frac{\partial s}{\partial y}, c\right\} \longrightarrow \{\bar{u}, \bar{v}\}$$
$$\mathbb{R}^{N_X \times N_Y \times 4} \longrightarrow \mathbb{R}^{N_X \times N_Y \times 2}$$

Surrogate/emulation (between variables)

Model of glaciers:

Mass balance: $$\frac{\partial h}{\partial t} + \nabla \cdot (\bar{\mathbf{u}} h) = \mathrm{SMB}$$

Momentum balance (2D; in x,y):

$$\partial_x(4h\eta\partial_x u + 2h\eta\partial_y v) + \partial_y(h\eta(\partial_x v + \partial_y u)) - \beta^2 u = \rho g h(\partial_x s \cos\alpha$$
$$\partial_y(4h\eta\partial_y v + 2h\eta\partial_x u) + \partial_x(h\eta(\partial_y u + \partial_x v)) - \beta^2 v = \rho g h \partial_y s \cos\alpha.$$

Expensive!

$$\beta = c^{-1/2m} |\boldsymbol{v}_b|^{\frac{1-m}{2m}}$$

$$\mathcal{M}: \quad \left\{h, \frac{\partial s}{\partial x}, \frac{\partial s}{\partial y}, c\right\} \longrightarrow \{\bar{u}, \bar{v}\}$$
$$\mathbb{R}^{N_X \times N_Y \times 4} \longrightarrow \mathbb{R}^{N_X \times N_Y \times 2}$$

Cheap!

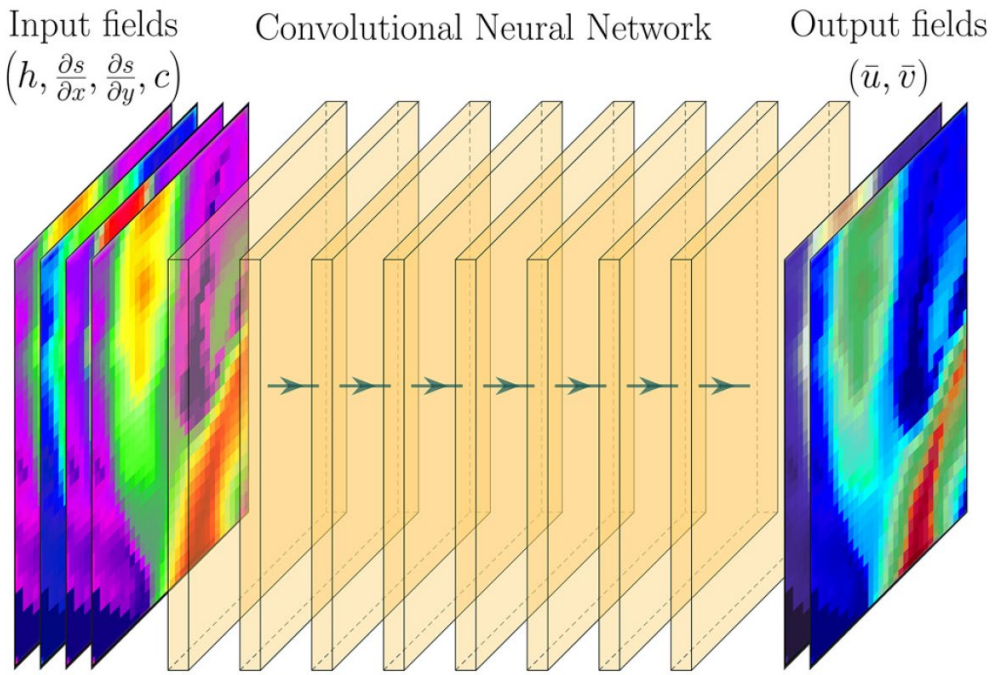


Fig. 3. The function we aim to emulate by learning from hybrid SIA + SSA or Stokes realizations maps geometrical fields (thickness and surface slopes) and basal sliding parametrization to ice flow fields.

Tutorial: https://vimeo.com/884003820

Surrogate/emulation (between variables)

Model of glaciers:

Mass balance: $$\frac{\partial h}{\partial t} + \nabla \cdot (\bar{\mathbf{u}} h) = \mathrm{SMB}$$

Momentum balance (3D; in x,y,z):

$$-\nabla \cdot \boldsymbol{\sigma} = \rho \mathbf{g}, \quad \text{in} \quad V, \qquad \boldsymbol{\sigma} = \boldsymbol{\tau} - PI, \quad \boldsymbol{\tau} = 2\mu D(\mathbf{u}),$$

$$D(\mathbf{u}) = \begin{pmatrix} \partial_x u_x & \frac{1}{2}(\partial_y u_x + \partial_x u_y), & \frac{1}{2}(\partial_z u_x) \\ \frac{1}{2}(\partial_y u_x + \partial_x u_y) & \partial_y u_y & \frac{1}{2}(\partial_z u_y) \\ \frac{1}{2}(\partial_z u_x) & \frac{1}{2}(\partial_z u_y) & -\partial_x u_x - \partial_y u_y \end{pmatrix}$$

$$\boldsymbol{\sigma} \cdot \mathbf{n} = 0, \qquad P = 0, \qquad \text{on} \quad \Gamma_s$$

$$[(I - \mathbf{n}\mathbf{n}^T)\tau] \cdot \mathbf{n} = -c^{-m} |(I - \mathbf{n}\mathbf{n}^T) \cdot \mathbf{u}|^{m-1} (I - \mathbf{n}\mathbf{n}^T) \cdot \mathbf{u}$$

$$\rightarrow \quad \mathcal{N}_\lambda : \{h_H, s_H, A_H, c_H, H_H\} \longrightarrow \{\mathbf{u}_H, \mathbf{v}_H\}$$

$$\mathbb{R}^{N_X \times N_Y \times 5} \longrightarrow \mathbb{R}^{N_X \times N_Y \times N_Z \times 2}$$

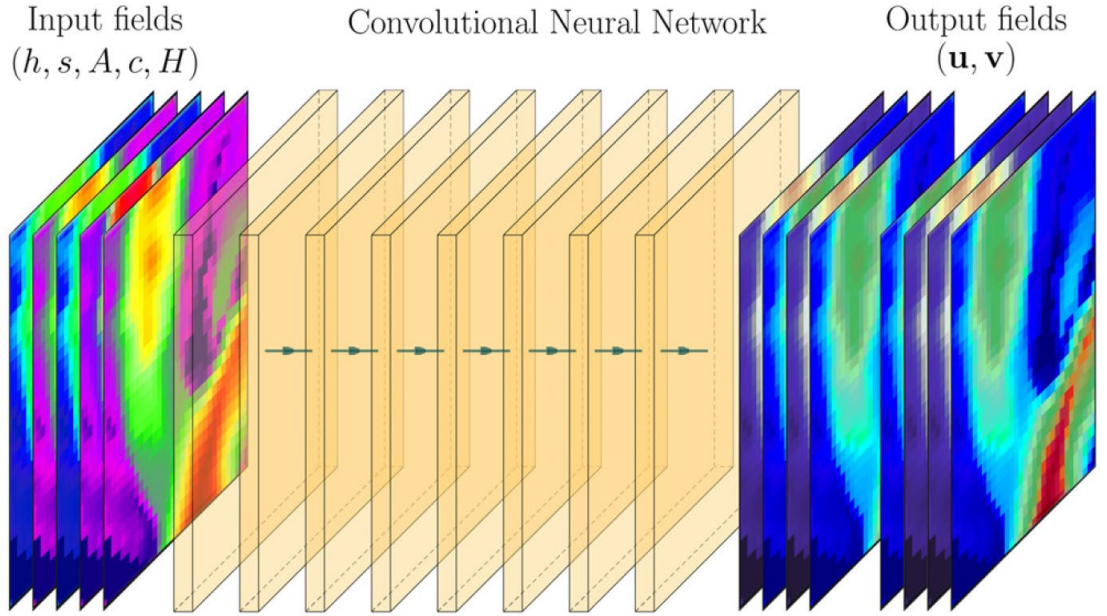


Figure 3. Our emulator consists of a CNN that maps geometrical (thickness and surface topography), ice-flow parameters (shearing and basal sliding) and spatial resolution inputs to 3D ice-flow fields.

Tutorial: https://vimeo.com/884003820

Surrogate/emulation (between variables)

Model of glaciers:

Mass balance: $$\frac{\partial h}{\partial t} + \nabla \cdot (\bar{\mathbf{u}} h) = \text{SMB}$$

Momentum balance (3D; in x,y,z):

$$-\nabla \cdot \boldsymbol{\sigma} = \rho \mathbf{g}, \text{ in } V, \quad \boldsymbol{\sigma} = \boldsymbol{\tau} - PI, \quad \boldsymbol{\tau} = 2\mu D(\mathbf{u}),$$

$$D(\mathbf{u}) = \begin{pmatrix} \partial_x u_x & \frac{1}{2}(\partial_y u_x + \partial_x u_y), & \frac{1}{2}(\partial_z u_x) \\ \frac{1}{2}(\partial_y u_x + \partial_x u_y) & \partial_y u_y & \frac{1}{2}(\partial_z u_y) \\ \frac{1}{2}(\partial_z u_x) & \frac{1}{2}(\partial_z u_y) & -\partial_x u_x - \partial_y u_y \end{pmatrix}$$

$$\boldsymbol{\sigma} \cdot \mathbf{n} = 0, \quad P = 0, \quad \text{on } \Gamma_s$$

$$[(I - \mathbf{n}\mathbf{n}^T)\tau] \cdot \mathbf{n} = -c^{-m} |(I - \mathbf{n}\mathbf{n}^T) \cdot \mathbf{u}|^{m-1} (I - \mathbf{n}\mathbf{n}^T) \cdot \mathbf{u}$$

$$\rightarrow \quad \mathcal{N}_\lambda : \{h_H, s_H, A_H, c_H, H_H\} \longrightarrow \{\mathbf{u}_H, \mathbf{v}_H\}$$

$$\mathbb{R}^{N_X \times N_Y \times 5} \longrightarrow \mathbb{R}^{N_X \times N_Y \times N_Z \times 2}$$

Physics-informed:

CNN $\lambda = \{\lambda_i, \ i = 1, \ \ldots, \ N\}$ that minimise:

$$\lambda = \text{argmin}\left(\mathcal{J}_{p_H}(\mathcal{N}_\lambda(p_H))\right), \tag{20}$$

$$\mathcal{J}_{p_H}(\mathbf{v}_H) = \int_\Omega \left(\frac{2A_H^{-\frac{1}{n}}}{1+\frac{1}{n}} \int_{s_H - h_H}^{s_H} |D_H(\mathbf{v}_H)|^{1+\frac{1}{n}} \, dz \right.$$
$$+ \frac{c_H^{-m}}{1+m} |\mathbf{v}_H|_M^{1+m} \, dS \tag{18}$$
$$\left. + \rho g \int_{s_H - h_H}^{s_H} (\nabla s_H \cdot \mathbf{v}_H) \, dz \right) d\Omega.$$

JAMES | Journal of Advances in Modeling Earth Systems

Research Article | Open Access

A Variational LSTM Emulator of Sea Level Contribution From the Antarctic Ice Sheet

Peter Van Katwyk, Baylor Fox-Kemper, Hélène Seroussi, Sophie Nowicki, Karianne J. Bergen

Plain Language Summary

An emulator is a computational model that is designed to rapidly approximate another more complex, computationally intensive model. In this work, we propose a new emulator of the Antarctic ice sheet that provides faster and more accurate projections of the future sea level rise due to the melting of the ice sheet. This new emulator uses a technology called neural networks (NNs), a common tool within the field of artificial intelligence. We show that the NN-based emulator produces accurate sea level projections while also being able to quantify the inherent uncertainty associated with future sea level rise. With these improvements, this emulator will enable scientists to better investigate the ice sheet itself and learn how it connects to Earth's dynamic behavior. In addition, due to the efficiency of the proposed emulator, not only are many avenues of research opened for further improvements of ice sheet emulators, but also a wider range of scientists will be able to contribute to future development. This will lead to a better understanding of the drivers of sea level rise as well as more informed decision-making and mitigation efforts.

ML for cryosphere modeling

30+ papers since 2021!

- **Surrogate/emulation model (forward modeling):**
  GP (Edwards et al. 2021, Hill et al. 2024)
  CNN (IGM: Jouvet et al. 2021, 2023a,b, Verjans & Robel 2024)
  DeepONet (Howard, Perego et al. 2023, He, Perego et al. 2023)
  LSTM (Katwyk et al. 2023)
  GNN (Rahnemoonfar and Koo 2025)

- **Data assimilation (inverse modeling):**
  Bayesian inference + NN (Brinkerhoff et al. 2021)
  PINN (Riel, Michew et al. 2021, Diffice: Wang, Lai et al. 2022, 2025, Cheng, Morlighem et al. 2024, Steidl, Bamber et al. 2025)
  GP (Downs et al. 2022, Brinkerhoff et al. 2024)
  UDE (ODINN: Bolibar, Sapienza et al. 2023)
  CNN (DeepBedMap: Leong & Horgan 2020, Jouvet 2023a,b, DeepTopoNet: Tama et al. 2025)

Inversed modeling:

credit: NASA/Goddard Space Flight Center Scientific Visualization Studio

Prediction of ice dynamics relies on the basal friction and viscosity structure of ice sheet, which are not easily measurable.

Inverse modeling (find basal friction)

- Given u, v at the ice surface
  h at sparse locations
  & A, s
- Find c and h

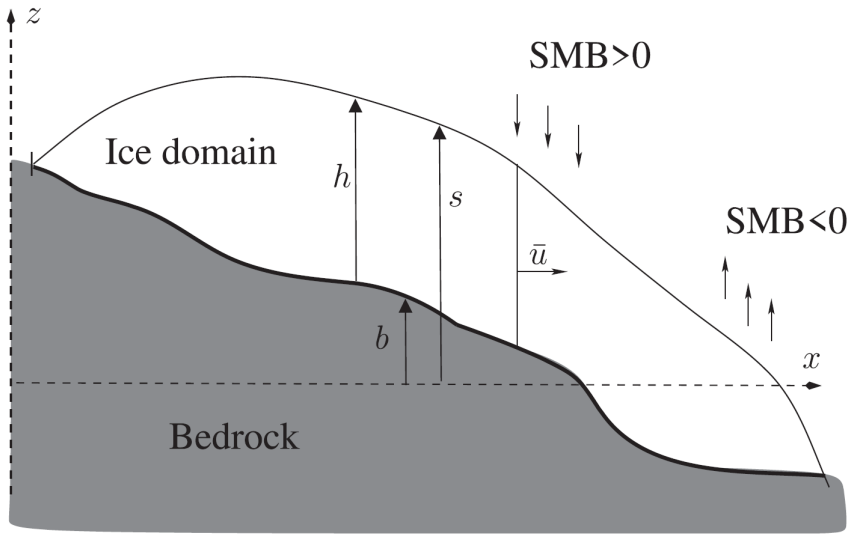


Fig. 2. Cross-section of a glacier with notations.

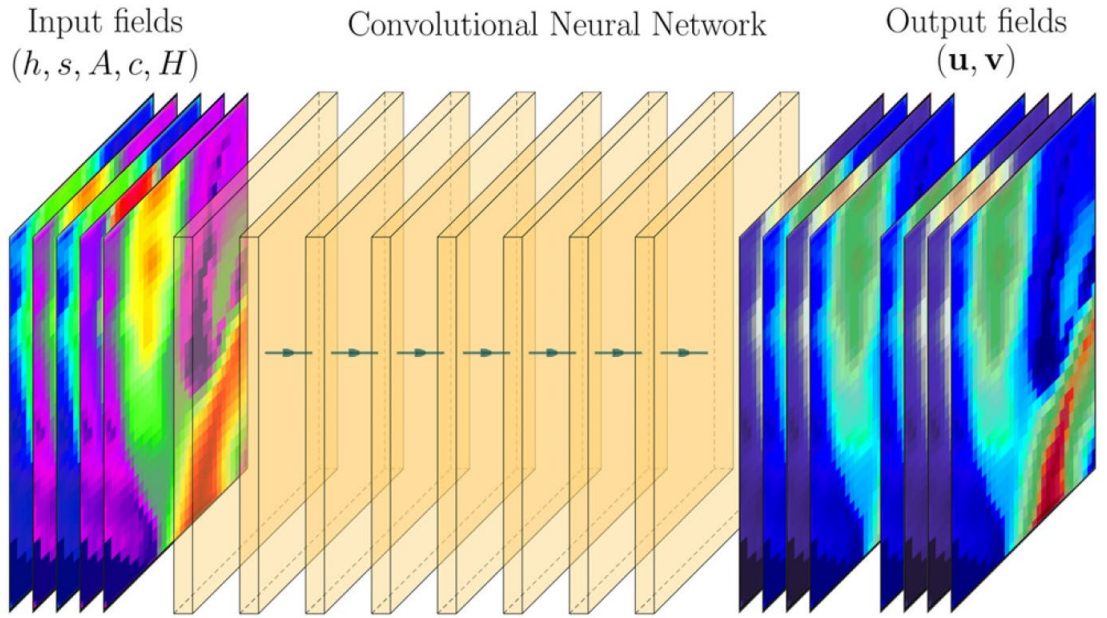


Figure 3. Our emulator consists of a CNN that maps geometrical (thickness and surface topography), ice-flow parameters (shearing and basal sliding) and spatial resolution inputs to 3D ice-flow fields.

Jouvet 2023a,b Instructed Glacier Model (IGM)

Inverse modeling (find basal friction)

- Given u, v at the ice surface
  h at sparse locations
  & A, s
- Find c and h

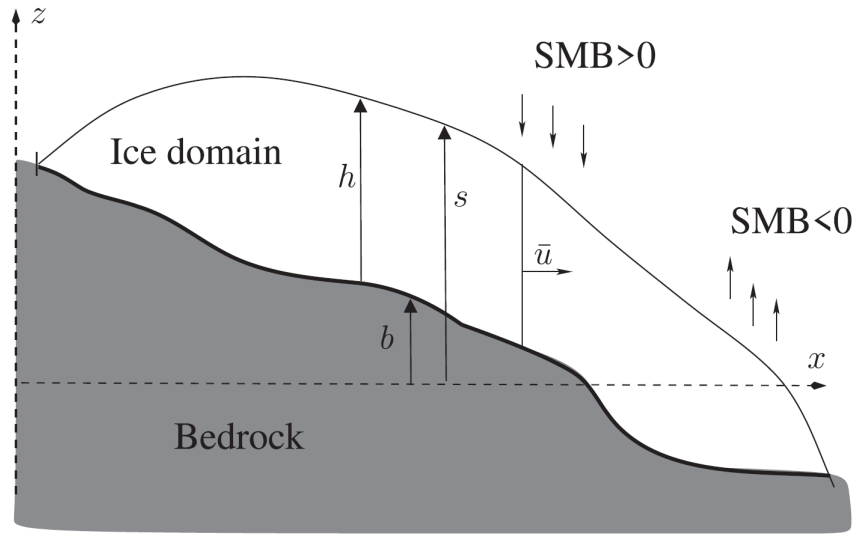
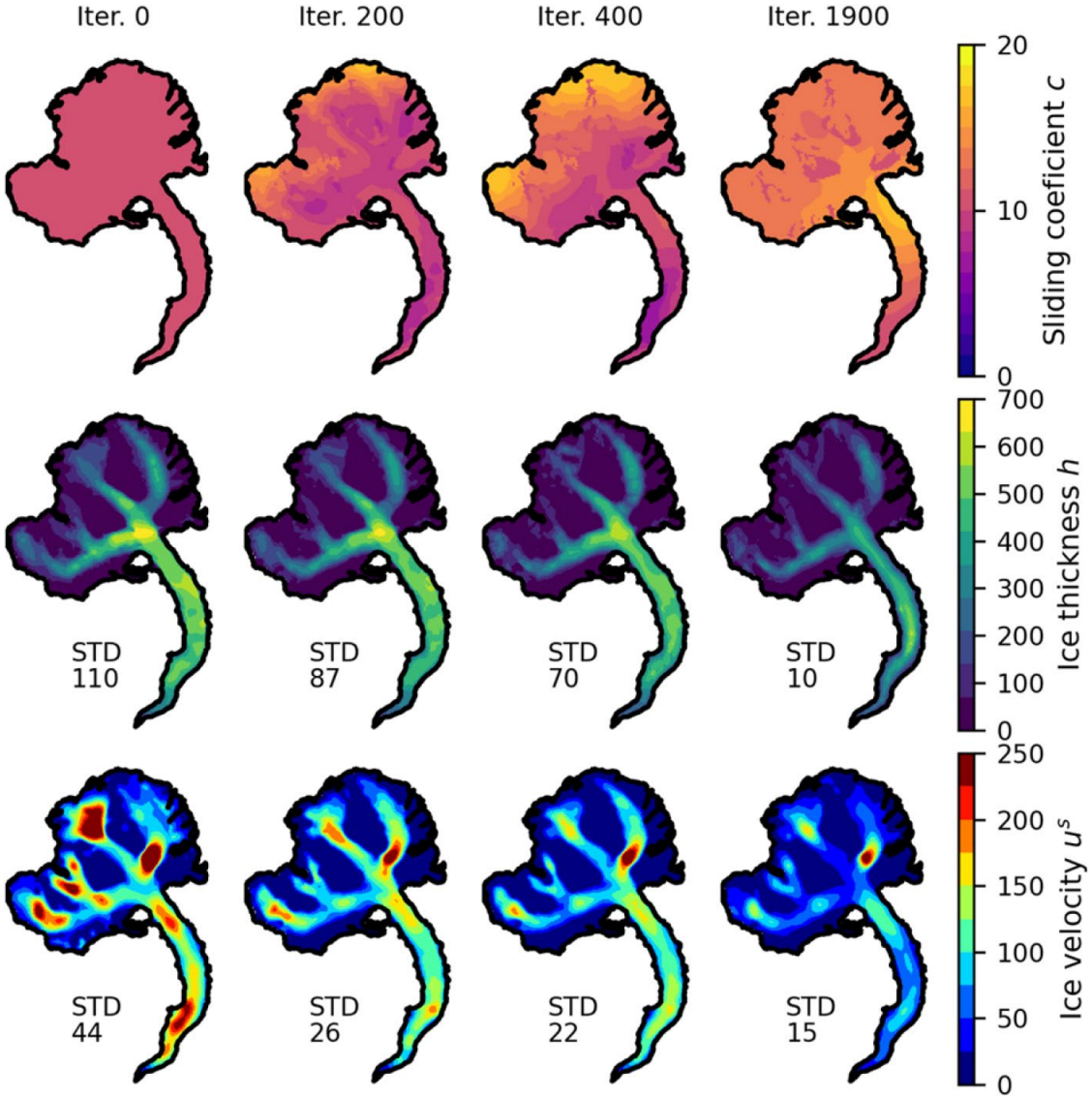


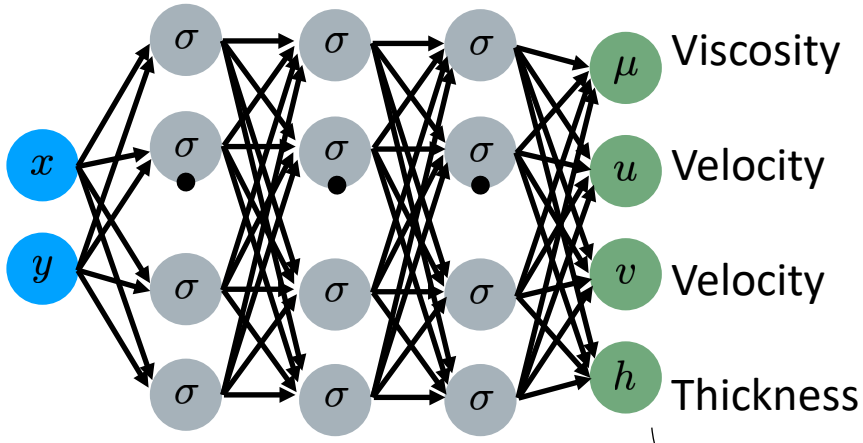
Fig. 2. Cross-section of a glacier with notations.

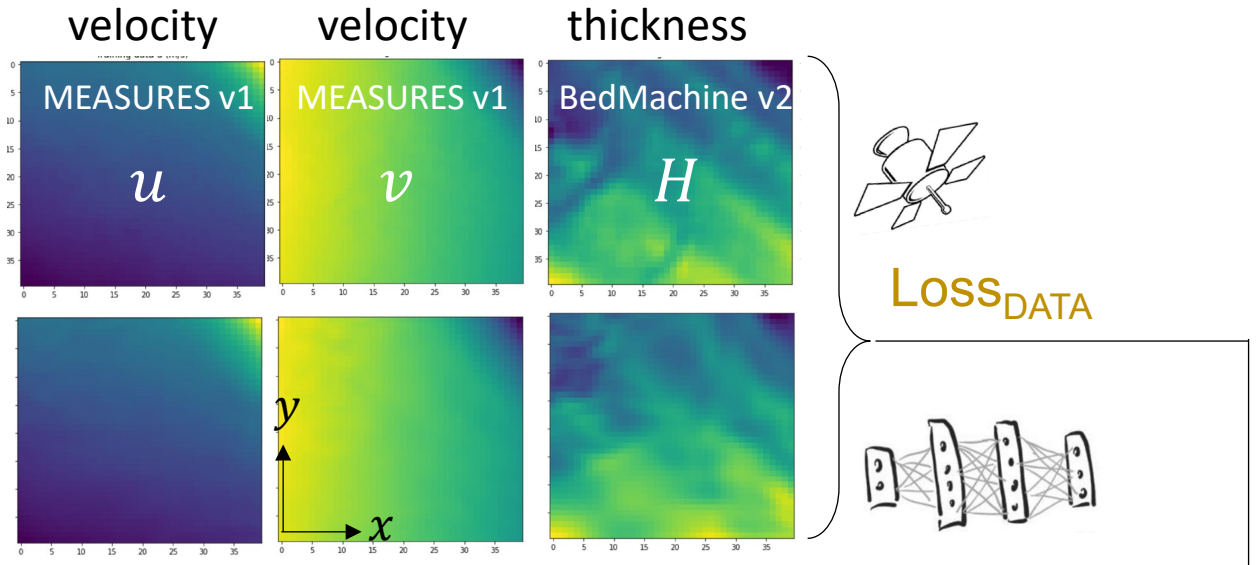


Wang, Lai et al. 2022, 2025 DIFFICE.jax

Inverse modeling (find viscosity)

A deep neural network





Governing equations

$$e_1 = \frac{\partial}{\partial x}\left(4\mu h\frac{\partial u}{\partial x} + 2\mu h\frac{\partial v}{\partial y}\right) + \frac{\partial}{\partial y}\left(\mu h\frac{\partial u}{\partial y} + \mu h\frac{\partial v}{\partial x}\right) - h\frac{\partial h}{\partial x}$$

$$e_2 = \frac{\partial}{\partial y}\left(4\mu h\frac{\partial v}{\partial y} + 2\mu h\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial x}\left(\mu h\frac{\partial u}{\partial y} + \mu h\frac{\partial v}{\partial x}\right) - h\frac{\partial h}{\partial y}$$

Loss_{EQN}

Loss function

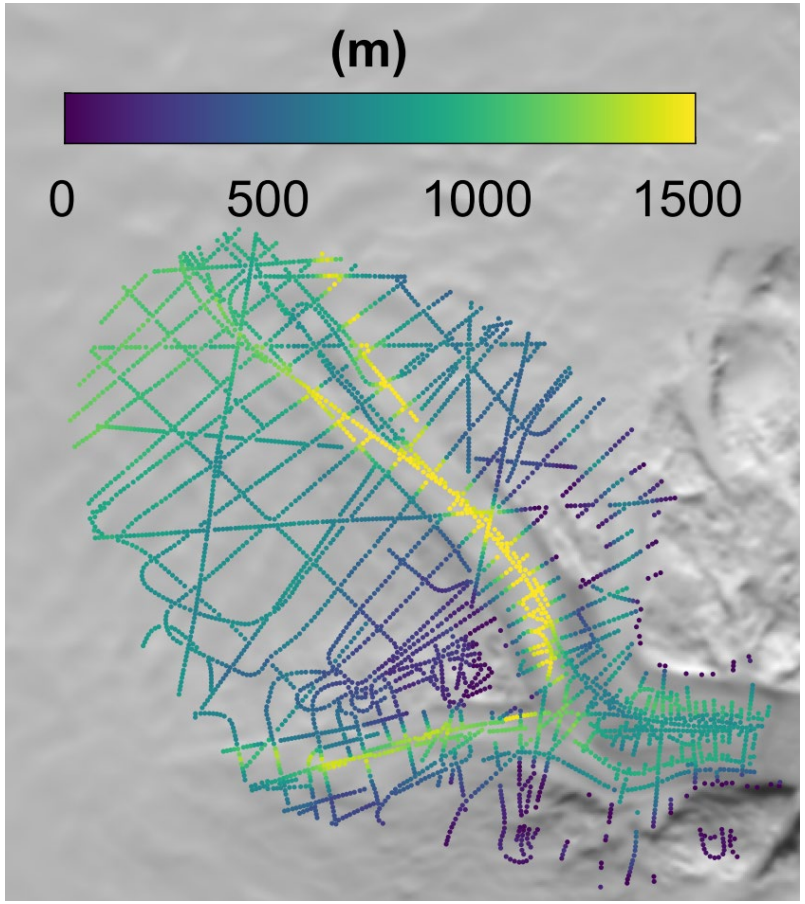
Update neural net parameters

$$\text{Loss} = (1-\gamma)\text{Loss}_{\text{DATA}} + \gamma\text{Loss}_{\text{EQN}} \qquad \gamma \in [0,1]$$

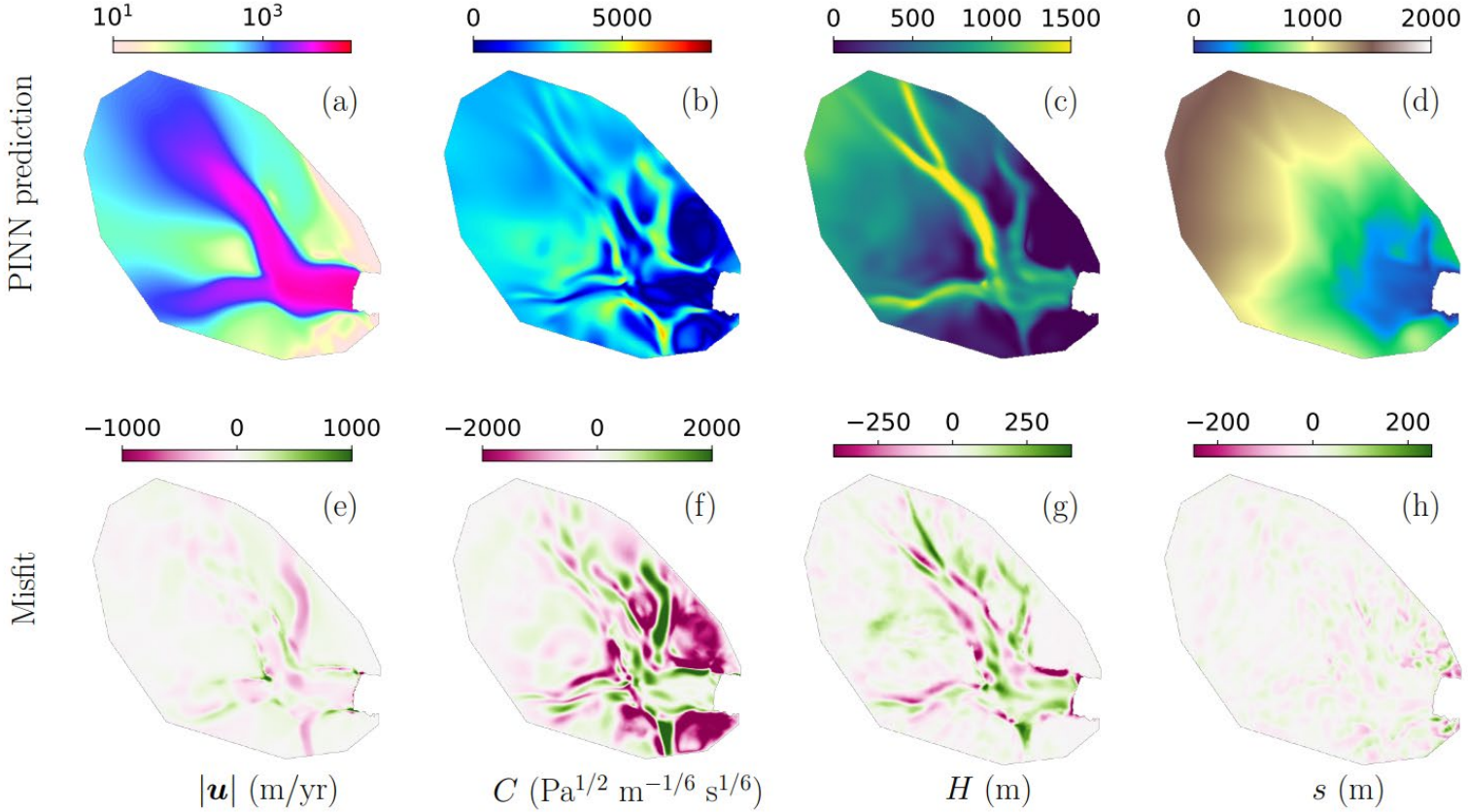
Cheng, Morlighem, and Francis 2024

Inverse modeling (find basal friction and thickness)

The actual observed thickness



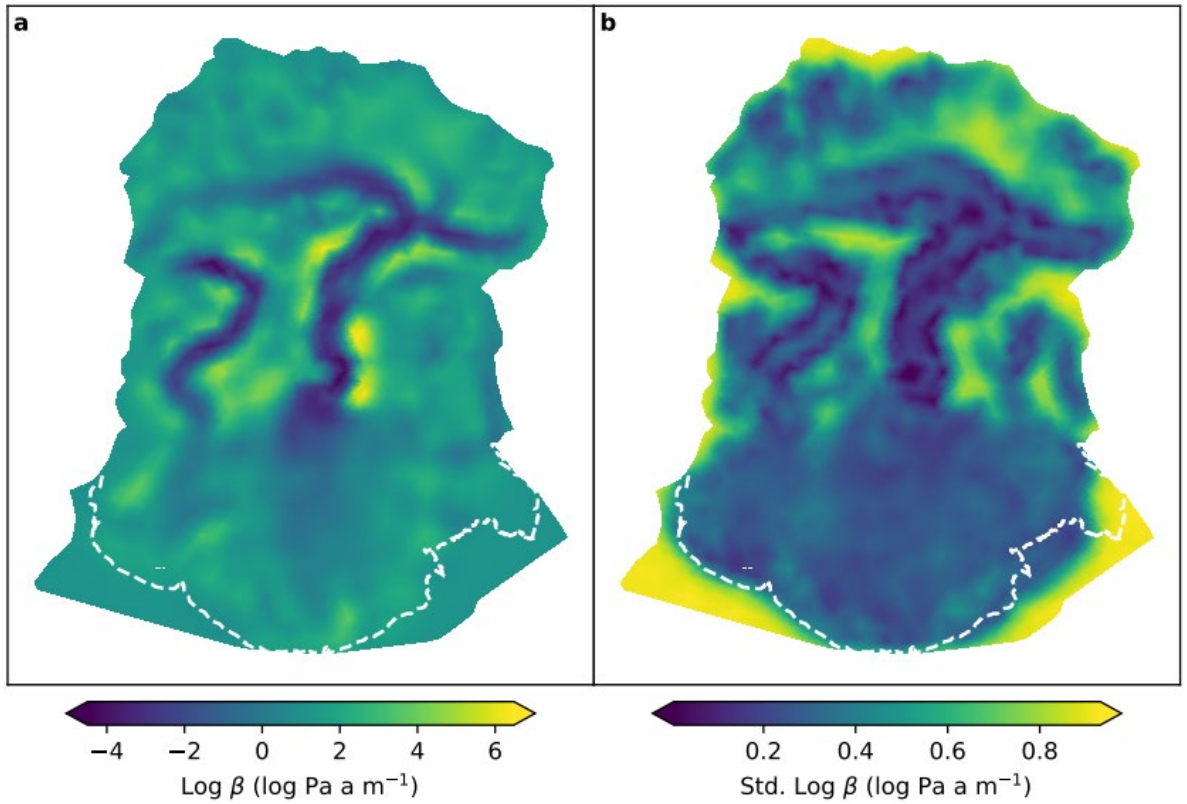
Inferring both ice thickness and basal friction



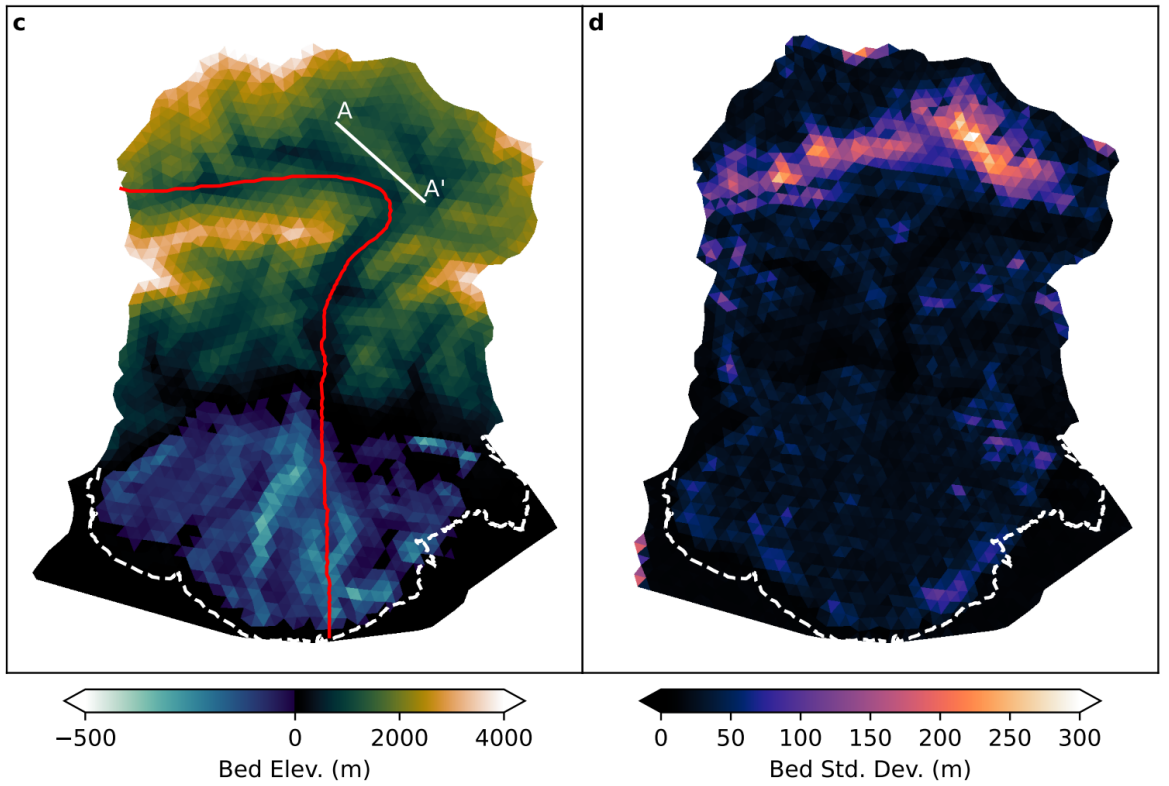
Brinkerhoff et al. 2024

Inverse modeling (find basal friction, bed topography, and surface mass balance with UQ)

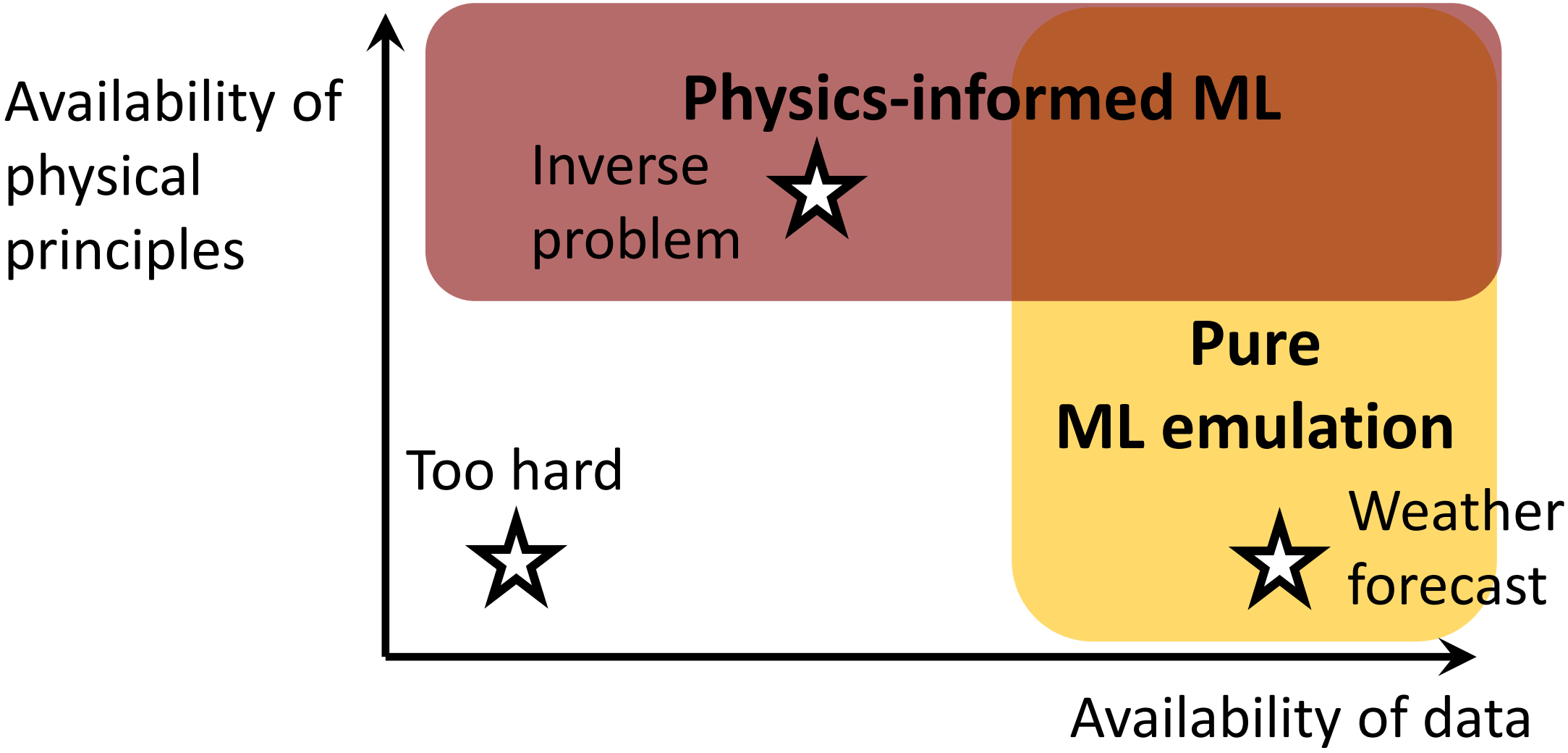
Modelled **basal traction** averaged over 1985- 2019



Inferred **bed topography**



Data and physics



Outline

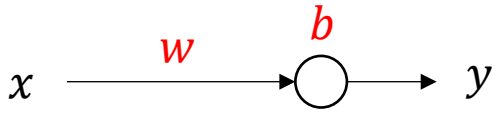
- Why neural networks?
- NN basics:
  Universal function approximation
  Gradient descent
  Back propagation
- Advanced topics: challenges of using NN in scientific problems
- Coding exercise

What is a neural network?

An analytical model of output y as a function of input x, containing some fitting parameters

1. Linear Regression Model:

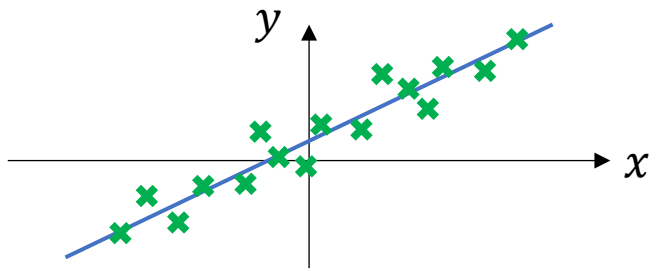
$$y = wx + b$$



Given observations of $\{x_d^i, y_d^i\}_i^n$

Find the w and b that minimizes

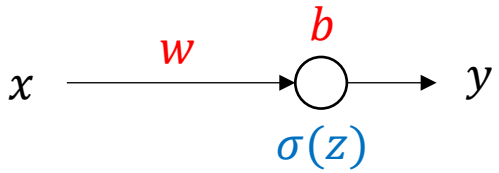
$$J = \sum_{i=1}^{n} \left(y\left(x_d^i\right) - y_d^i\right)^2$$

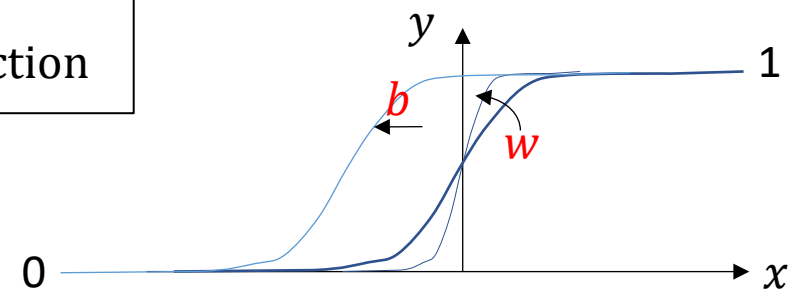


2. Logistic Regression Model: make output 0 to 1

$$y = \sigma(wx + b),$$

$$\text{where } \sigma(z) = \frac{1}{1 + e^{-z}} \text{ is a sigmoid function}$$



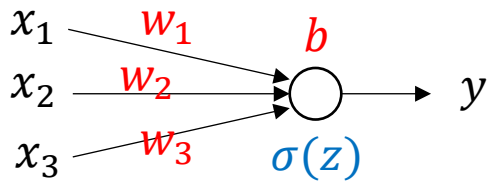


What is a neural network?

An analytical model of output y as a function of input x, containing some fitting parameters

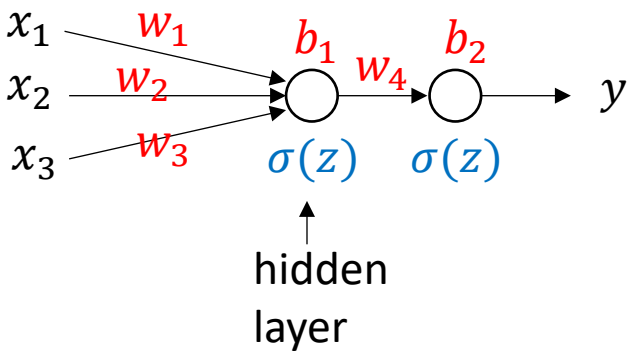
2. Logistic Regression Model: make output -1 to 1

$$y = \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3 + b),$$
$$\text{where } \sigma(z) = \frac{1}{1+e^{-z}} \text{ is a sigmoid function}$$



3. Neural network:

$$y = \sigma(w_4 \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3 + b_1) + b_2),$$
$$\text{where } \sigma(z) \text{ is a nonlinear activation function}$$



Common choices of $\sigma(z)$

$sigmoid(z)$

$\sin(z)$
$\cos(z)$
$\tanh(z)$

} Output ranges from -1 to 1

…

Input: $x \in \mathbb{R}^2$, Output: $y \in \mathbb{R}^1$ https://www.deeplearning.ai/ai-notes/initialization/index.html

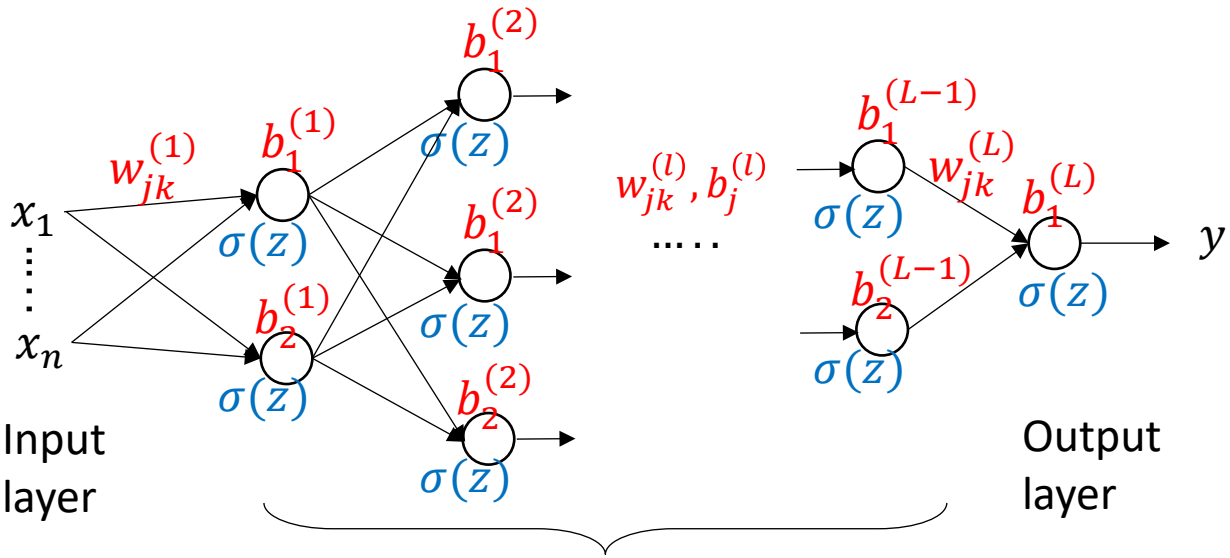
What is a neural network?

An analytical model of output y as a function of input x, containing some fitting parameters

3. Neural network: maps $x \in \mathbb{R}^n$ to $y \in \mathbb{R}^m$

More generally…

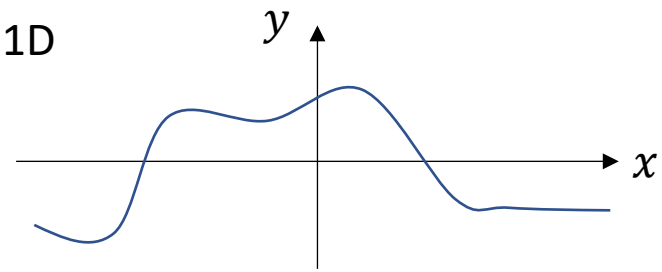
$$y = fn(w_{jk}^{(l)}, b_j^{(l)}, x_i),$$
where $\sigma(z)$ is a nonlinear activation function



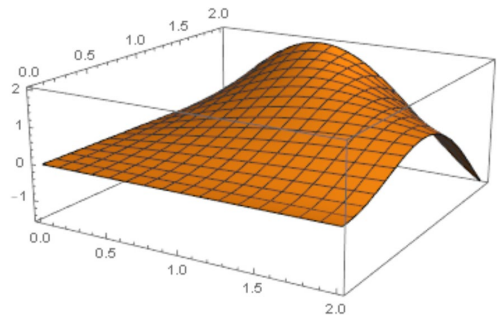
Neural network is a general function approximation

-> Universal function approximation

1D



2D

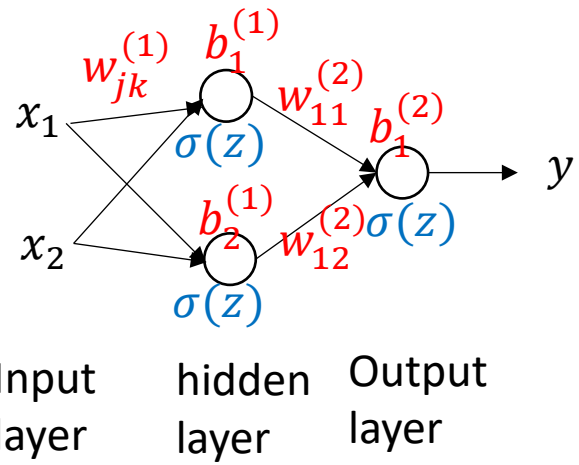


n-Dimension surface….

Why are activation function nonlinear?

Common choices of $\sigma(z)$

$sigmoid(z)$
$\sin(z)$
$\cos(z)$
$\tanh(z)$
…



- If activation function is linear, NN can only represent a linear function

If activation fn is linear, e.g. $\sigma(z) = z$ …

Input x_1, x_2

In hidden layer, j=1,2

$$z_j^{(1)} = \sum_{k=1}^{2} w_{jk}^{(1)} x_k + b_j^{(1)}$$

$$a_j^{(1)} = \sigma\left(z_j^{(1)}\right) = z_j^{(1)}$$

In output layer, j=1

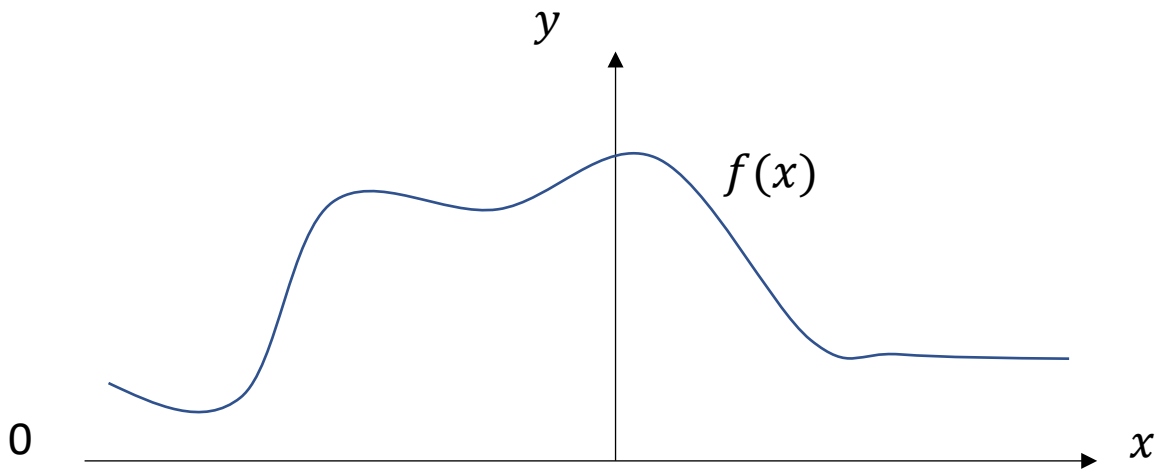
$$z_j^{(2)} = \sum_{k=1}^{2} w_{jk}^{(2)} a_k^{(1)} + b_j^{(2)}$$

$$a_j^{(2)} = \sigma\left(z_j^{(2)}\right) = z_j^{(2)}$$

$$y = a_1^{(2)} = \sum_{k=1}^{2} w_{1k}^{(2)} \left(\sum_{l=1}^{2} w_{kl}^{(1)} x_l + b_k^{(1)} \right) + b_1^{(2)} = Ax_1 + Bx_2 + C$$

NN is a Universal Function Approximator

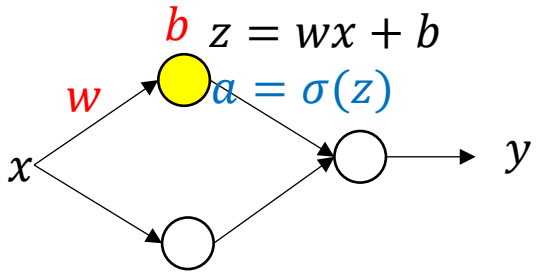
- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)

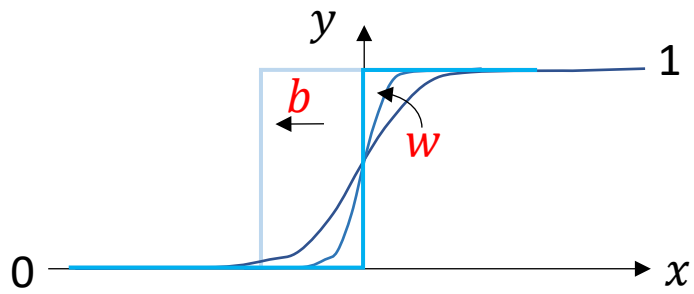


Goal:
Use a NN to approximate $f(x)$

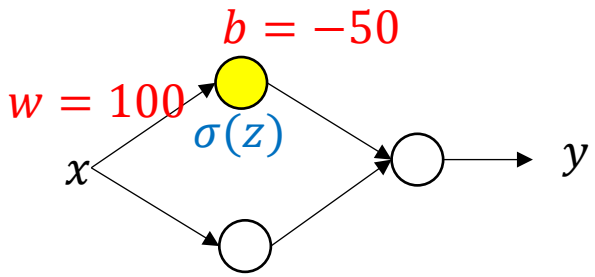
NN is a Universal Function Approximator

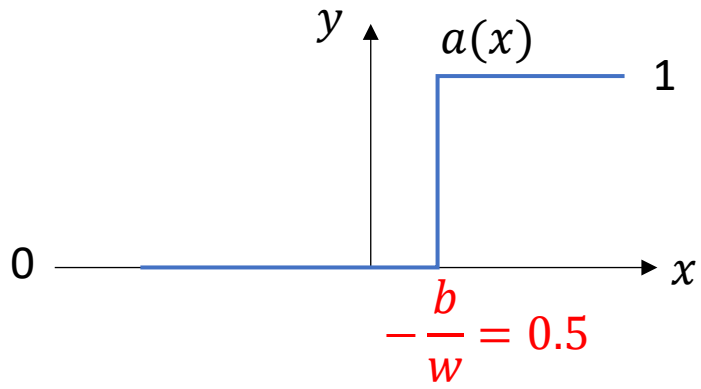
- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)





Large w gives a step

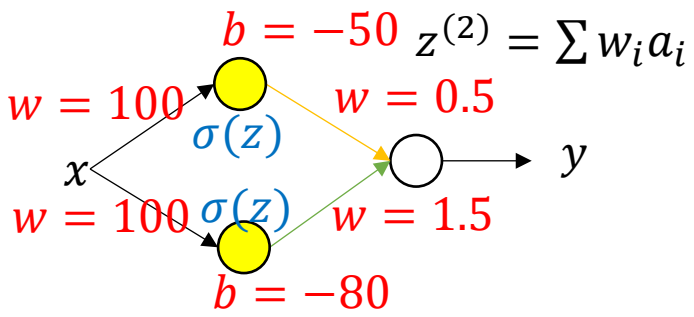


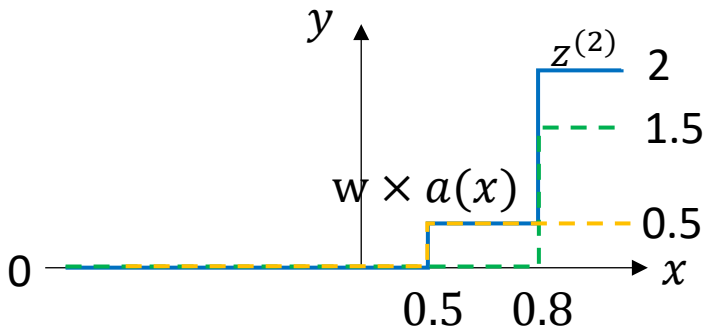


$-\frac{b}{w}$ determines the location of the step

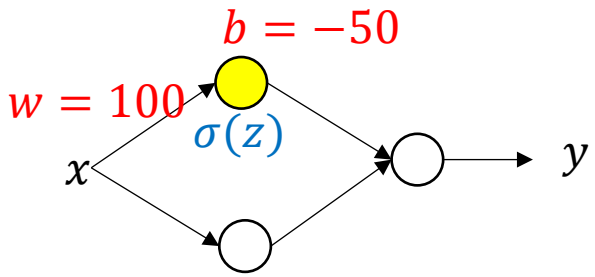
NN is a Universal Function Approximator

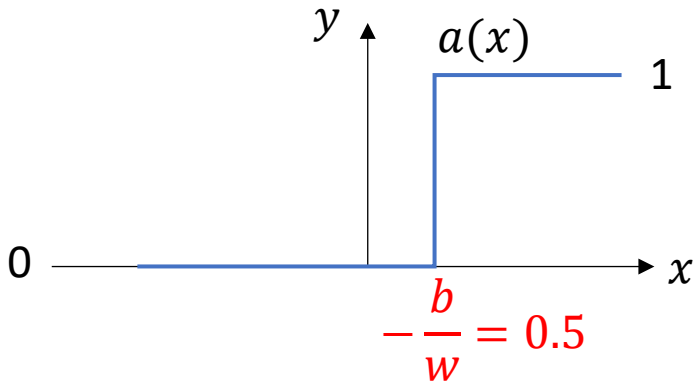
- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)





Superposition of two steps

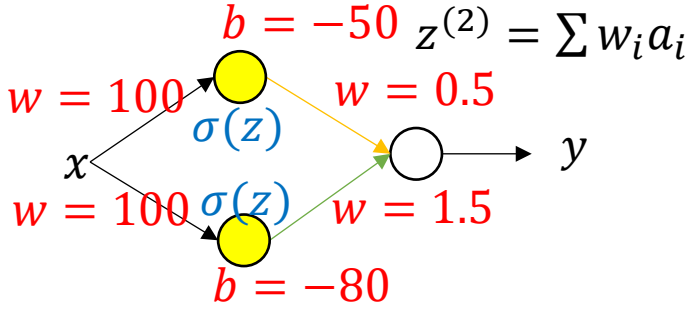


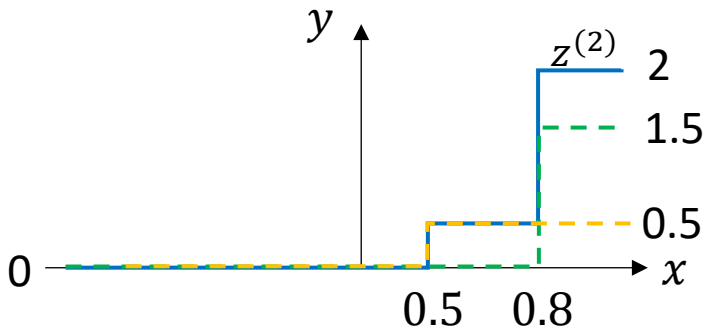


$-\frac{b}{w}$ determines the location of the step

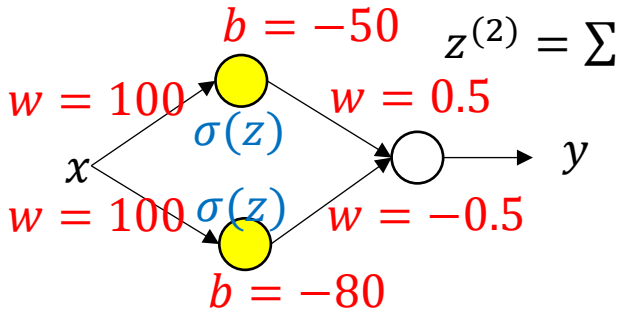
NN is a Universal Function Approximator

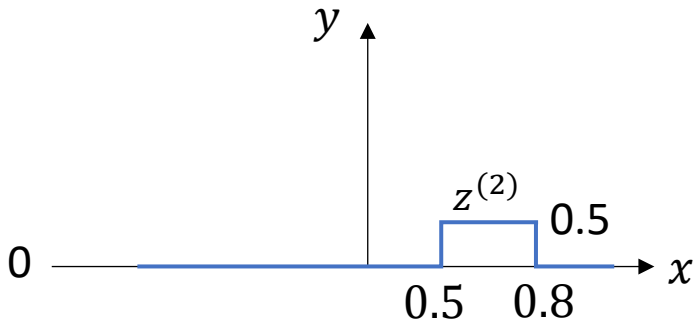
- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)





Superposition of two steps

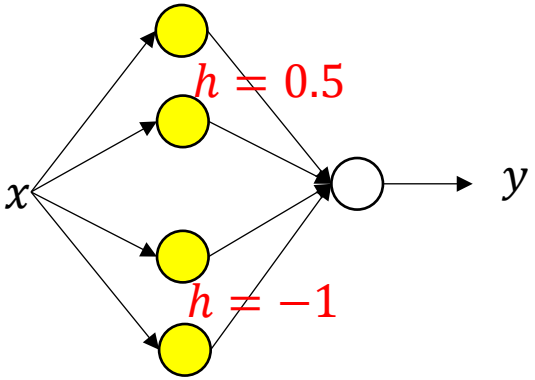


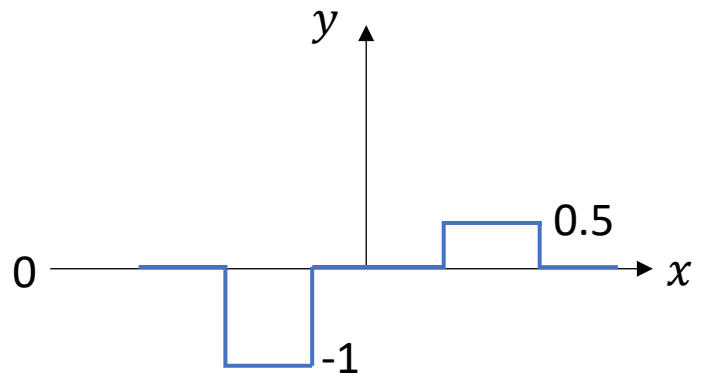


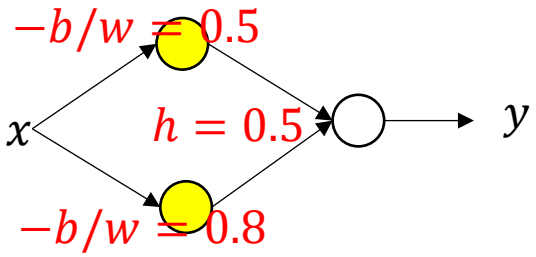
Create a column of height h=0.5

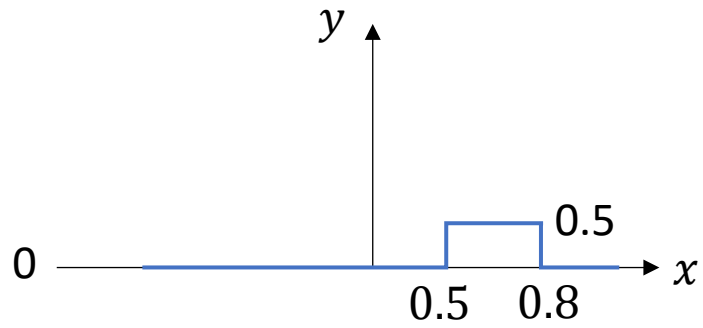
NN is a Universal Function Approximator

- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)





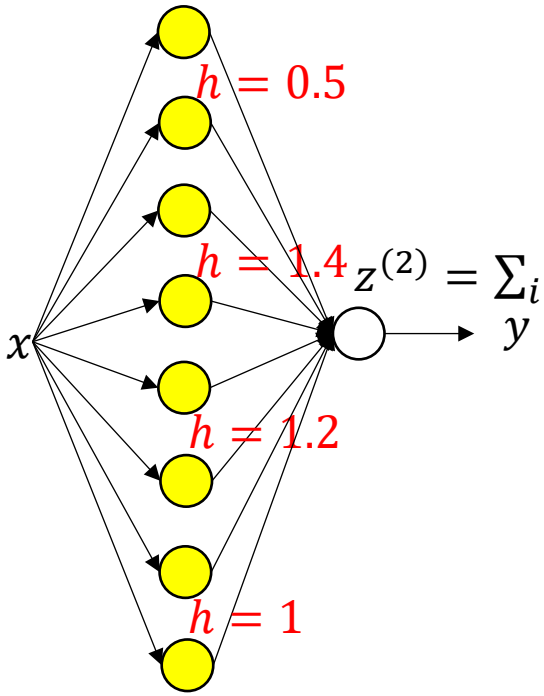


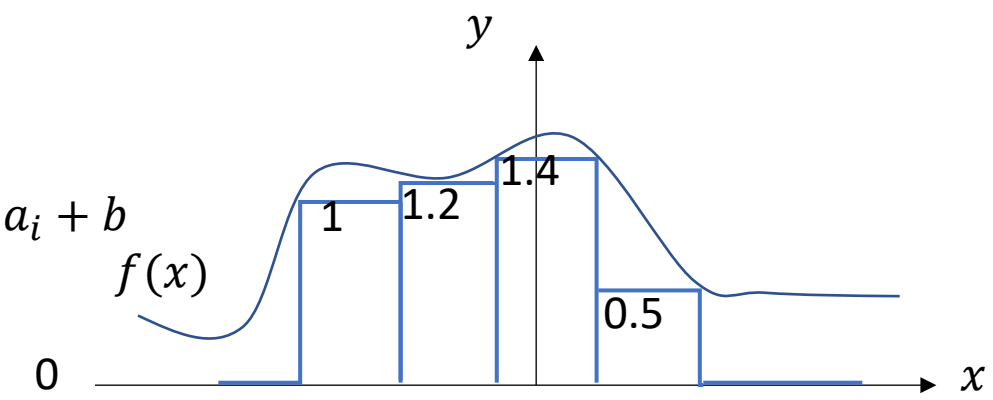


Create a column of height h=0.5

NN is a Universal Function Approximator

- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)

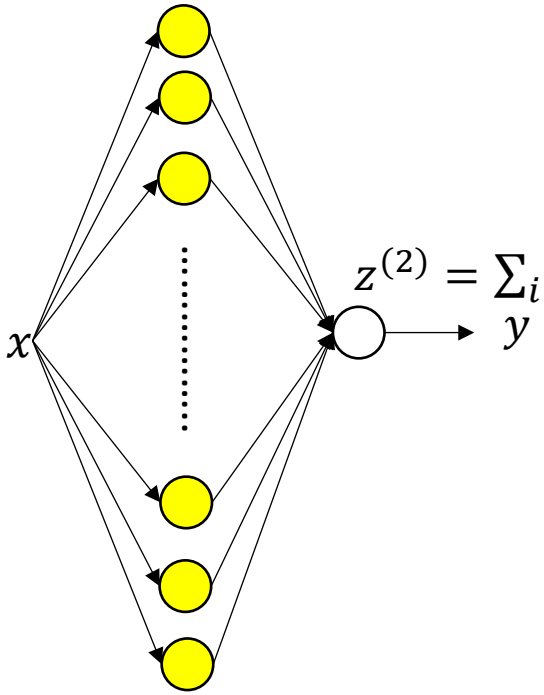


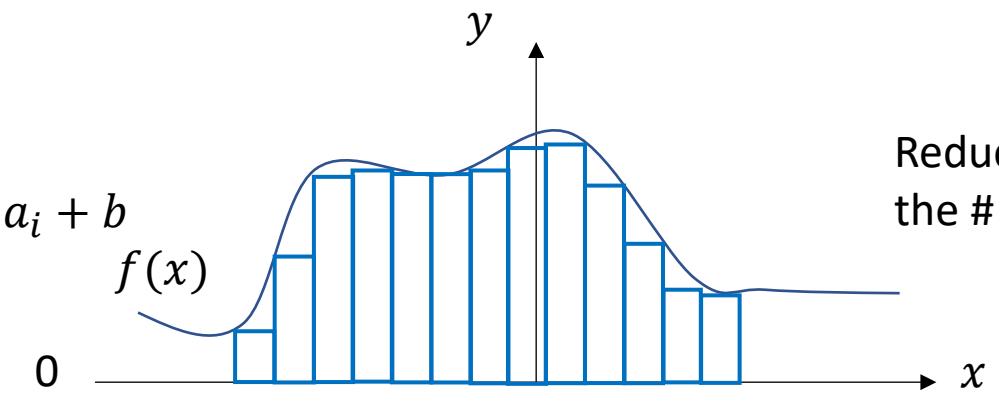


Superposition of columns

NN is a Universal Function Approximator

- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)





Reduce column widths + increasing the # of neurons to approximate $f(x)$

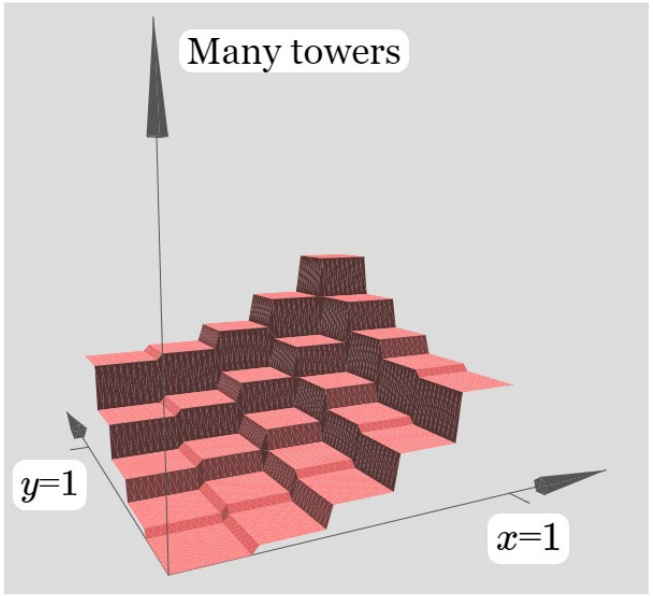
For a given continuous smooth $f(x)$ and an arbitrarily small $\epsilon > 0$
There exist $z^{(2)}$ so that

$$\int |z^{(2)}(x) - f(x)|\, dx < \epsilon$$

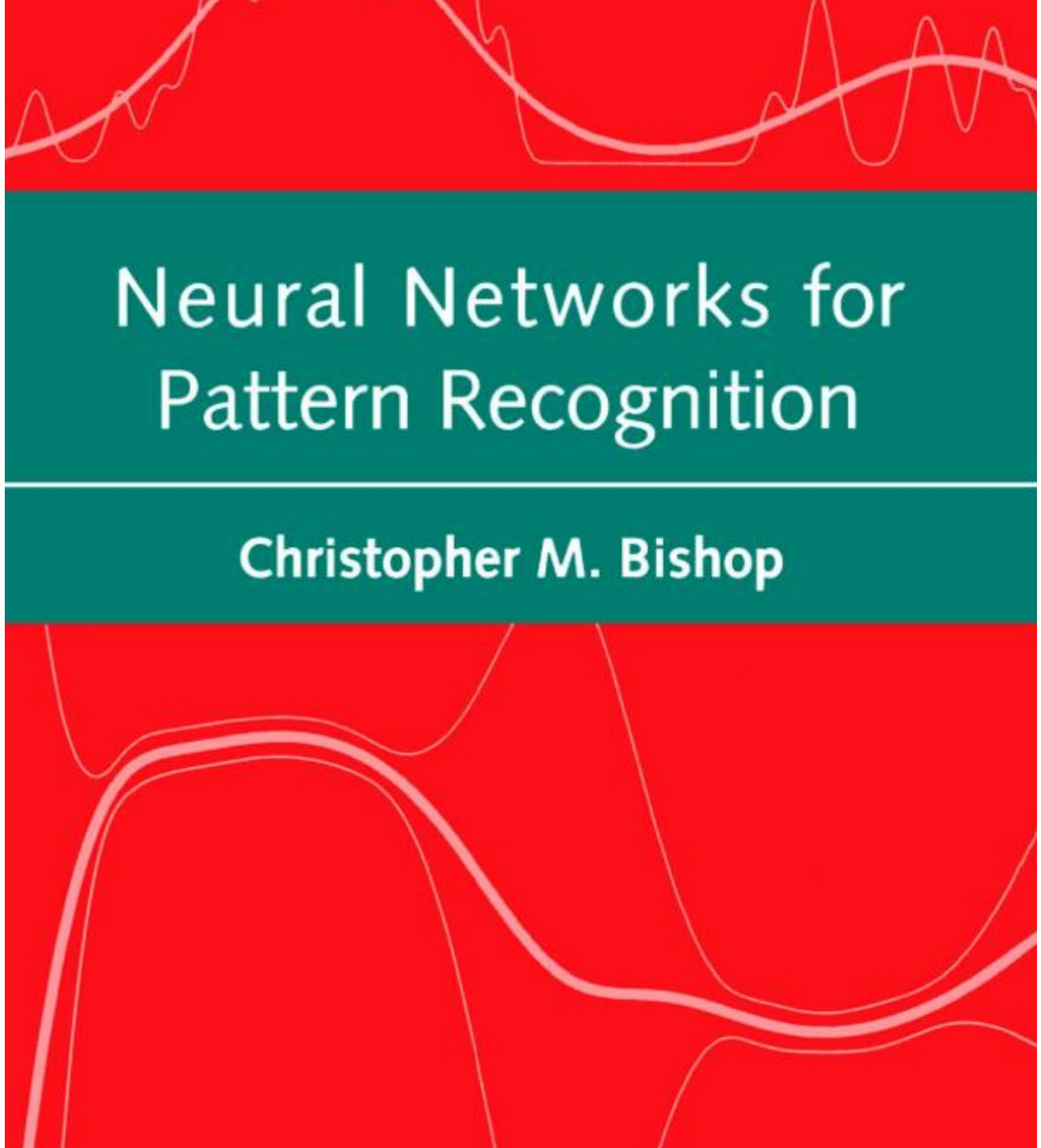
NN is a Universal Function Approximator

- Approximate a 2D surface

What should be the input/output units of NN?



http://neuralnetworksanddeeplearning.com/chap4.html

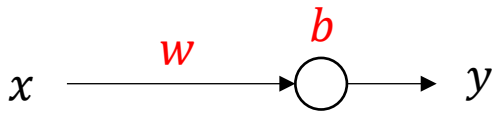


Chapter 4
The Multi-layer Perceptron

Now we know it is possible to tune weights and biases in a NN to approximate functions. We still need an automated method to find the correct weights and biases.

How to find w and b?

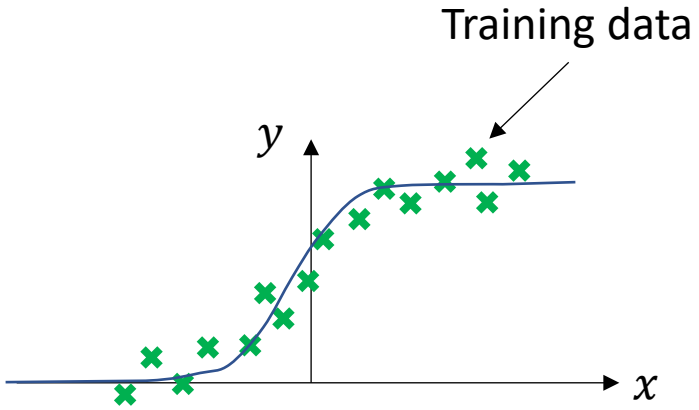
- E.g., Model $y = \sigma(wx + b)$



Given observations of $\{x_d^i, y_d^i\}_i^m$

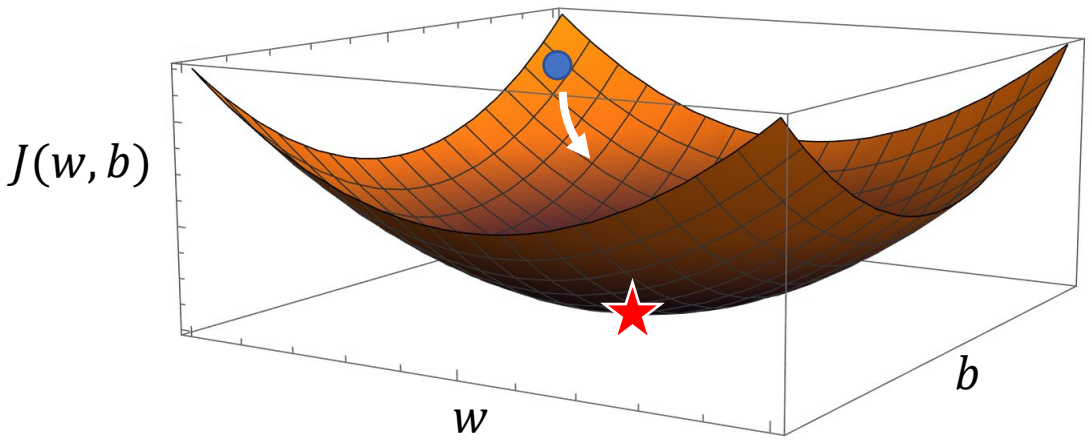
Our goal is to find the model parameters w, b that minimizes the cost function

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2$$



How to find w and b?

- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2$

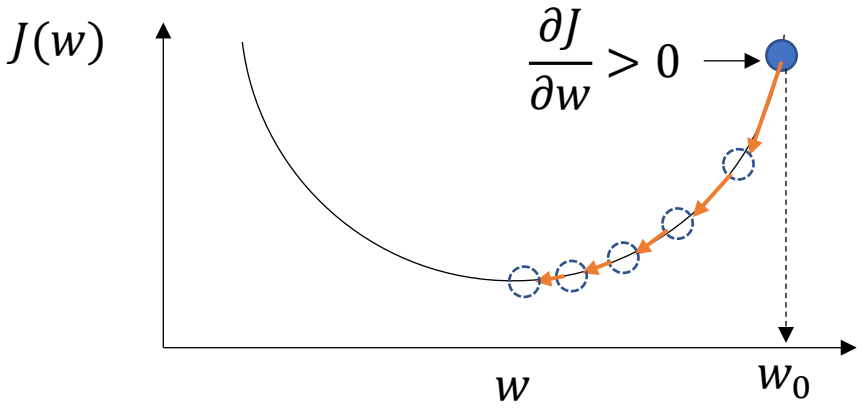


Find the w, b that minimizes $J(w,b)$

i.e. $\frac{\partial J}{\partial w} = \frac{\partial J}{\partial b} = 0$

How to find w and b? Gradient descent

- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2$



I. calculate the slope $\frac{\partial J}{\partial w}$ corresponds to an initial w

II. Adjust w according to the local slope

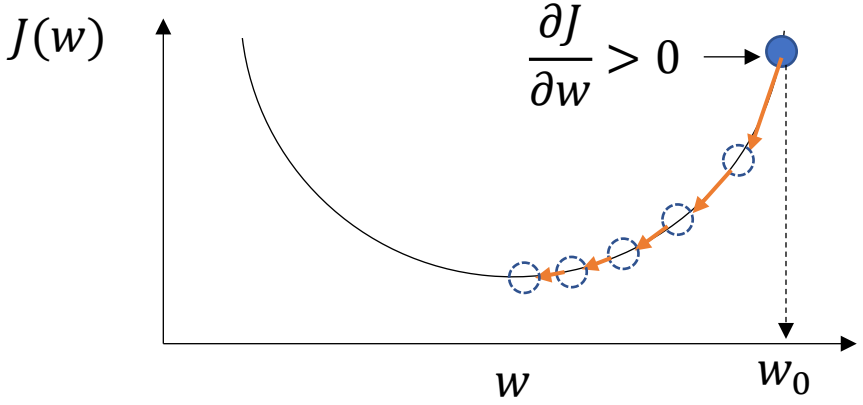
$w_{new} = w_{old} - \alpha \frac{\partial J}{\partial w}$, $\alpha > 0$ is the learning rate

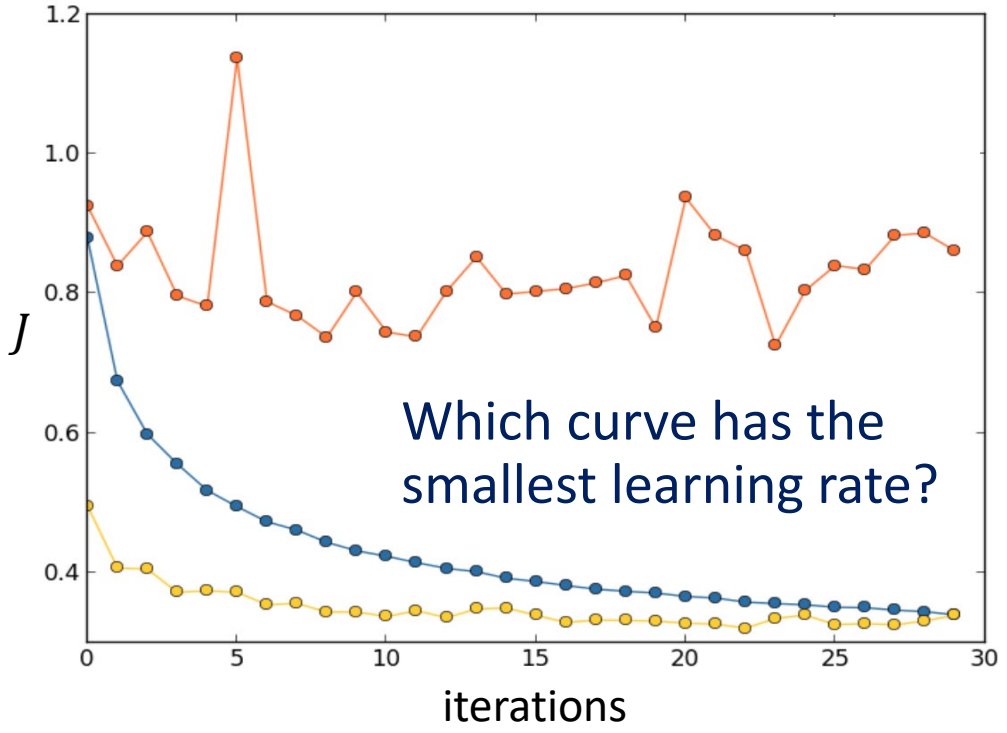
III. Iterate until $\frac{\partial J}{\partial w} = 0$

https://developers.google.com/machine-learning/crash-course/fitter/graph

How to find w and b? Gradient descent

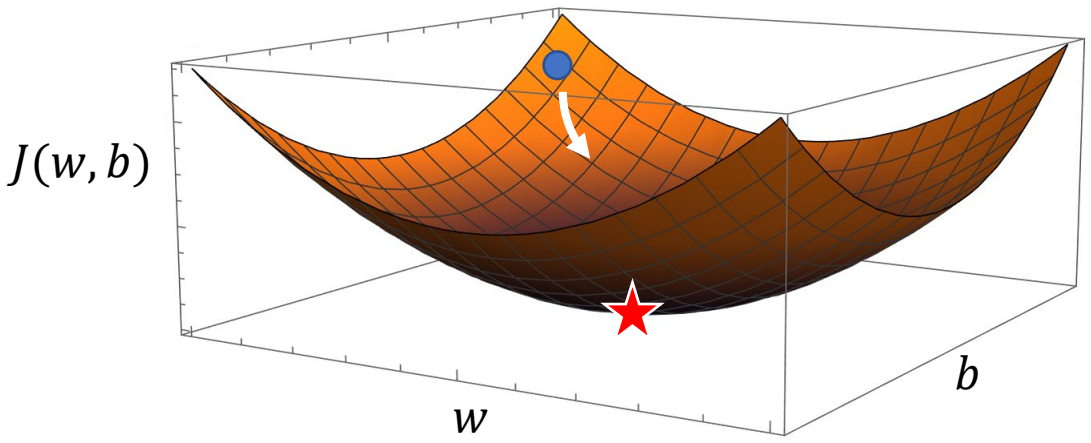
- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2$





How to find w and b? Gradient descent

- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2$



Gradient on a surface

- $-\nabla J$ gives the direction of the steepest decrease of J

$$-\nabla J(w,b) = -\left(\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}\right)$$

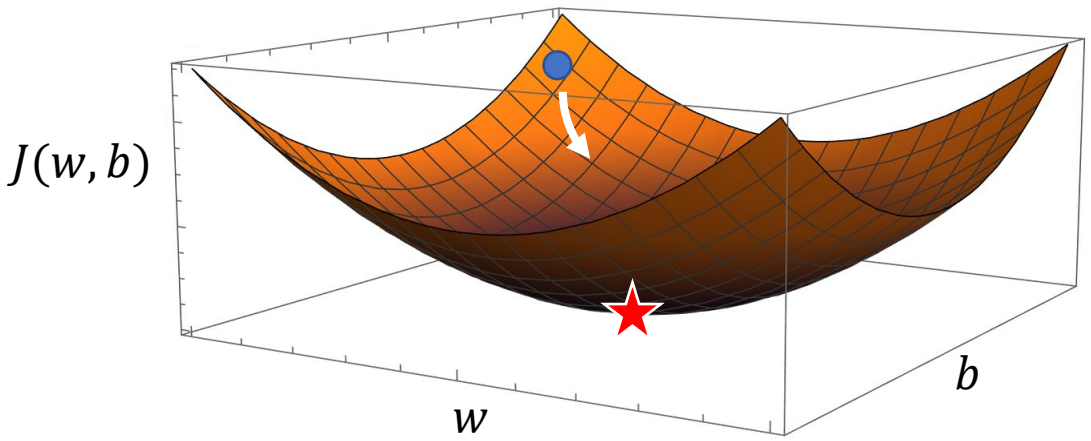
e.g. $-\nabla J(w_0, b_0) = -(10,1)$ What does this mean?

Changing w reduces J 10 times faster than changing b

- $-\nabla J$ tells you which weights and biases reduce cost function J the fastest!

How to find w and b? Gradient descent

- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2$



Gradient on a surface

- $-\nabla J$ gives the direction of the steepest decrease of J

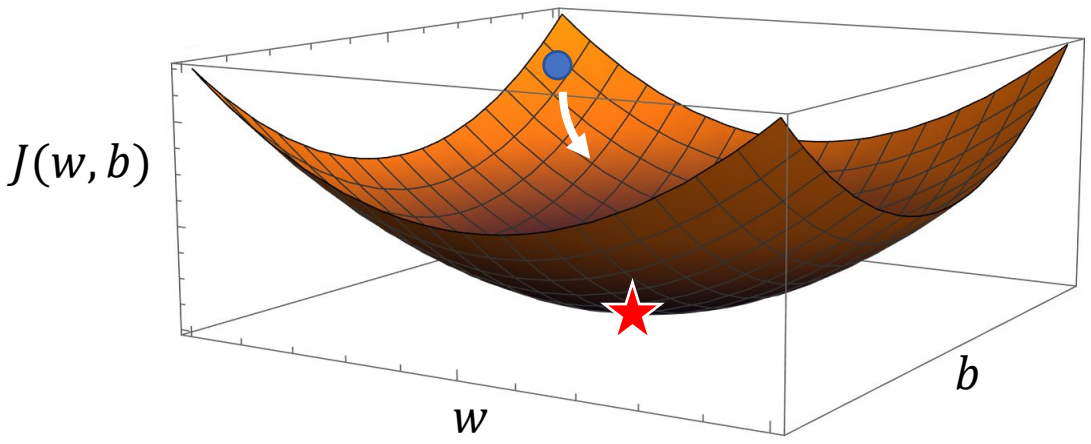
$$-\nabla J(w,b) = -\left(\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}\right)$$

- $(w_{new}, b_{new}) = (w_{old}, b_{old}) - \alpha\, \nabla J(w,b)$, α is the learning rate
- Iterate until $\nabla J = 0$

https://www.deeplearning.ai/ai-notes/optimization/

How to find w and b? Gradient descent

- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2$



Gradient on a surface

- $-\nabla J$ gives the direction of the steepest decrease of J

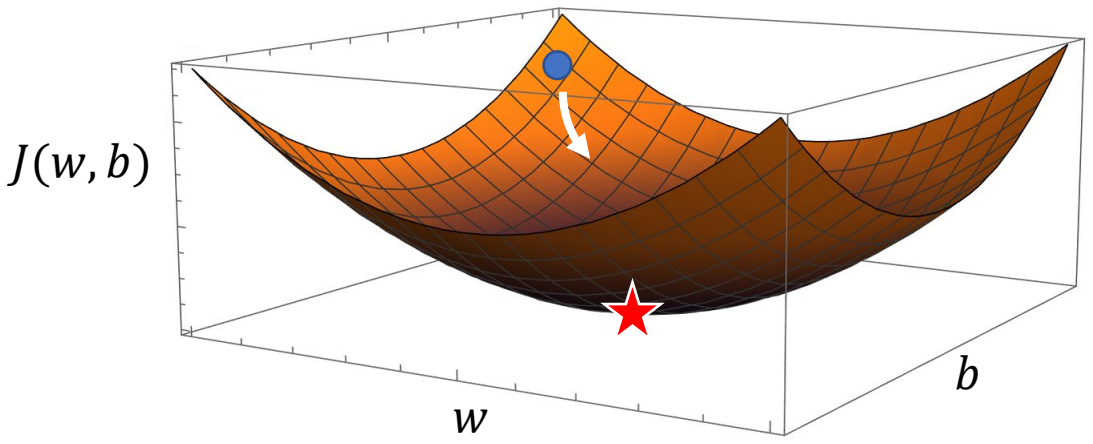
$$-\nabla J(w,b) = -\left(\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}\right)$$

But how are $\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}$ calculated?

How to find w and b? Gradient descent

- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2 = \frac{1}{m}\sum_{i=1}^{m} L(y^i, y_d^i)$

Defined for every example i



What would $\frac{dL}{dw}$ be for a linear model $\sigma(z) = z$?

$$L \equiv (y(x) - y_d)^2,\ J = \frac{1}{m}\sum_{i=1}^{m} L \rightarrow \frac{\partial J}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial L}{\partial w}$$

Back propagation (chain rule)

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial (wx+b)}\frac{\partial (wx+b)}{\partial w} = 2(y - y_d)\sigma' x$$

1) For one example:

$$\frac{\partial L}{\partial w}\left(\mathrm{w}, \mathrm{b}, \mathrm{x}_\mathrm{d}^i, \mathrm{y}_\mathrm{d}^i\right) = 2\left(\sigma\left(w\mathrm{x}_\mathrm{d}^i + b\right) - \mathrm{y}_\mathrm{d}^i\right)\sigma'\ \mathrm{x}_\mathrm{d}^i$$

2) For the full data set:

$$\frac{\partial J}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial L}{\partial w} = \frac{1}{m}\sum_{i=1}^{m} 2\left(\sigma\left(w\mathrm{x}_\mathrm{d}^i + b\right) - \mathrm{y}_\mathrm{d}^i\right)\sigma'\mathrm{x}_\mathrm{d}^i$$

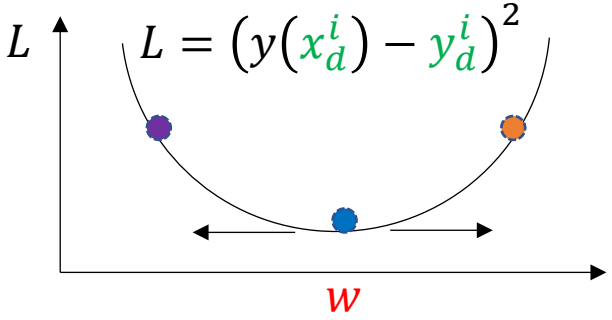
https://www.deeplearning.ai/ai-notes/optimization/ How would data shape the loss landscape?

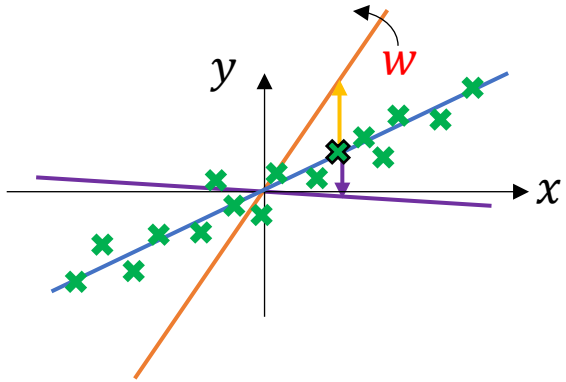
Example:

- Linear Regression Model $y = wx$
- Cost function $J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2 = \frac{1}{m}\sum_{i=1}^{m} L(y^i, y_d^i)$

Defined for every example i

1) For one example:



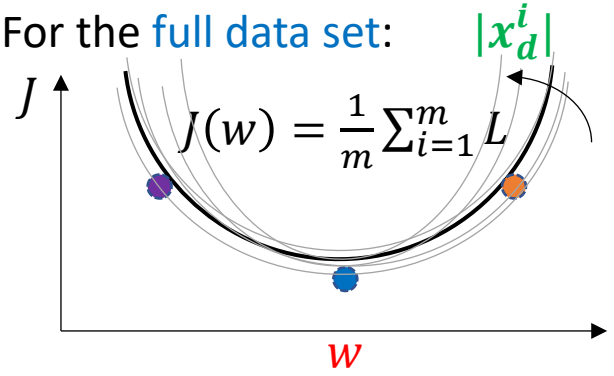


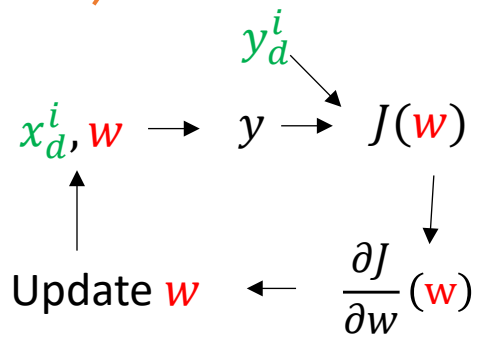
$L \equiv (y(x) - y_d)^2, \; J = \frac{1}{m}\sum_{i=1}^{m} L \rightarrow \frac{\partial J}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial L}{\partial w}$

Back propagation (chain rule)

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial (wx)}\frac{\partial (wx)}{\partial w} = 2(y - y_d)x$$

2) For the full data set:





1) For one example:

$$\frac{\partial L}{\partial w}\left(\mathrm{w}, \mathrm{b}, \mathrm{x}_\mathrm{d}^i, \mathrm{y}_\mathrm{d}^i\right) = 2\left(wx_d^i - \mathrm{y}_\mathrm{d}^i\right)\mathrm{x}_\mathrm{d}^i$$

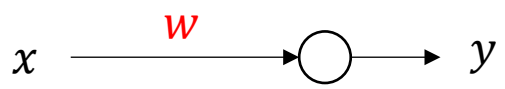
This gradient is defined for every example i, and is a function of w

2) For the full data set:

$$\frac{\partial J}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial L}{\partial w} = \frac{1}{m}\sum_{i=1}^{m} 2\left(wx_d^i - \mathrm{y}_\mathrm{d}^i\right)\mathrm{x}_\mathrm{d}^i$$

https://www.deeplearning.ai/ai-notes/optimization/

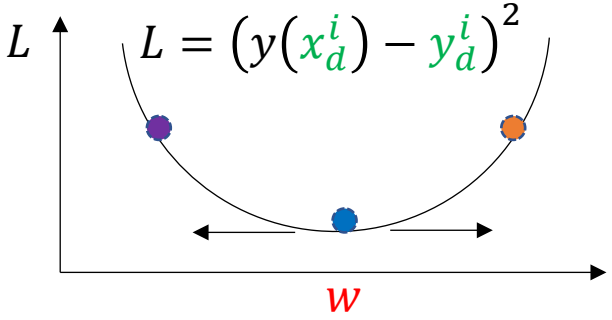
Example:

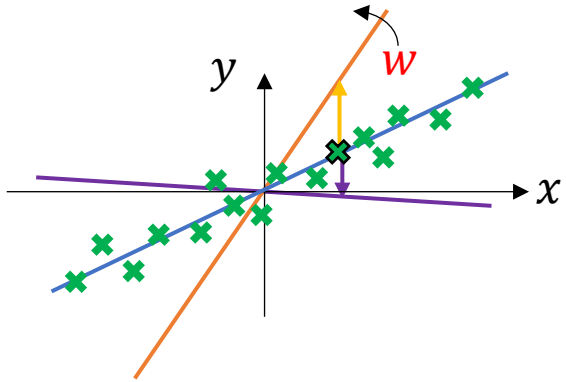


- Linear Regression Model $y = wx$
- Cost function $J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2 = \frac{1}{m}\sum_{i=1}^{m} L(y^i, y_d^i)$

Defined for every example i

1) For one example:





$$L \equiv (y(x) - y_d)^2, \quad J = \frac{1}{m}\sum_{i=1}^{m} L \rightarrow \frac{\partial J}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial L}{\partial w}$$

Back propagation (chain rule)

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial (wx)}\frac{\partial (wx)}{\partial w} = 2(y - y_d)x$$

1) For one example:

$$\frac{\partial L}{\partial w}\left(\mathrm{w}, \mathrm{b}, \mathrm{x}_\mathrm{d}^i, \mathrm{y}_\mathrm{d}^i\right) = 2\left(wx_d^i - \mathrm{y}_\mathrm{d}^i\right)\mathrm{x}_\mathrm{d}^i$$

This gradient is defined for every example i, and is a function of w

2) For the full data set:

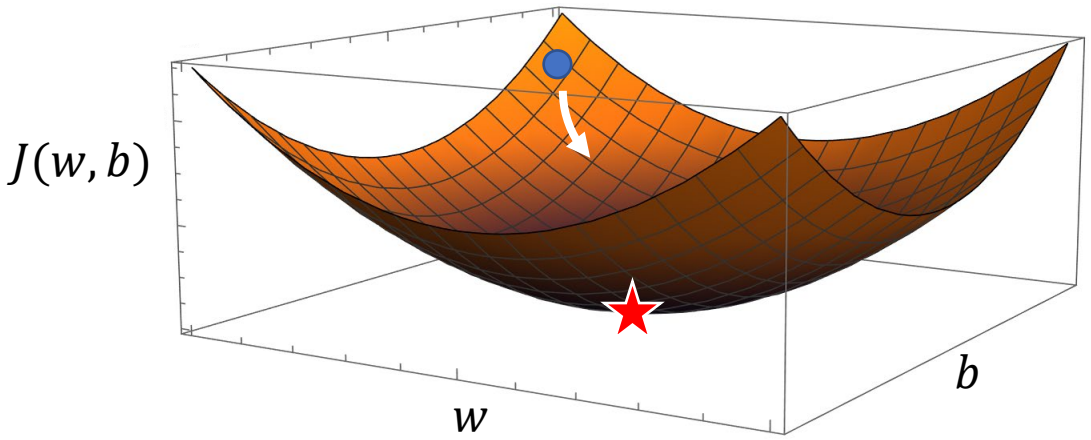
$$\frac{\partial J}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial L}{\partial w} = \frac{1}{m}\sum_{i=1}^{m} 2\left(wx_d^i - \mathrm{y}_\mathrm{d}^i\right)\mathrm{x}_\mathrm{d}^i$$

What would happen if x_d^i are very large/small?

Note that $\frac{\partial J}{\partial w}$ is analytical fn of $\mathrm{w}, x_d^i, \mathrm{y}_\mathrm{d}^i$!

How to find w and b? Gradient descent

- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2 = \frac{1}{m}\sum_{i=1}^{m} L(y^i, y_d^i)$



- $-\boldsymbol{\nabla} J(w,b) = -\left(\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}\right)$ **for a given data set at a given $\boldsymbol{w}$, $\boldsymbol{b}$ is known analytically**
- $(w_{new}, b_{new}) = (w_{old}, b_{old}) - \alpha\, \nabla J(w,b)$, α is the learning rate
- Iterate until $\nabla J = 0$

So far we have discussed…

- Universal function approximator
- Gradient descent: a method to find weights and biases that minimize J
- Calculate $-\boldsymbol{\nabla} J(w,b) = -\left(\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}\right)$ for a simple model $y = \sigma(wx + b)$
  - One example
  - A full dataset

We know how to find w and b for a simple model $y = \sigma(wx + b)$.

What about finding w and b in a neural network?

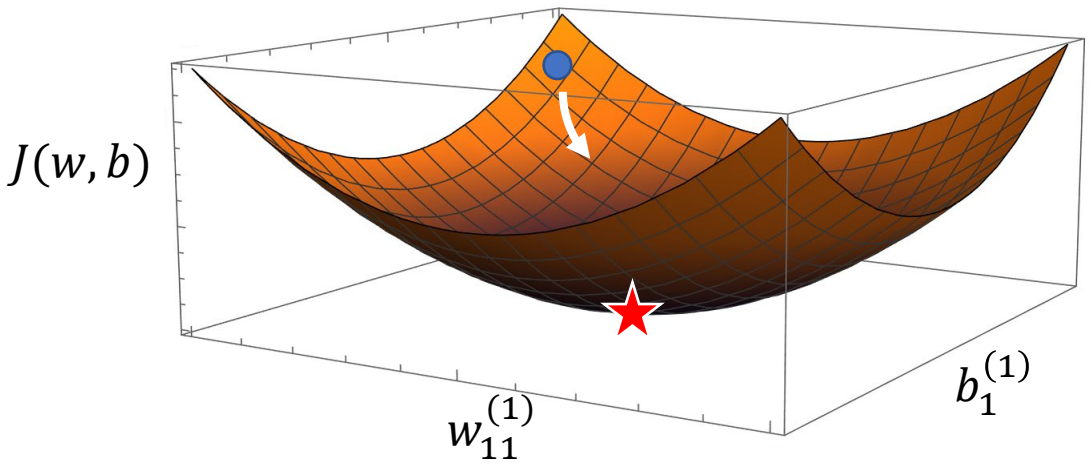
We know how to find w and b for a simple model $y = \sigma(wx + b)$.

What about finding w and b in a neural network?

$$y = a^{(2)} = \sigma(\boldsymbol{W}^{(2)}\sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) + \boldsymbol{b}^{(2)})$$

How to find w and b in a neural network?

- E. g., Model $y = a^{(2)} = \sigma(\boldsymbol{W}^{(2)}\sigma(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) + \boldsymbol{b}^{(2)})$
- Cost function $J(\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}) = \frac{1}{m}\sum_{i=1}^{m} L(y^i, y_d^i)$



- For the full data set (i=1…m), compute

$$-\boldsymbol{\nabla} J = -\left(\frac{\partial J}{\partial w_{jk}^{(l)}}, \ldots, \frac{\partial J}{\partial b_j^{(l)}}\right)$$

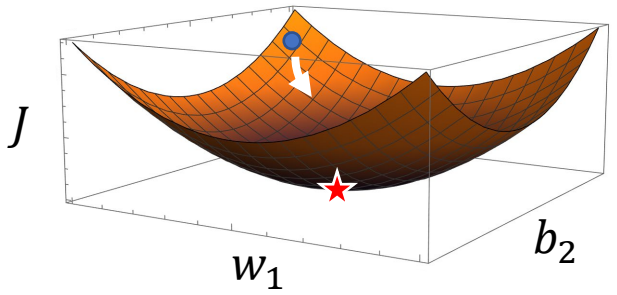
- ${w_{jk}}_{new}^{(l)} = {w_{jk}}_{old}^{(l)} - \alpha \frac{\partial J}{\partial w_{jk}^{(l)}}$, ${b_j}_{new}^{(l)} = {b_j}_{old}^{(l)} - \alpha \frac{\partial J}{\partial b_j^{(l)}}$

  α is the learning rate

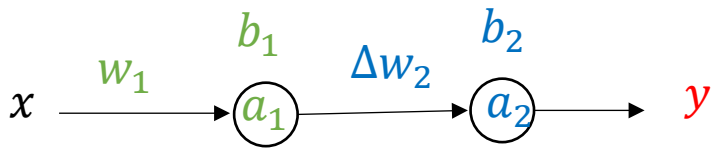
- Iterate until $\boldsymbol{\nabla} J = 0$

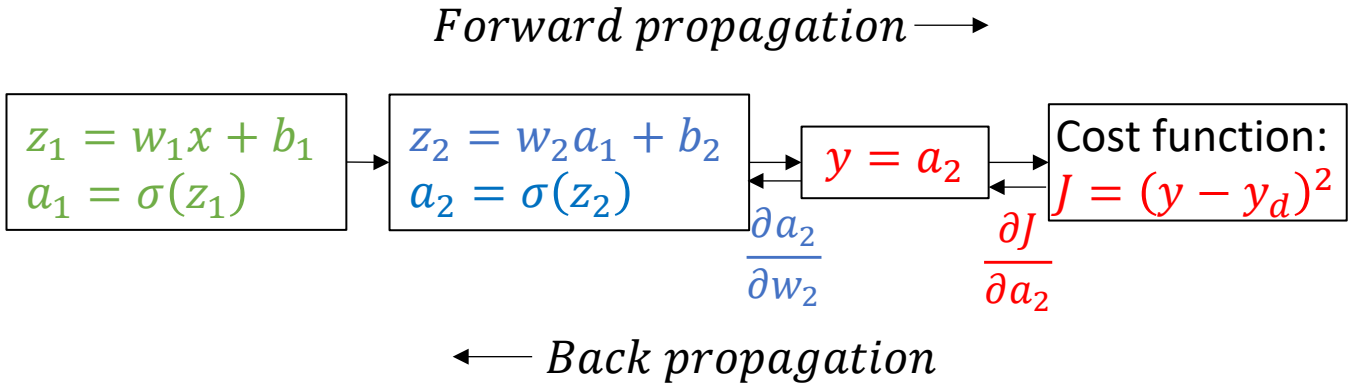
Forward & Back Propagation: 1. single neuron

Exercise:
Model: $y = \sigma(w_2\sigma(w_1 x + b_1) + b_2)$



∇J vary with w_1, w_2, b_1, b_2



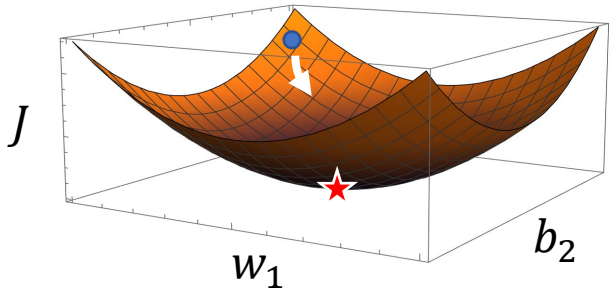


How important is w_2 for changing the cost function J?

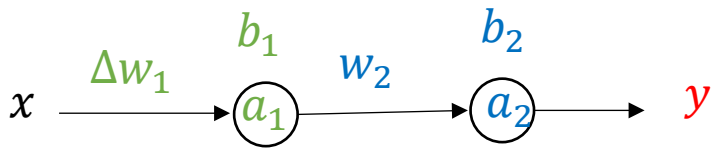
$$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$
$$= 2(y - y_d)\sigma'(z_2)a_1$$

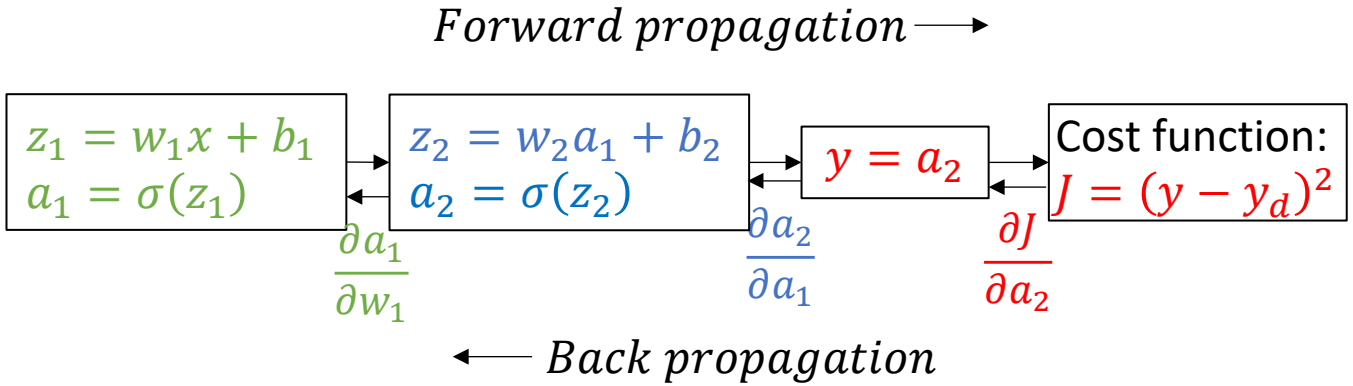
Forward & Back Propagation: 1. single neuron

Exercise:
Model: $y = \sigma(w_2\sigma(w_1x + b_1) + b_2)$



∇J vary with w_1, w_2, b_1, b_2





How important is w_1 for changing the cost function J?

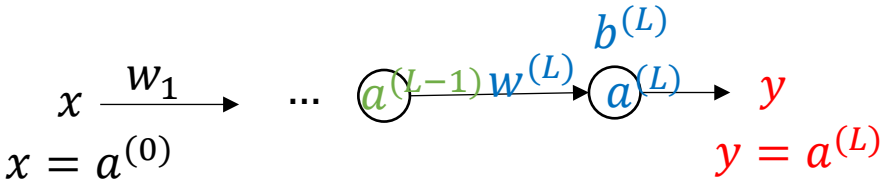
$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial a_2}\frac{\partial a_2}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$
$$= 2(y - y_d)\sigma'(z_2)w_2\sigma'(z_1)x$$

When training a neural network

Repeat these steps:

1. Forward propagate an input
2. Compute the cost function
3. Compute the gradients of the cost with respect to parameters using backpropagation
4. Update each parameter using the gradients, according to the optimization algorithm

Forward & Back Propagation: 1. single neuron



Forward propagation

Input: $a^{(0)} = x$

...

$z^{(L-1)} = w^{(L-1)} a^{(L-2)} + b^{(L-1)}$

$a^{(L-1)} = \sigma(z^{(L-1)})$

$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$

$a^{(L)} = \sigma(z^{(L)})$

Output: $y = a^{(L)}$

Cost function $J = (y - y_d)^2$

Back propagation (use chain rules)

$$\frac{\partial J}{\partial w^{(L)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial w^{(L)}} = 2(a^{(L)} - y_d)\, \sigma'(z^{(L)})\, a^{(L-1)}$$

$$\frac{\partial J}{\partial b^{(L)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial b^{(L)}} = 2(a^{(L)} - y_d)\, \sigma'(z^{(L)})$$

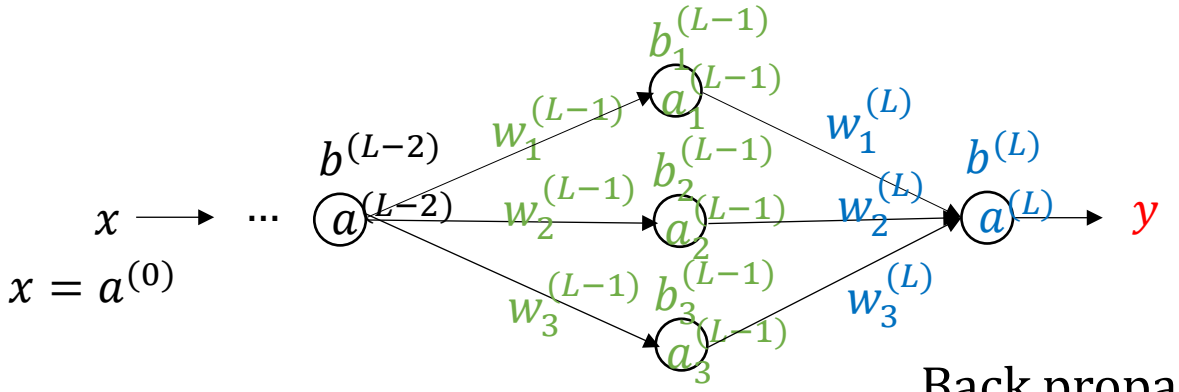
$$\frac{\partial J}{\partial w^{(L-1)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial w^{(L-1)}}$$

$$= 2(a^{(L)} - y_d)\, \sigma'(z^{(L)}) w^{(L)} \sigma'(z^{(L-1)}) a^{(L-2)}$$

$$\frac{\partial J}{\partial b^{(L-1)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial b^{(L-1)}}$$

$$= 2(a^{(L)} - y_d)\, \sigma'(z^{(L)}) w^{(L)} \sigma'(z^{(L-1)})$$

Forward & Back Propagation: 2. multiple hidden units



Forward propagation

Input: $a^{(0)} = x$

...

$z_k^{(L-1)} = w_k^{(L-1)} a^{(L-2)} + b_k^{(L-1)}$

$a_k^{(L-1)} = \sigma(z_k^{(L-1)})$

$z^{(L)} = \sum_{k=1}^{3} w_k^{(L)} a_k^{(L-1)} + b^{(L)}$

$a^{(L)} = \sigma(z^{(L)})$

Output: $y = a^{(L)}$

Cost function $J = (y - y_d)^2$

Back propagation (use chain rules)

$$\frac{\partial J}{\partial w_k^{(L)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial w_k^{(L)}} = 2(a^{(L)} - y_d)\, \sigma'(z^{(L)})\, a_k^{(L-1)}$$

$$\frac{\partial J}{\partial b^{(L)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial b^{(L)}} = 2(a^{(L)} - y_d)\, \sigma'(z^{(L)})$$

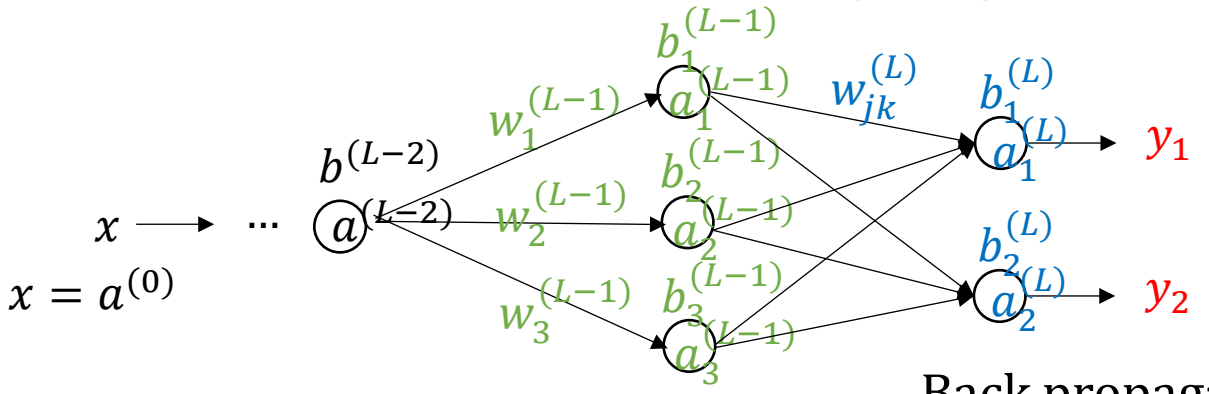
$$\frac{\partial J}{\partial w_k^{(L-1)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_k^{(L-1)}}{\partial w_k^{(L-1)}}$$

$$= 2(a^{(L)} - y_d)\, \sigma'(z^{(L)}) w_k^{(L)} \sigma'(z_k^{(L-1)}) a^{(L-2)}$$

$$\frac{\partial J}{\partial b_k^{(L-1)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_k^{(L-1)}}{\partial b_k^{(L-1)}}$$

$$= 2(a^{(L)} - y_d)\, \sigma'(z^{(L)}) w_k^{(L)} \sigma'(z_k^{(L-1)})$$

Forward & Back Propagation: 3. multiple outputs



Forward propagation

Input: $a^{(0)} = x$

...

$z_k^{(L-1)} = w_k^{(L-1)} a^{(L-2)} + b_k^{(L-1)}$

$a_k^{(L-1)} = \sigma(z_k^{(L-1)})$

$z_j^{(L)} = \sum_{k=1}^{3} w_{jk}^{(L)} a_k^{(L-1)} + b_j^{(L)}$

$a_j^{(L)} = \sigma(z_j^{(L)})$

Output: $y_j = a_j^{(L)}$

Cost function $J = \sum_{j=1}^{2} (y_j - y_{d,j})^2$

Back propagation (use chain rules)

$$\frac{\partial J}{\partial w_{jk}^{(L)}} = \frac{\partial J}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial w_{jk}^{(L)}} = 2(a_j^{(L)} - y_{d,j})\, \sigma'(z_j^{(L)}) a_k^{(L-1)}$$

$$\frac{\partial J}{\partial b_j^{(L)}} = \frac{\partial J}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial b_j^{(L)}} = 2(a_j^{(L)} - y_{d,j})\, \sigma'(z_j^{(L)})$$

$$\frac{\partial J}{\partial w_k^{(L-1)}} = \sum_{j=1}^{2} \frac{\partial J}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_k^{(L-1)}}{\partial w_k^{(L-1)}}$$

$$= \sum_{j=1}^{2} 2(a_j^{(L)} - y_{d,j})\, \sigma'(z_j^{(L)}) w_{jk}^{(L)}\, \sigma'(z_k^{(L-1)}) a^{(L-2)}$$

$$\frac{\partial J}{\partial b_k^{(L-1)}} = \sum_{j=1}^{2} \frac{\partial J}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_k^{(L-1)}}{\partial b_k^{(L-1)}}$$

$$= \sum_{j=1}^{2} 2(a_j^{(L)} - y_{d,j})\, \sigma'(z_j^{(L)}) w_{jk}^{(L)}\, \sigma'(z_k^{(L-1)})$$

Summary

- Universal function approximator
- Gradient descent: a method to find weights and biases that minimize J
- Calculate $\boldsymbol{\nabla} J$
  - One example
  - A full dataset
- Forward and backpropagation (a clever way to get gradients of J)

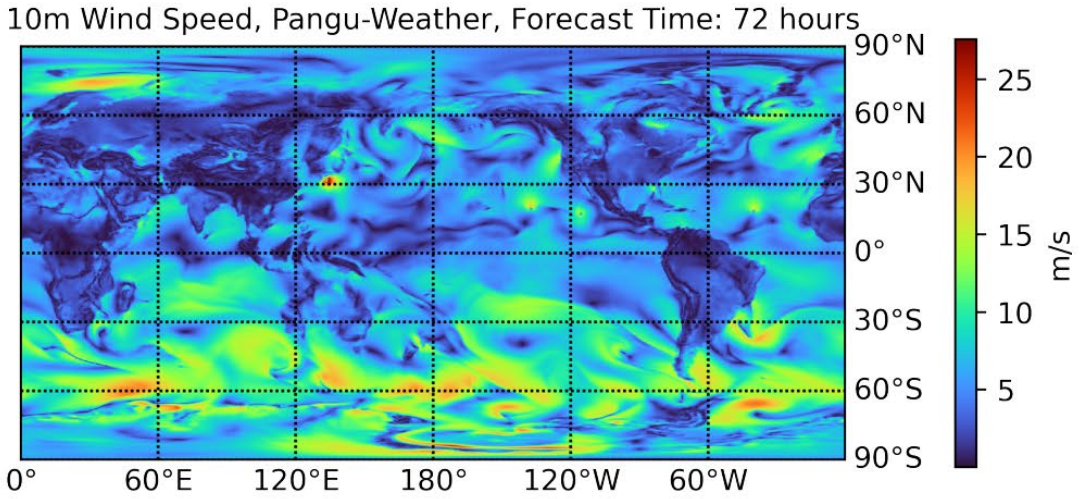
Outline

- Why neural networks?
- NN basics:
  Universal function approximation
  Gradient descent
  Back propagation
- Advanced topics: challenges of using NN is scientific problems
- Coding exercise

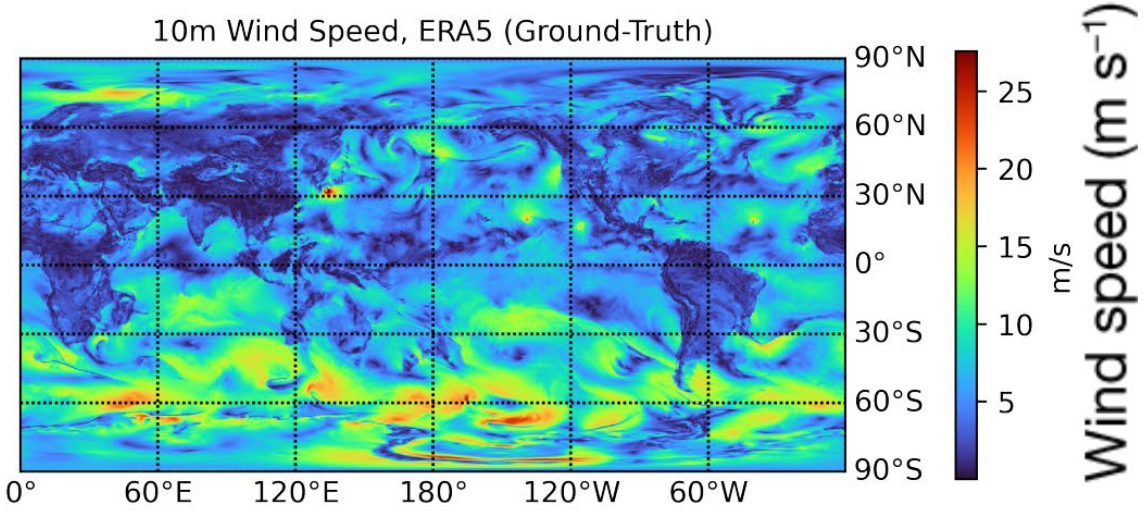
Some challenges:

AI weather forecast

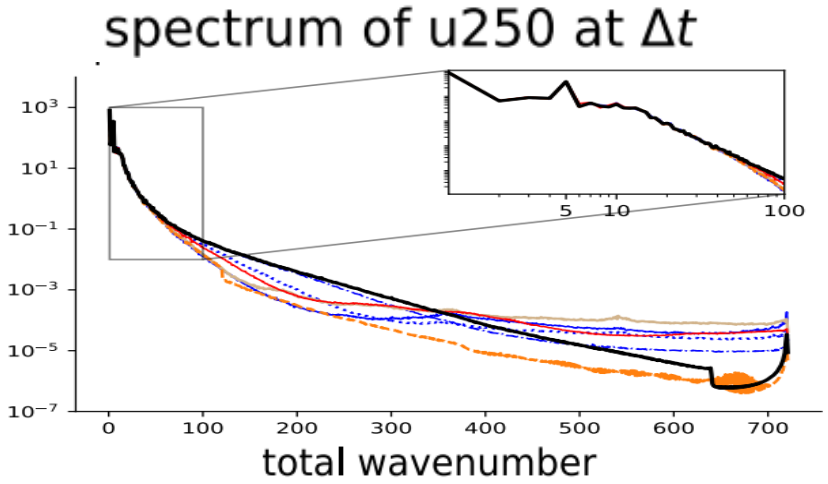
Bi et al (2023) Nature

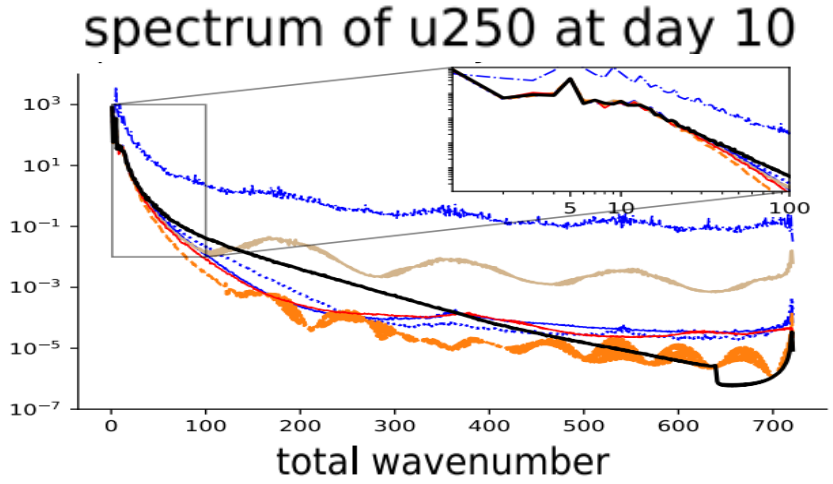


Training data (Re-analysis)



- Spectral bias of NN:





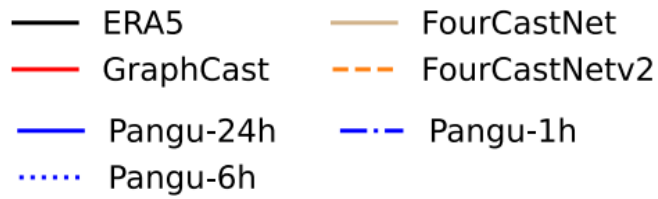
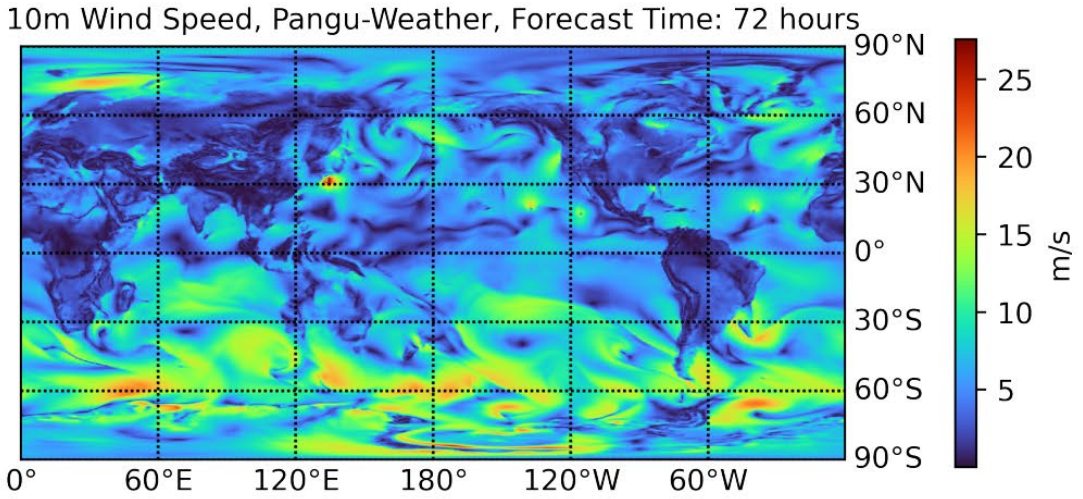


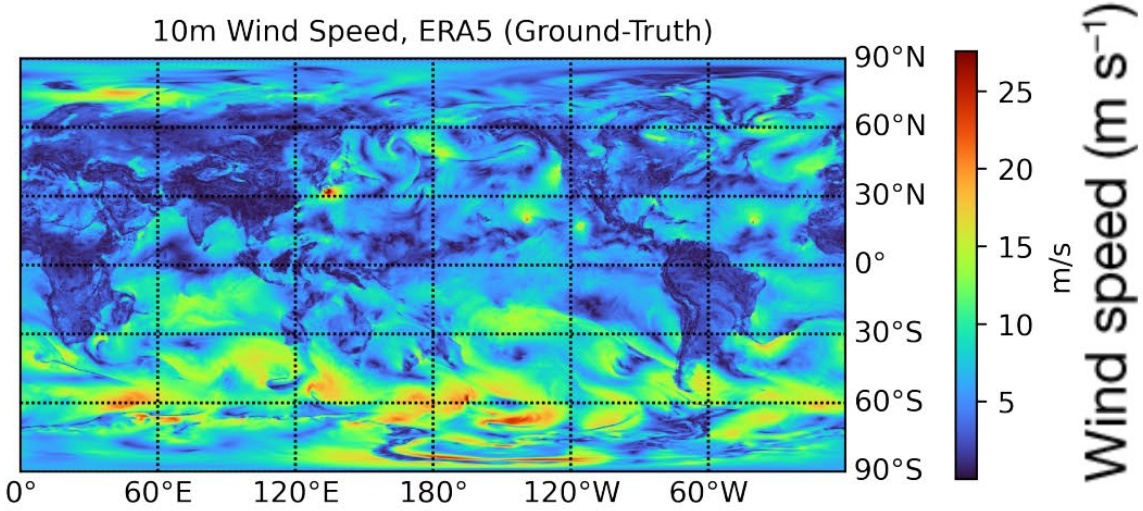
Image source: Qiang Sun & Pedram Hassanzadeh

Spectral bias is bad for multiscale problems

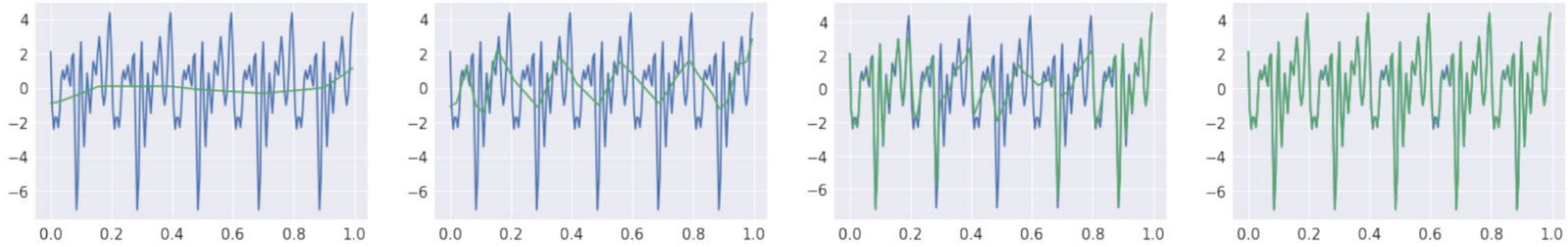
AI weather forecast Bi et al (2023) Nature



Training data (Re-analysis)



- Spectral bias of NN: Rahaman et al (2019), Frequency Principle: Xu et al (2020)

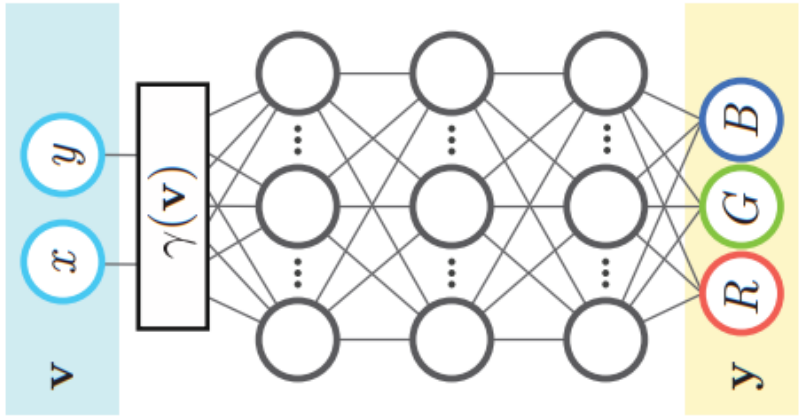


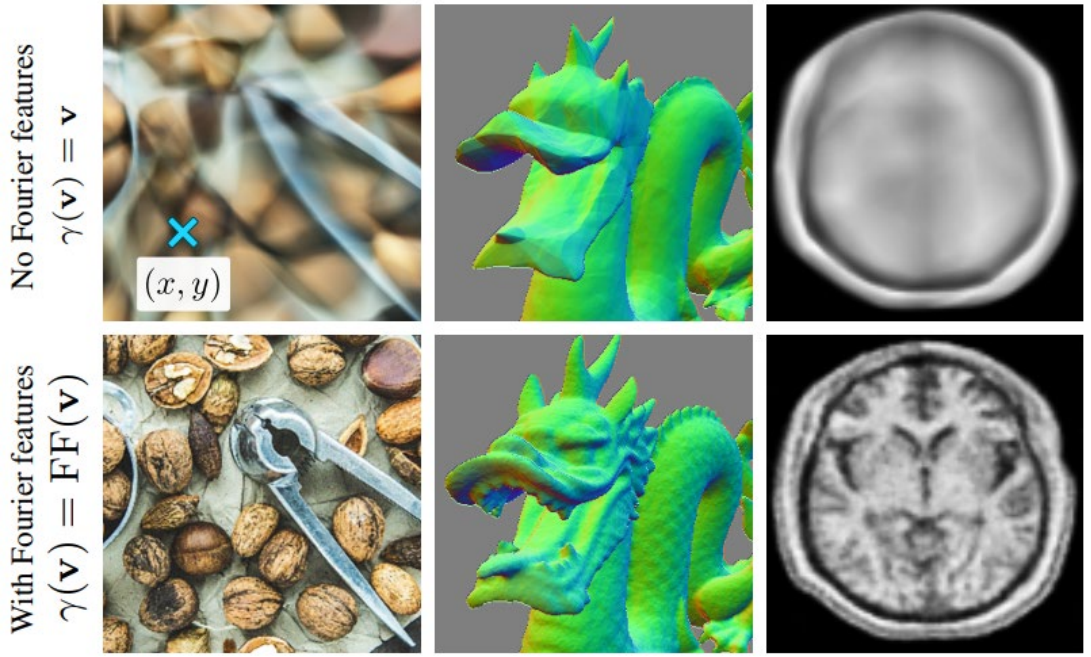
(a) Iteration 100 (b) Iteration 1000 (c) Iteration 10000 (d) Iteration 80000

Spectral bias is bad for multiscale problems

- One solution: Fourier feature $\gamma(\mathbf{v})$ Tancik et al (2020), Wang et al (2021)
  $\gamma(\mathbf{v})$ maps the input coordinates to a feature space before passing them to NN layers.

$$\gamma(\mathbf{v}) = \left[a_1 \cos(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}), a_1 \sin(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}), \ldots, a_m \cos(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v}), a_m \sin(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v})\right]^{\mathrm{T}}$$





(b) Image regression $(x,y) \rightarrow$ RGB | (c) 3D shape regression $(x,y,z) \rightarrow$ occupancy | (d) MRI reconstruction $(x,y,z) \rightarrow$ density

Spectral bias: A Fourier analysis of NN

Xu et al. (2022)

For a single-hidden-layer NN with input $x \in \mathbb{R}$ and output $u \in \mathbb{R}$

$$u(x) = \sum_{j=1}^{m} w_j^{(1)} \sigma(w_j^{(0)} x + b_j), \quad \text{where} \quad \theta_j = \{w_j^{(0)}, w_j^{(1)}, b_j\}$$

The mean square loss measures the distance between the NN output $u(x)$ and ground truth $u_g(x)$

$J \equiv \int_{-\infty}^{\infty} \frac{1}{2}(u(x) - u_g(x))^2 dx = \int_{-\infty}^{\infty} \frac{1}{2}|\hat{u}(k) - \hat{u}_g(k)|^2 dk$, where $\hat{u}(k)$ denotes the Fourier transform of $u(x)$

The loss at frequency k is $J(k) = \frac{1}{2}|\hat{u} - \hat{u}_g|^2$

For tanh activation function, the gradient of loss at $J(k)$ with respect to NN parameters θ_j is

$$\left|\frac{\partial J(k)}{\partial \theta_j}\right| \approx A(k) e^{-\left|\frac{\pi k}{2w_j^{(0)}}\right|}$$ where $A(k) \in [0, +\infty)$ is the amplitude of $\hat{u} - \hat{u}_g$

This gradient decay rapidly with frequency k!! This is a simple explanation of the spectral bias.

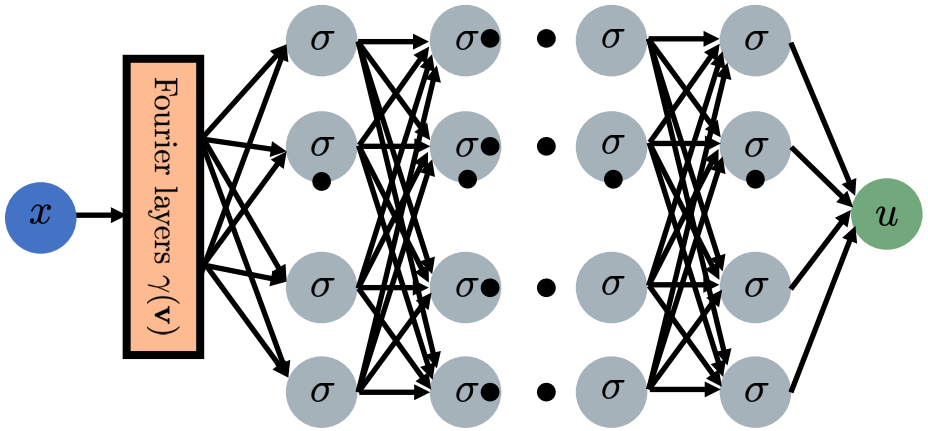
Capturing high-frequency functions

- For multiscale problems, it is important to rescale the NN weights so that it actives the learning of high-frequency (large k) functions

$$\left|\frac{\partial J(k)}{\partial \theta_j}\right| \approx A(k) e^{-\left|\frac{\pi k}{2w_j^{(0)}}\right|}$$

We make $w_j^{(0)} \propto k$

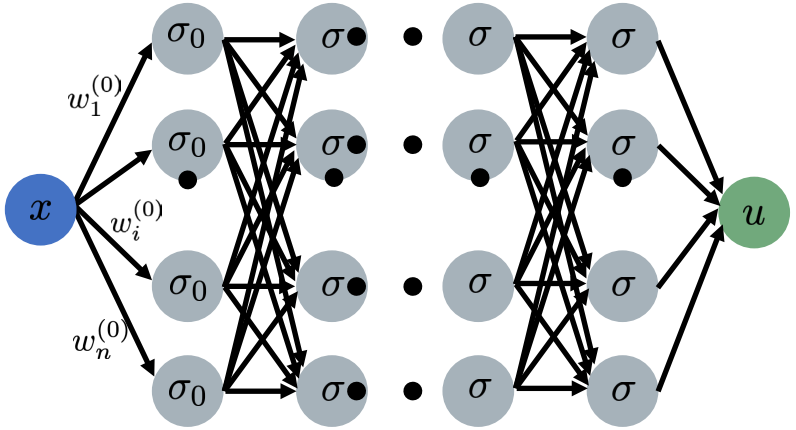
Fourier Feature nets Tancik *et. al.* (2020)



$$\Phi(\mathbf{x}) = \begin{bmatrix} \cos(\mathbf{Bx}) \\ \sin(\mathbf{Bx}) \end{bmatrix} \quad \mathbf{B} \sim \mathcal{N}(0, \kappa^2)$$

Gaussian distribution with standard deviation κ

Our implementation: Wang & Lai, J. Comput. Phys. (2024)



First layer: $\sigma_0(x) = \sin\left(\kappa w_i^{(0)} x + b_i^{(0)}\right)$

κ: scaling factor

Main idea

Wang & Lai, J. Comput. Phys. (2024)

- When NN training error is small, the learning becomes really slow. NN learns better when $u \approx O(1)$. Inspired by the principals of perturbation theory, MSNN includes a superposition of multi-stage NNs, with each stage using a new NN to fit the residue rescaled to O(1) from the previous NN.

$$u_c^{(n)}(x) = u_0(x) + \epsilon_1 u_1(x) + \epsilon_2 u_2(x) + \ldots + \epsilon_n u_n(x) = \sum_{j=0}^{n} \epsilon_j u_j(x)$$

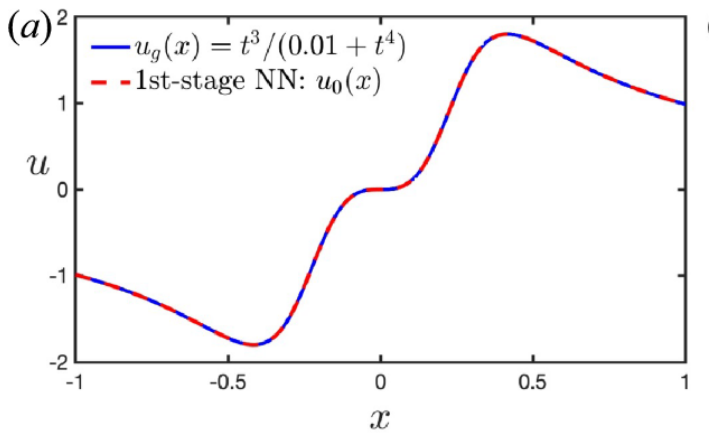
where $1 \gg \epsilon_1 \gg \epsilon_2 \gg \ldots \gg \epsilon_n$

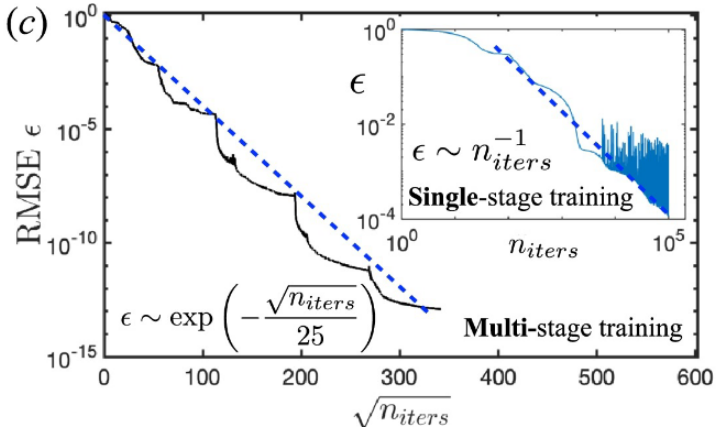
$u_0(x)$, $u_1(x)$ and $u_n(x)$ are all of order $\mathcal{O}(1)$ NNs of different stages

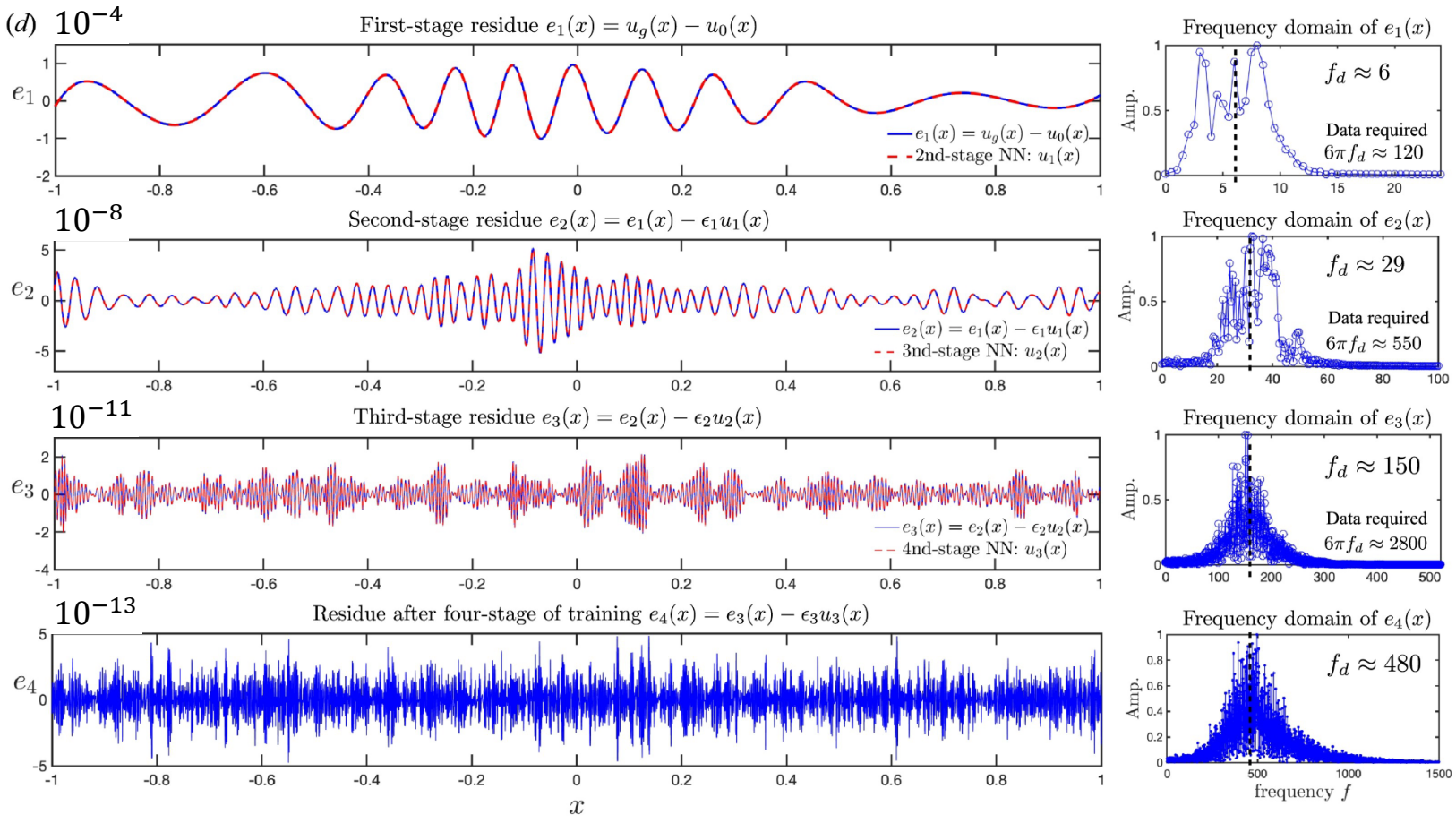
Related work with similar flavors: Multi-level neural networks: Aldirany et al. (2023), Multifidelity neural networks: Howard et al. (2023, 2024), Precision Machine Learning: Michaud et al. (2023)…etc

Multistage-NN (MSNN)

Wang & Lai, J. Comput. Phys. (2024)







Can multistage-NN be used to tackle the spectral bias in multiscale problems?

Example:

How precisely can a NN approximate a 2D fluid flow?

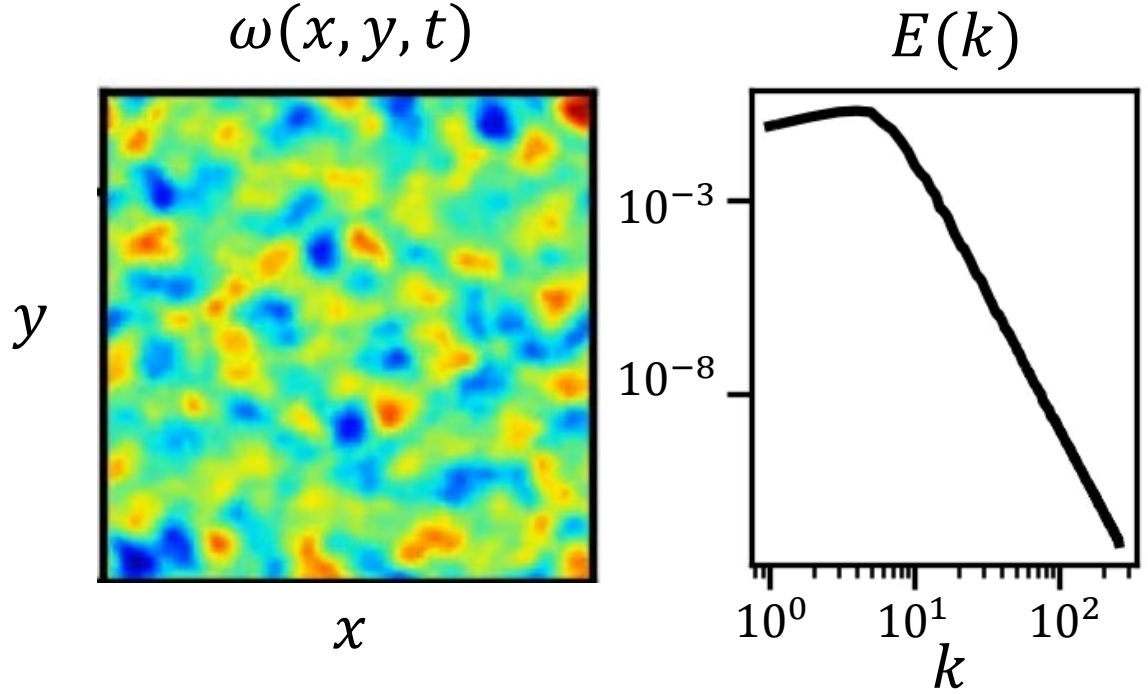
2D incompressible Navier-Stokes eqn:

ω: vorticity
ψ: stream function

$$\frac{\partial \omega}{\partial t} + \frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y} = \frac{1}{\mathrm{Re}}\nabla^2\omega$$

$$\nabla^2\psi = -\omega$$

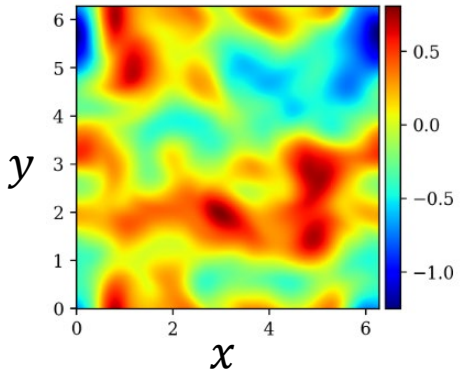
Training data:
Numerical solution at $\mathrm{Re} = 2000$



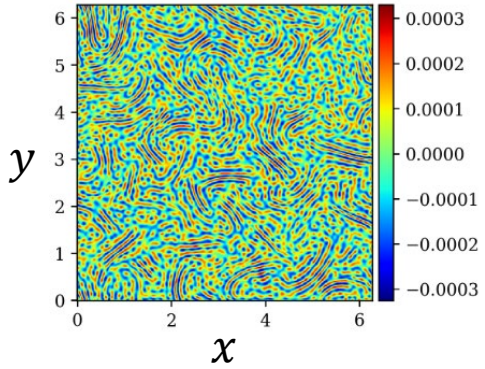
Energy spectrum: $E(k,t) = \sum_{k-\Delta k \le |\boldsymbol{k}| \le k+\Delta k} \frac{1}{2}k^2|\tilde{\psi}(\boldsymbol{k},t)|^2$

E.g. 2D incompressible Navier-Stokes

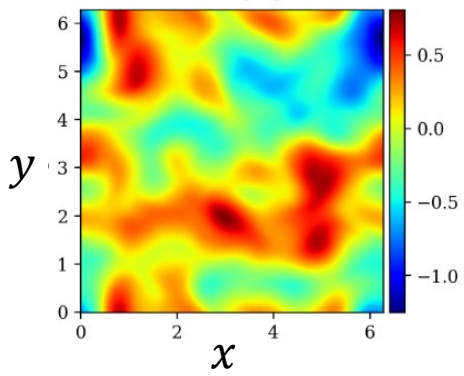
Target $\psi(x, y)$



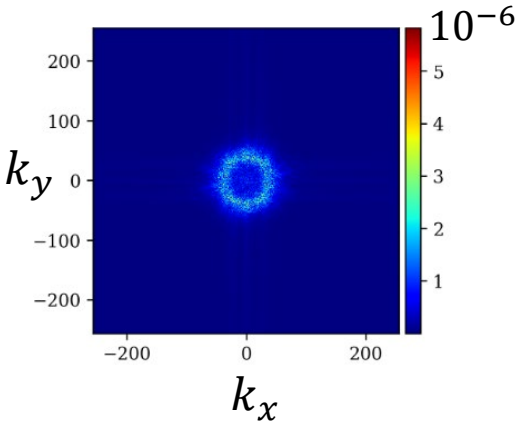
Error



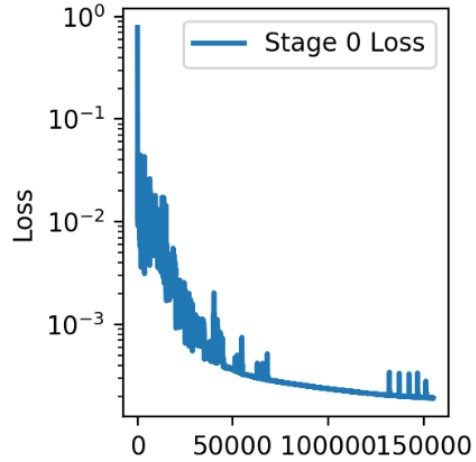
NN $\psi(x, y)$



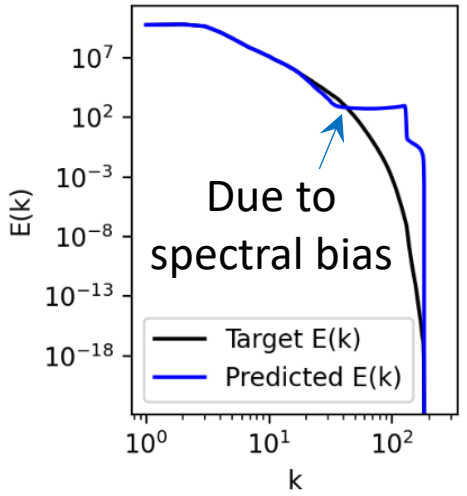
Error in Fourier domain



Loss convergence



Energy spectrum

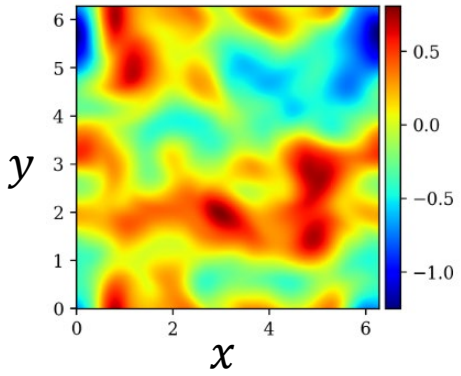


E.g. 2D incompressible Navier-Stokes

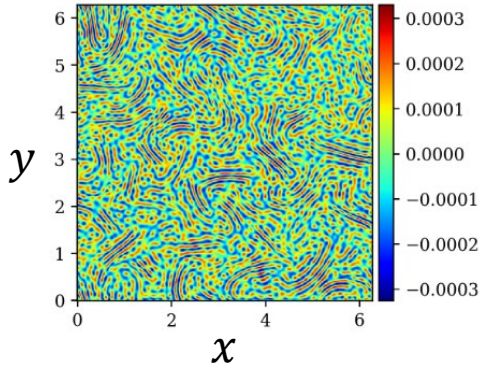
Stage 0

Stage 1

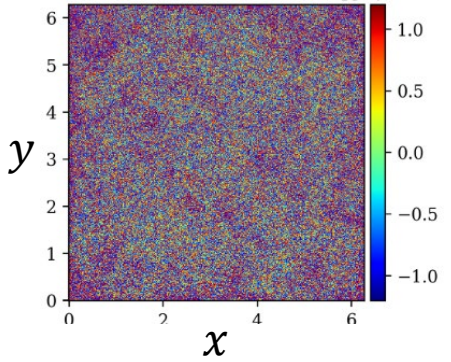
Target $\psi(x, y)$



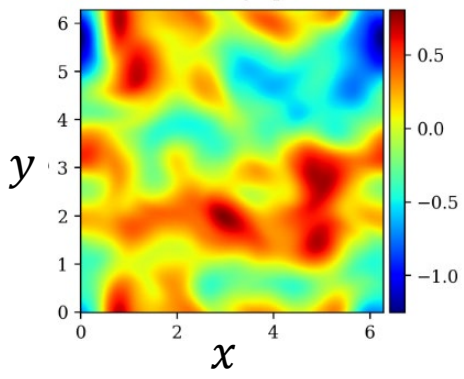
Error



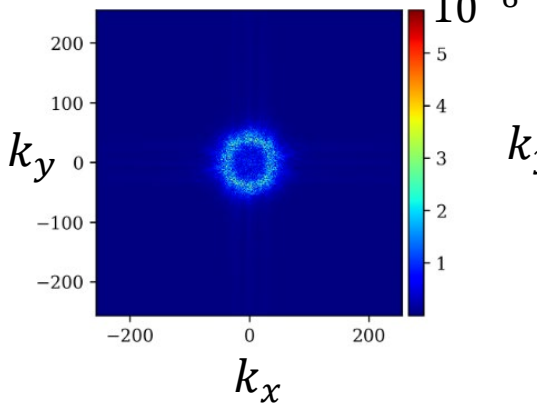
Error



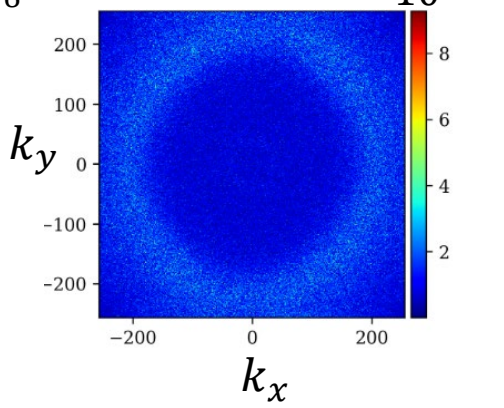
NN $\psi(x, y)$



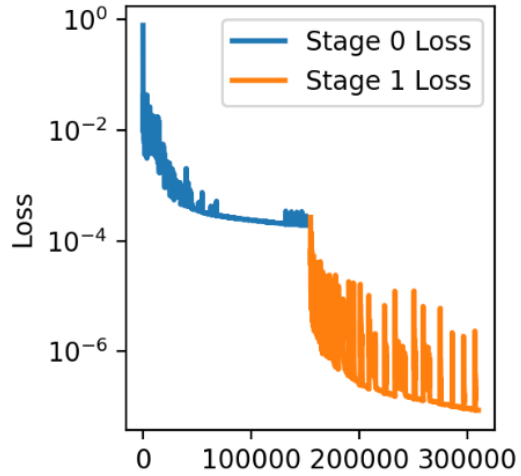
Error in Fourier domain



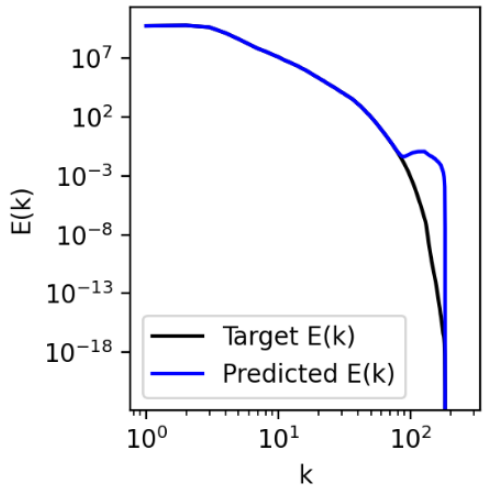
Error in Fourier domain



Loss convergence

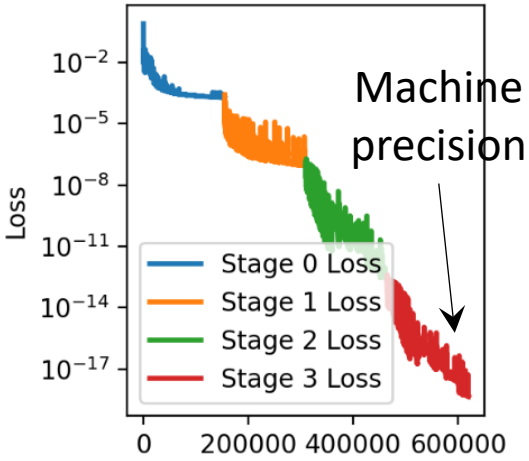


Energy spectrum



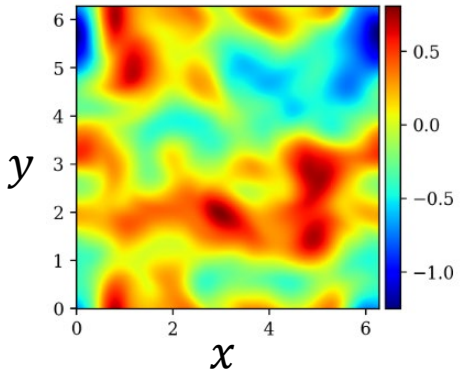
E.g. 2D incompressible Navier-Stokes

Loss convergence

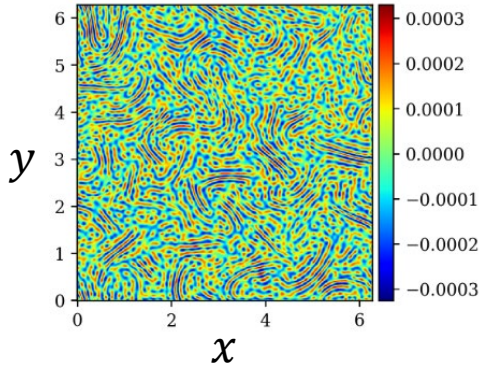


Stage 0 **Stage 1** **…. Stage 3**

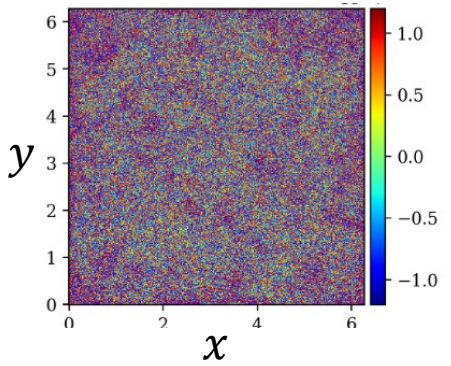
Target $\psi(x, y)$



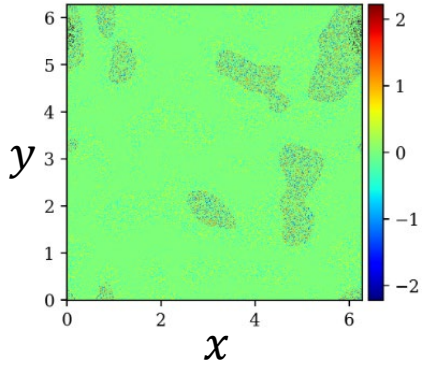
Error



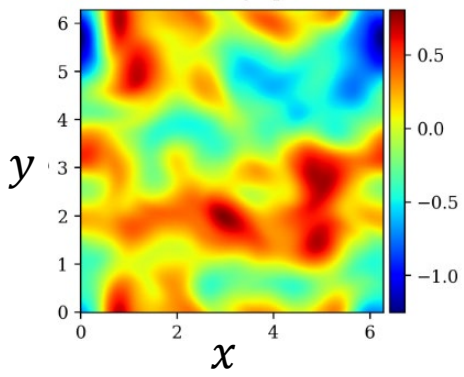
Error 10^{-7}



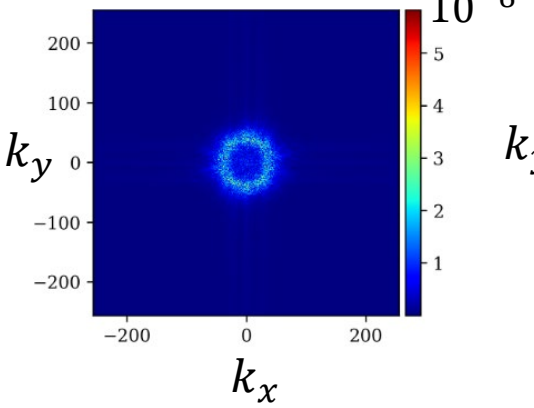
Error 10^{-16}



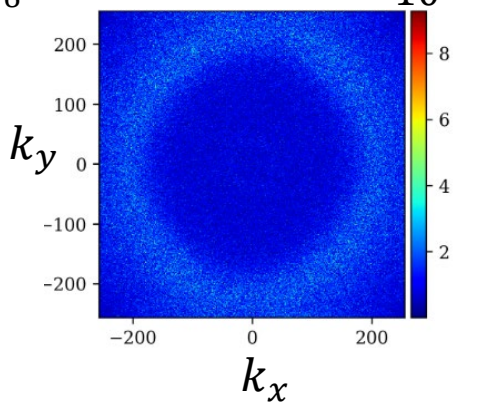
NN $\psi(x, y)$



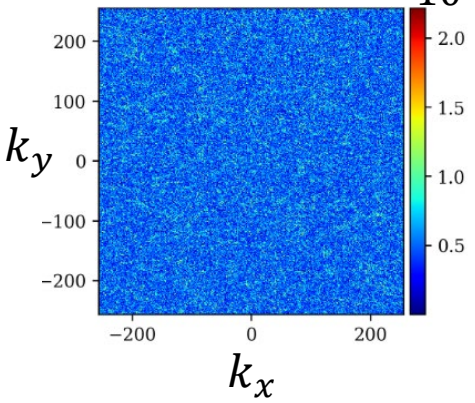
Error in Fourier domain



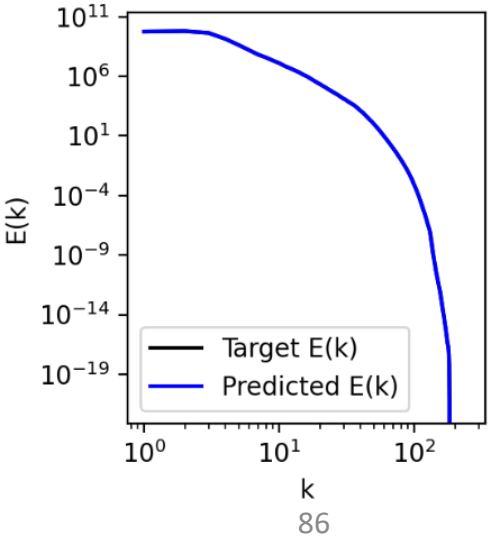
Error in Fourier domain



Error in Fourier domain



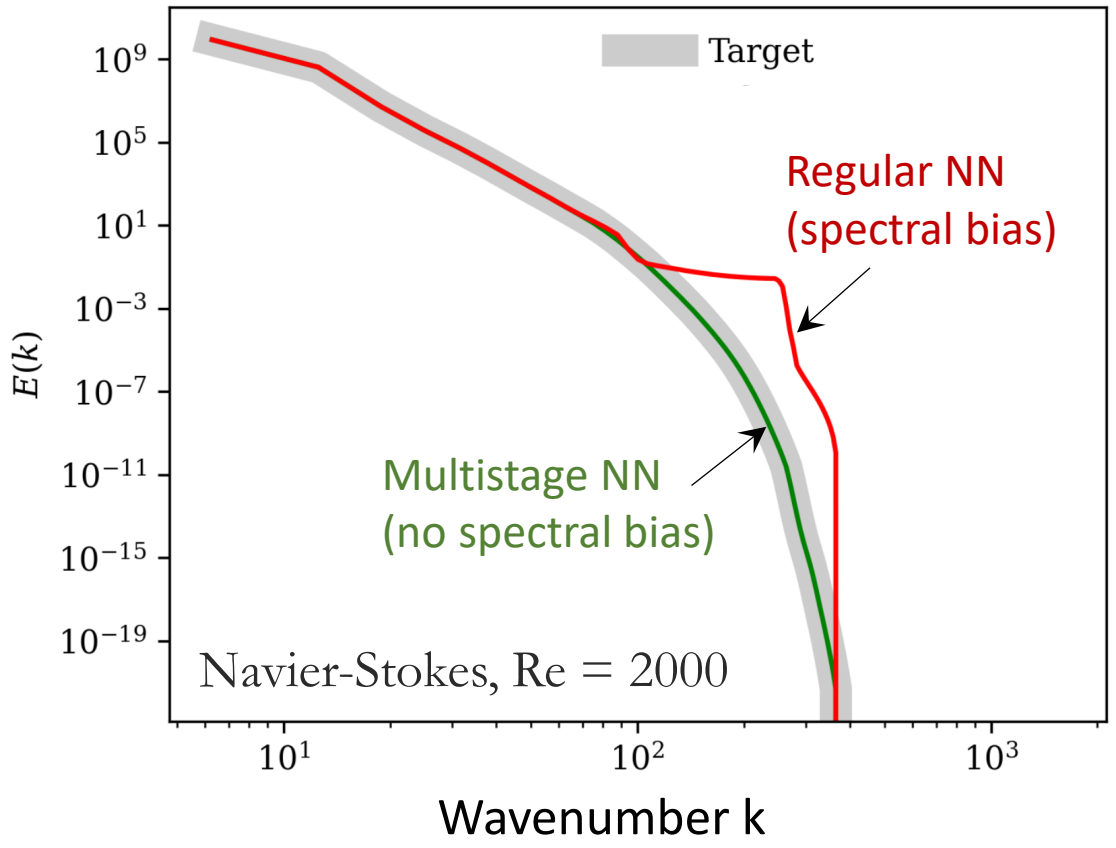
Energy spectrum



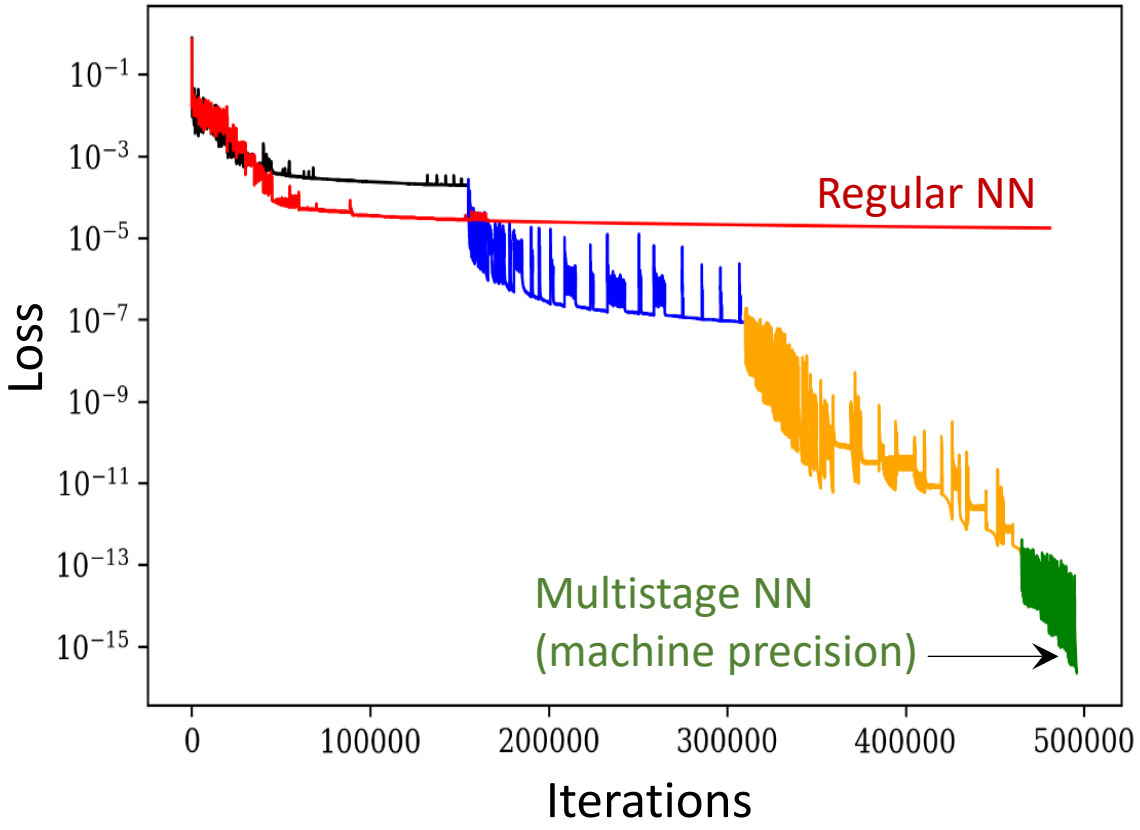
Multistage NN can tackle the spectral bias & accelerate loss convergence

Ng, Wang & Lai, ICML workshop (2024), arXiv:2407.17213

Energy spectrum



Loss convergence

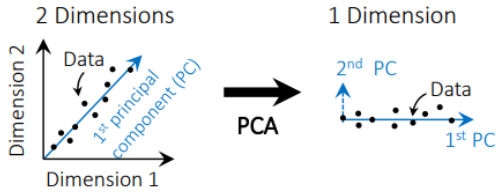


Lastly, taking a step back, how has ML impact climate science as a whole more broadly?

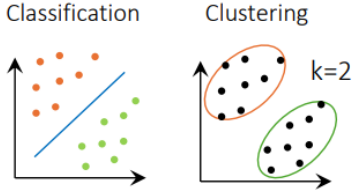
Machine Learning for Climate Physics

I. Data-informed knowledge discovery

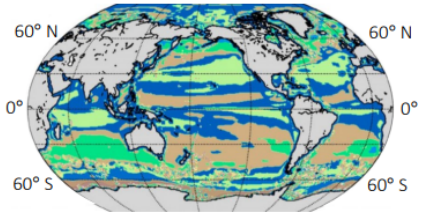
a. Dimensionality reduction



b. Supervised vs unsupervised



c. Discover ocean dynamical regimes with clustering and physics-informed validation

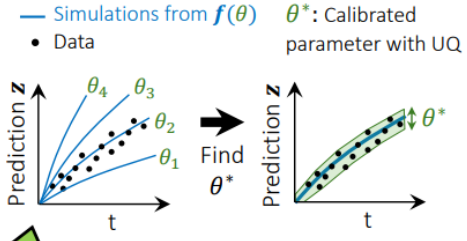


II. Data-informed model discovery

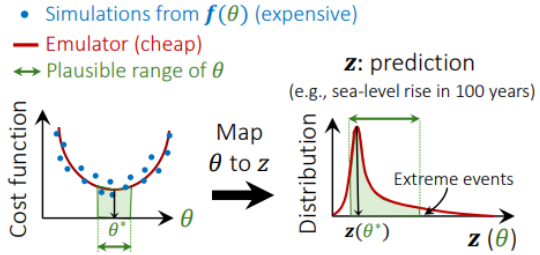
Example: $\dot{\boldsymbol{z}} \equiv [\dot{x}, \dot{y}, \dot{z}]^T = \boldsymbol{f}(\boldsymbol{z}(t), \boldsymbol{\theta})$, where $\mathbf{z}(t)$ are the states, t is time, $\boldsymbol{\theta}$ are model parameters

	Parametric Estimation θ	State Estimation $\boldsymbol{z}(t)$	Structural Estimation f
Goal	Discover parameters of a **mathematical model** that best fits a given dataset.	Discover **states** of a system given sparse, noisy and discrete data.	Discover a **mathematical model** that best fits a given dataset.
Example	Ensemble Kalman Inversion (EKI)	Physics-Informed Neural Networks, Physics-informed Neural Operator	Sparse Regression (e.g., SINDy), Genetic Algorithm
Given	$x(t), y(t), z(t)$ at lots of times, and mathematical form of the model $\boldsymbol{f}$	$\mathbf{x}(t), y(t)$ at sparse times, and the mathematical form of $\boldsymbol{f}$, including $\boldsymbol{\theta}$	$x(t), y(t), z(t)$ at lots of times
Predict	$\boldsymbol{\theta}$	State variables $\mathbf{x}(t), y(t), z(t)$ at lots of times	The mathematical form of the model $\boldsymbol{f}$, including $\boldsymbol{\theta}$

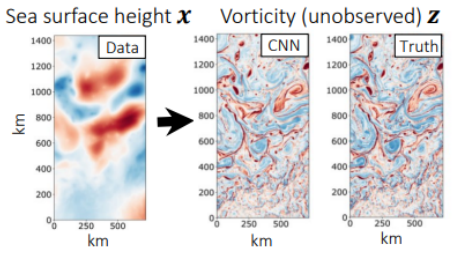
d. Parameter estimation



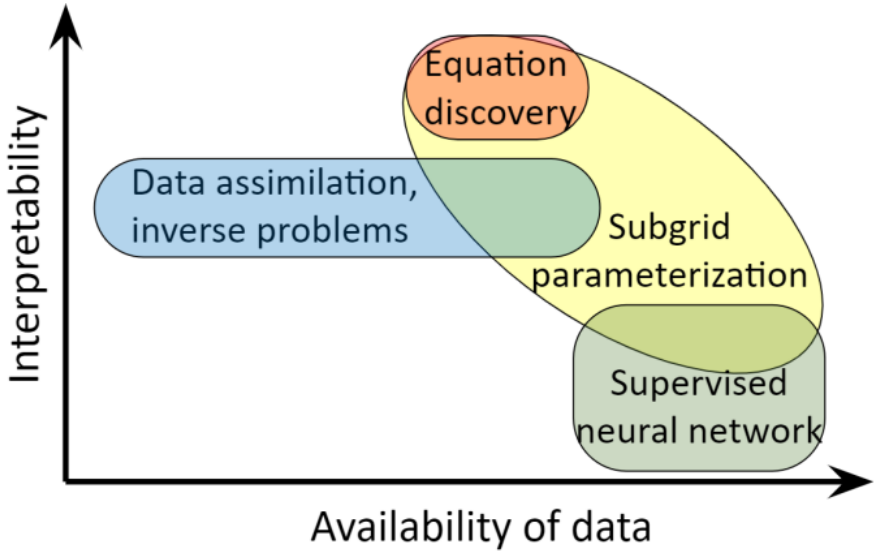
e. Uncertainty quantification (UQ)



f. State estimation example:



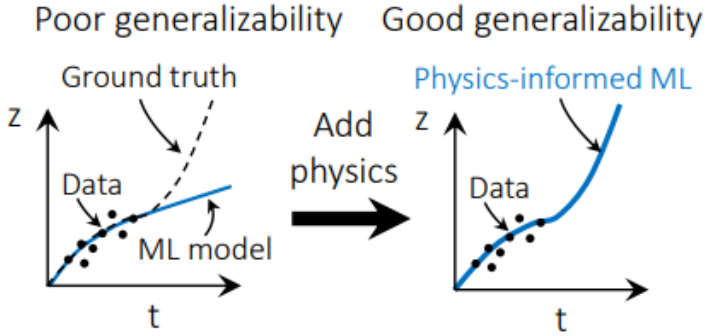
Lai et al., Annu. Rev. Condens. Matter Phys. (2025). doi.org/10.1146/annurev-conmatphys-043024-114758



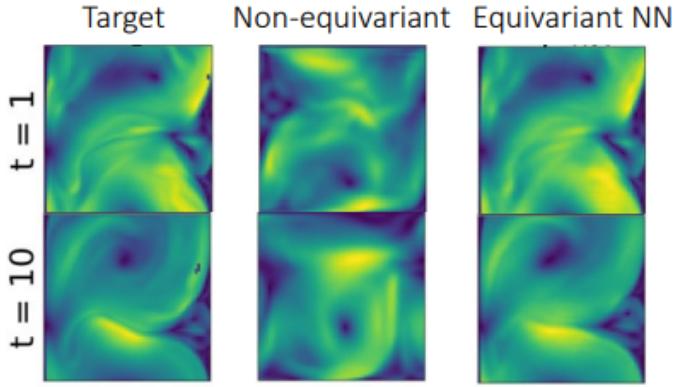
Machine Learning for Climate Simulations

	Subgrid-scale (SGS) modeling	Machine-learning emulator
Goal	Learn a SGS model $\boldsymbol{f}(\bar{\boldsymbol{z}})$ such that $\dot{\bar{\boldsymbol{z}}} = \boldsymbol{f}(\bar{\boldsymbol{z}}, \boldsymbol{\theta}) + \underbrace{\overline{\boldsymbol{f}(\boldsymbol{z})} - \boldsymbol{f}(\bar{\boldsymbol{z}})}_{\approx \Pi(\bar{\boldsymbol{z}})}$	Discover a model **parameterized by ML** that best fits a given dataset
Example	In LES $\bar{\boldsymbol{z}}$ is the coarse-grained $\boldsymbol{z}$, and Π includes Reynolds stress	ClimateBench, FourcastNet, Pengu Weather, GraphCast, ACE
Given	$\bar{\boldsymbol{f}}, \boldsymbol{\theta}, \bar{\boldsymbol{z}}$ at lots of t and its high-resolution counterpart $\boldsymbol{z}$	data of states from previous time step(s) $\boldsymbol{z}(t_i)$
Predict	$\boldsymbol{NN}(\bar{\boldsymbol{z}}, \boldsymbol{\gamma}) = \Pi(\bar{\boldsymbol{z}})$ or symbolic eqn	State at future time step(s) $\boldsymbol{z}(t_{i+1})$

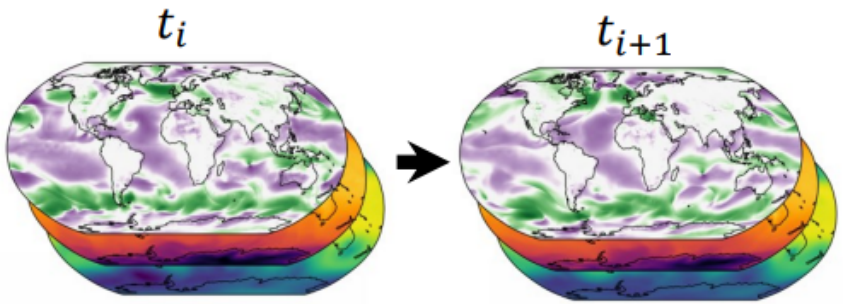
g. Physics-informed ML



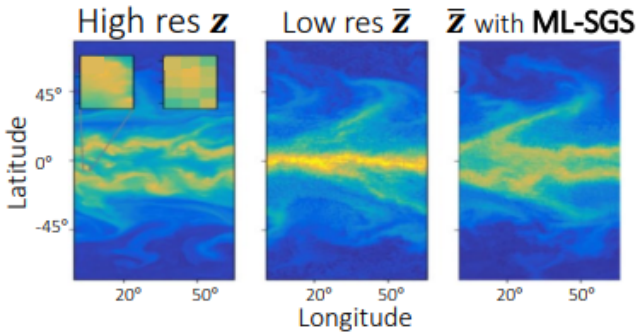
h. Better generalization capacity via incorporating symmetries into NN



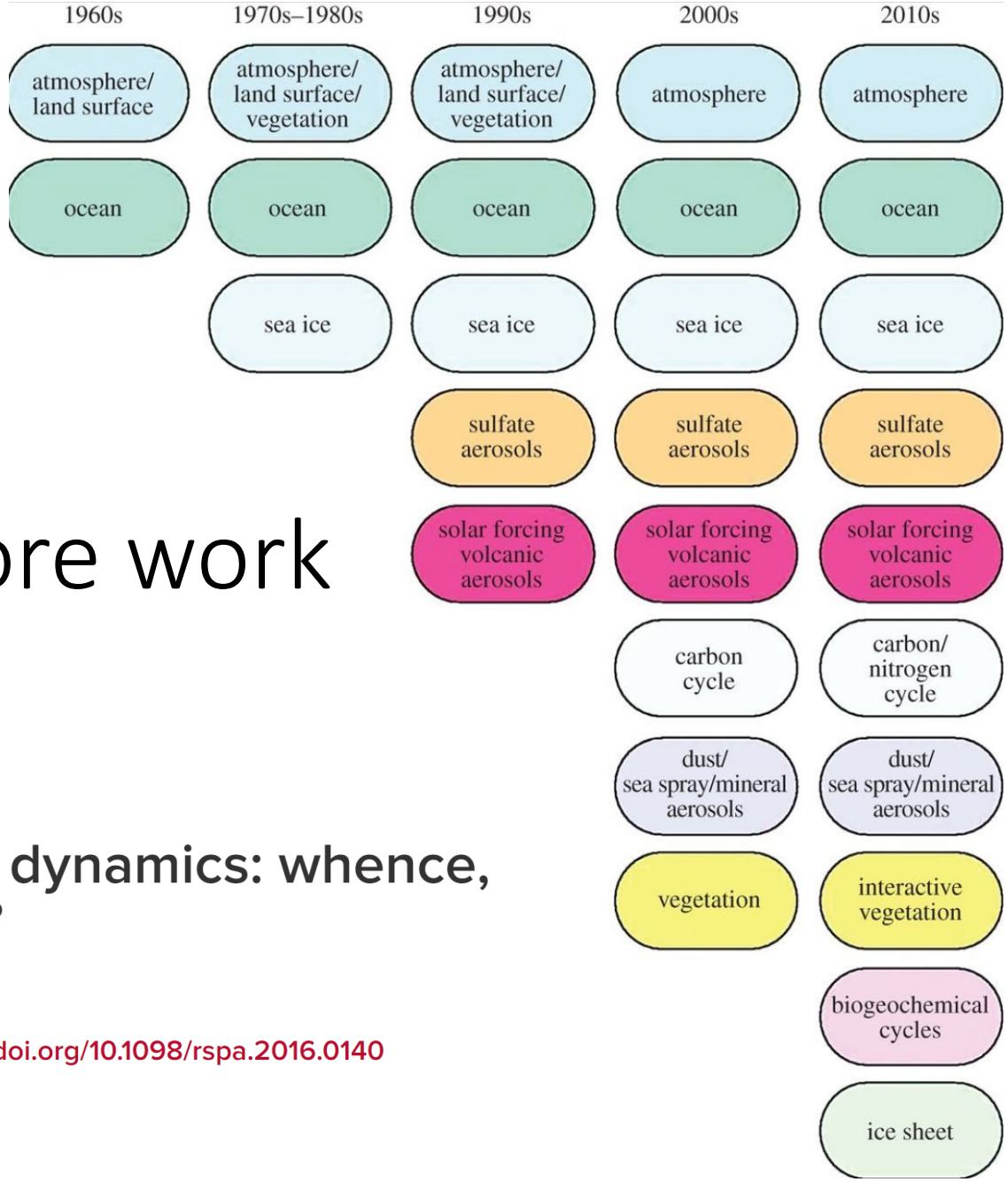
i. Weather forecast via ML emulator



j. SGS parametrization



Random slide:
Ice-sheet model needs more work



Geophysical fluid dynamics: whence, whither and why?

Geoffrey K. Vallis ✉

Published: **01 August 2016** | https://doi.org/10.1098/rspa.2016.0140

The best way to learn ML is through projects

- Don't worry about failures. In fact, the more the better. This is the only way neural networks learns. This is how we learn!
- Analogy: