

Neural networks basics

by C. Yao Lai for 2024 Glamacles summer school

ML for cryosphere observations

40+ papers since 2019!

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

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 (Dirschler et al. 2020, Dell et al. 2021)
- Supraglacial water – Greenland (Dunmire et al. 2021*)
- 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*)

List compiled by Yao

ML for cryosphere modeling

20+ papers since 2021!

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

- **Surrogate/emulation model (forward modeling):**

CNN (IGM: Jouvet et al. 2021, Verjans & Robel 2024)

DeepONet (Howard, Perego et al. 2023, He, Perego et al. 2023)

LSTM (Katwyk et al. 2023)

GNN (Rahnemoonfar and Koo 2024)

- **Data assimilation (inverse modeling):**

Bayesian inference + NN (Brinkerhoff et al. 2021)

PINN (Riel et al. 2021, Diffice: Wang, Lai et al. 2022, Cheng, Morlighem et al. 2024)

GP (Downs et al. 2022)

UDE (ODINN: Bolibar, Sapienza et al. 2023)

CNN (Jouvet 2023)

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. tensorflow, 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!

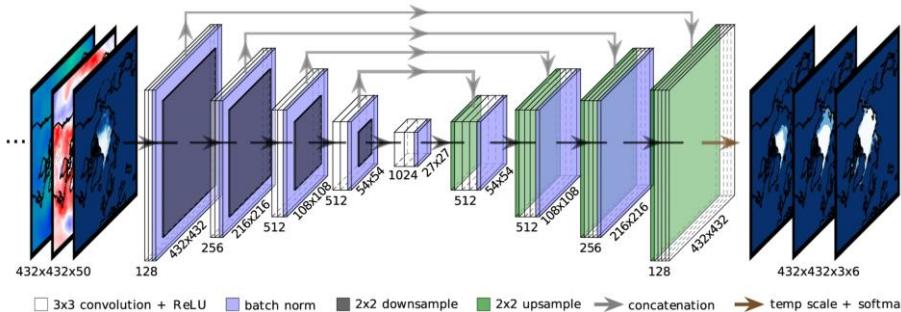
CNN for parameterizing climate data

- 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](#)



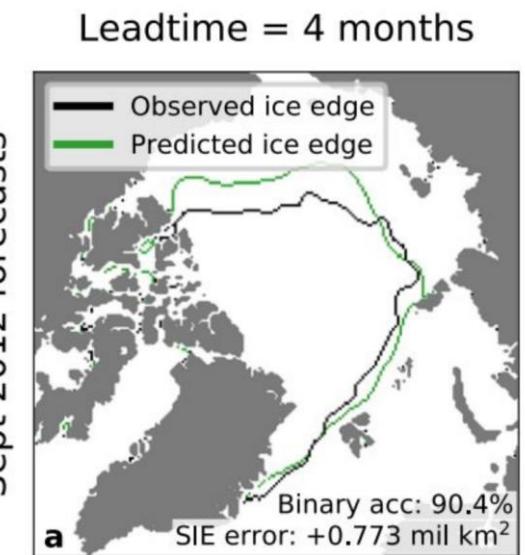
Input: 50 different climate variables (e.g. temperature, wind speed...etc)

Output: 3 sea ice concentration classes in 6 forecast months

Leadtime = 1 month



Sept 2012 forecasts



Leadtime = 4 months

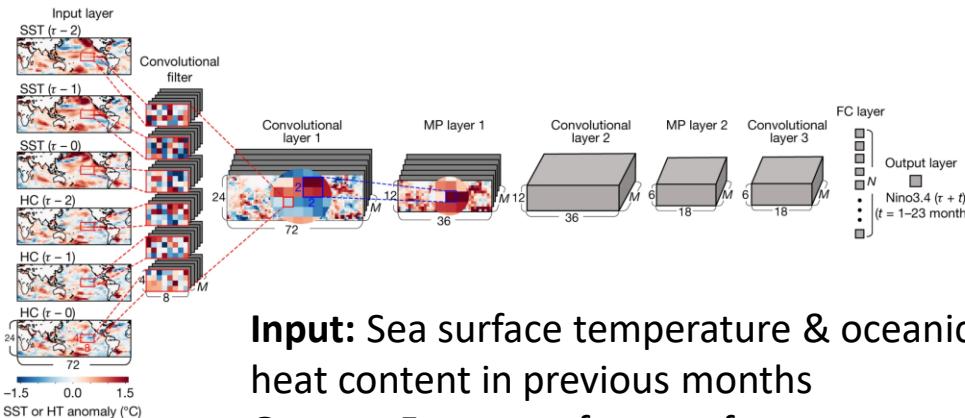
CNN for parameterizing climate data

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

Deep learning for multi-year ENSO forecasts

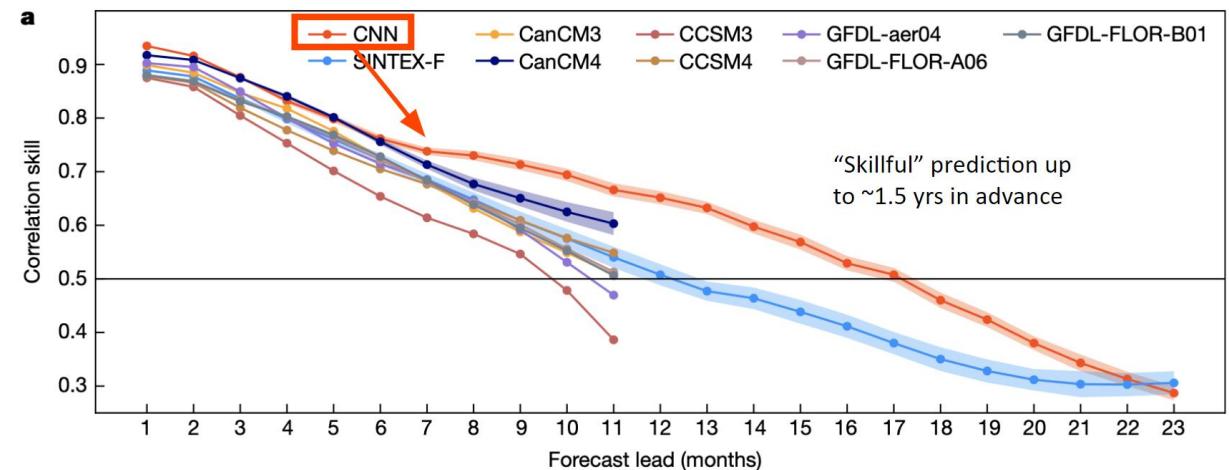
Yoo-Geun Ham✉, Jeong-Hwan Kim & Jinq-Jia Lu

Nature **573**, 568–572 (2019) | Cite this article



Input: Sea surface temperature & oceanic heat content in previous months

Output: Forecast of sea surface temperature timeseries 1-23 months later

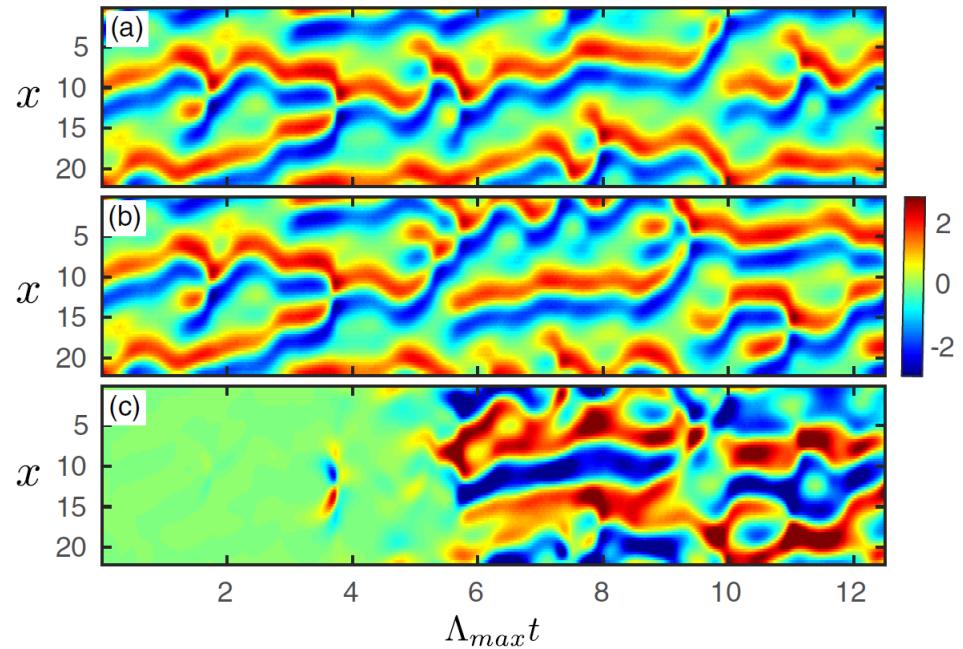


Surrogate models

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

Kuramoto-Sivashinsky (KS)

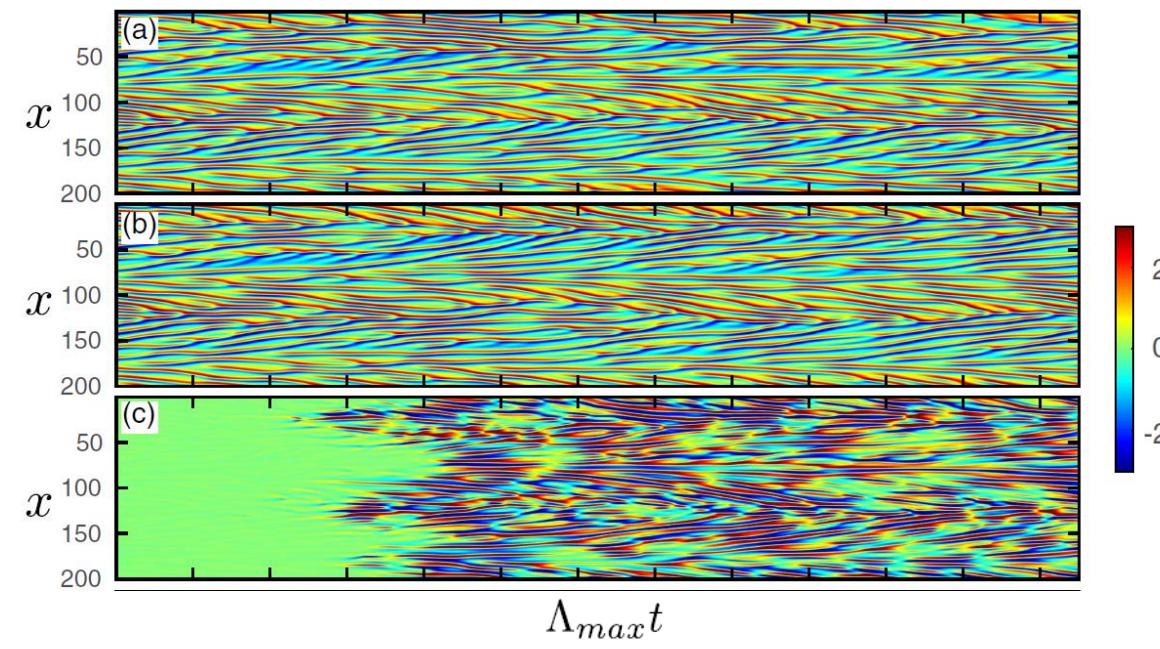
$$y_t = -yy_x - y_{xx} - y_{xxxx} + \mu \cos\left(\frac{2\pi x}{\lambda}\right)$$



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}

¹Institute for Research in Electronics and Applied Physics, University of Maryland, College Park, Maryland 20742, USA



Physics-free ML: Using machine learning for model-free prediction of spatiotemporally chaotic systems purely from observations of the system's past evolution.

Surrogate models for simulations

Model of time-dependent systems:

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

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

Train a 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.

Training data (Numerical simulation)

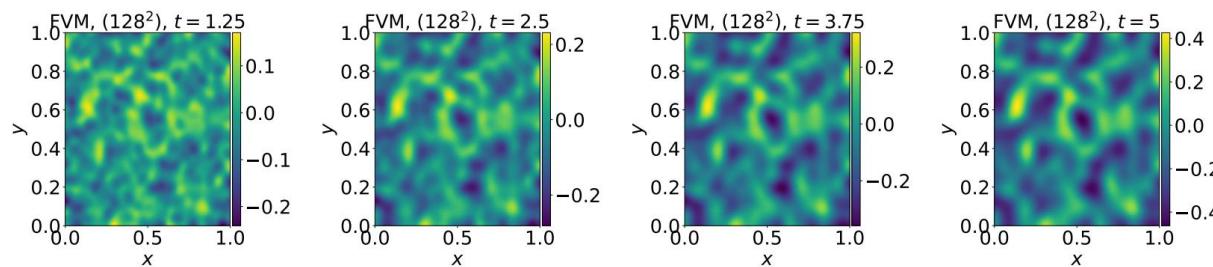


Figure 15: Visualization of the time evolution of the 2D diffusion-reaction equation data.

ML surrogate (FNO)

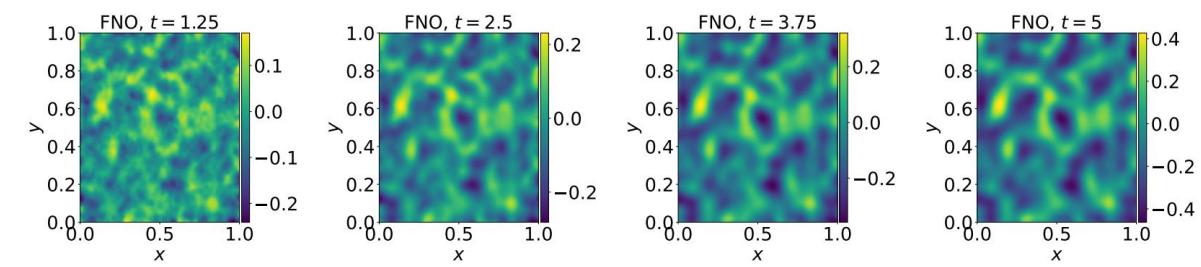


Figure 16: Visualization of the time evolution of the 2D diffusion-reaction equation predicted using FNO.

PDEBENCH:

<https://arxiv.org/pdf/2210.07182.pdf>

Github:

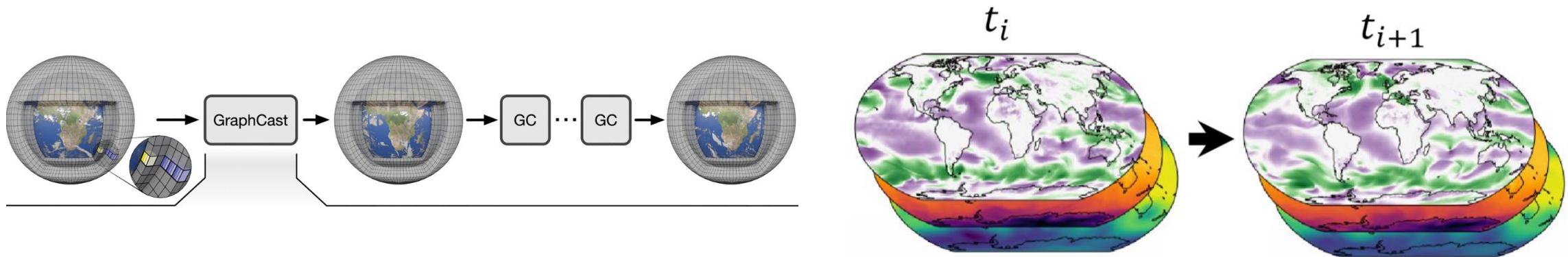
<https://github.com/pdebenc/PDEBench>

Surrogate models for simulations

Model of weather:

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

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



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.

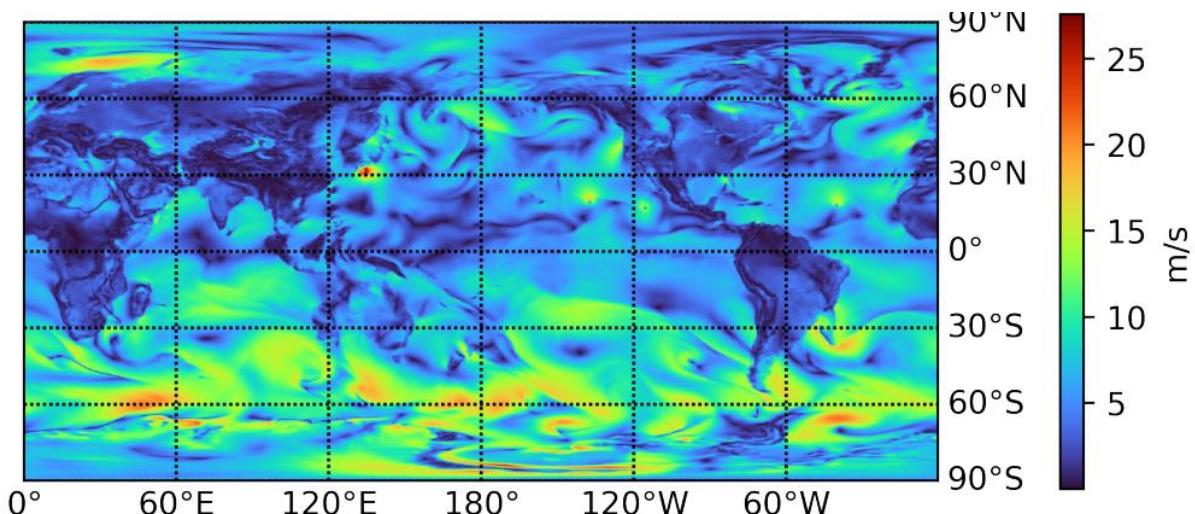
ML weather forecast models

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?

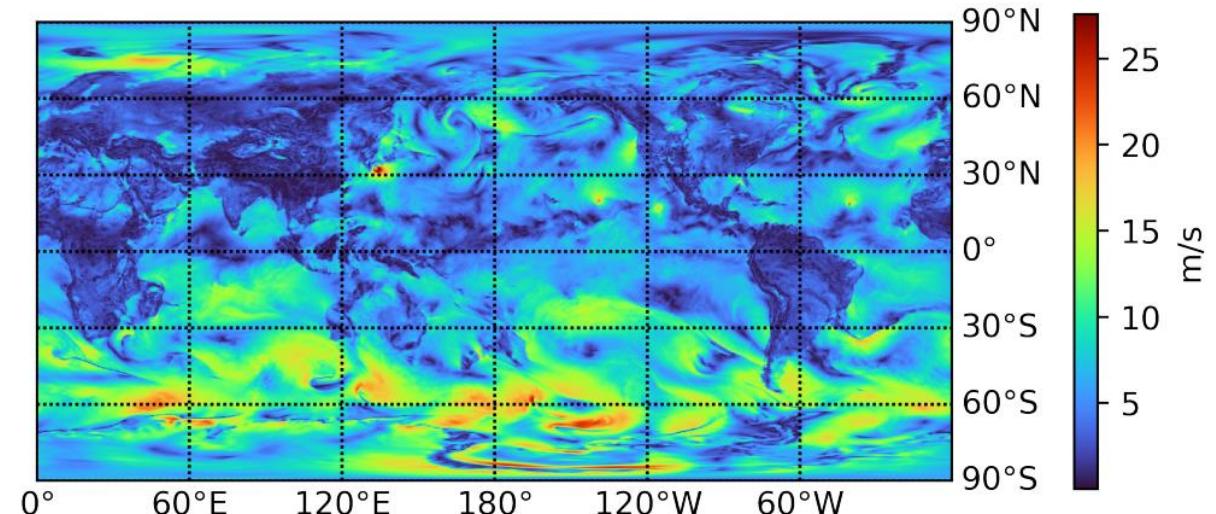
A



AI weather forecast

Bi et al (2023) Nature

B



Training data
(Numerical weather prediction simulation)

Outline

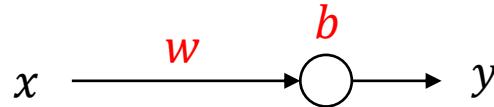
- Why neural networks?
- NN basics:
 - Universal function approximation
 - Gradient descent
 - Back propagation
- Advanced topics: challenges of using NN in scientific problems
- Coding exercise

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:

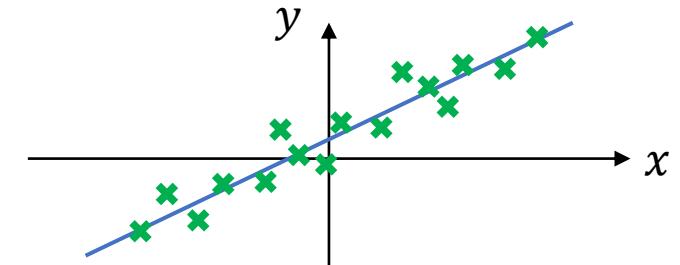
$$y = \mathbf{w}x + b$$



Given observations of $\{\mathbf{x}_d^i, y_d^i\}_i^n$

Find the w and b that minimizes

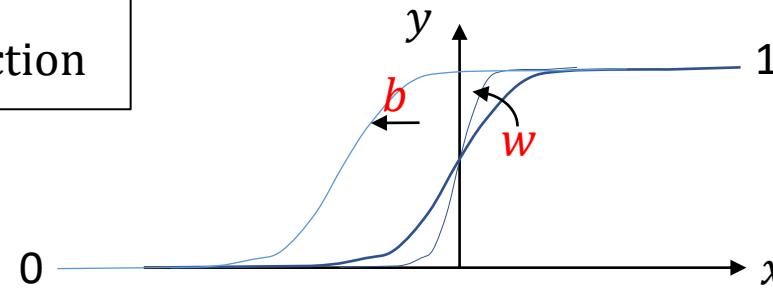
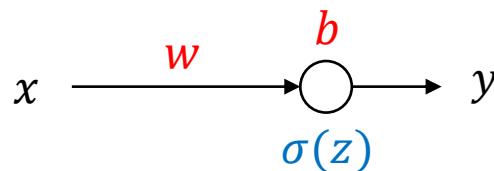
$$J = \sum_{i=1}^n (y(\mathbf{x}_d^i) - y_d^i)^2$$



2. Logistic Regression Model: make output 0 to 1

$$y = \sigma(\mathbf{w}x + b),$$

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



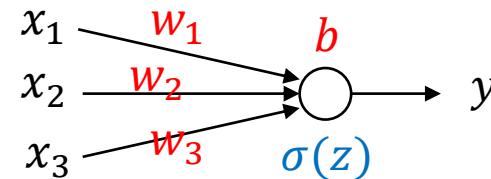
What is a neural network?

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

$$y = \sigma(w_1x_1 + w_2x_2 + w_3x_3 + b),$$

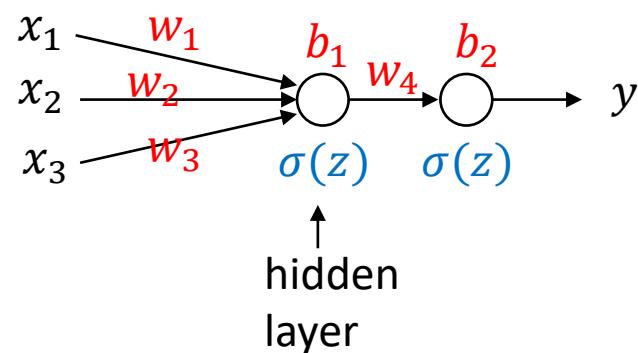
where $\sigma(z) = \frac{1}{1 + e^{-z}}$ is a sigmoid function



3. Neural network:

$$y = \sigma(w_4\sigma(w_1x_1 + w_2x_2 + w_3x_3 + b_1) + b_2),$$

where $\sigma(z)$ is a nonlinear activation function



Common choices of $\sigma(z)$

$\text{sigmoid}(z)$
 $\sin(z)$
 $\cos(z)$
 $\tanh(z)$

...

Output ranges from -1 to 1

What is a neural network?

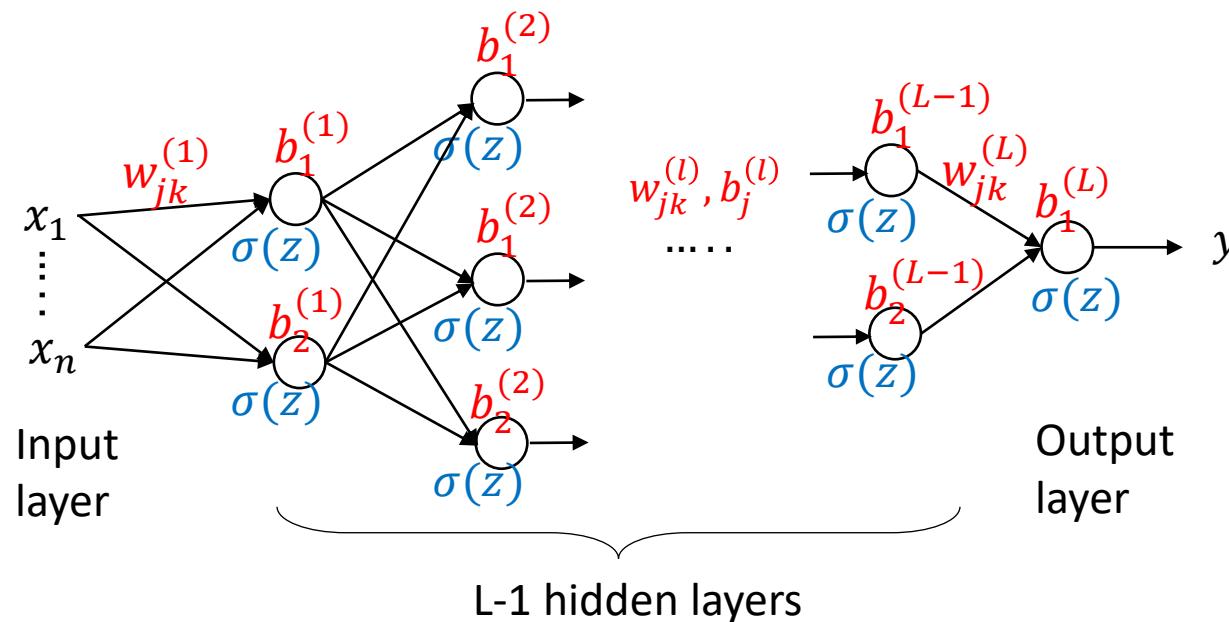
An analytical model of output y as a function of input x , containing some fitting parameters

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

More generally...

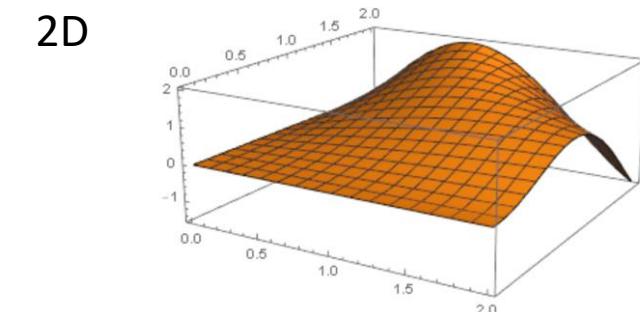
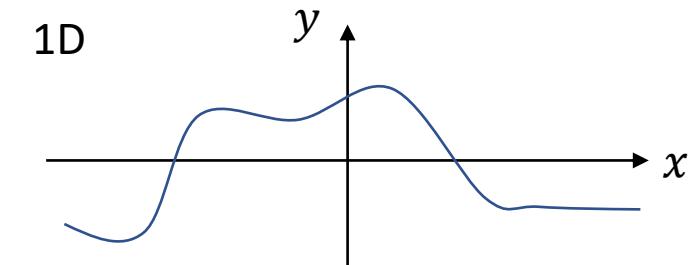
$$y = fn(w_{jk}^{(l)}, b_j^{(l)}, x_i),$$

where $\sigma(z)$ is a nonlinear activation function



Neural network is a general function approximation

-> Universal function approximation



n-Dimension surface....

Why are activation function nonlinear?

Common choices of $\sigma(z)$

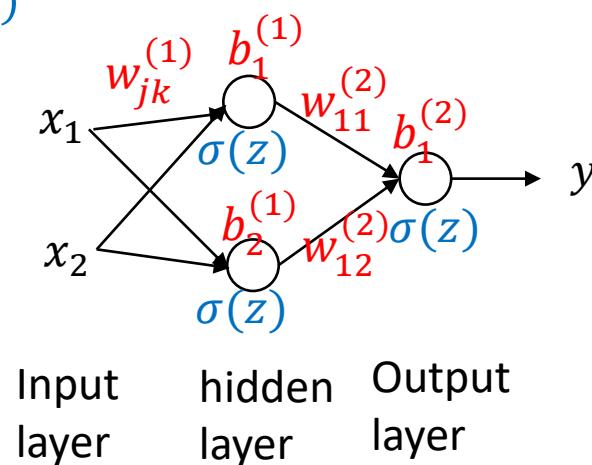
$\text{sigmoid}(z)$

$\sin(z)$

$\cos(z)$

$\tanh(z)$

...



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

Input x_1, x_2

In hidden layer, $j=1, 2$

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

In output layer, $j=1$

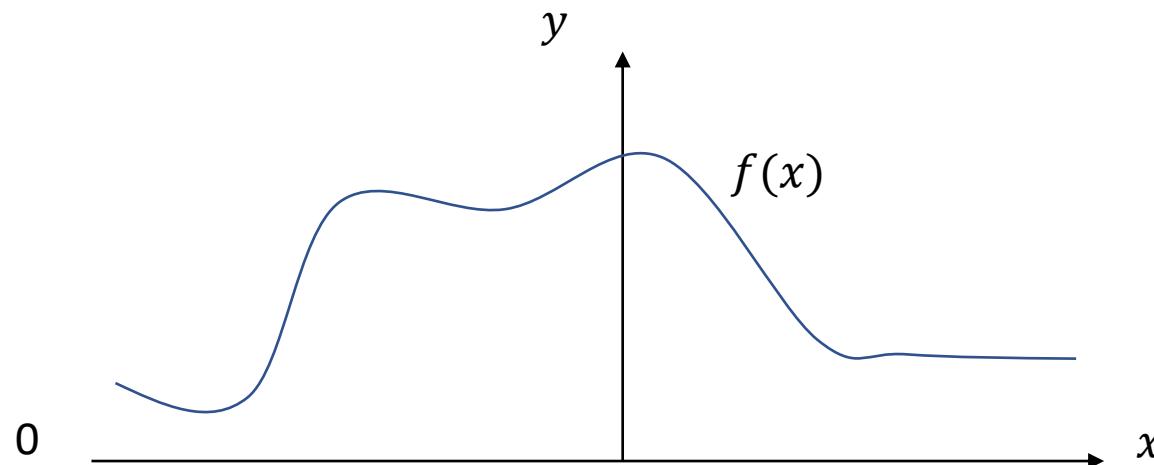
$$z_j^{(2)} = \sum_{k=1}^2 w_{jk}^{(2)} a_k^{(1)} + b_j^{(2)}$$
$$a_j^{(2)} = \sigma(z_j^{(2)}) = z_j^{(2)}$$

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

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

NN is a Universal Function Approximator

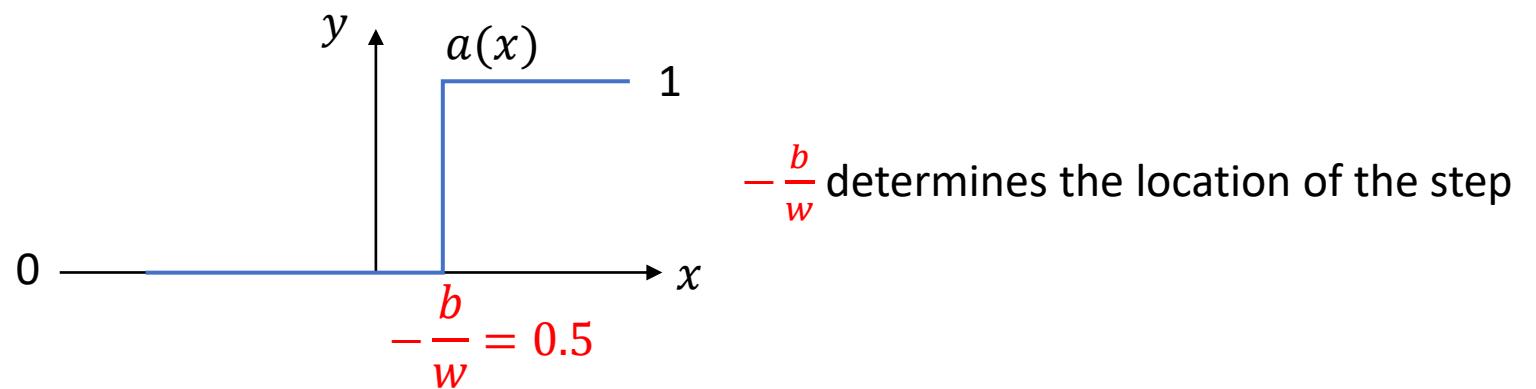
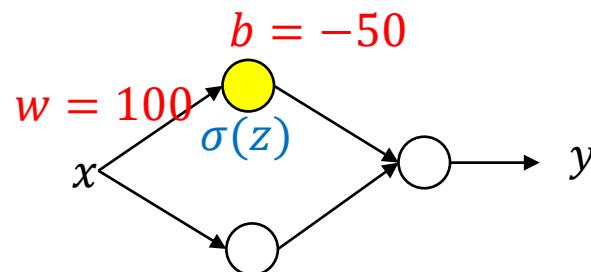
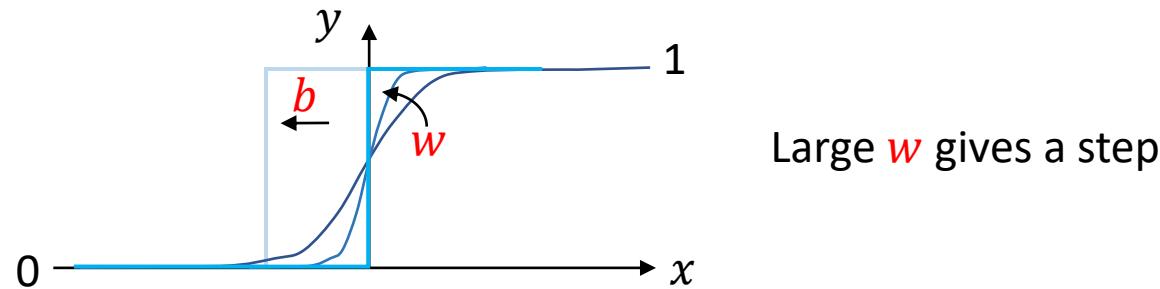
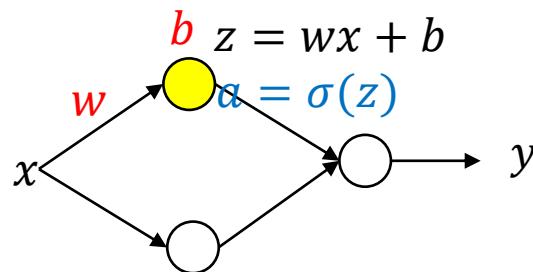
- NN can approximate continuous and smooth functions
A visual proof (for sigmoid activation)



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

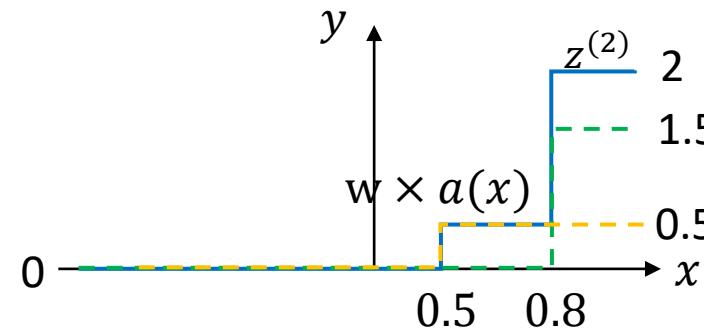
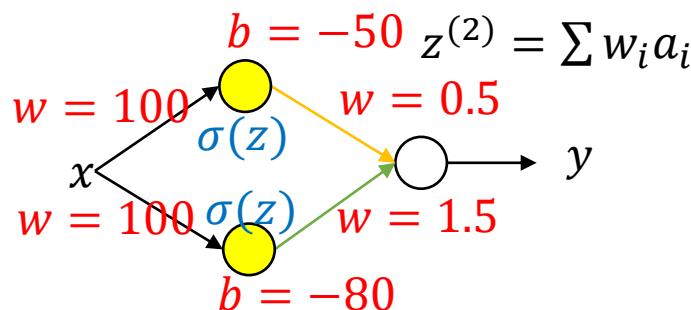
NN is a Universal Function Approximator

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

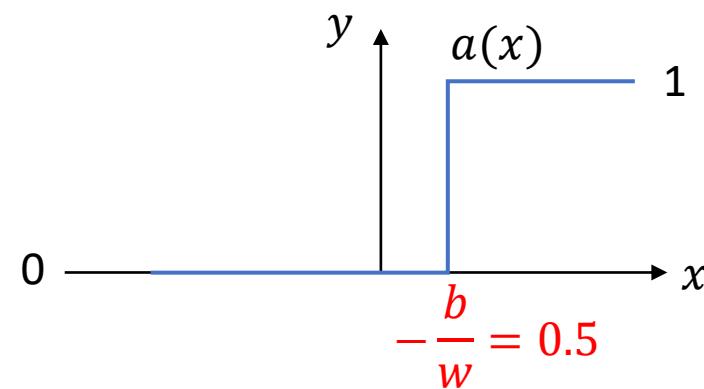
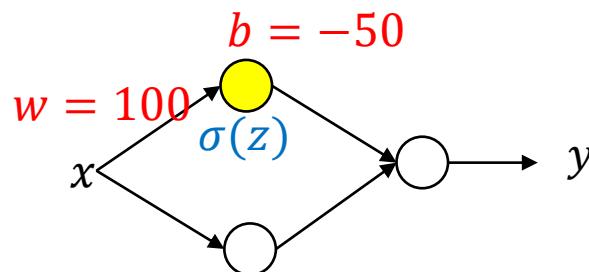


NN is a Universal Function Approximator

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



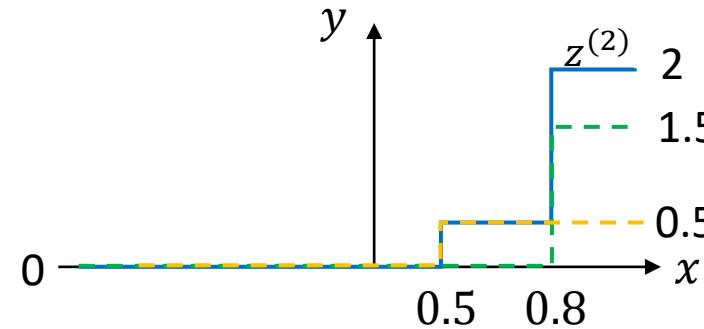
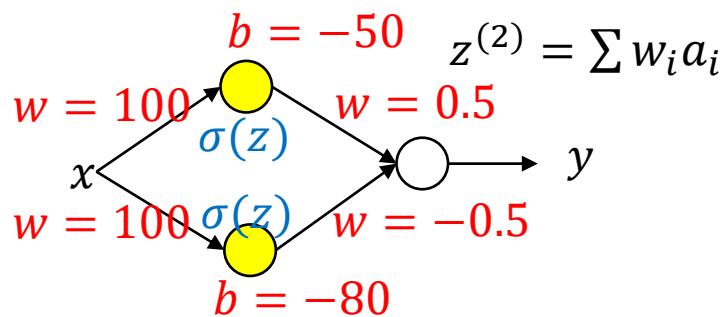
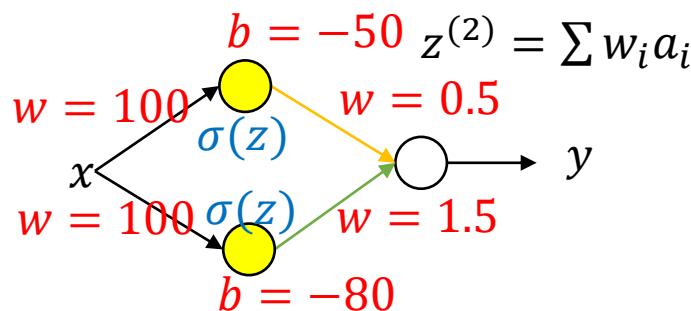
Superposition of two steps



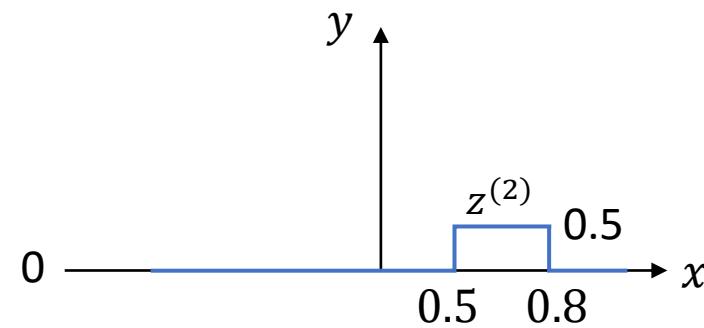
$-\frac{b}{w}$ determines the location of the step

NN is a Universal Function Approximator

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



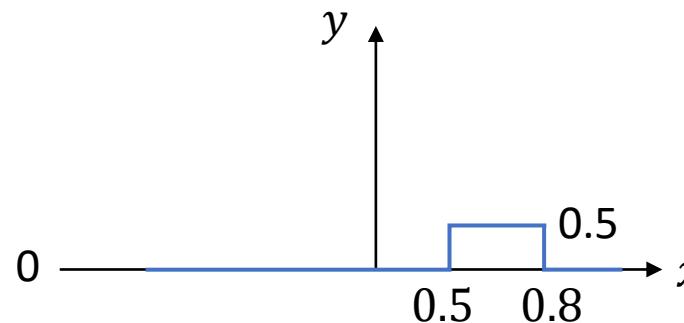
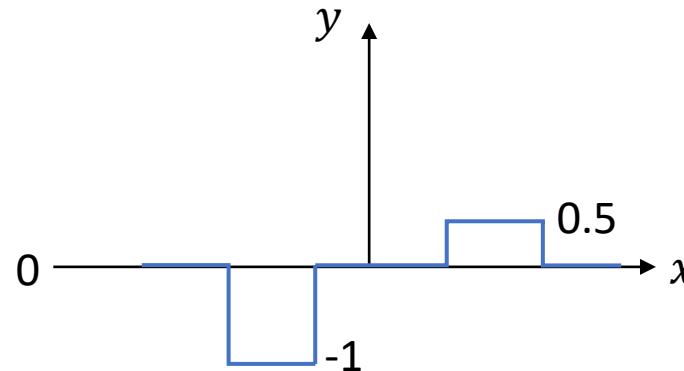
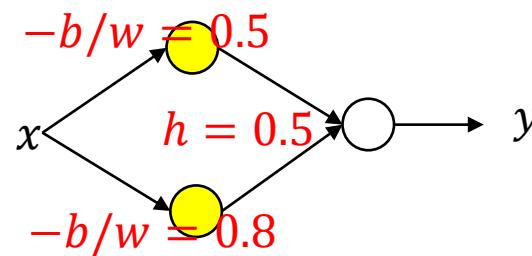
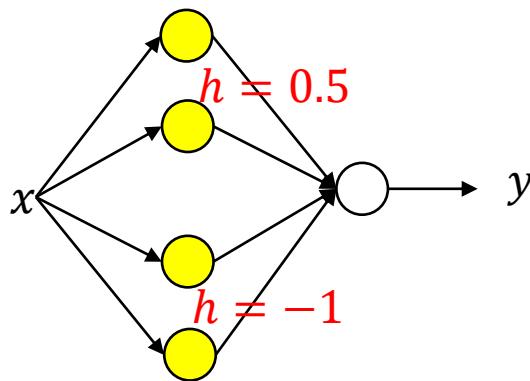
Superposition of two steps



Create a column of height $h=0.5$

NN is a Universal Function Approximator

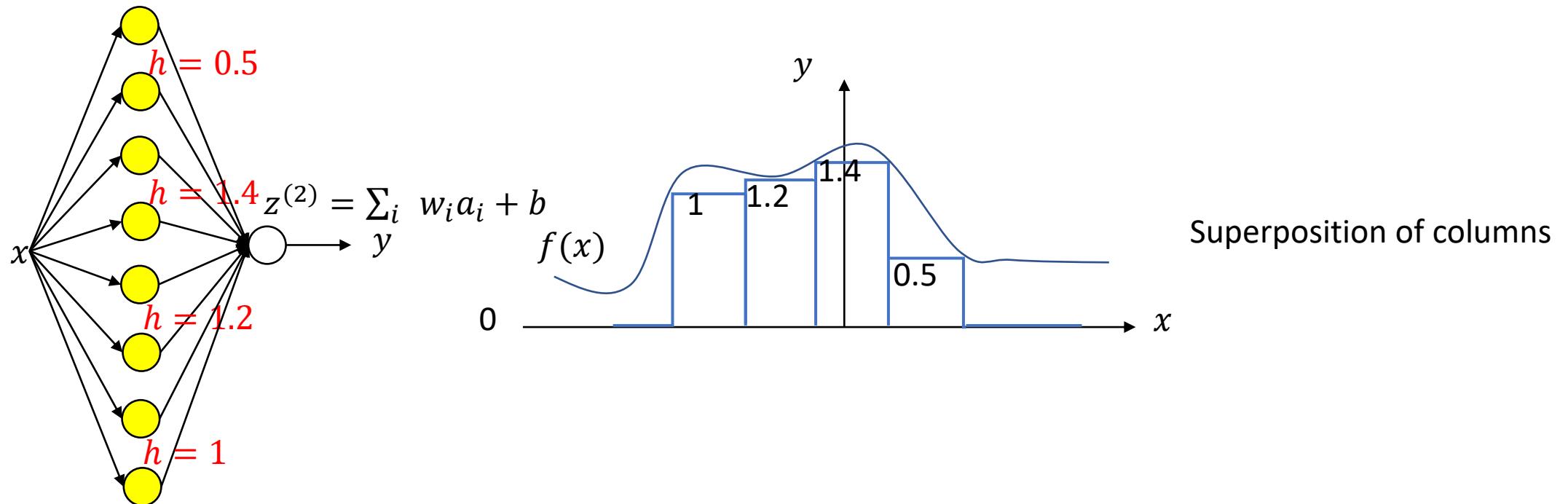
- NN can approximate continuous and smooth functions
A visual proof (for sigmoid activation)



Create a column of height $h=0.5$

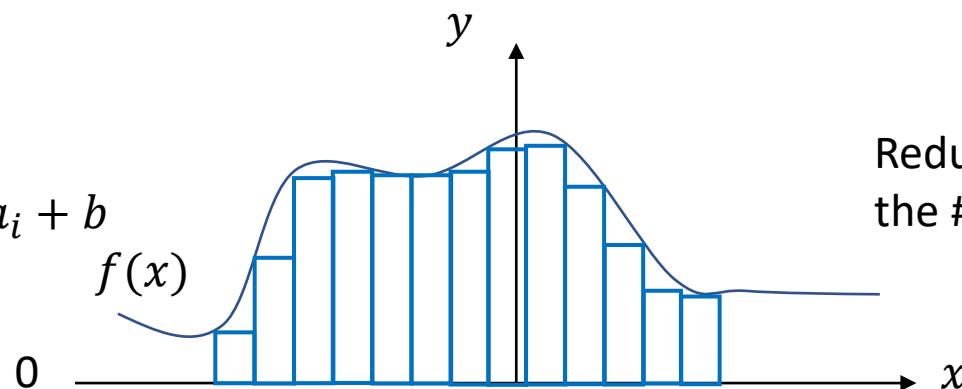
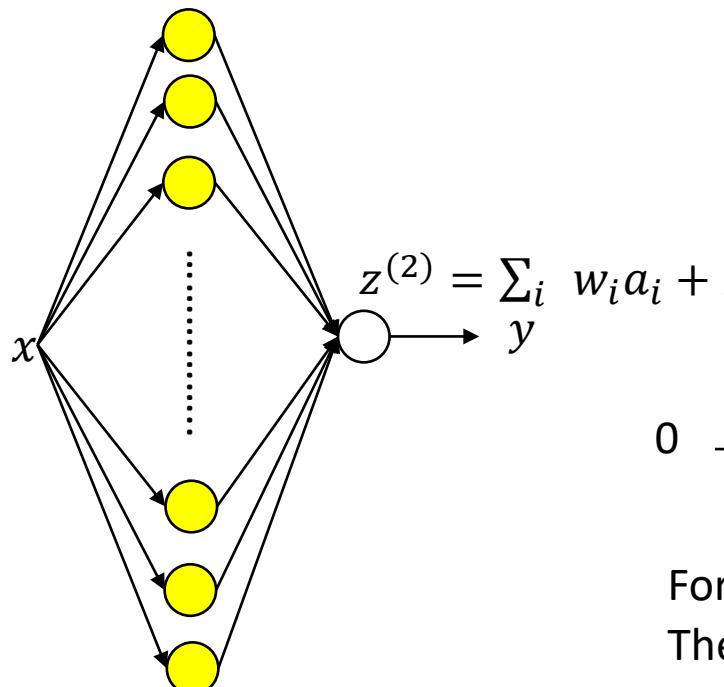
NN is a Universal Function Approximator

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



NN is a Universal Function Approximator

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



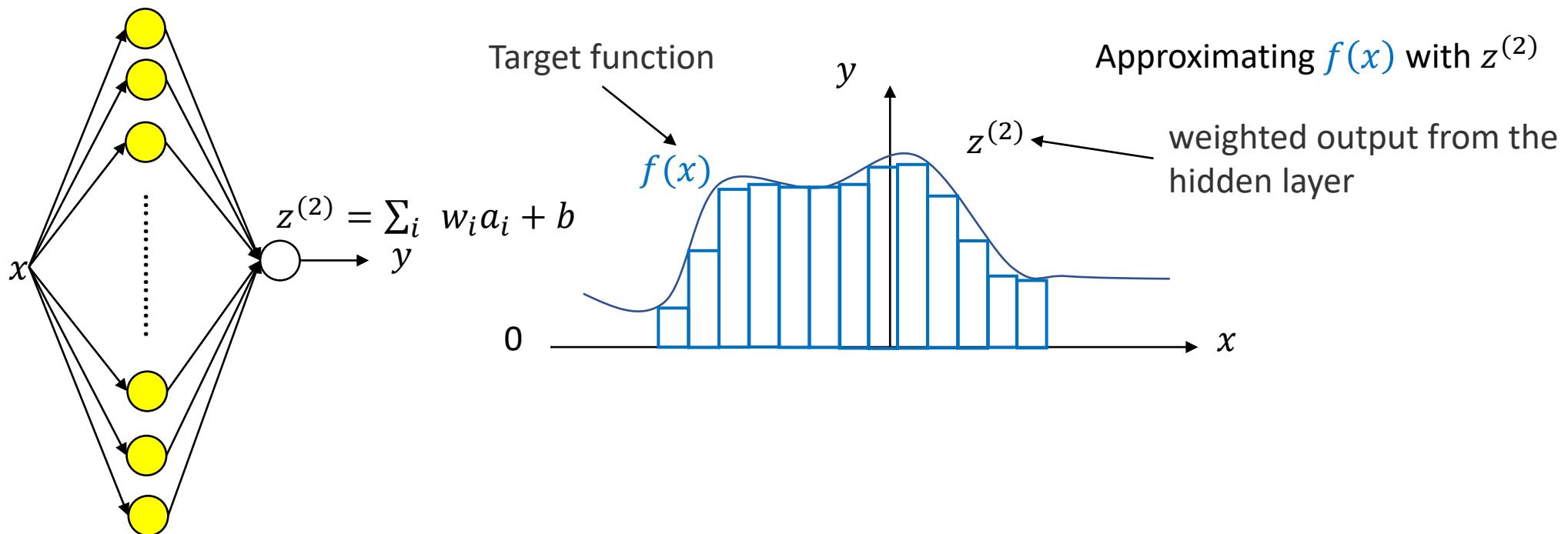
Reduce column widths + increasing
the # of neurons to approximate $f(x)$

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

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

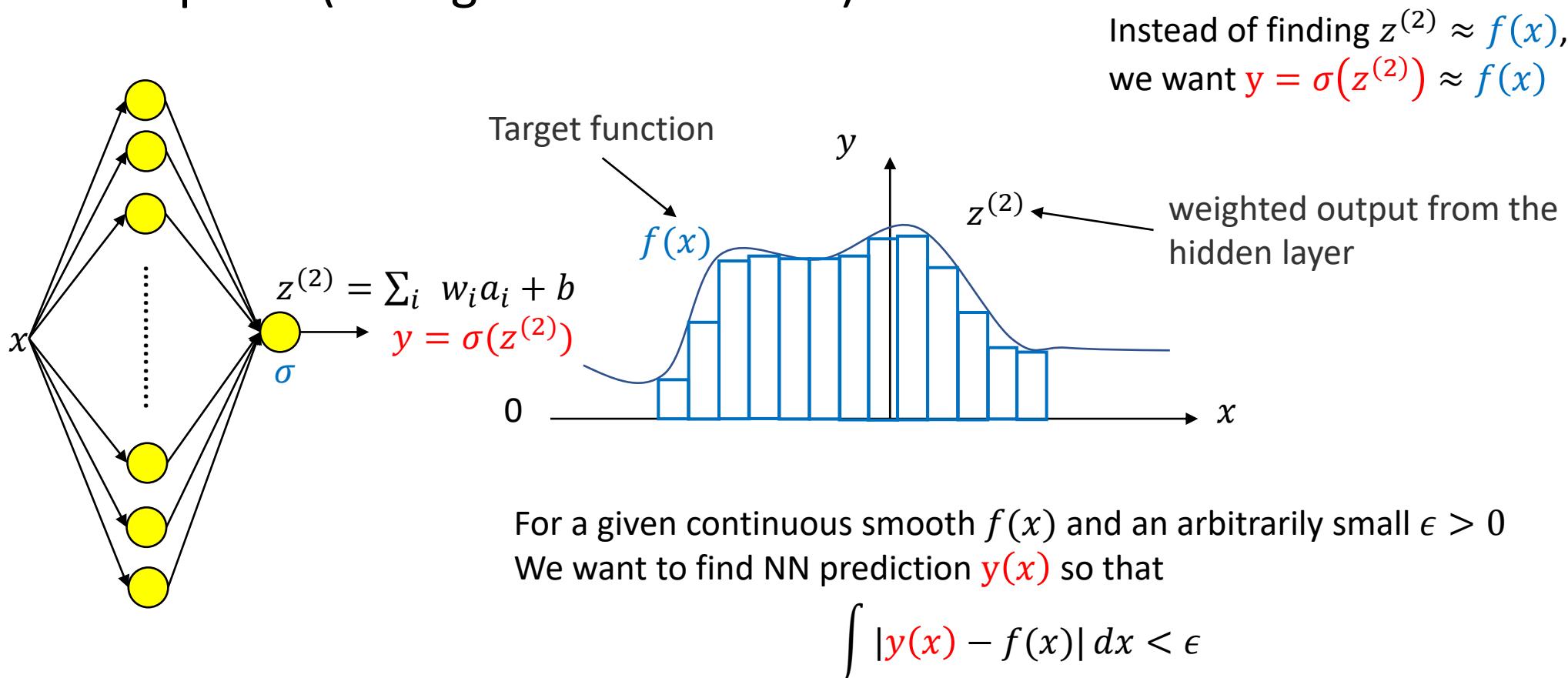
NN is a Universal Function Approximator

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



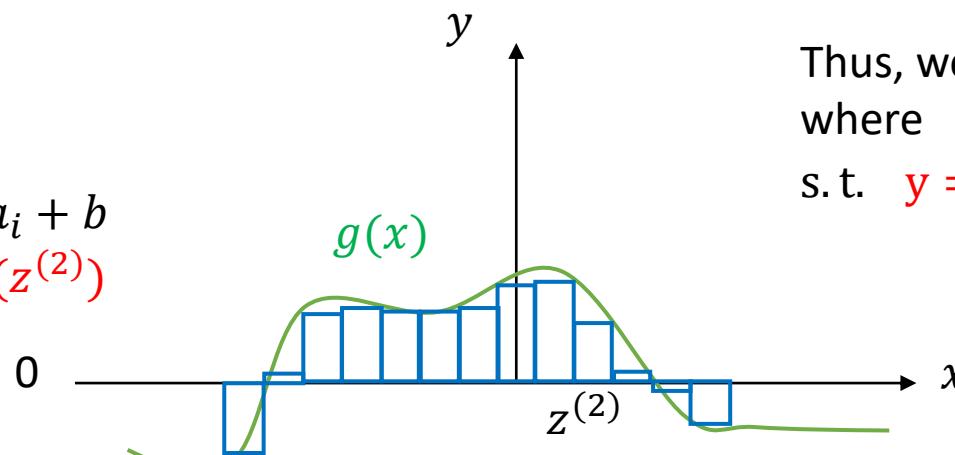
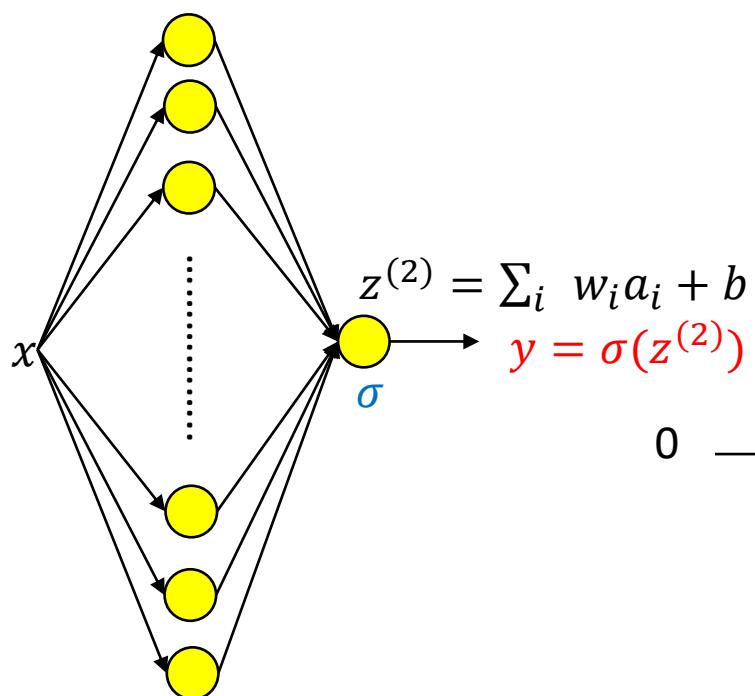
NN is a Universal Function Approximator

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



NN is a Universal Function Approximator

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

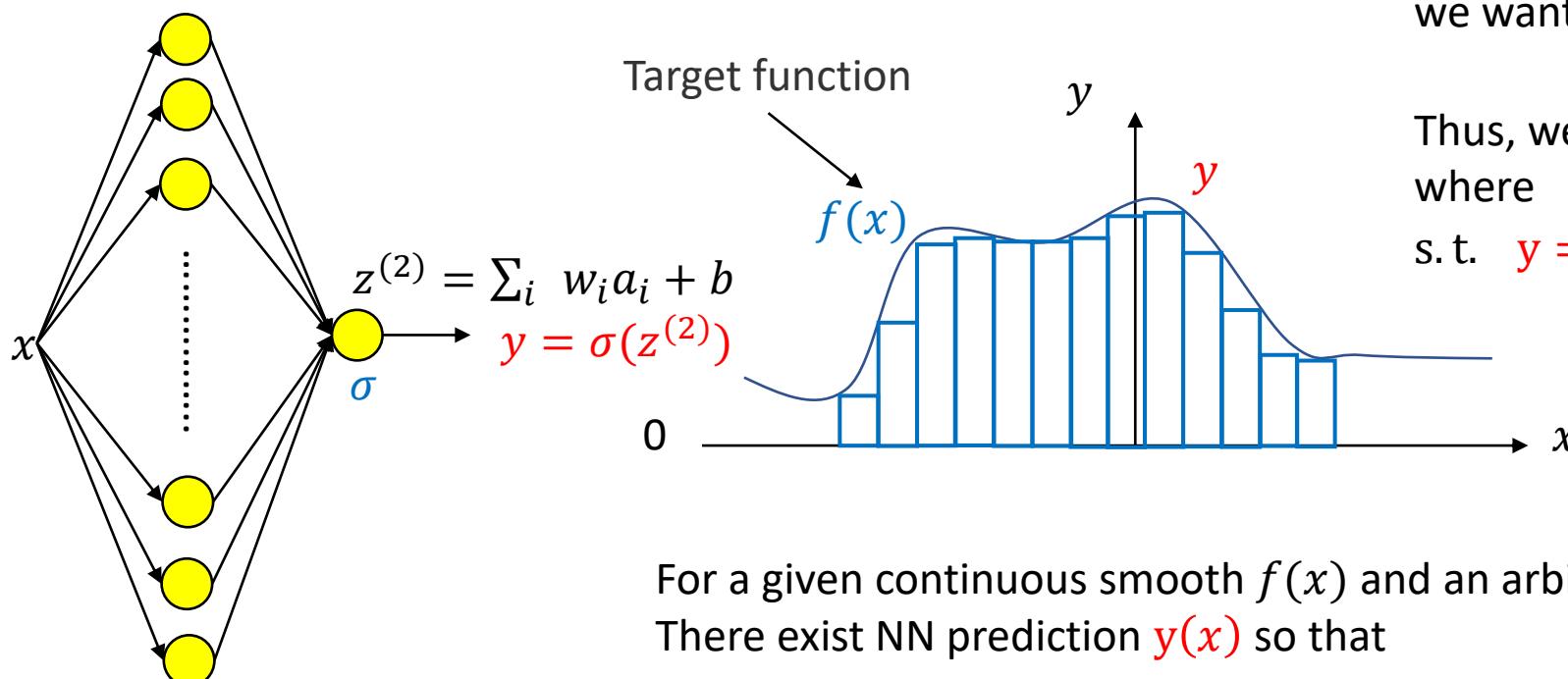


Instead of finding $z^{(2)} \approx f(x)$,
we want $y = \sigma(z^{(2)}) \approx f(x)$

Thus, we want to find $z^{(2)} \approx g(x)$
where $\sigma(g(x)) = f(x)$
s.t. $y = \sigma(z^{(2)}) \approx \sigma(g(x)) = f(x)$

NN is a Universal Function Approximator

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



$$\int |y(x) - f(x)| dx < \epsilon$$

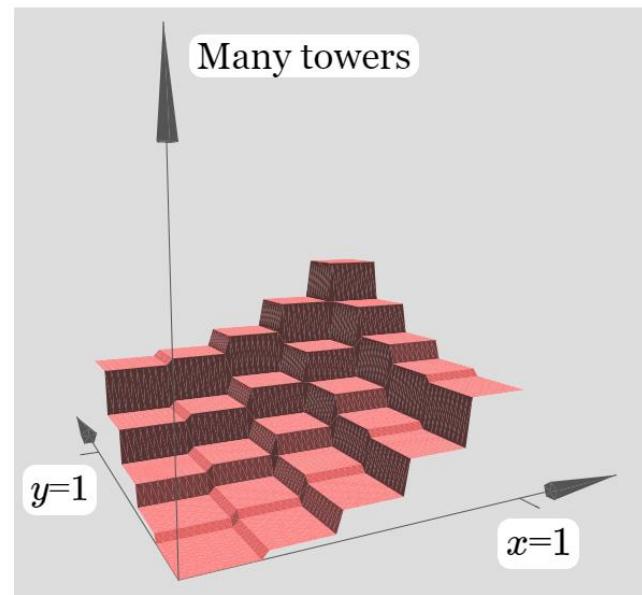
Instead of finding $z^{(2)} \approx f(x)$,
we want $y = \sigma(z^{(2)}) \approx f(x)$

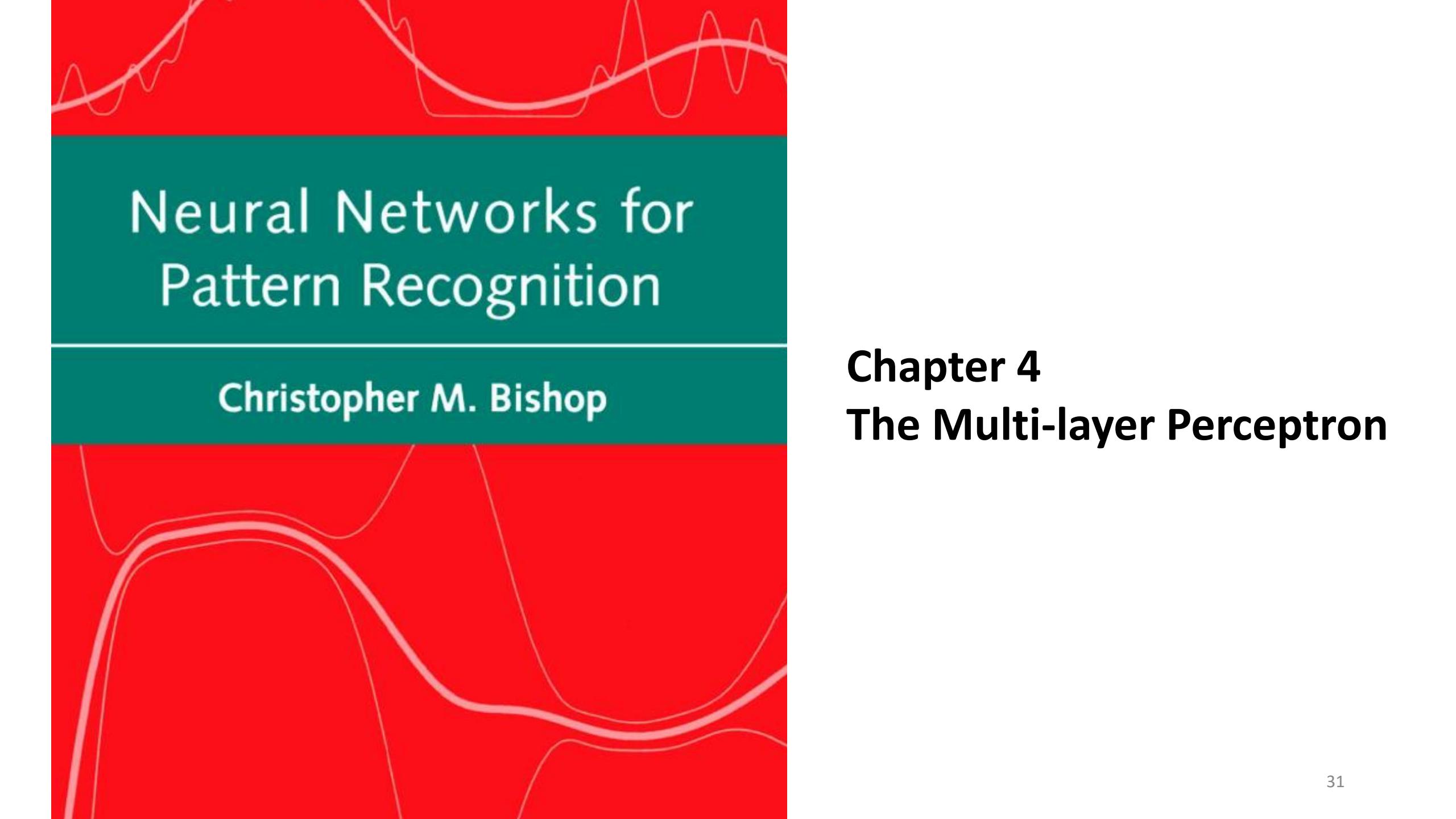
Thus, we want to find $z^{(2)} \approx g(x)$
where $\sigma(g(x)) = f(x)$
s.t. $y = \sigma(z^{(2)}) \approx \sigma(g(x)) = f(x)$

NN is a Universal Function Approximator

- Approximate a 2D surface

What should be the input/output units of NN?





Neural Networks for Pattern Recognition

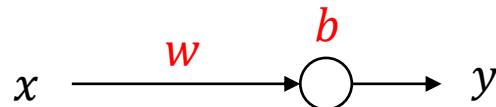
Christopher M. Bishop

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?

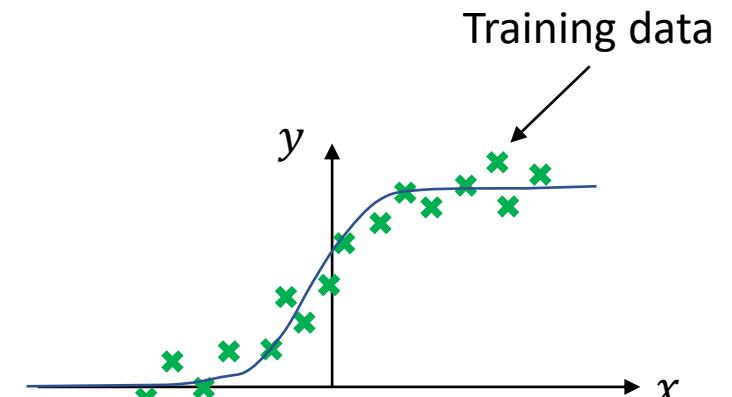
- E.g., Model $y = \sigma(wx + b)$



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

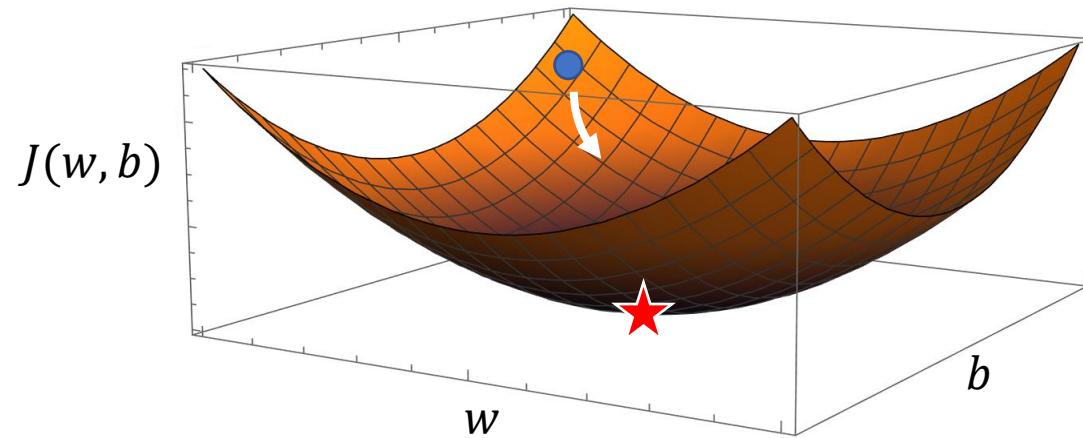
Our goal is to find the model parameters w, b that minimizes the cost function

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



How to find w and b?

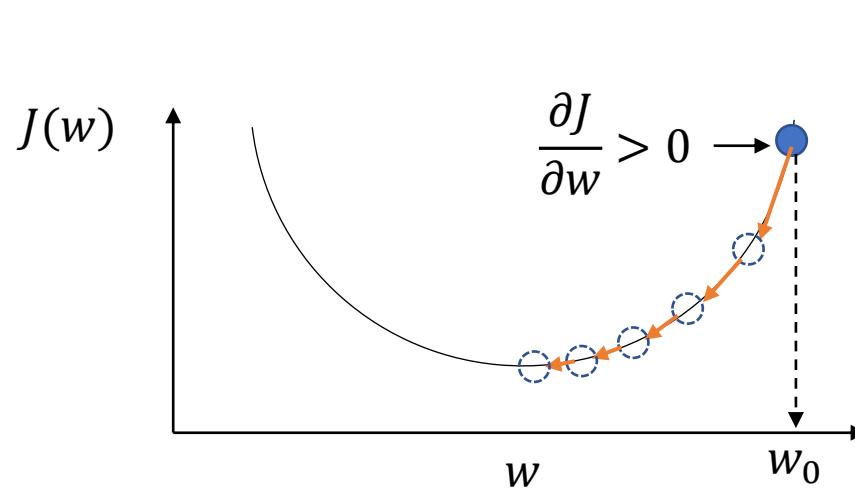
- E.g., Model $y = \sigma(\mathbf{w}x + b)$
- Cost function $J(w, b) = \frac{1}{m} \sum_{i=1}^m (y(x_d^i) - y_d^i)^2$



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 $y = \sigma(\mathbf{w}x + b)$
- Cost function $J(w, b) = \frac{1}{m} \sum_{i=1}^m (y(x_d^i) - y_d^i)^2$

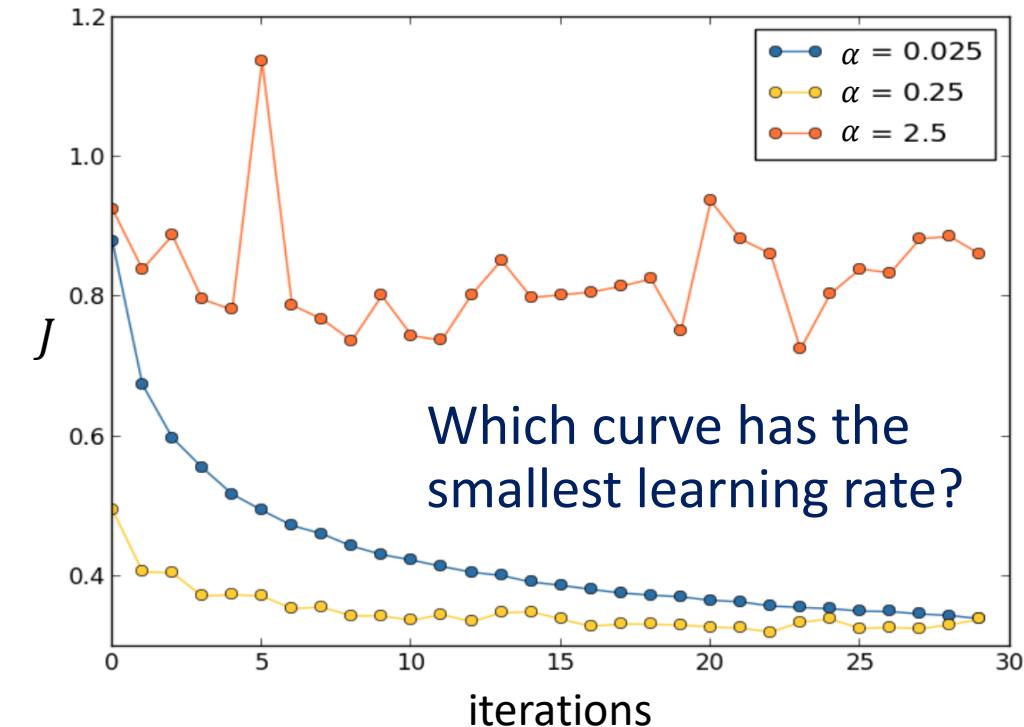
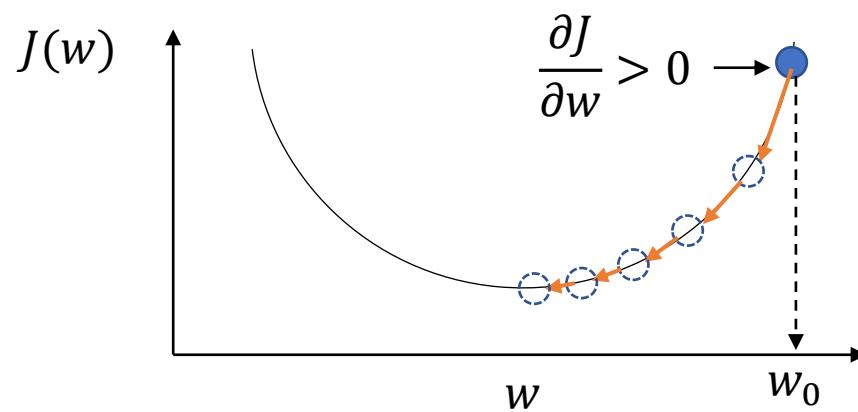


- I. calculate the slope $\frac{\partial J}{\partial w}$ corresponds to an initial w
- II. Adjust w according to the local slope
 $w_{new} = w_{old} - \alpha \frac{\partial J}{\partial w}$, $\alpha > 0$ is the learning rate
- III. Iterate until $\frac{\partial J}{\partial w} = 0$

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

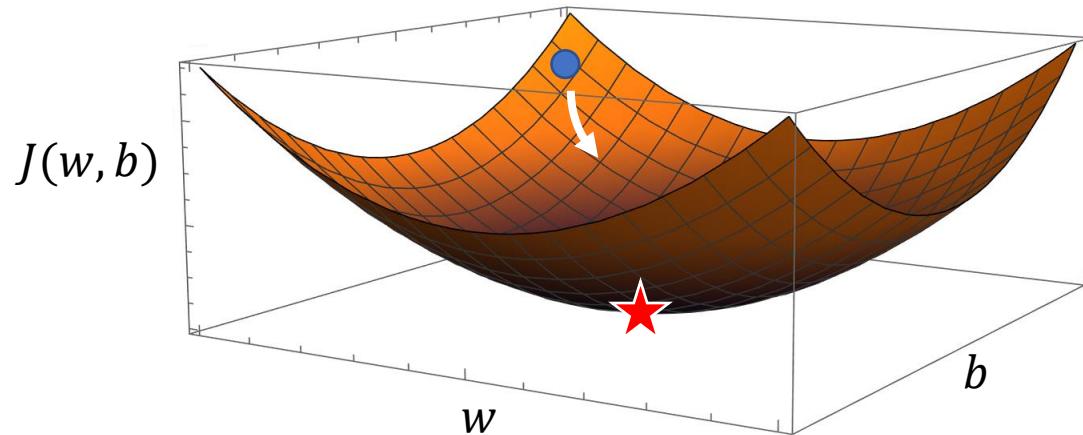
How to find w and b? Gradient descent

- E.g., Model $y = \sigma(\mathbf{w}\mathbf{x} + b)$
- Cost function $J(w, b) = \frac{1}{m} \sum_{i=1}^m (y(\mathbf{x}_d^i) - \mathbf{y}_d^i)^2$



How to find w and b? Gradient descent

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



Gradient on a surface

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

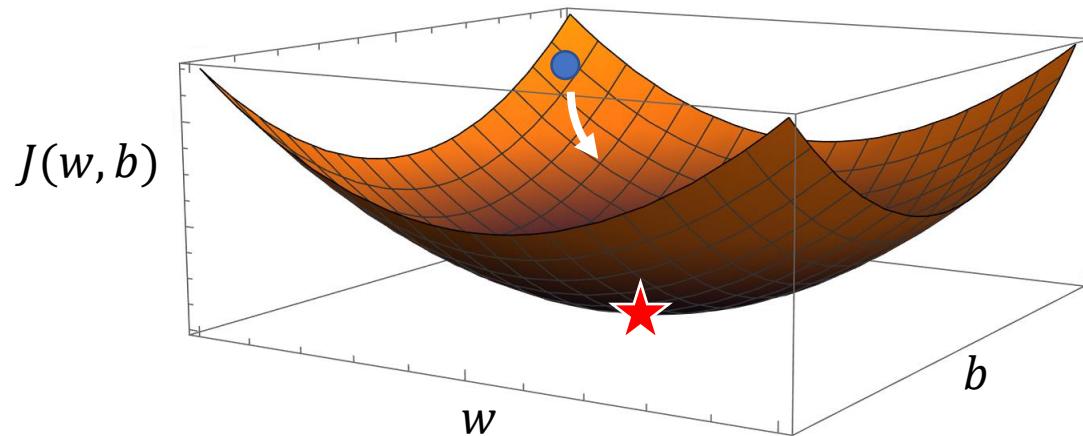
$$-\nabla J(w, b) = -\left(\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}\right)$$

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 $y = \sigma(\mathbf{w}x + b)$
- Cost function $J(w, b) = \frac{1}{m} \sum_{i=1}^m (y(x_d^i) - y_d^i)^2$

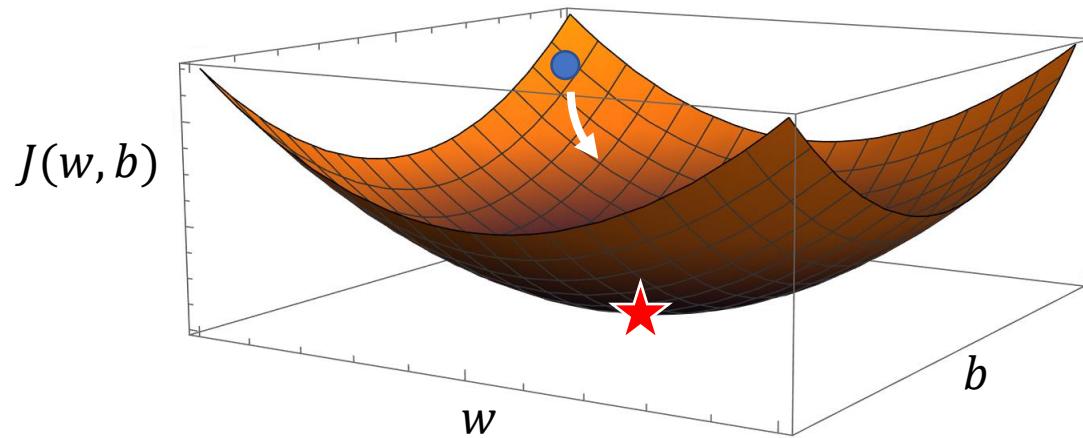


Gradient on a surface

- $-\nabla J$ 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 $\nabla J = 0$

How to find w and b? Gradient descent

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



Gradient on a surface

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

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

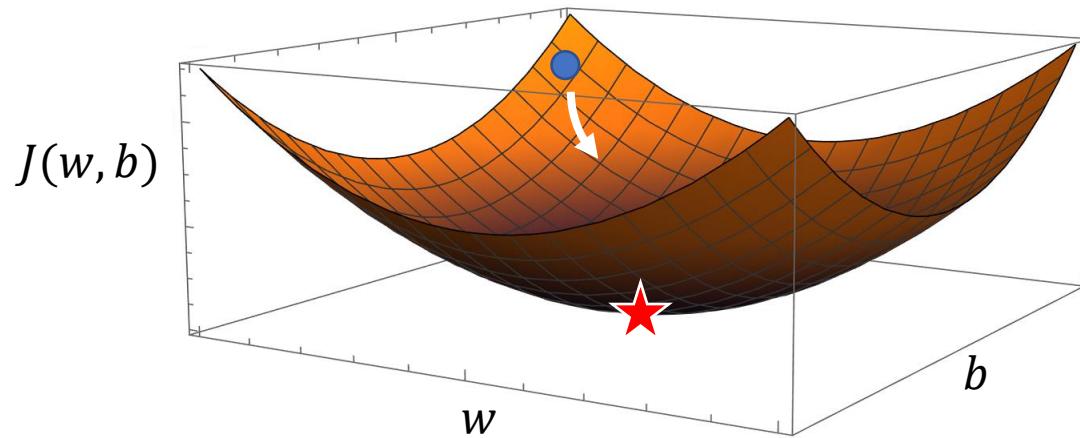
But how are $\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}$ calculated?

How to find w and b? Gradient descent

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

Defined for every example i

- Cost function $J(w, b) = \frac{1}{m} \sum_{i=1}^m (y(x_d^i) - y_d^i)^2 = \frac{1}{m} \sum_{i=1}^m \overbrace{L(y^i, y_d^i)}^{\text{Defined for every example i}}$



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

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

Back propagation (chain rule)

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

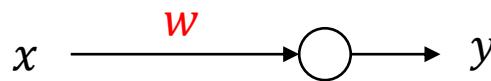
1) For one example:

$$\frac{\partial L}{\partial w}(w, b, x_d^i, y_d^i) = 2(\sigma(wx_d^i + b) - y_d^i) \sigma' x_d^i$$

2) For the full data set:

$$\frac{\partial J}{\partial w} = \frac{1}{m} \sum_{i=1}^m \frac{\partial L}{\partial w} = \frac{1}{m} \sum_{i=1}^m 2(\sigma(wx_d^i + b) - y_d^i) \sigma' x_d^i$$

Example:

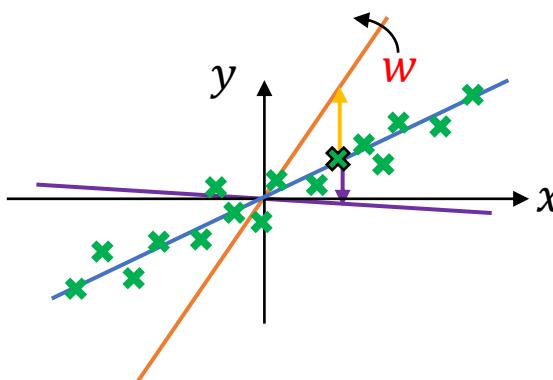
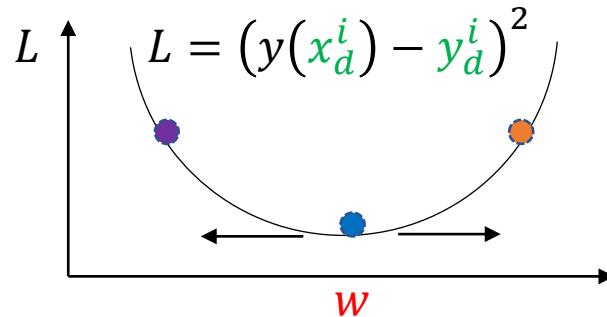


- Linear Regression Model $y = \mathbf{w}x$

Defined for every example i

- Cost function $J(w, b) = \frac{1}{m} \sum_{i=1}^m (y(\mathbf{x}_d^i) - \mathbf{y}_d^i)^2 = \frac{1}{m} \sum_{i=1}^m L(y^i, \mathbf{y}_d^i)$

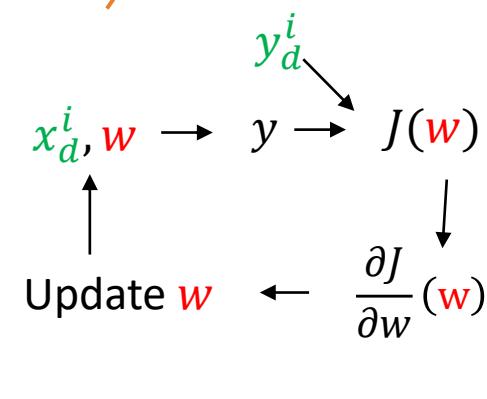
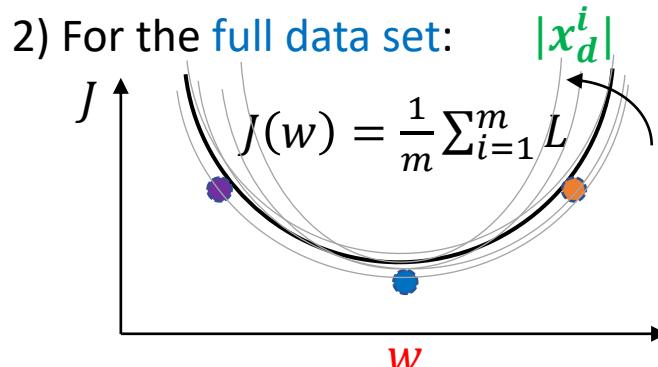
1) For one example:



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

Back propagation (chain rule)

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



1) For one example:

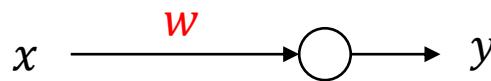
$$\frac{\partial L}{\partial w}(w, b, \mathbf{x}_d^i, \mathbf{y}_d^i) = 2(\mathbf{w}\mathbf{x}_d^i - \mathbf{y}_d^i)\mathbf{x}_d^i$$

2) For the full data set:

$$\frac{\partial J}{\partial w} = \frac{1}{m} \sum_{i=1}^m \frac{\partial L}{\partial w} = \frac{1}{m} \sum_{i=1}^m 2(\mathbf{w}\mathbf{x}_d^i - \mathbf{y}_d^i)\mathbf{x}_d^i$$

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

Example:

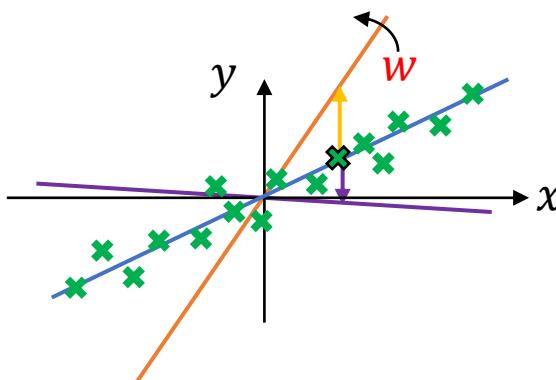
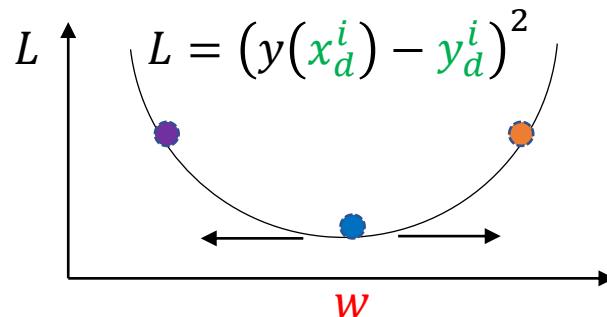


- Linear Regression Model $y = \mathbf{w}x$

Defined for every example i

- Cost function $J(w, b) = \frac{1}{m} \sum_{i=1}^m (y(\mathbf{x}_d^i) - \mathbf{y}_d^i)^2 = \frac{1}{m} \sum_{i=1}^m L(y^i, \mathbf{y}_d^i)$

1) For one example:



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

Back propagation (chain rule)

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

What would happen if \mathbf{x}_d^i are very large/small?

Note that $\frac{\partial J}{\partial w}$ is analytical fn of $w, \mathbf{x}_d^i, \mathbf{y}_d^i$!

1) For one example:

$$\frac{\partial L}{\partial w}(w, b, \mathbf{x}_d^i, \mathbf{y}_d^i) = 2(\mathbf{w}\mathbf{x}_d^i - \mathbf{y}_d^i) \mathbf{x}_d^i$$

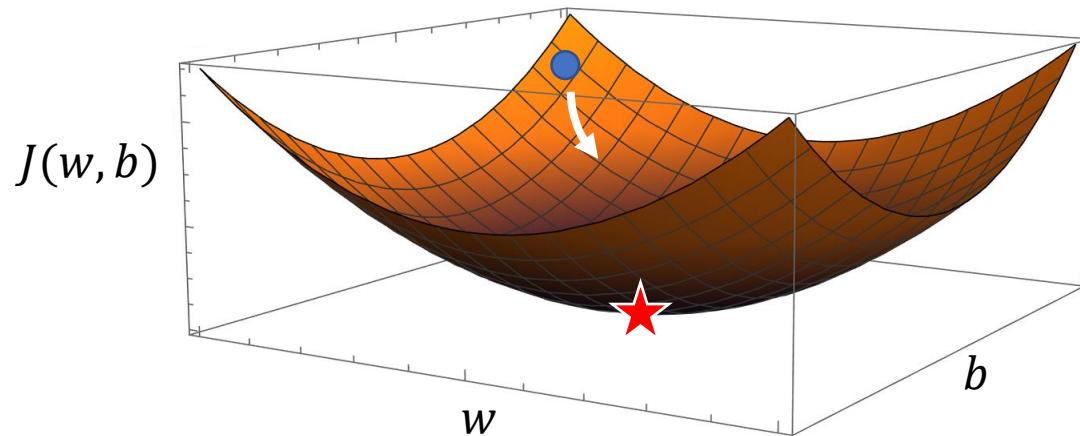
This gradient is defined for every example i, and is a function of w

2) For the full data set:

$$\frac{\partial J}{\partial w} = \frac{1}{m} \sum_{i=1}^m \frac{\partial L}{\partial w} = \frac{1}{m} \sum_{i=1}^m 2(\mathbf{w}\mathbf{x}_d^i - \mathbf{y}_d^i) \mathbf{x}_d^i$$

How to find w and b? Gradient descent

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



- $-\nabla J(w, b) = -\left(\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}\right)$ **for a given data set at a given w, 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 $-\nabla J(w, b) = -\left(\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}\right)$ 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
 $y = \sigma(wx + b)$.

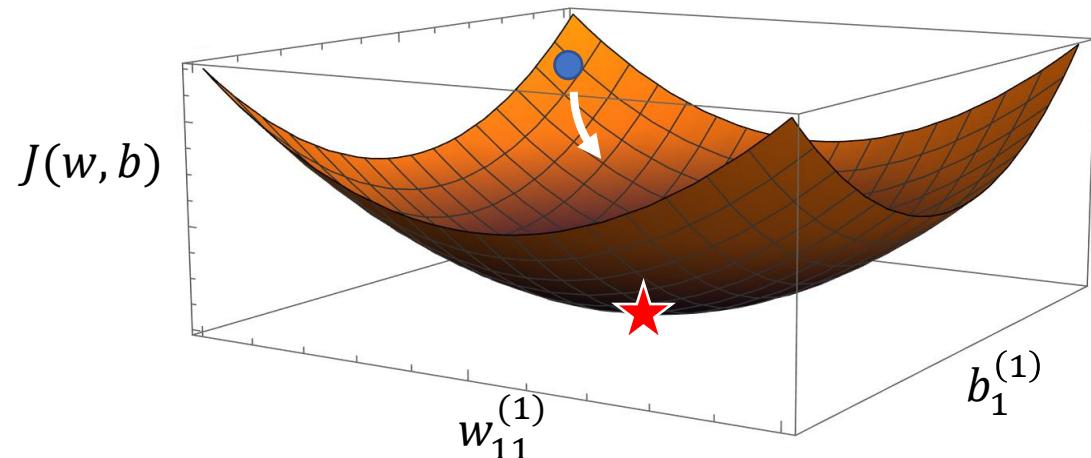
What about finding w and b in a neural network?

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?
 $y = a^{(2)} = \sigma(\mathbf{W}^{(2)}\sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)})$

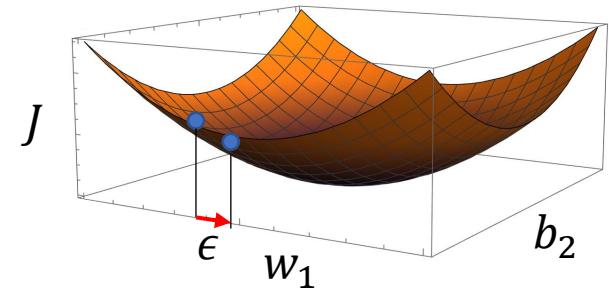
How to find w and b in a neural network?

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



- For the full data set ($i=1\dots m$), compute
$$-\nabla J = -\left(\frac{\partial J}{\partial w_{jk}^{(l)}}, \dots, \frac{\partial J}{\partial b_j^{(l)}}\right)$$
- $w_{jk}^{(l) new} = w_{jk}^{(l) old} - \alpha \frac{\partial J}{\partial w_{jk}^{(l)}}, \quad b_j^{(l) new} = b_j^{(l) old} - \alpha \frac{\partial J}{\partial b_j^{(l)}}$
 α is the learning rate
- Iterate until $\nabla J = 0$

How are $\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial b_1} \dots$ calculated?

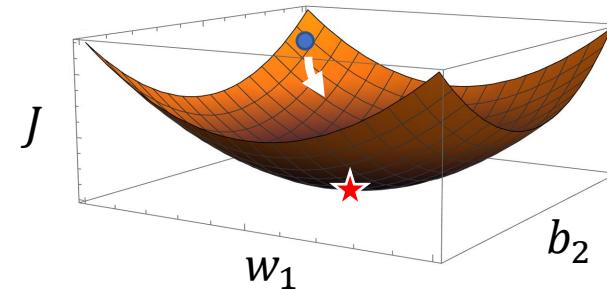


- i. Slow: Calculate J at two adjacent locations, evaluate their differences
$$\frac{\partial J}{\partial w_j} = \frac{J(w_0 + \epsilon e_j) - J(w_0)}{\epsilon}$$
For 1M weights, have to do 1M forward passes
- ii. Fast: Back propagation
Compute derivatives w.r.t. all weights analytically with chain rules

Forward & Back Propagation: 1. single neuron

Exercise:

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

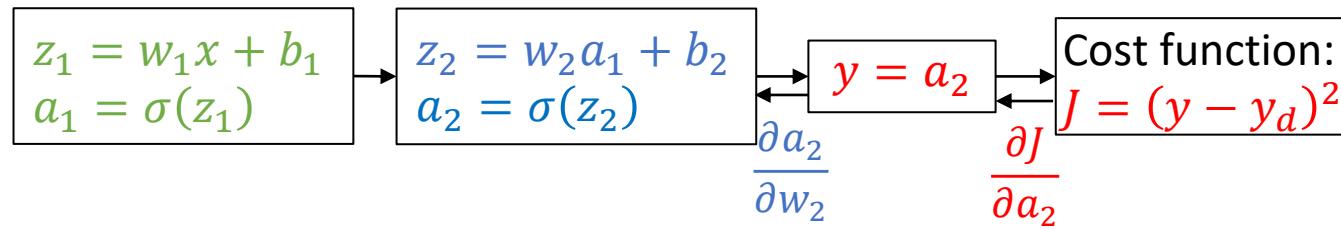


∇J vary with
 w_1, w_2, b_1, b_2



How important is w_2 for changing the cost function J ?

Forward propagation →



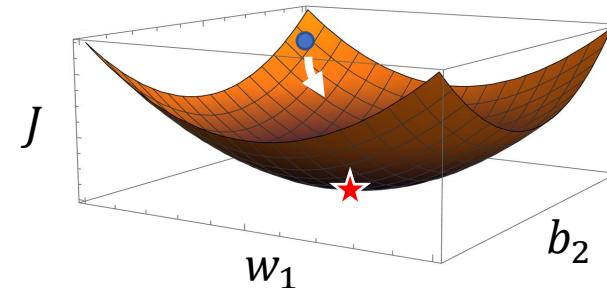
← Back propagation

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

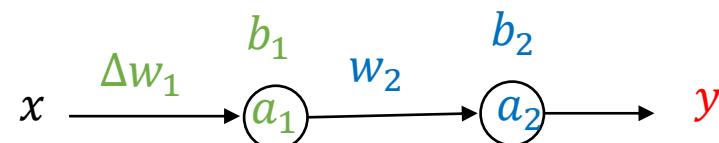
Forward & Back Propagation: 1. single neuron

Exercise:

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

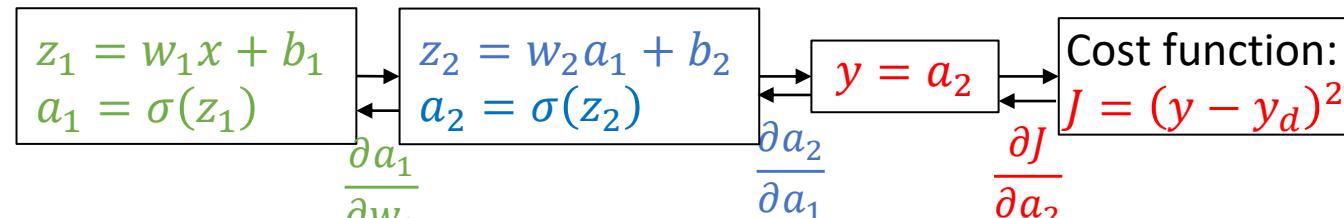


∇J vary with
 w_1, w_2, b_1, b_2



How important is w_1 for changing the cost function J ?

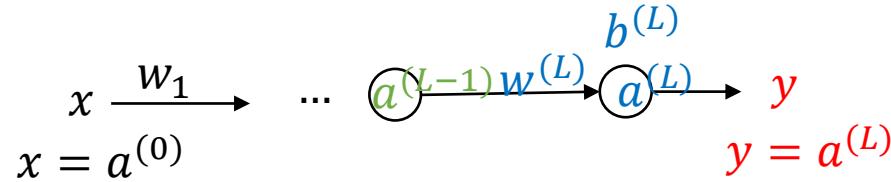
Forward propagation →



← Back propagation

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

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)

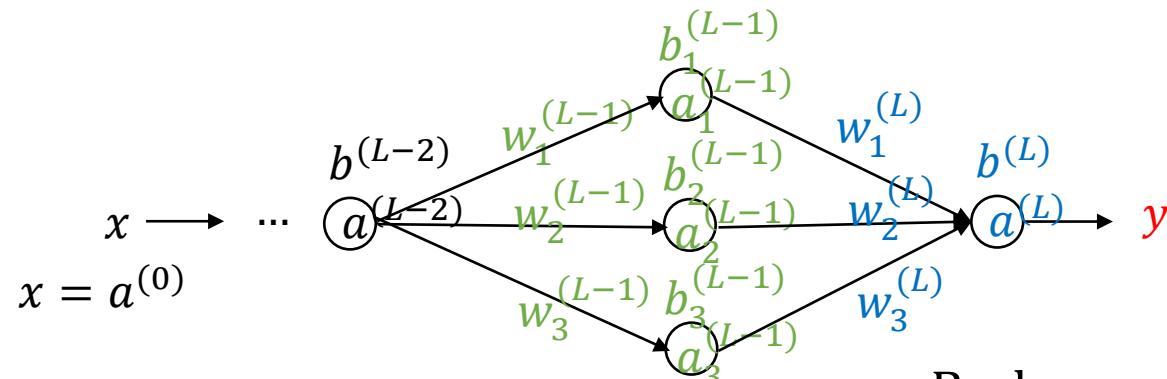
$$\frac{\partial J}{\partial w^{(L)}} =$$

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

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

$$\begin{aligned} \frac{\partial J}{\partial b^{(L-1)}} &= \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial b^{(L-1)}} \\ &= 2(a^{(L)} - y_d) \sigma'(z^{(L)}) w^{(L)} \sigma'(z^{(L-1)}) \end{aligned}$$

Forward & Back Propagation: 2. multiple hidden units



Back propagation (use chain rules)

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 $J = (y - y_d)^2$

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

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

$$\begin{aligned} \frac{\partial J}{\partial w_k^{(L-1)}} &= \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_k^{(L-1)}}{\partial w_k^{(L-1)}} \\ &= 2(a^{(L)} - y_d) \sigma'(z^{(L)}) w_k^{(L)} \sigma'(z_k^{(L-1)}) a^{(L-2)} \end{aligned}$$

$$\begin{aligned} \frac{\partial J}{\partial b_k^{(L-1)}} &= \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_k^{(L-1)}}{\partial b_k^{(L-1)}} \\ &= 2(a^{(L)} - y_d) \sigma'(z^{(L)}) w_k^{(L)} \sigma'(z_k^{(L-1)}) \end{aligned}$$

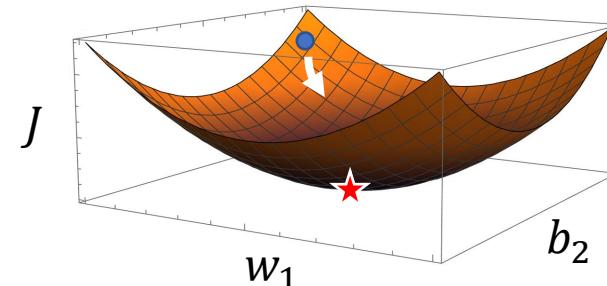
Forward & Back Propagation: 1. single neuron

Exercise:

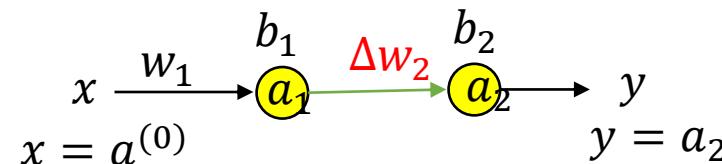
Model: $y = \sigma(w_2\sigma(w_1x + b_1) + b_2)$

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

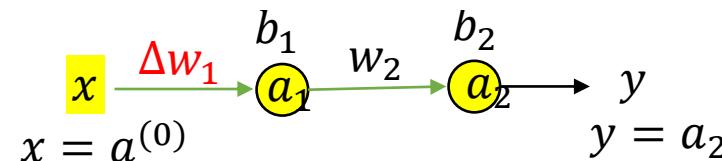
$$\frac{\partial J}{\partial w_2} = 2(y - y_d)\sigma'(z_2)a_1$$



∇J vary with
 w_1, w_2, b_1, b_2



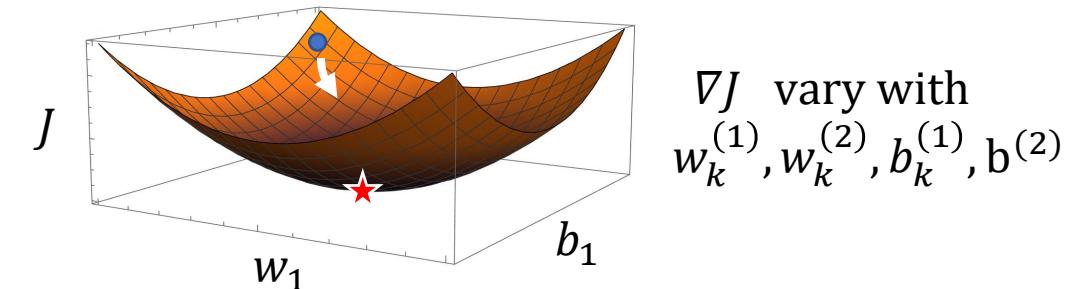
$$\frac{\partial J}{\partial w_1} = 2(y - y_d)\sigma'(z_2)\sigma'(z_1)x$$



Forward & Back Propagation: 2. multiple hidden units

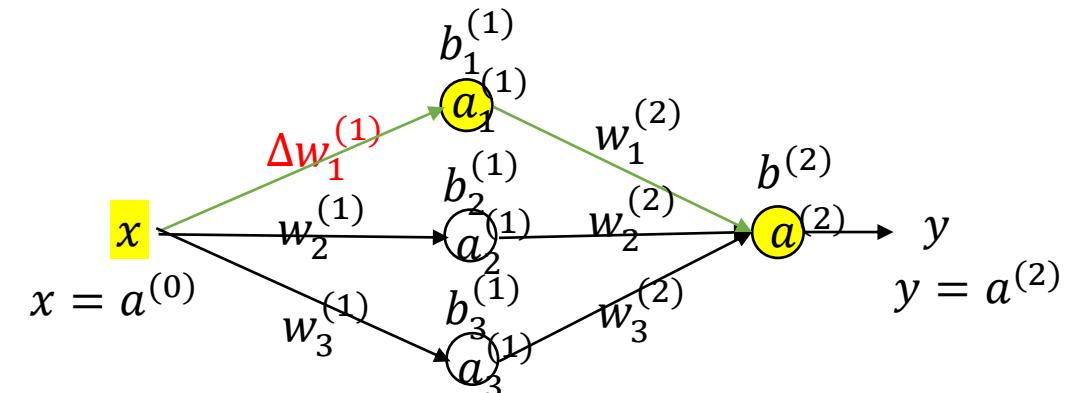
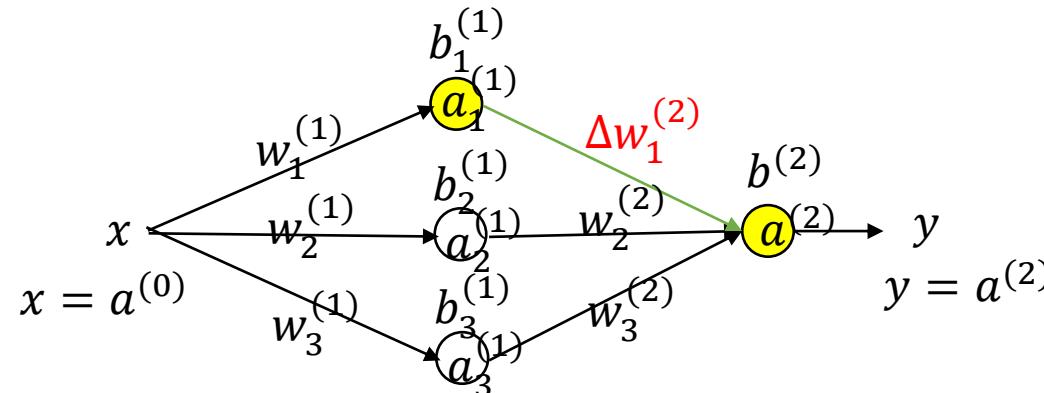
Exercise:

$$\text{Model: } y = \sigma \left(\sum_{k=1}^3 w_k^{(2)} \sigma \left(w_k^{(1)} x + b_k^{(1)} \right) + b^{(2)} \right)$$



$$\begin{aligned} \frac{\partial J}{\partial w_1^{(2)}} &= 2(a^{(2)} - y_d) \sigma'(z^{(2)}) a_1^{(1)} \\ &= \text{fn}(x, y_d, w_k^{(1)}, w_k^{(2)}, b_k^{(1)}, b^{(2)}) \end{aligned}$$

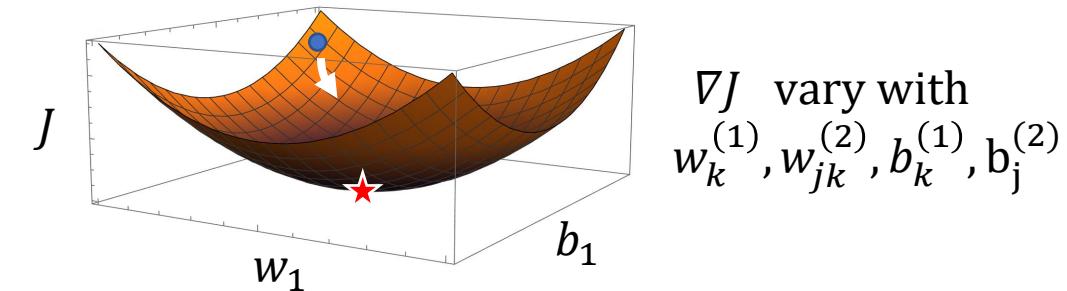
$$\begin{aligned} \frac{\partial J}{\partial w_1^{(1)}} &= 2(a^{(2)} - y_d) \sigma'(z^{(2)}) w_1^{(2)} \sigma'(z_1^{(1)}) x \\ &= \text{fn}(x, y_d, w_k^{(1)}, w_k^{(2)}, b_k^{(1)}, b^{(2)}) \end{aligned}$$



Forward & Back Propagation: 3. multiple outputs

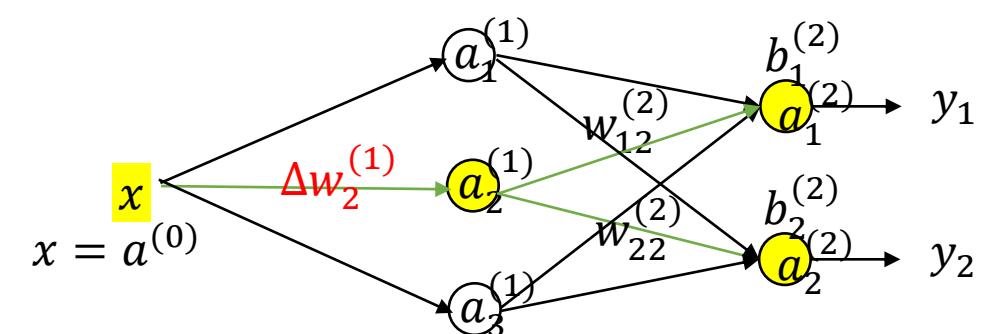
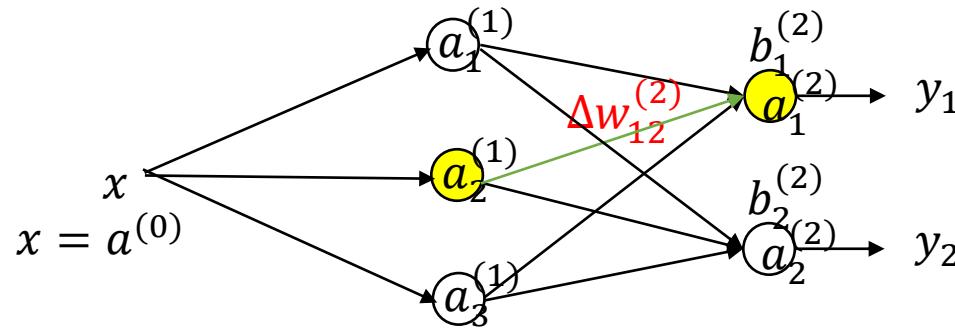
Exercise:

$$\text{Model: } y_j = \sigma \left(\sum_{k=1}^3 w_{jk}^{(2)} \sigma \left(w_k^{(1)} x + b_k^{(1)} \right) + b_j^{(2)} \right)$$

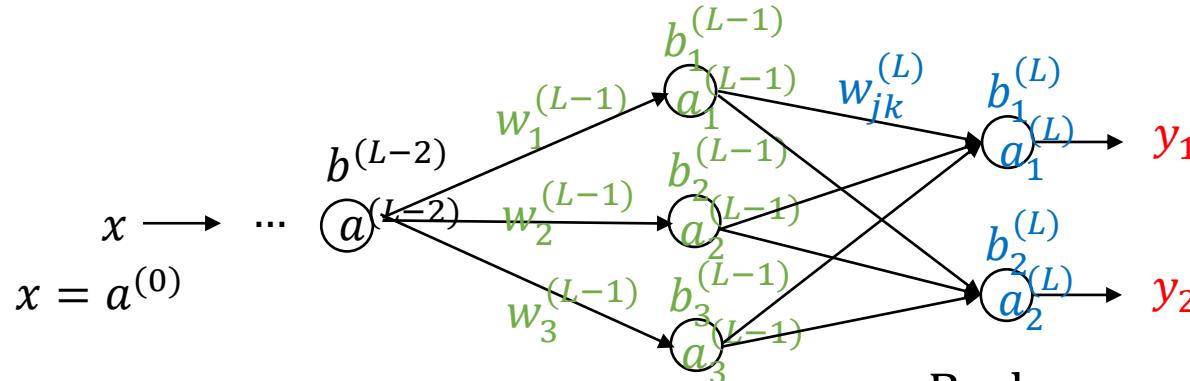


$$\begin{aligned} \frac{\partial J}{\partial w_{12}^{(2)}} &= 2 \left(a_1^{(2)} - y_{d,1} \right) \sigma'(z_1^{(2)}) a_2^{(1)} \\ &= \text{fn}(x, y_{d,1}, w_k^{(1)}, w_{1k}^{(2)}, b_k^{(1)}, b_1^{(2)}) \end{aligned}$$

$$\begin{aligned} \frac{\partial J}{\partial w_2^{(1)}} &= \sum_{j=1}^2 2(a_j^{(2)} - y_{d,j}) \sigma'(z_j^{(2)}) w_{j2}^{(2)} \sigma'(z_2^{(1)}) x \\ &= \text{fn}(x, y_{d,j}, w_k^{(1)}, w_{jk}^{(2)}, b_k^{(1)}, b_j^{(2)}) \end{aligned}$$



Forward & Back Propagation: 3. multiple outputs



Back propagation (use chain rules)

Forward propagation

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

...

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

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

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

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

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

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

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

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

$$\begin{aligned} \frac{\partial J}{\partial w_k^{(L-1)}} &= \sum_{j=1}^2 \frac{\partial J}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_k^{(L-1)}}{\partial w_k^{(L-1)}} \\ &= \sum_{j=1}^2 2(a_j^{(L)} - y_{d,j}) \sigma'(z_j^{(L)}) w_{jk}^{(L)} \sigma'(z_k^{(L-1)}) a_k^{(L-2)} \end{aligned}$$

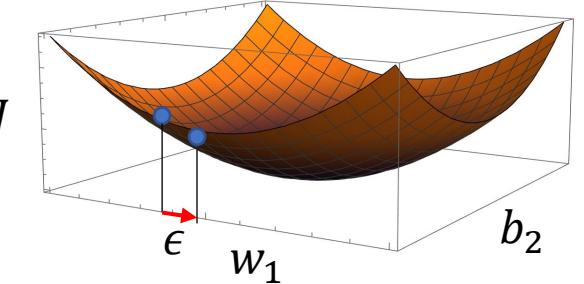
$$\begin{aligned} \frac{\partial J}{\partial b_k^{(L-1)}} &= \sum_{j=1}^2 \frac{\partial J}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_k^{(L-1)}}{\partial b_k^{(L-1)}} \\ &= \sum_{j=1}^2 2(a_j^{(L)} - y_{d,j}) \sigma'(z_j^{(L)}) w_{jk}^{(L)} \sigma'(z_k^{(L-1)}) \end{aligned}$$

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

It is really nice that J is differentiable!



i. Slow: Calculate J at two adjacent locations, evaluate their differences

$$\frac{\partial J}{\partial w_j} = \frac{J(w_0 + \epsilon e_j) - J(w_0)}{\epsilon} \quad \text{For 1M weights, have to do 1M forward passes}$$

ii. Fast: Back propagation

Enabling us to simultaneously compute all the partial derivatives $\frac{\partial J}{\partial w_j}$ using just one forward pass through the network, followed by one backward pass through the network.

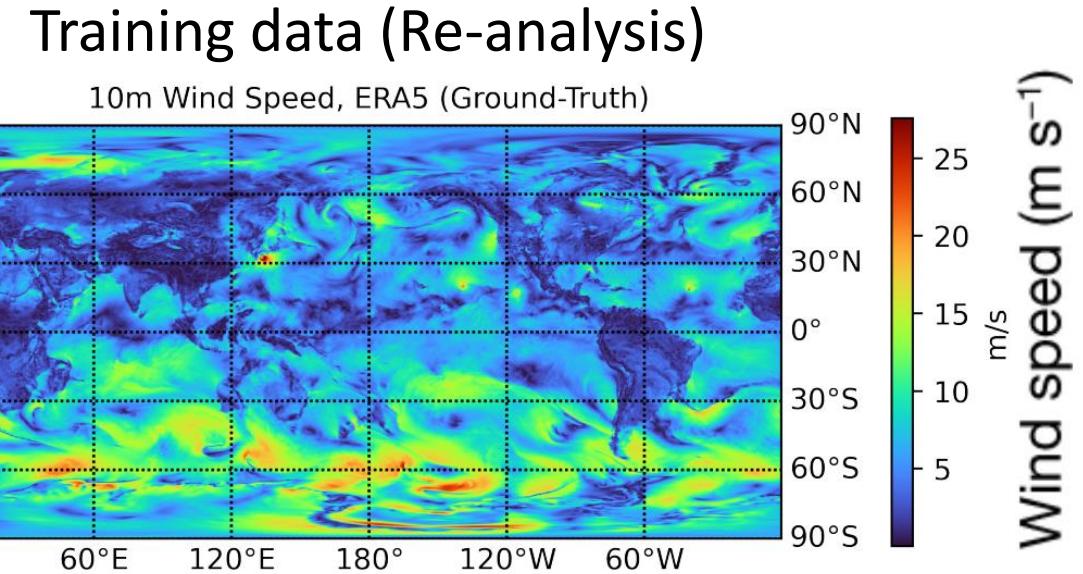
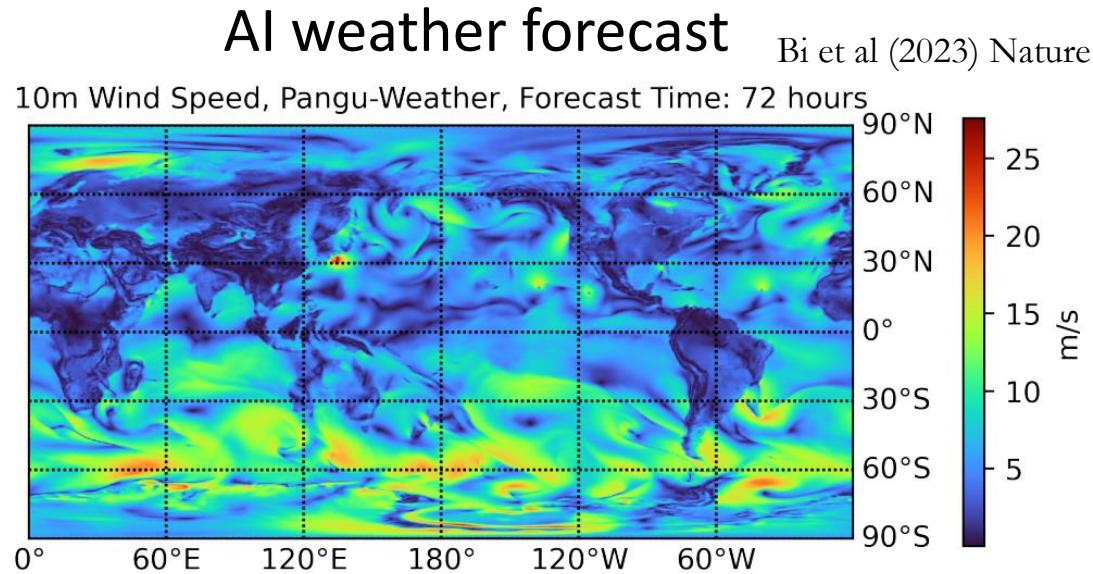
Summary

- Universal function approximator
- Gradient descent: a method to find weights and biases that minimize J
- Calculate ∇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 in scientific problems
- Coding exercise

Some challenges:



- Spectral bias of NN:

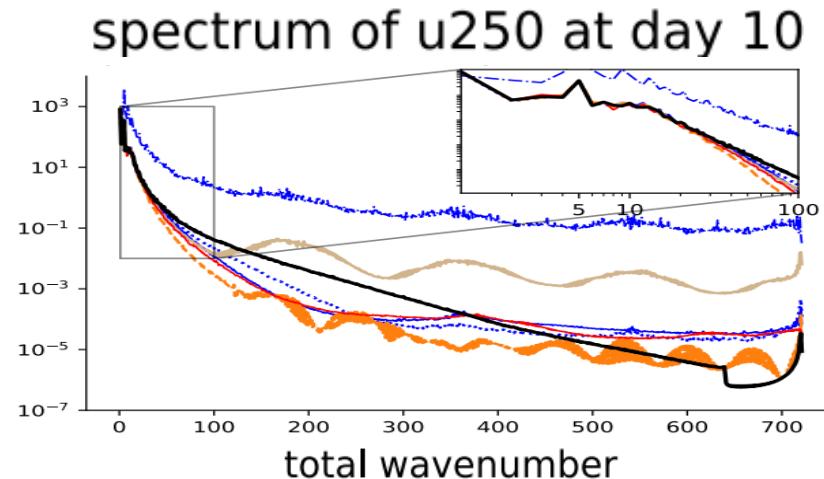
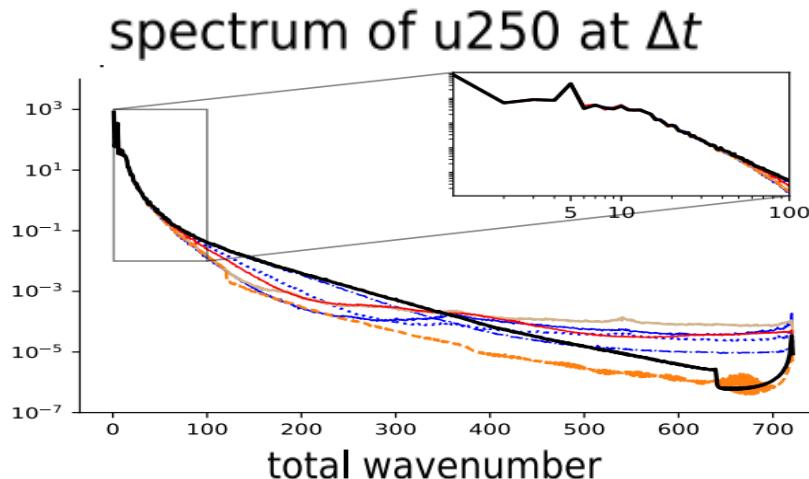
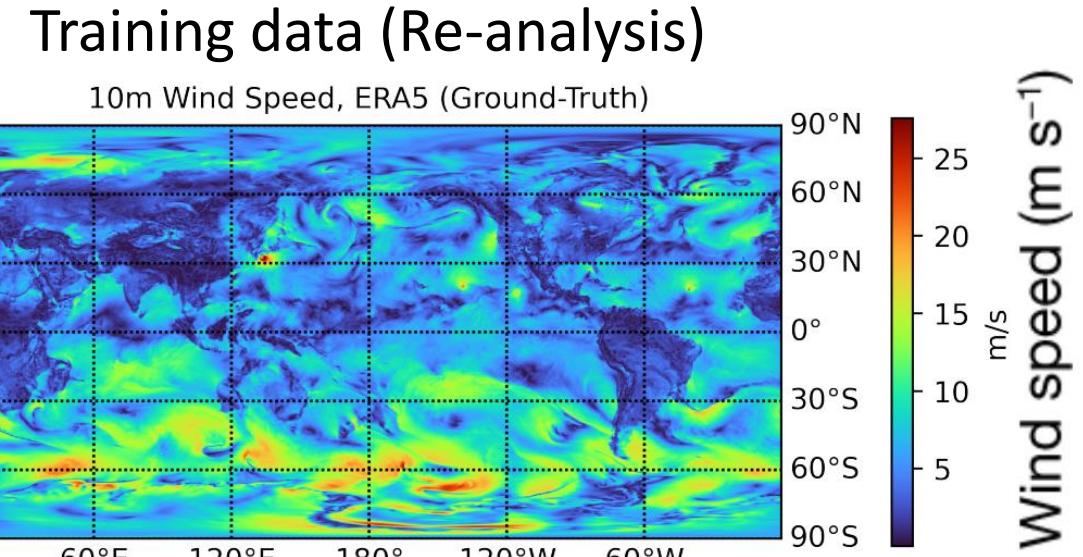
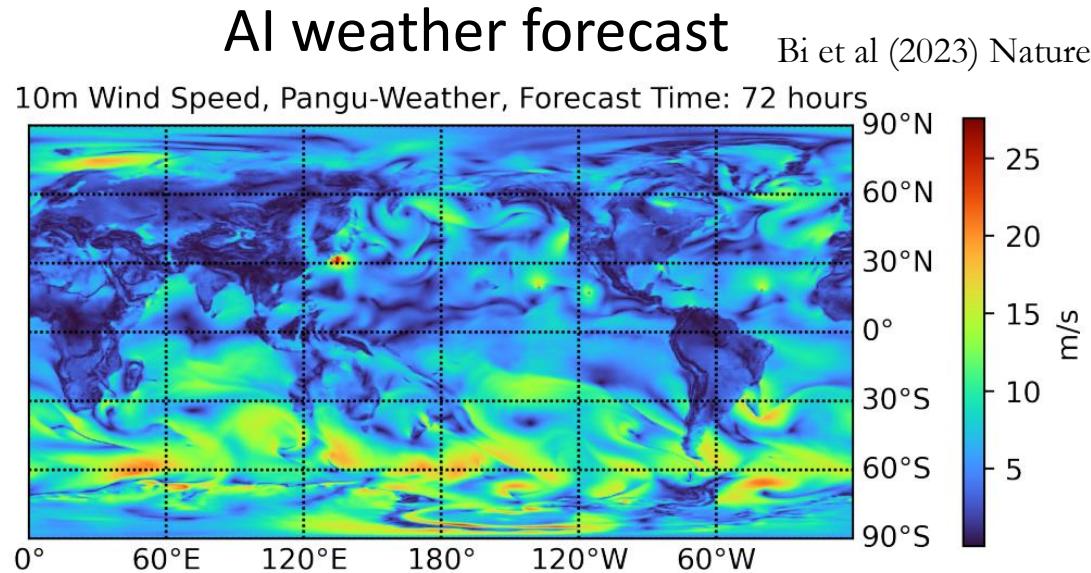
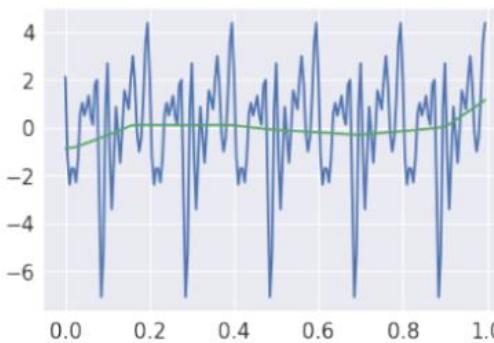


Image source: Qiang Sun & Pedram Hassanzadeh 66

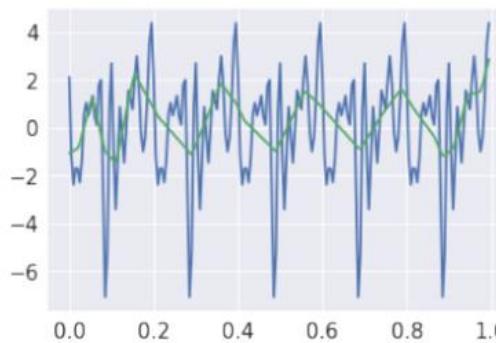
Spectral bias is bad for multiscale problems



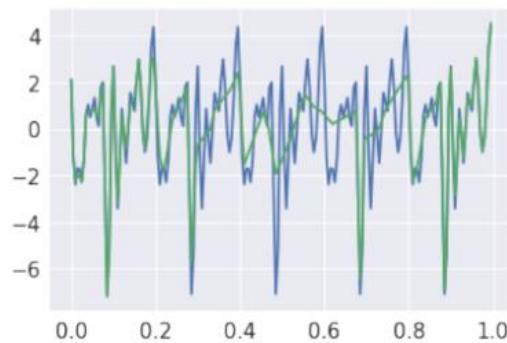
- Spectral bias of NN: Rahaman et al (2019), Frequency Principle: Xu et al (2020)



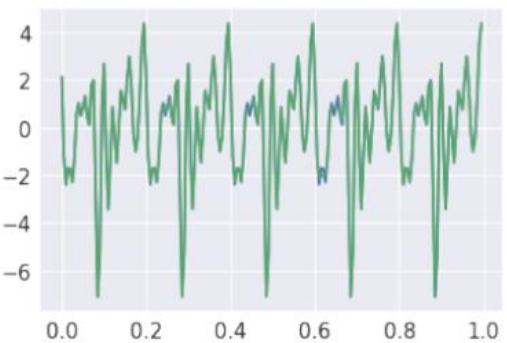
(a) Iteration 100



(b) Iteration 1000



(c) Iteration 10000

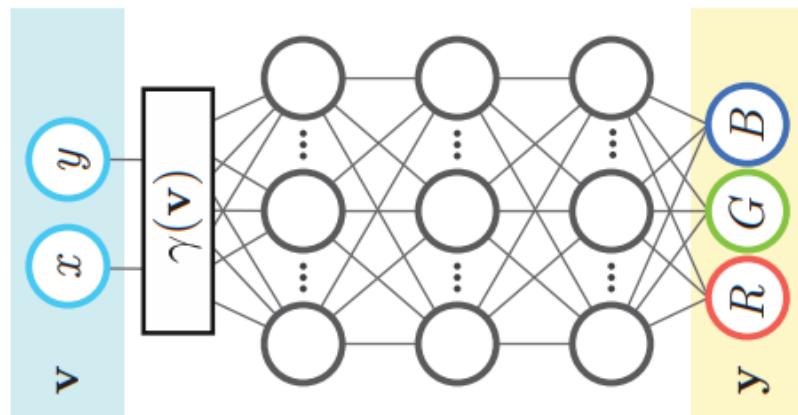


(d) Iteration 80000

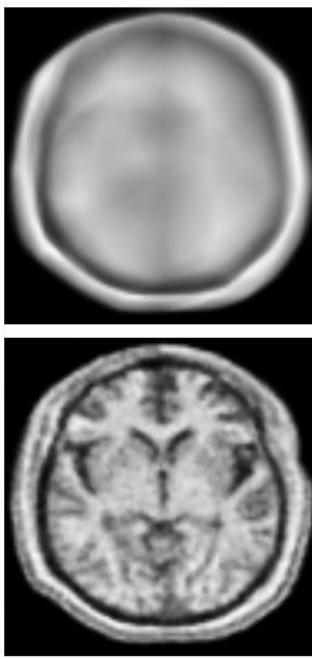
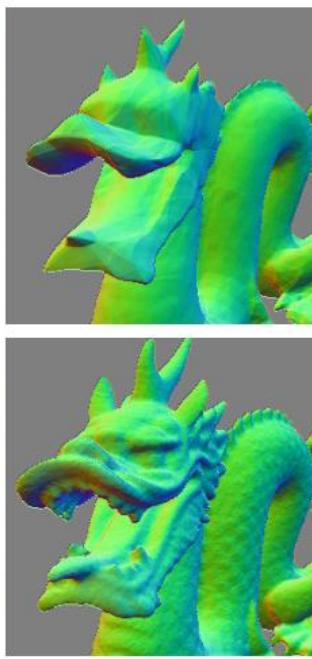
Spectral bias is bad for multiscale problems

- 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.

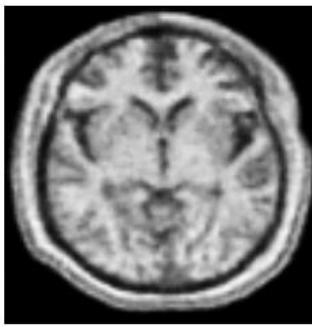
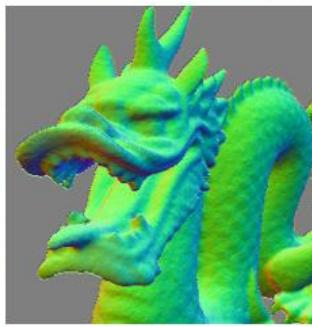
$$\gamma(\mathbf{v}) = [a_1 \cos(2\pi \mathbf{b}_1^T \mathbf{v}), a_1 \sin(2\pi \mathbf{b}_1^T \mathbf{v}), \dots, a_m \cos(2\pi \mathbf{b}_m^T \mathbf{v}), a_m \sin(2\pi \mathbf{b}_m^T \mathbf{v})]^T$$



No Fourier features
 $\gamma(\mathbf{v}) = \mathbf{v}$



With Fourier features
 $\gamma(\mathbf{v}) = \text{FF}(\mathbf{v})$



(b) Image regression
 $(x, y) \rightarrow \text{RGB}$

(c) 3D shape regression
 $(x, y, z) \rightarrow \text{occupancy}$

(d) MRI reconstruction
 $(x, y, z) \rightarrow \text{density}$

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 $u \in \mathbb{R}$

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

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

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

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

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

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

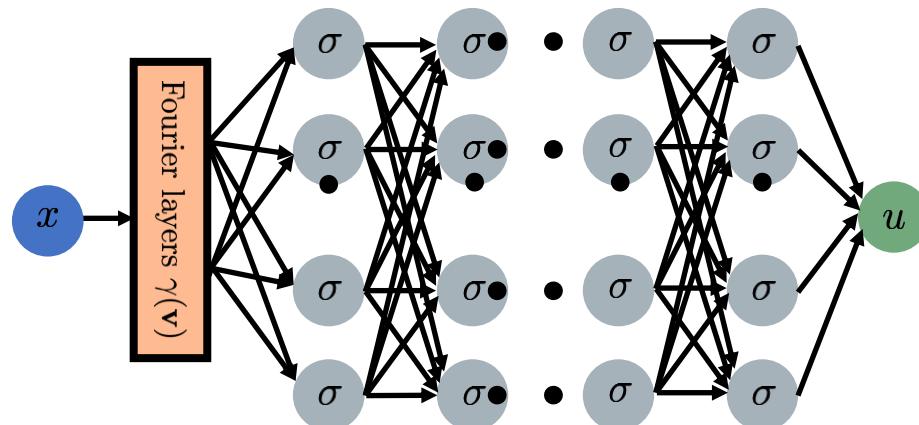
Capturing high-frequency functions

- For multiscale problems, it is important to rescale the NN weights so that it activates 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 $w_j^{(0)} \propto k$

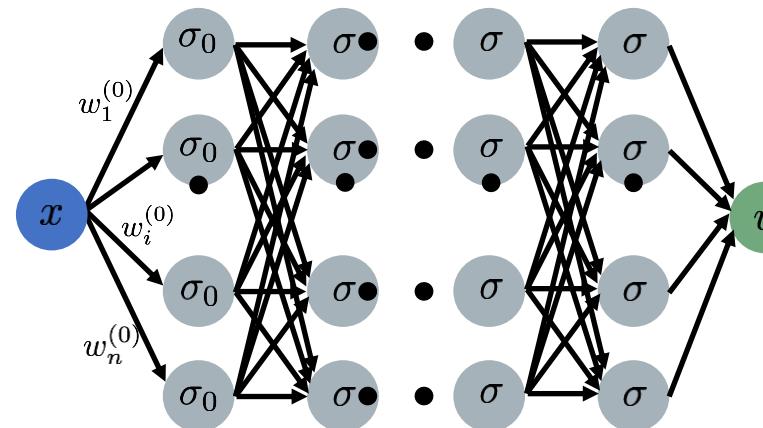
Fourier Feature nets Tancik *et. al.* (2020)



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

Gaussian distribution with standard deviation κ

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



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

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.

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

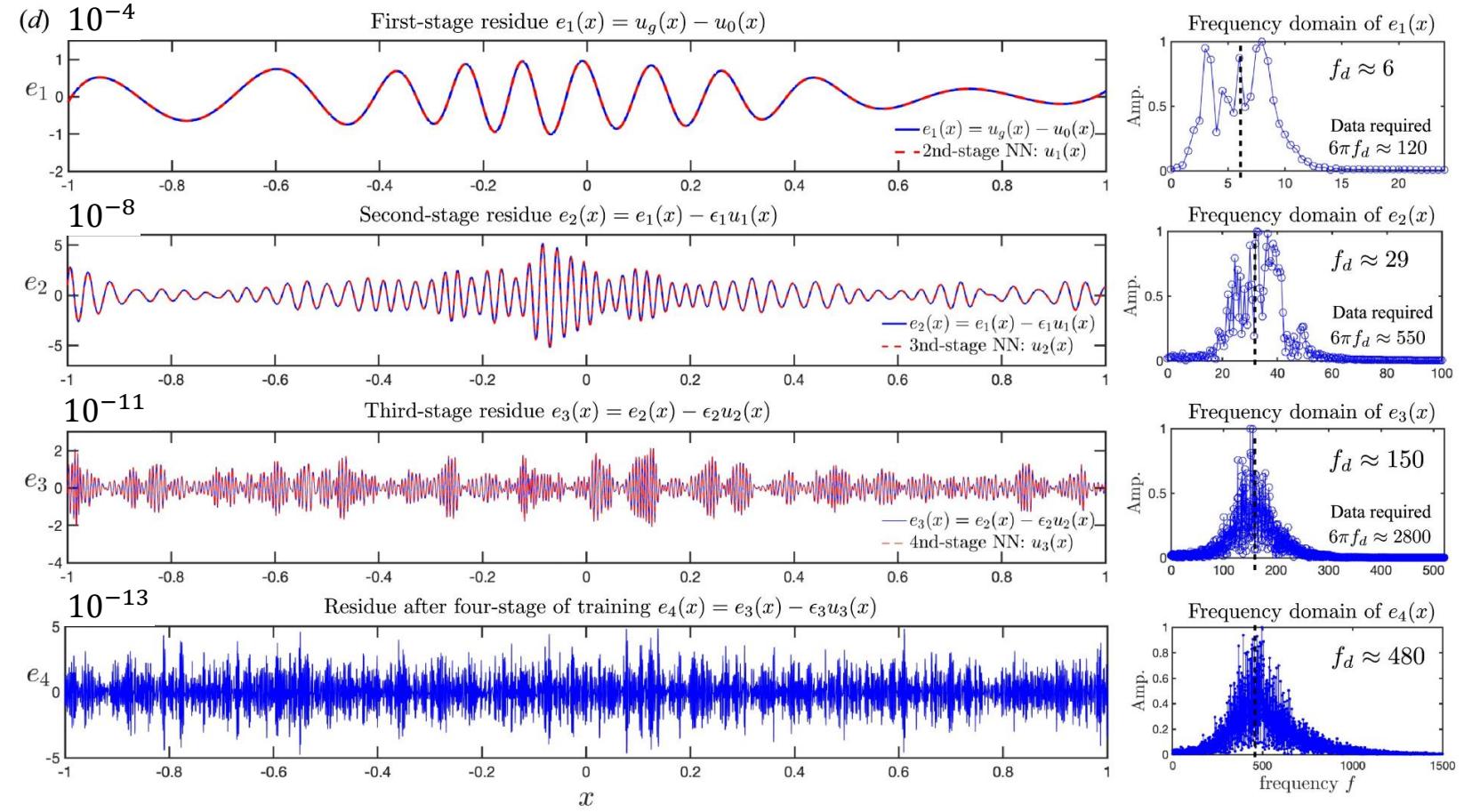
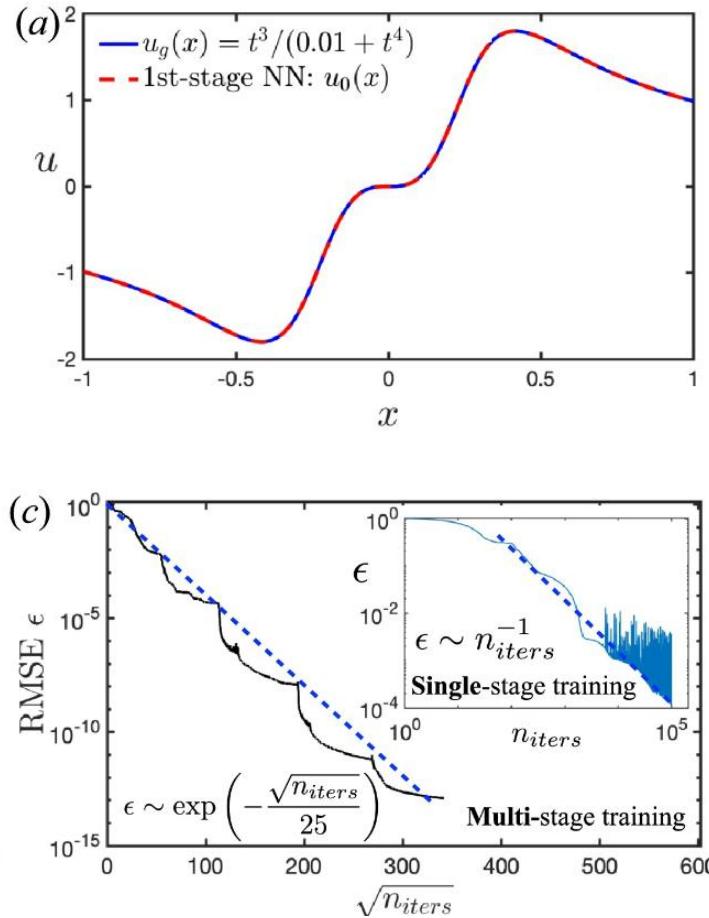
where $1 \gg \epsilon_1 \gg \epsilon_2 \gg \dots \gg \epsilon_n$

$u_0(x)$, $u_1(x)$ and $u_n(x)$ are all of order $O(1)$ 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

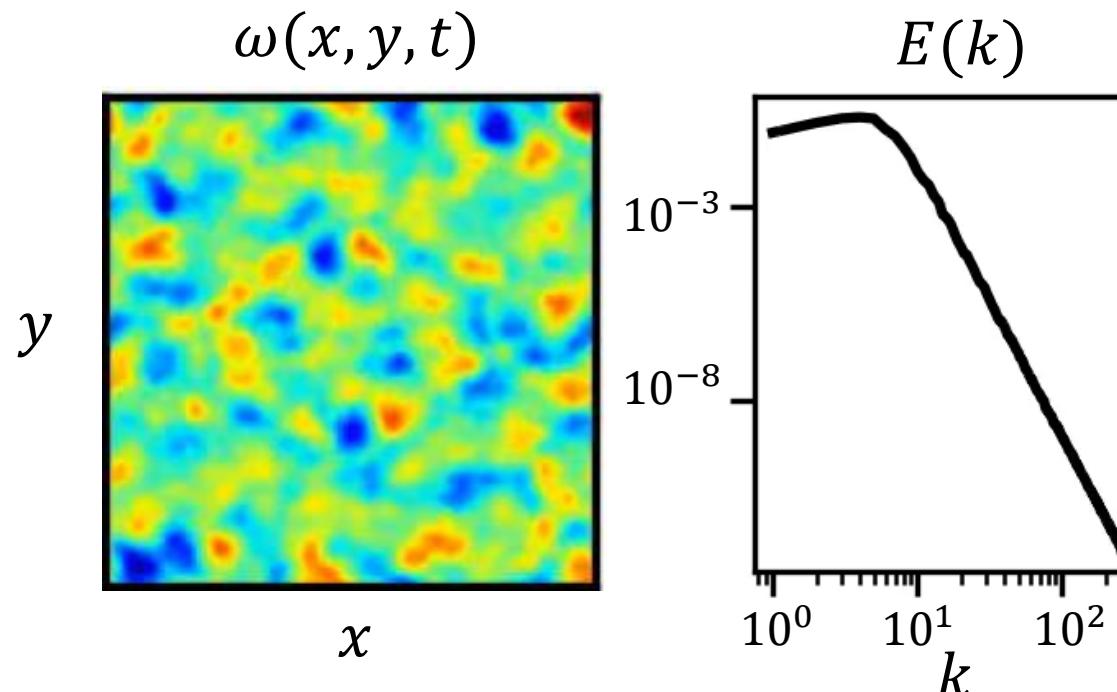
ψ : stream function

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

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

Training data:

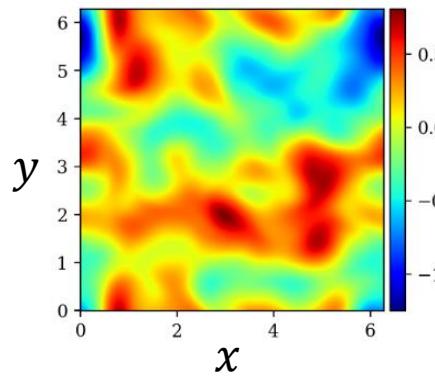
Numerical solution at $\text{Re} = 2000$



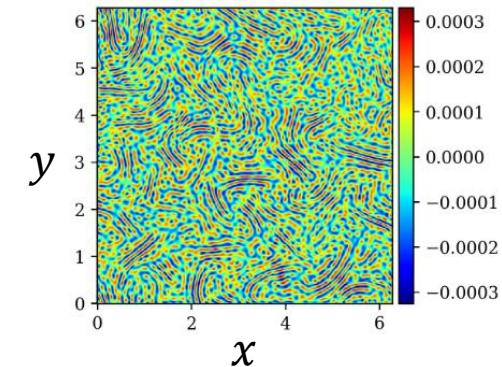
Energy spectrum: $E(k, t) = \sum_{k-\Delta k \leq |\mathbf{k}| \leq k+\Delta k} \frac{1}{2} k^2 |\tilde{\psi}(\mathbf{k}, t)|^2$

E.g. 2D incompressible Navier-Stokes

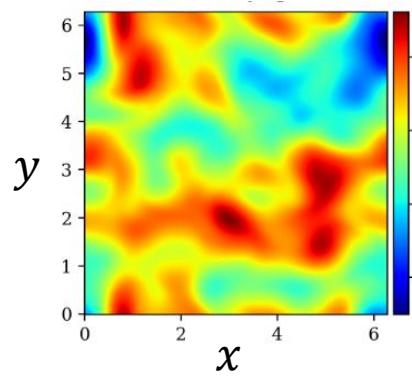
Target $\psi(x, y)$



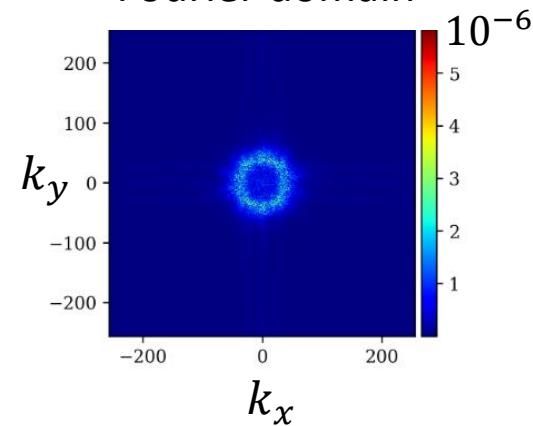
Error



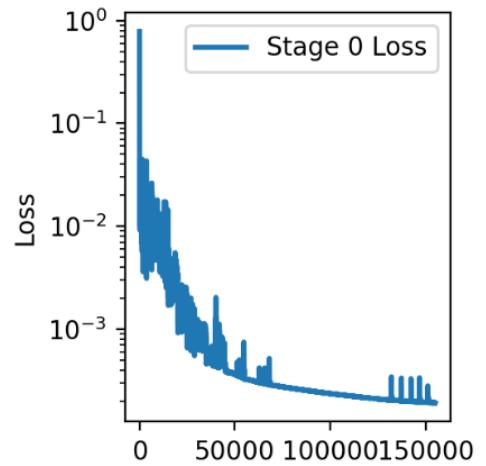
NN $\psi(x, y)$



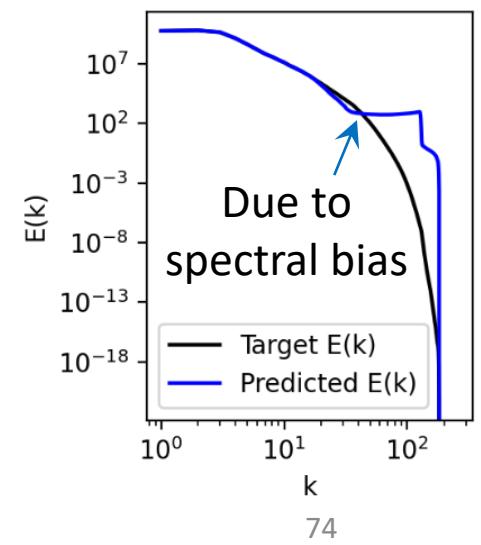
Error in Fourier domain



Loss convergence

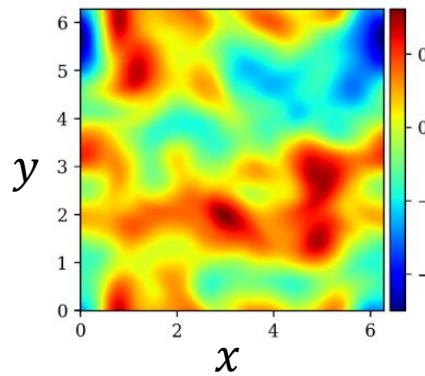


Energy spectrum



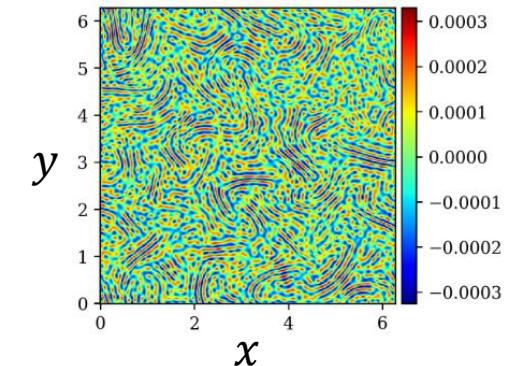
E.g. 2D incompressible Navier-Stokes

Target $\psi(x, y)$



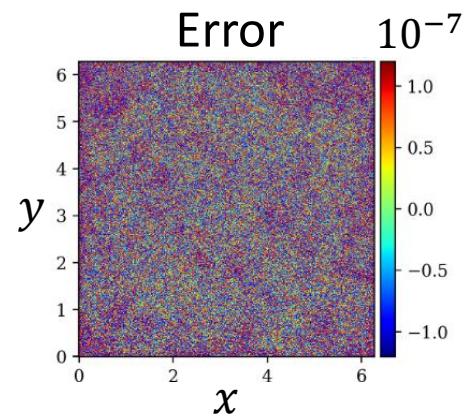
Stage 0

Error

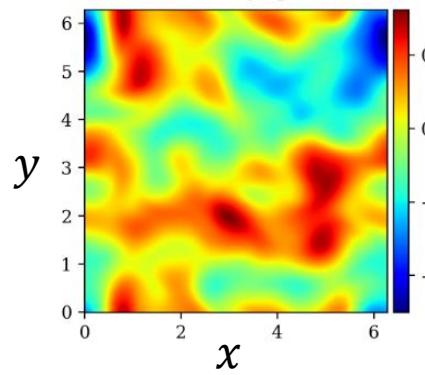


Stage 1

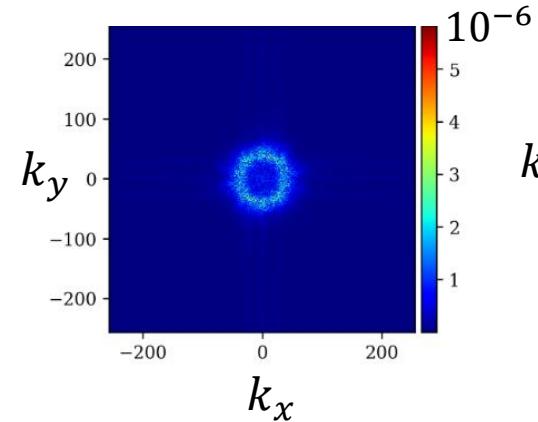
Error



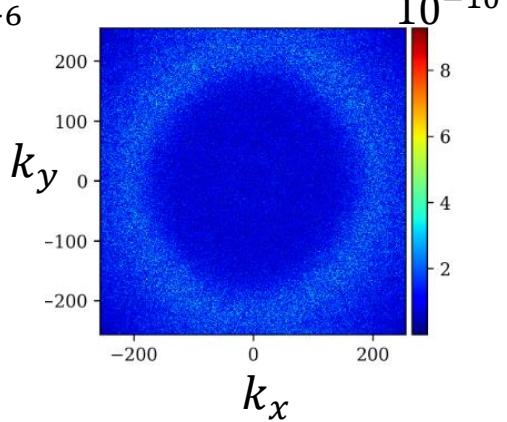
NN $\psi(x, y)$



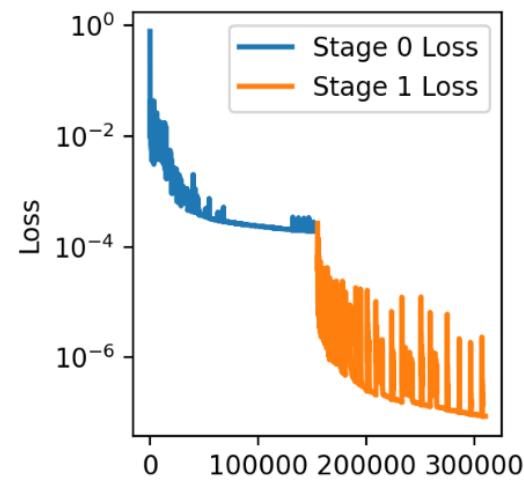
Error in Fourier domain



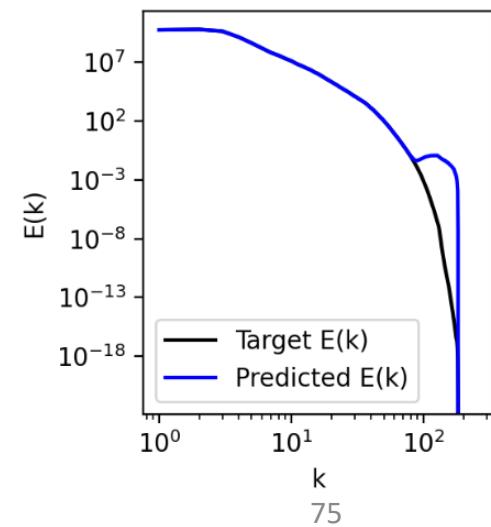
Error in Fourier domain



Loss convergence



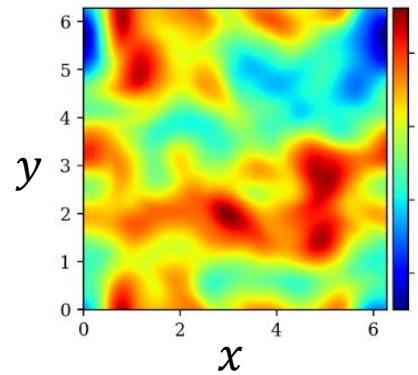
Energy spectrum



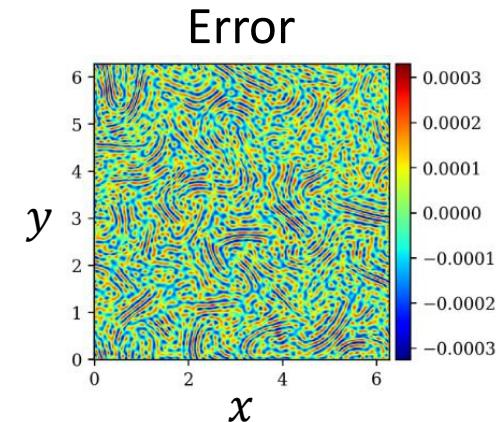
E.g. 2D incompressible Navier-Stokes

Loss convergence

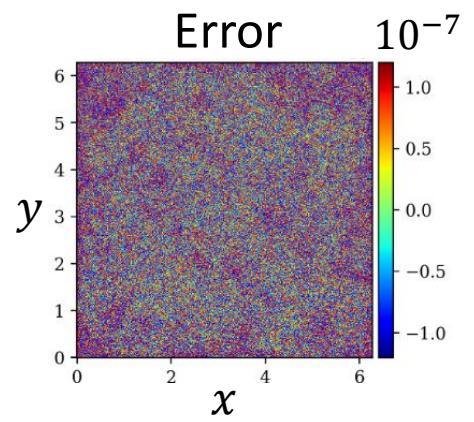
Target $\psi(x, y)$



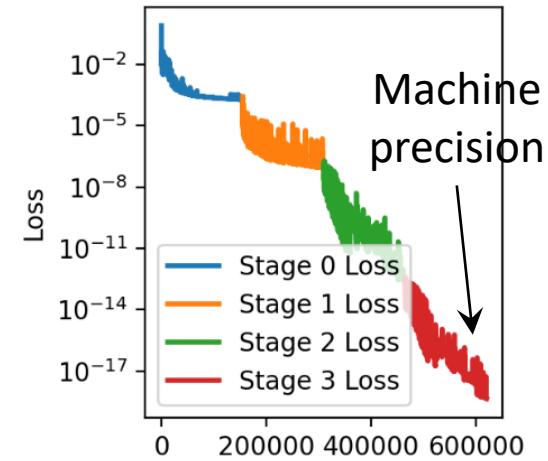
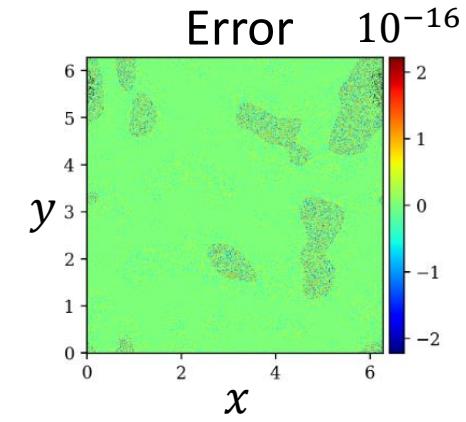
Stage 0



Stage 1

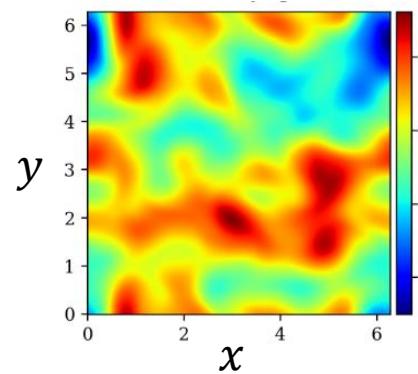


.... **Stage 3**

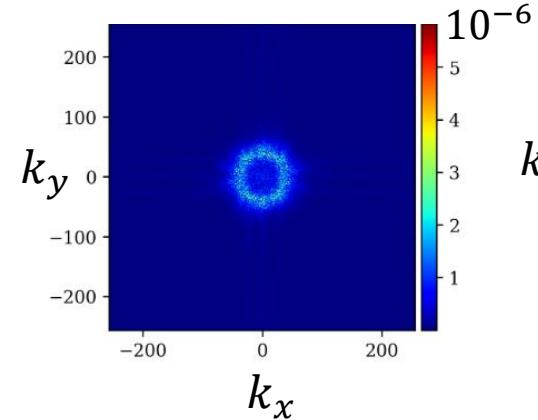


Energy spectrum

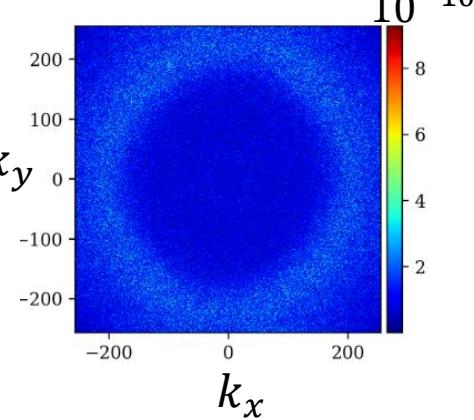
NN $\psi(x, y)$



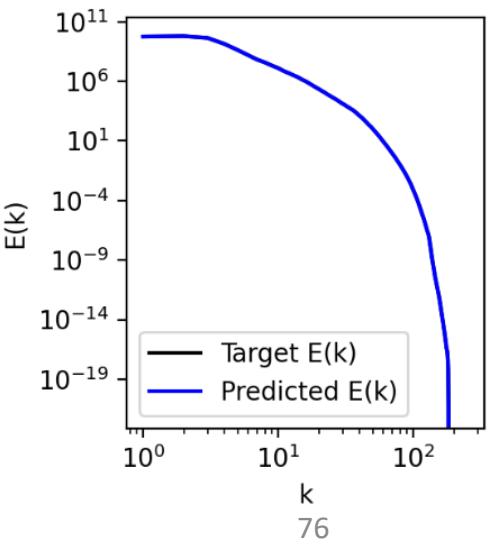
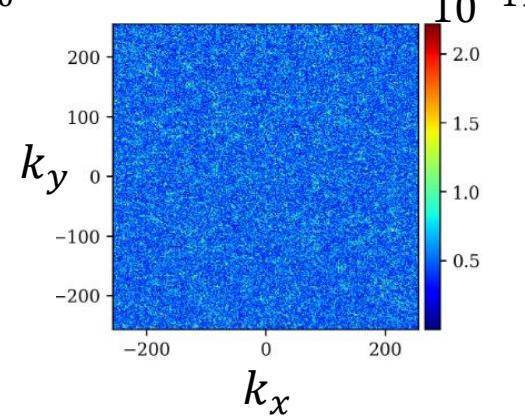
Error in Fourier domain



Error in Fourier domain

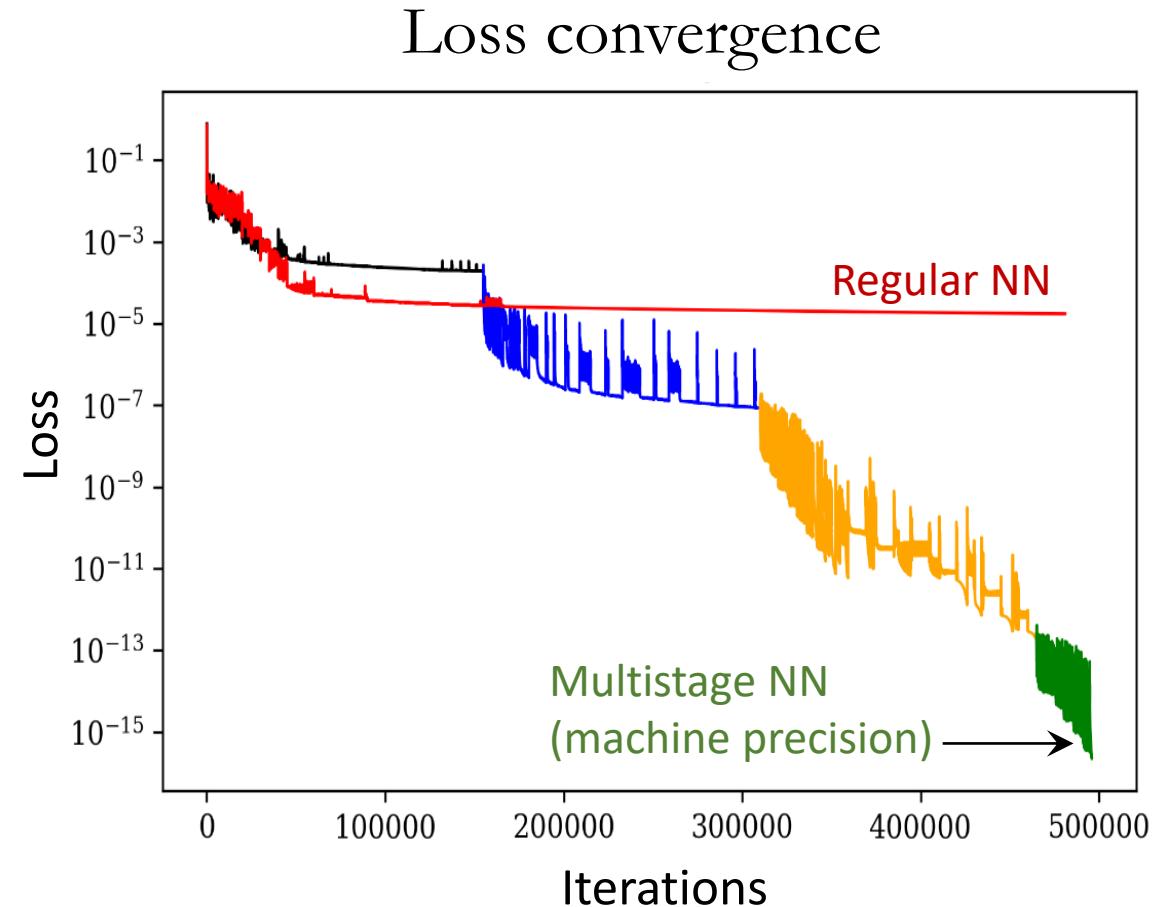
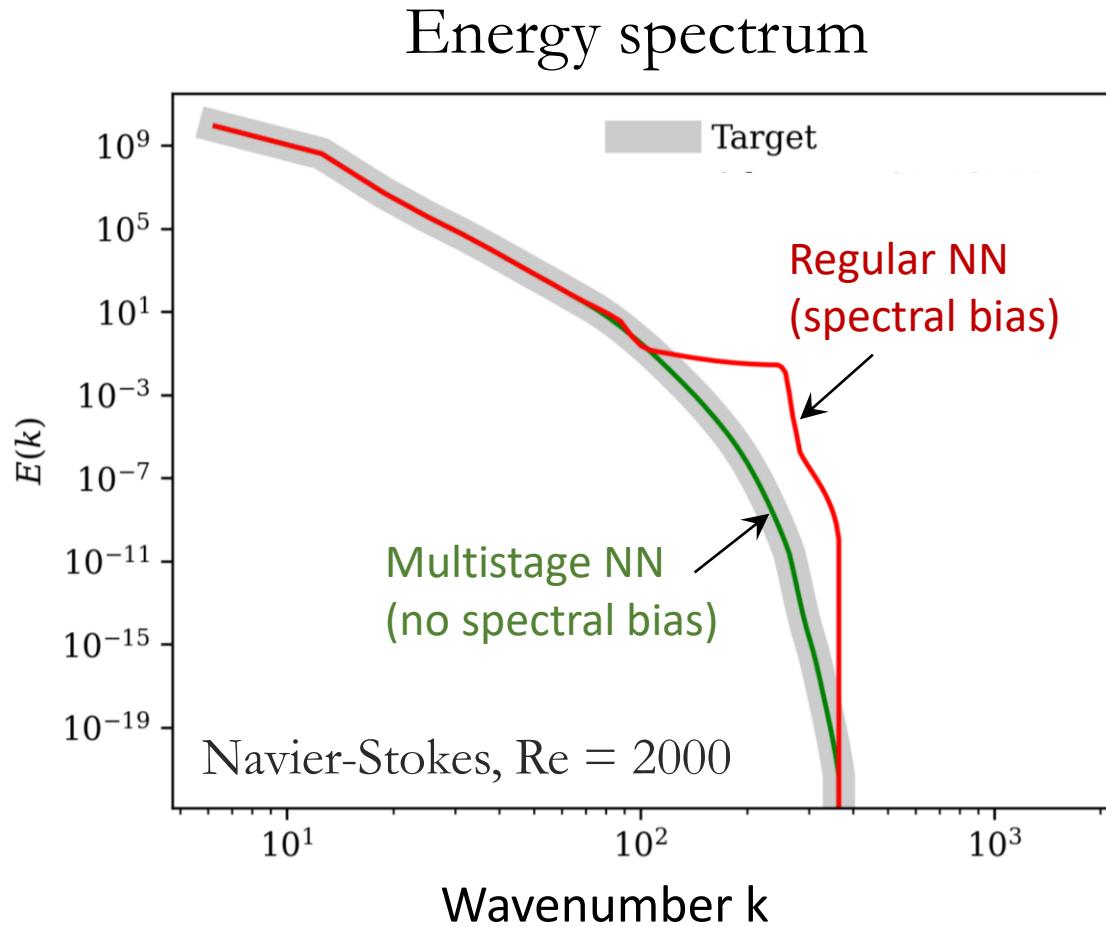


Error in Fourier domain



Multistage NN can tackle the spectral bias & accelerate loss convergence

Ng, Wang & Lai, ICML, accepted
<https://openreview.net/pdf?id=59hdCpJyHx>



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

Machine Learning for Climate Physics and Simulations

[arXiv:2404.13227](https://arxiv.org/abs/2404.13227)

Ching-Yao Lai,¹ Pedram Hassanzadeh,² Aditi Sheshadri³, Maike Sonnewald⁴, Raffaele Ferrari⁵, Venkatramani Balaji⁶

¹Department of Geophysics, Stanford University, Stanford, CA 94305, USA;
email: cyaolai@stanford.edu

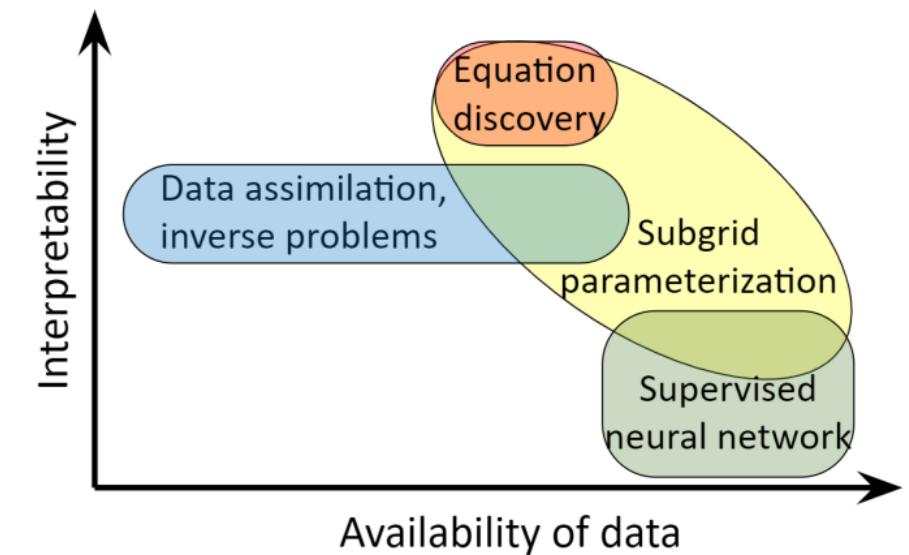
²Department of Geophysical Sciences and Committee on Computational and Applied Mathematics, University of Chicago, Chicago, IL 60637, USA

³Department of Earth System Sciences, Stanford University, Stanford, CA 94305, USA

⁴Department of Computer Science, University of California Davis, Davis, CA 95616, USA

⁵Department of Earth, Atmospheric, and Planetary Sciences, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

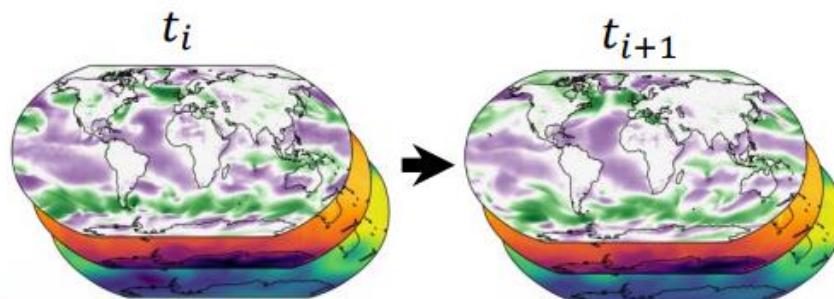
⁶Schmidt Sciences, USA



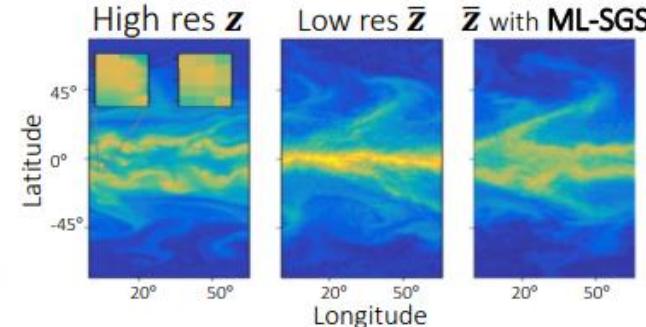
Machine Learning for Climate Simulations

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

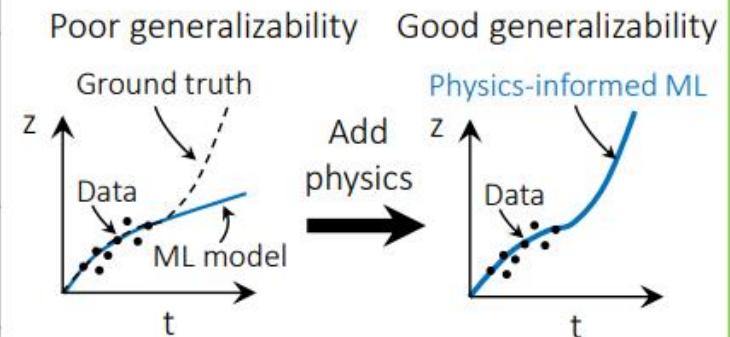
i. Weather forecast via ML emulator



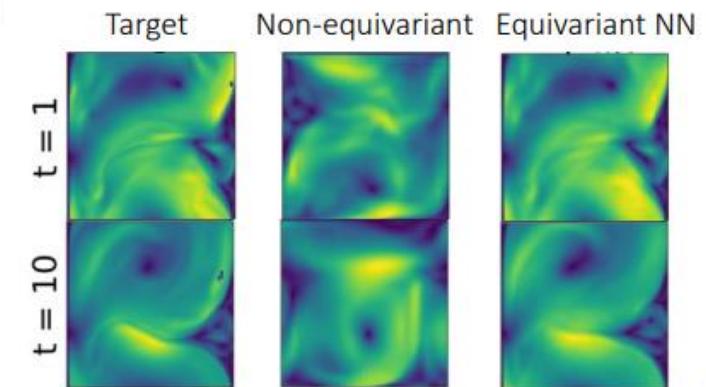
j. SGS parametrization



g. Physics-informed ML



h. Better generalization capacity via incorporating symmetries into NN

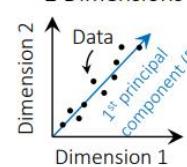


Machine Learning for Climate Physics

I. Data-informed knowledge discovery

a. Dimensionality reduction

2 Dimensions



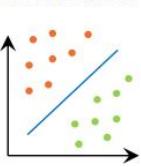
PCA

1 Dimension

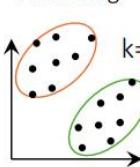


b. Supervised vs unsupervised

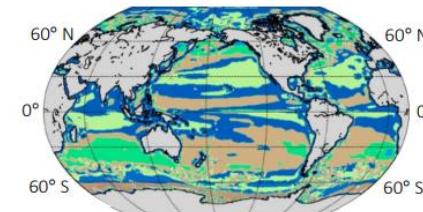
Classification



Clustering



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



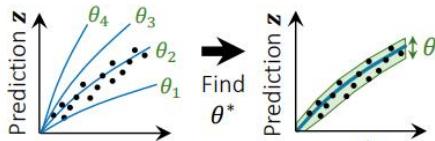
II. Data-informed model discovery

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

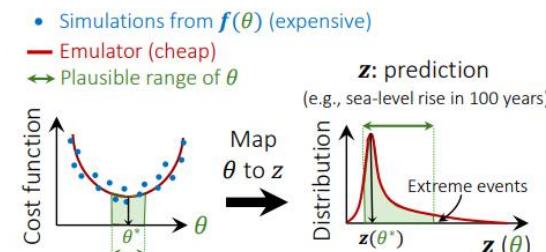
	Parametric Estimation $\boldsymbol{\theta}$	State Estimation $\mathbf{z}(t)$	Structural Estimation \mathbf{f}
Goal	Discover parameters of a mathematical model that best fits a given dataset.	Discover states of a system given sparse, noisy and discrete data.	Discover a mathematical model that best fits 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 \mathbf{f}	$x(t), y(t)$ at sparse times, and the mathematical form of \mathbf{f} , including $\boldsymbol{\theta}$	$x(t), y(t), z(t)$ at lots of times
Predict	$\boldsymbol{\theta}$	State variables $x(t), y(t), z(t)$ at lots of times	The mathematical form of the model \mathbf{f} , including $\boldsymbol{\theta}$

d. Parameter estimation

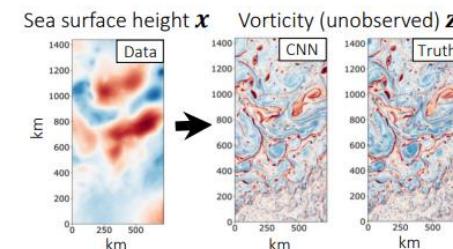
- Simulations from $\mathbf{f}(\boldsymbol{\theta})$
- Data
- θ^* : Calibrated parameter with UQ



e. Uncertainty quantification (UQ)



f. State estimation example:



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:

