# MACHINE LEARNING IN PHYSICS

An overview with applications to fluid mechanics

David Sondak
UT San Antonio
4/14/2024

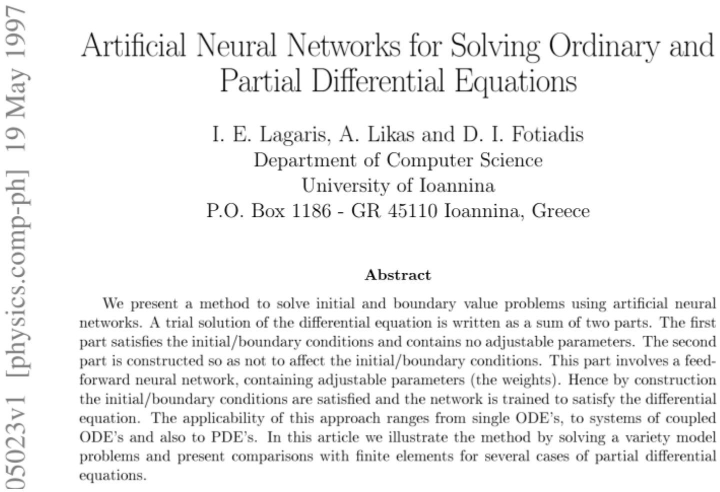
# TABLE OF CONTENTS

# MOTIVATION

# MOTIVATION FOR ML IN PHYSICS

- Machine learning naturally leads to data-driven modeling


- In physics, we already know the governing conservation laws --- why not use them?

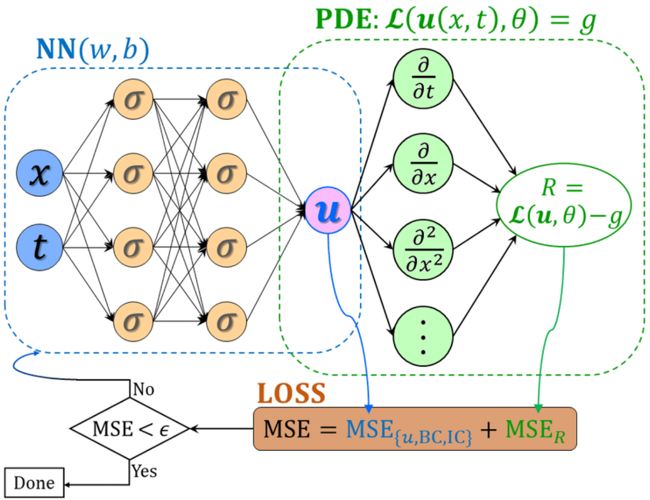

05023v1 [physics.comp-ph] 19 May 1997

Artificial Neural Networks for Solving Ordinary and Partial Differential Equations

I. E. Lagaris, A. Likas and D. I. Fotiadis
Department of Computer Science
University of Ioannina
P.O. Box 1186 - GR 45110 Ioannina, Greece

**Abstract**

We present a method to solve initial and boundary value problems using artificial neural networks. A trial solution of the differential equation is written as a sum of two parts. The first part satisfies the initial/boundary conditions and contains no adjustable parameters. The second part is constructed so as not to affect the initial/boundary conditions. This part involves a feed-forward neural network, containing adjustable parameters (the weights). Hence by construction the initial/boundary conditions are satisfied and the network is trained to satisfy the differential equation. The applicability of this approach ranges from single ODE's, to systems of coupled ODE's and also to PDE's. In this article we illustrate the method by solving a variety model problems and present comparisons with finite elements for several cases of partial differential equations.



# A PARTIAL SUCCESS STORY IN FLUIDS

- The Reynolds-averaged momentum equation for an incompressible fluid is

$$\rho\frac{\partial\overline{\boldsymbol{u}}}{\partial t}+\rho\nabla\cdot(\overline{\boldsymbol{u}}\otimes\overline{\boldsymbol{u}})=\nabla\cdot\left[-\overline{P}\boldsymbol{I}+\mu\left(\nabla\overline{\boldsymbol{u}}+\nabla\overline{\boldsymbol{u}}^{T}\right)-\boxed{\overline{\rho\boldsymbol{u}'\otimes\boldsymbol{u}'}}\right].$$

- A tremendous number of models have been proposed for the anisotropic Reynolds stress tensor.
- Ling et al. (JFM, 2016) proposed a **physics-based neural network** to represent the normalized anisotropic Reynolds stress tensor and obtained promising results.

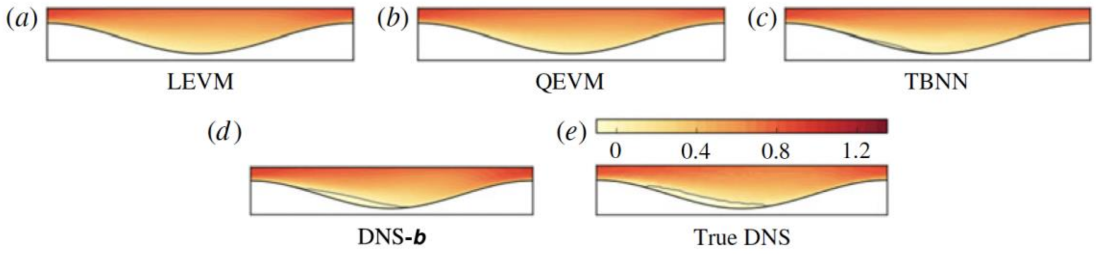



FIGURE 5. Contours of streamwise velocity normalized by bulk velocity in the wavy wall test case, zoomed into the near-wall region. Separated regions outlined in grey.

Many groups around the world are hunting for physics-based ML algorithms.

# WHAT IS MACHINE LEARNING?

# DATA SCIENCE IS NOT NEW

- Not a new field; it goes back to the ancients

Data on trajectories of astrophysical objects!

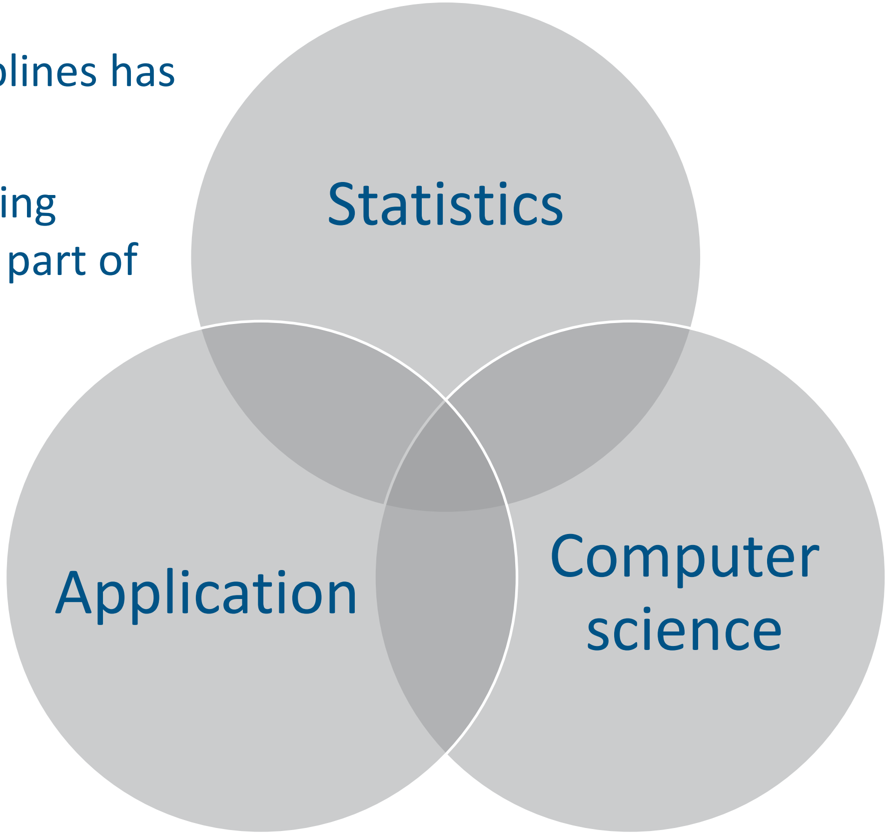
- Linear regression is one of the first complete data-driven algorithms
  - It's machine learning without the machines!

# THE BIG PICTURE

## Machine Learning in Context

- Access to a deluge of data in various disciplines has given rise to the field of “data science”
- New algorithms for working with and making models and predictions from this data are part of “machine learning”
- Considerable success in a variety of fields
  - Natural language processing (NLP)
  - Personalized health care
  - Self-driving cars
  - Business
  - …
- The governing laws are often unknown
  - Use data to gain insight

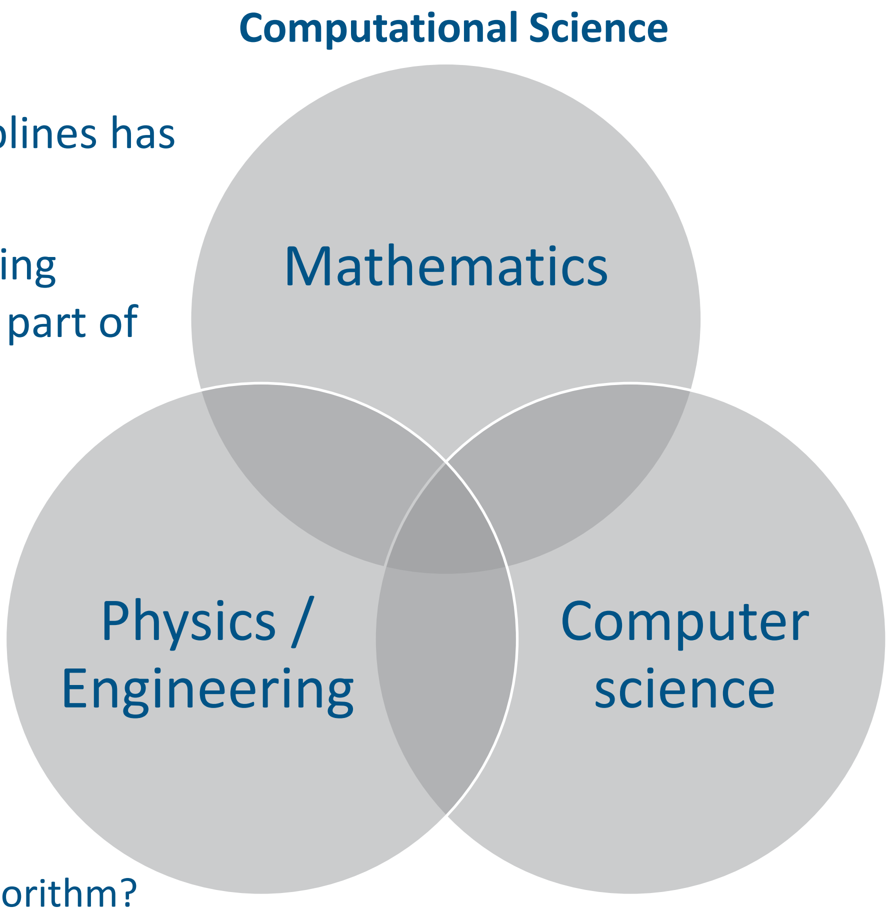
**Data Science**



# THE BIG PICTURE

## Machine Learning in Context

- Access to a deluge of data in various disciplines has given rise to the field of "data science"
- New algorithms for working with and making models and predictions from this data are part of "machine learning"
- Considerable success in a variety of fields
  - Natural language processing (NLP)
  - Personalized health care
  - Self-driving cars
  - Business
  - …
- **What about physics?**
  - How to build a physics-aware, data-driven algorithm?

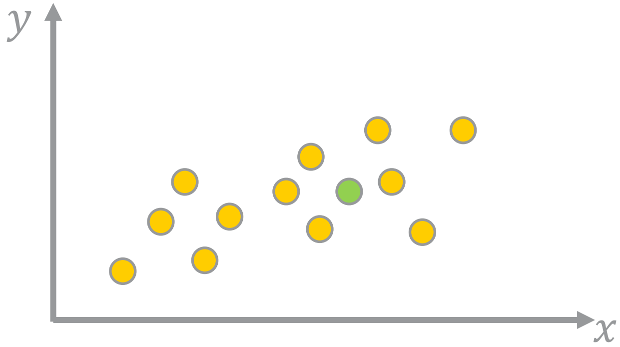


# TERMINOLOGY AND POPULAR ALGORITHMS

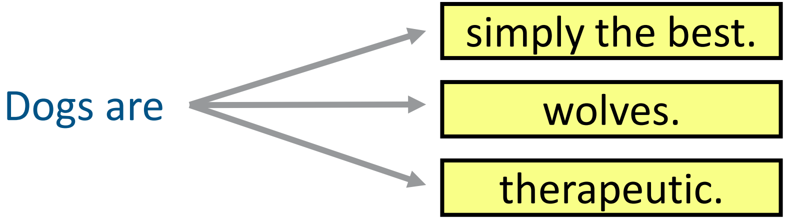
# TYPES OF MACHINE LEARNING TASKS

- Classification
  - Assign the input to a category
  - Fun example
    - Input to algorithm: Snapshots of a flow field
    - Output: Classification category (0:turbulent, 1:laminar)

- Regression
  - Predict a numerical value given some input
  - Most classic example: Linear regression




- Transcription and translation
  - Predict the next words in a sentence



# TERMINOLOGY: DATASET

$$\{(y, x)\}$$

Observations

Features

- **Observations**: Dog breed (e.g. aussie, lab, husky)
- **Features**: Size, ears, eyes, snout, color, fur length, etc
- **Note**: We don’t always have labeled data and this is sometimes ok.

# TERMINOLOGY: A MODEL

$$\hat{y} = f(x;\theta)$$

Prediction

Mathematical model

Inputs

Parameters

**Note**: Showing a parametric model here, but this need not be the case.

- The mathematical model can really be anything
  - Algebraic model, differential equation, etc.
- Often, we are interested in the inverse problem:
  - Given experimental data, can we determine the parameters?
  - Given experimental data, can we determine the *model*?
- How is this accomplished?

# TERMINOLOGY: LOSS

a.k.a. Cost function, objective function, …

- Question: How do we know if the model is making acceptable predictions?
- An Answer: Introduce a metric that indicates how the model is performing.

Loss function $\longrightarrow$ $$L[y, \hat{y}_\theta]$$

$$\hat{y}_\theta = f(x; \theta)$$

- All different kinds of loss functions: $L^2$, $L^p$, $H_1$, …
- Many specialized loss functions --- not necessarily true norms
- Can turn this into an argmin problem over the parameters $\theta$.
- How to perform this minimization problem?

# TERMINOLOGY: OPTIMIZATION

Where the magic happens

$$\theta_* = \underset{\theta}{\operatorname{argmin}}\, L[y, \hat{y}(x;\theta)]$$

"Best" model parameters - These ones minimize the loss.

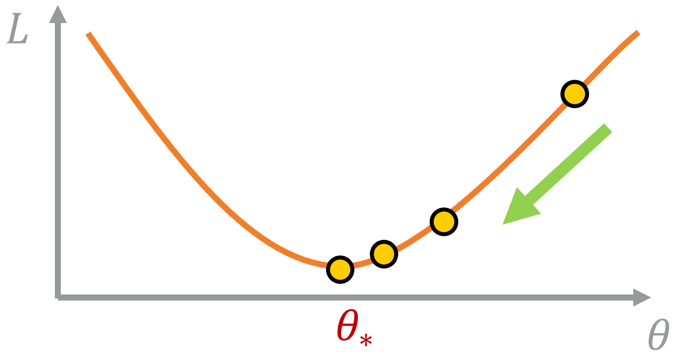
- Modern machine learning relies on variants of *gradient descent:* Stochastic GD

Updated parameters $\longrightarrow$

$$\theta_{k+1} = \theta_k - \lambda \frac{\partial L}{\partial \theta}$$

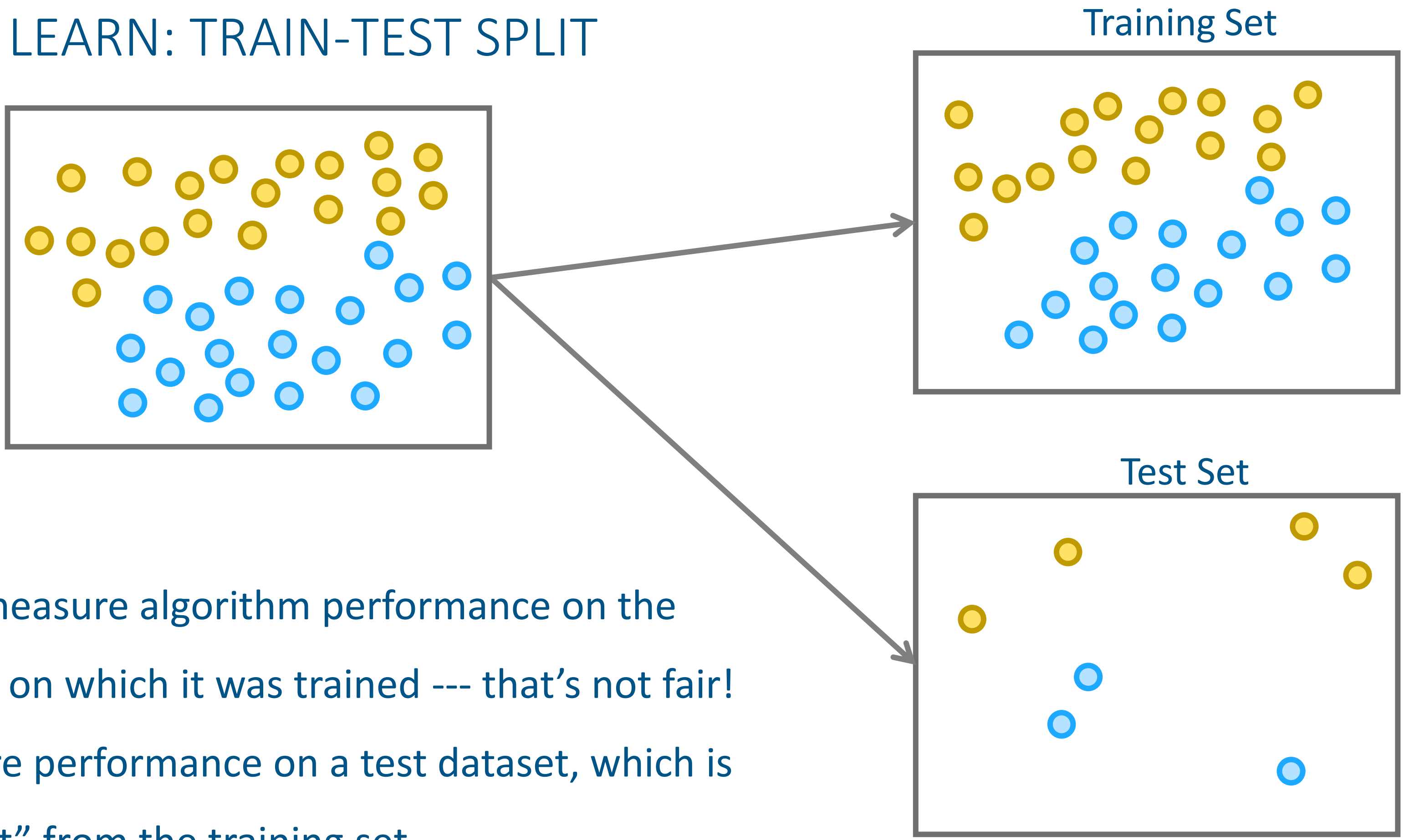
"Old" parameters

"Learning" rate (a.k.a. step size)



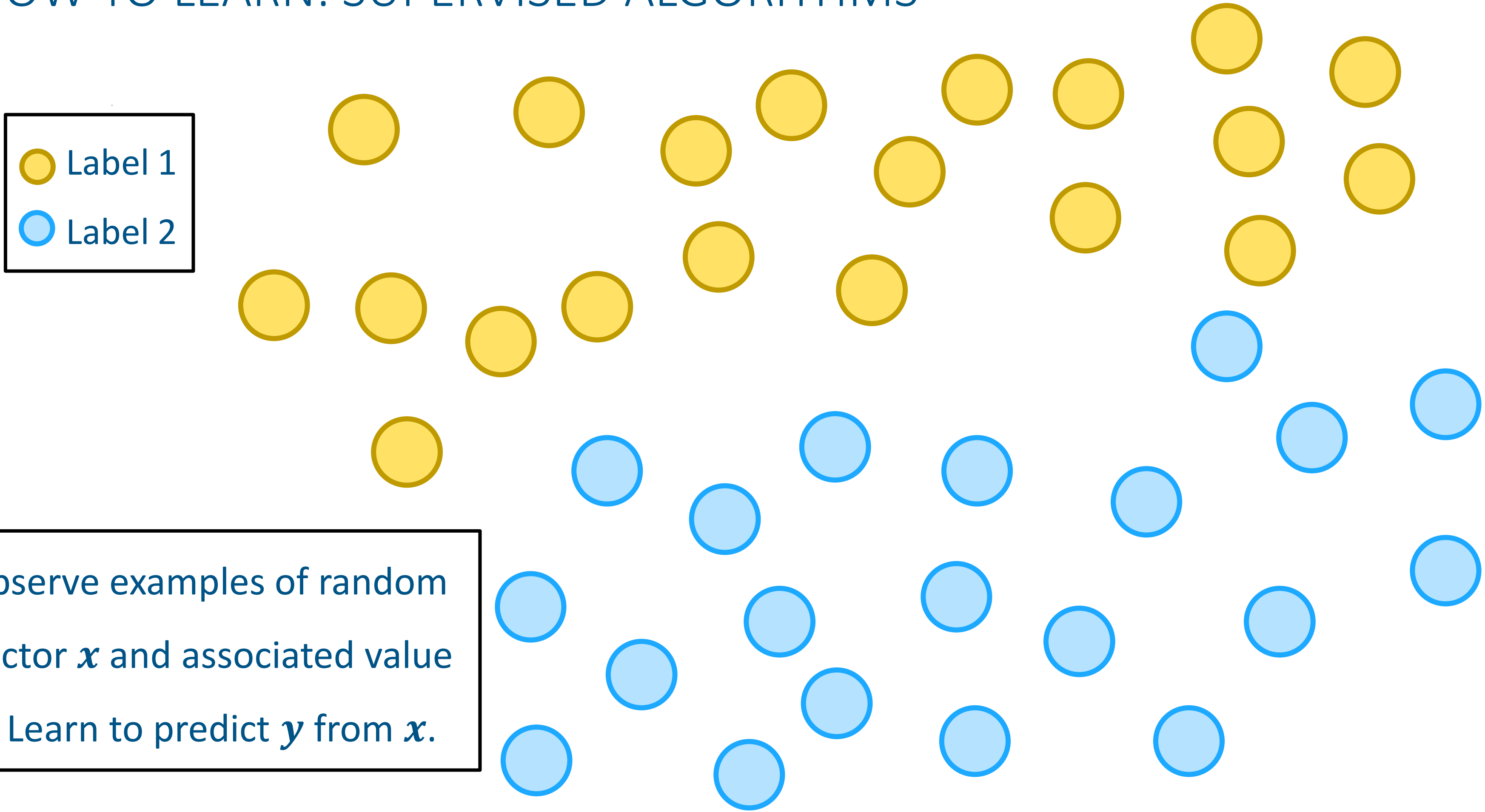

- Sensitive to initial guess $\theta_0$ and *very* sensitive to $\lambda$
- Works for $L$ convex in $\theta$

# HOW TO LEARN: TRAIN-TEST SPLIT



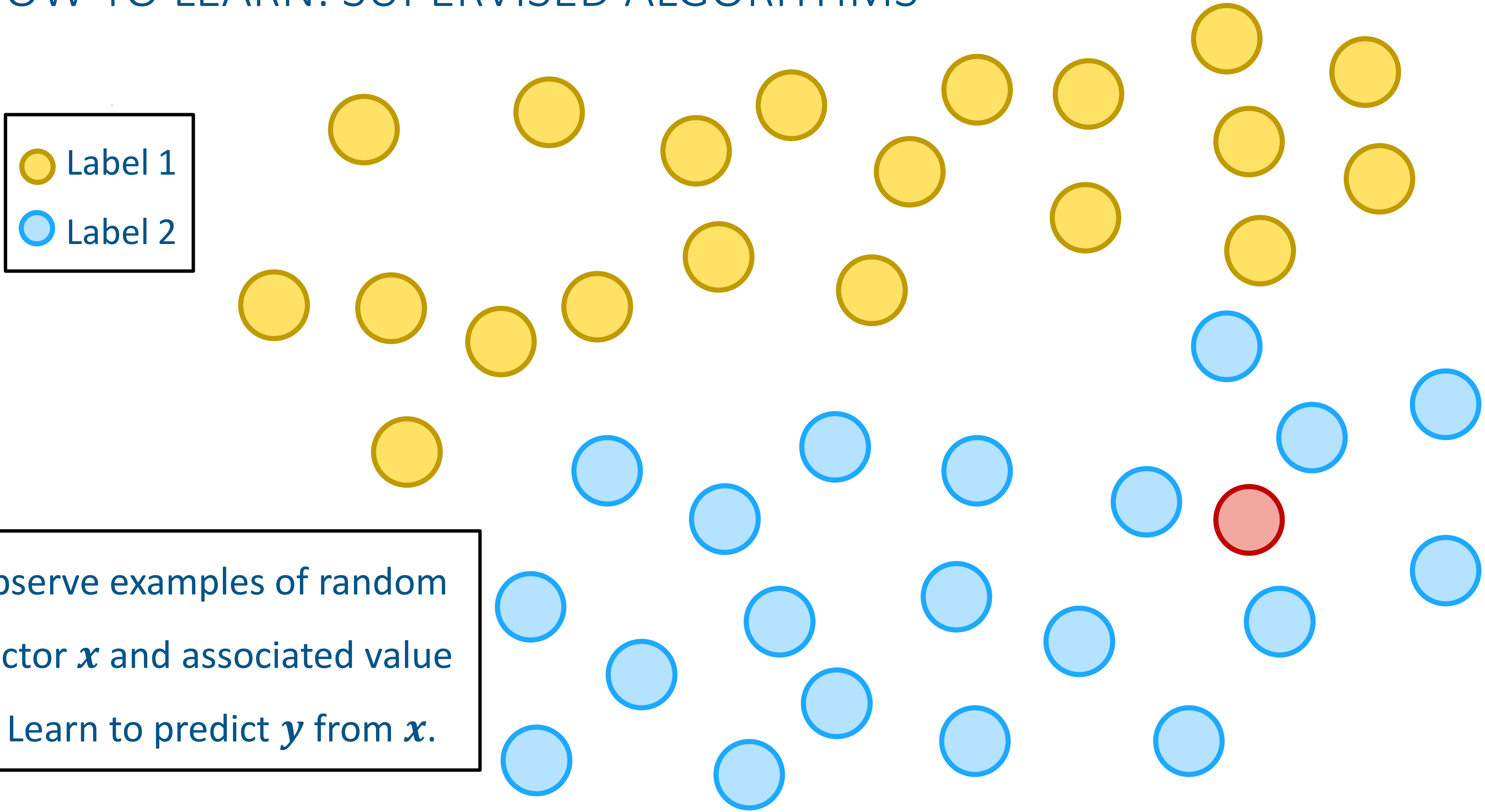

- Don't measure algorithm performance on the dataset on which it was trained --- that's not fair!
- Measure performance on a test dataset, which is "distinct" from the training set.

# HOW TO LEARN: SUPERVISED ALGORITHMS



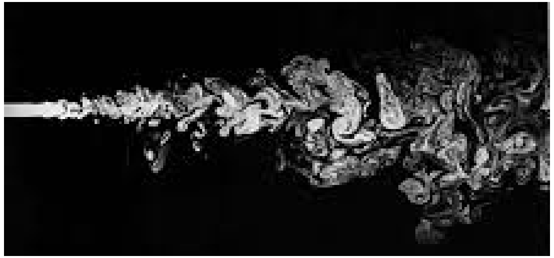

Observe examples of random vector $\boldsymbol{x}$ and associated value $\boldsymbol{y}$. Learn to predict $\boldsymbol{y}$ from $\boldsymbol{x}$.

# HOW TO LEARN: SUPERVISED ALGORITHMS

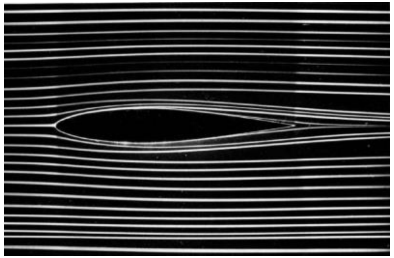



Observe examples of random vector $\boldsymbol{x}$ and associated value $\boldsymbol{y}$. Learn to predict $\boldsymbol{y}$ from $\boldsymbol{x}$.

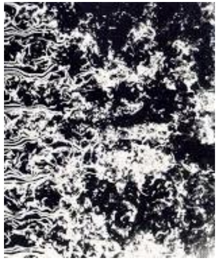
# A LABELED DATASET



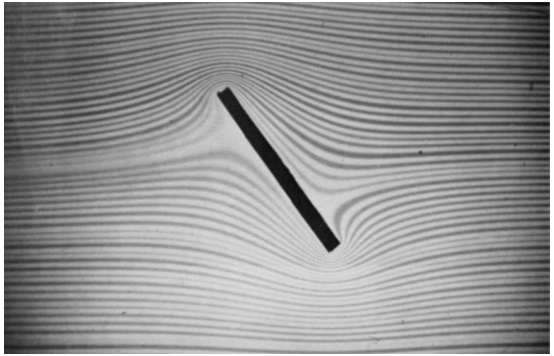
Turbulent



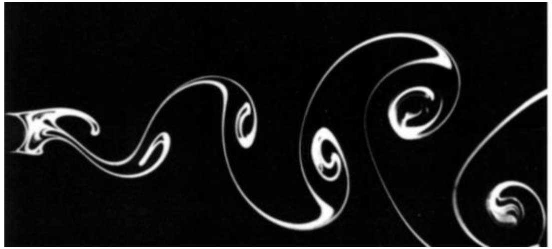
Laminar



Turbulent



Laminar



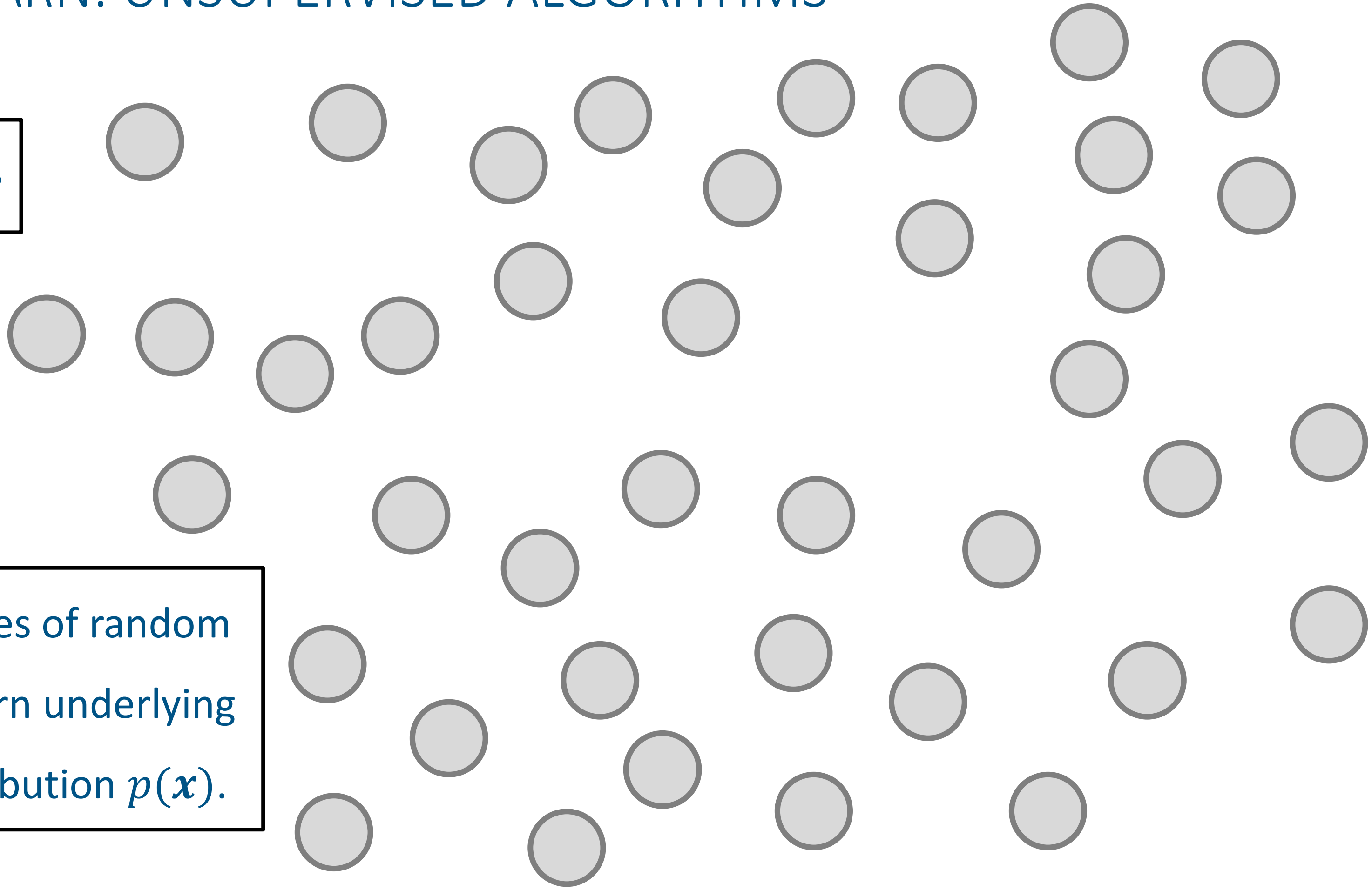
???



Images from van Dyke’s album of fluid motion

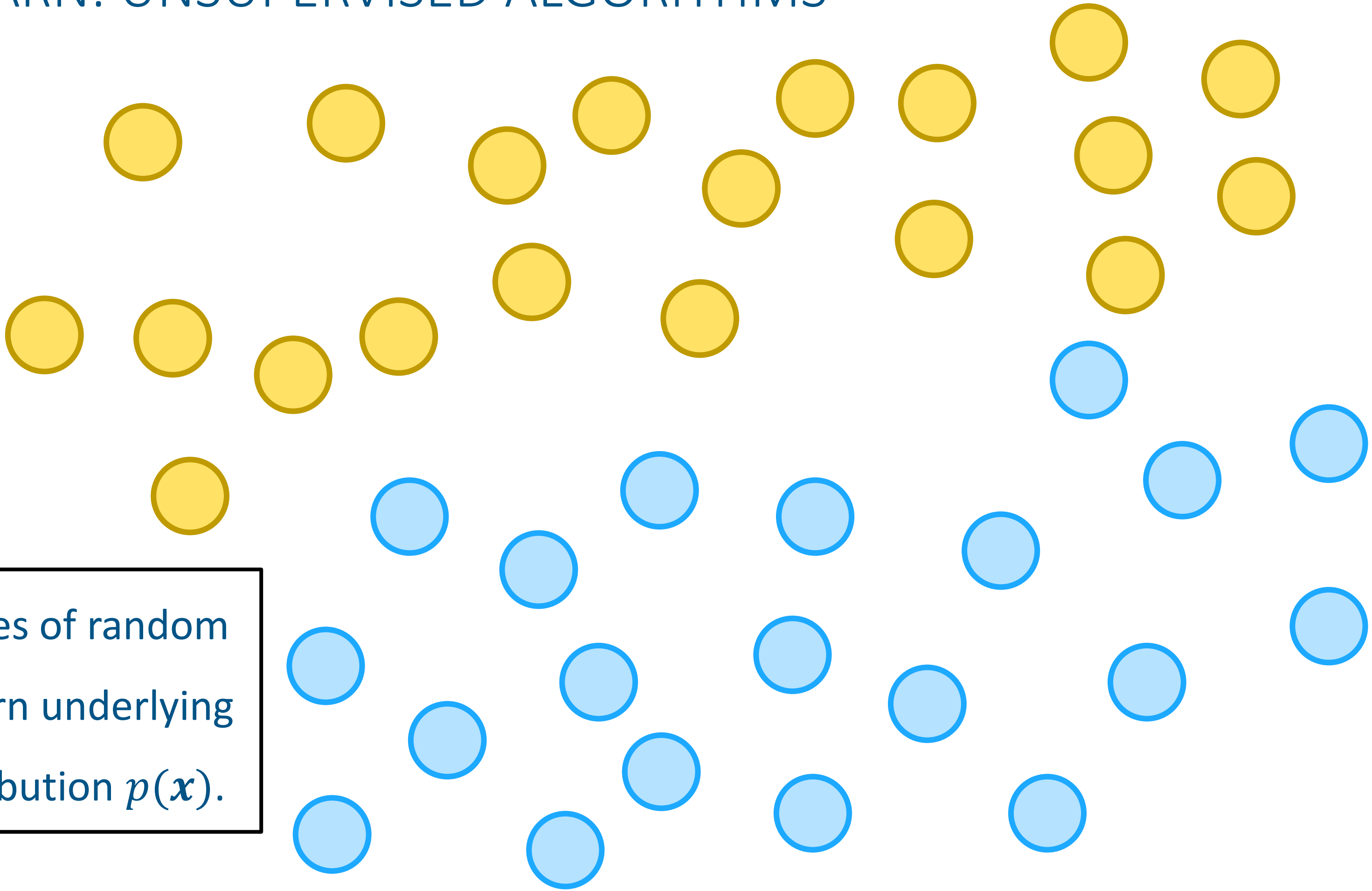
# HOW TO LEARN: UNSUPERVISED ALGORITHMS

No labels



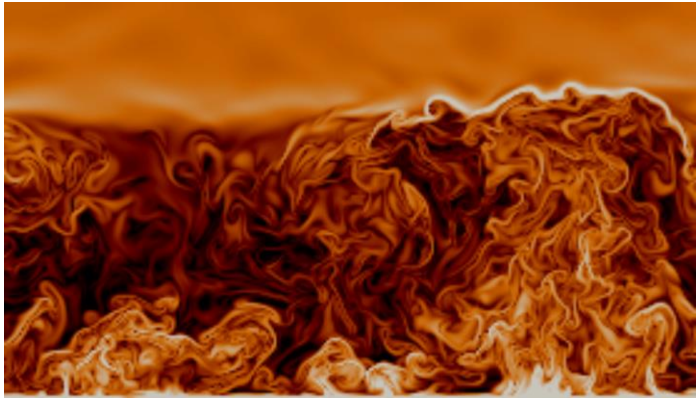
Observe examples of random vector $\boldsymbol{x}$ and learn underlying probability distribution $p(\boldsymbol{x})$.

# HOW TO LEARN: UNSUPERVISED ALGORITHMS



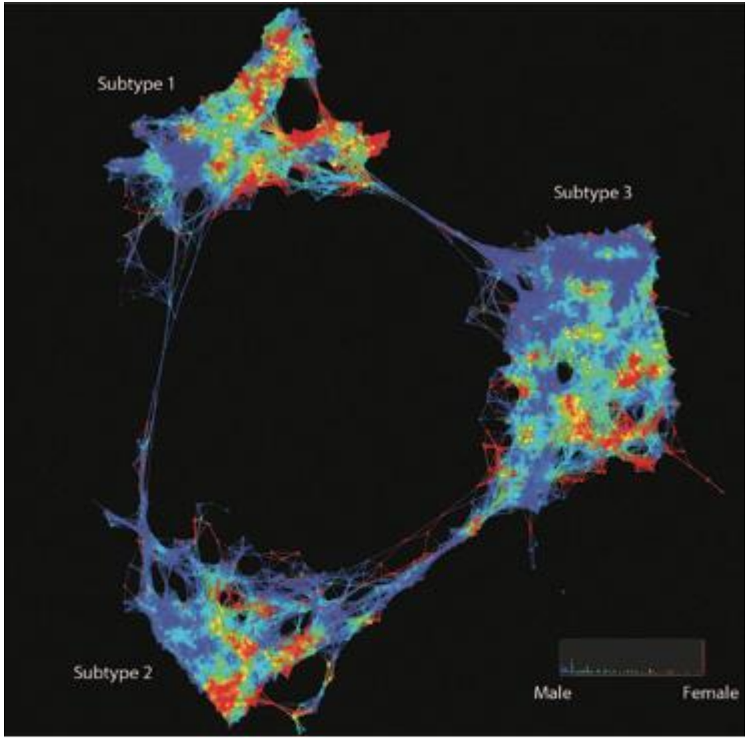
Observe examples of random vector $\boldsymbol{x}$ and learn underlying probability distribution $p(\boldsymbol{x})$.

# AN UNLABELED DATASET



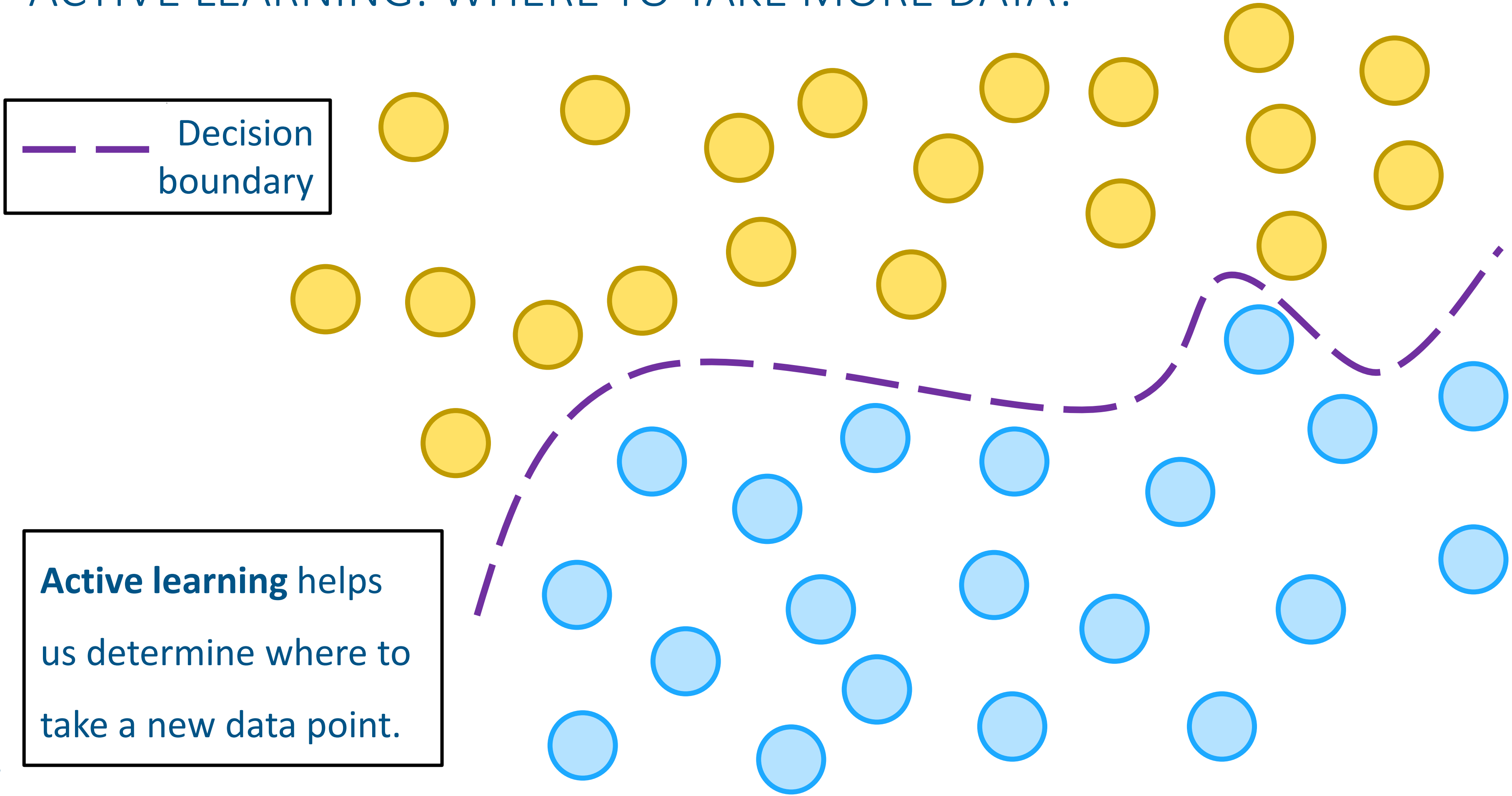
European Geosciences Union blog

Find and extract structure in the dataset.



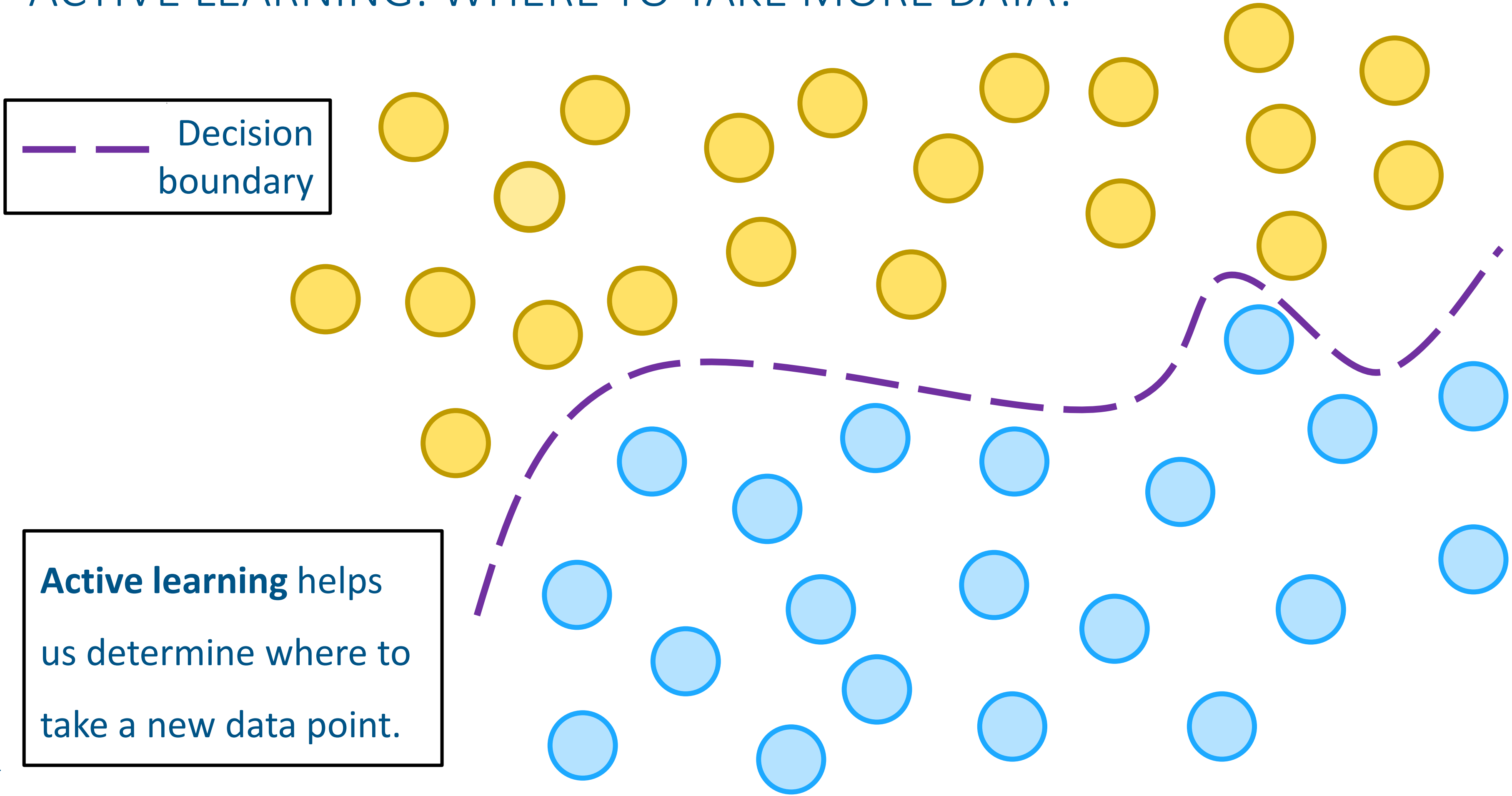

Identification of type 2 diabetes subgroups through topological analysis of patient similarity; Li et al., Science Translational Medicine, 2015

# ACTIVE LEARNING: WHERE TO TAKE MORE DATA?



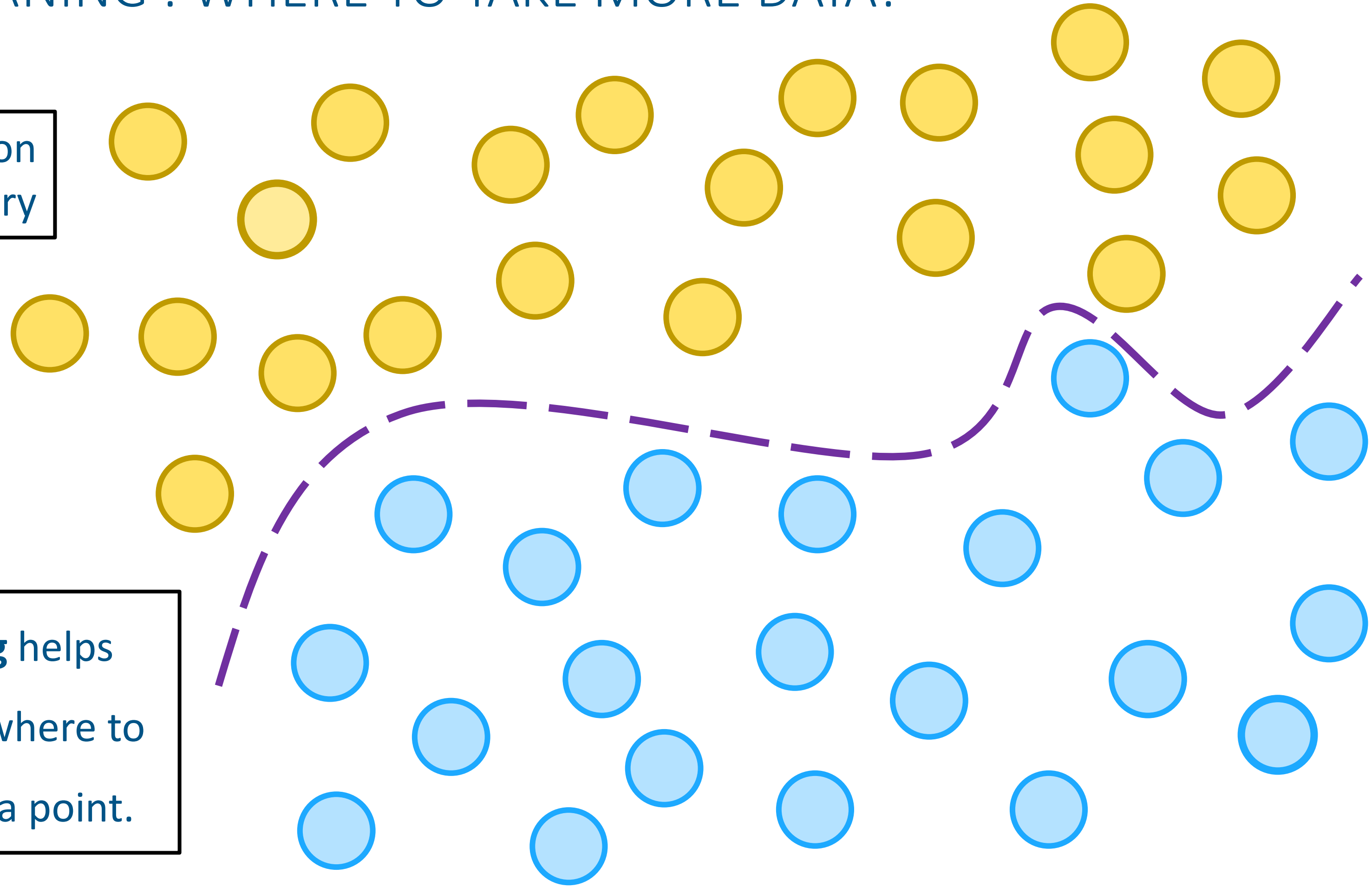

**Active learning** helps us determine where to take a new data point.

# ACTIVE LEARNING: WHERE TO TAKE MORE DATA?



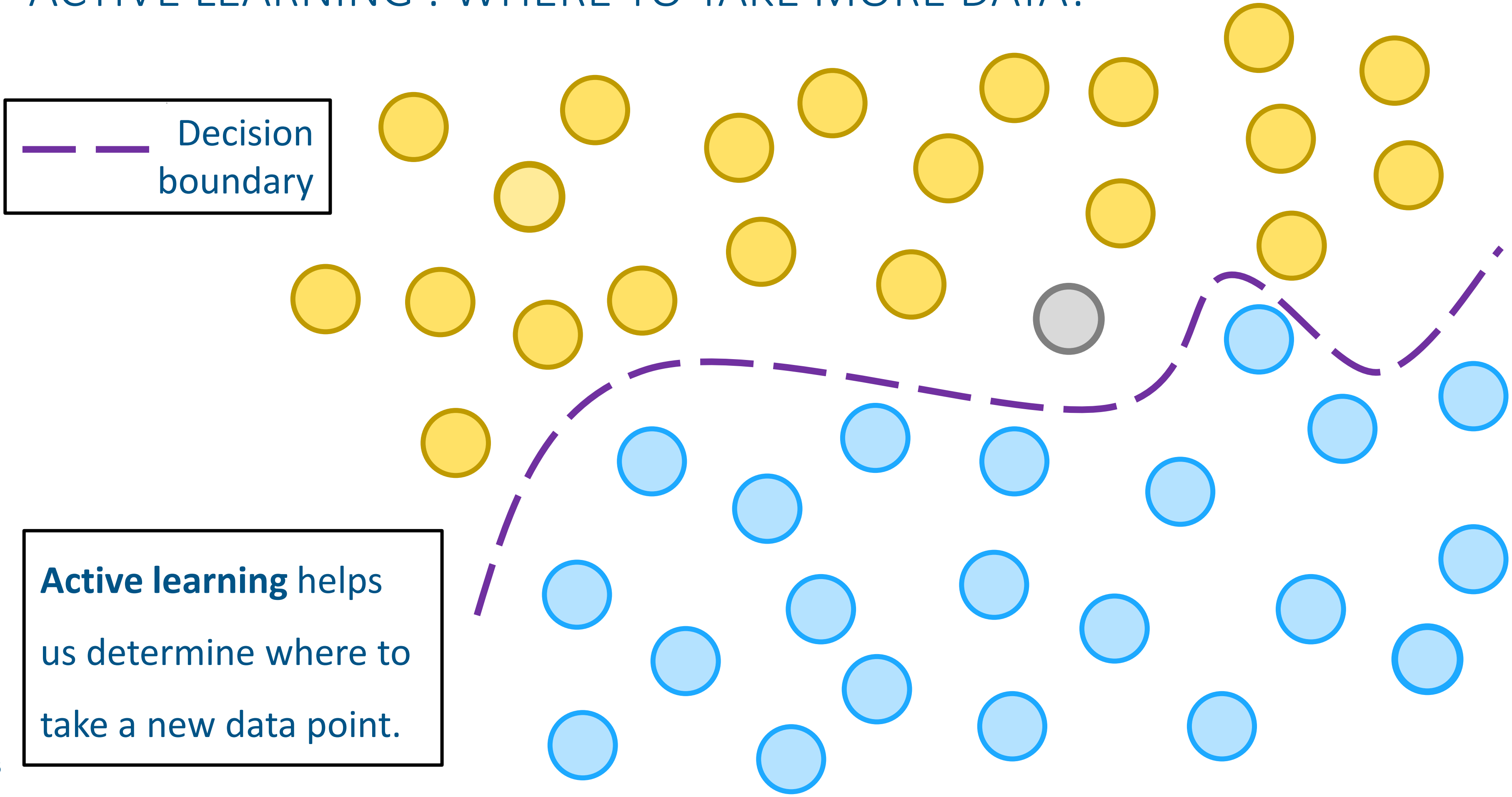

**Active learning** helps us determine where to take a new data point.

# ACTIVE LEARNING : WHERE TO TAKE MORE DATA?



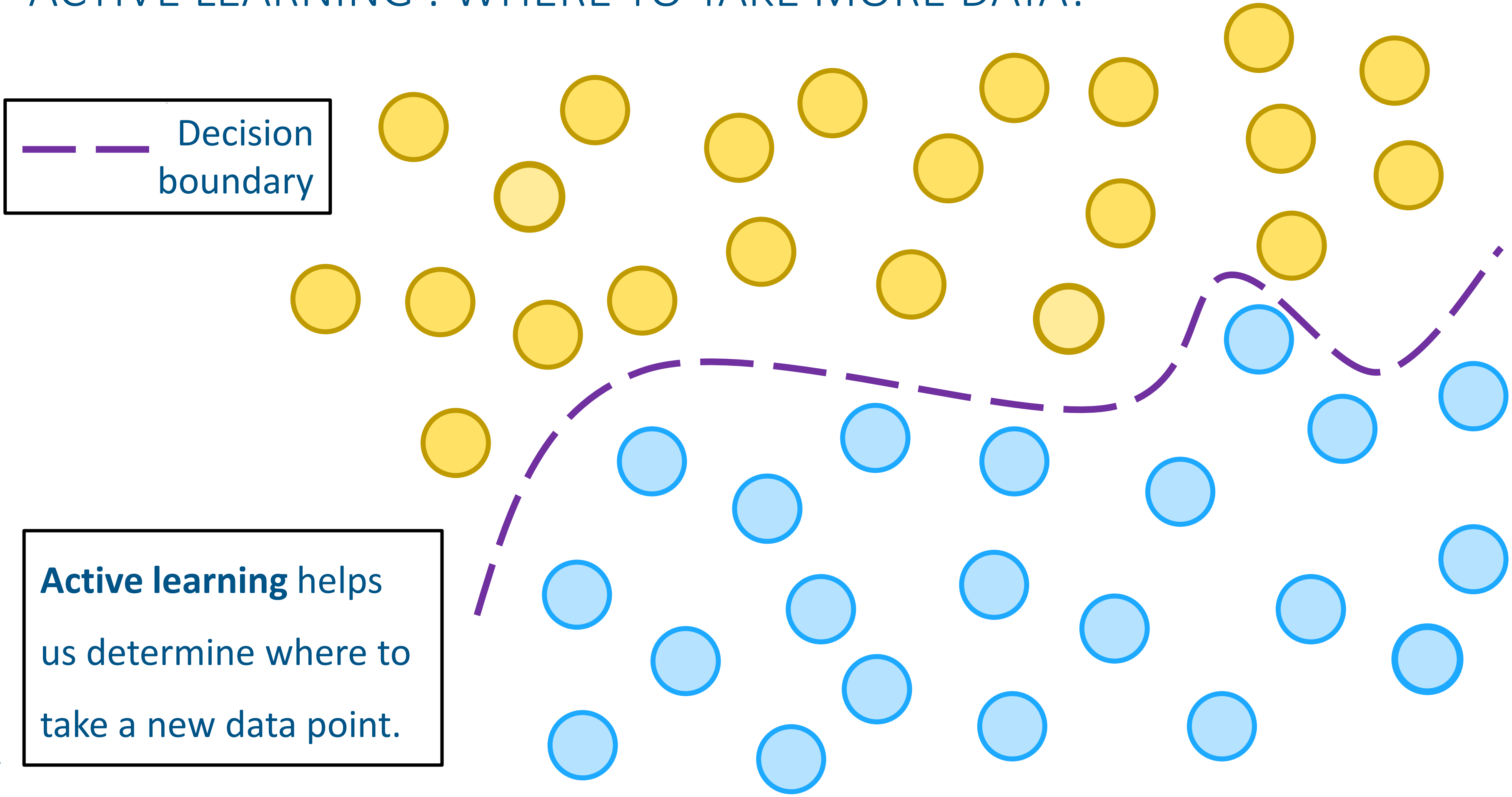

**Active learning** helps us determine where to take a new data point.

# ACTIVE LEARNING : WHERE TO TAKE MORE DATA?



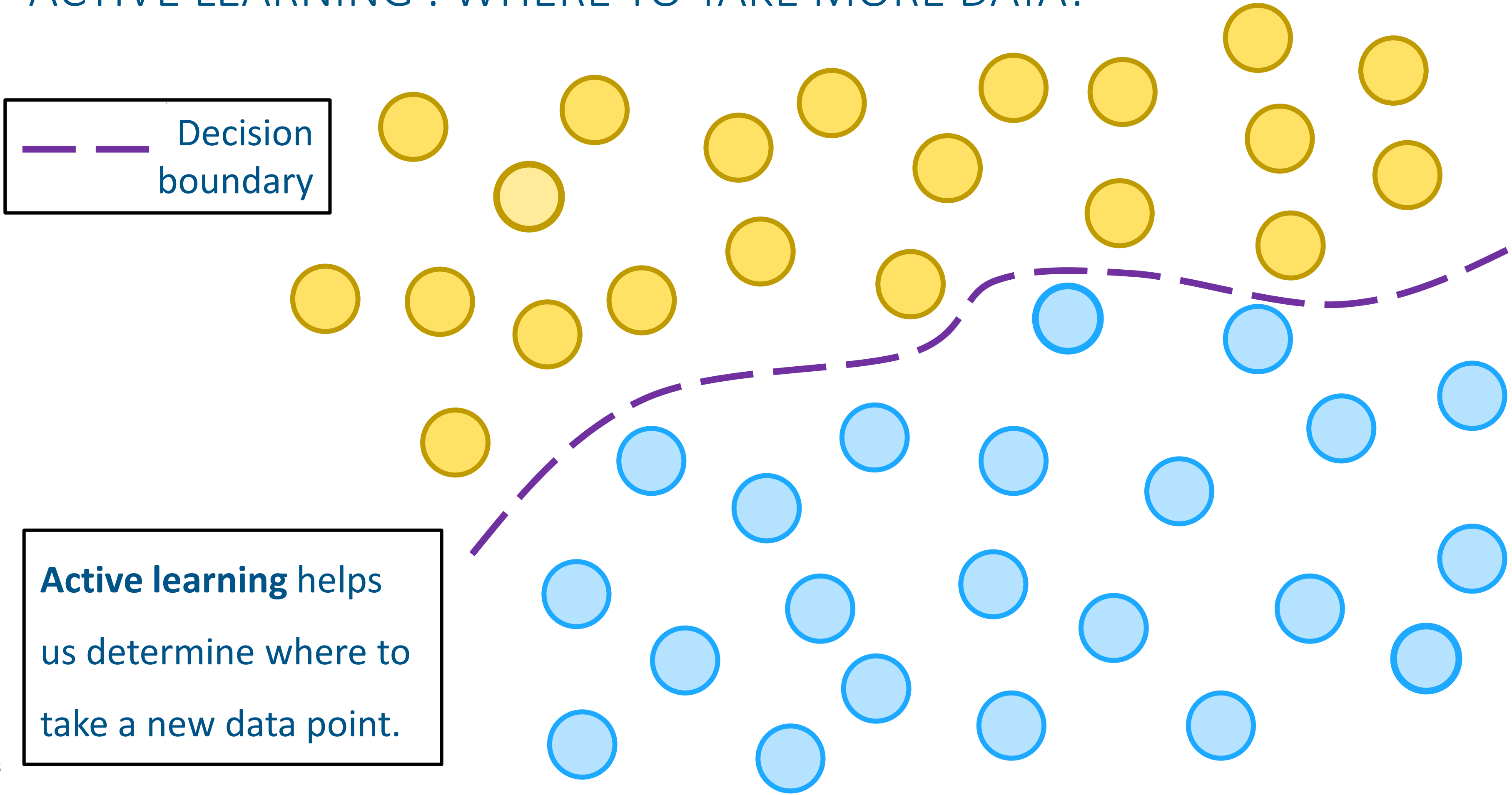

**Active learning** helps us determine where to take a new data point.

# ACTIVE LEARNING : WHERE TO TAKE MORE DATA?




**Active learning** helps us determine where to take a new data point.

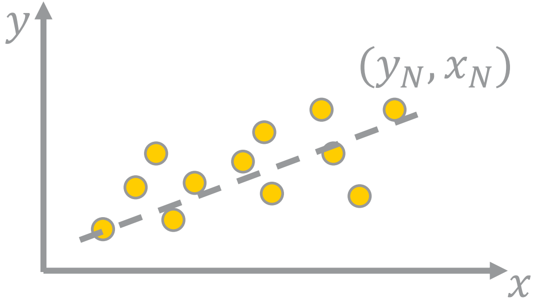
# ACTIVE LEARNING : WHERE TO TAKE MORE DATA?




**Active learning** helps us determine where to take a new data point.

# EXAMPLE: LINEAR REGRESSION

Machine Learning without the machines




$$\hat{\boldsymbol{y}} = w\boldsymbol{x}; \boldsymbol{y}, \boldsymbol{x} \in R^N$$

More generally,

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w} = \boldsymbol{x}_1 w_1 + \boldsymbol{x}_2 w_2 + \cdots + \boldsymbol{x}_M w_M$$

"Feature" matrix $\boldsymbol{X} \in R^{N\times M}$

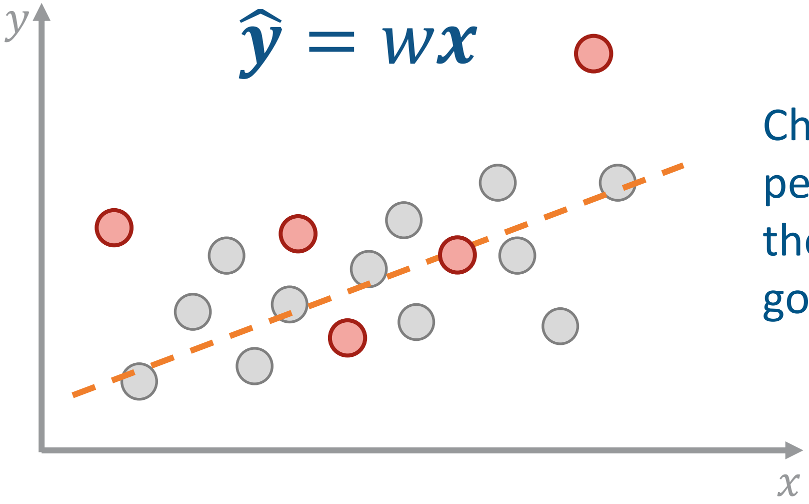
$M$ features

Find $\boldsymbol{w} \in R^M$ that minimizes error between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$. We choose the error. **Let's use $\boldsymbol{L_2}$.**

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^2$$

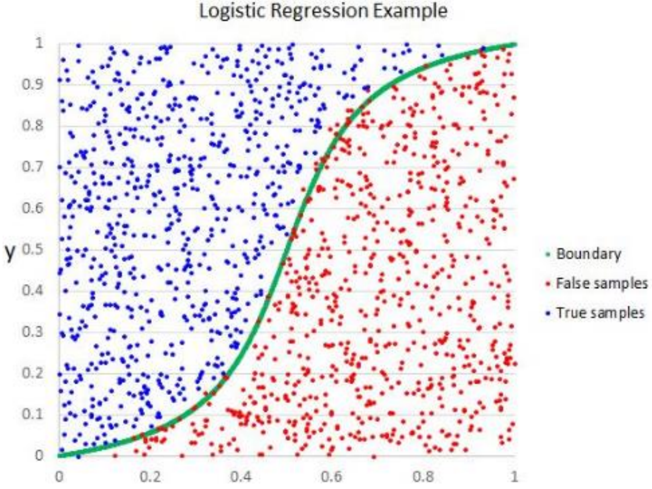
After some calculus we get

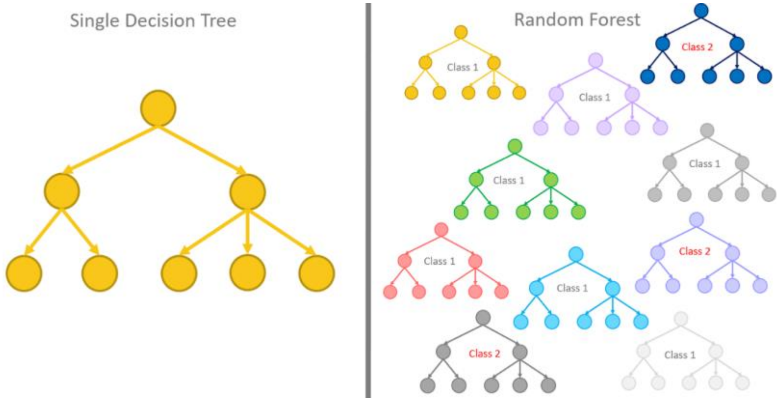
$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$




Check the performance on the test set. Any good?

# A FEW POPULAR ALGORITHMS




Deep Neural Network

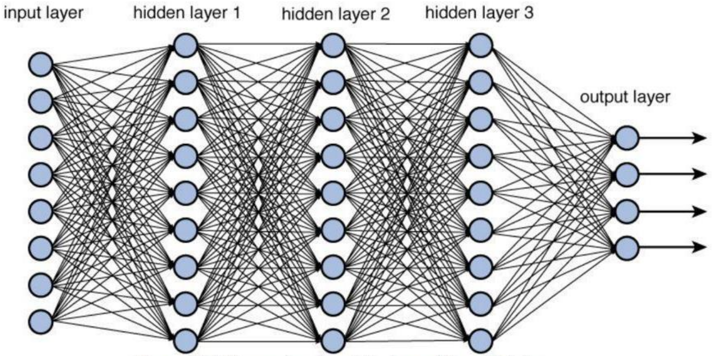
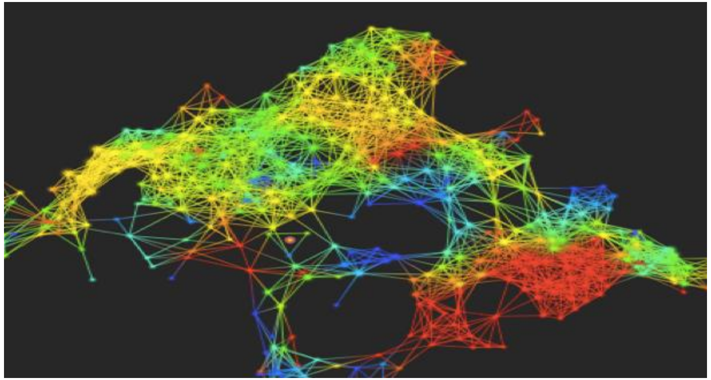



Figure 12.2 Deep network architecture with multiple layers.

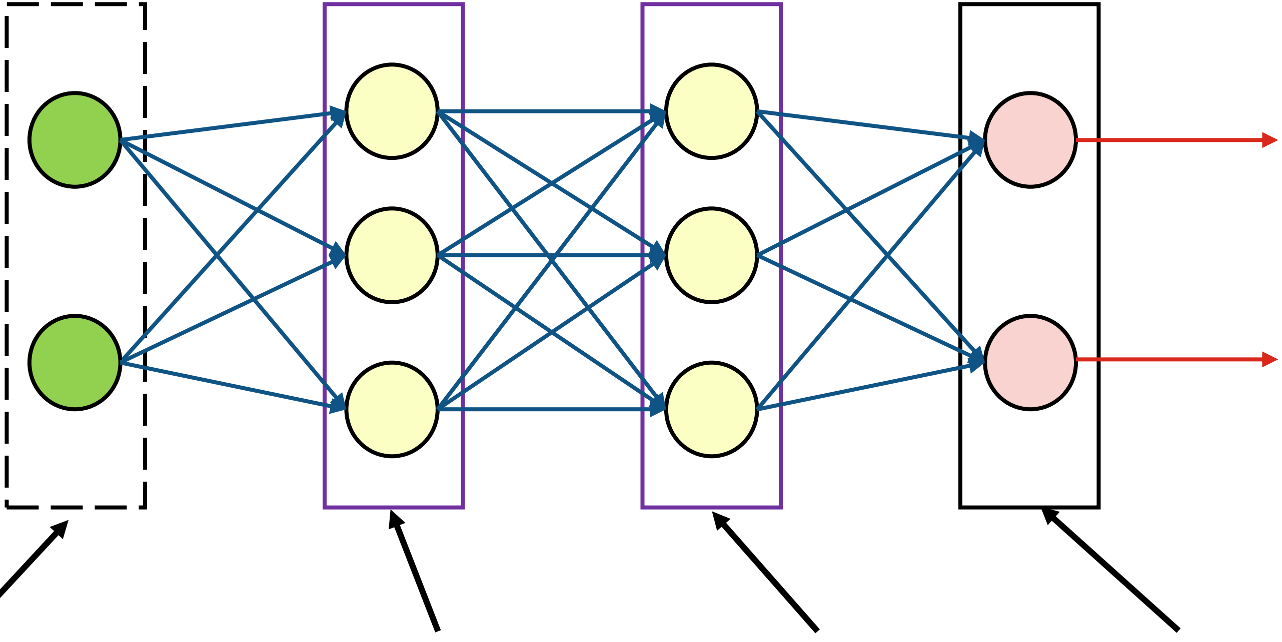


**Topological Data Analysis (TDA)**

# A CRASH COURSE ON NEURAL NETWORKS

# NEURAL NETWORKS: BASIC ANATOMY

Neural networks are universal function approximators.



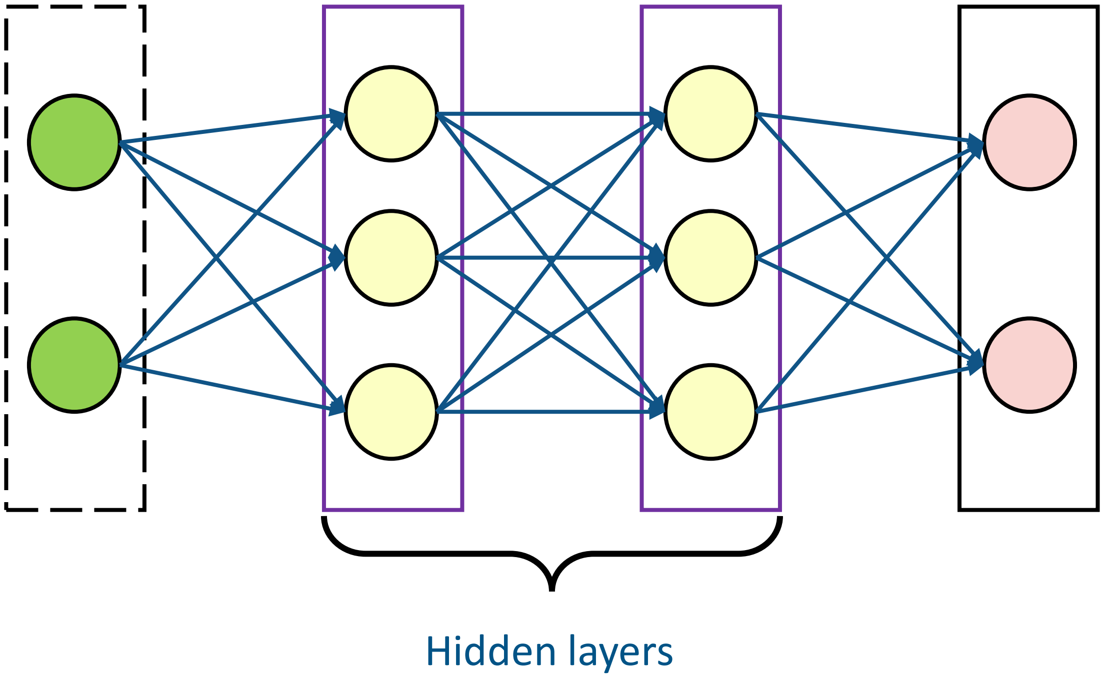
Input “layer”
Not much happens here.

Hidden layer 1

Hidden layer 2

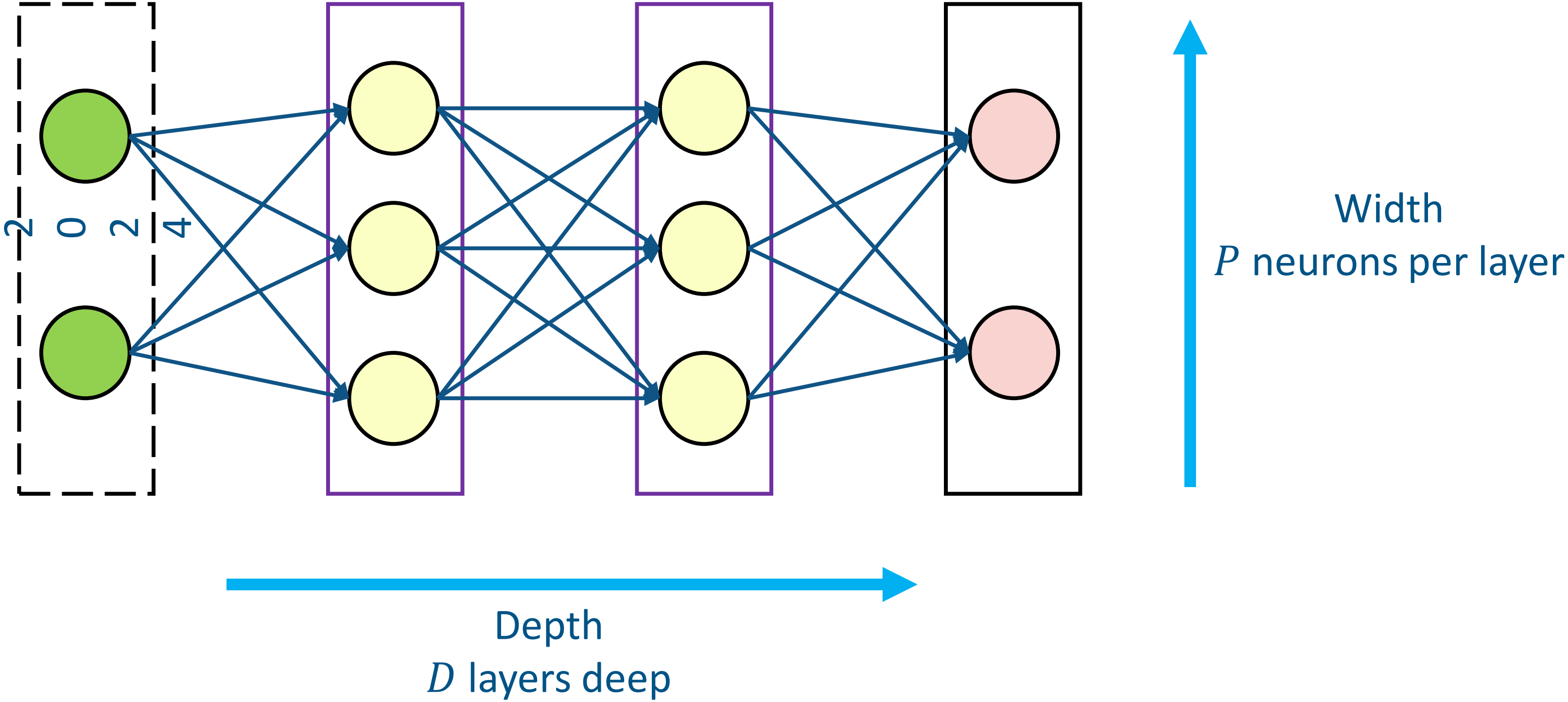
Output layer

# NEURAL NETWORKS: HIDDEN LAYERS

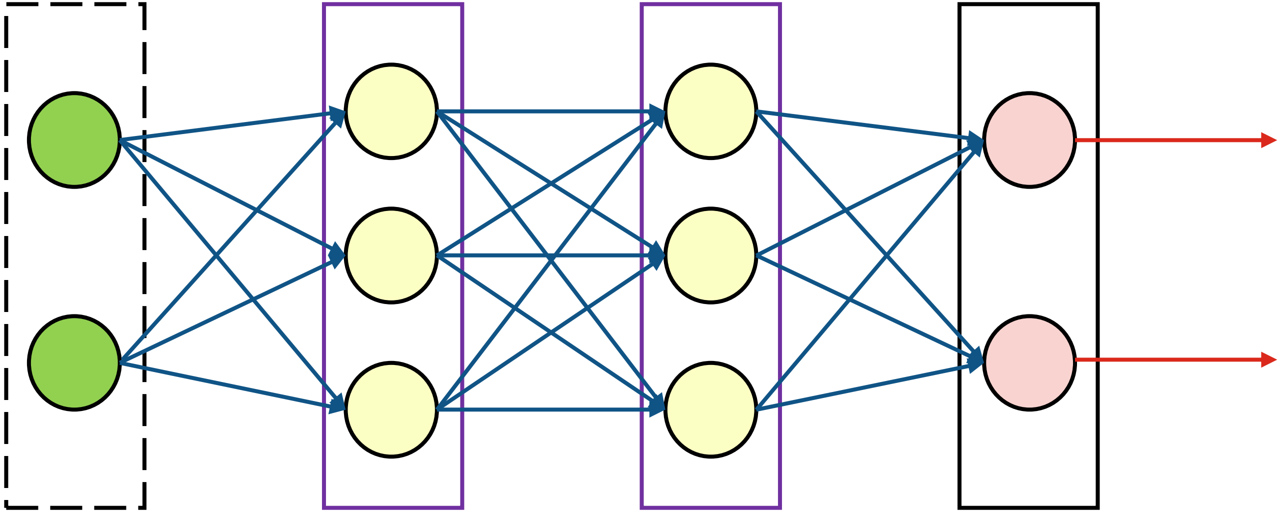



**Note**: More than 1 hidden layer means “deep”.

# NEURAL NETWORKS: DEPTH AND WIDTH



# NEURAL NETWORKS: BASIC CALCULATIONS



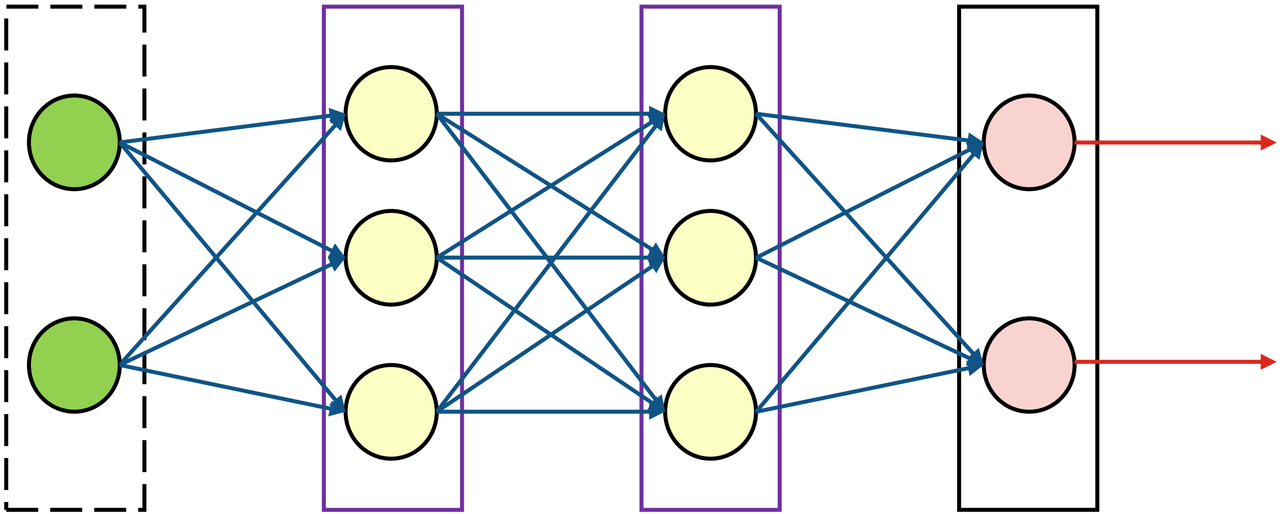
Input "layer"

$$\boldsymbol{h}_1 = \phi(\boldsymbol{w}_1 \cdot \boldsymbol{x} + \boldsymbol{b}_1)$$

$$\hat{\boldsymbol{y}} = \boldsymbol{w}_3 \cdot \boldsymbol{h}_2 + \boldsymbol{b}_3$$

$$\boldsymbol{h}_2 = \phi(\boldsymbol{w}_2 \cdot \boldsymbol{h}_1 + \boldsymbol{b}_2)$$

# NEURAL NETWORKS: COMPOSITION OF FUNCTIONS

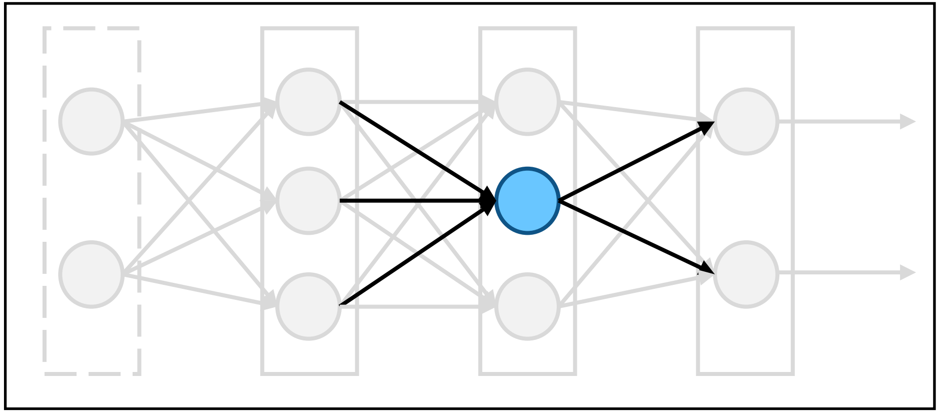


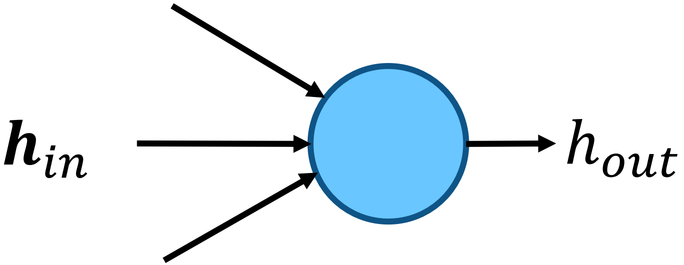
The output of a neural network is a composition of functions:

$$\hat{\boldsymbol{y}} = \boldsymbol{w}_3 \cdot \phi(\boldsymbol{w}_2 \cdot \phi(\boldsymbol{w}_1 \cdot \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2) + \boldsymbol{b}_3$$

This is reminiscent of early work by Kolmogorov on function approximation by composition of functions.

# NEURAL NETWORKS: OPENING UP A NEURON






$\boldsymbol{h}_{in} \in R^P$ $\quad$ $h_{out} \in R$

$$h_{out} = \phi(\boldsymbol{w} \cdot \boldsymbol{h}_{in} + b)$$

Activation function

Weights (need to find)

Bias (need to find)

<u>Examples of Activation Functions</u>

$$\underbrace{\phi(z) = \frac{1}{1+e^{-z}}}$$

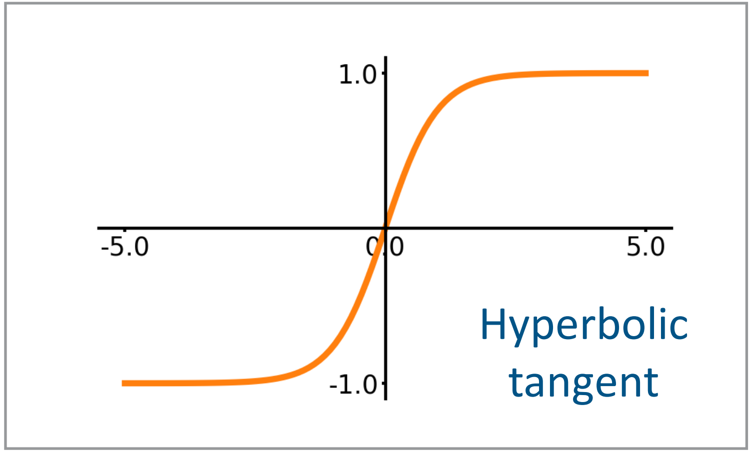
Sigmoid

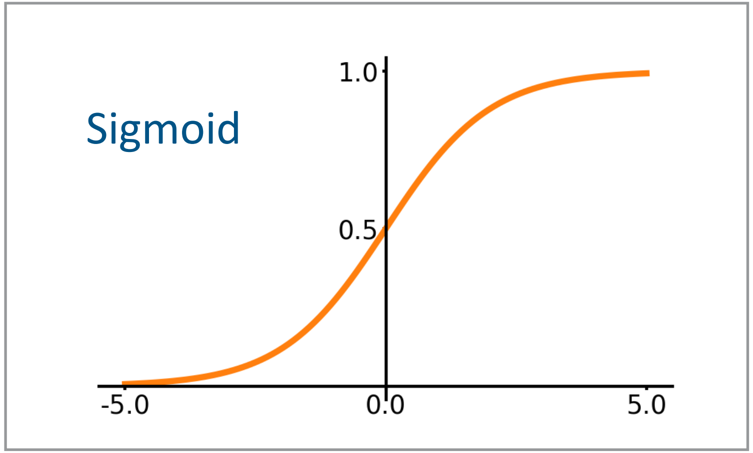
$$\underbrace{\phi(z) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases}}$$

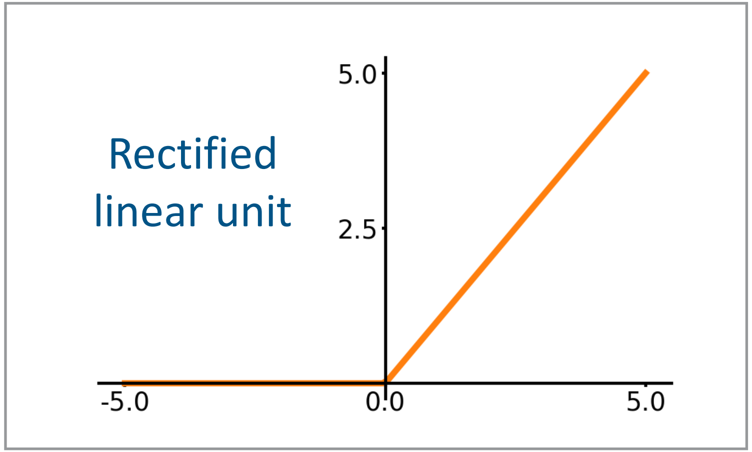
Rectified linear unit

$\phi(z) = \tanh(z)$ Hyperbolic tangent

# NEURAL NETWORKS: EXAMPLES OF ACTIVATION FUNCTIONS

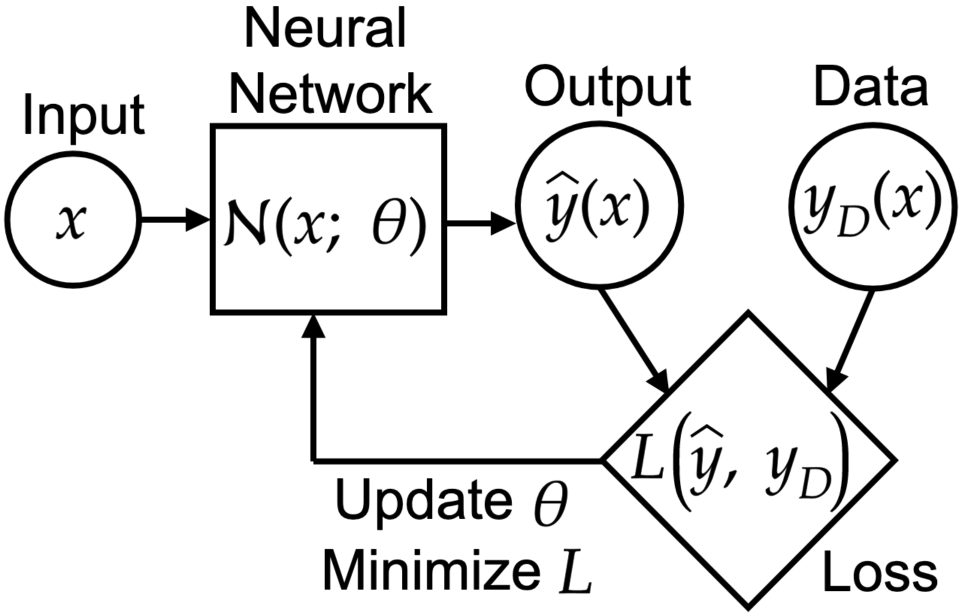



- Called “activation” for historical reasons
  - Back to early work on activating neurons
  - Also reason for the name “neural network”
- Without activation, the NN just gives a (complicated) linear regression
- Best activation is problem dependent

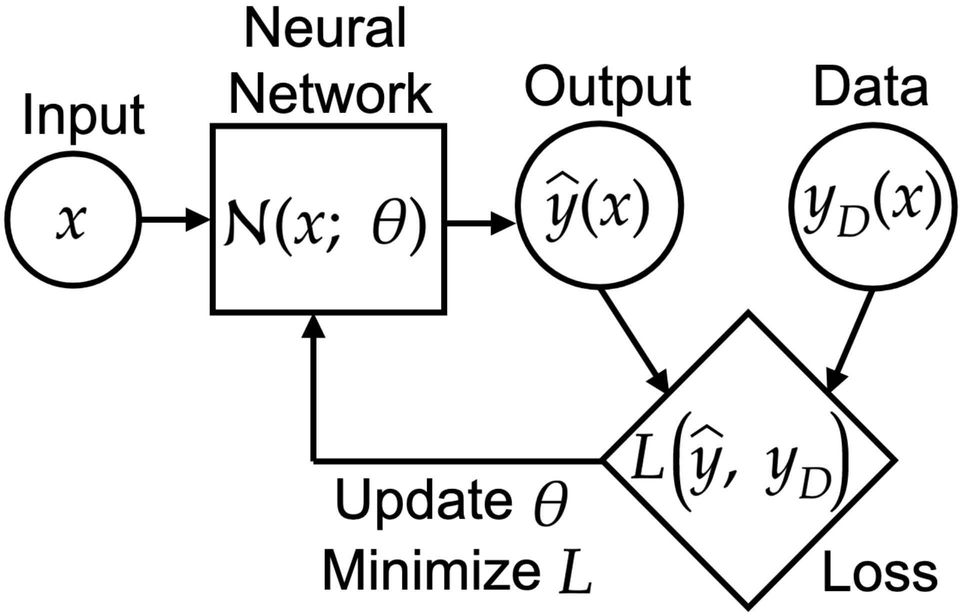
# NEURAL NETWORKS: BASIC PIPELINE




Basically the same as the usual ML pipeline.

1. Split data into train and test sets
2. Split train into train and validation
3. Optimize NN parameters on training set
   - Use stochastic gradient descent
   - Use **automatic differentiation** to take derivatives
4. Monitor performance on validation set
5. Assess performance on test set

# NEURAL NETWORKS: BASIC PIPELINE




Basically the same as the usual ML pipeline.

**Algorithm:** Training a NN

**Data:** $\{(y_i, x_i)\}_{\text{train}}$, $\{(y_i, x_i)\}_{\text{val}}$

**Result:** $\mathcal{N}(x; \theta)$

initialize $\theta$, $tol$, $\lambda$

**while** $\ell > tol$ **do**

  **for** $x \in X_{train}$ **do**

    $\widehat{y} \leftarrow \mathcal{N}(x; \theta)$

    $\ell \mathrel{+}= L(\widehat{y}, y_{\text{train}})$

  **end**

  $$\theta \leftarrow \theta - \lambda \frac{\partial \ell}{\partial \theta}$$

  **for** $x \in X_{val}$ **do**

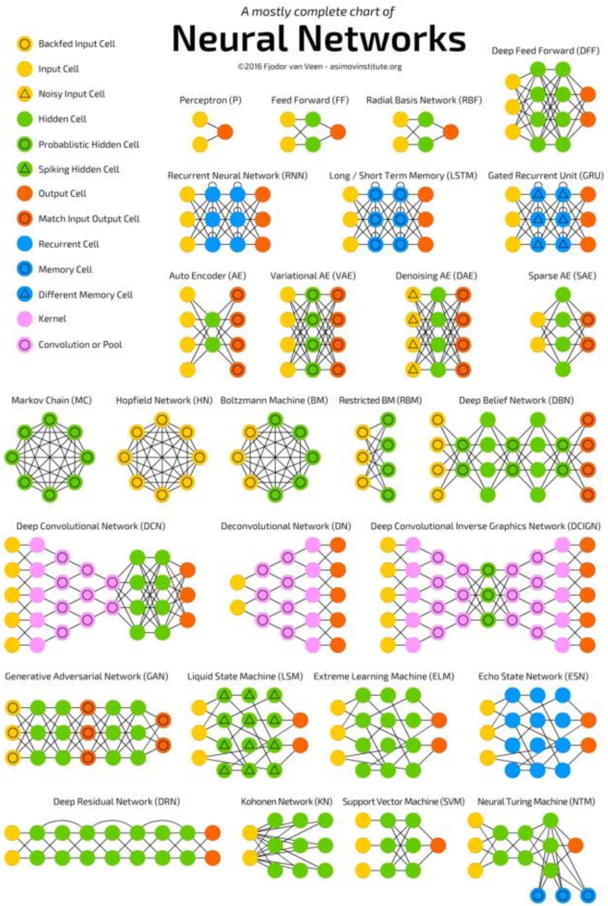
    $\widehat{y} \leftarrow \mathcal{N}(x; \theta)$

    $\ell_{\text{val}} \mathrel{+}= L(\widehat{y}, y_{\text{val}})$

  **end**

**end**

# NEURAL NETWORKS: A ZOO OF ARCHITECTURES



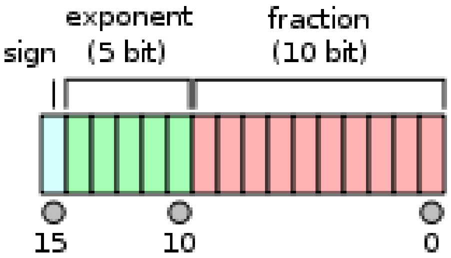

- Fully-connected (the “original”)
- Convolutional (wildly successful)
- Autoencoders (self-supervised and very cool)
  - Variational autoencoders
  - Convolutional autoencoders
- Recurrent (good for sequential data)
- Echo state / reservoir
- Graph networks
- U-nets
- And many, many more

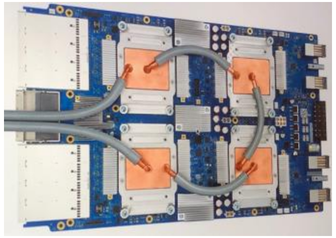


https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

# A FEW TAKEAWAYS

- Origins in neuroscience research
  - Can now think of NNs as mathematical and algorithmic tools
- Many different types of “unit” architectures
- Nowadays these units are being used to build up more sophisticated networks
- Can be challenging to train NNs b/c they are so data-hungry
  - Use GPUs (and others) to speed things up
  - Use less precision (even half precision!)



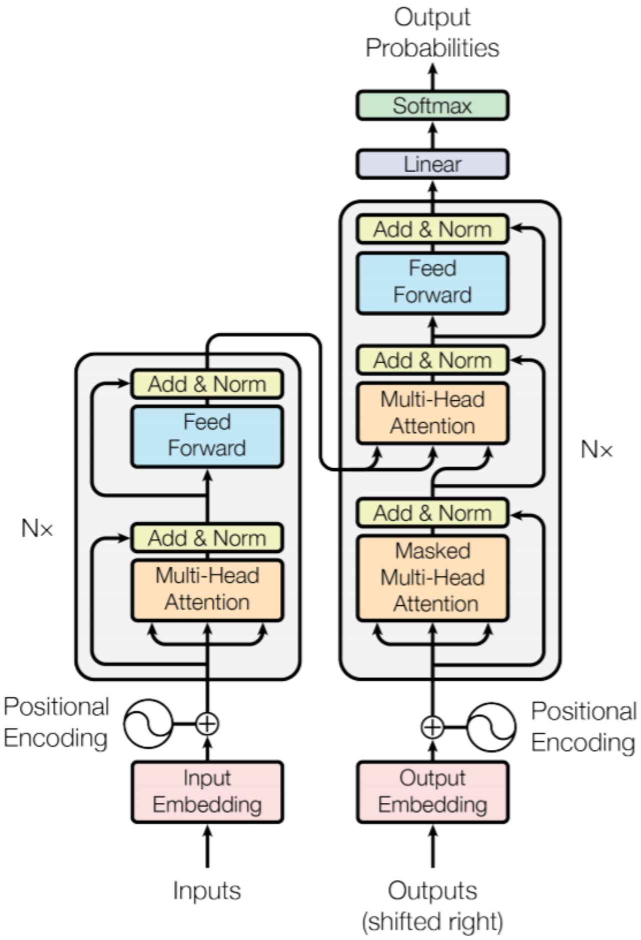



Figure 1: The Transformer - model architecture.

# APPLICATIONS WITH AN EMPHASIS ON FLUIDS

# APPLICATIONS IN PHYSICS AND FLUIDS

- Crucial to embed physics → Really non-negotiable
- The landscape is enormous and rapidly developing.

**An Incomplete List of Current Work**

Physics Informed Neural Networks
- Marios' talk

Fourier networks

Graph Neural Networks

Inverse problems

Generative models
- Generative adversarial NNs

Closure models

Superresolution

Devising new numerical methods

SINDy – Can be used to develop reduced models
- Sparse identification of nonlinear dynamics

Transfer learning
- An ML continuation strategy

Transfer learning

Reinforcement learning

# APPLICATIONS IN PHYSICS AND FLUIDS

- Crucial to embed physics → Really non-negotiable
- The landscape is enormous and rapidly developing.

**An Incomplete List of Current Work**

Physics Informed Neural Networks

- Marios' talk

Fourier networks

Graph Neural Networks

Inverse problems

Generative models

- Generative adversarial NNs

**Closure models**

**Superresolution**

Devising new numerical methods

SINDy

- Sparse identification of nonlinear dynamics

Transfer learning

- An ML continuation strategy

Transfer learning

Reinforcement learning

# APPLICATIONS IN PHYSICS AND FLUIDS

- Crucial to embed physics → Really non-negotiable
- The landscape is enormous and rapidly developing.

**An Incomplete List of Current Work**

Physics Informed Neural Networks

- Marios' talk

Fourier networks

Graph Neural Networks

Inverse problems

Generative models

- Generative adversarial NNs

**Closure models**

**Superresolution**

Devising new numerical methods

SINDy

- Sparse identification of nonlinear dynamics

Transfer learning

- An ML continuation strategy

Transfer learning

Reinforcement learning

# EXAMPLES

# EXAMPLE 1: CLOSURE MODELS

The Reynolds-averaged momentum equation for an incompressible fluid is

$$\rho\frac{\partial\overline{\boldsymbol{u}}}{\partial t}+\rho\nabla\cdot(\overline{\boldsymbol{u}}\otimes\overline{\boldsymbol{u}})=\nabla\cdot\left[-\overline{P}\boldsymbol{I}+\mu\left(\nabla\overline{\boldsymbol{u}}+\nabla\overline{\boldsymbol{u}}^{T}\right)-\boxed{\overline{\rho\boldsymbol{u}'\otimes\boldsymbol{u}'}}\right].$$

The Boussinesq approximation for the anisotropic Reynolds stress tensor is

$$\boldsymbol{a}=\overline{\boldsymbol{u}'\otimes\boldsymbol{u}'}-\frac{2}{3}k\boldsymbol{I}\approx-\nu_T\left(\nabla\overline{\boldsymbol{u}}+\nabla\overline{\boldsymbol{u}}^{T}\right),\qquad k=\frac{1}{2}\left(\overline{\boldsymbol{u}'\cdot\boldsymbol{u}'}\right).$$

This very convenient closure results in

$$\frac{\partial\overline{\boldsymbol{u}}}{\partial t}+\nabla\cdot(\overline{\boldsymbol{u}}\otimes\overline{\boldsymbol{u}})=\nabla\cdot\left[-\frac{1}{\rho}\left(\overline{P}+\frac{2}{3}\rho k\right)\boldsymbol{I}+(\nu+\nu_T)\left(\nabla\overline{\boldsymbol{u}}+\nabla\overline{\boldsymbol{u}}^{T}\right)\right].$$

**But what is $\boldsymbol{\nu_T}$?**

A tremendous number of models have been proposed over the years ($k-\epsilon$, $k-\omega$, SA, renormalization, …)

# A FIRST APPROACH WITH MACHINE LEARNING

- Neural nets are really good at approximating functions, so let's approximate $\boldsymbol{a}$.
- We naively collect velocity and pressure data from experiments (or DNS) and throw it into a neural network.

$$\boldsymbol{u}, P \longrightarrow \boxed{\text{NN}} \longrightarrow \boldsymbol{a} \qquad \Rightarrow \boldsymbol{a} = \boldsymbol{a}(\boldsymbol{u}, P)$$

**Terrible idea!** Doesn't even enforce Galilean invariance.

- A better approach to embed Galilean invariance:

$$\nabla \boldsymbol{u} \longrightarrow \boxed{\text{NN}} \longrightarrow \boldsymbol{a} \qquad \Rightarrow \boldsymbol{a} = \boldsymbol{a}(\nabla \boldsymbol{u})$$

**Better.** Now Galilean invariant.

- But even more can be done.
  - Want it to be more interpretable
  - Want it to be even more physical
  - Easy to make $\boldsymbol{a}$ symmetric.
  - What else?

# TENSOR BASIS NEURAL NETWORK (LING ET AL, JFM, 2016)

- The most general effective eddy viscosity model (for incompressible fluids) is (Pope, 1975)

$$\boldsymbol{b} = \sum_{n=1}^{10} g^{(n)} \underbrace{(\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5)}_{\text{Invariants of } \boldsymbol{a}.} \boldsymbol{T}^{(n)}(\nabla \mathbf{u})$$

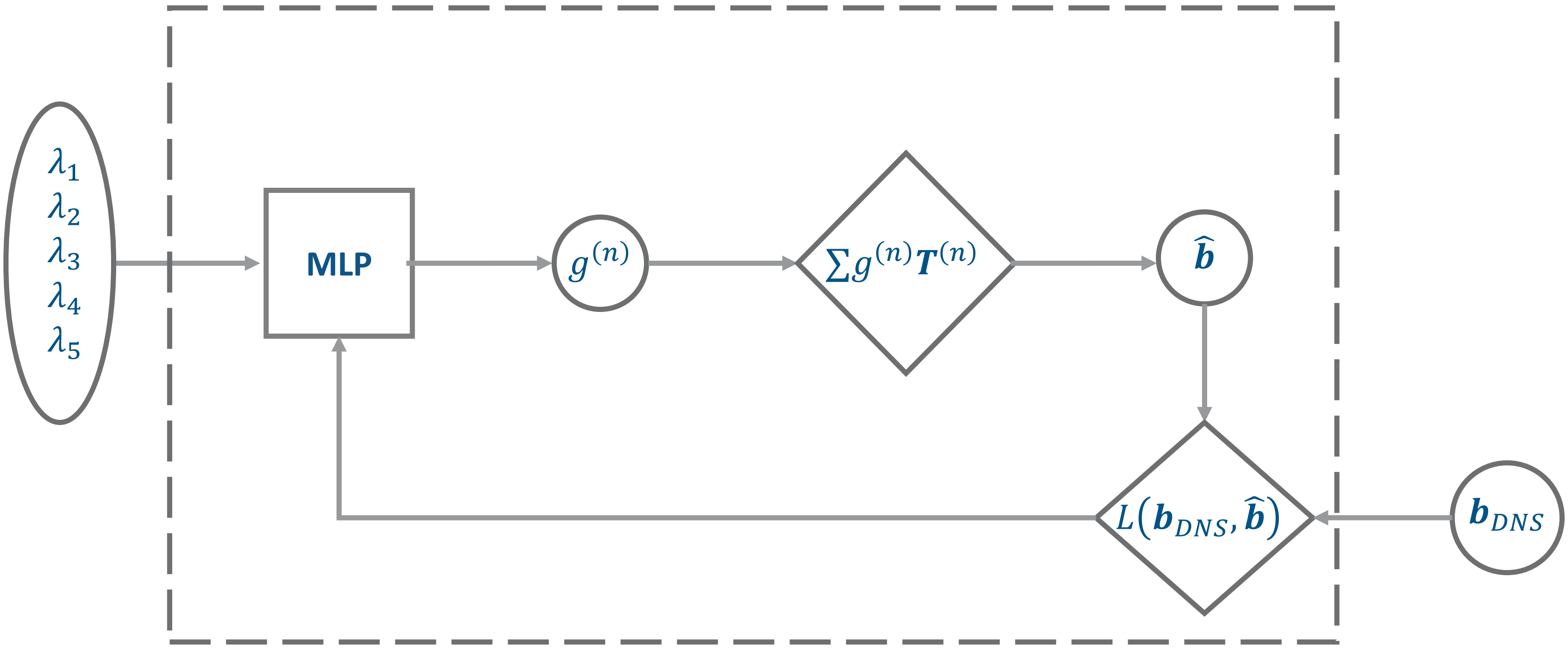
Normalized anisotropy tensor

Unknown coefficients

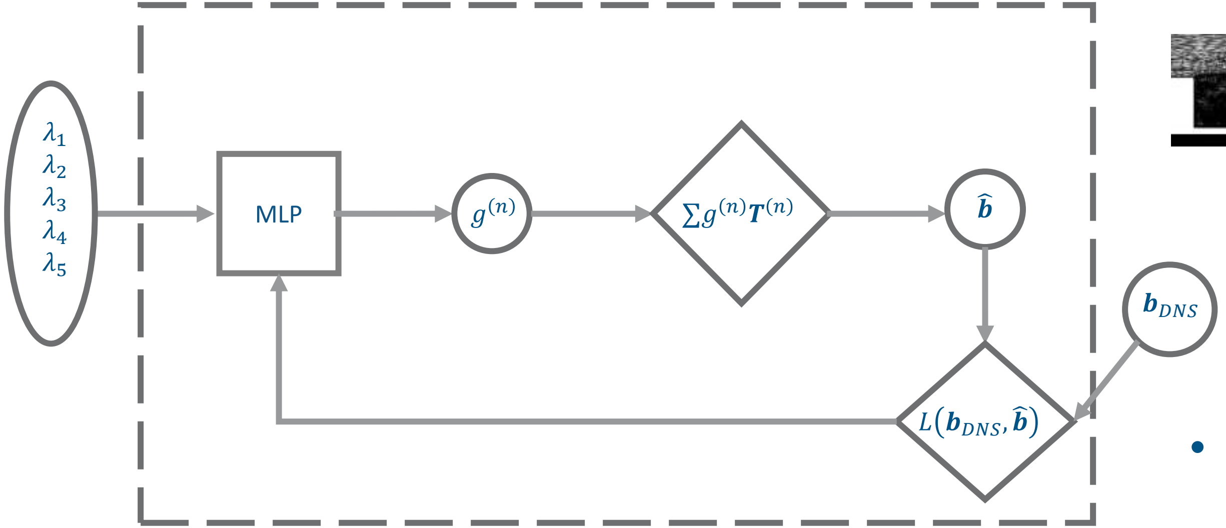
Tensor basis

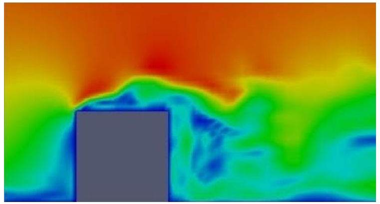
- The invariants $\{\lambda_j\}, j = 1, \ldots 5$ are known functions of $\nabla \boldsymbol{u}$. These will be the inputs.
- The form of $\boldsymbol{T}^{(n)}$, also known: $\boldsymbol{T}^{(1)} = \boldsymbol{S}, \boldsymbol{T}^{(2)} = \boldsymbol{S\Omega} - \boldsymbol{\Omega S}, \boldsymbol{T}^{(8)} = \boldsymbol{S\Omega S}^2 - \boldsymbol{S}^2\boldsymbol{\Omega S}$
- $g^{(n)}$ is a complicated function
  - Learn it with a NN
- Note: Keeping one term gives $\boldsymbol{a} = g^{(1)}\boldsymbol{S}$.
  - Can be used as a sanity check of the NN --- $g^{(1)}$ should be the coefficient of the linear term.
  - $\Rightarrow$ Some interpretability!

# TENSOR BASIS NEURAL NETWORK (LING ET AL, JFM, 2016)



# TENSOR BASIS NEURAL NETWORK (LING ET AL, JFM, 2016)



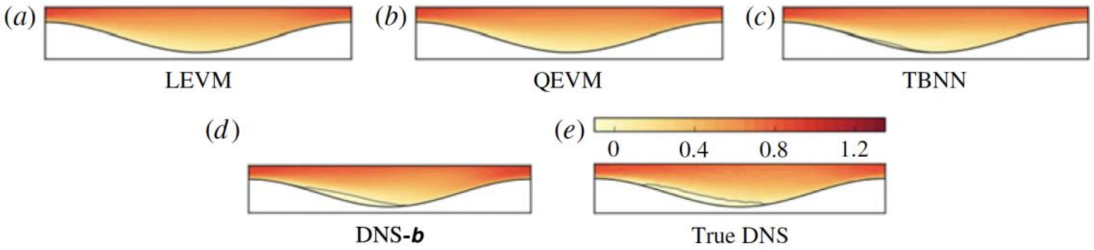



FIGURE 5. Contours of streamwise velocity normalized by bulk velocity in the wavy wall test case, zoomed into the near-wall region. Separated regions outlined in grey.

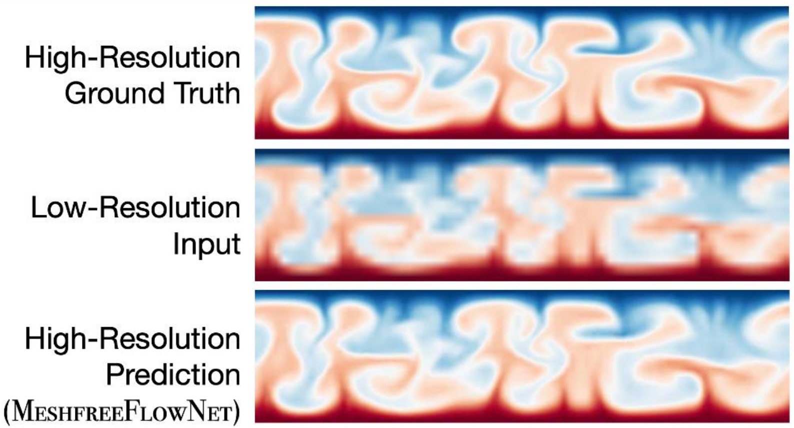
- Promising performance, but improvements possible.
- Unlikely to perform well on $Re$ much higher than training.
- Needs a lot of DNS data.

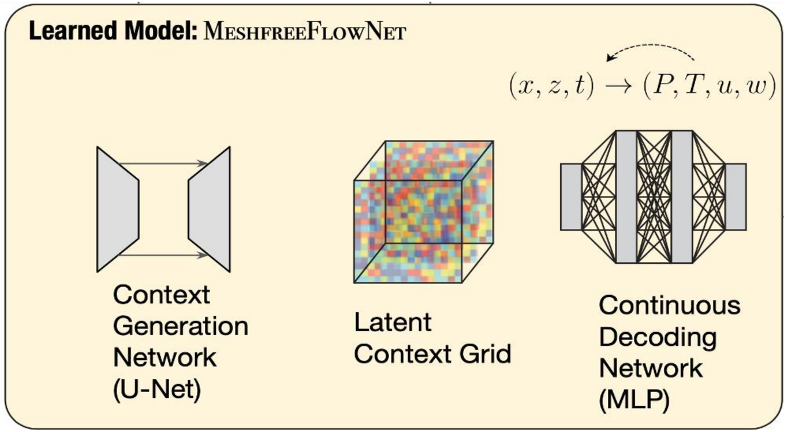
# TENSOR BASIS NEURAL NETWORK (LING ET AL, JFM, 2016)

- So how would this work in real life?
- Gather as much data as possible, covering as many flow regimes as possible.
- Design the MLP
  - # of layers, # of nodes, initial weights, activation function, appropriate loss, select optimizer
- Split data into training, validation, and test sets and train the network on some GPUs
- Once the network is trained, it can be used in a RANS simulation
  - Run PDE solver
  - Evaluate NN by passing in RANS fields as input
  - NN is a fully differentiable function, so feel free to take derivatives if necessary
- Can update NN offline as more data becomes available for training
  - The training will be faster b/c the NN only needs to be updated, not trained from scratch.

# A FEW COMMENTS ON SUPERRESOLUTION

- Is it possible to produce a physically realistic solution from a low-resolution snapshot?
- Substantial work in this area for images as well as physics problems.
- One recent example is MeshfreeFlowNet.
  - Performed well on Rayleigh-Benard convection for modest Rayleigh numbers ($Ra \leq 10^8$)
  - Combines a 3D U-Net and a MLP
  - Shown to scale up to 128 GPUs
  - Performed well on different initial and boundary conditions.



# CHALLENGES AND POSSIBILITIES

# ML & PHYSICS: TAKEAWAYS AND CHALLENGES

- Access to data
  - ▲ Many algorithms are data hungry
  - ■ How to get data?
  - ■ Where to store it?
- Extrapolation and generalization
  - ▲ Pretty good at interpolating to different flow regimes, geometries, etc.
  - ▼ Risky to extrapolate
- Cost of training
  - ▼ Requires substantial computational resources
  - ▲ Usually very easy to evaluate after a model is trained
- New approaches for blending algorithm development with data
  - ▲ Learn closure models and reduced models from data
  - ▲ Generate high resolution images from low resolution snapshots
  - ▲ New strategies for working with data (e.g. active learning and transfer learning)

Crucial to blend physics into the data.

$$F[\boldsymbol{u}] - \boldsymbol{f} = \boldsymbol{0}$$

# REFERENCES

- Too many papers to list here and it keeps growing!


  - Here is a quasi-recent summary as of 2019: https://github.com/gdportwood/Turb_ML_Papers/blob/master/NeurIPS2019.md.
  - Brenner, M.P., Eldredge, J.D. and Freund, J.B., 2019. Perspective on machine learning for advancing fluid mechanics. Physical Review Fluids, 4(10), p.100501.
  - Fonda, E., Pandey, A., Schumacher, J. and Sreenivasan, K.R., 2019. Deep learning in turbulent convection networks. Proceedings of the National Academy of Sciences, 116(18), pp.8667-8672.
  - Kops, S.M., Saunders, D.J., Rietman, E.A. and Portwood, G.D., 2019. Unsupervised Machine Learning to Teach Fluid Dynamicists to Think in 15 Dimensions. arXiv preprint arXiv:1907.10035.
  - Alber, M., Tepole, A.B., Cannon, W.R., De, S., Dura-Bernal, S., Garikipati, K., Karniadakis, G., Lytton, W.W., Perdikaris, P., Petzold, L. and Kuhl, E., 2019. Integrating machine learning and multiscale modeling perspectives, challenges, and opportunities in the biological, biomedical, and behavioral sciences. npj Digital Medicine, 2(1), pp.1-11.
  - Raissi, M., Perdikaris, P. and Karniadakis, G.E., 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial di erential equations. Journal of Computational Physics, 378, pp.686-707.
  - Ling, J., Kurzawski, A. and Templeton, J. (2016) Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, Journal of Fluid Mechanics. Cambridge University Press, 807, pp. 155166. doi: 10.1017/jfm.2016.615.
  - Duraisamy, K., Iaccarino, G., and Xiao, H., Turbulence Modeling in the Age of Data," Annual Review of Fluid Mechanics, 2019.
  - J.-L. Wu, H. Xiao, and E. G. Paterson., Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework, Physical Review Fluids, 073602, 2018,

IF WE ™