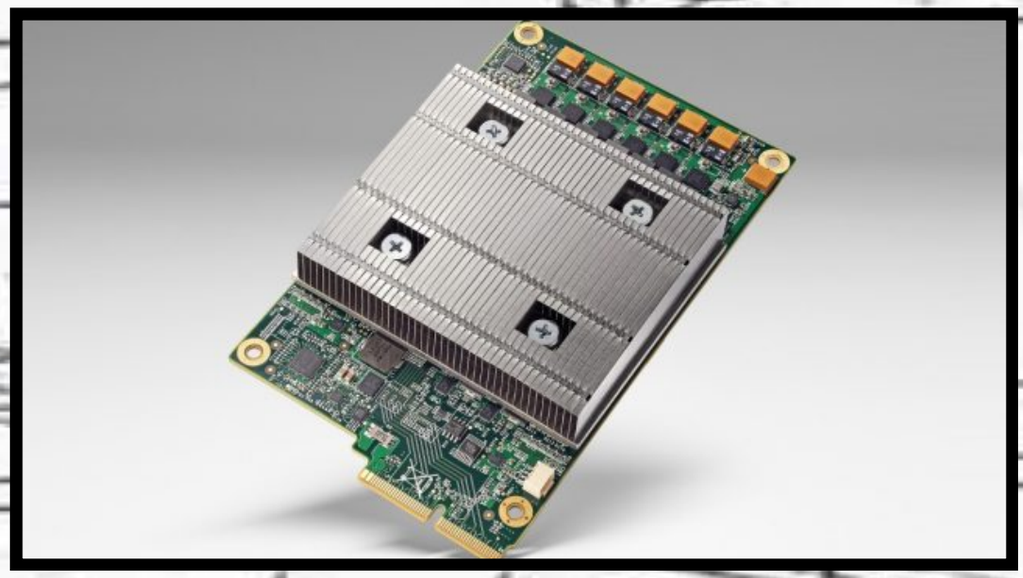
Introduction to Machine Learning in Particle Physics



James “Jamie” Gainer

March 28, 2018

slides will be available at: https://github.com/JamieGainer

Goals and Outline

1. Self-contained introduction to machine learning for folks in astro/particle physics

2. Applications to physics, focusing on particle physics

- More on machine learning than physics applications

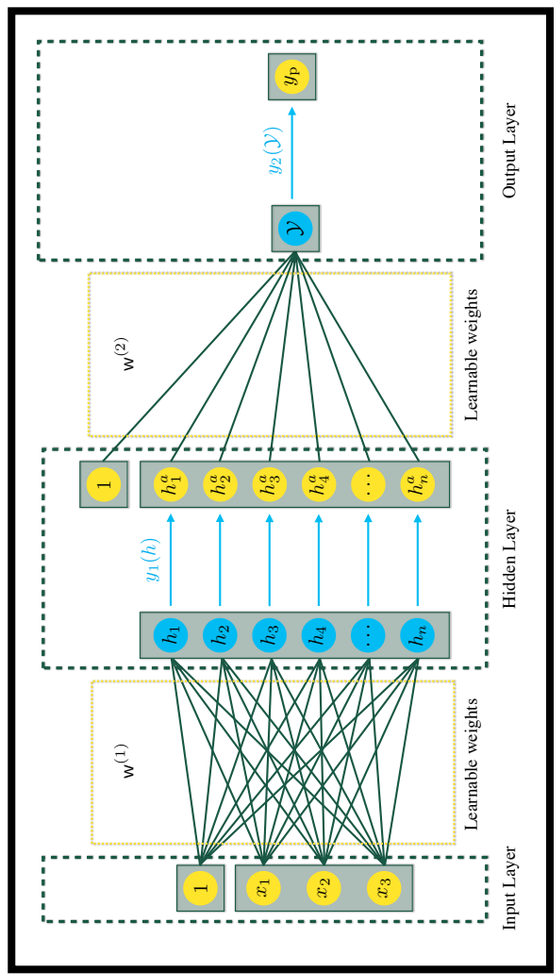
- Outline

    1. What is Machine Learning

    2. Short History of Machine Learning

    3. Machine Learning Basics/ Introduction to Main Algorithms

    4. Machine Learning in Physics

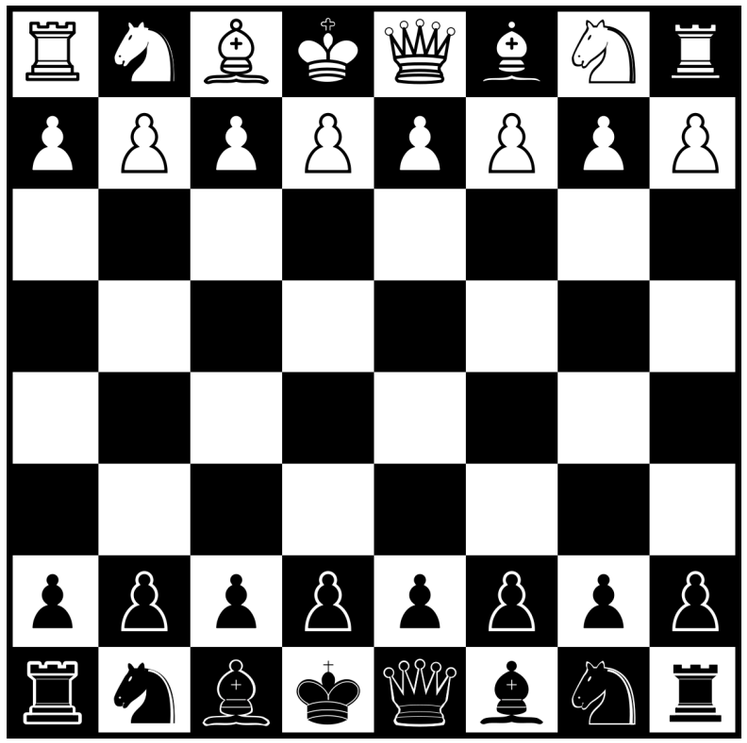


What is Machine Learning?

- **Machine Learning** (ML) is an approach to **Artificial Intelligence** (AI) where the algorithms learn from the data
- **Supervised**: algorithms "trained" from data where we know "the right answer".
  - (Astro-)particle physics: training on samples of Monte Carlo simulated signal and background events
  - Uses knowledge of which MC events are signal and which are background
- **Unsupervised**: no known right answers. Algorithms explore the data to look for patterns.
- Is this definition trivial?
- Machine Learning defined partly by being different than another approach to AI: **expert systems**.

What are Expert Systems?

- **Expert systems** are the result of an approach to AI also called "**symbolic AI**" or "**good old-fashioned AI**"
  - "symbolic" because based on rules/ equations (manipulation of symbols with assigned meanings)
  - "good old-fashioned" because dominant approach in 1970's - 1990's
- Figure out how to solve problems and tell the computer
- Example: **Chess**



Expert System Example: Chess

- Chess from an expert system

1. Select an “opening”, a sequence of early moves and responses.

2. Tell the computer how to rank moves and positions.

  - 

  - good to control center
  - good to protect king

- All of these are human-discovered rules fed to the computer

3. Use computing power to look through many positions, as many moves ahead as possible.

Deep Blue vs. Kasparov

- Deep Blue: chess-playing supercomputer built by IBM
- Defeated world champion Gary Kasparov 3.5-2.5 in 1997

- Deep Blue was an expert system
  - Evaluated ~20 million positions per move (~7-8 moves ahead)
  - Using program which codified wisdom of human chess players

Go

- We talked about chess: what about **Go**?

- Simpler game in terms of rules, much harder to solve with a computer:

- 19 x 19 board: hundreds of possible moves per turn versus ~10-20 for chess

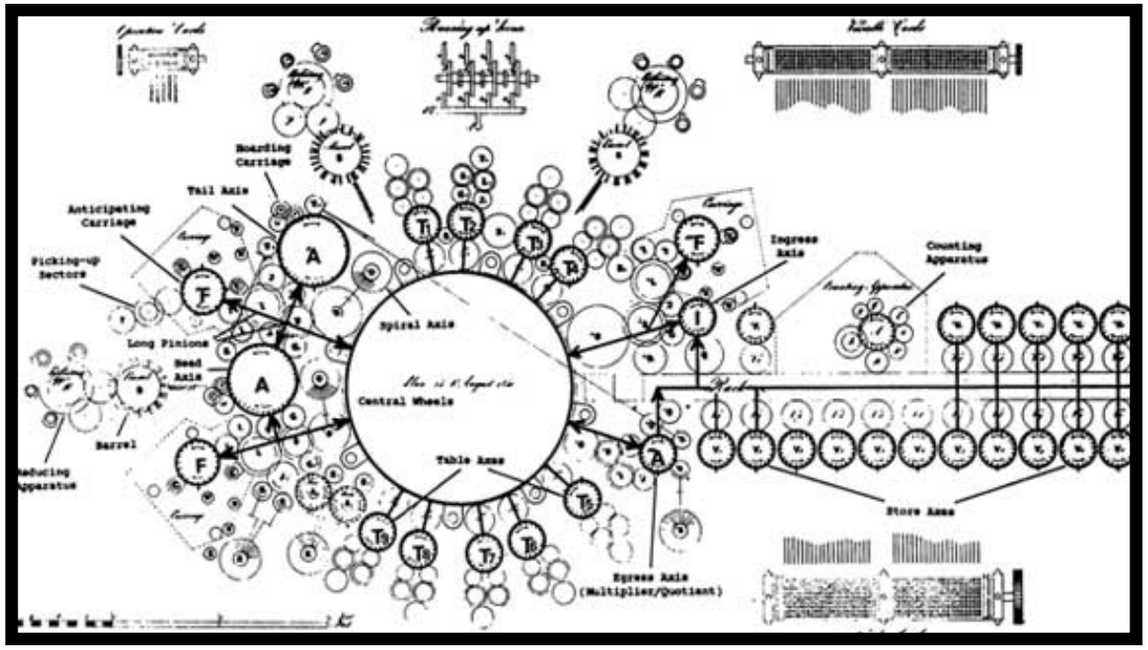
AlphaGo

- **AlphaGo**: program by Google DeepMind to play Go
    - Based on machine learning, not an expert system
    - Trained on 100K games of strong amateurs to learn how to mimic a good (but not great) human player
    - Then trains by playing against itself millions of times
      (reinforcement learning: reward strategies which lead to wins,
      penalize those which lead to losses)
    - Defeated European Champion Fan Hui 5-0 in October 2015
        - First time a computer had beaten a human professional Go player without a handicap
    - Defeated 18-time World Champion Lee Sedol 4-1 in March 2016
        - ~80 million people watched the first match
    - Made many moves which were puzzling to human experts
- **AlphaZero**: Better Go player that also plays Shogi and Chess
    - Trained entirely from self-play

History of AI

- I'll go through a brief summary of the history of computing, AI, and machine learning

- Along the way I'll highlight a few algorithms that we will look into in more detail after learning the history

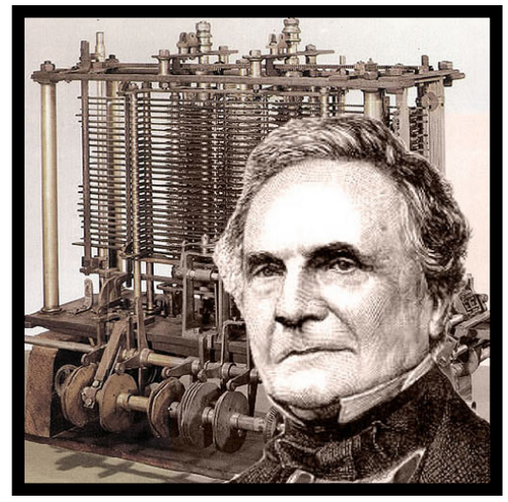


Blueprints for Charles Babbage's Analytical Engine, a 19th century mechanical computer

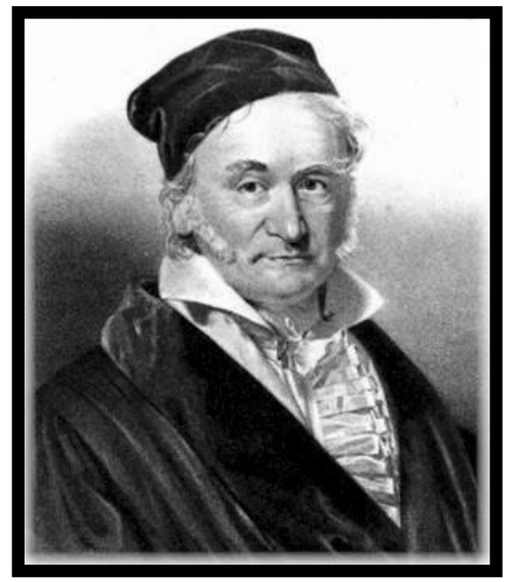
Navlab 1, an early attempt at self-driving cars from the 1980's

History of AI: The 19th Century

- Need **computers** for AI!
- Calculating machines (abacus, etc.) go back to antiquity
- Invention of modern (though not electronic) computers in the 1820's by Charles Babbage, Lucasian Professor at Cambridge
  - Memory distinct from processing, I/O
  - Later models used punchcards
  - "Analytical Engine": 675 bytes, 7 Hz
  - My (2015) Laptop: 250 GB, 2.2 GHz
  - His assistant, Ada Byron Lovelace, wrote the first known program for one his machines: maybe the world's first computer program!
  - None of Babbage's machines were successfully constructed during his lifetime (1791 - 1871)



- First **machine learning algorithm**: **least squares**
  - Gauss: 1795
  - Legendre: 1805 (first to publish)
  - We'll discuss how regression/ curve fitting is machine learning and how it relates to more sophisticated procedures in a few slides



History of AI: 1900-1960

- Invention of electronic computers

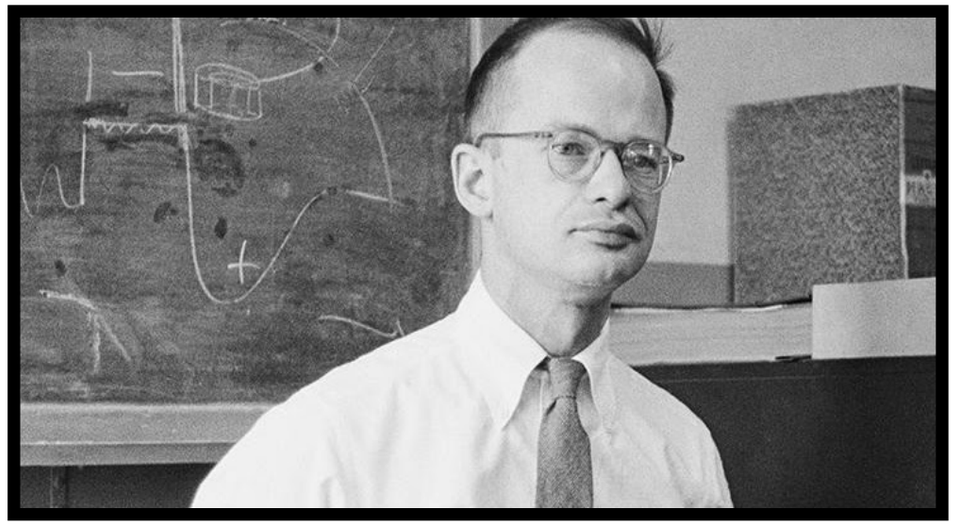
- UNIVAC: first commercially available computer

- ~1.5 MB memory, 1.9 kHz

- **Neural networks**: described in 1942 by Walter Pitts.
  Further work with Warren McCullough, 1943

- **Perceptron**: machine learning algorithm: originally implemented as a machine (1957)

- I'll have more to say about neural networks and perceptrons later



History of AI: 1960's - 1990's

- **Expert Systems and the AI Winter**

- Perceptron, AI more generally, may have been overhyped:
  "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence." (New York Times, 1958)

- Marvin Minsky and others pointed out limitations of the perceptron in the late 60's

- (D)ARPA became less willing to fund pure research (late 60's and early 70's)

- Reaction to limitations of early ML (especially compared with early promise) + limited computing power led to an increased focus on expert systems

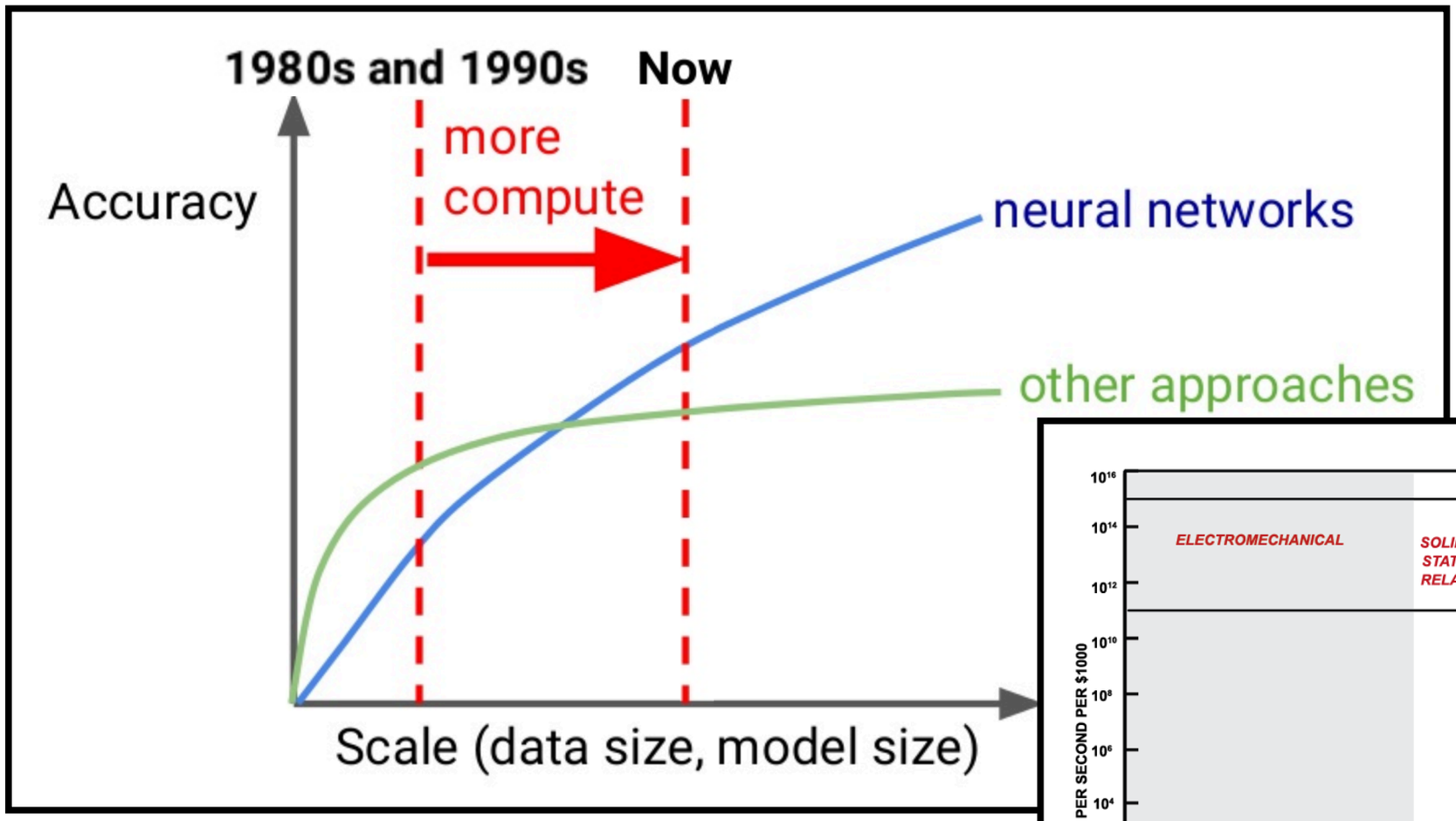
- In early 80's industry of "LISP machines":
  workstations optimized for running (expert system) AI programs using a language called LISP

    - Market was disrupted by Sun workstations, "microcomputers" from IBM and apple

- Expert systems hard to scale

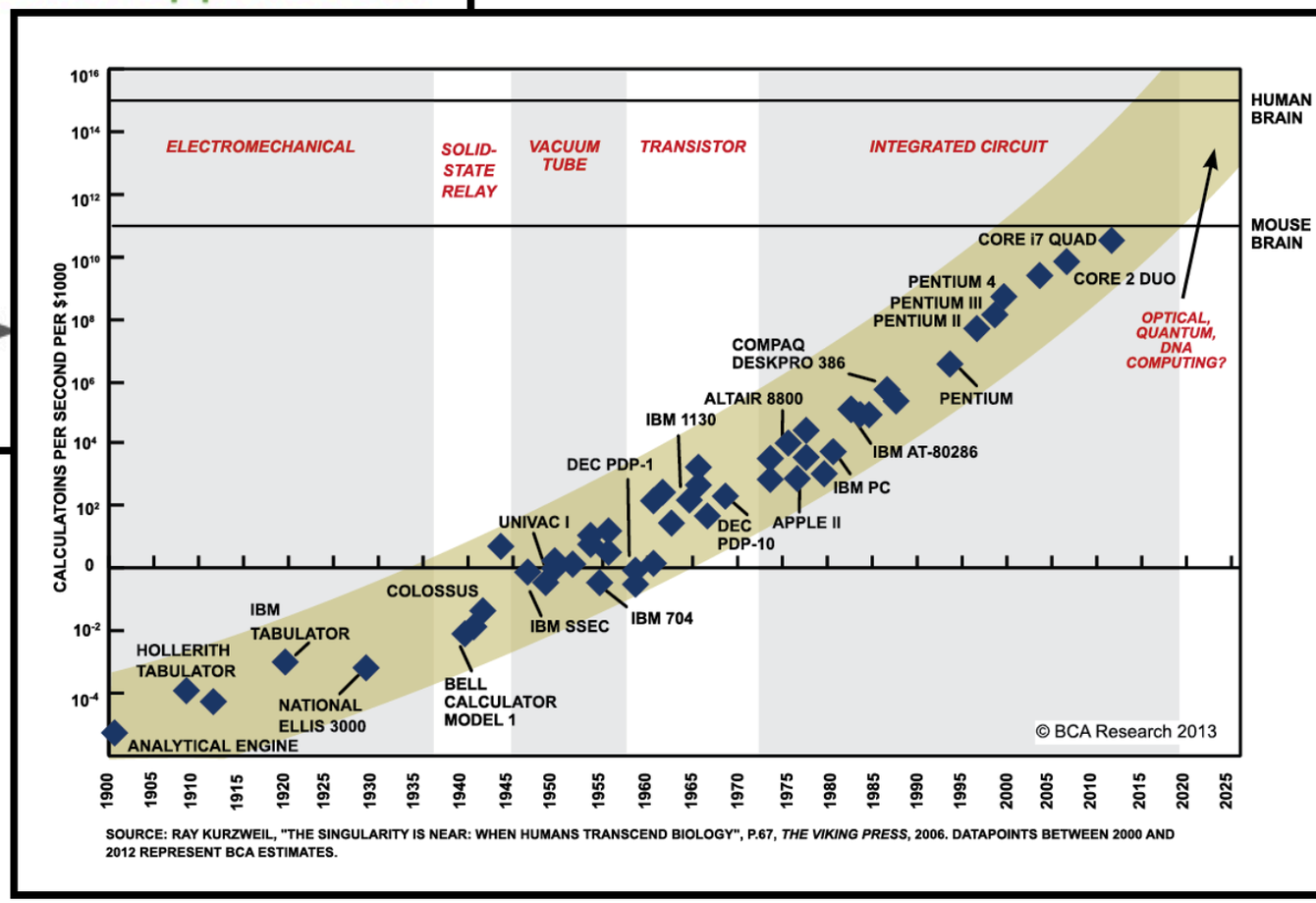
- Further funding cuts in 80's and 90's

The 2000's: Rebirth of ML



Adapted from Jeff Dean talk at AI Frontiers

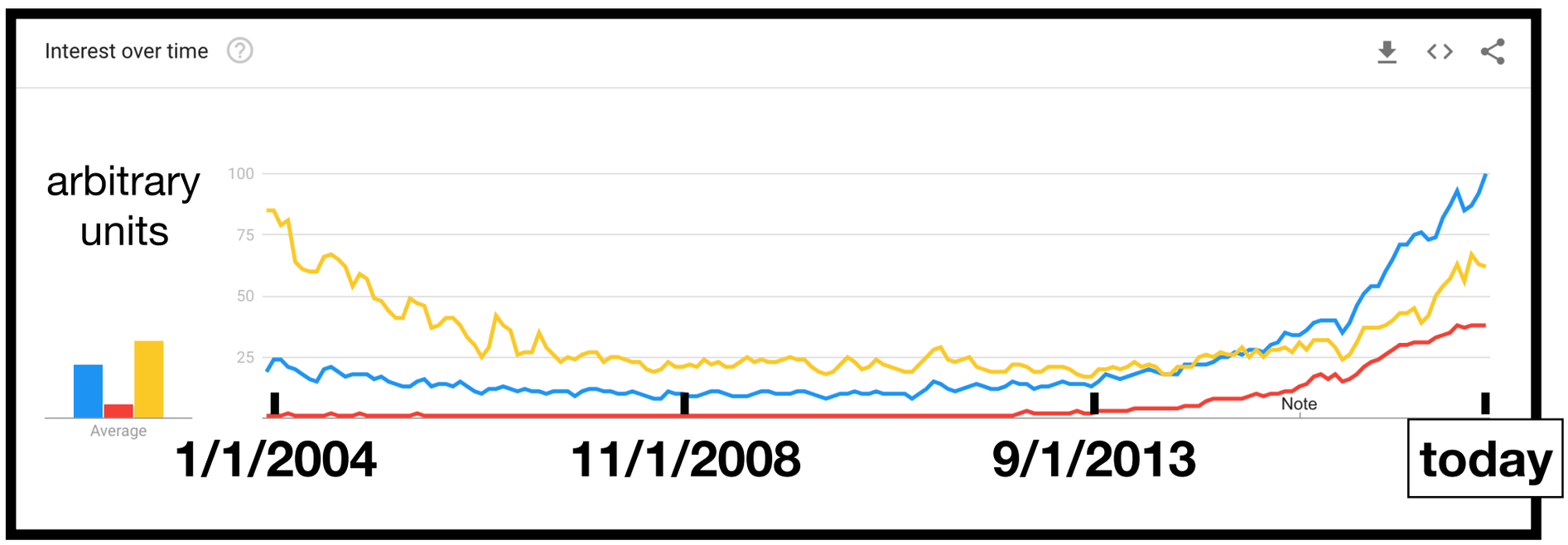
From Ray Kurzweil:
"The Singularity Is Near"



- Growth in computer power has given machine learning algorithms the edge over expert systems in many domains

The 2010's: Deep Learning

- Driven by improvements in processor power
    - ML has grown dramatically in prominence within AI
    - Neural nets have grown dramatically in prominence within ML
    - "Deep learning", i.e., ML with "deep" neural nets has grown dramatically in prominence within neural net approaches



Google Searches for

machine learning

artificial intelligence

deep learning

Example 1: Regression

- Secretly an introduction standard terminology and workflow for supervised machine learning algorithms

- This is a **supervised algorithm**: we have a **data set** which we will **train** on: lets assume we have a set of n vectors $\{\mathbf{x}_i\}$ where each $\mathbf{x}_i \in R^m$. We also have n function values $\{y_i\}$,
  such that $y_i = f_{true}(\mathbf{x}_i) + noise_i$

- In machine learning terminology the vectors are called "**samples**", the dimensions of the vector space are called **features**, and the vector space itself is called **feature space**

- Two important cases

  - The $\{y_i\}$ take on two or more integer values corresponding to **class labels**.
    Then we have a **classification** problem

  - The $\{y_i\}$ take on continuous real number variables

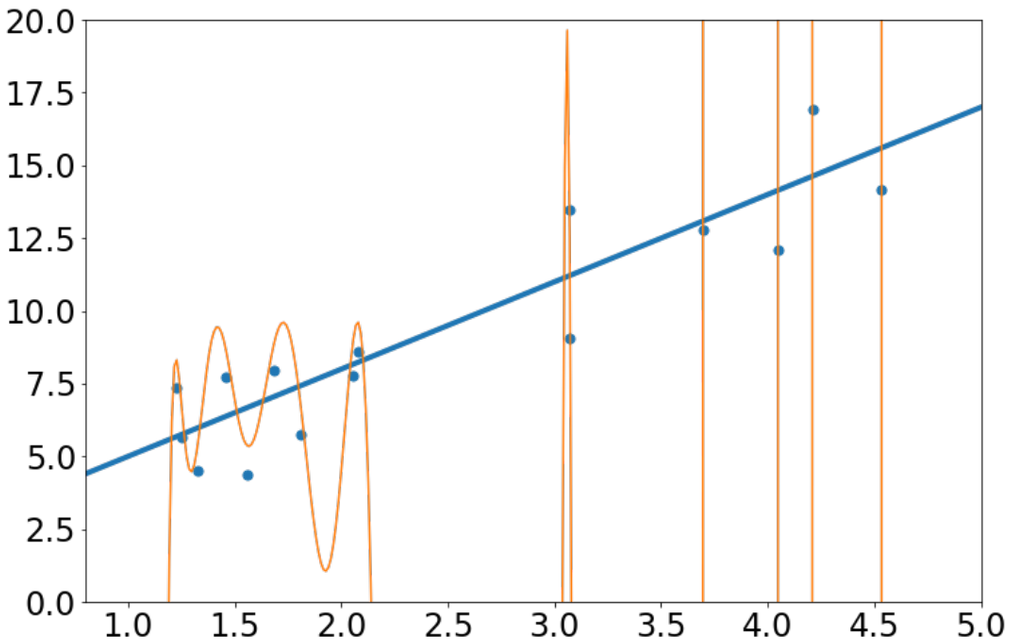
Least Squares Regression

- Let's go through how to perform a regression analysis in more detail
- Goal is to choose $f_{model}(\boldsymbol{\theta}, \mathbf{x})$: a fitting function
    - It depends on **parameters**, $\boldsymbol{\theta}$
    - E. g., linear: $f_{model}(\boldsymbol{\theta}, \mathbf{x}) = \boldsymbol{\alpha} \cdot \mathbf{x} + \beta$:
      $\boldsymbol{\alpha}$, β are parameters
    - Choice of f_{model} (how many terms, what those terms are, etc.): **hyperparameters**
- To fit, we minimize an **objective function** (**loss function**, **loss metric**) with respect to the parameters
- The classic example of an objective function for regression is the sum of squared error, e.g.

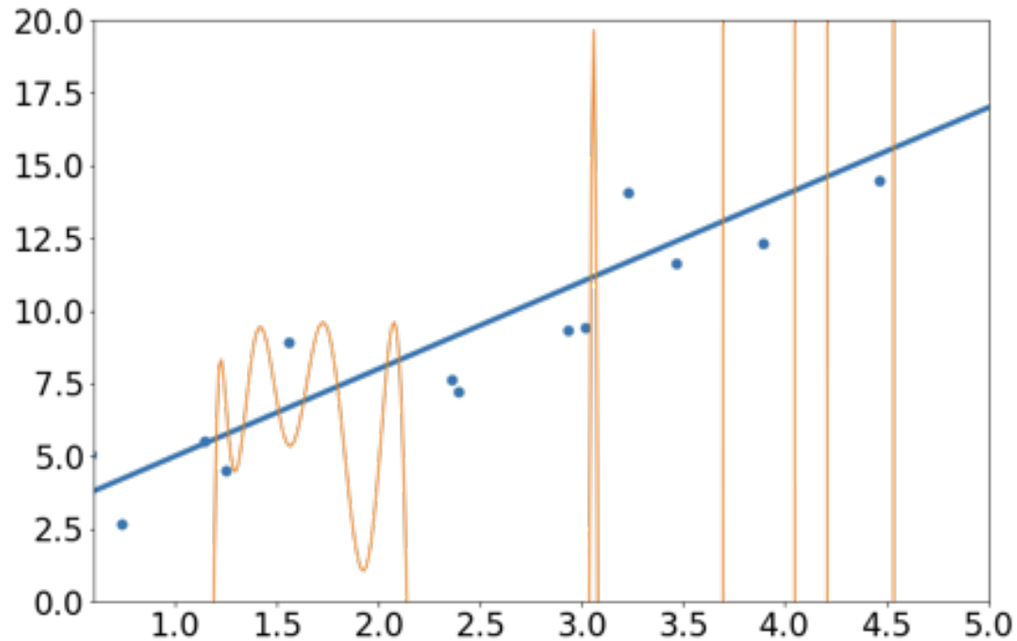
$$O(\boldsymbol{\theta}) = \Sigma \, (y_i - f_{model}(\boldsymbol{\theta}, \mathbf{x}_i))^2$$

- Minimizing the objective function (i.e., least squares) can be challenging due to local minima
- If gradient is easy to calculate (depends on $f_{model}(\boldsymbol{\theta}, \mathbf{x})$) we can use it in optimization, e.g. steepest descent minimization
- This approach is not suited to classification: fit the log odds, use a likelihood-derived objective function (related to entropy)

Avoiding Overfitting



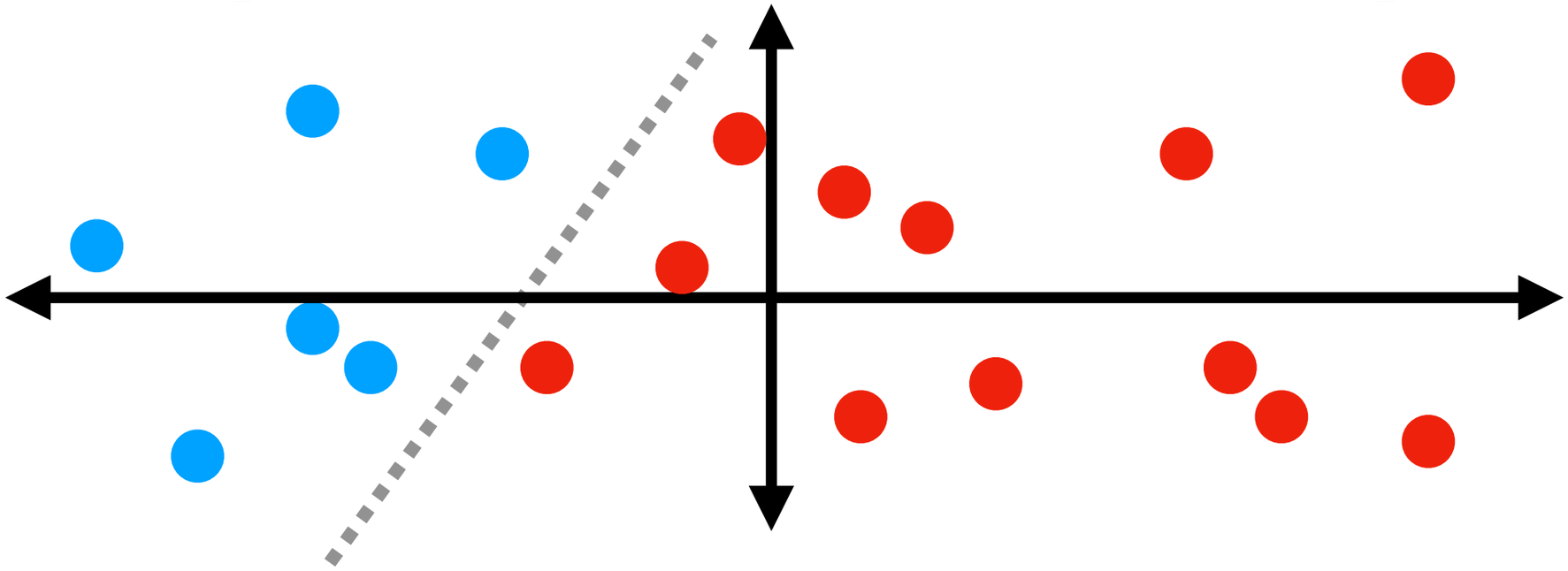
- Major problem: avoid **overfitting**!!! (Spuriously good fit on training data)
- Especially a problem with powerful ML techniques: many parameters so easy to overfit
- We avoid this by dividing our data set into three parts



- **Training Set**: Use to obtain parameters by optimizing the objective function
- **Validation Set**: Use to test various hyperparameters, make sure we are not overfitting on the training data
- **Test set**: See how well our final choice of parameters and hyperparameters will work on data we didn't fit to

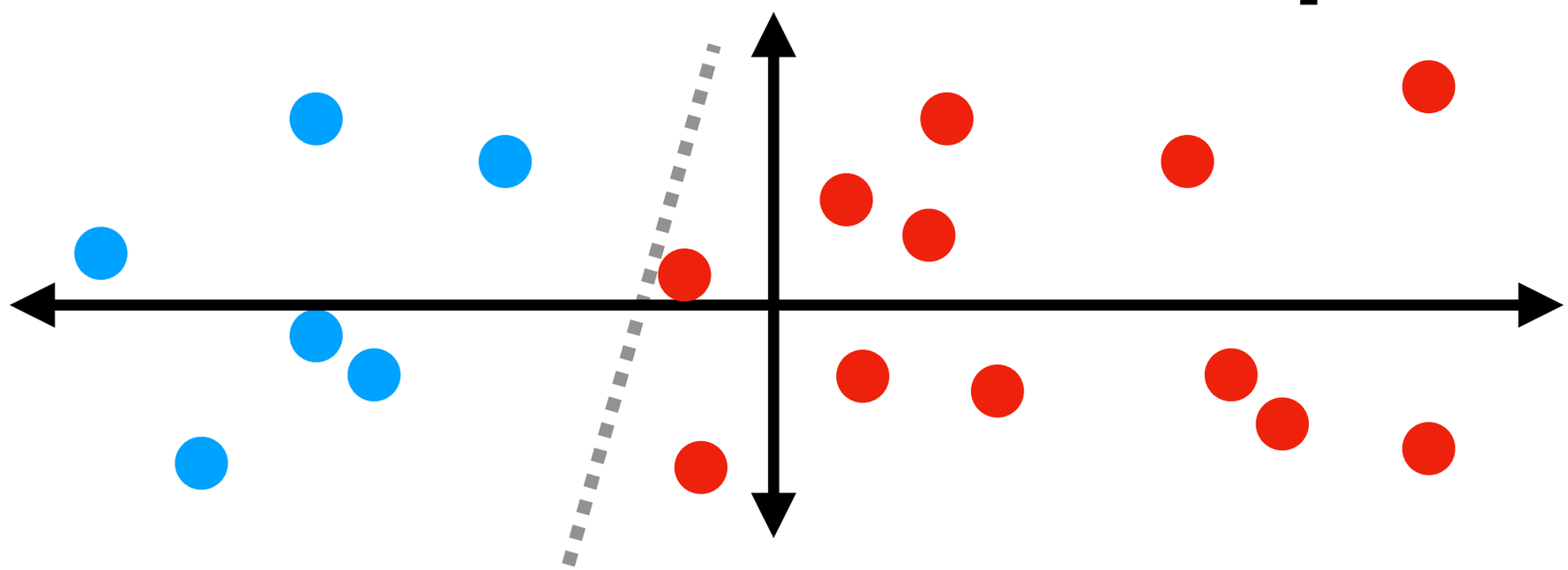
Another (not mutually exclusive) approach is regularization

Example 2: The Perceptron



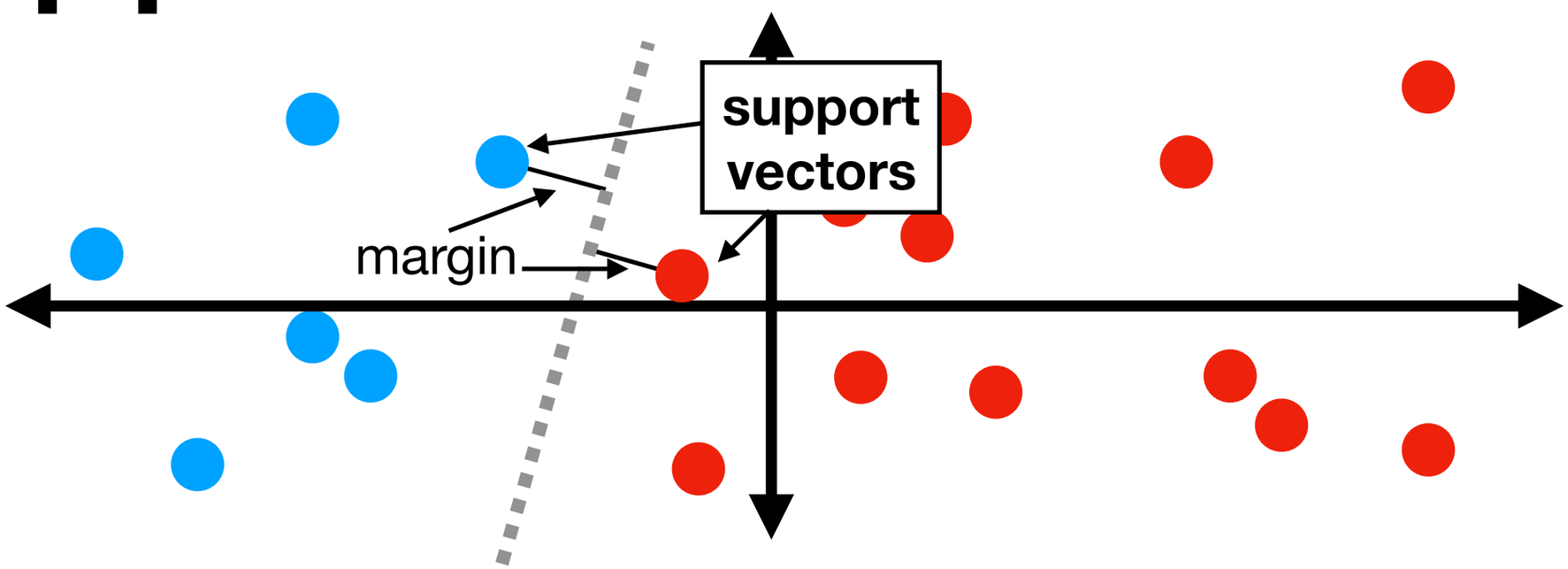
- Algorithm for classifying linearly separable data
    - I.e., finding the hyperplane that separates points in two classes
- Weight vector, **w**, and bias, b, define a hyperplane
- Correct choice of **w** and b will have $\mathbf{w}\cdot\mathbf{x}_i + b < 0$ for the red points and $\mathbf{w}\cdot\mathbf{x}_i + b > 0$ for the blue points
- Each step: calculate $\mathbf{w}\cdot\mathbf{x}_i + b$ for each point
    - If $\mathbf{w}\cdot\mathbf{x}_i + b = y_i$ where y_i is -1 for red and 1 for blue, then do nothing
    - Otherwise add $y_i\,\mathbf{x}_i$ to the weight vector for the next iteration
    - Can treat b as w_0 with $x_{i,0} = 1$, treat the same way as other components of **w**

A drawback of Perceptrons



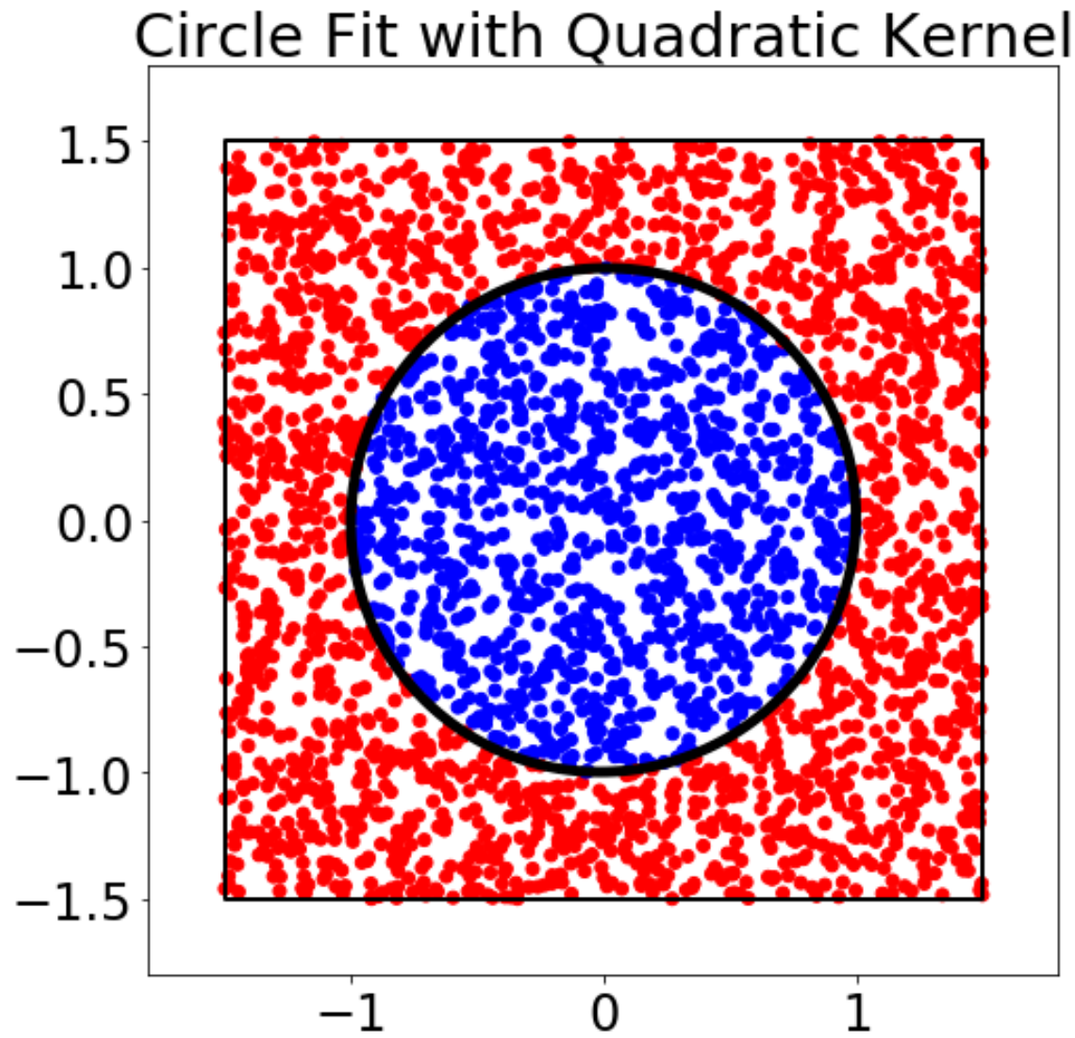
- One drawback of the perceptron is that the algorithm stops once all the points are classified correctly
- But many hyperplanes will correctly classify
- The hyperplane chosen by the algorithm may be too close to the data
    - Bad for applying the boundary to new data:
      not robust with respect to noise or limited statistics in the training set

Support Vector Machines



- The distance from the nearest points to the hyperplane is called the "margin"
- The nearest points are "support vectors"
- Algorithm: Support Vector Machines (SVM)
- We can use Lagrange multipliers, etc. to optimize the margin subject to constraints
- Only support vectors affect the optimal hyperplane (via their dot products) physically reasonable result

The Kernel Trick



- Of course not all data is linearly separable
- Nonlinearly separable data can be handled by transforming the data
- Example: unit circle
- Transform $(x,y) \rightarrow (x^2, y^2, \sqrt{2}\,xy)$
- In the new space the data is linearly separable
- We mentioned that the boundary depends only on scalar product of support vectors
- Replacing scalar product with "kernel function" $K(x_i, x_j)$ lets us perform this transformation implicitly
- E.g., $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2 = x_i^2\, x_j^2 + y_i^2\, y_j^2 + 2\, x_i\, y_i\, x_j\, y_j$ gives us the dot product in the transformed space: "**kernel trick**"
- This is a powerful result, but unfortunately SVM with non-linear kernels can be computationally intensive: part of why SVMs are less popular now than in early 2000's

Example 3: Neural Networks

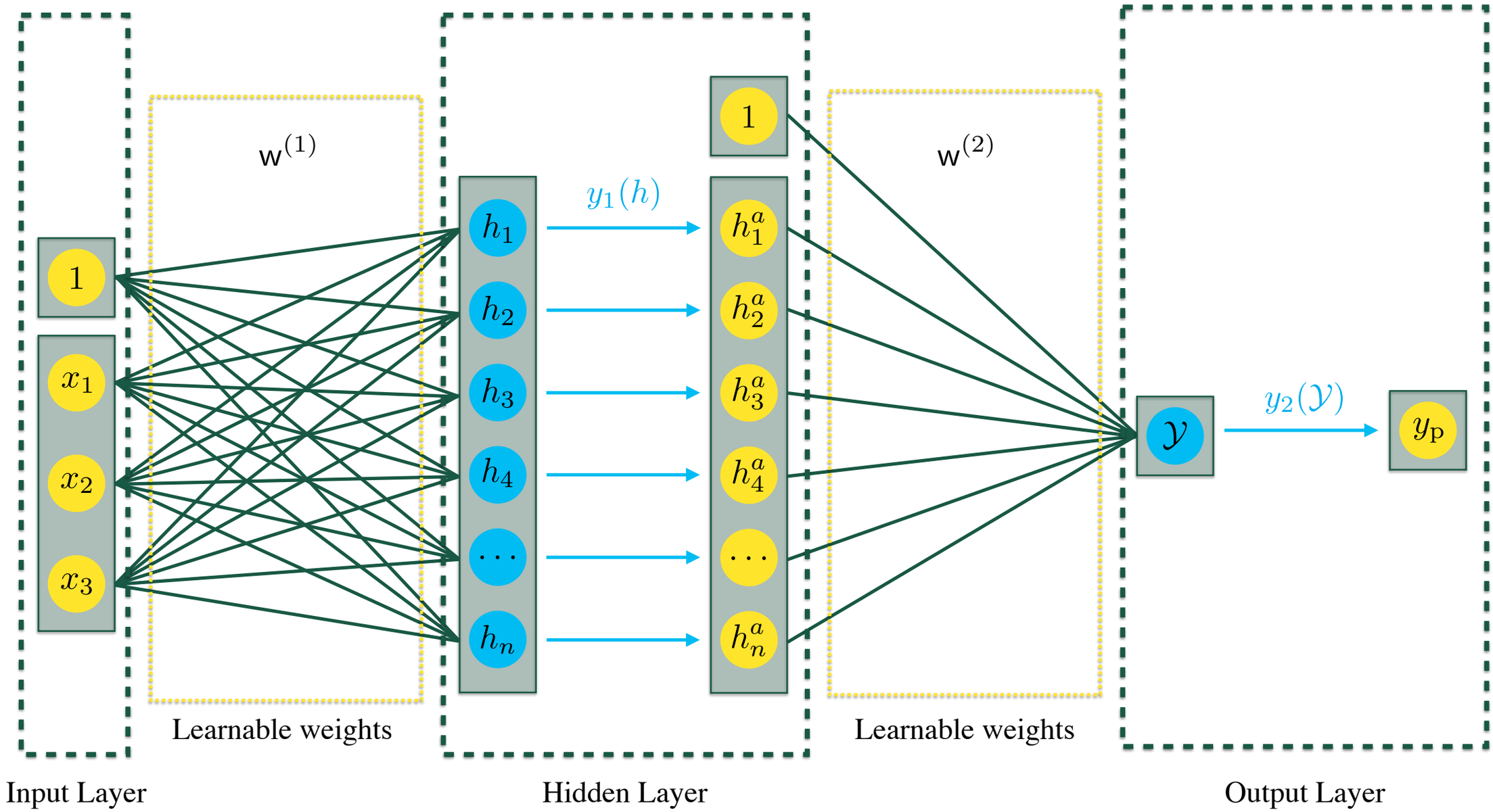


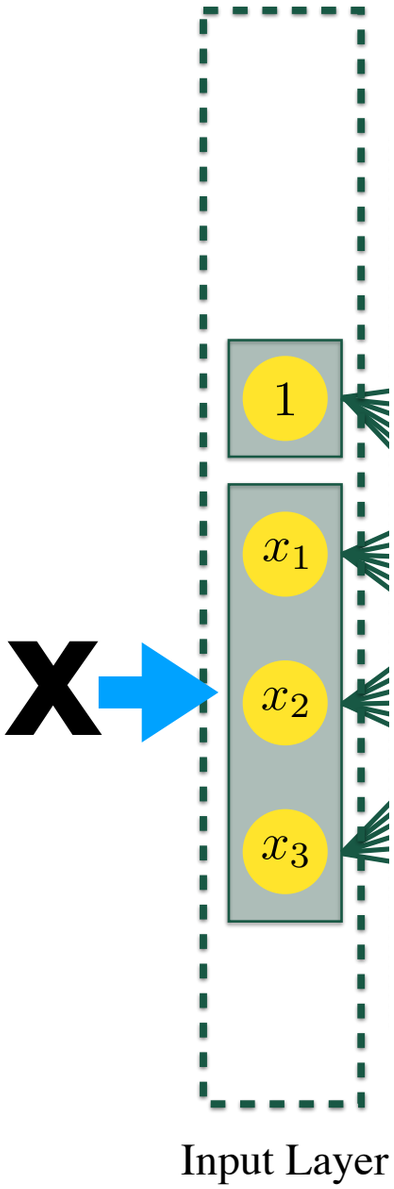
Diagram from Cohen, Freytsis, Ostdiek, 2017

- I'm going to explain how a neural network works going layer by layer

Flashback to Regression

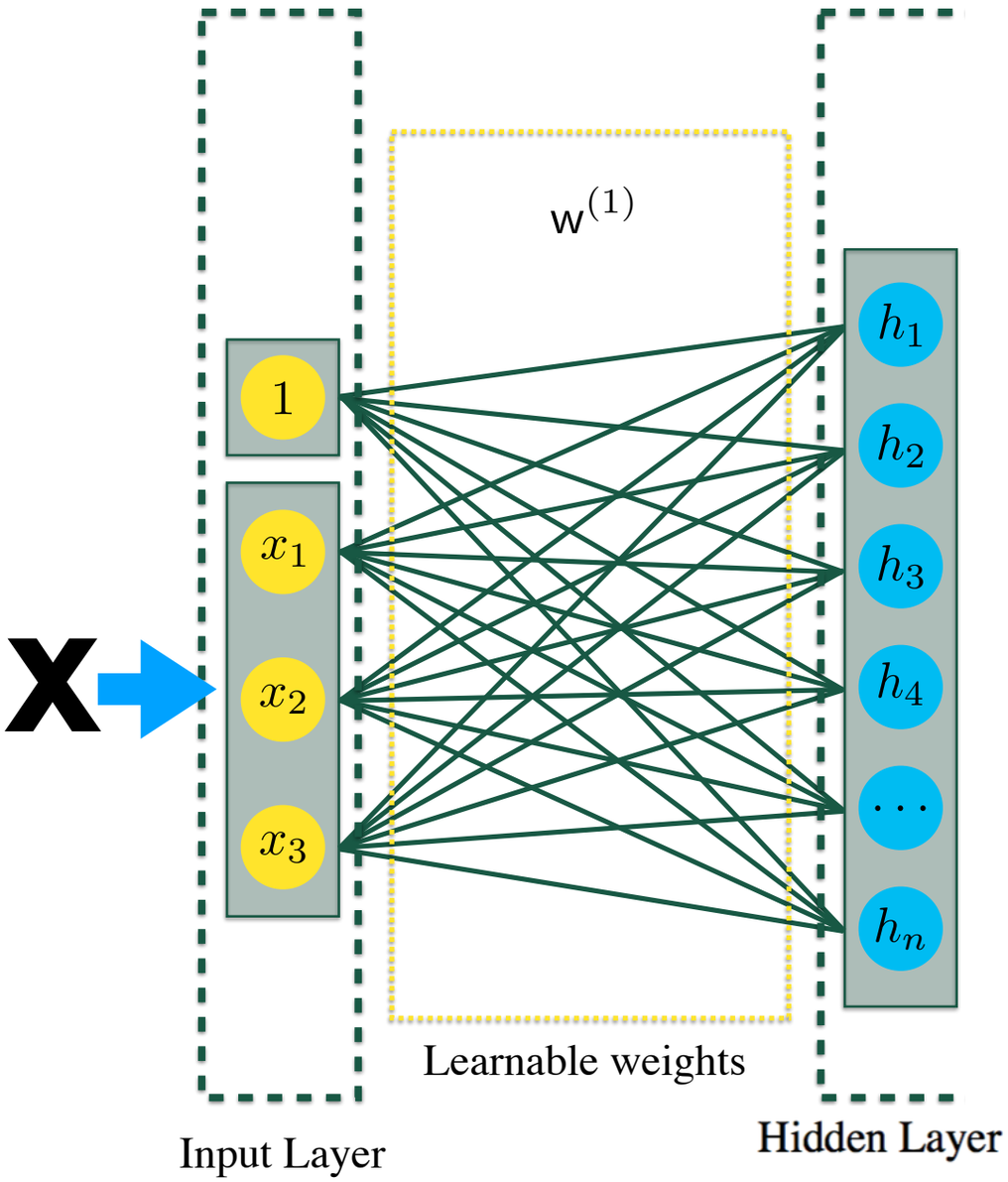
- Remember in regression or classification the goal is to find $f_{model}(\boldsymbol{\theta}, \mathbf{x})$ where $\boldsymbol{\theta}$ are the parameters and $\mathbf{x}$ is a vector in our input space
- We then find the best values of $\boldsymbol{\theta}$ by minimizing an objective function
- Here our $f_{model}(\boldsymbol{\theta}, \mathbf{x})$ will be a neural net with a particular architecture
    - The parameters, $\boldsymbol{\theta}$, that we are obtaining in our fit are the weights, $\mathbf{w}$, of the neural network
- Same workflow as for other classification and regression problems: just a fancy function, $f_{model}(\boldsymbol{\theta}, \mathbf{x})$

Intro to Neural Nets



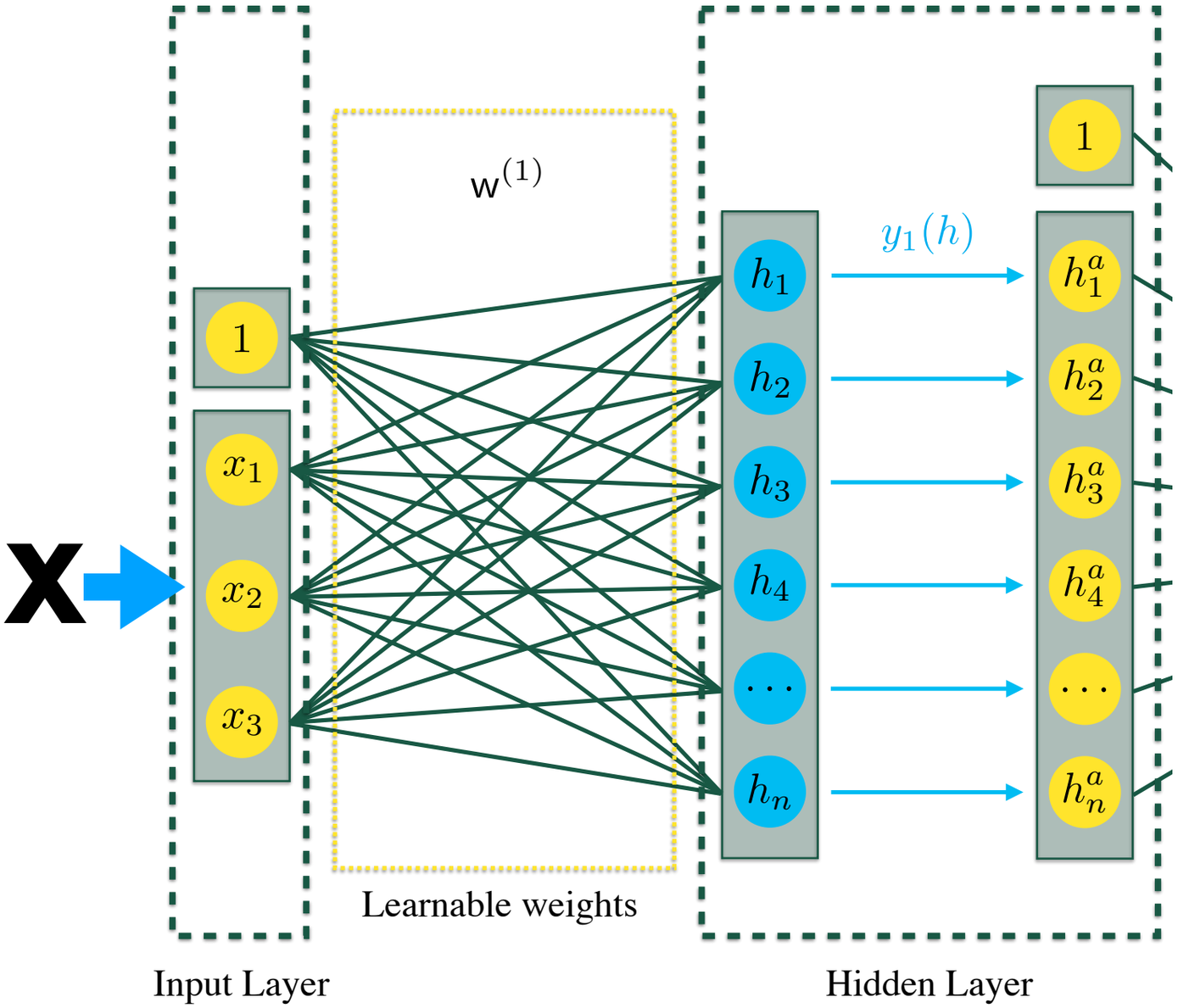
- We have an **input layer** of **neurons**, one for each dimension of **x**/ feature
- Also a **bias neuron**
- Take in **x**: output x_i (or 1)

Intro to Neural Nets



- We now see weights…
- and a hidden layer of n neurons
- We can describe the weights as w_{ij}, where i ranges from 1 to n and j ranges from 0 (for the bias neuron) to 3 (in this case)
- Inputs to the hidden layer neurons are given by matrix multiplication
  - e.g., input to $h_1 = w_{10} + w_{11} x_1 + w_{12} x_2 + w_{13} x_3$
  - "tensors": "TensorFlow"

Intro to Neural Nets



- Each neuron in the hidden layer takes its input and applies a function y(h) to this input

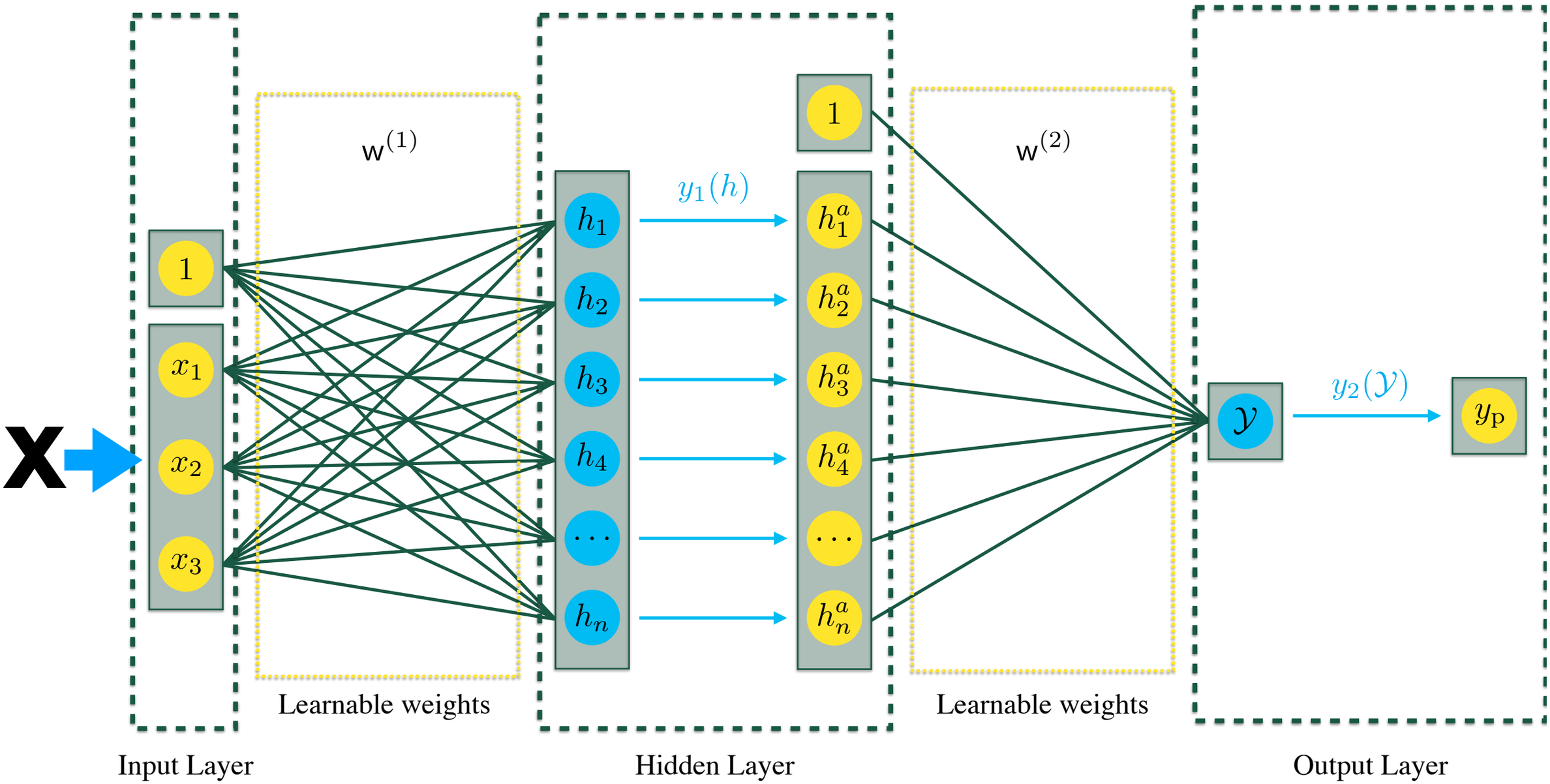
- We want this function to be

  1. Be easy to evaluate
  2. Have compact output
  3. Be analytically differentiable
  4. Be monotonic

- In practice y(h) is generally the sigmoid/ logistic function $y(h) = 1/(1 + e^{-x})$ or y(h) = tanh(h)

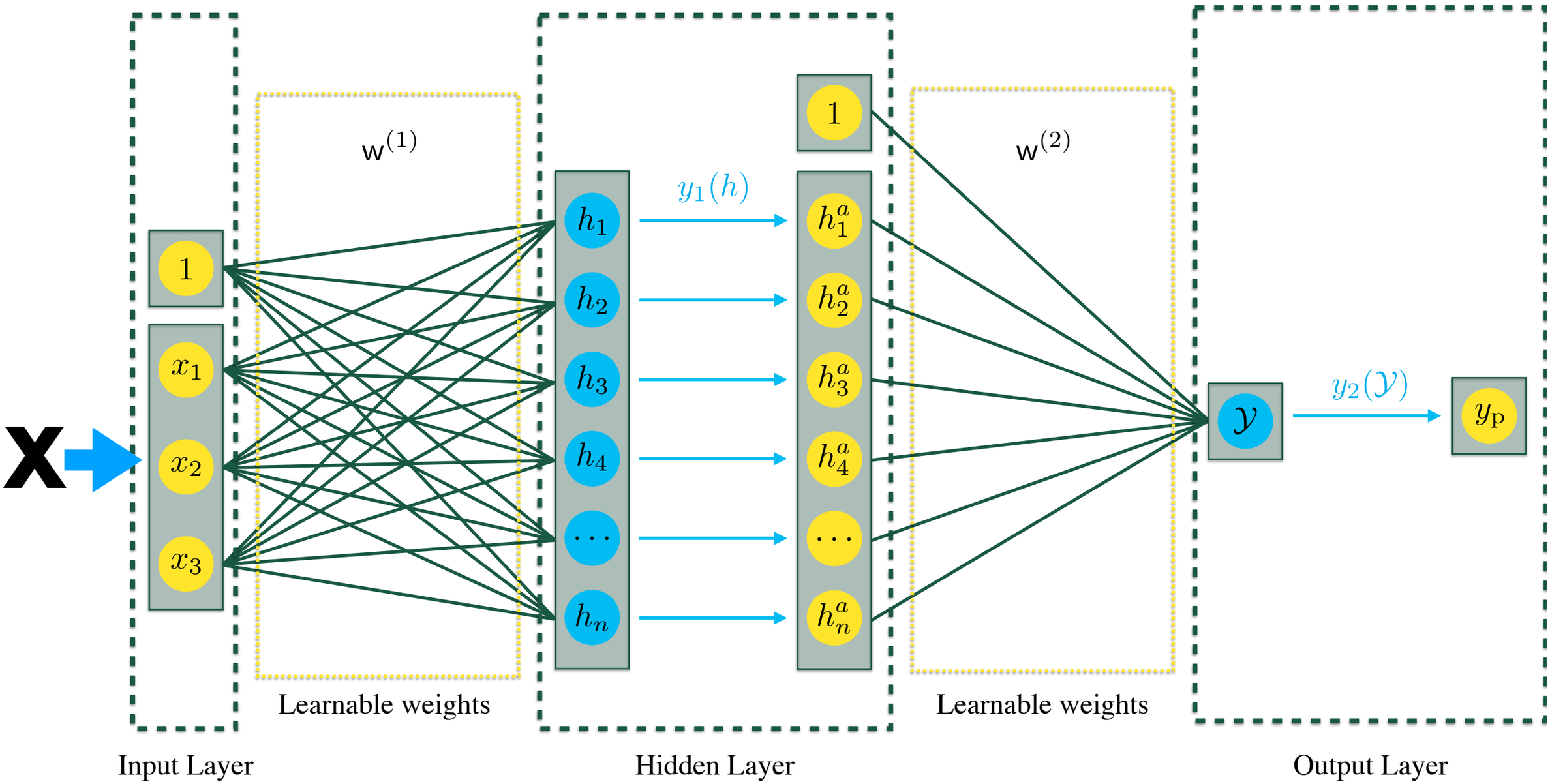
- y(h) is called the "**activation**" of the neuron

Intro to Neural Nets



- Now we have another set of weights, which multiply the outputs of the hidden layer neurons, giving us the input to the output neuron

Intro to Neural Nets



- The output neuron can apply an activation function to its input: we then have our final value of f(**w**, **x**)

Backpropagation

- Since we chose the activation functions y_1(input), y_2(input) to be analytically differentiable, for fixed $\{\mathbf{x}_i\}$, we can easily calculate the gradient of the objective function (like sum of squares) with respect to the weights, **w** using the chain rule

- Doing the calculation which gives us the gradient with respect to each weight in a specific, efficient, manner is called **backpropagation**

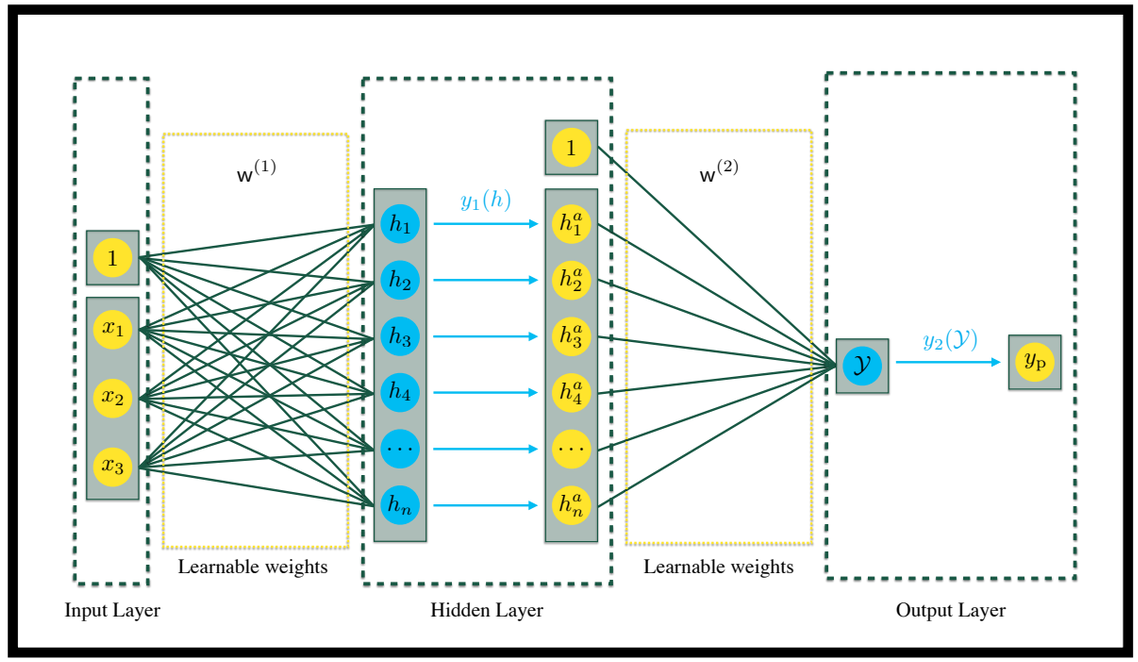
- We can also obtain other information, e.g, second derivates (Hessian)

- Using gradients (and more depending on the algorithm), we can then minimize the cost function with respect to **w**

- Generally there will be many local minima in **w** space, so optimization may be non-trivial

Variants

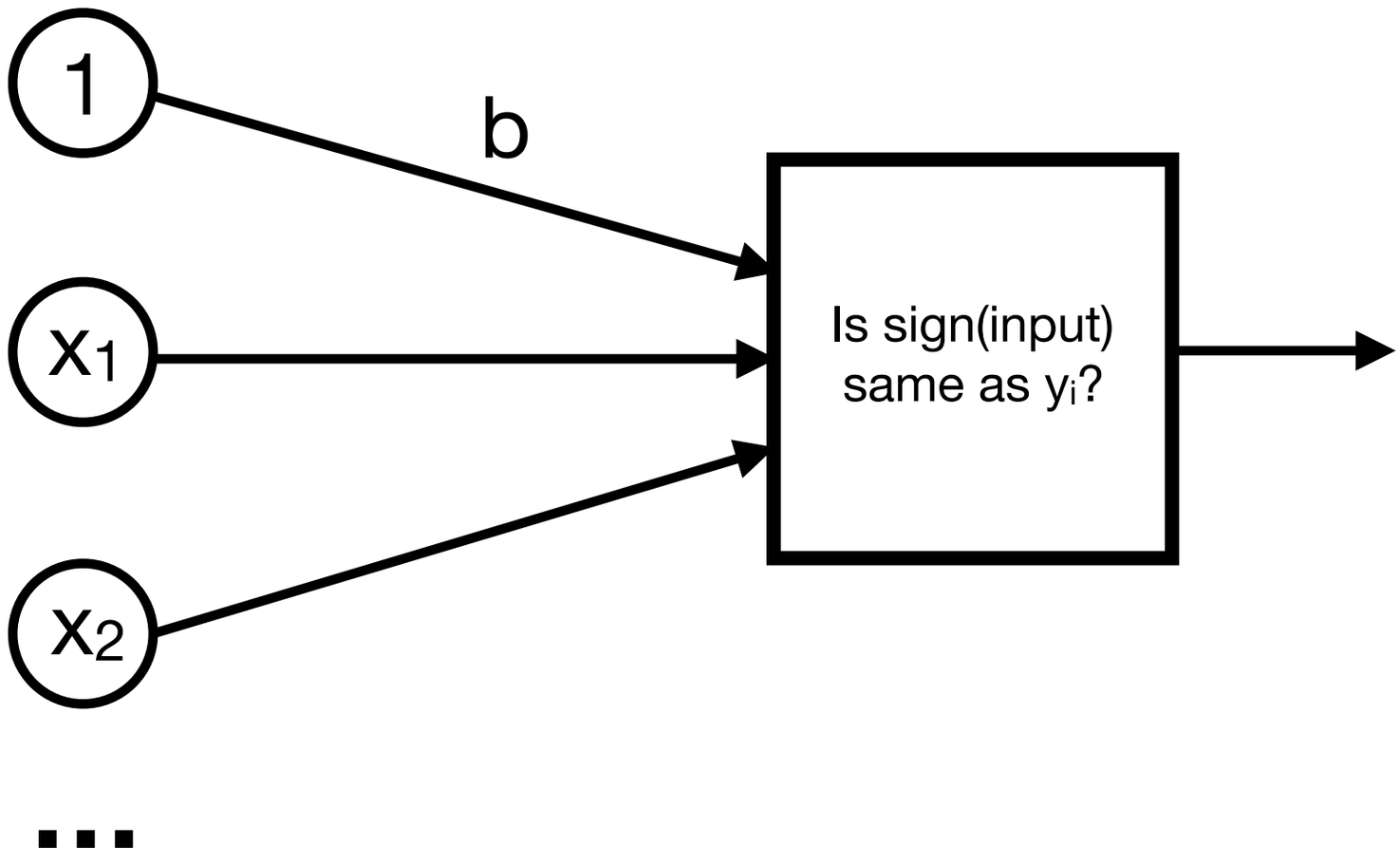
- In our example
  - Single hidden layer
  - Data moves in one direction/
    All weight arrows: →



- Single hidden layer = “shallow” neural network
- Multiple hidden layers: “**deep learning**”
- Data moves in one direction: **“feedforward”**
- The alternative, having outputs from neurons go back to an earlier layer, leads to **“recursive neural networks”**

Perceptrons are Neural Nets

- In the perceptron algorithm, we evaluate **w**·**x** + b



- So a perceptron is an extremely shallow (no hidden layer) neural network
- Sometimes other neural networks are referred to as “multilayer perceptrons”

Example 4: Boosted Decision Trees

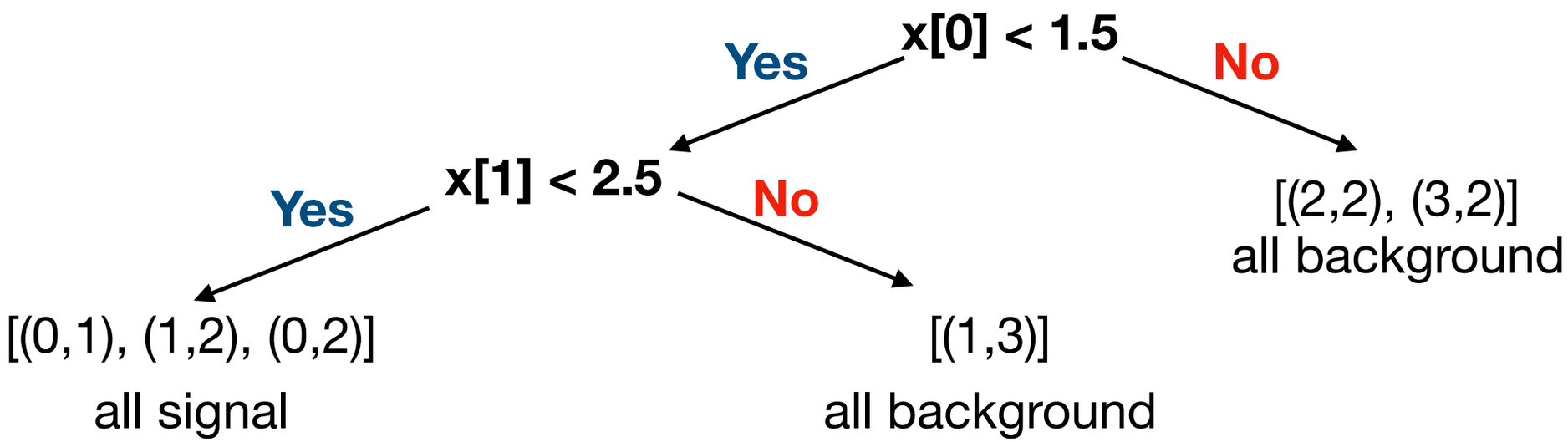
- **Decision tree**: series of yes/no questions that identify bins of data with given (e.g.) signal purity

- Example:
  signal: [(0,1), (1,2), (0,2)]
  background: [(1,3), (2,2), (3,2)]

- bin 1: x[0] < 1.5: “Yes” bin has 3 signal and 1 background events
  “No” bin has 0 signal and 2 background events

- No need to subdivide the “No” bin

- A cut of x[1] < 2.5 divides the “Yes” bin into a bin with 3 signal events and a bin with 1 background event



Example 4: Boosted Decision Trees

- In the example we showed the decision tree is a very good classifier (every point in our training set is correctly classified)
- Always possible for a sufficiently long tree (unless there are degenerate data points with different labels)
- In general this will not be possible for a short tree
- Long trees will generally overfit: so we want short trees, which will be robust with respect to noise, statistical variance in the training set
- So we will have a weak classifier: bins at the base of the tree will not be pure signal or background but mixed
- It turns out we can build stronger classifiers out of weak classifiers by **boosting**, a program of reweighing events which are misclassified, then rerunning the algorithm, …
- **Boosted decision trees** are an important algorithm in particle physics

ML in Particle Physics

- Machine learning is an important part of data analysis in experimental particle physics

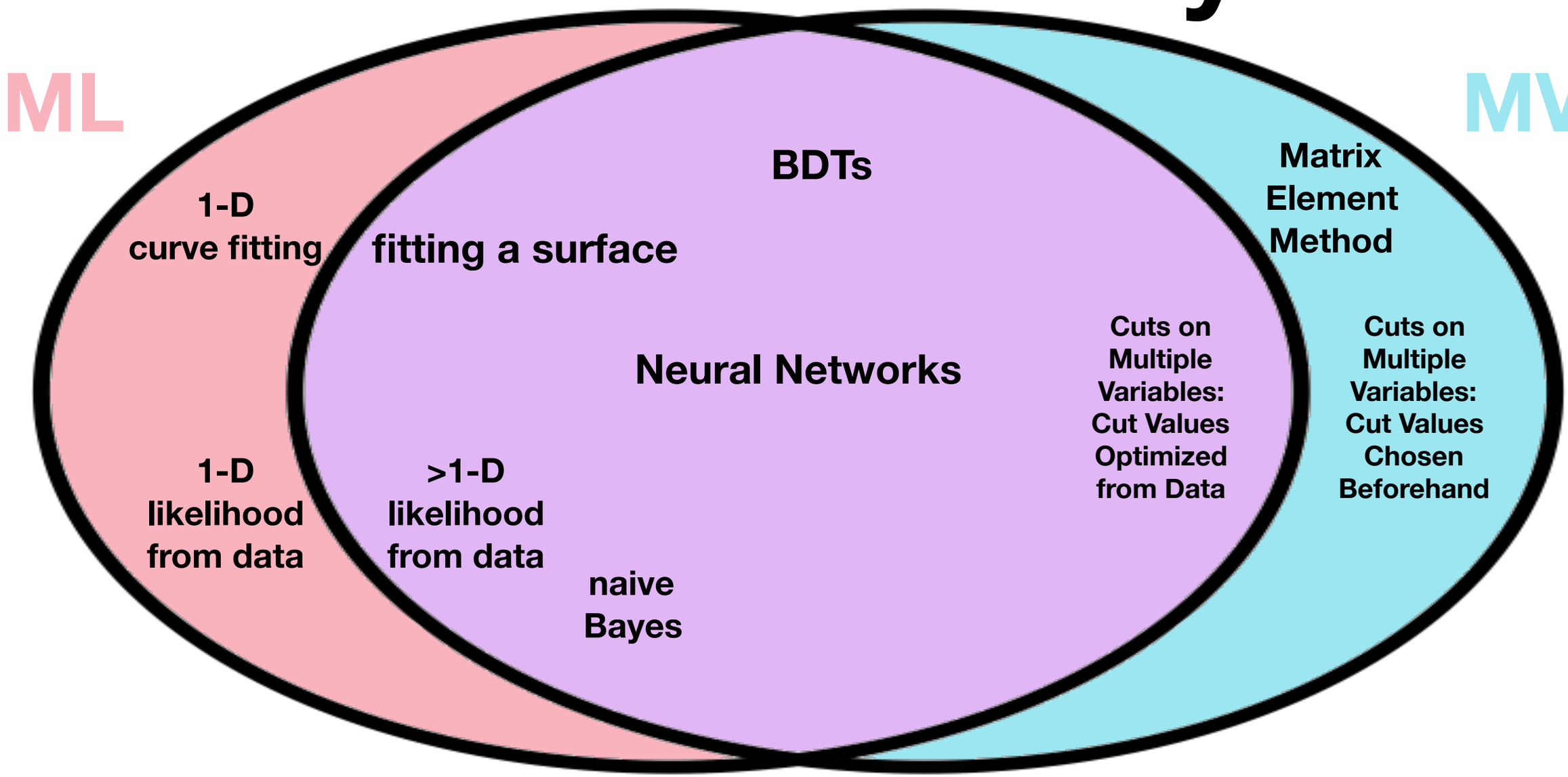
- Two major use cases

    1. **Identifying physics objects**
       (e.g., b-tagging from tracking and calorimeter information)
    2. Determining underlying process for events:
       signal versus background

- Besides curve fitting (regression) main algorithms include

    - Neural networks (mostly shallow, though moving toward deep learning)
    - Boosted decision trees
    - Likelihoods derived from data

Multivariate Analyses



- Much of the discussion of more sophisticated analyses in particle physics has historically been in the context of comparing single variable and multivariate analyses (MVA)

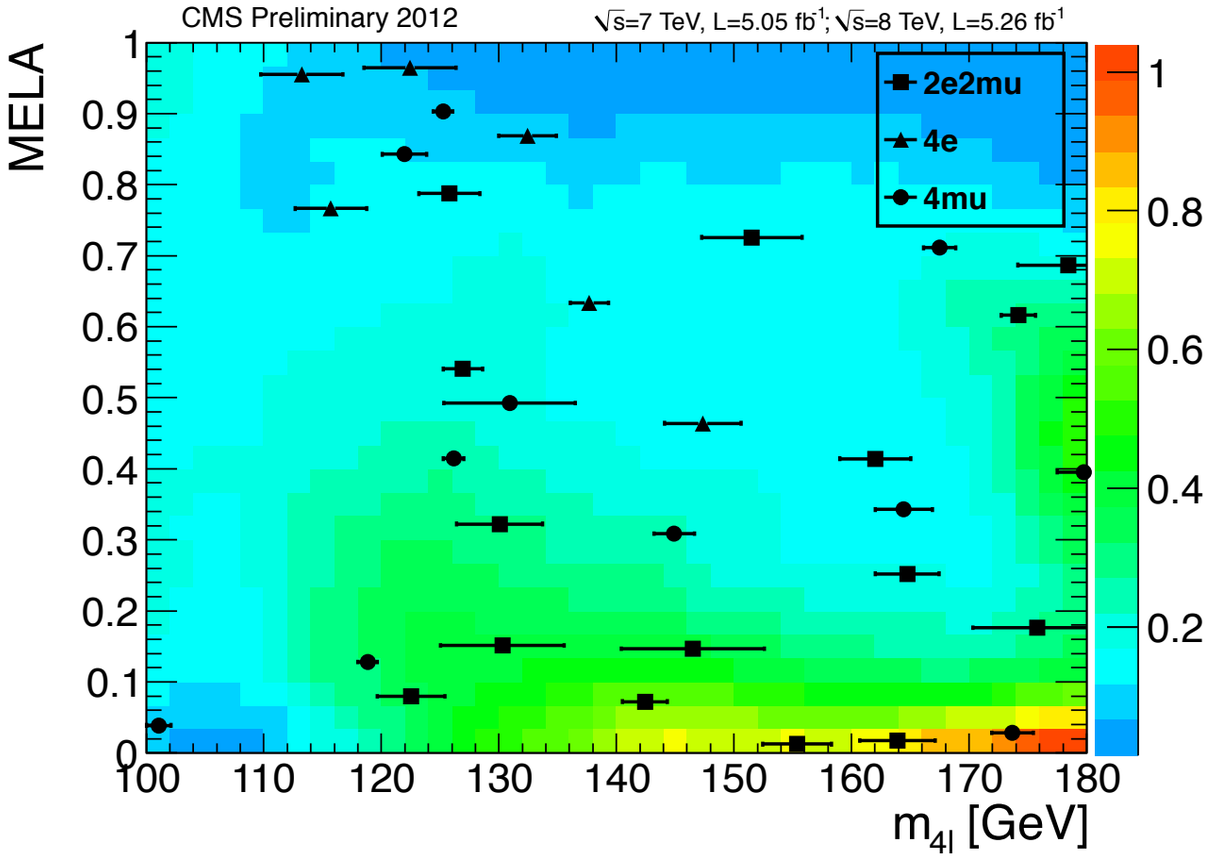
- Example: TMVA in ROOT: many tools for machine learning analyses

ML and MVA

- It's clear why we would want to use more variables in our analyses: more information
    - Though if we can make a discovery with a single variable, that's great!
- Do MVA need to be ML algorithms?
- No!
    1. Matrix Element Method (MEM)
        - An approach where we determine likelihoods directly from the differential cross sections for the signal and background processes
    2. Multiple cuts with cut values chosen without looking at simulated data

Why ML?

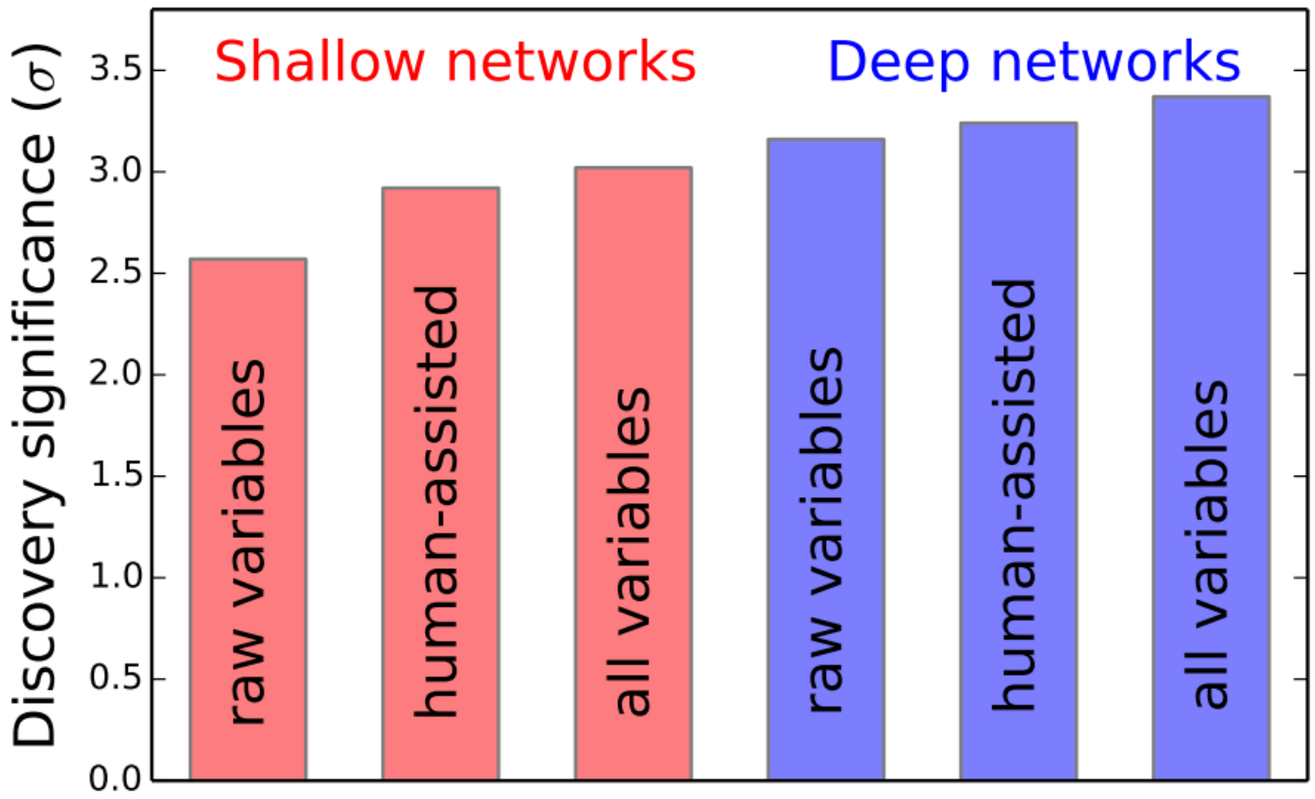
- Why use ML to perform MVA?
    - Lets us model the effects of hadronization, showering, and detector response via MC simulation rather than, e.g., analytically
    - Use PYTHIA, Herwig, Geant, etc. to generate realistic events, then use function from ML to classify signal and background events
- In practice some ML is almost always necessary: MEM for Higgs discovery used non-ML likelihood ratio as a variable/ feature



- CMS MEM analysis for Higgs discovery in 4-leptons.
- Obtained the 2-D likelihood of MEM/ “MELA” variable versus 4-lepton invariant mass from simulation (ML)
- Colors on plot give MELA/ m4l values for background distribution from simulation

Deep Learning in Particle Physics

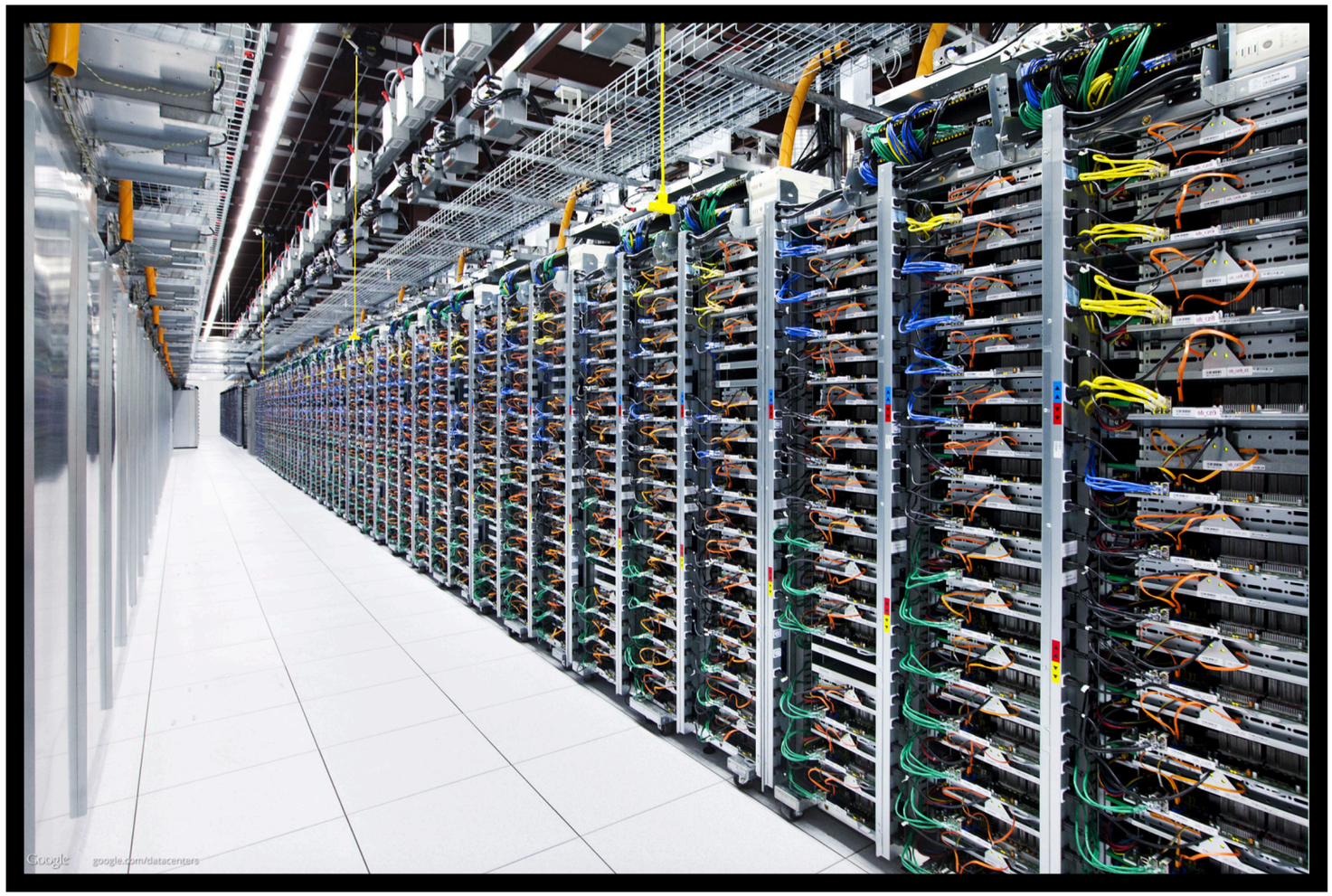
- Deep learning (multi-layer neural nets) beginning to be used in particle physics
    - Increase performance
    - At the cost of more work to train the network



Discovery significance in H to $\tau^+\tau^-$
Baldi, Sadowski, Whiteson 2015

Physics and Tech: Big Data

- Both Particle Physics and Silicon Valley need to deal with a ton of data
- **LHC**:
    - P = peta = 10^{15}
    - Currently ≳ 200 PB total
    - ≈ 30 PB per year
- **Google**: 10-15 EB in 2010
    - E = exo = 10^{18}
    - Much more today
    - Couldn't find specific value
- **All the data in the world**
    - 16 ZB in 2016
    - ≈160 ZB in 2025
    - Z = zetta = 10^{21}
    - Numbers from Seagate (data storage company)



A Google data center in Oklahoma

Physics and Tech: Different Goals

- In particle physics discovery is the goal
- Discovery ~ observing data whose p-value is $< \varepsilon$
    - 3σ: $\varepsilon \approx 1.3 \times 10^{-3}$
    - 4σ: $\varepsilon \approx 3.2 \times 10^{-5}$
    - **5σ: $\varepsilon \approx 2.9 \times 10^{-7}$**
    - 6σ: $\varepsilon \approx 9.9 \times 10^{-10}$
- Sensitivity depends on accurate modeling of tails of background distributions
- Amazon (e.g.) is generally less interested in ppm customer behavior than in what typical users do
- Important in both sectors to understand why analyses work
    - Physics need to understand tails of distribution makes it even more critical

Conclusions

- Machine learning in its broadest sense has always been part of experimental particle physics

- Many algorithms, more yet to be developed?

- Great excitement in recent years from deep neural networks

- Which will play a larger and larger role in particle physics, displacing shallow neural networks and boosted decision trees

- This is a fast-developing area: stay tuned!