Machine learning at CoDaS-HEP 2024

Jim Pivarski

Princeton University – IRIS-HEP

July 25, 2024

Now we have two ways to make computer programs

Craftsmanship

- Programming "by hand"
- Allows for precise control
- Complexity limited by a human mind or a team's ability to communicate

Farming

- Machine learning
- Allows for extremely nuanced solutions
- Still needs human help to steer it toward the "right" solution

Machine learning in a nutshell

Write an algorithm that generates output that depends on a huge number of internal parameters.

Machine learning in a nutshell

Write an algorithm that generates output that depends on a huge number of internal parameters.

Vary those parameters until the algorithm returns the expected result (supervised learning) or until it finds patterns according to some desired metric (unsupervised learning).

Machine learning in a nutshell

Write an algorithm that generates output that depends on a huge number of internal parameters.

Vary those parameters until the algorithm returns the expected result (supervised learning) or until it finds patterns according to some desired metric (unsupervised learning).

Then use the trained algorithm on new problems.

Write an algorithm that generates output that depends on a huge number of internal parameters.

Vary those parameters until the algorithm returns the expected result (supervised learning) or until it finds patterns according to some desired metric (unsupervised learning).

Then use the trained algorithm on new problems.

If this sounds like fitting data with a function, you're right.

Goals of this mini-course

1. Understand how your physics background prepares you for machine learning.
2. Don't approach it as a black box/dark art.
3. Get a little familiar with some common tools and techniques.

A very brief history of HEP and ML

Main point: High Energy Physics (HEP) has always needed Machine Learning (ML).

It's just becoming *possible* now.

First HEP experiments adopted computers in a major way

Late 1940's—early 1950's was the "beginning" of HEP as we know it:

- Accelerators provided higher energy with higher flux than observed in nature.
- Collisions with fixed targets produced new particles to discover.
- Computers quantified particle trajectories, reconstructed invisible (neutral) particles, and rejected backgrounds.

Example: Luis Alvarez's group \$9M accelerator, \$2M bubble chamber, \$0.2M IBM 650.

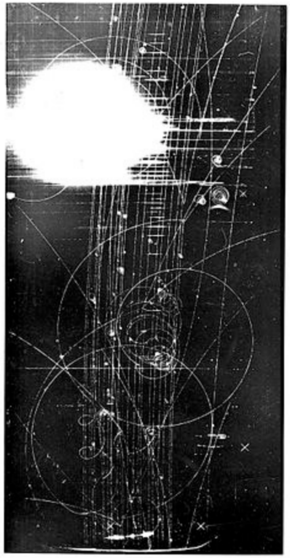
Identifying tracks was beyond the capabilities of software

Madeleine (née Goldstein)
Isenberg, UCLA class of '65

"We scanners would review each frame of film, and per the brief instructions we had been given, looked for any 'unusual activity.'

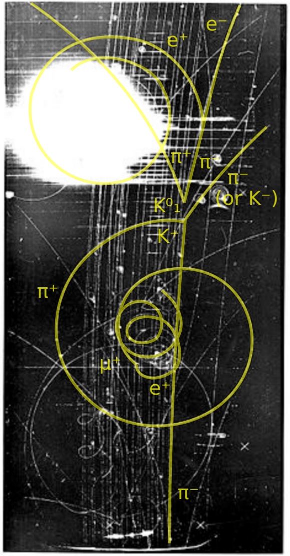
"The scanner had to use both hands, a joystick in each, and turn them clockwise or anti-clockwise, to align a double crosshair cursor at several sequential positions on a track."

https://www.physics.ucla.edu/marty/HighEnergyPhysics.pdf



Detectors make uninterpreted event displays.

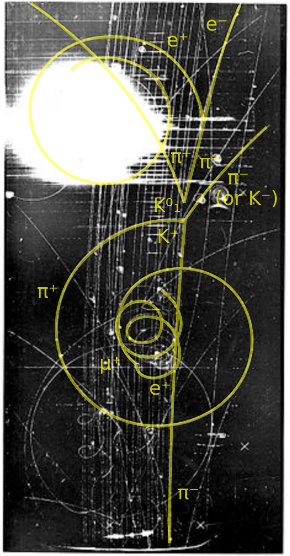
Fast pattern recognition tasks are an essential part of HEP



Detectors make uninterpreted event displays.

Raw signals must be interpreted as particles.

Fast pattern recognition tasks are an essential part of HEP



Detectors make uninterpreted event displays.

Raw signals must be interpreted as particles.

Capacity for discovery scales with the number of interpreted events.

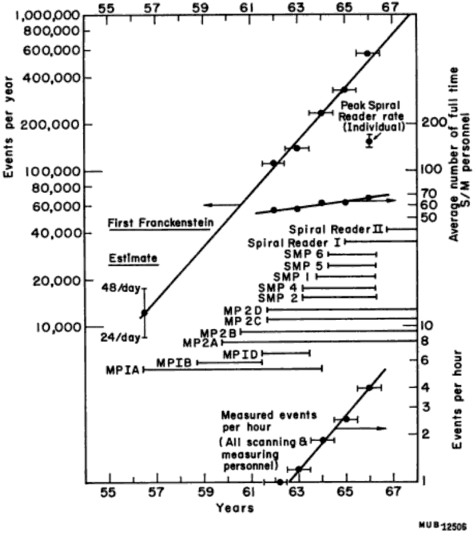
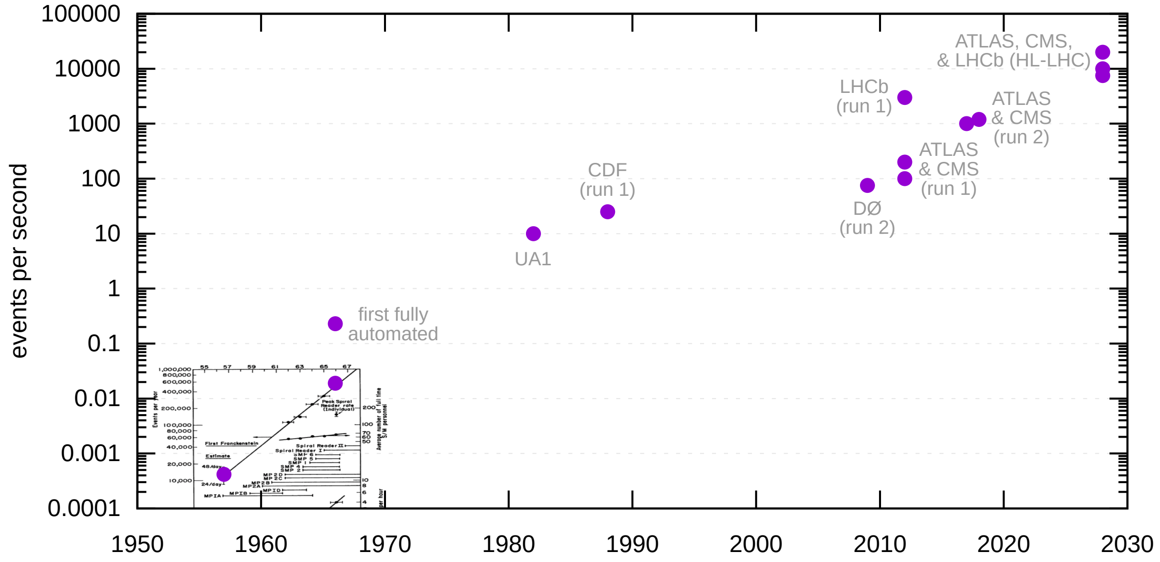


Fig. 9.
Measuring Rates.

Pattern recognition had to be automated to reach today's rates



Until recently, most HEP pattern-recognition consisted of hand-written heuristics, rather than ML (some still is).

Until recently, most HEP pattern-recognition consisted of hand-written heuristics, rather than ML (some still is).

The history of Artificial Intelligence (AI) is also split between what we would now call hand-written algorithms and learned algorithms.

Symbolic AI versus Connectionist AI

Symbolic

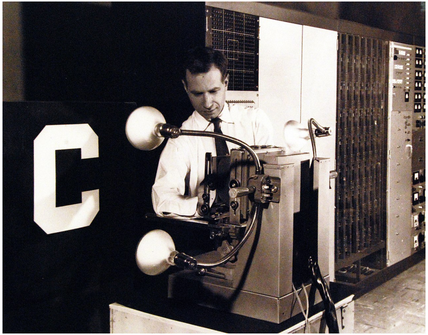
- Symbol manipulation and logic
- Searches through problem-space
- Hand-written common-sense rules

Examples: parsing, theorem-proving, chess-playing, expert systems

Connectionist

- Stimulus correlated to response only by strengths of internal connections
- No *explicit* symbols or rules
- *Effective* symbols/rules may arise

Examples: neural networks



Theory: Pitts & McCullock (1943).

Rosenblatt's perceptron machine (1958) attempted to recognize images of letters by adjusting free parameters with motors.

Made extravagant claims; reality hit hard.

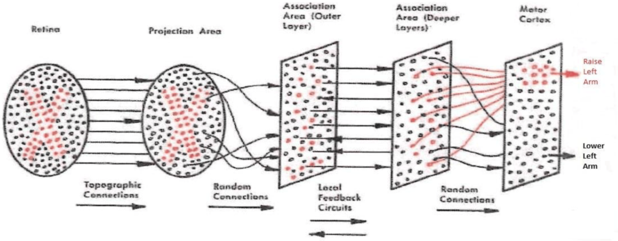


FIG. 1 — Organization of a biological brain. (Red areas indicate active cells, responding to the letter X.)

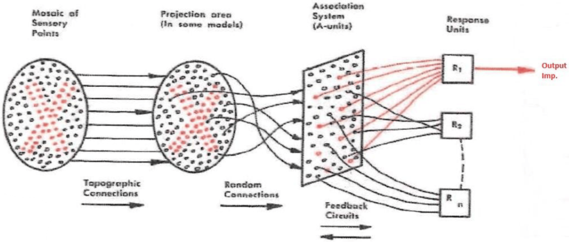
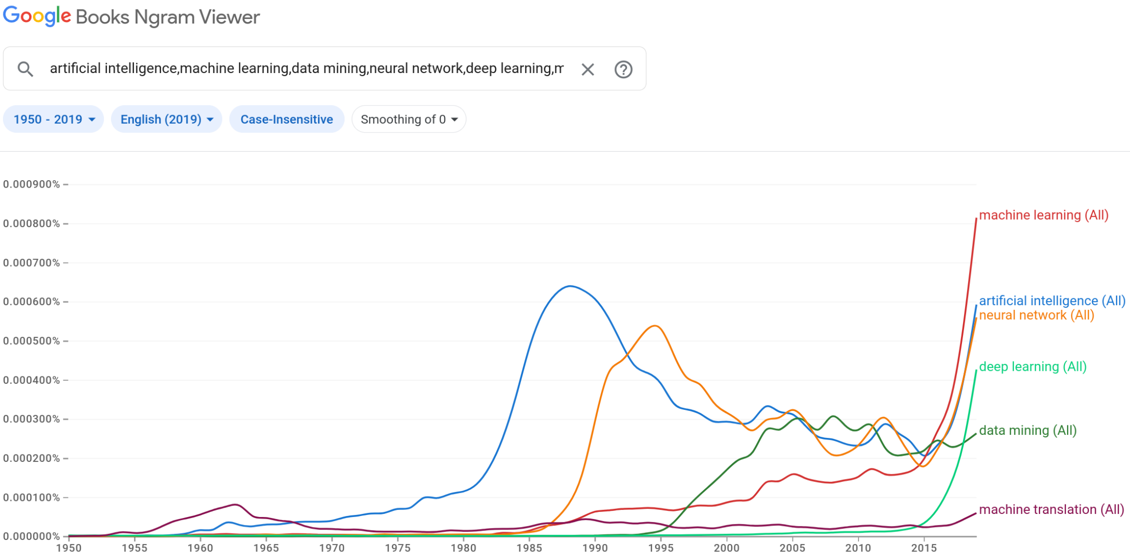
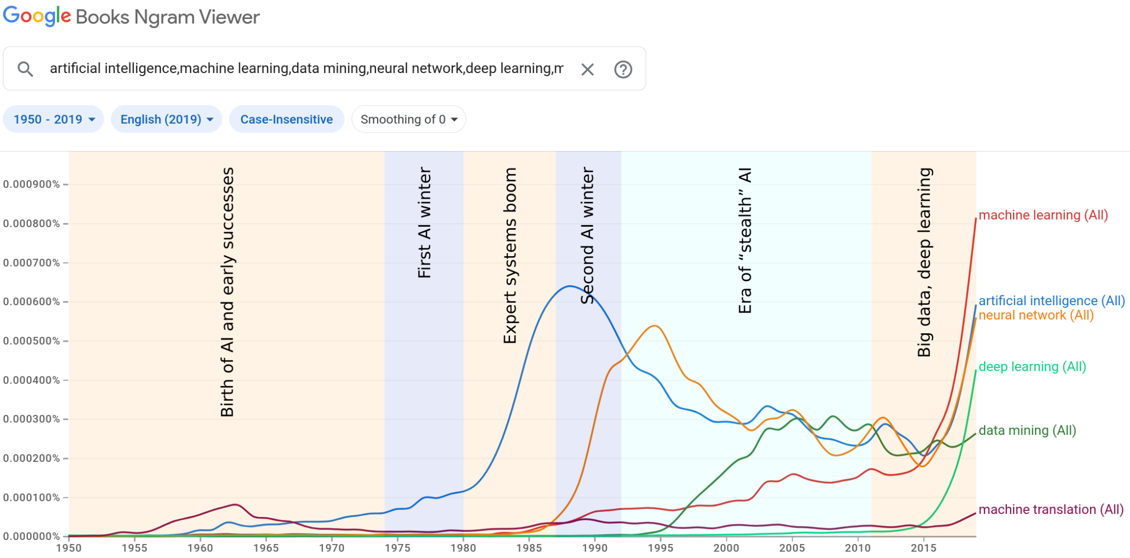


FIG. 2 — Organization of a perceptron.

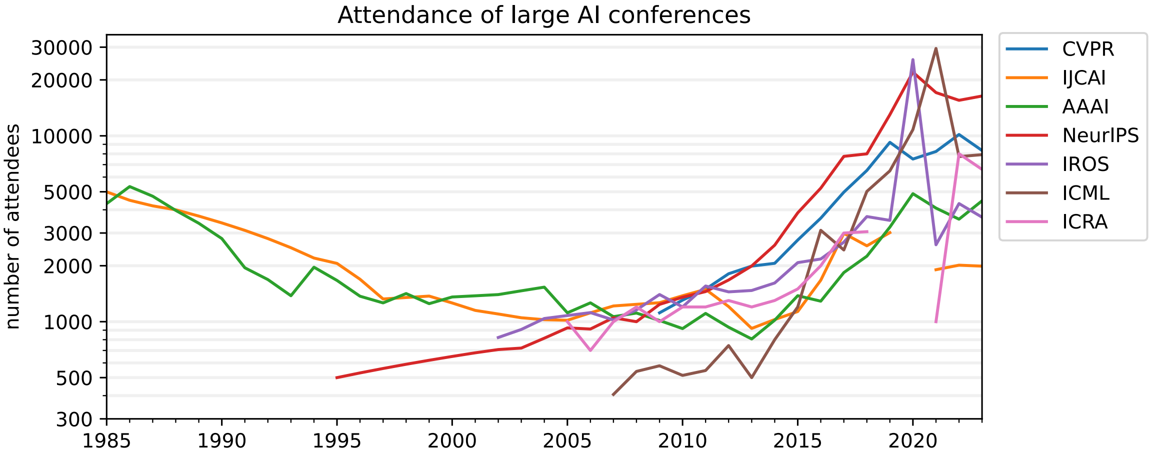
The ups and downs of AI: as mentioned in books



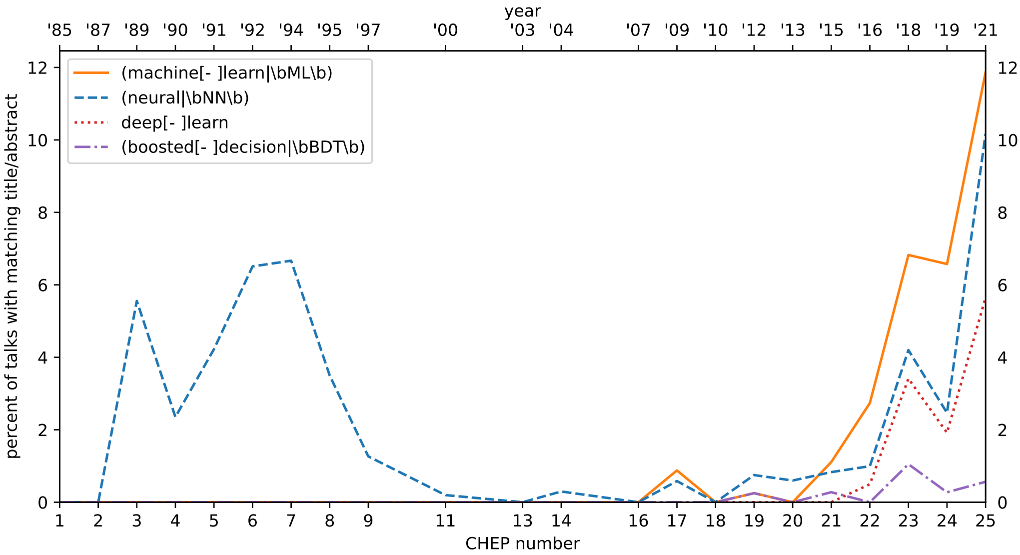
The ups and downs of AI: according to Henry Kautz (funding)



The ups and downs of AI: in conference attendance



The ups and downs of AI: among physicists at CHEP



Machine learning techniques that are not neural networks:

- Naive Bayes classifier
- k-nearest neighbors
- Principal Component Analysis (PCA)
- generalized additive models, LOWESS fitting
- decision trees, (boosted) random forests, AdaBoost
- k-means clustering, Gaussian processes, hierarchical clustering
- Support Vector Machines (SVMs)
- Hidden Markov Models (HMM)
- *and many more!*

Machine learning techniques that are not neural networks:

- Naive Bayes classifier
- k-nearest neighbors
- Principal Component Analysis (PCA)
- generalized additive models, LOWESS fitting
- decision trees, (boosted) random forests, AdaBoost
- k-means clustering, Gaussian processes, hierarchical clustering
- Support Vector Machines (SVMs)
- Hidden Markov Models (HMM)
- *and many more!*

(These are techniques I learned about and used when I was a data scientist, up to 2015, *just before* the deep learning boom.)

In this mini-course, we'll only cover neural networks.

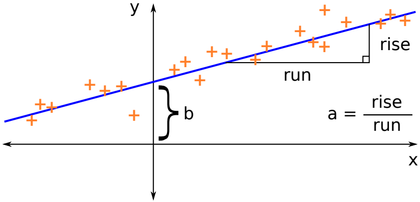
In this mini-course, we'll only cover neural networks.

(There's enough to talk about.)

The rest of this PDF talk: what is a neural network?

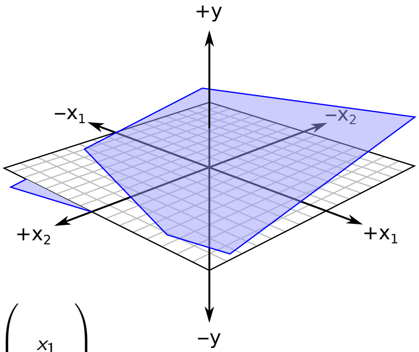
Switch to Jupyter: why does a neural network work?

Simplest neural network is a linear fit



$$\underbrace{a}_{\text{free parameters in the fit}} \cdot \underbrace{x}_{\text{input values}} + \underbrace{b}_{\text{free parameters}} = \underbrace{y}_{\text{output values}} = ax + b$$

Simplest neural network is a linear fit



$$\underbrace{\begin{pmatrix} a_1 & a_2 \end{pmatrix}}_{\text{free parameters in the fit}} \cdot \underbrace{\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}}_{\text{input values}} + \underbrace{b}_{\text{free parameters}} = \underbrace{y}_{\text{output values}} = a_1 x_1 + a_2 x_2 + b$$

Simplest neural network is a linear fit

$$\underbrace{\begin{pmatrix} a_1 & a_2 & \dots & a_{10} \end{pmatrix}}_{\text{free parameters in the fit}} \cdot \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{10} \end{pmatrix}}_{\text{input values}} + \underbrace{b}_{\text{free parameters}} = \underbrace{y}_{\text{output values}} = a_1 x_1 + a_2 x_2 + \dots a_{10} x_{10} + b$$

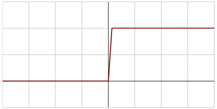
Simplest neural network is a linear fit

$$\underbrace{\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,10} \\ a_{2,1} & a_{2,2} & \dots & a_{2,10} \\ a_{3,1} & a_{3,2} & \dots & a_{3,10} \\ a_{4,1} & a_{4,2} & \dots & a_{4,10} \\ a_{5,1} & a_{5,2} & \dots & a_{5,10} \end{pmatrix}}_{\text{free parameters in the fit}} \cdot \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{10} \end{pmatrix}}_{\text{input values}} + \underbrace{\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix}}_{\text{free parameters}} = \underbrace{\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix}}_{\text{output values}} = \begin{matrix} a_{1,1}x_1 + a_{1,2}x_2 + \dots a_{1,10}x_{10} + b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \dots a_{2,10}x_{10} + b_2 \\ a_{3,1}x_1 + a_{3,2}x_2 + \dots a_{3,10}x_{10} + b_3 \\ a_{4,1}x_1 + a_{4,2}x_2 + \dots a_{4,10}x_{10} + b_4 \\ a_{5,1}x_1 + a_{5,2}x_2 + \dots a_{5,10}x_{10} + b_5 \end{matrix}$$

Next-simplest passes it through a non-linear function f

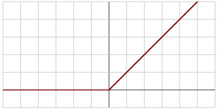
$$
f\left[\underbrace{\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,10} \\ a_{2,1} & a_{2,2} & \dots & a_{2,10} \\ a_{3,1} & a_{3,2} & \dots & a_{3,10} \\ a_{4,1} & a_{4,2} & \dots & a_{4,10} \\ a_{5,1} & a_{5,2} & \dots & a_{5,10} \end{pmatrix}}_{\text{free parameters in the fit}} \cdot \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{10} \end{pmatrix}}_{\text{input values}} + \underbrace{\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix}}_{\text{free parameters}}\right] = \underbrace{\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix}}_{\text{output values}} = \begin{matrix} f[a_{1,1}x_1 + a_{1,2}x_2 + \dots a_{1,10}x_{10} + b_1] \\ f[a_{2,1}x_1 + a_{2,2}x_2 + \dots a_{2,10}x_{10} + b_2] \\ f[a_{3,1}x_1 + a_{3,2}x_2 + \dots a_{3,10}x_{10} + b_3] \\ f[a_{4,1}x_1 + a_{4,2}x_2 + \dots a_{4,10}x_{10} + b_4] \\ f[a_{5,1}x_1 + a_{5,2}x_2 + \dots a_{5,10}x_{10} + b_5] \end{matrix}
$$

The non-linear function f is called an "activation function"



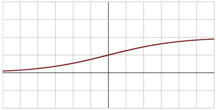
binary step

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$



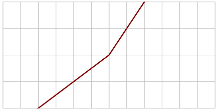
rectified linear unit (ReLU)

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$



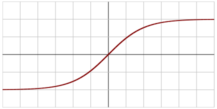
logistic (soft step)

$$f(x) = \frac{1}{1 + e^{-x}}$$



"leaky" ReLU

$$f(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$



hyperbolic tangent

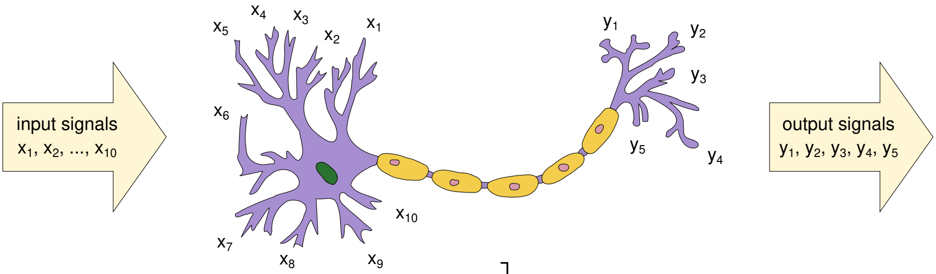
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

sigmoid linear unit ("swish")

$$f(x) = \frac{x}{1 + e^{-x}}$$

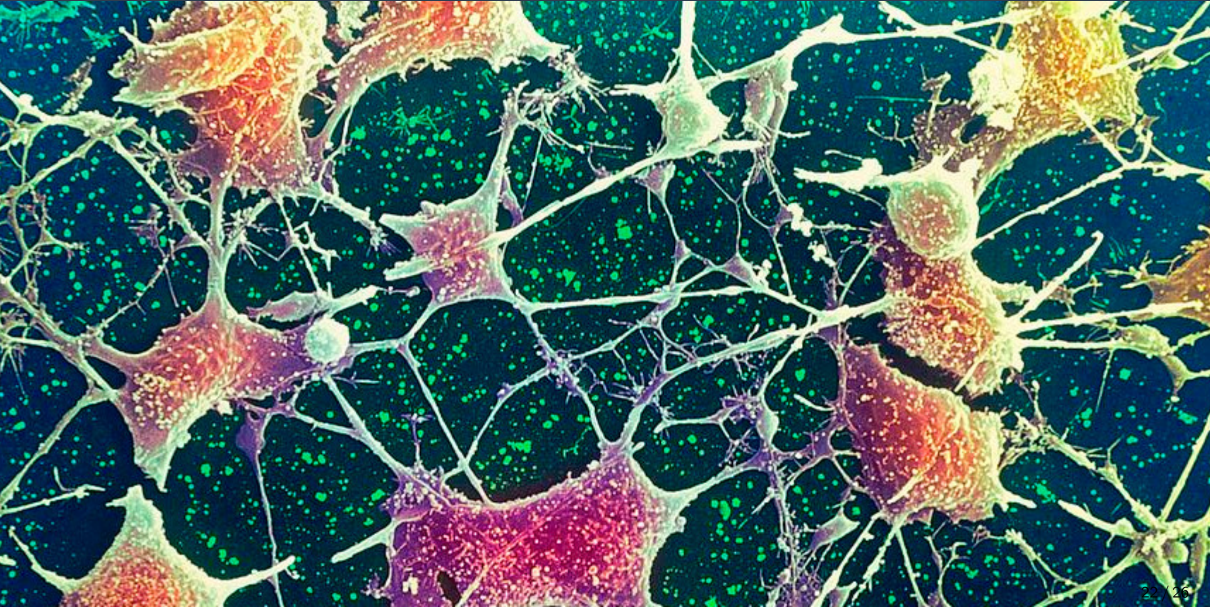
There are many choices, but ReLU is the simplest and most common.

Neural networks take inspiration from neurons in the brain

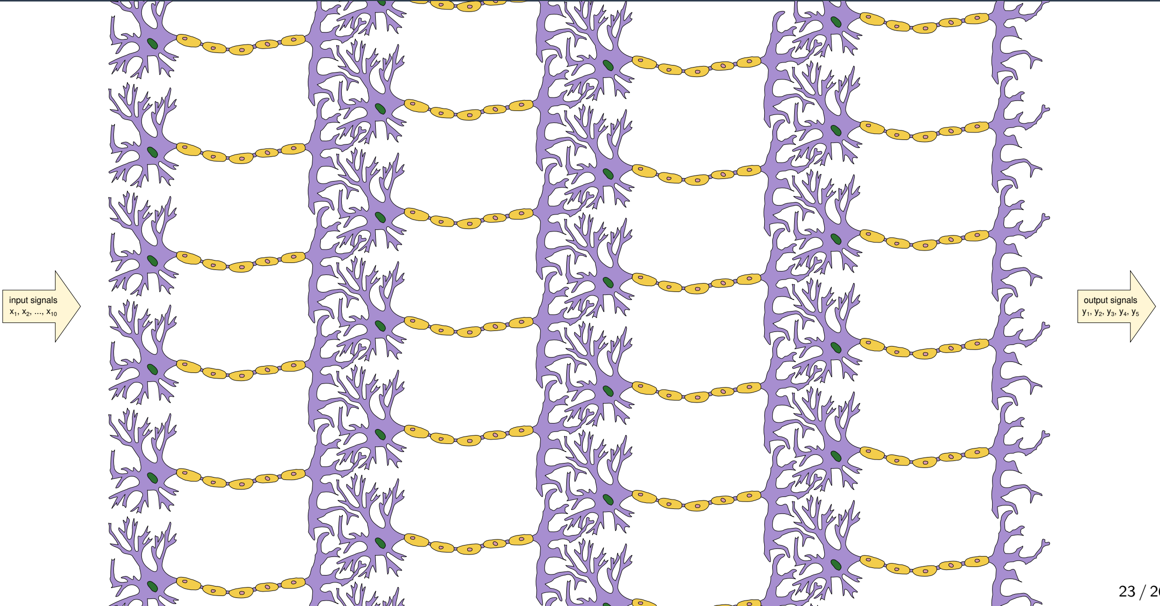


$$f\left[\underbrace{\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,10} \\ a_{2,1} & a_{2,2} & \dots & a_{2,10} \\ a_{3,1} & a_{3,2} & \dots & a_{3,10} \\ a_{4,1} & a_{4,2} & \dots & a_{4,10} \\ a_{5,1} & a_{5,2} & \dots & a_{5,10} \end{pmatrix}}_{\text{free parameters in the fit}} \cdot \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{10} \end{pmatrix}}_{\text{input values}} + \underbrace{\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix}}_{\text{free parameters}}\right] = \underbrace{\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix}}_{\text{output values}} = \begin{matrix} f[a_{1,1}x_1 + a_{1,2}x_2 + \dots a_{1,10}x_{10} + b_1] \\ f[a_{2,1}x_1 + a_{2,2}x_2 + \dots a_{2,10}x_{10} + b_2] \\ f[a_{3,1}x_1 + a_{3,2}x_2 + \dots a_{3,10}x_{10} + b_3] \\ f[a_{4,1}x_1 + a_{4,2}x_2 + \dots a_{4,10}x_{10} + b_4] \\ f[a_{5,1}x_1 + a_{5,2}x_2 + \dots a_{5,10}x_{10} + b_5] \end{matrix}$$

Neural networks take inspiration from neurons in the brain



Neural networks take inspiration from neurons in the brain



To do the same thing with our model, take the output of one "activation + linear transform" and use it as the input to the next:

$$f\left(a_{i,j}^{\text{layer 1}} \cdot x_j + b_i^{\text{layer 1}}\right)$$

Neural networks take inspiration from neurons in the brain

To do the same thing with our model, take the output of one "activation + linear transform" and use it as the input to the next:

$$f\left(a_{i,j}^{\text{layer 2}} \cdot \boxed{f\left(a_{i,j}^{\text{layer 1}} \cdot x_j + b_i^{\text{layer 1}}\right)} + b_i^{\text{layer 2}}\right)$$

Neural networks take inspiration from neurons in the brain

To do the same thing with our model, take the output of one "activation + linear transform" and use it as the input to the next:

$$f\left(a_{i,j}^{\text{layer 3}} \cdot \boxed{f\left(a_{i,j}^{\text{layer 2}} \cdot \boxed{f\left(a_{i,j}^{\text{layer 1}} \cdot x_j + b_i^{\text{layer 1}}\right)} + b_i^{\text{layer 2}}\right)} + b_i^{\text{layer 3}}\right)$$

Neural networks take inspiration from neurons in the brain

To do the same thing with our model, take the output of one "activation + linear transform" and use it as the input to the next:

$$f\left(a_{i,j}^{\text{layer 4}} \cdot \boxed{f\left(a_{i,j}^{\text{layer 3}} \cdot \boxed{f\left(a_{i,j}^{\text{layer 2}} \cdot \boxed{f\left(a_{i,j}^{\text{layer 1}} \cdot x_j + b_i^{\text{layer 1}}\right)} + b_i^{\text{layer 2}}\right)} + b_i^{\text{layer 3}}\right)} + b_i^{\text{layer 4}}\right)$$

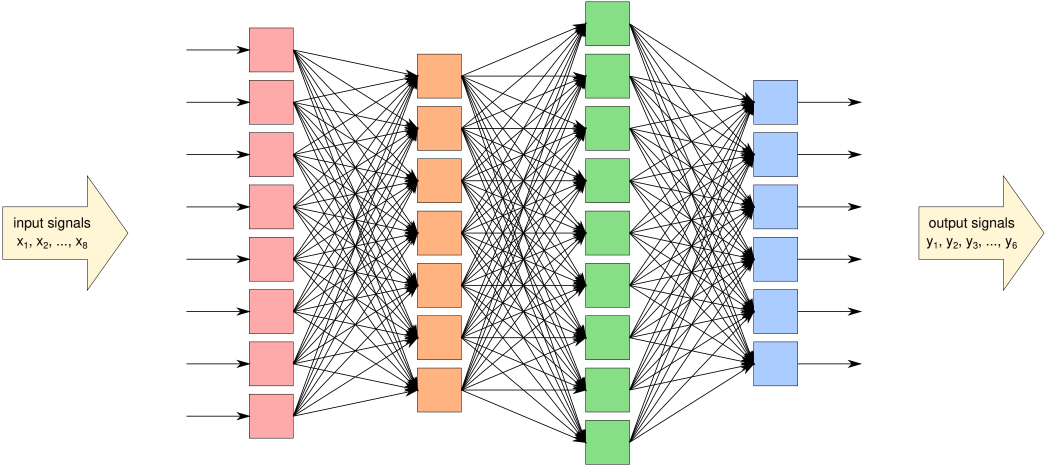
Neural networks take inspiration from neurons in the brain

To do the same thing with our model, take the output of one "activation + linear transform" and use it as the input to the next:

$$f\left(a_{i,j}^{\text{layer 4}} \cdot \boxed{f\left(a_{i,j}^{\text{layer 3}} \cdot \boxed{f\left(a_{i,j}^{\text{layer 2}} \cdot \boxed{f\left(a_{i,j}^{\text{layer 1}} \cdot x_j + b_i^{\text{layer 1}}\right)} + b_i^{\text{layer 2}}\right)} + b_i^{\text{layer 3}}\right)} + b_i^{\text{layer 4}}\right)$$

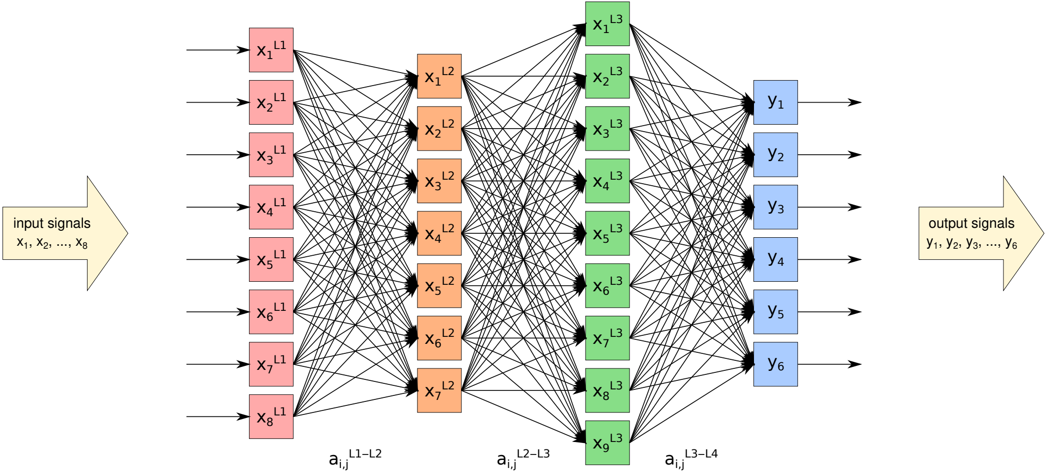
Without the activation functions, we'd lose the structure: linear transformations of linear transformations collapse down to a single linear transformation.

It's usually drawn like this



The lines indicate that every output from one layer is included in the linear transformation of the next layer. ("There's an $a_{i,j}$ for every x_j and y_i.")

It's usually drawn like this



The lines indicate that every output from one layer is included in the linear transformation of the next layer. ("There's an $a_{i,\,j}$ for every x_j and y_i.")

But. . . why does that work?

What's so special about this linear-nonlinear sandwich?

But. . . why does that work?

What's so special about this linear-nonlinear sandwich?

(Time to switch to Jupyter.)