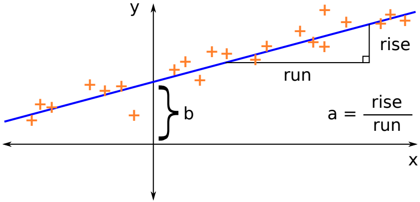
What is a neural network?

Jim Pivarski

Princeton University – IRIS-HEP

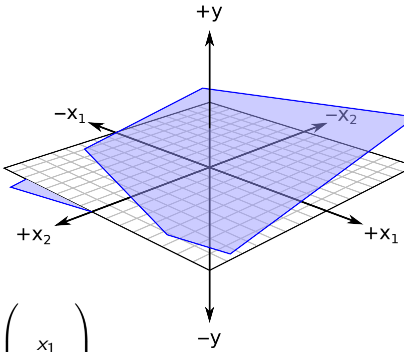
July 8, 2024

Equation of a line (fitting *a* and *b* to measurements *y* versus *x*)



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

Equation of a plane (height y versus 2D coordinates x_1 and x_2)



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

Equation of a hyperplane (N-dimensional)

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

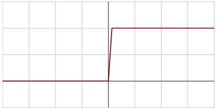
General linear transformation: many inputs, many outputs

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

Pass through function f to make it non-linear

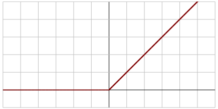
$$
f\left[\underbrace{\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,10} \\ a_{2,1} & a_{2,2} & \dots & a_{2,10} \\ a_{3,1} & a_{3,2} & \dots & a_{3,10} \\ a_{4,1} & a_{4,2} & \dots & a_{4,10} \\ a_{5,1} & a_{5,2} & \dots & a_{5,10} \end{pmatrix}}_{\text{free parameters in the fit}} \cdot \underbrace{\begin{pmatrix}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

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



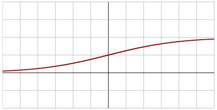
binary step

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



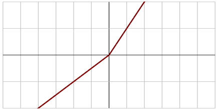
rectified linear unit (ReLU)

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



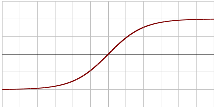
logistic (soft step)

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



"leaky" ReLU

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



hyperbolic tangent

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

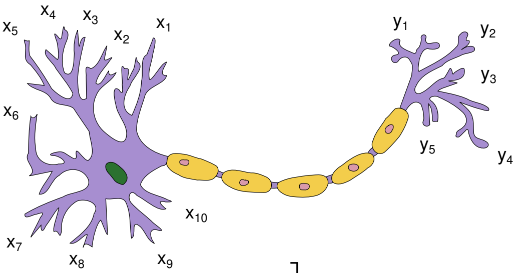
sigmoid linear unit ("swish")

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

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

Neural networks take inspiration from neurons in the brain

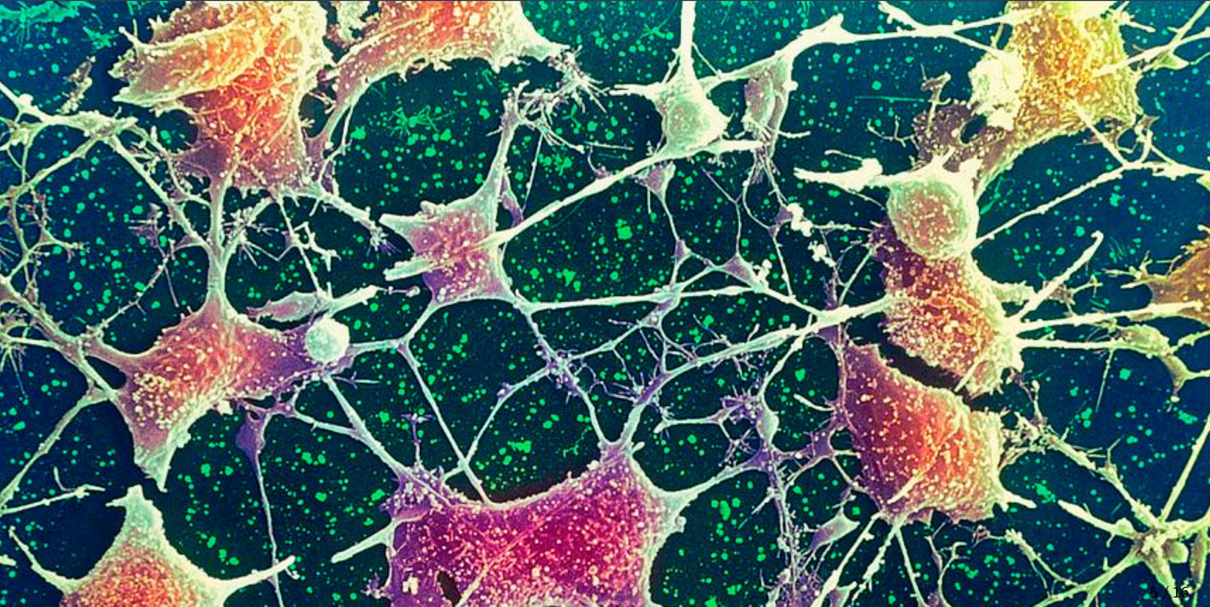
input signals
$x_1, x_2, \ldots, x_{10}$



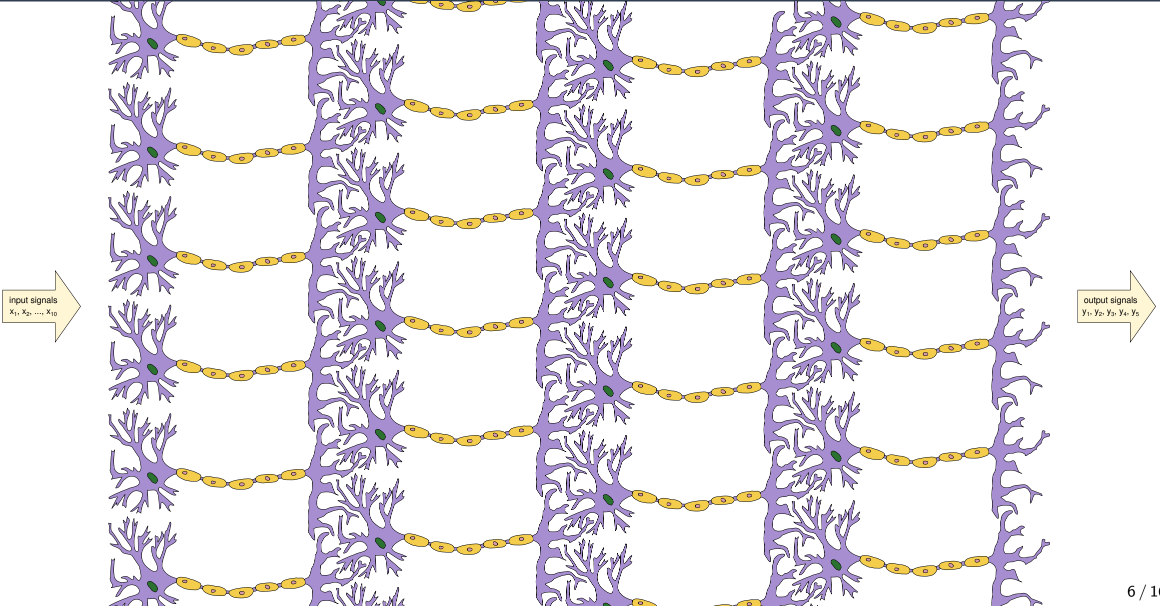
output signals
y_1, y_2, y_3, y_4, y_5

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

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:

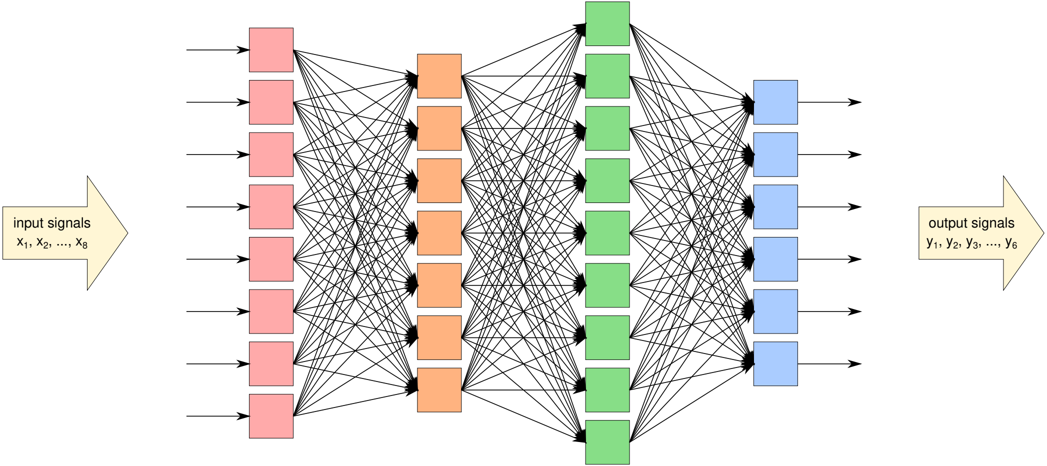
$$f\left(a_{i,j}^{\text{layer 3}} \cdot \boxed{f\left(a_{i,j}^{\text{layer 2}} \cdot \boxed{f\left(a_{i,j}^{\text{layer 1}} \cdot x_j + b_i^{\text{layer 1}}\right)} + b_i^{\text{layer 2}}\right)} + b_i^{\text{layer 3}}\right)$$

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:

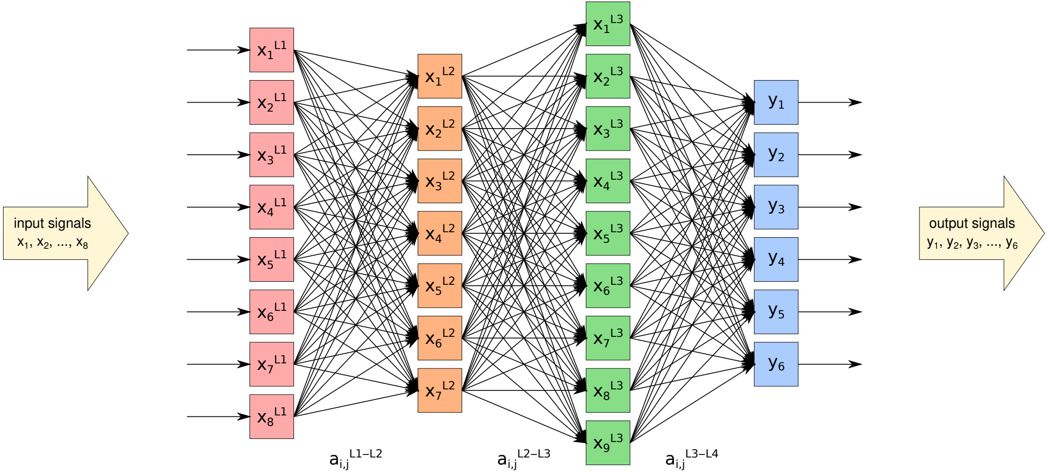
$$f\left(a_{i,j}^{\text{layer 4}} \cdot \boxed{f\left(a_{i,j}^{\text{layer 3}} \cdot \boxed{f\left(a_{i,j}^{\text{layer 2}} \cdot \boxed{f\left(a_{i,j}^{\text{layer 1}} \cdot x_j + b_i^{\text{layer 1}}\right)} + b_i^{\text{layer 2}}\right)} + b_i^{\text{layer 3}}\right)} + b_i^{\text{layer 4}}\right)$$

It's usually drawn like this



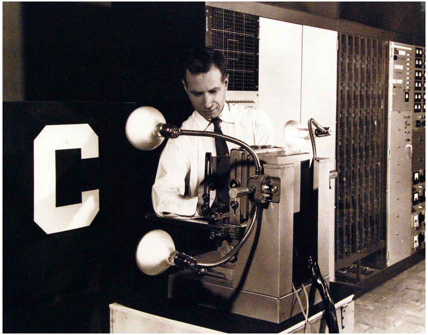
The lines indicate that every output from one layer is included in the linear transformation of the next layer. ("There's an $a_{i,j}$ for every x_j and y_i.")

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.")

Connectivism: automatic learning by fitting the $a_{i,j}$, b_i parameters



Frank Rosenblatt's perceptron machine (1958) attempted to recognize images of letters.

The free parameters were adjusted with motors. This system eventually learned left-versus-right (not much more).

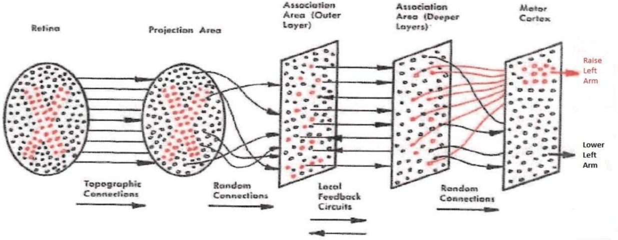


FIG. 1 — Organization of a biological brain. (Red areas indicate active cells, responding to the letter X.)

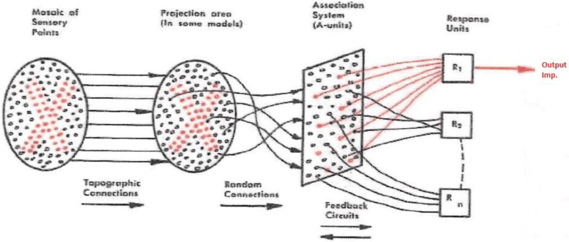
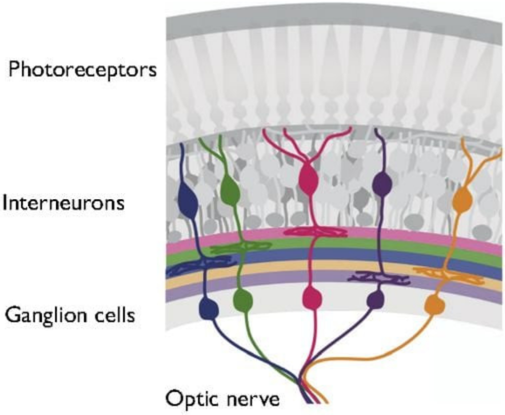


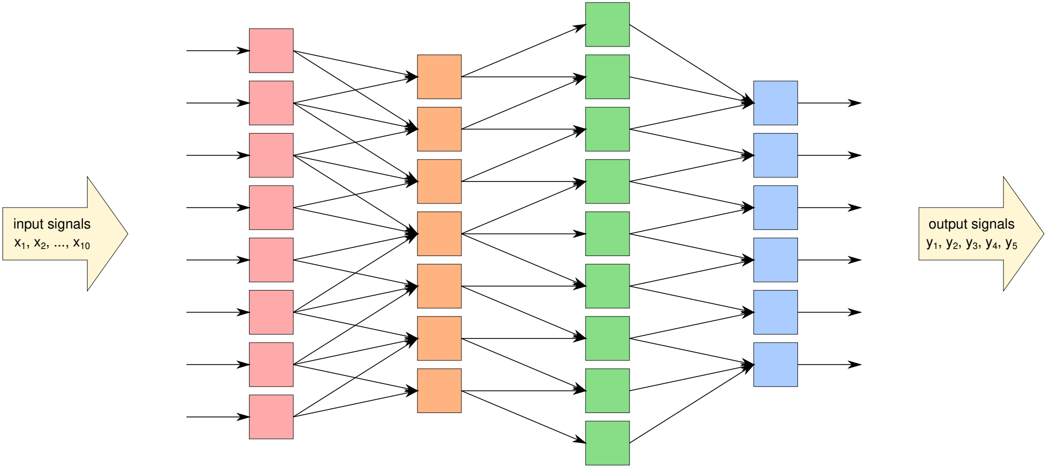
FIG. 2 — Organization of a perceptron.

More inspiration from nature: eye-neurons are not fully connected



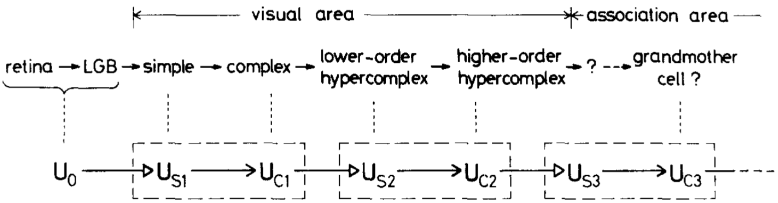
Neurons in one layer are connected to only a few of the neurons in the next layer.

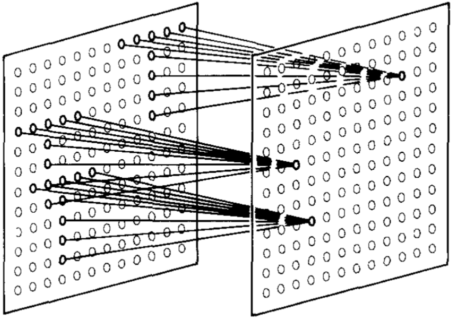
“Convolutional” (restricted) neural network

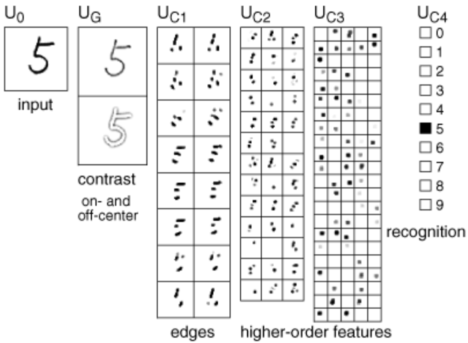


This sets a lot of $a_{i,j}$ parameters to zero and doesn't let them be tuned in the fit.

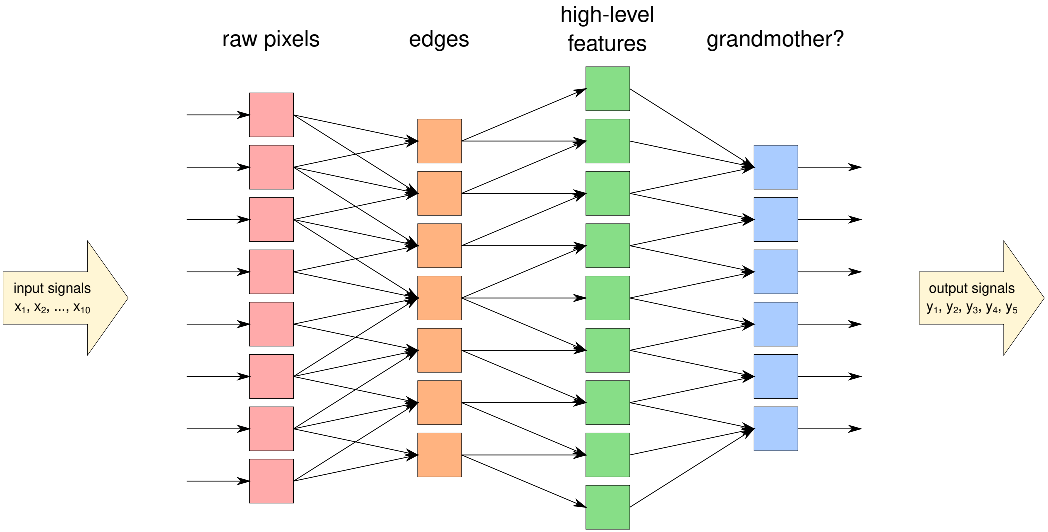
Kunihiko Fukushima's neocognitron (1980)



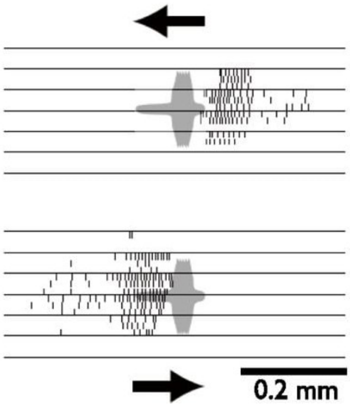




Each layer of a neural network is more abstract than the last

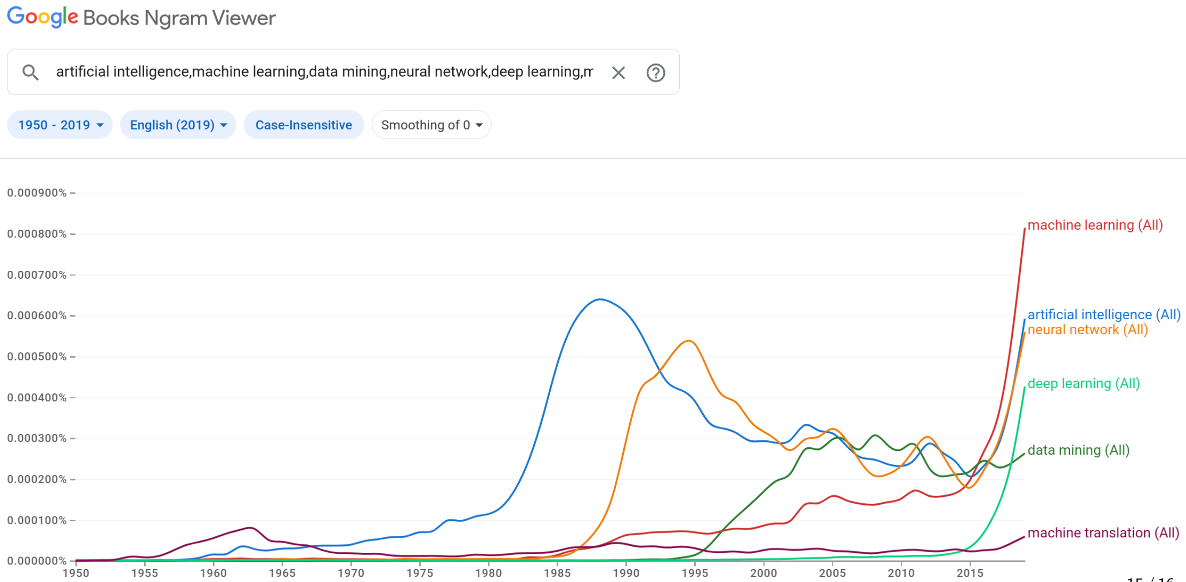


Fun fact: mice have hawk-shaped features in their visual networks



Y. Zhang, I. Kim, J. Sanes, *The most numerous ganglion cell type of the mouse retina is a selective feature detector* (2012), https://doi.org/10.1073/pnas.1211547109

These ideas have been studied for a long time



Why are neural networks only a big thing now?

1. At first, neural networks were much worse than "symbolic" (rule-based) approaches. Learning without explicit rules seemed like magical thinking or hype.

Why are neural networks only a big thing now?

1. At first, neural networks were much worse than “symbolic” (rule-based) approaches. Learning without explicit rules seemed like magical thinking or hype.

2. Important algorithmic discoveries: gradient descent, backpropagation, ReLU activation, convolutional and recurrent topologies, Xavier initialization, transfer learning, batch normalization, dropout, the attention mechanism, ...

Why are neural networks only a big thing now?

1. At first, neural networks were much worse than "symbolic" (rule-based) approaches. Learning without explicit rules seemed like magical thinking or hype.

2. Important algorithmic discoveries: gradient descent, backpropagation, ReLU activation, convolutional and recurrent topologies, Xavier initialization, transfer learning, batch normalization, dropout, the attention mechanism, . . .

3. Large enough datasets, already digitized, to expose all the nuances of human-like behaviors: the world wide web.

Why are neural networks only a big thing now?

1. At first, neural networks were much worse than "symbolic" (rule-based) approaches. Learning without explicit rules seemed like magical thinking or hype.
2. Important algorithmic discoveries: gradient descent, backpropagation, ReLU activation, convolutional and recurrent topologies, Xavier initialization, transfer learning, batch normalization, dropout, the attention mechanism, ...
3. Large enough datasets, already digitized, to expose all the nuances of human-like behaviors: the world wide web.
4. Large enough compute farms and GPUs to analyze the above.