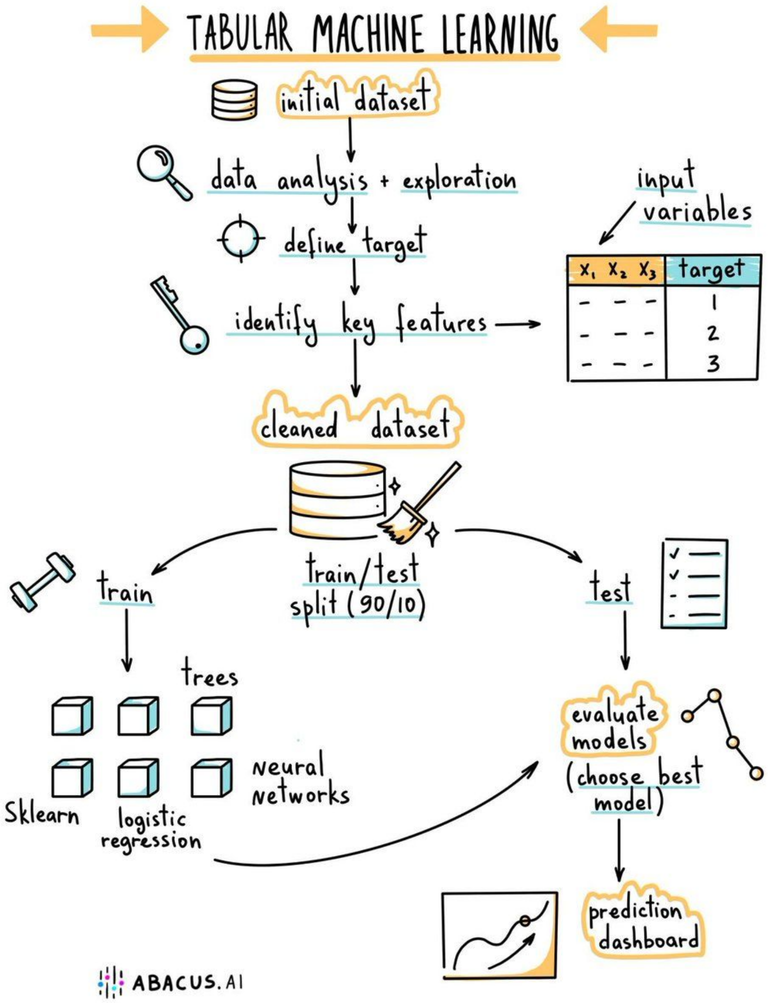
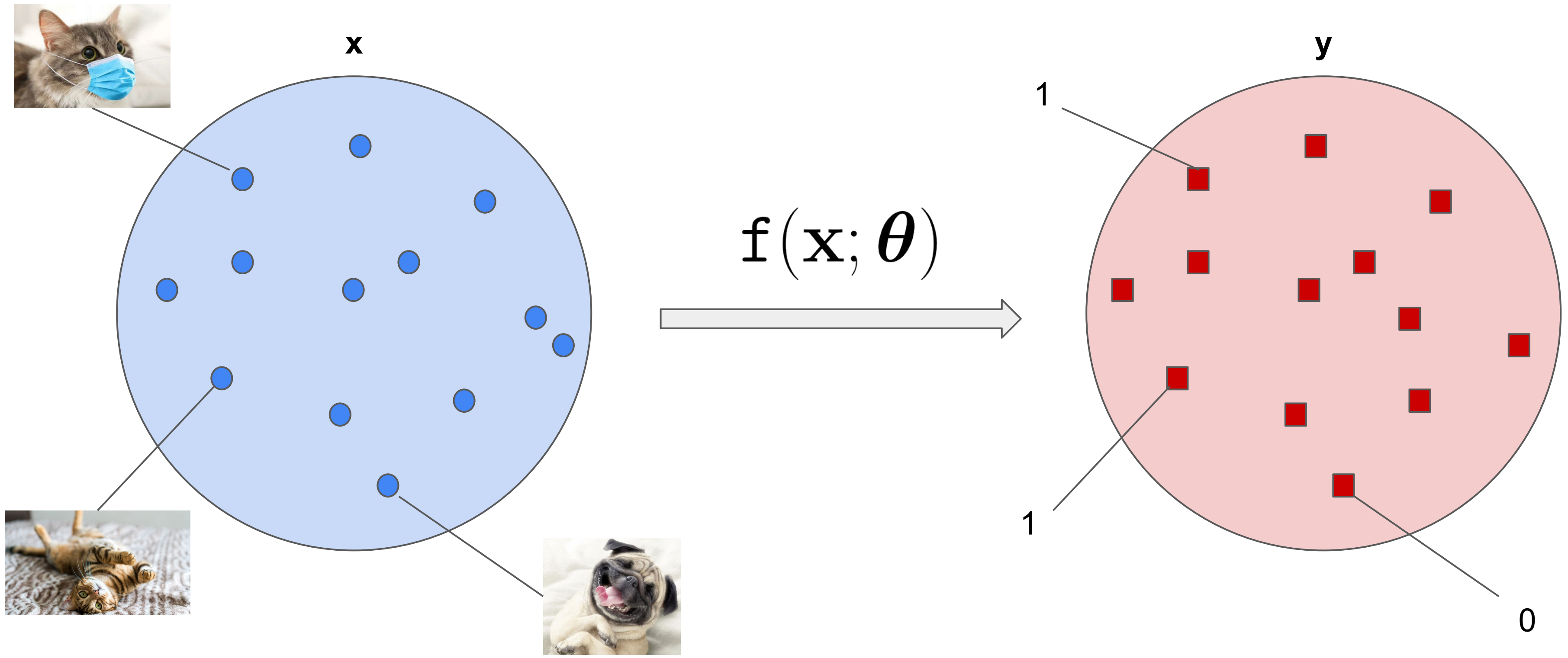
Neural Networks

University of Victoria - PHYS 555

previously…



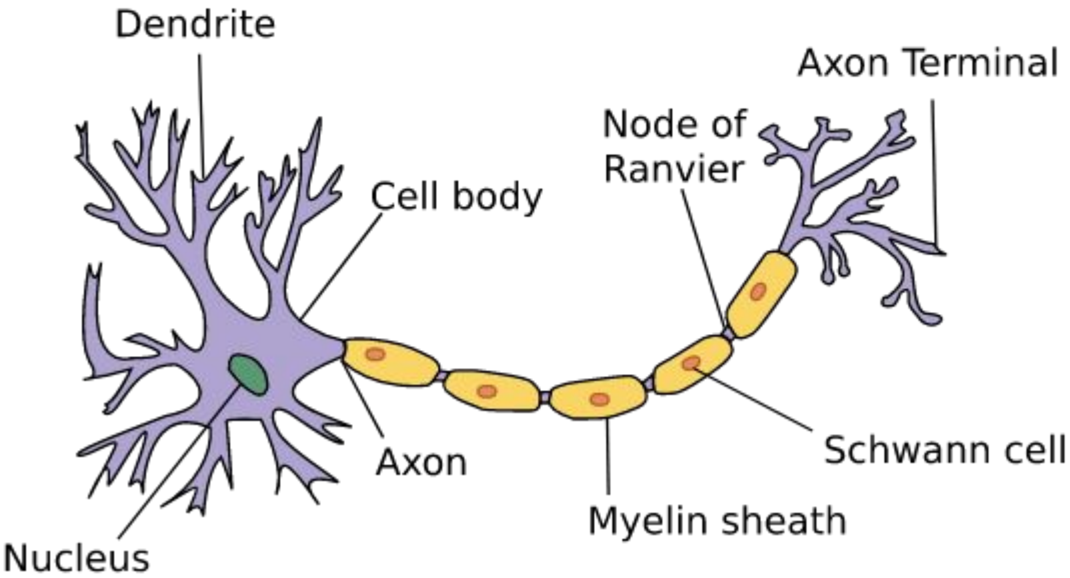
TABULAR MACHINE LEARNING
initial dataset
data analysis + exploration
define target
identify key features
input variables
x1 x2 x3 target
1
2
3
cleaned dataset
train
train/test split (90/10)
test
trees
Sklearn
logistic regression
neural networks
evaluate models
(choose best model)
prediction dashboard
ABACUS.AI



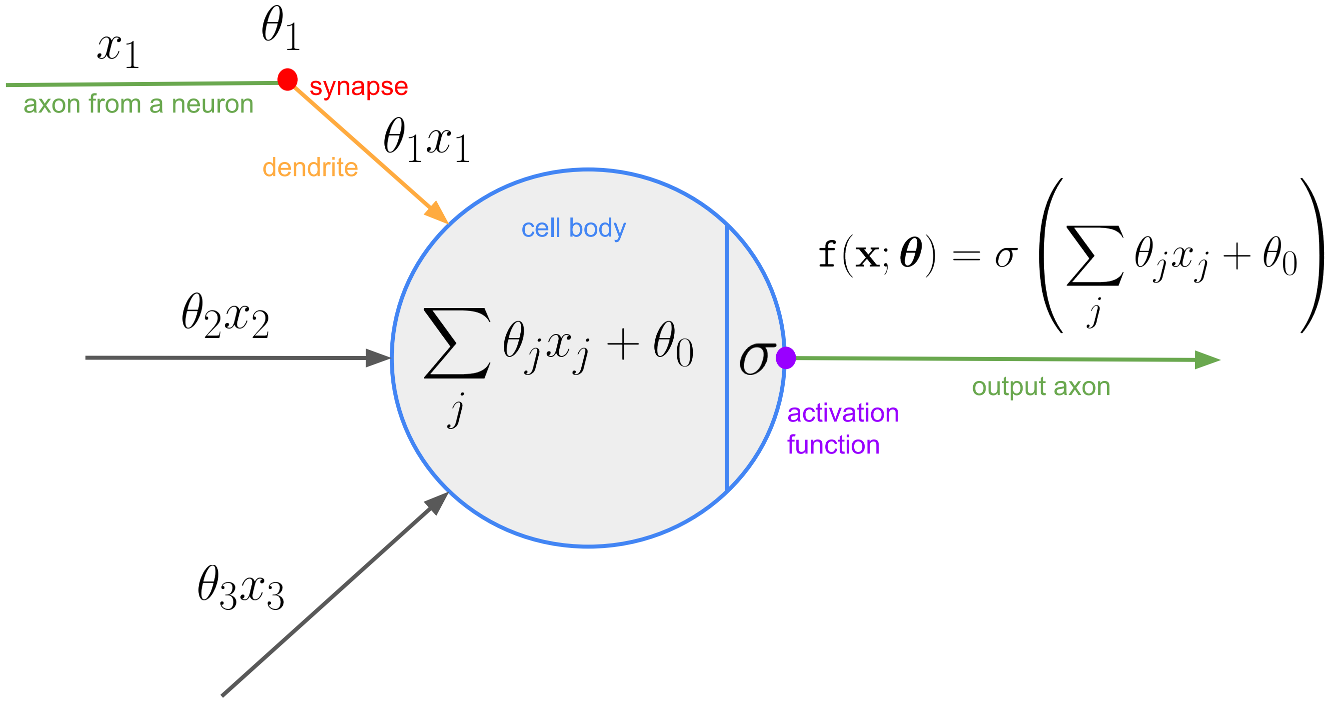
x
y
f(x; θ)
1
1
0

simplest function ?

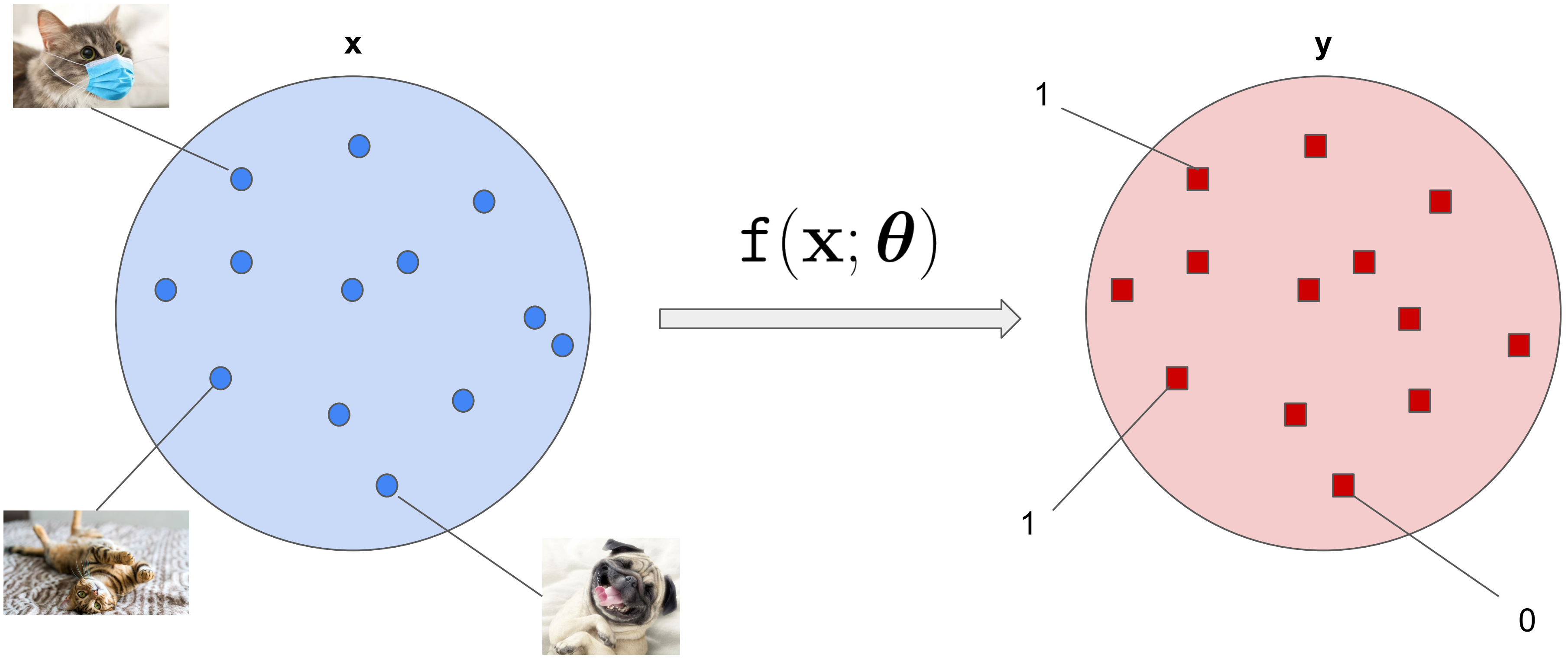
a neuron


Dendrite
Axon Terminal
Node of
Ranvier
Cell body
Axon
Schwann cell
Myelin sheath
Nucleus

lose analogy

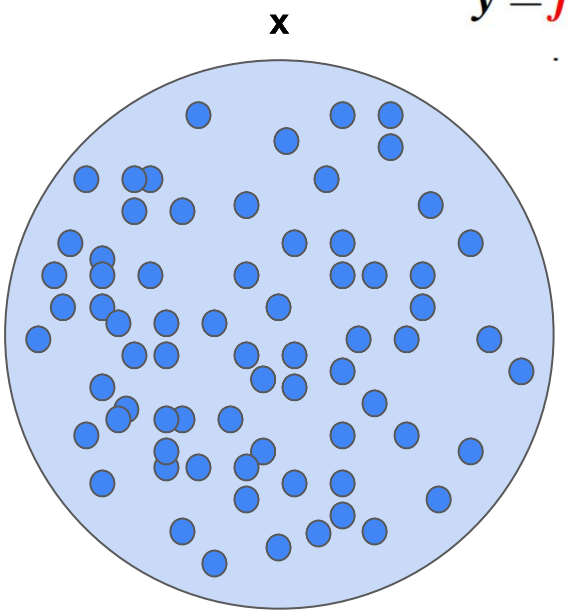


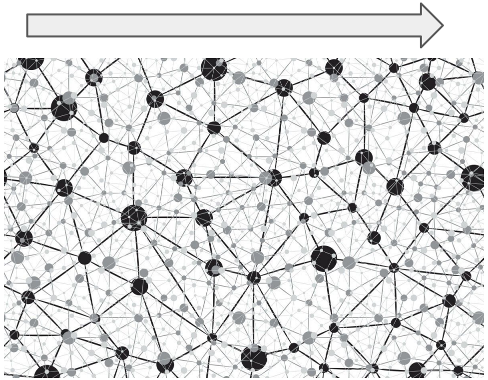
a neural network is a piecewise functional mapping…

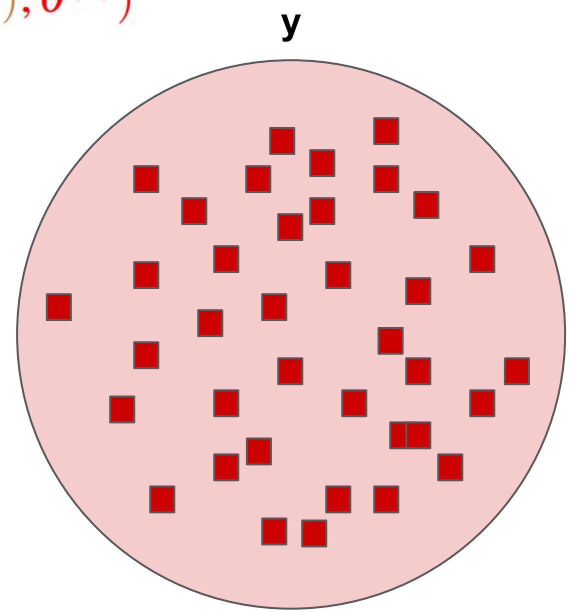


composition and scaling properties

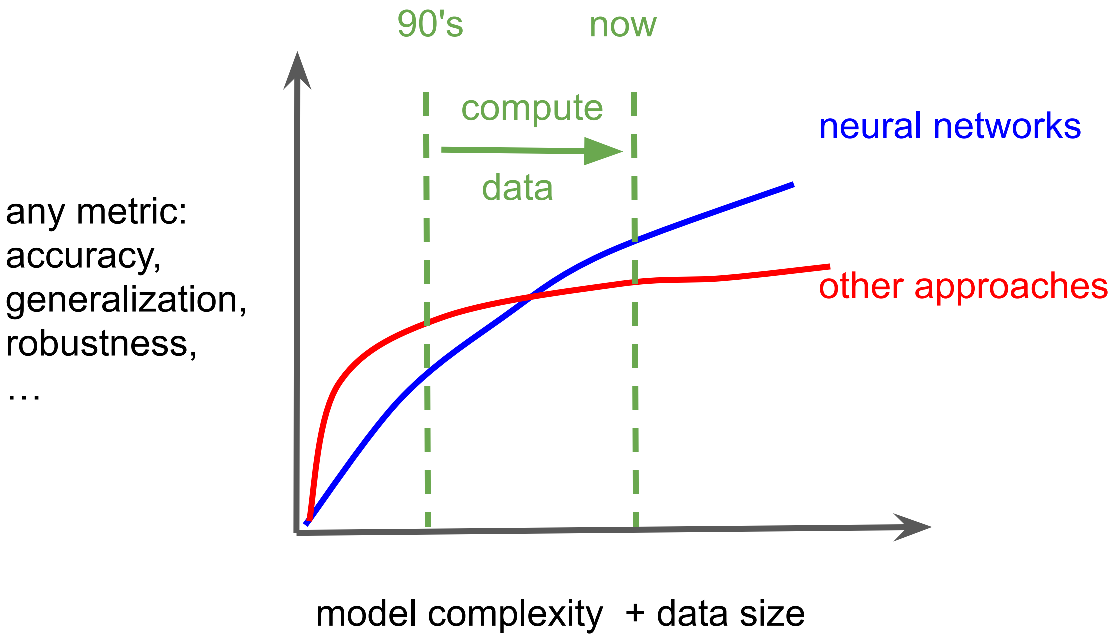
$$\hat{\boldsymbol{y}} = \boldsymbol{f}^{(L)}\left(\cdots \boldsymbol{f}^{(2)}\left(\boldsymbol{f}^{(1)}\left(\boldsymbol{x};\boldsymbol{\theta}^{(1)}\right);\boldsymbol{\theta}^{(2)}\right);\boldsymbol{\theta}^{(L)}\right)$$





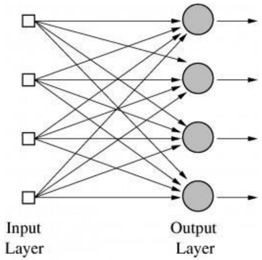


a gloomy future



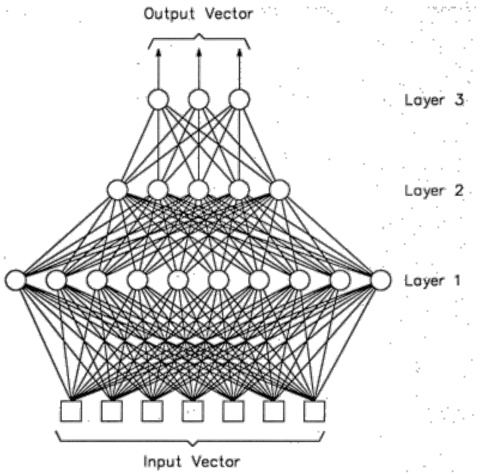
From a single layer to multiple layers

- 1 perceptron = 1 decision

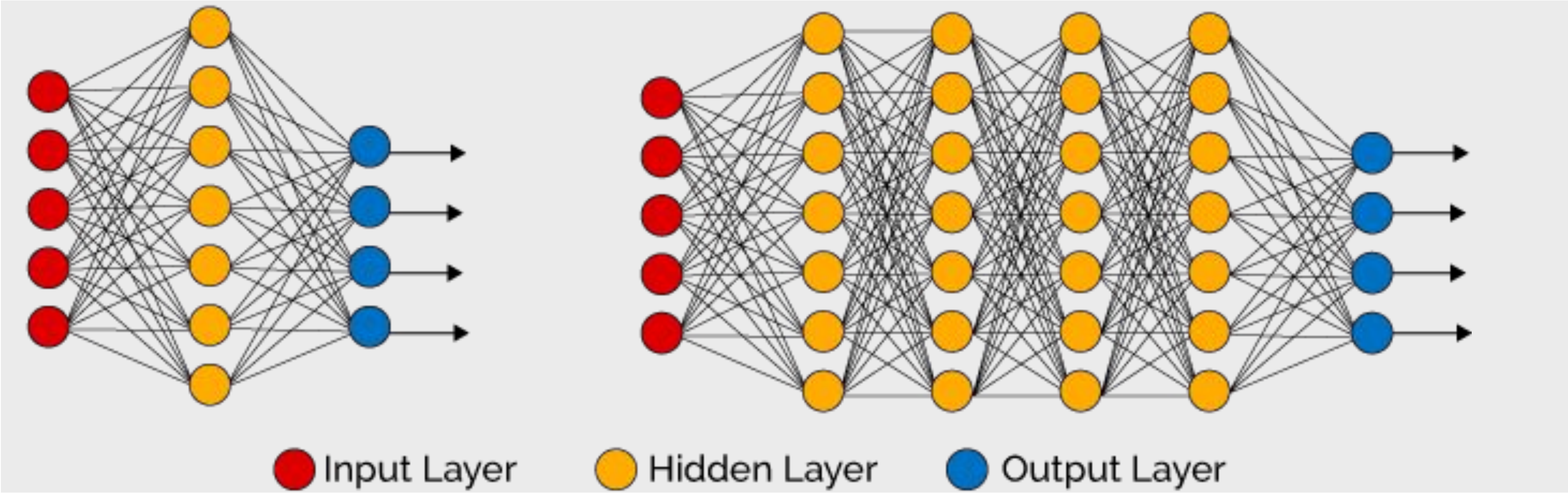


What about multiple decisions?

- Add layers → neural network
- MLP = Multi Layer Perceptron
  - one layer as input to the next
  - non-linearities between layers



Should we keep adding layers?



Universal approximation theorem

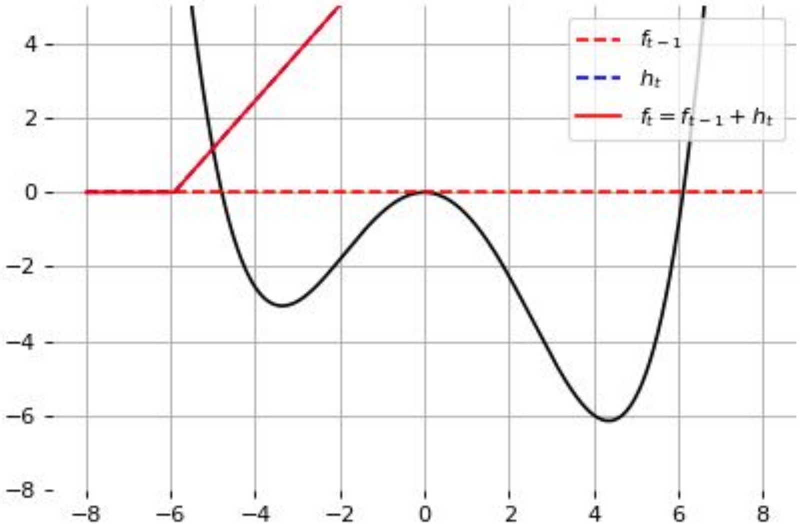
A single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units

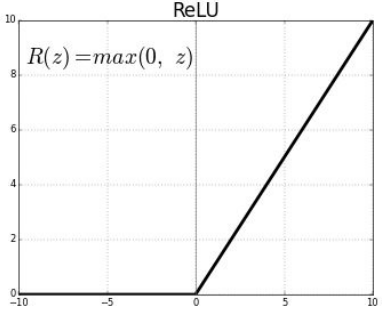
Hornik, 1991

Consider the 1-layer MLP: $f(x) = \sum w_i \mathrm{ReLU}(x + b_i).$

This model can approximate smooth 1D function, provided enough hidden units.

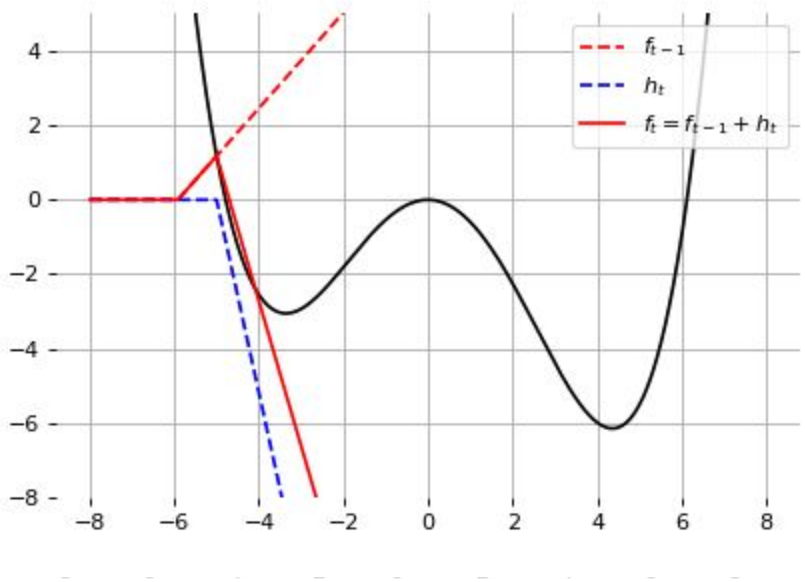
Approximating a function with ReLU





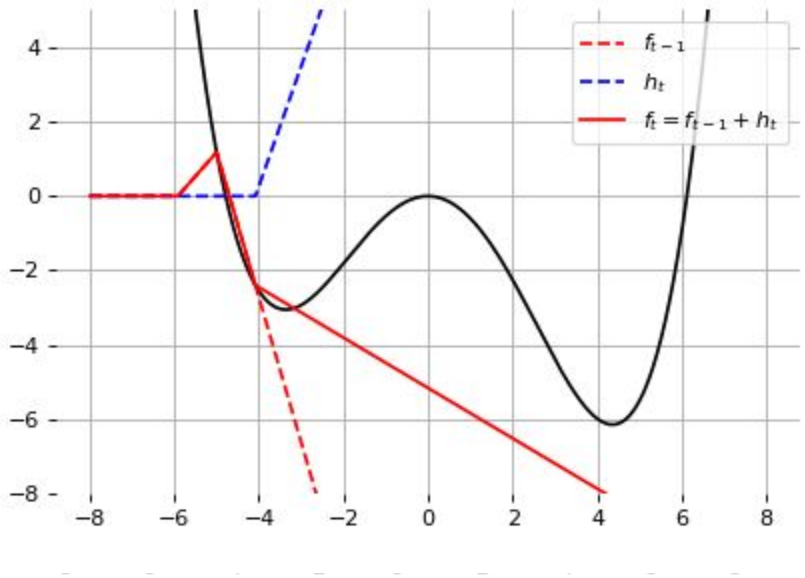
$$f(x) = \sum w_i \mathrm{ReLU}(x + b_i)$$

Approximating a function with ReLU



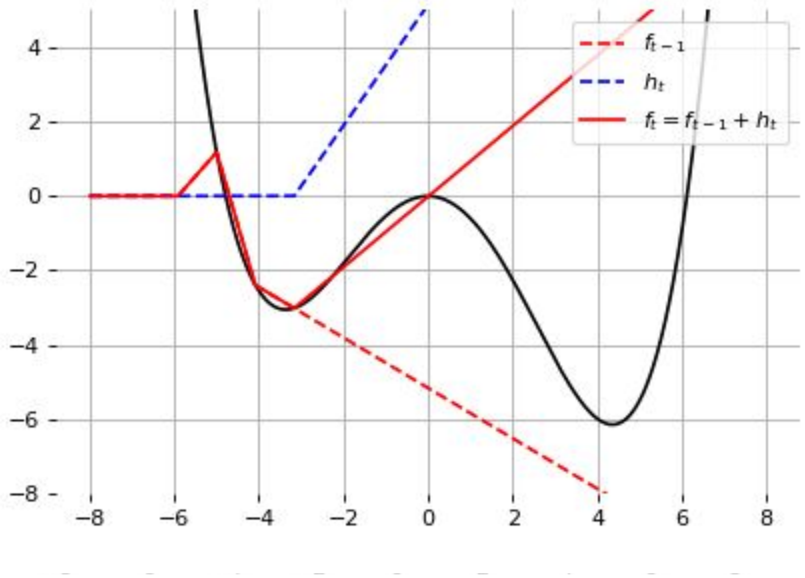
$$f(x) = \sum w_i \mathrm{ReLU}(x + b_i)$$

Approximating a function with ReLU



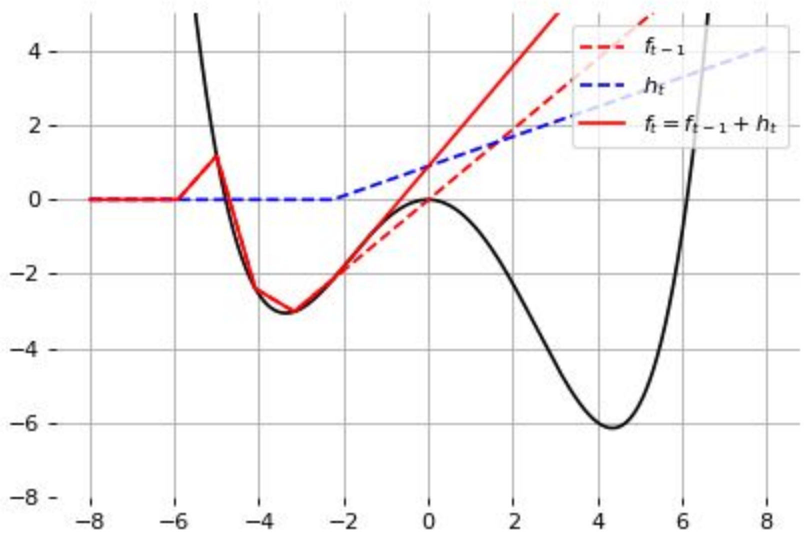
$$f(x) = \sum w_i \mathrm{ReLU}(x + b_i)$$

Approximating a function with ReLU



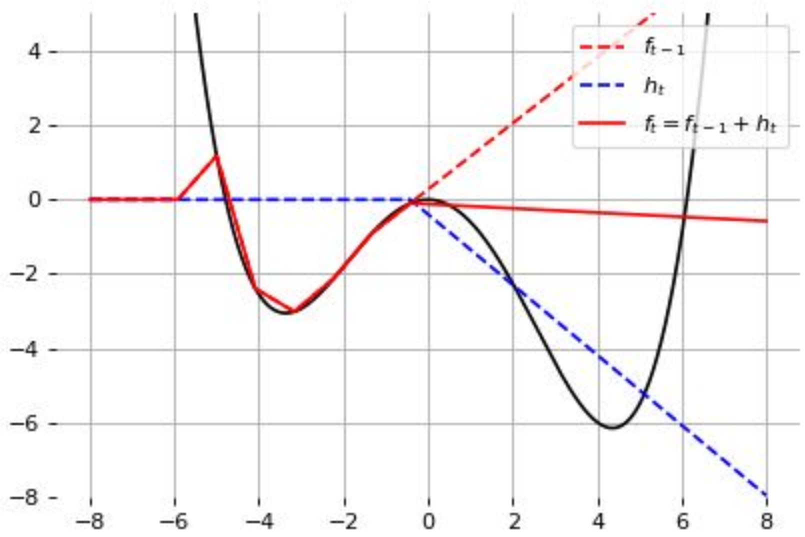
$$f(x) = \sum w_i \mathrm{ReLU}(x + b_i)$$

Approximating a function with ReLU



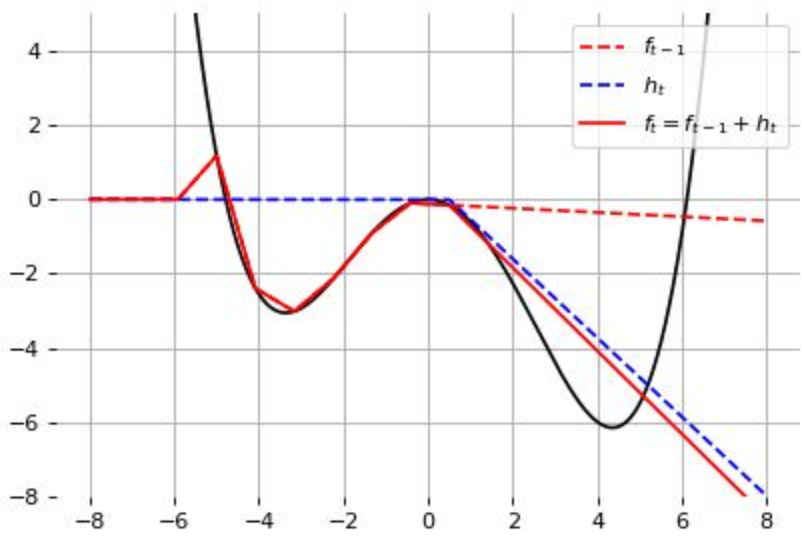
$$f(x) = \sum w_i \mathrm{ReLU}(x + b_i)$$

Approximating a function with ReLU



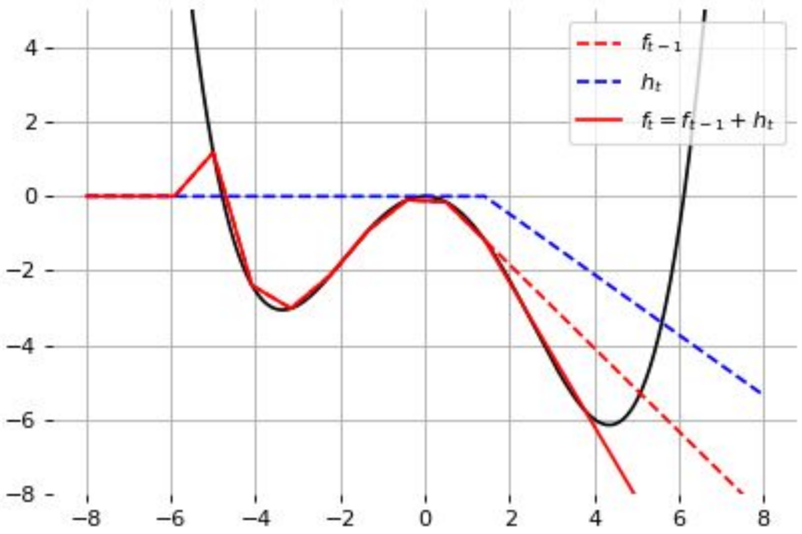
$$f(x) = \sum w_i \mathrm{ReLU}(x + b_i)$$

Approximating a function with ReLU



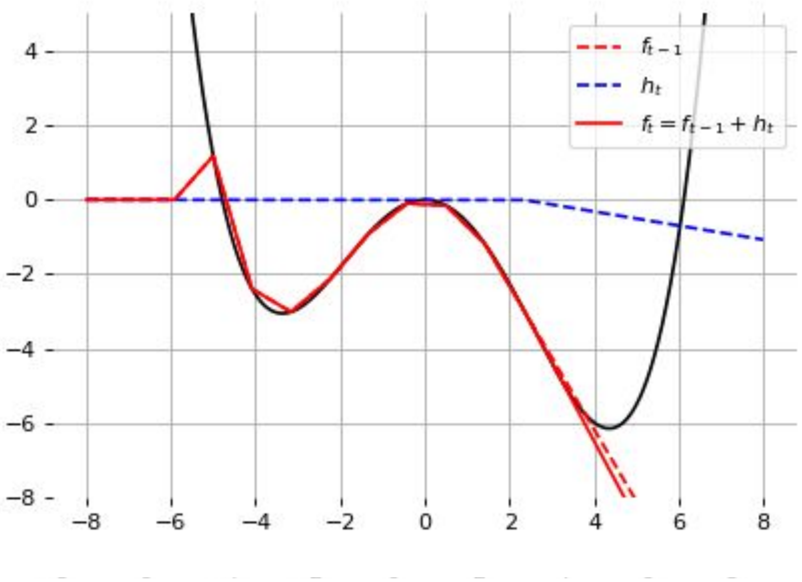
$$f(x) = \sum w_i \mathrm{ReLU}(x + b_i)$$

Approximating a function with ReLU



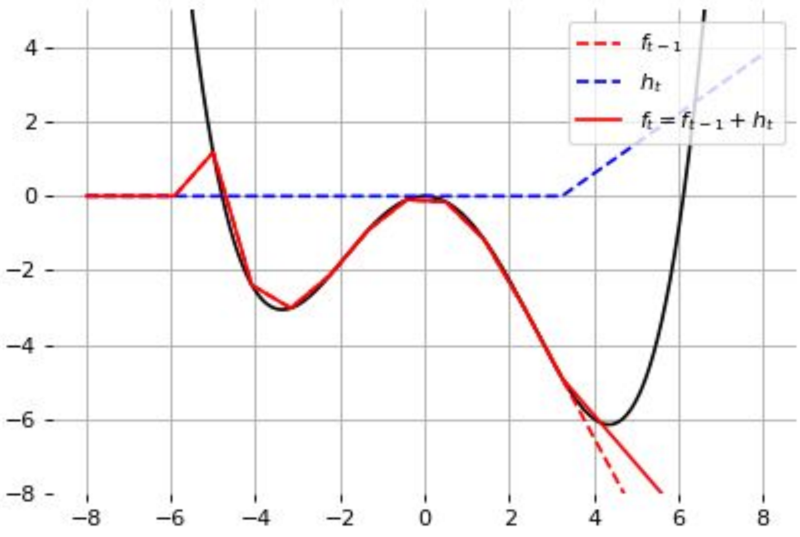
$$f(x) = \sum w_i \mathrm{ReLU}(x + b_i)$$

Approximating a function with ReLU



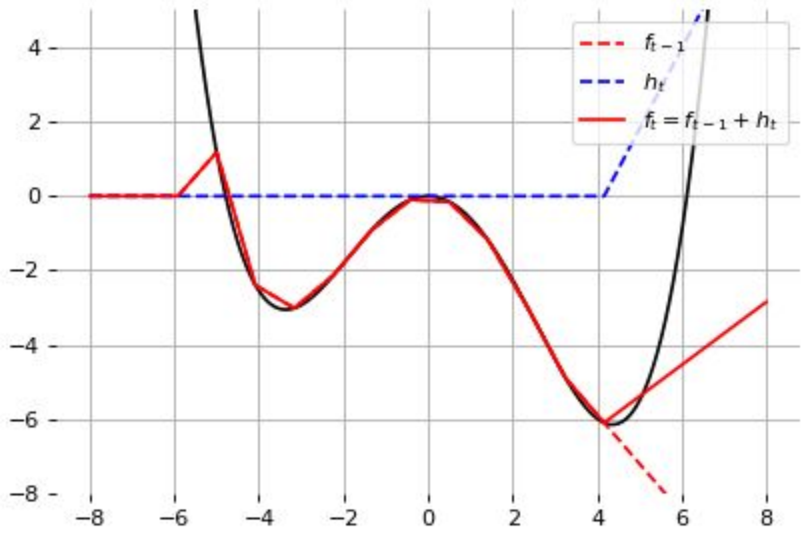
$$f(x) = \sum w_i \mathrm{ReLU}(x + b_i)$$

Approximating a function with ReLU



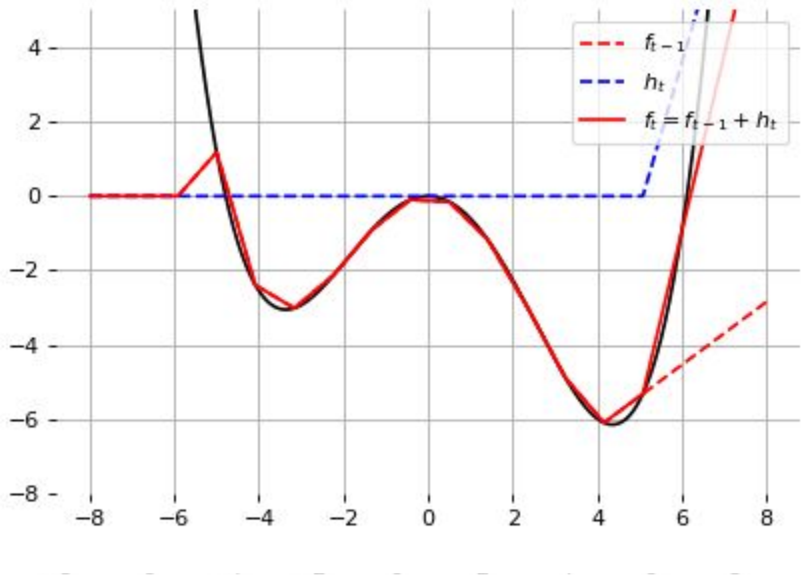
$$f(x) = \sum w_i \mathrm{ReLU}(x + b_i)$$

Approximating a function with ReLU



$$f(x) = \sum w_i \mathrm{ReLU}(x + b_i)$$

Approximating a function with ReLU

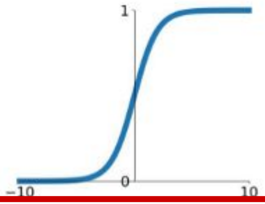


$$f(x) = \sum w_i \mathrm{ReLU}(x + b_i)$$

Activation functions

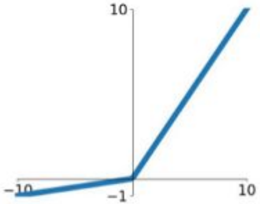
Sigmoid

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



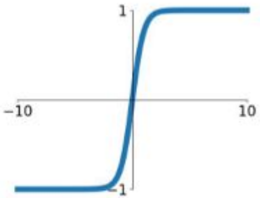
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

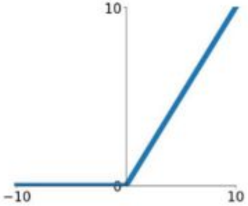


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

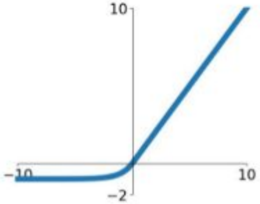
ReLU

$\max(0, x)$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

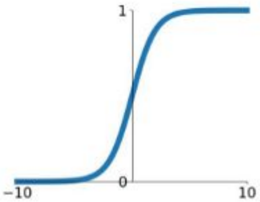


Often used as hidden layer activations

Activation functions

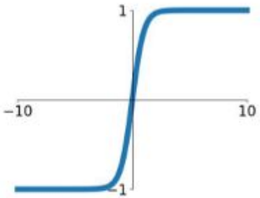
Sigmoid

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



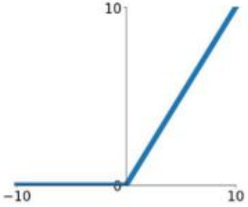
tanh

$\tanh(x)$



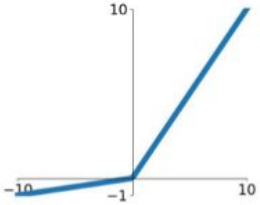
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

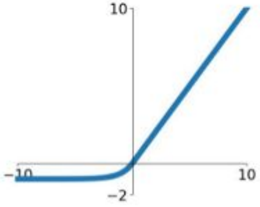


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Often used for output layers

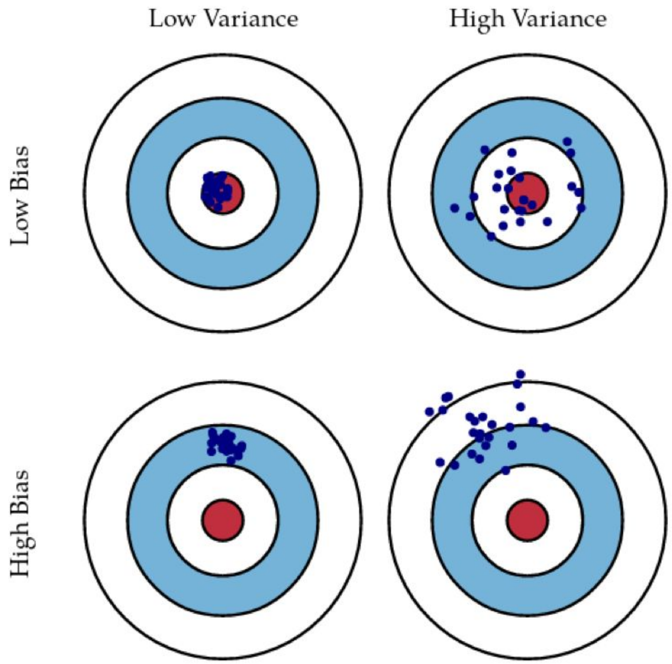
Play with [Activation Functions]

Bias-variance tradeoff

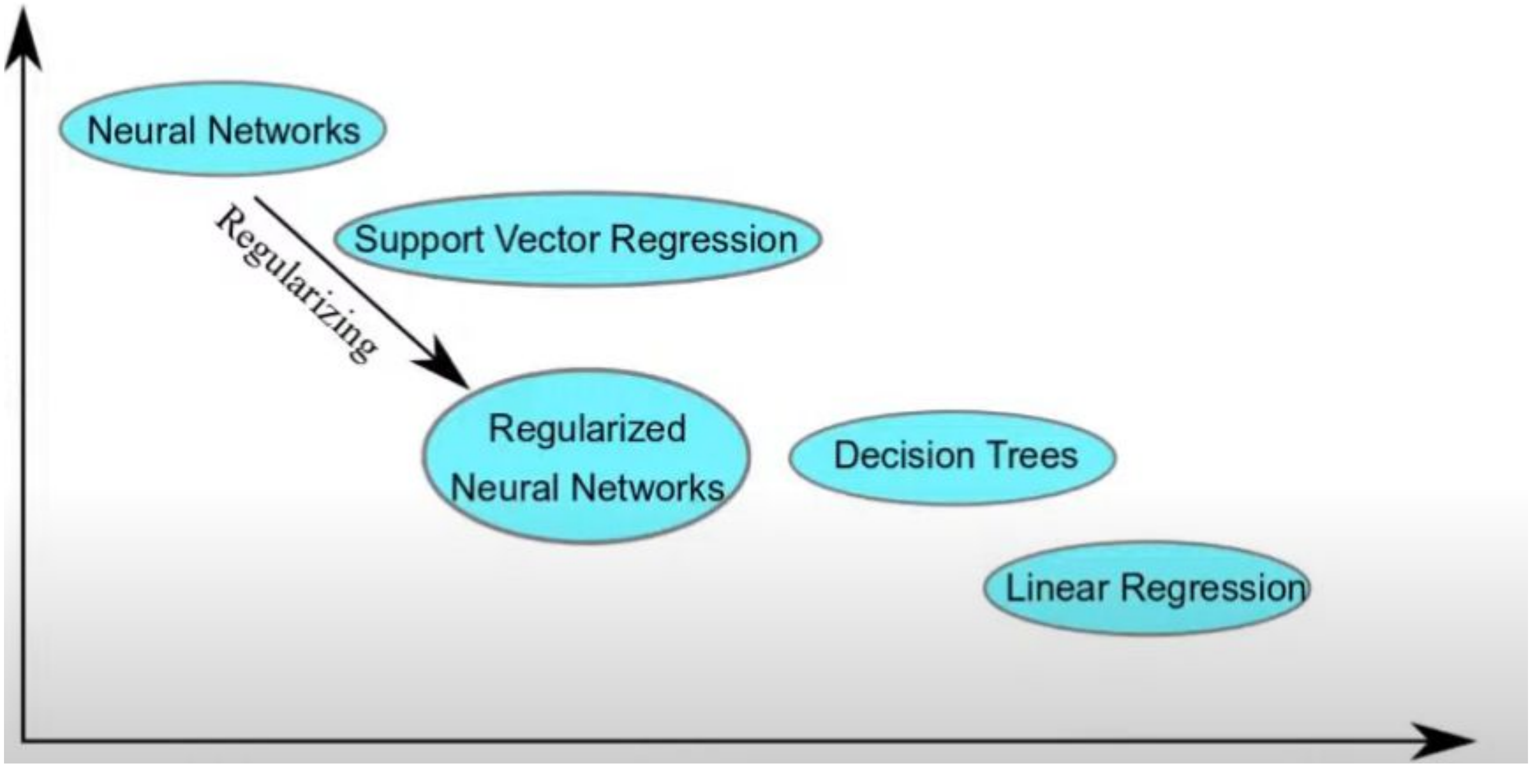
Classically:

There is a tradeoff between how accurate a model is and how it should generalize to unseen data

Ex: regression   MSE = Bias^2 + Variance



Where is the bias and the variance?

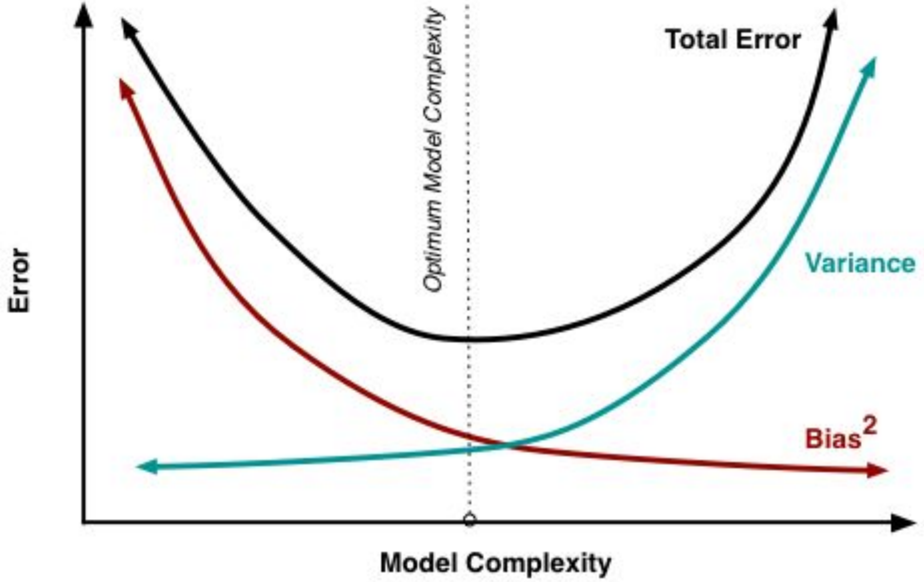


Bias-variance tradeoff revisited

- Reducing capacity should increase the bias term
- Decreasing capacity should increase the variance term

Deeper models should lead to larger "total" errors.

BUT…..



Deep and wide models: double descent

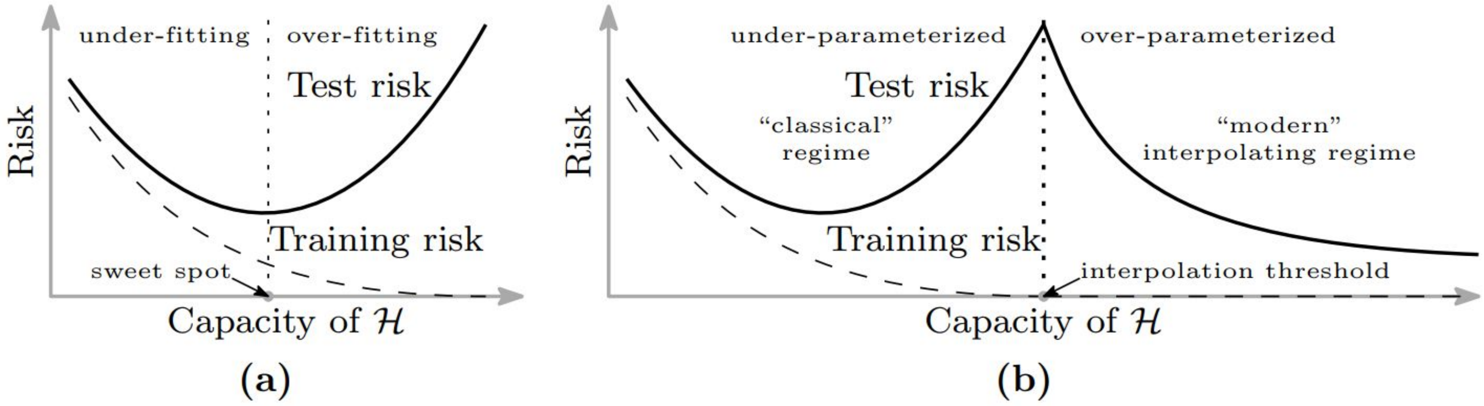
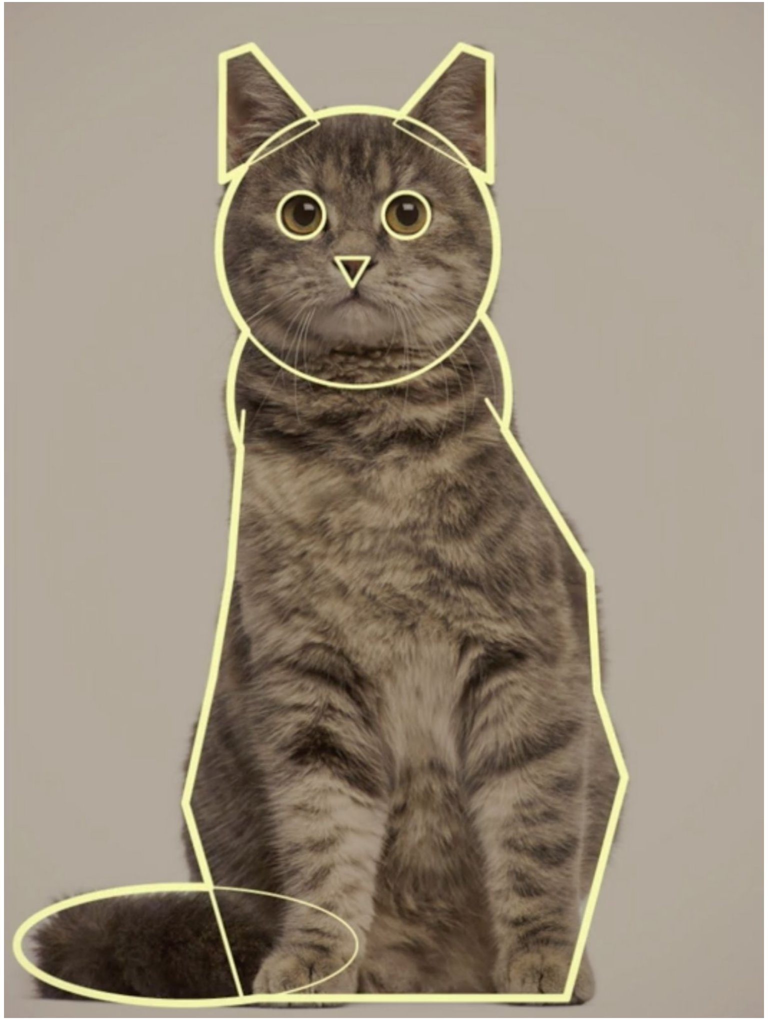


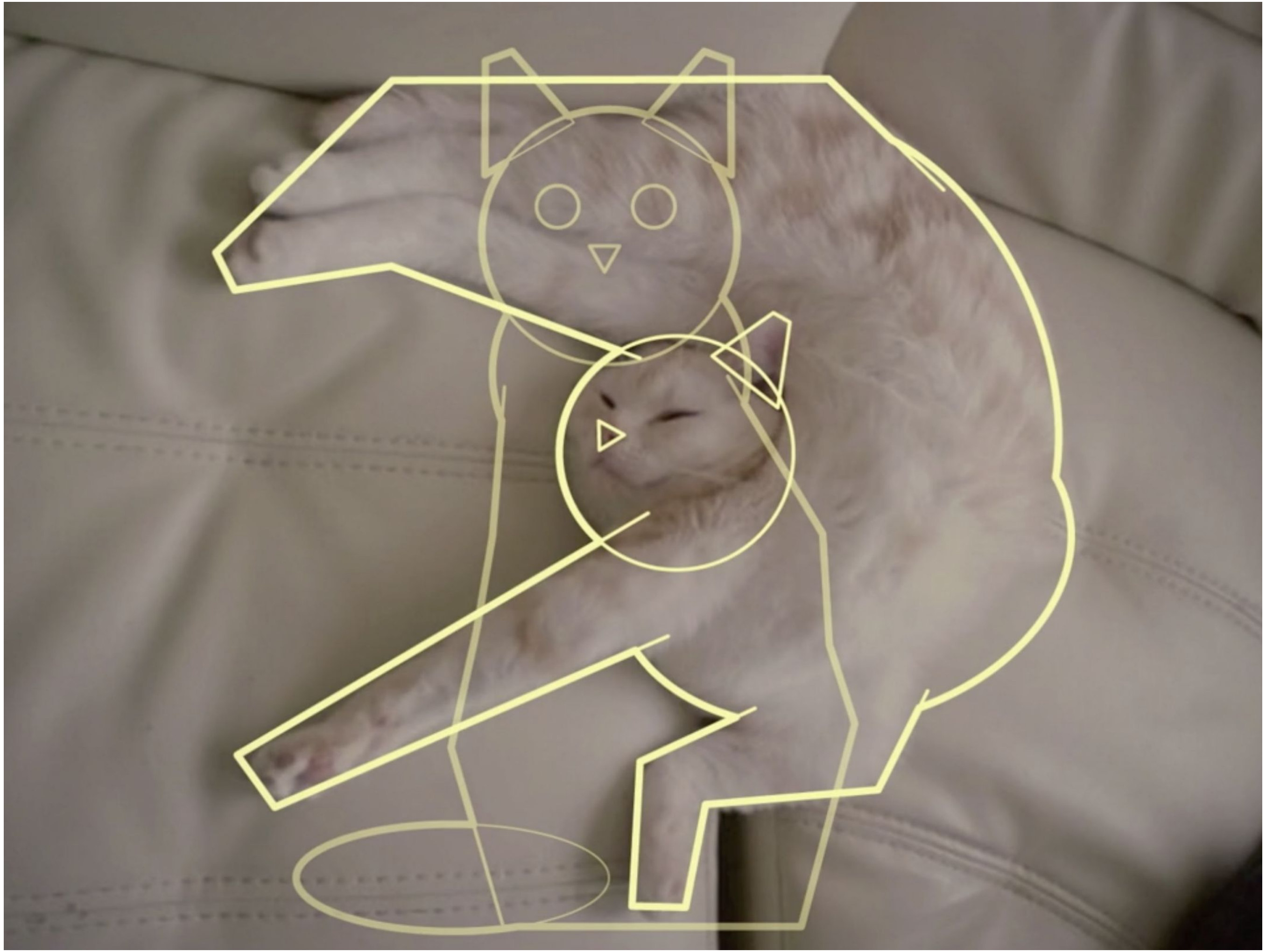
Figure 1: **Curves for training risk (dashed line) and test risk (solid line).** (**a**) The classical *U-shaped risk curve* arising from the bias-variance trade-off. (**b**) The *double descent risk curve*, which incorporates the U-shaped risk curve (i.e., the "classical" regime) together with the observed behavior from using high capacity function classes (i.e., the "modern" interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

Belkin (2018)

[Brady Neal Blog on the bias-variance tradeoff]

Write a computer program to classify cats and dogs?

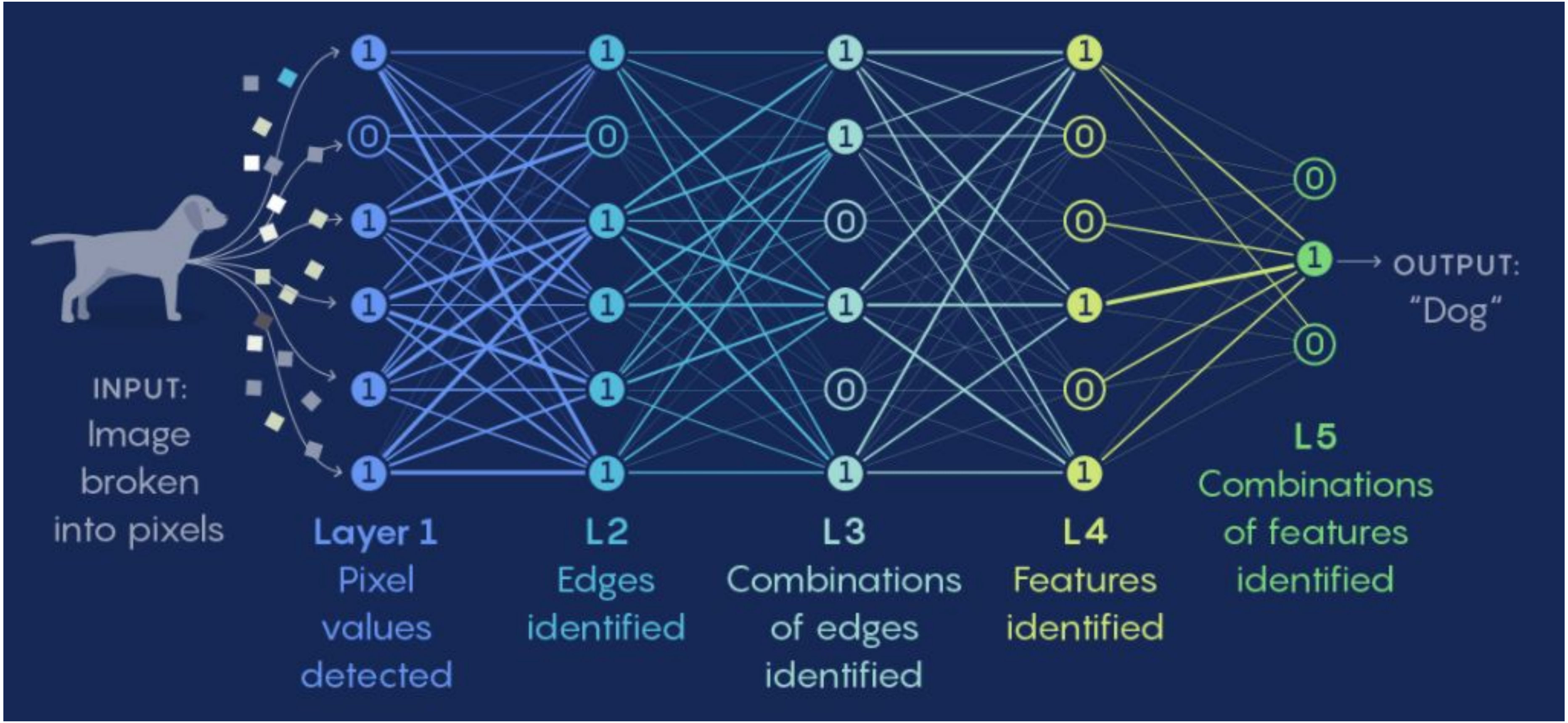




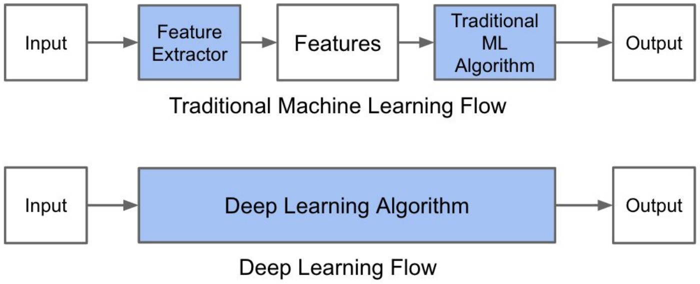
Classic approach:

1. **Define features: ellipses, polygons**
2. **Combinations of features is enough to specify a cat.**

Deep learning approach


INPUT:
Image broken into pixels
Layer 1
Pixel values detected
L2
Edges identified
L3
Combinations of edges identified
L4
Features identified
L5
Combinations of features identified
OUTPUT:
"Dog"

What is deep learning?



- Started in the 90's
- Today deep learning is not "just add more layers"
- One definition from Yann LeCun (Dec 2019):

Deep Learning is a methodology by which one constructs a model as a (possibly dynamic) graph of parameterized functions, feeding into an objective function optimized through some sort of gradient-based method.

- *To be "deep" the graph must have multiple non-linear stages from input to output.*
- *This depth allows the system to learn internal representations*
- *This definition does not specify the learning paradigm (supervised, unsupervised, reinforcement), nor the architecture, nor the objective.*

Why does deep learning work now?

More data

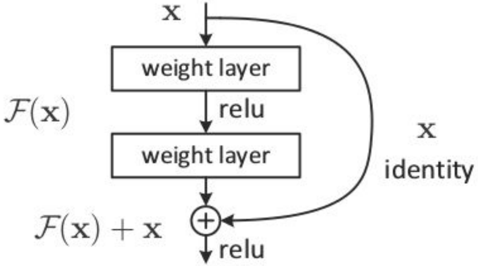
Fast compute

Efficient software

Caffe2 Microsoft CNTK TensorFlow theano

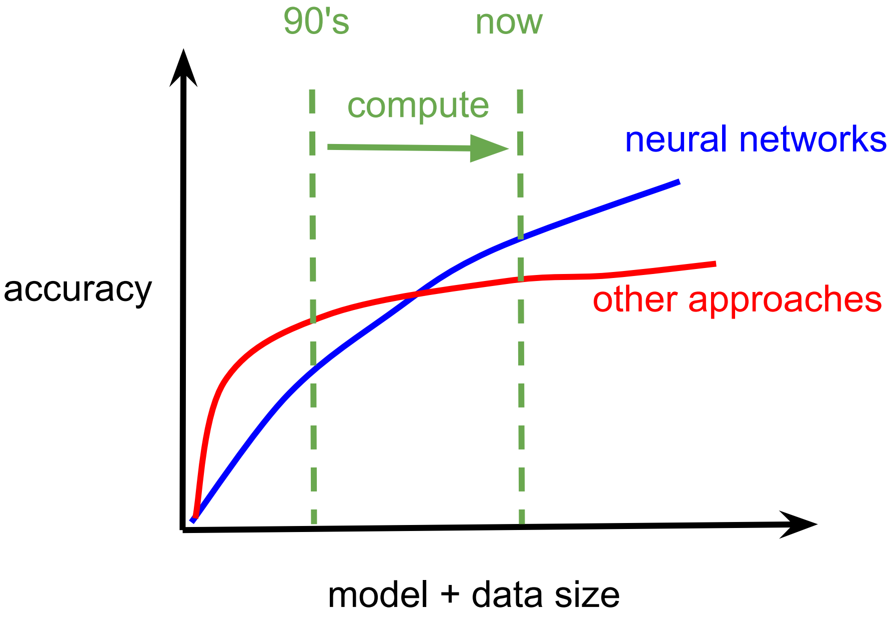
Chainer PYTORCH dmlc mxnet K

Algorithms



Why Going Deep?

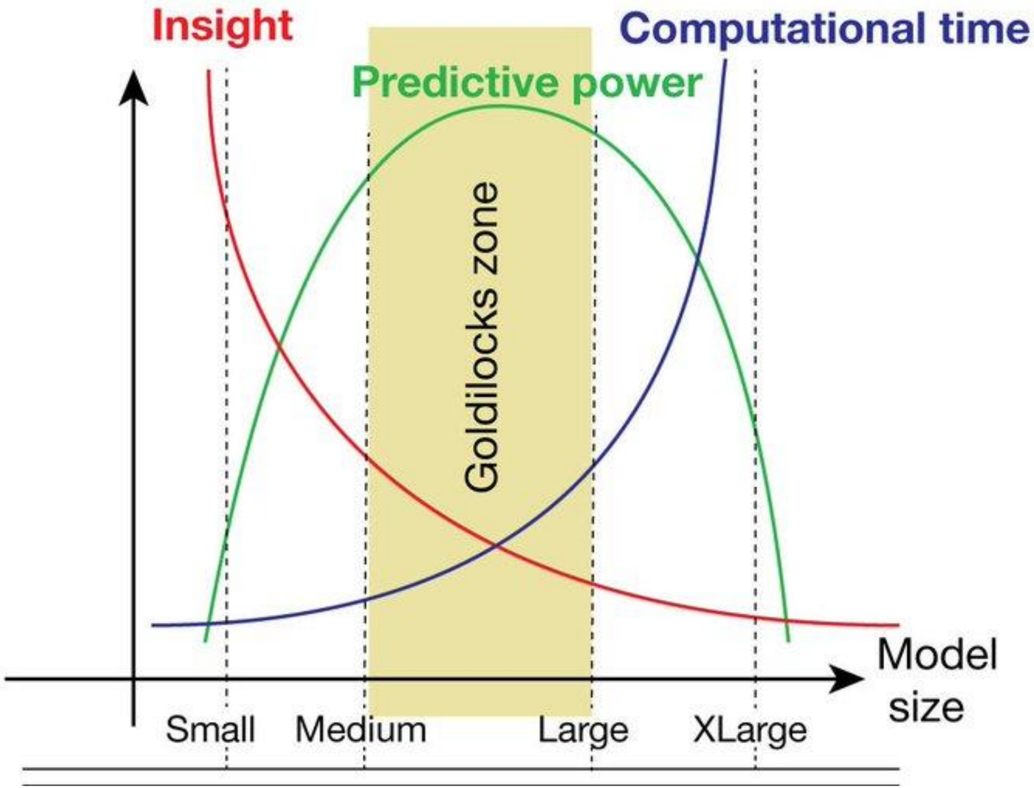
1. Practicality: wide layers require much more computation
2. Hierarchical learning: composing from previous layers
3. Similarity in biological systems
4. Empirical results



Challenges for deep learning

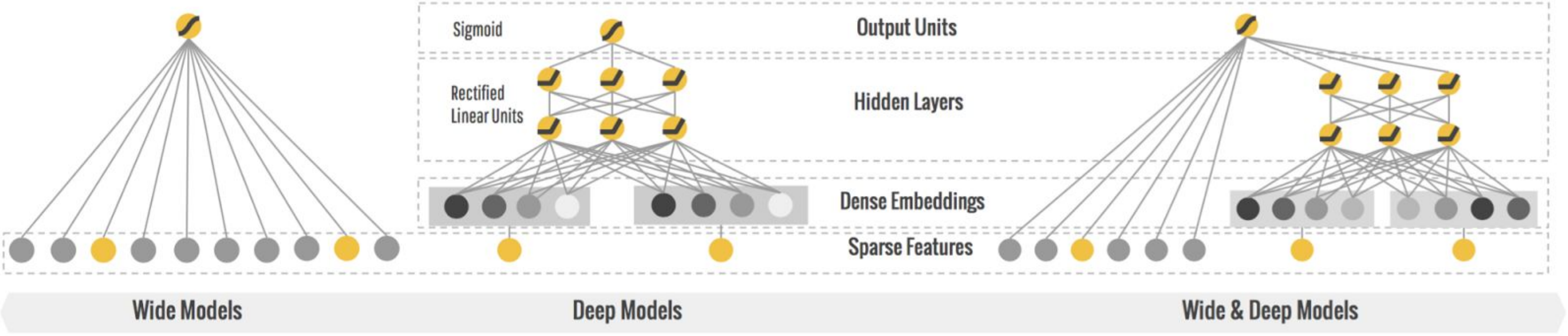
- Nonlinear optimization of millions of parameters
- Designing a deep neural network
- Dealing with very large data sets
- Human interpretation
- Generalization? (training set ≠ reality)

The Right Balance to Design a Network



deep vs. wide neural networks

- With the same number of nodes, represent more complex functions
- Empirically deep is more efficient
- Hierarchical structure of learned features
- Distributed representations



Resources

Deep Learning Books

Theoretical Books

- Deep Learning (2016) - Goodfellow, Bengio, Courville - UVic Library Link
  Reference book on deep learning. Slightly outdated, but authoritative

- Probabilistic Machine Learning (2022) - Murphy
  Recently updated series of in depth books of statistics, machine learning and deep learning.

Practical Books

- Deep Learning with Python (2021) - Chollet - UVic Library
  Very clear and insightful of deep learning. keras and tensorflow.

- Dive into Deep Learning (2022) - Zhang, Lipton, Li, Smola
  Free, open and continuously updated book with examples simultaneously in pytorch, tensorflow and mxnet

- Hands-on Machine Learning with scikit-learn, keras and tensorflow (2019) - Géron - UVic Library Link
  Very clear book and include practicalities for both standard ML and DL.

Some Deep Learning Online courses

- deeplearning.ai
  Various shorter courses. Mostly tensorflow based.
- CNN for Visual Recognition
  Popular introduction course on CNN. Well done, and good practical tricks.
- Deep Learning U Geneva
  Detailed computer vision heavy course with Pytorch.
- Deep Learning NYU.
  Modern course with a different approach. Well done and educational program
- fastai: Deep Learning for Coders
  A practical and efficient deep learning without much math. Active community. pytorch+fastai based, and with a book companion.

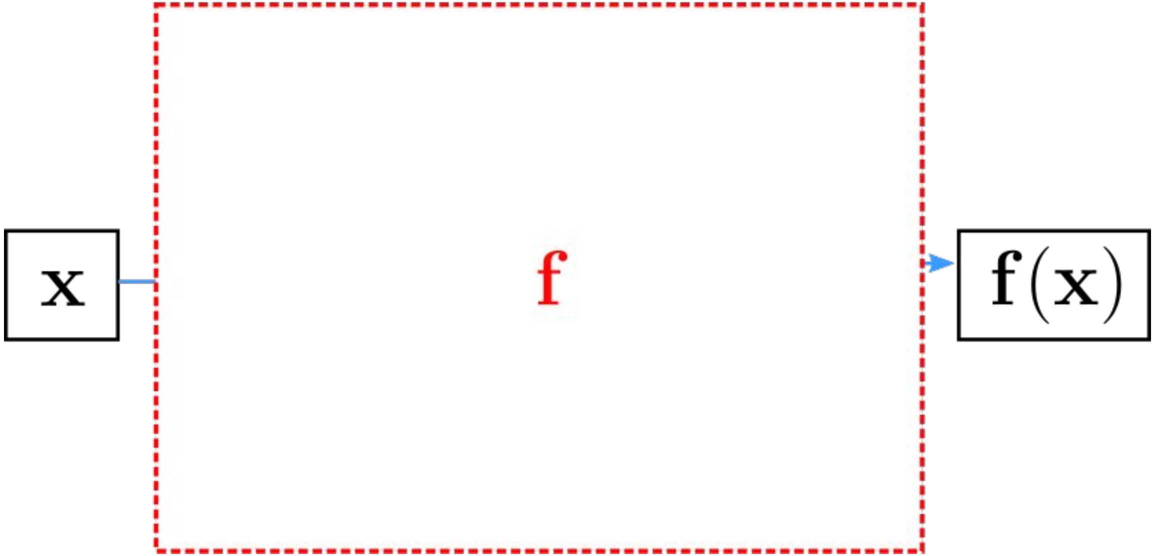
Deep Learning Computation
(Appendix)

Neural network computational graph

neural network

=

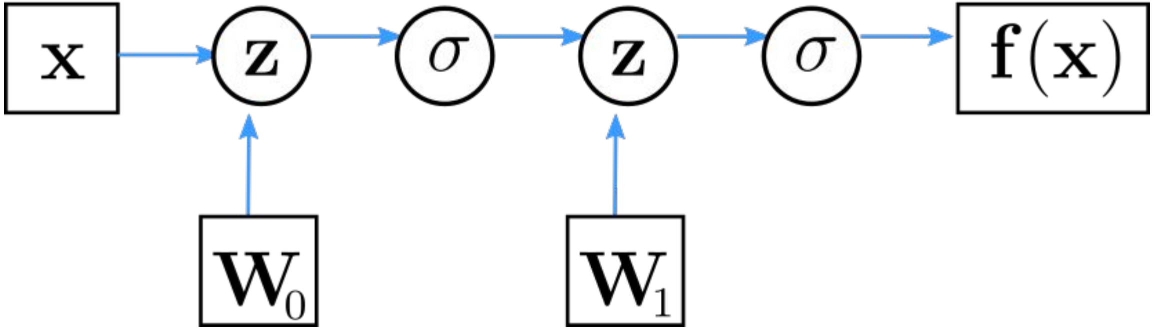
parameterized function



Neural network computational graph

Directed graph of functions

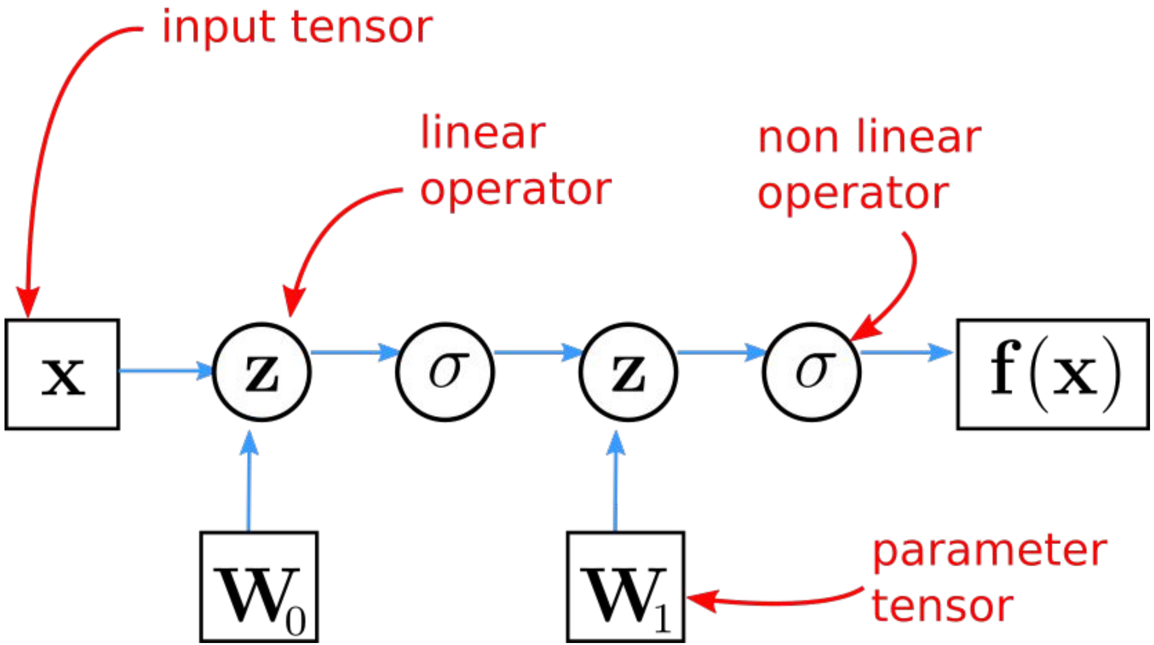
depending on weights



Neural network computational graph

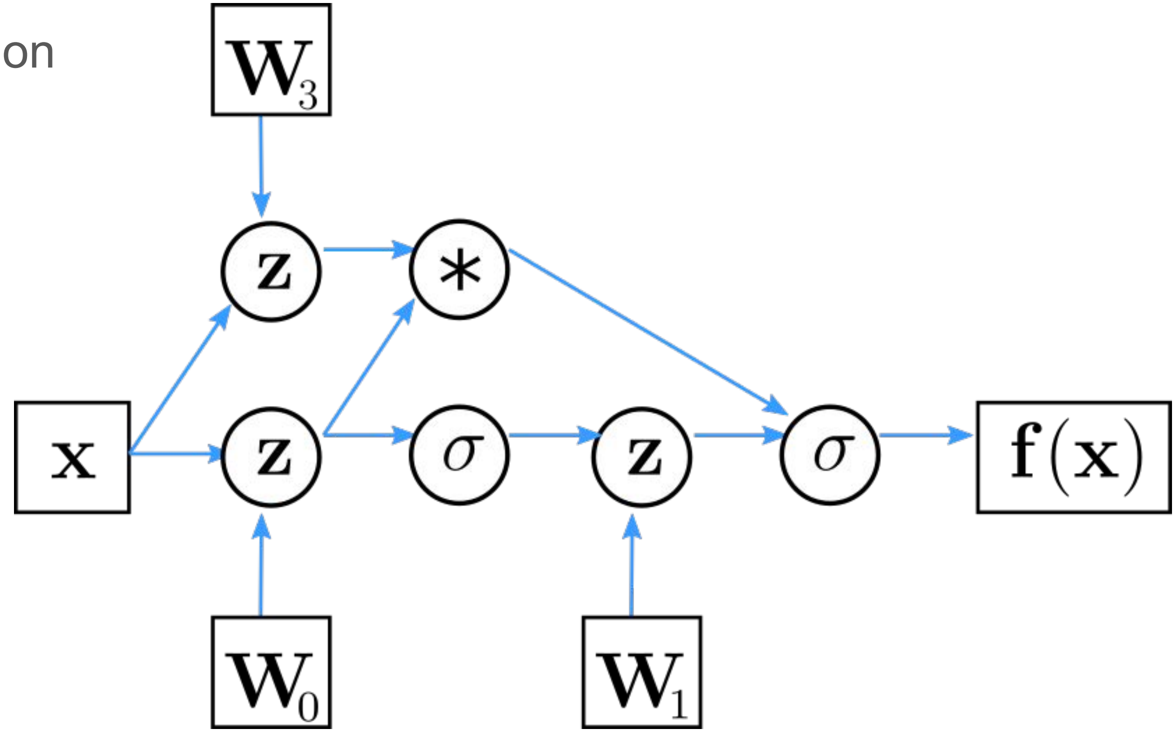
Combination of linear

and nonlinear functions

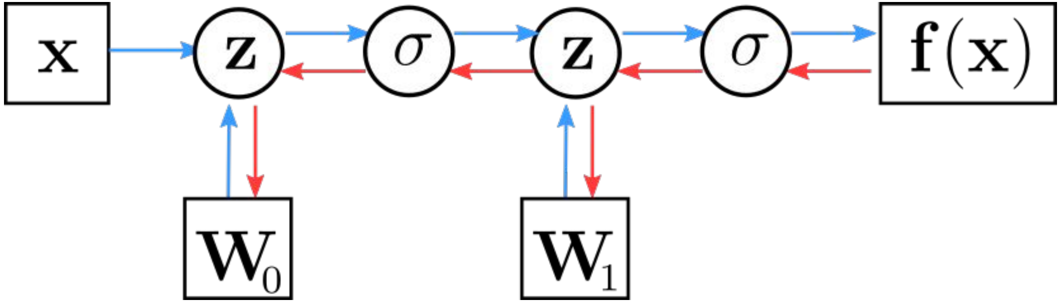


Neural network computational graph

Not only sequential application of functions



Computation of differentiable modules



Automatic computation of all gradients!

- tensorflow < 2, theano, mxnet: static graph computation
- tensorflow > 2, pytorch: dynamic differentiable modules
- if you need a lot more derivatives including second order: jax