Introduction to Deep Learning

Jesús Fernández-Villaverde[1] and Galo Nuño[2]

August 28, 2023

[1]University of Pennsylvania

[2]Banco de España

The problem

- We want to approximate ("learn") an unknown function:

$$y = f(\mathbf{x})$$

  where y is a scalar and $\mathbf{x} = \{x_0 = 1, x_1, x_2, ..., x_N\}$ a vector (including a constant).

- We care about the case when N is large (possibly in the thousands!).

- Easy to extend to the case where y is a vector (e.g., a probability distribution), but notation becomes cumbersome.

- In economics, $f(\mathbf{x})$ can be a value function, a policy function, a pricing kernel, a conditional expectation, a classifier, ...

A neural network

- An artificial neural network (a.k.a. a connectionist system) is a approximation to $f(\mathbf{x})$ of the form:

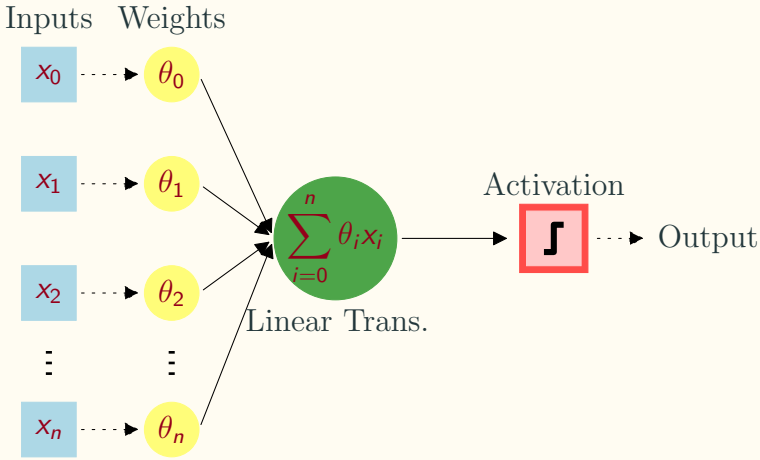
$$y = f(\mathbf{x}) \cong g^{NN}(\mathbf{x}; \theta) = \theta_0 + \sum_{m=1}^{M} \theta_m \phi(z_m)$$

  where $\phi(\cdot)$ is an arbitrary activation function and:

$$z_m = \sum_{n=0}^{N} \theta_{n,m} x_n$$

- The x_n's are known as the features of the data, which belong to a feature space $\mathcal{X}$.
- The $\phi(z_m)$'s are known as the representation of the data (a generalized linear model).
- M is known as the width of the model (wide vs. thin networks).
- "Training" the network: We select θ such that $g^{NN}(\mathbf{x}; \theta)$ is as close to $f(\mathbf{x})$ as possible given some relevant metric (e.g., the ℓ_2 norm).

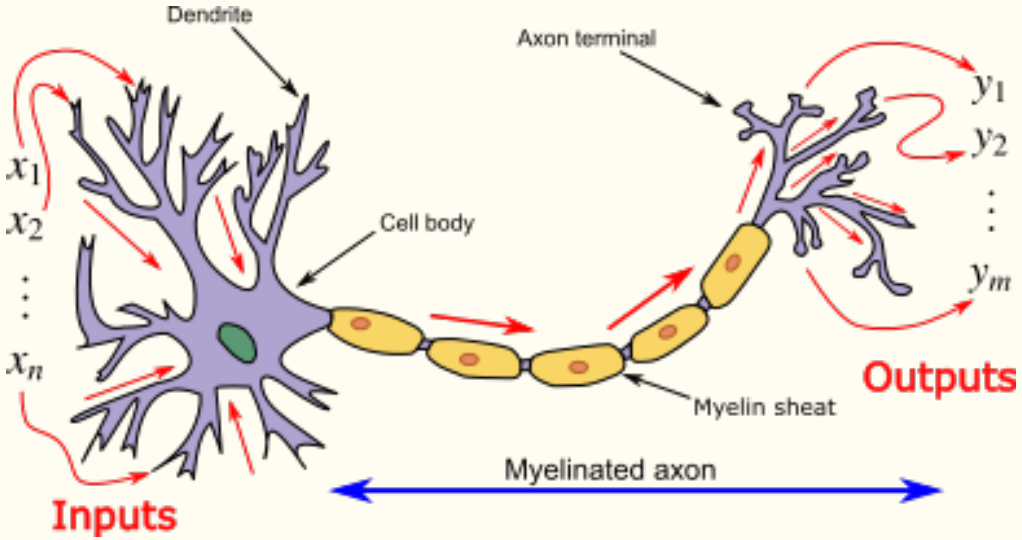
Flow representation



Intuition

- Intuition 1: A biological interpretation, but I do not find it too useful. Closer to econometrics (e.g., NOLS, semiparametric regression, and sieves) and differential geometry.
- Intuition 2: We look for representations of the features of the data that are informationally efficient.
- Intuition 3 (more advanced): We look for translations and rotations of the data that deliver a more convenient geometry by moving from a parent space to a simpler one.

The biological analog



Comparison with other approximations

- Compare:

$$f(\mathbf{x}) \cong g^{NN}(\mathbf{x};\theta) = \theta_0 + \sum_{m=1}^{M} \theta_m \phi \left(\sum_{n=0}^{N} \theta_{n,m} x_n \right)$$

  with a standard projection:

$$f(\mathbf{x}) \cong g^{CP}(\mathbf{x};\theta) = \theta_0 + \sum_{m=1}^{M} \theta_m \phi_m(\mathbf{x})$$

  where ϕ_m is, for example, a Chebyshev polynomial.

- We exchange the rich parameterization of coefficients for the parsimony of basis functions.
- How we determine the coefficients will also be different, but this is somewhat less important.

Why do neural networks "work"?

- Neural networks consist entirely of chains of tensor operations: we take **x**, we perform affine transformations, and apply an activation function.
- Thus, these tensor operations are geometric transformations of **x**. In fact, a better name for neural networks could be *chained geometric transformations*.
- In other words: a neural network is a complex geometric transformation in a high-dimensional space.
- Deep neural networks look for convenient geometrical representations of high-dimensional manifolds.
- The success of any functional approximation problem is to search for the right geometric space in which to perform it, not to search for a "better" basis function.
- Think about:

$$y = k^{\alpha} l^{1-\alpha} \Rightarrow \log y = \alpha \log k + (1-\alpha) \log l$$

Deep learning, I

- A deep learning network is an acyclic *multilayer* composition of $J > 1$ neural networks:

$$z_m^0 = \theta_{0,m}^0 + \sum_{n=1}^{N} \theta_{n,m}^0 x_n$$

and

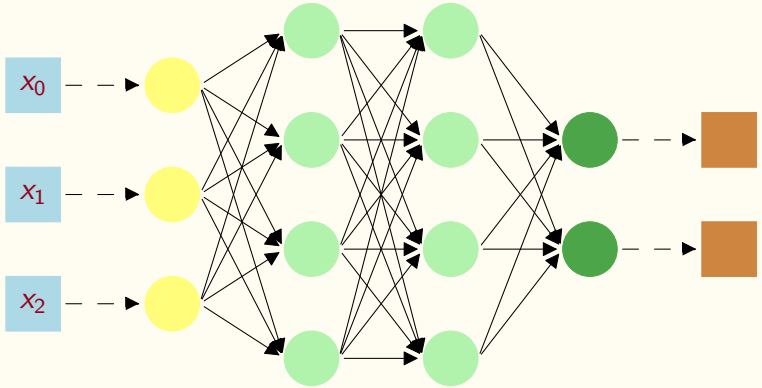
$$z_m^1 = \theta_{0,m}^1 + \sum_{m=1}^{M^{(1)}} \theta_m^1 \phi^1 \left(z_m^0\right)$$

...

$$y \cong g^{DL}(\mathbf{x}; \theta) = \theta_0^J + \sum_{m=1}^{M^{(J)}} \theta_m^J \phi^J \left(z_m^{J-1}\right)$$

where the $M^{(1)}, M^{(2)}, ...$ and $\phi^1(\cdot), \phi^2(\cdot), ...$ are possibly different across each layer of the network.

- A deep network creates new representations by composing older representations.



Input Values Hidden Layer 1 Output Layer

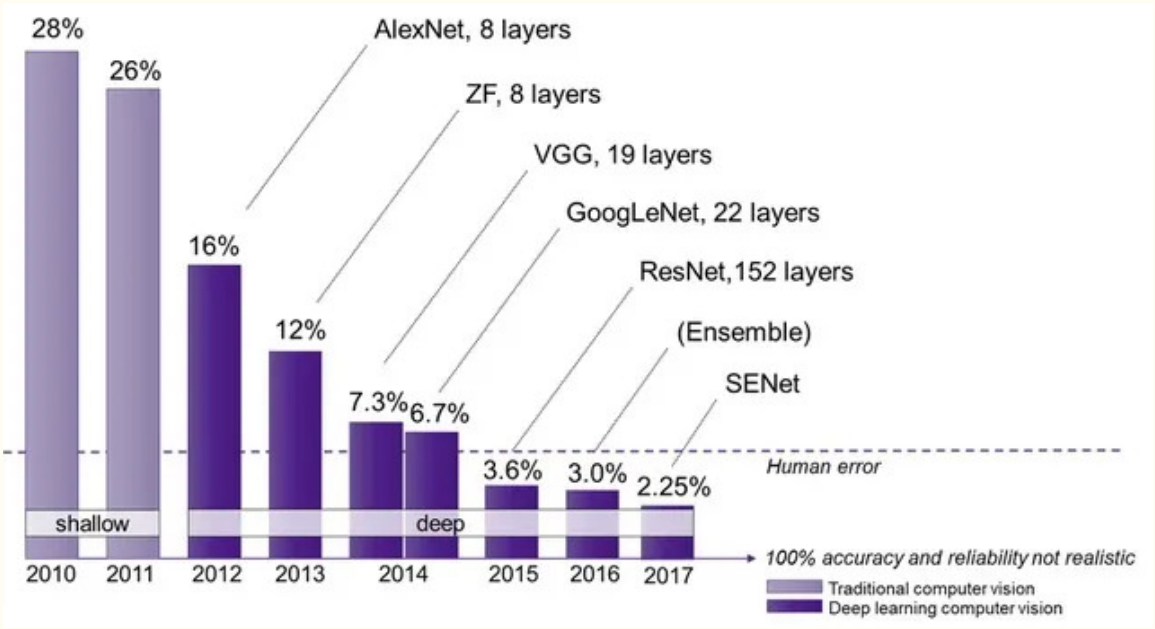
Input Layer Hidden Layer 2

Deep learning, II

- Sometimes known as deep feedforward neural networks or, of fully connected, multilayer perceptrons (MLPs).
- "Feedforward" comes from the fact that the composition of neural networks can be represented as a directed acyclic graph, which lacks feedback. We can have more general recurrent structures.
- J is known as the depth of the network (deep vs. shallow networks).
- The case $J = 1$ is a standard neural network.
- As before, we can select θ such that $g^{DL}(\mathbf{x}; \theta)$ approximates a target function $f(\mathbf{x})$ as closely as possible under some relevant metric.
- All other aspects (selecting $\phi(\cdot)$, J, M, ...) are known as the network architecture. We will discuss extensively at the end of this slide block how to determine them.

Why do deep neural networks “work” better?

- Why do we want to introduce hidden layers?
  1. It works! Evolution of ImageNet winners.
  2. The number of representations increases exponentially with the number of hidden layers while computational cost grows linearly.
  3. Intuition: hidden layers induce highly nonlinear behavior in the joint creation of representations without the need to have domain knowledge (used, in other algorithms, in some form of greedy pre-processing).


28%
26%
16%
12%
7.3%
6.7%
3.6%
3.0%
2.25%
AlexNet, 8 layers
ZF, 8 layers
VGG, 19 layers
GoogLeNet, 22 layers
ResNet,152 layers
(Ensemble)
SENet
Human error
shallow
deep
2010
2011
2012
2013
2014
2015
2016
2017
100% accuracy and reliability not realistic
Traditional computer vision
Deep learning computer vision

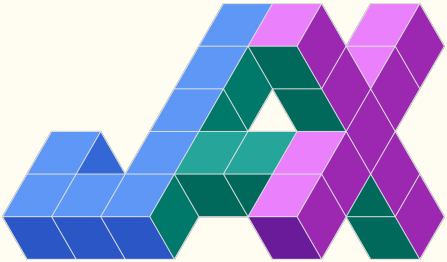
Some consequences

- Because of the previous arguments, neural networks can efficiently approximate extremely complex functions.
- In particular, under certain (relatively weak) conditions:
  1. Neural networks are universal approximators.
  2. Neural networks break the "curse of dimensionality."
- Furthermore, neural networks are easy to code, stable, and scalable for multiprocessing (neural networks are built around tensors).

Further advantages

- Neural networks and deep learning often require less "inside knowledge" by experts on the area.
- While results can be highly counter-intuitive, deep neural networks deliver excellent performance.
- Outstanding open source libraries (`Tensorflow`, `Keras`, `Pytorch`, `JAX`) that integrate well with easy scripting languages (`Python`).
- Newer algorithms: batch normalization, residual connections, and depthwise separable convolutions.
- More recently, development of dedicated hardware (TPUs, AI accelerators, FPGAs) are likely to maintain a hedge for the area.
- The richness of an ecosystem is key for its long-run success.


TensorFlow



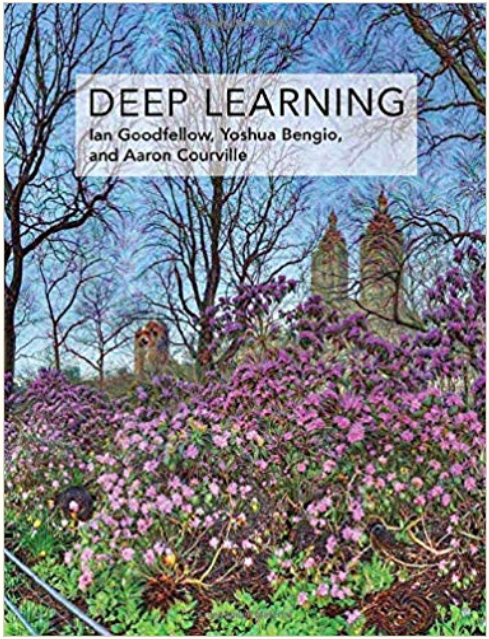

PyTorch
Lightning

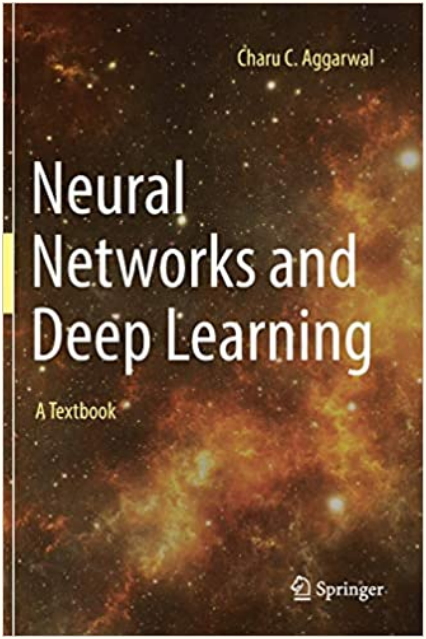

flux

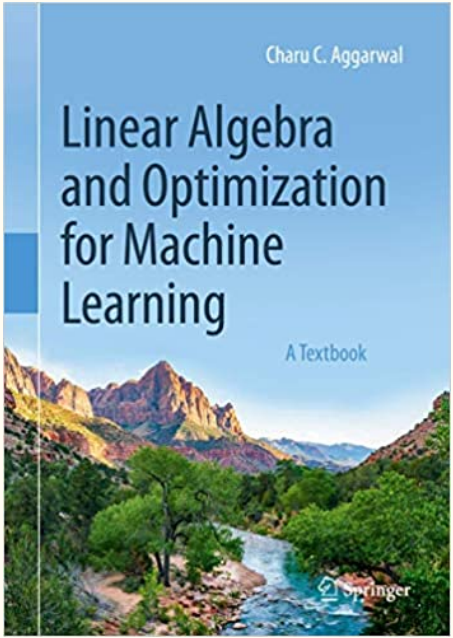
Limitations of neural networks and deep learning

- While neural networks and deep learning can work extremely well, there is no such a thing as a silver bullet.
- Clear and serious trade-offs in real-life applications.
- We often require tens of thousands of observations to properly train a deep network.
- Of course, sometimes "observations" are endogenous (we can simulate them) and we can implement data augmentation, but if your goal is to forecast GDP next quarter, it is unlikely a deep neural network will beat an ARIMA(n,p,q) (at least only with macro variables).
- Issues of interpretation.
- We are very far from any type of general human intelligence. Think about the process of designing a rocket.

References


DEEP LEARNING
Ian Goodfellow, Yoshua Bengio,
and Aaron Courville

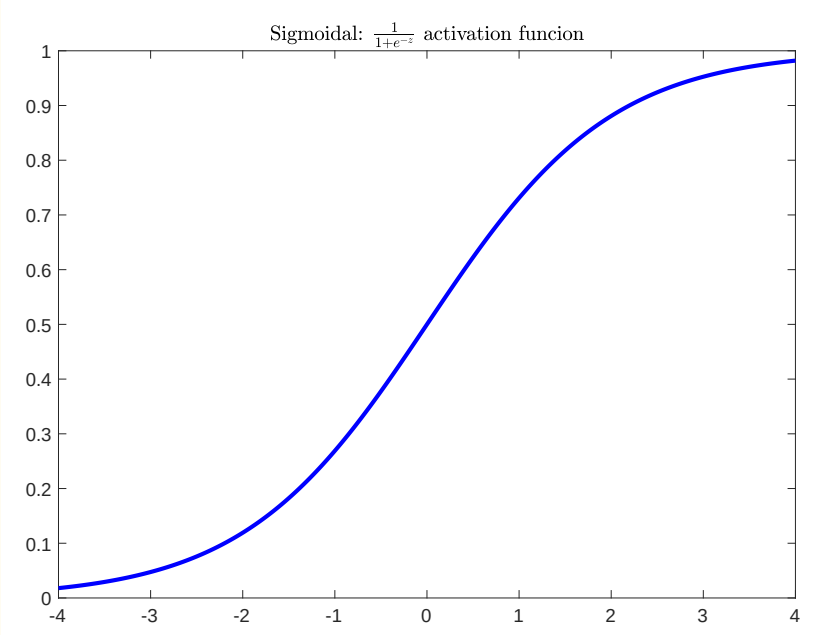

Charu C. Aggarwal
Neural Networks and Deep Learning
A Textbook
Springer

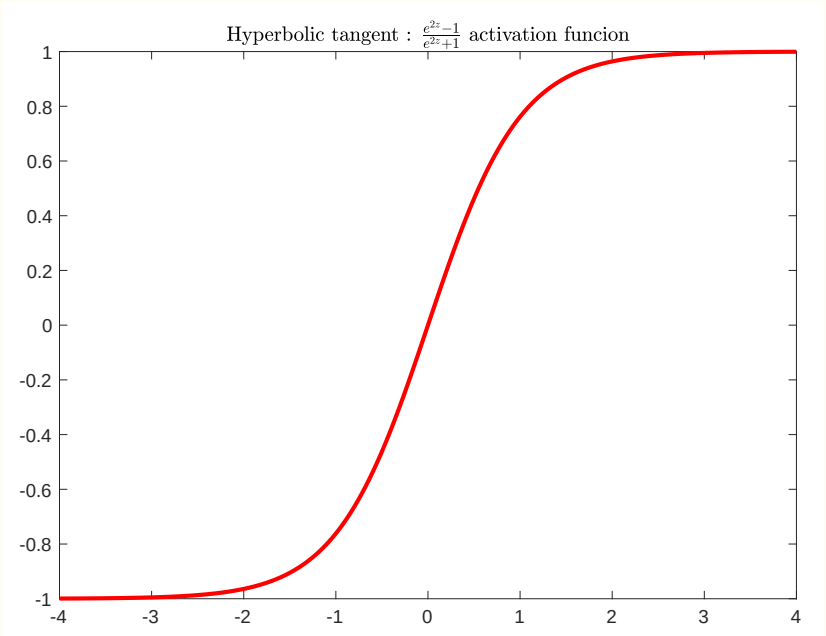

Charu C. Aggarwal
Linear Algebra and Optimization for Machine Learning
A Textbook
Springer

Digging deeper

Activation functions I

- Traditionally:
  1. Identity function:
  $$\phi(z) = z$$
  Used in linear regression.
  2. A sigmoidal function:
  $$\phi(z) = \frac{1}{1 + e^{-z}}$$
  3. Step function (a limiting case as z grows quickly):
  $$\phi(z) = 1 \text{ if } z > 0, \phi(z) = 0 \text{ otherwise.}$$
  4. Hyperbolic tangent:
  $$\phi(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$


Sigmoidal: $\frac{1}{1+e^{-z}}$ activation funcion
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-4
-3
-2
-1
0
1
2
3
4


Hyperbolic tangent : $\frac{e^{2z}-1}{e^{2z}+1}$ activation funcion
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-4
-3
-2
-1
0
1
2
3
4

Activation functions II

- Some activation functions that have gained popularity recently:

1. Rectified linear unit (ReLU):

$$\phi(z) = \max(0, z)$$

2. Parametric ReLU:

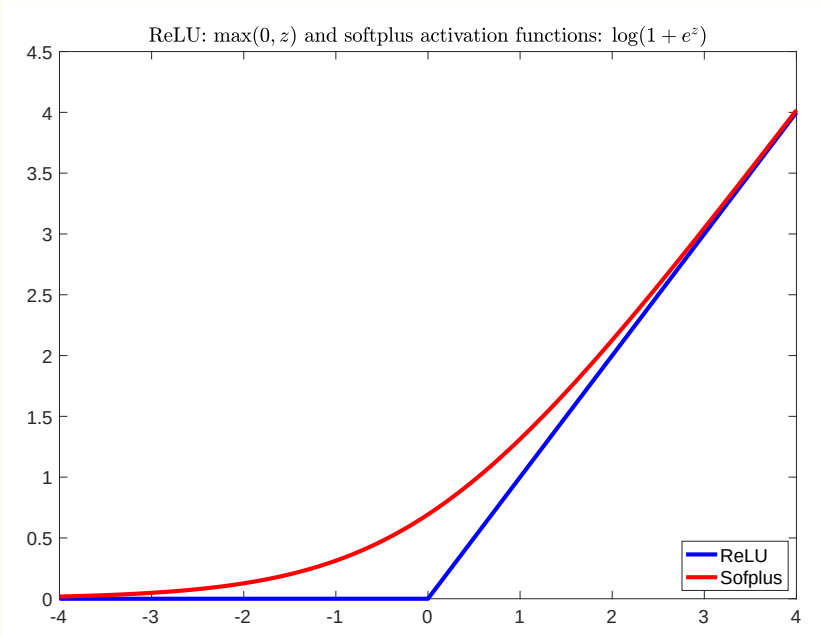
$$\phi(z) = \max(z, az)$$

3. Continuously Differentiable Exponential Linear Units (CELU):

$$\phi(z) = \max(0, z) + \min(0, \alpha(e^{x/\alpha} - 1))$$

4. Softplus:

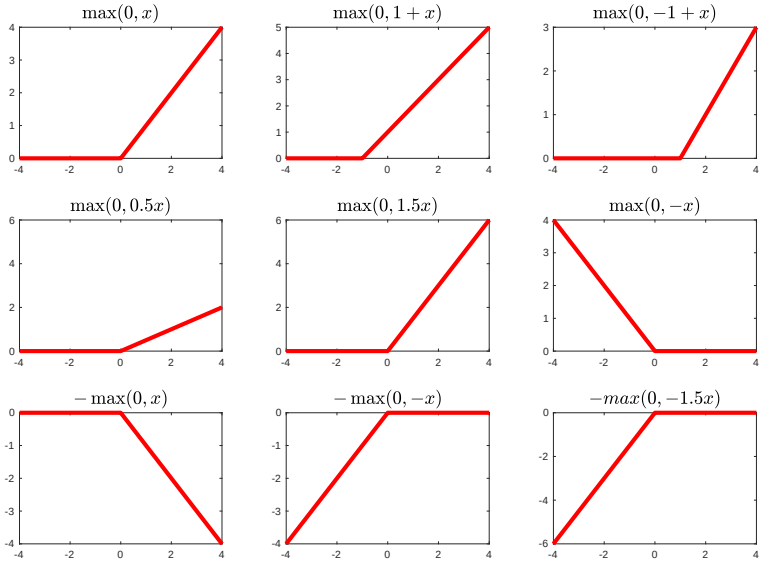
$$\phi(z) = \log(1 + e^z)$$


ReLU: $\max(0, z)$ and softplus activation functions: $\log(1 + e^z)$
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-4
-3
-2
-1
0
1
2
3
4
ReLU
Sofplus

Interpretation

- θ_0 controls the activation threshold.
- The level of the θ_i's for $i > 0$ control the activation rate (the higher the θ_i's, the harder the activation).
- Some textbooks separate the activation threshold and scaling coefficients from θ as different coefficients in ϕ, but such separation moves notation farther away from standard econometrics.
- But in practice θ does not have a structural interpretation, so the identification problem is of secondary importance.

Different ReLUs: $\theta_i \max(0, \theta_{i,0} + \theta_{i,1} x)$



Two classic (yet remarkable) results I

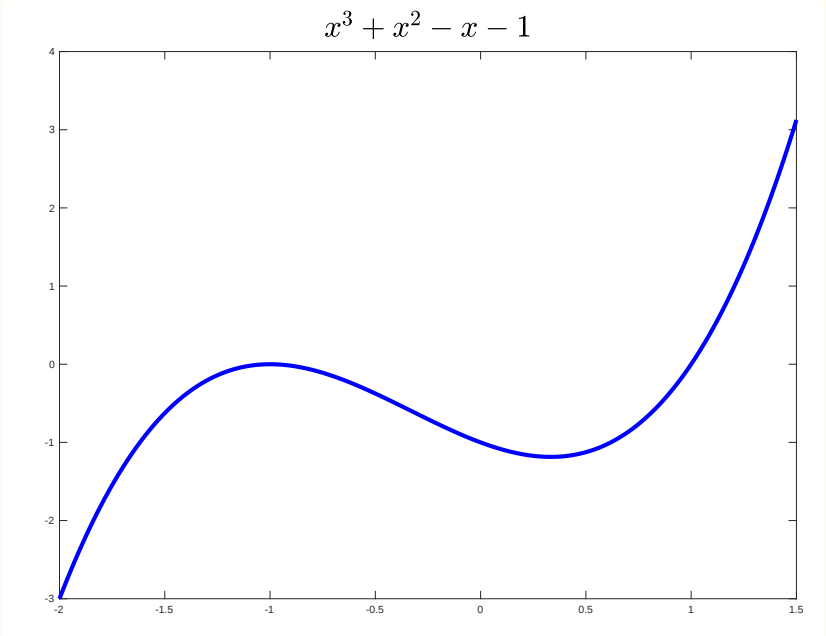
Borel measurable function

A map $f : X \to Y$ between two topological spaces is called Borel measurable if $f^{-1}(A)$ is a Borel set for any open set A on Y (the Borel sets are all the open sets built through the operations of countable union, countable intersection, and relative complement).

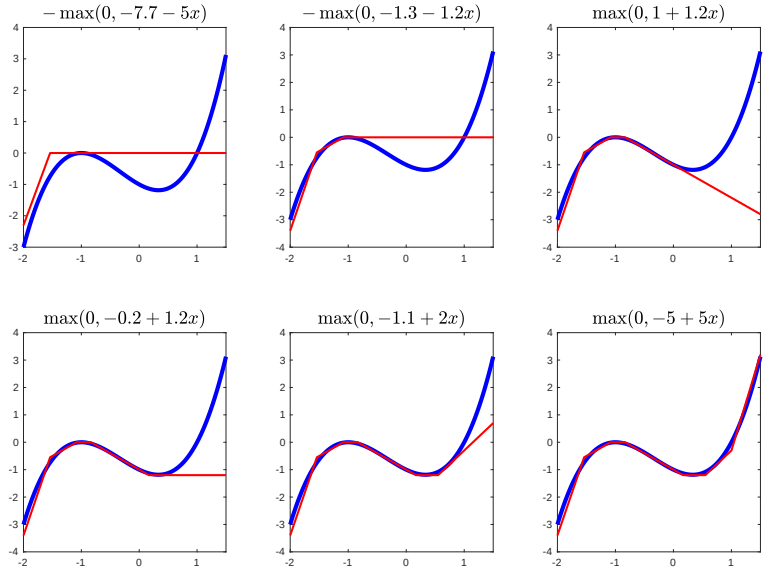
Universal approximation theorem: Hornik, Stinchcombe, and White (1989)

A neural network with at least one layer can approximate any Borel measurable function mapping finite-dimensional spaces to any desired degree of accuracy.

- Intuition of the result.
- Comparison with other results in series approximations.


$x^3 + x^2 - x - 1$
4
3
2
1
0
-1
-2
-3
-2
-1.5
-1
-0.5
0
0.5
1
1.5

A six ReLUs approximation



Two classic (yet remarkable) results II

- Assume, as well, that we are dealing with the class of functions for which the Fourier transform of their gradient is integrable.

Breaking the curse of dimensionality: Barron (1993)

A one-layer NN achieves integrated square errors of order $\mathcal{O}(1/M)$, where M is the number of nodes. In comparison, for series approximations, the integrated square error is of order $\mathcal{O}(1/(M^{2/N}))$ where N is the dimensions of the function to be approximated.

- More general theorems by Leshno *et al.* (1993) and Bach (2017).
- What about Chebyshev polynomials? Splines? Problems of convergence and generalization ("extrapolation").
- There is another, yet more subtle curse of dimensionality: data availability. We will return to this concern while dealing with symmetries

Playing with multiple layers

- Often, fewer neurons in higher layers allow for compression of learning into fewer features. In fact, intermediate features are many times interesting by themselves.
- We can also add multidimensional outputs.
- Or even to produce, as output, a probability distribution, for example, using a softmax layer:

$$y_m = \frac{e^{z_m^{J-1}}}{\sum_{m=1}^{M} e^{z_m^{J-1}}}$$

Training

Loss function

- We need to specify a loss function to train the network (i.e., select θ).
- A natural loss function: the quadratic error function $\mathcal{E}\left(\theta; \mathbf{Y}, \widehat{\mathbf{y}}\right)$:

$$\begin{aligned}
\theta^* &= \arg\min_{\theta} \mathcal{E}\left(\theta; \mathbf{Y}, \widehat{\mathbf{y}}\right) \\
&= \arg\min_{\theta} \sum_{l=1}^{L} \mathcal{E}\left(\theta; y_l, \widehat{y}_l\right) \\
&= \arg\min_{\theta} \frac{1}{2} \sum_{l=1}^{L} \left\| y_l - g\left(\mathbf{x}_l; \theta\right) \right\|^2
\end{aligned}$$

- Where from do the observations $\mathbf{Y}$ come? Observed data vs. simulated epochs.
- Initial θ come from a normal distribution $\mathcal{N}(0, \sigma)$. For example: $\sigma = 4\sqrt{\frac{2}{n_{input}+n_{output}}}$, but other choices are possible.

Alternative loss functions

- Other loss functions can be used.
- For instance, we can add regularization terms:
  1. ℓ_1 (LASSO): $\lambda \sum_{i=1} |\theta_i|$.
  2. ℓ_2 (ridge regression, aka as Tikhonov regularization): $\lambda \sum_{i=1} \theta_i^2$.
  3. A combination of both norms (elastic net): $\lambda_1 \sum_{i=1} |\theta_i| + \lambda_2 \sum_{i=1} \theta_i^2$.

Backpropagation

- We can easily calculate $\mathcal{E}(\theta^*; Y, \widehat{\mathbf{y}})$ and $\nabla\mathcal{E}(\theta^*; Y, \widehat{\mathbf{y}})$ for a given θ^*.
- In particular, for the gradient, we use *backpropagation* (Rumelhart *et al.*, 1986):

$$\begin{aligned}
\frac{\partial\mathcal{E}\left(\theta; y_l, \widehat{y}_l\right)}{\partial\theta_0} &= y_l - g\left(\mathbf{x}_l; \theta\right) \\
\frac{\partial\mathcal{E}\left(\theta; y_l, \widehat{y}_l\right)}{\partial\theta_m} &= \left(y_l - g\left(\mathbf{x}_l; \theta\right)\right)\phi\left(z_m\right), \text{ for } \forall m \\
\frac{\partial\mathcal{E}\left(\theta; y_l, \widehat{y}_l\right)}{\partial\theta_{n,m}} &= \left(y_l - g\left(\mathbf{x}_l; \theta\right)\right)\theta_m x_n \phi'\left(z_m\right), \text{ for } \forall n, m
\end{aligned}$$

  where $\phi'(z)$ is the derivative of the activation function.

- The derivative $\phi'(z)$ will be trivial to evaluate if we use a ReLU. Also, modern libraries use automatic differentiation, which interacts particularly well with backpropagation.
- Backpropagation will be particularly important when we use multiple layers.

An example

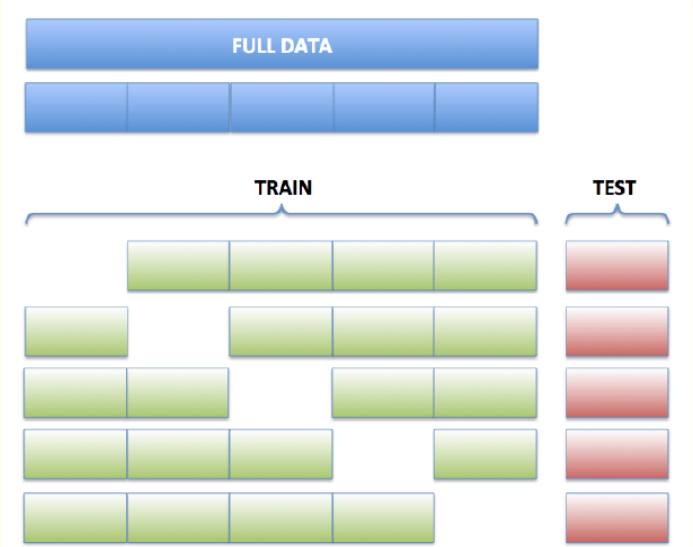
- Let us go back to our simple function $x^3 + x^2 - x - 1$.
- Let us train a 3-layer network.
- Simple code in `Matlab`.
- Suggested exercise: write an equivalent code in `Python` with `PyTorch` or `JAX`.

Architecture design

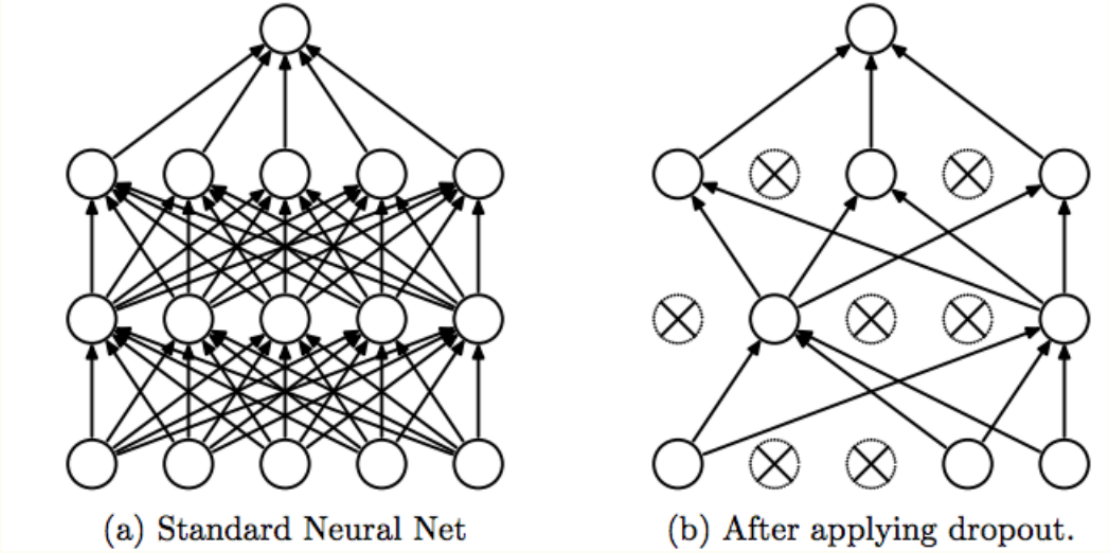
Architecture design

- Before, we have taken many aspects of the network architecture as given.
- But in practice, you need to design them (hence, importance of having access to a good deep learning library).
- Choices ("hyperparameters"):
  1. $\phi(\cdot)$: activation function.
  2. M: number of neurons.
  3. J: number of layers.
  4. Number and size of epochs.
- Notation for whole architecture: $\mathcal{A}$.
- Use $\mathcal{E}\left(\theta; \mathbf{Y}, \widehat{\mathbf{y}}\right)$ with some form of regularization (ℓ_1 or ℓ_2), cross-validation, or dropout.

Cross-validation



Dropout



(a) Standard Neural Net

(b) After applying dropout.

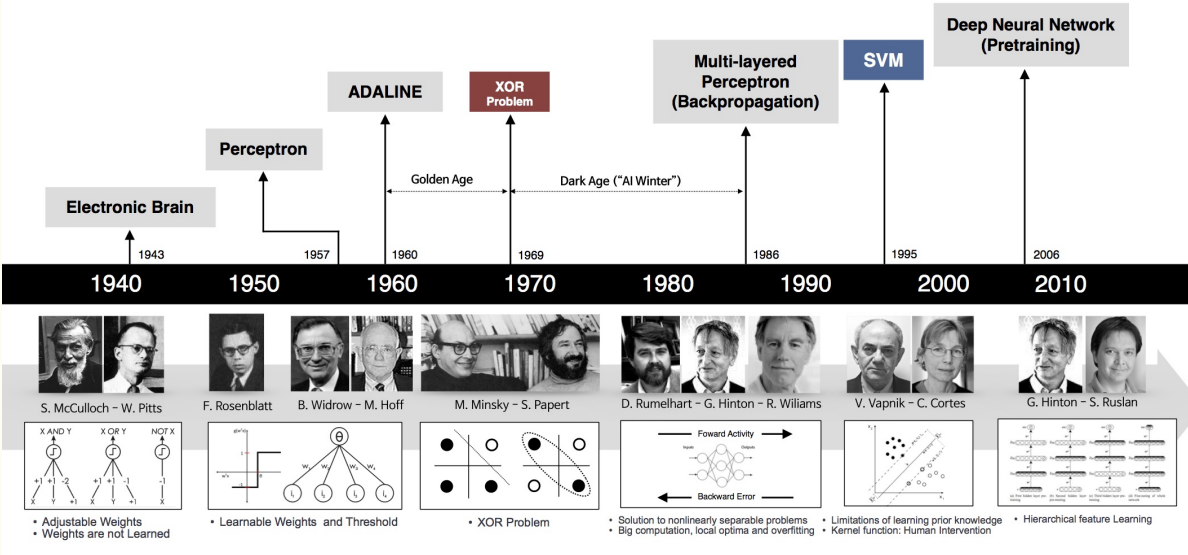
An online illustration

- We can play with many of these hyperparameters easily with the right libraries.
- Nothing substitutes practice.
- An interesting additional webpage: https://playground.tensorflow.org/.
- You can play with all the aspects of the architecture in several standard problems (from easy to challenging).
- Spend some time with this webpage!

Further ideas

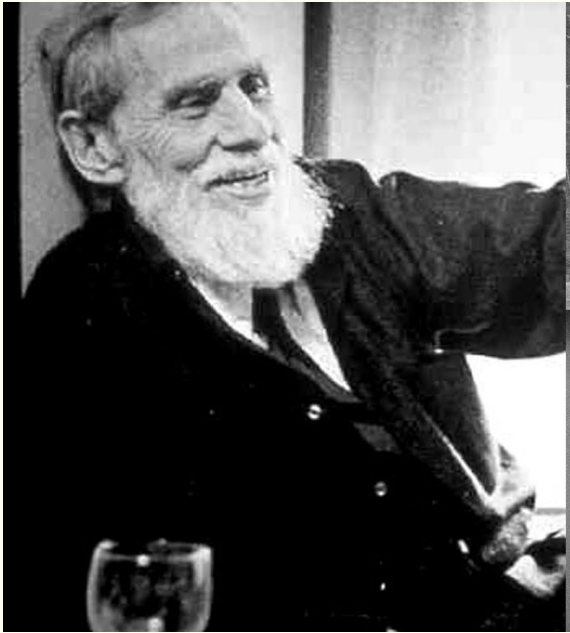
- Principles:
  1. Trade-off error/computational time.
  2. Better to err on the side of too many M.
- Double descent phenomenon (we will come back to this point later).

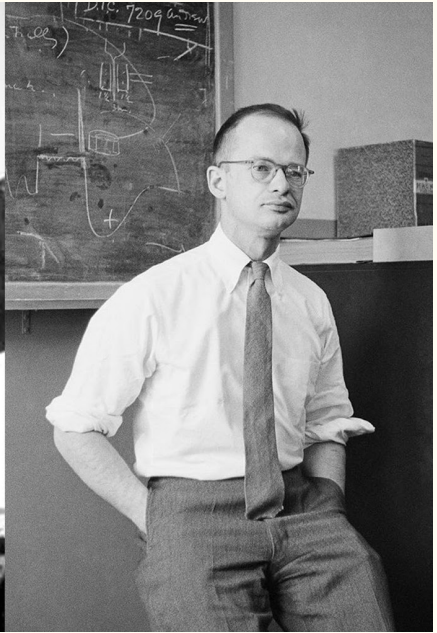
Appendix: Historical background


Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
XOR Problem
ADALINE
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

A bit of history, I

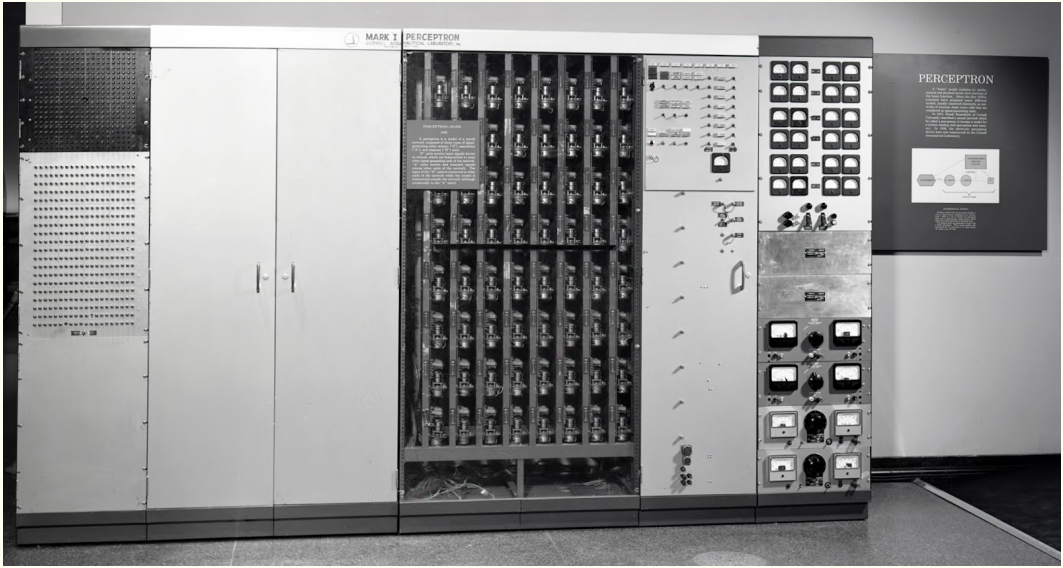
- Original idea of neural networks goes back to 1943: Warren McCulloch (1898-1969) and Walter Pitts (1923-1969): "A Logical Calculus of the Ideas Immanent in Nervous Activity."
- Inspired by:
  1. Turing's ideas on computation. Much of it developed in detail in "Computing Machinery and Intelligence," which is arguably the most influential paper in the history of Computer Science.
  2. The work on mathematical biology of Nicolas Rashevsky (1899-1972), Pitts's advisor.
  3. Propositional logic by Alfred North Whitehead and Bertrand Russell.
- Donald Hebb (1949) proposes an updated rule modifying the connection strengths between neurons (i.e., Hebbian learning).





A bit of history, II

- Perceptron by Frank Rosenblatt (1928-1971) in the late 1950s and early 1960s: the simplest feedforward neural networks that yields a universal approximator.
- However, `XOR` problem identified by Minsky and Papert (1969) led to a move toward expert systems in AI (although scope of `XOR` problem was misunderstood at the time).
- Neural networks enjoyed a brief spike of popularity in the late 1980s and early 1990s, but largely abandoned by late 1990s.


MARK I PERCEPTRON
PERCEPTRON

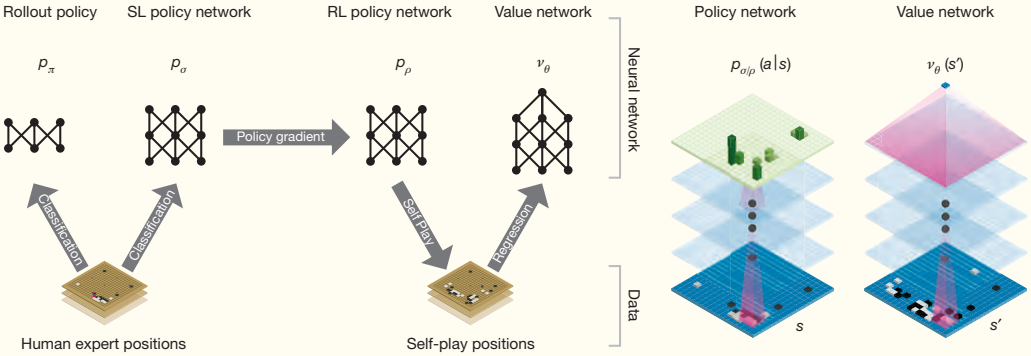
Current interest

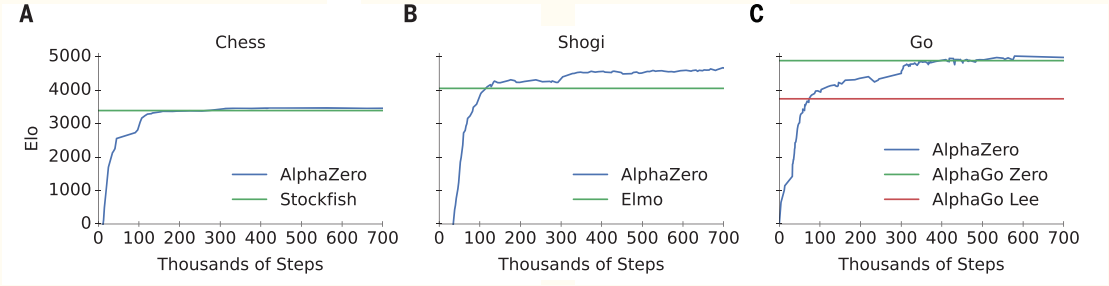
- Neural networks revived in the second half of the 2000s.
- Why?
  1. Suddenly, the large computational and data requirements required to train the networks efficiently became available at a reasonable cost.
  2. New algorithms such as *backpropagation* through stochastic gradient descent became popular (although they were already known).
- Some well-known successes (Krizhevsky, Sutskever, and Hinton, 2012) and industrial applications: deep learning quickly replaced SVM, random forest, and gradient boosted trees as most powerful learning algorithm.
- Currently, neural networks are among the most active areas of research in computer science and applied math.

AlphaGo

- Big splash: AlphaGo vs. Lee Sedol in March 2016.
- Silver *et al.* (2018): now applied to chess, shogi, Go, and StarCraft II.
- Check also:
  1. https://deepmind.com/research/alphago/.
  2. https://www.alphagomovie.com/
  3. https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii
- Very different than Deep Blue vs. Kasparov (1997): expert systems of AI.
- New and surprising strategies.
- However, you need to keep this accomplishment in perspective.


ALPHAGO
01:58:30
LEE SEDOL
01:59:11
Mind
Match
AlphaGo
Lee Sedol


Rollout policy
SL policy network
RL policy network
Value network
p_π
p_σ
p_ρ
v_θ
Policy gradient
Classification
Classification
Self Play
Regression
Neural network
Data
Human expert positions
Self-play positions
Policy network
Value network
$p_{\sigma/\rho}(a|s)$
$v_\theta(s')$
s
s'


A
Chess
Elo
5000
4000
3000
2000
1000
0
AlphaZero
Stockfish
0 100 200 300 400 500 600 700
Thousands of Steps
B
Shogi
AlphaZero
Elmo
0 100 200 300 400 500 600 700
Thousands of Steps
C
Go
AlphaZero
AlphaGo Zero
AlphaGo Lee
0 100 200 300 400 500 600 700
Thousands of Steps