# Why is Deep Learning so effective?

Matilde Marcolli and Doris Tsao

Ma191b Winter 2017
Geometry of Neuroscience

## The unreasonable effectiveness of deep learning

This lecture is based entirely on the paper:

Reference:

- Henry W. Lin and Max Tegmark, *Why does deep and cheap learning work so well?*, arXiv:1608.08225

General Question: Why does deep learning work so well?

## Summary

- Theorems proving that neural networks can approximate arbitrary (smooth) functions
- However, one expects large complexity
- Observed behavior: "cheap learning", exponentially fewer parameters than "generic case"
- Proposed explanation: the laws of physics select a particular class of functions that are sufficiently "mathematically simple" to allow "cheap learning" to work.
- Similarities between Deep Learning and Statistical Mechanics

## Characteristics of a neural network

- **Expressibility**: the class of functions the neural network can compute/approximate
- **Efficiency**: the number of parameters required to approximate a given function (or amount of resources used)
- **Learnability**: time required for the network to learn good parameters for approximating a function

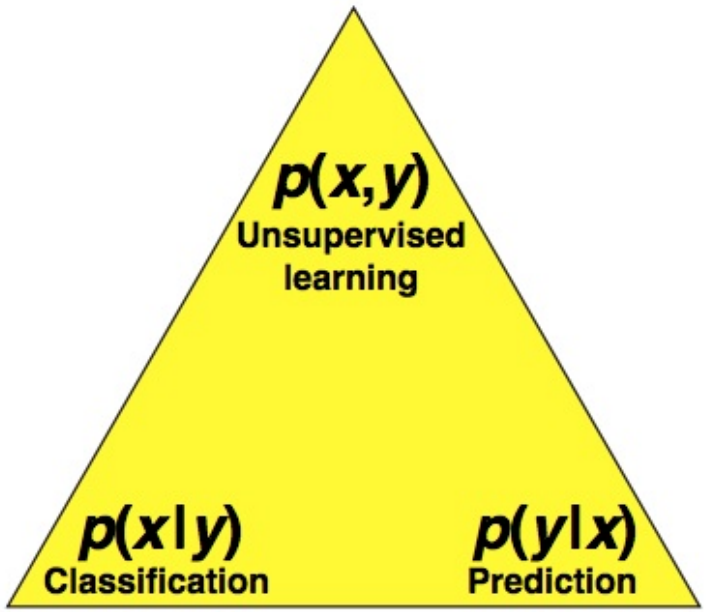
Apparent paradox: neural networks approximate functions well in practice, even though set of functions much larger than set of possible networks

Plausible Answer: the functions we need to compute to describe the real world only come from a much smaller subset

First Step: Shallow Neural Networks

• this case already shows simplifications that occur due to symmetries, locality, and algebraicity (polynomial functions)

• goal: compute a probability distribution $p$ given data sets for random vectors $x$ and $y$: viewing $y$ as stochastic function of $x$ and estimating $p(x|y)$ or $p(y|x)$, or approximate joint distribution $p(x, y)$ without causality assumptions




(Figure from Lin–Tegmark)

## Simplifying features of the probability distribution

- expect $p(y|x)$ to have a simpler form than $p(x|y)$

random vector $y$ refers to what is measured (pixels in an image; magnetization values of a physical system; etc.) usually subject to symmetries and follows a simple physical model, while $x$ (a category of images like "cats", "dogs", "faces"; a type of metal in a physical system) does not have any special shape or symmetry

- Bayes' theorem

$$p(x|y) = \frac{p(y|x)p(x)}{\sum_{x'} p(y|x')p(x')}$$

with $p(x)$ a priori probability distribution of $x$

- Consider the quantities

$$H_x(y) = -\log p(y|x) \quad \text{and} \quad \mu_x = -\log p(x)$$

*Hamiltonian* in physics or *self-information* in information theory

- Boltzmann form of the probability distribution

$$p_x(y) = p(x|y) = \frac{1}{Z(y)} e^{-(H_x(y)+\mu_x)}, \quad \text{with} \quad Z(y) = \sum_x e^{-(H_x(y)+\mu_x)}$$

recasting of Bayes formula as statistical physics

- assemble as a vector $p = (p_x)_{x \in X}$ over finite set $X$

$$p(y) = \frac{1}{Z(y)} e^{-(H(y)+\mu)},$$

with notation $e^{-v} = (e^{-v_1}, \ldots, e^{-v_n})$ for $v = (v_1, \ldots, v_n)$

## How well is $p(y)$ approximated by a neural network?

- $n$-layered feedforward neural network maps vectors through a sequence of linear and non-linear transformations

$$f(v) = \sigma_n A_n \sigma_{n-1} \cdots \sigma_2 A_2 \sigma_1 A_1 \, v$$

$A_i$ = affine transformations $A_i v = M_i v + b_i$, matrices $M_i$ and "bias vectors" $b_i$;
$\sigma_i$ = nonlinear transformations

- different possible choices of $\sigma_i$: the softmax case

$$\sigma(v) = \frac{e^v}{\sum_i e^{v_i}}$$

- Boltzmann form as softmax:

$$p(y) = \sigma(-H(y) - \mu).$$

## Computing the Hamiltonian

- this reduces the question to computing Hamiltonian in a neural network
- if can get Hamiltonian, also get $p(y)$ by adding a softmax layer to the network
- when is a Hamiltonian computable by a neural network?
- first reasonable assumption: approximation by polynomials in the form of power series expansion

$$H_x(y) = h + \sum_i h_i y_i + \sum_{i \leq j} h_{ij} y_i y_j + \sum_{i \leq j \leq k} h_{ijk} y_i y_j y_k + \cdots$$

if $y$ has $n$ components $y_i$, then there are $(n+d)!/(n!d!)$ terms of degree up to $d$

• to efficiently approximate a polynomial in a neural network need an efficient approximation of multiplication using a small number of neurons, then repeated additions and multiplications give polynomials

• case with *continuous* input variables

• Using a *smooth* non-linearity, for instance the *sigmoid activation function*

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

consider a neural network of the form

$$f = A_2 \sigma A_1$$

input layer dim=2; hidden layer dim=4; output layer dim=1

• Taylor expansion of $\sigma$

$$\sigma(u) = \sigma_0 + \sigma_1 u + \sigma_2 u^2 + \cdots$$

assume $\sigma_2 \neq 0$

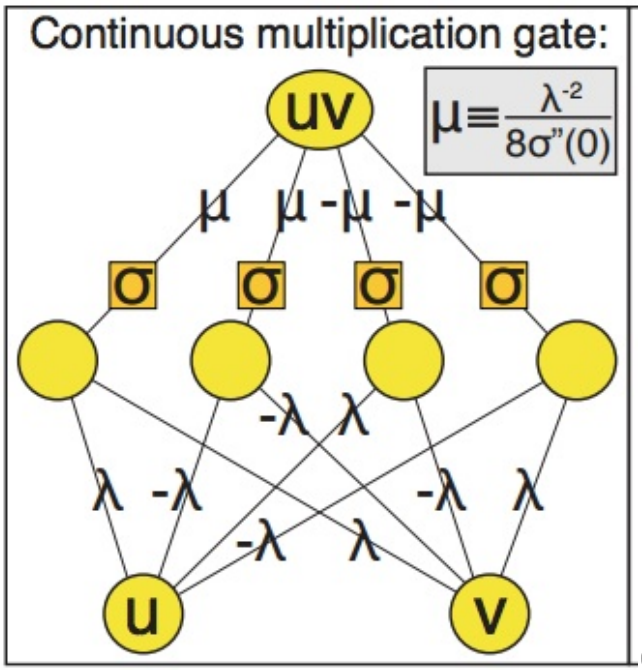
- the quantity $m(u, v)$ given by

$$m(u, v) = \frac{\sigma(u+v) + \sigma(-u-v) - \sigma(u-v) - \sigma(-u+v)}{8\sigma_2}$$

is an approximation to multiplication because it is

$$m(u, v) = uv(1 + \mathcal{O}(u^2 + v^2))$$

- multiplication approximation is good when $u, v$ are small: can make them small by scaling $A_1 \mapsto \lambda A_1$, then compensate the scaling with $A_2 \mapsto \lambda^{-2} A_2$; better approximations for $\lambda \to \infty$




(Figure from Lin–Tegmark)

- case with *discrete* input variables

- take $y = (y_i)_{i=1}^n$ vector of *binary* variables $y_i \in \{0, 1\}$; since $y_i^2 = y_i$ simpler form of Hamiltonian

$$H_x(y) = h + \sum_i h_i y_i + \sum_{i<j} h_{ij} y_i y_j + \sum_{i<j<k} h_{ijk} y_i y_j y_k + \cdots$$

(with other coefficients); finite sum of $2^n$ terms, final term $h_{1\cdots n} y_1 \cdots y_n$

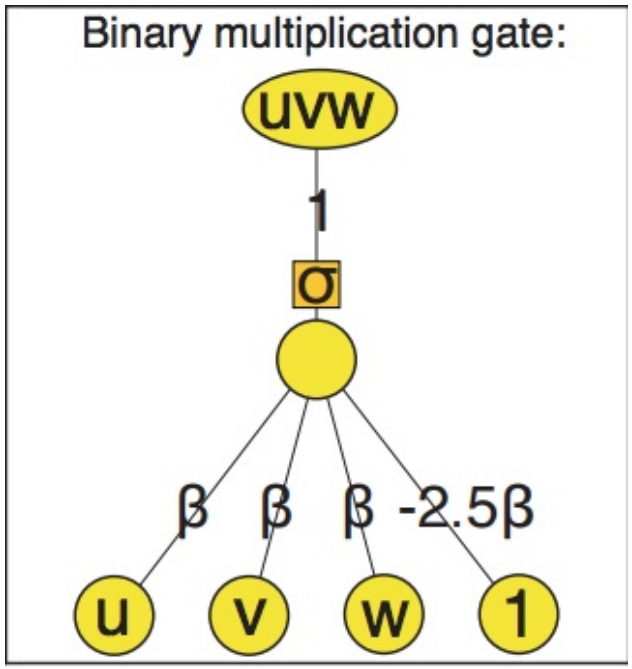
- parameters $h_{i_1 \ldots i_k}$ completely determine $H_x(y)$

- in continuous case multiplication approximation two variables at a time: many multiple layers to approximate polynomials

- discrete binary variables case: $H(y)$ in only three layers (second layer evaluates bit products, third layer linear combinations)
- multiplication approximators here takes product of an arbitrary number $n$ of bits in one step: finite number $\leq n^4$ of polynomial coefficients
- basic idea: for binary variables information on product from sum: e.g. $y_1 y_2 y_3 = 1$ if and only if $y_1 + y_2 + y_3 = 3$

- using sigmoid function $\sigma(u) = (1 + e^{-u})^{-1}$ compute product of a set $K$ of bits, with $k = \#K$, by

$$\prod_{i \in K} y_i = \lim_{\beta \to \infty} \sigma\left(-\beta(k - \frac{1}{2} - \sum_{i \in K} y_i)\right)$$




(Figure from Lin–Tegmark)

square= apply $\sigma$; circle= add; labelled line= multiply by label constant; 1= bias term

- Summary: approximation of Hamiltonian in neural network with number of neurons proportional to number of multiplications (continuous case) or to number of terms (discrete binary case)
- still too large a class of functions: continuous case $(n+d)!/(n!d!)$ coefficients for degree $d$ in $n$ variables; discrete binary case all functions are polynomial
- additional reasonable requirements:
  - Low order polynomials
  - Locality
  - Symmetry

- Low order:
  - many probability distributions can be approximated by multivariable Gaussians

$$p(y) = e^{h + \sum_i h_i y_i - \sum_{ij} h_{ij} y_i y_j}$$

    So can consider Hamiltonians that are *quadratic* polynomials
  - *maximum entropy* probability distribution with assigned constraints on lower momenta, expectation values $\langle y_1^{\alpha_1} \cdots y_n^{\alpha_n} \rangle$ gives polynomial Hamiltonian degree $\leq \sum_i \alpha_i$
  - translations, rotations, scaling are *linear* operations; Fourier transform also

- Locality:
  - in discretization of physical systems: nearest neighbor approximations
  - degree $d$ of Hamiltonian no greater than the number of neighbors in a spin system (binary variables)

- Markov networks: spins at vertices, edges= dependencies, $N_c$ = min number of cliques whose union covers network, $S_c$ = size of largest clique: number of required neurons $\leq N_c 2^{S_c}$

- for fixed $S_c$ linear in $N_c \sim$ number of vertices, so locality: number of neurons scales linearly in number of spin variables $n$

- Symmetry:
  - further reduces number of parameters that describe Hamiltonian
  - typical invariance of probability distribution: translations and rotation
  - example: $f(v)$ linear vector valued, if translation invariant then implementable by a convolution (just $n \log_2(n)$ instead of $n^2$ parameters, via Fast Fourier Transform)

Second Step: Increasing Depth of Neural Network

- Depth improvements related to:
  - hierarchical/compositional structure
  - no good way of "flattening" neural networks reflecting these structures

- Dynamics in classical statistical physics well described by Markov processes
  - so it is reasonable to assume have a set of data vectors $x_i$ with probability distributions $p_i = (p_i)_x = p(x_i)$ determined by a Markov process

$$p_i = M_i p_{i-1}, \quad \Rightarrow \quad p_n = M_n M_{n-1} \cdots M_1 p_0$$

  - generative process is a matrix product $M_n \cdots M_1$

• Reversing generative hierarchy: goal is to obtain information on the input $x_0$ from the output $x_n$, when the generative process is a Markov process $p_i = M_i p_{i-1}$, to obtain best estimate of $p(x_0|x_n) = p(x|y)$

• generative process depends on a limited number of parameters (parameters of the Markov model, $M_i$)

• while most functions are indistinguishable from random functions, most images look like noise, most numbers look like random numbers, etc.

• Hamiltonians of simple physical systems are very non-random; Hamiltonians of probability distributions associated to data sets like linguistic text/structured images/music are also far from random

• notions of complexity: Kolmogorov versus Gell-Mann complexity (later discussion)

Sufficient Statistics: reversing the generative hierarchy

- given $p(x|y)$ a *sufficient statistics* $T(y)$ is defined by

$$p(x|y) = p(x|T(y)),$$

$T(y)$ contains all the information about $y$ that is needed to determine all the information about $x$ that depend on $y$

- *minimal* sufficient statistics $T_{min}(y)$ is a sufficient statistics for any other sufficient statistics $T(y)$
- given a sufficient statistics $T(y)$ there is a function $f$ such that

$$T_{min}(y) = f(T(y))$$

$T_{min}$ is an "information distiller" (optimal data compression retaining all information relevant to $x$)

## Sufficient statistics and Markov chains

- $x_0 \mapsto x_i \mapsto \cdots x_i \mapsto \cdots x_n$ generated by Markov process
- $T_i =$ minimal sufficient statistics for $p(x_i|x_n)$
- there are functions $f_i$ with $T_i = f_i \circ T_{i+1}$
- this means the generative hierarchy can be optimally reversed one step at a time, with $f_i$ optimally undoing the $i$-th step

## More details

- because Markov process, by Bayes formula have

$$p(x_i|x_{i+k}, x_{i+1}) = \frac{p(x_{i+k}|x_i, x_{i+1})p(x_i|x_{i+1})}{p(x_{i+k}|x_{i+1})}$$

$$= \frac{p(x_{i+k}|x_{i+1})p(x_i|x_{i+1})}{p(x_{i+k}|x_{i+1})} = p(x_i|x_{i+1})$$

- get from this

$$p(x_i|x_n) = \sum_{x_{i+1}} p(x_i|x_{i+1}, x_n)p(x_{i+1}|x_n) = \sum_{x_{i+1}} p(x_i|x_{i+1})p(x_{i+1}|T_{i+1}(x_n))$$

only depending on $x_n$ through $T_{i+1}(x_n)$, so $T_{i+1}$ sufficient statistics for $p(x_i|x_n)$

- know $T_i$ is minimal sufficient statistics, so there is a function with $T_i = f_i \circ T_{i+1}$

- with $f_0(T_0(x_n)) = p(x_0, T_0(x_n))$ and $f_n(x_n) = T_{n-1}(x_n)$ get

$$p(x_0|x_n) = f_0 \circ f_1 \circ \cdots \circ f_n(x_n)$$

- structure of inference process reflects structure of generative process: neural network approximating $p(x|y)$ approximates a composition of functions $f_0 \circ \cdots \circ f_n$
- here is the main reason why depth of neural network matters: approximation of a composition of $n$ functions works best in a network with $\geq n$ hidden layers
- heuristic idea: the functions $f_i$ compress data into forms with increasing invariance (eliminating irrelevant transformations)

## Approximate Information Distillation

- minimal sufficient statistics are practically difficult to compute (some optimal data compression problem) $\Rightarrow$ characterization of sufficient statistics via information theory
- mutual information random variables $X, Y$

$$\mathcal{I}(X, Y) = \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

- data processing inequality (data compression)

$$\mathcal{I}(x, y) \geq \mathcal{I}(x, f(y))$$

sufficient statistics $T(y)$ characterized by

$$\mathcal{I}(x, y) = \mathcal{I}(x, T(y))$$

because retaining all the information about $y$ that is relevant to $x$

- Note: sometimes useful $f$ information distillation not sufficient statistic, loss of mutual information but significant reduction in complexity of Hamiltonian $H_x(f(y))$ (examples from cosmology)

## Renormalization

• Renormalization in statistical physics: vector $y$ of random variables (microscopic degrees of freedom) and $R =$ coarse graining operator that leaves Hamiltonian invariant up to change in parameters

• probability distribution (Boltzmann) specified by Hamiltonian $H_x(y)$, with parameter vector $x$

• $H_x(y)$ transformed to $H_{r(x)}(R(y))$

• iteration: $H_{r^n(x)}(R^n(y))$ and probability distribution $p(x|R^n(y))$ (composition of functions)

• but ... more like "supervised learning": *specified* features like long wavelengths, large momenta, macroscopic degrees of freedom...

• (more discussion on deep learning and renormalization to follow)

## No flattening

• Markov generative models give $p(x|y)$ as a composition of simpler functions $f_0 \circ f_1 \circ \cdots \circ f_n(y)$

• approximate each $f_i$ with a (shallow) neural network and (deep) stack of those to approximate composition $f_0 \circ f_1 \circ \cdots \circ f_n$

• this works, but is it *optimal*?

• if computed by shallower networks, does the flattening increase cost decreasing efficiency?

• given an $N$-layered neural network $f$, a flattening $f_\epsilon^\ell$ is an $\ell$-layered network with $\ell < N$ that approximates $f$ up to an $\epsilon$-error (in a suitable norm)

• *neuron-efficient flattening* if dimension $N_k$ of $k$-th inner layer of $f_\epsilon^\ell$ (number of neurons in layer) is less than for $f$

• *synapse efficient flattening* if number $N_s$ of non-zero entries of weight matrices of $f_\epsilon^\ell$ smaller than for $f$

• flattening cost of a network

$$C_n(f, \ell, \epsilon) = \min_{f_\epsilon^\ell} \frac{N_n(f_\epsilon^\ell)}{N_n(f)} \quad C_s(f, \ell, \epsilon) = \min_{f_\epsilon^\ell} \frac{N_s(f_\epsilon^\ell)}{N_s(f)}$$

• no flattening for $f$ if $C_n > 1$ or $C_s > 1$ (i.e. flattening always comes at a cost and efficient flattening is not possible)

• several known examples: e.g. a family of multivariable polynomials with exponential flattening cost; also exponential cost for tree-like hierarchical compositional forms; differential geometric methods used to shown flattening is exponentially inefficient