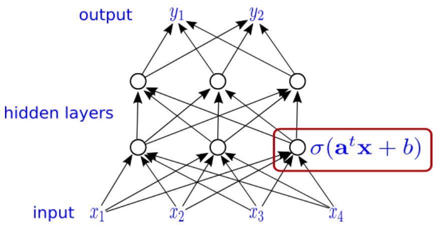
Statistical theory for deep neural networks

Lectures 1 & 2



Johannes Schmidt-Hieber

The problem

general belief that no or little theory can be developed for modern deep network architectures

- complex data structures $\rightsquigarrow$ no available statistical models
- combination of intricate network architectures with various regularization methods
- fitting a network is a non-linear problem in the network parameters
- non-convex function class
- ...

Why theory?

What is the use of theoretical results in a field that is (successfully) driven by trial and error?

- understand why deep learning works
- deep learning is a chaotic field (thousands of publications) $\rightsquigarrow$ mathematical theory can be useful to extract key concepts
- comparison with other methods
- selection of tuning parameters
- detecting limitations of deep learning
- improvements
- hybrid methods

organization of the course

Lectures:

- Theory for shallow networks
- Advantages of additional layers
- Statistical theory for deep ReLU networks
- Overparametrization

Shallow networks

- shallow neural network with one output is a function $f : \mathbb{R}^d \to \mathbb{R}$ of the form

$$f(\mathbf{x}) = \sum_{j=1}^{m} c_j \sigma\left(\mathbf{w}_j^\top \mathbf{x} + v_j\right), \quad \mathbf{w}_j \in \mathbb{R}^d,\ v_j, c_j \in \mathbb{R}.$$

- activation function $\sigma : \mathbb{R} \to \mathbb{R}$

Feedforward neural networks

- for $\mathbf{v} = (v_1, \dots, v_r)^\top, \mathbf{y} = (y_1, \dots, y_r)^\top \in \mathbb{R}^r$, define the shifted activation function $\sigma_{\mathbf{v}} : \mathbb{R}^r \to \mathbb{R}^r$ as

$$\sigma_{\mathbf{v}} = (\sigma(y_1 - v_1), \dots, \sigma(y_r - v_r))^\top.$$

- network architecture $(L, \mathbf{p})$
  - positive integer L called number of hidden layers/depth
  - width vector $\mathbf{p} = (p_0, \dots, p_{L+1}) \in \mathbb{N}^{L+2}$

Neural network with architecture $(L, \mathbf{p})$ is

$$f(\mathbf{x}) = W_L \sigma_{\mathbf{v}_L} W_{L-1} \sigma_{\mathbf{v}_{L-1}} \cdots W_1 \sigma_{\mathbf{v}_1} W_0 \mathbf{x},$$

- W_i is a $p_i \times p_{i+1}$ weight matrix
- $\mathbf{v}_i \in \mathbb{R}^{p_i}$ is a shift vector

Feedforward neural networks

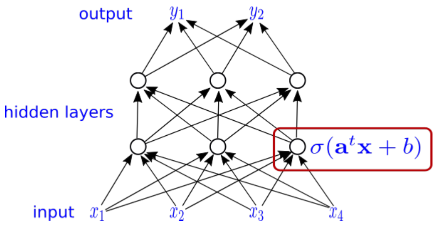
Neural network:

$$f(\mathbf{x}) = W_L \sigma_{\mathbf{v}_L} W_{L-1} \sigma_{\mathbf{v}_{L-1}} \cdots W_1 \sigma_{\mathbf{v}_1} W_0 \mathbf{x},$$

Comments:

- feedforward $\rightsquigarrow$ information is passed in one direction through the network
- network functions are build by alternating matrix-vector multiplications with the action of the non-linear activation function σ
- network architecture is given
- parameters generating the underlying function class are the matrices $W_0, \dots, W_L$ and the shift bectors $v_1, \dots, v_L$

Graph representation



- in CS, neural networks are introduced via graph representation
- nodes in the graph (also called *units*) are arranged in layers
- input layer is the first layer and the output layer the last layer
- layers that lie in between are called hidden layers
- number of hidden layers corresponds to L and the number of units in each layer generates the width vector $\mathbf{p}$
- Each node/unit in the graph representation stands for operation $\sigma(\mathbf{a}^t \cdot + b)$

Special types

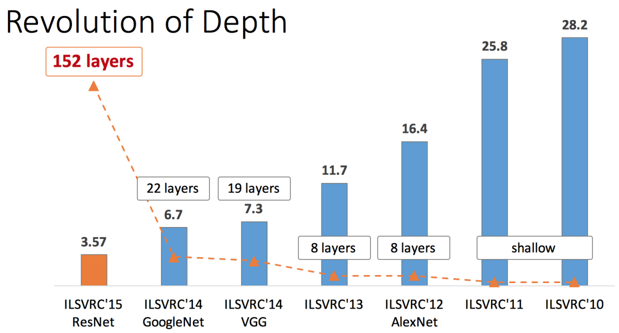
Neural network:

$$f(\mathbf{x}) = W_L \sigma_{\mathbf{v}_L} W_{L-1} \sigma_{\mathbf{v}_{L-1}} \cdots W_1 \sigma_{\mathbf{v}_1} W_0 \mathbf{x},$$

Comments:

- network is called sparse if W_i are sparse matrices
- i-th layer is fully connected $\rightsquigarrow$ W_i is dense
- for $L = 1$ network coincides with shallow networks
- if $L > 1$, network is called deep

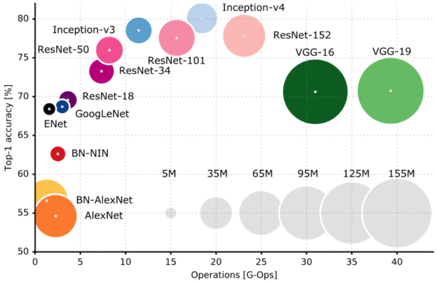
Depth



Source: Kaiming He, Deep Residual Networks

- Networks are deep
  - version of ResNet with 152 hidden layers
  - networks become deeper

High-dimensionality



Source: arxiv.org/pdf/1605.07678.pdf

- Number of network parameters is larger than sample size
    - AlexNet uses 60 million parameters for 1.2 million training samples

functions generated by shallow networks

Consider function class

$$\mathcal{F}_{m,\sigma} := \left\{ f = \sum_{j=1}^{m} c_j \sigma\left(\mathbf{w}_j^\top \cdot + v_j\right) \; : \; \mathbf{w}_j \in \mathbb{R}^d, v_j, c_j \in \mathbb{R} \right\}.$$

problems:

- how large is this class?
- how well can we approximate functions of a specific smoothness?
- or the function $f(x_1, x_2) = x_1 x_2$?

universal approximation

$$\mathcal{F}_{m,\sigma} := \left\{ f = \sum_{j=1}^{m} c_j \sigma\left(\mathbf{w}_j^\top \cdot + v_j\right) \; : \; \mathbf{w}_j \in \mathbb{R}^d, v_j, c_j \in \mathbb{R} \right\}.$$

- functions in the class $\mathcal{F}_{m,\sigma}$ have $m(d+2)$ real parameters
- nested spaces, e.g. $\mathcal{F}_{m,\sigma} \subseteq \mathcal{F}_{m',\sigma}$ whenever $m' \geq m$.

Universal approximation property: Shallow networks with activation function σ have the universal approximation property if for any $\varepsilon > 0$ and any continuous function f on $[0,1]^d$, there exists an integer $m = m(f, \varepsilon)$, such that

$$\inf_{g \in \mathcal{F}_{m,\sigma}} \|f - g\|_{L^\infty([0,1]^d)} \leq \varepsilon.$$

reduction to ridge functions

- many proofs first show universal approximation in dimension one
- univariate functions $\{\sigma(w \cdot + v) : w, v \in \mathbb{R}\}$ span the space of continuous functions
- statement does not involve scalar products anymore

afterwards, it is enough to show that the function space spanned by so called ridge functions

$$f = \sum_{j=1}^{m} g_j(\mathbf{w}_j^\top \cdot)$$

with g_j univariate and continuous has the universal approximation property

universal approximation for univariate functions

Theorem: Shallow networks with smooth activation function that is not a polynomial have universal approximation property for $d = 1$.

Proof:

- $\Delta_h^1 \sigma(t) := (\sigma(t + xh) - \sigma(t))/h$
- $\Delta_h^k \sigma(t) := \Delta_h^1 (\Delta_h^{k-1} \sigma)(t)$
- definition of the k-th derivative $\rightsquigarrow$

$$\left| \frac{\Delta_h^k \sigma(t)}{x^k} - \sigma^{(k)}(t) \right| \to 0, \quad \text{as } h \to 0$$

universal approximation for univariate functions (ctd.)

- σ not a polynomial $\leadsto$ there exists for each k a real number t_k with $\sigma^{(k)}(t_k) \neq 0$
- multiplying with x^k and division $\sigma^{(k)}(t_k)$ yields

$$\left| \frac{\Delta_h^k \sigma(t_k)}{\sigma^{(k)}(t_k)} - x^k \right| \to 0, \quad \text{as } h \to 0.$$

- for any $h > 0$, $(\sigma^{(k)}(t_k))^{-1} \Delta_h^k \sigma(t_k)$ can be realized by a shallow network with $k+1$ units
- $\leadsto$ build networks approximating the function $x \mapsto x^k$ arbitrarily well in sup-norm
- apply Weierstrass approximation theorem $\square$

some comments on the proof

- proof provides explicit construction of networks that closely resemble polynomials
- construction requires that some parameters are extremely small and others are very large
- uses only one point of the activation function to generate a specific power
- $\rightsquigarrow$ small perturbations of the activation function can lead to completely different properties
- networks can "zoom in" at local features of the activation function
- the universal approximation theorem can be extended to continuous activation functions using local smoothing

universal approximation via Fourier transform

- Fourier transform $\mathcal{F}f(\boldsymbol{\xi}) = \int e^{-i\boldsymbol{\xi}^\top \mathbf{x}} f(\mathbf{x})\, d\mathbf{x}$
- inverse Fourier transform $\mathcal{F}^{-1}f(\mathbf{x}) = (2\pi)^{-d} \int e^{i\mathbf{x}^\top \boldsymbol{\xi}} f(\boldsymbol{\xi})\, d\boldsymbol{\xi}$
- $f = \mathcal{F}^{-1}\mathcal{F}f$
- for any complex number z, $z = |z|e^{i\phi}$ for some real number $\phi = \phi(z)$
- $\rightsquigarrow$ there exists a real valued function $\phi(\mathbf{w})$ such that $\mathcal{F}f(\mathbf{w}) = e^{i\phi(\mathbf{w})}|\mathcal{F}f(\mathbf{w})|$
- Fourier inversion $\rightsquigarrow$

$$\begin{aligned} f(\mathbf{x}) &= \frac{1}{(2\pi)^d} \operatorname{Re} \int e^{i\mathbf{w}^\top \mathbf{x}} e^{i\phi(\mathbf{w})} |\mathcal{F}f(\mathbf{w})| d\mathbf{w} \\ &= \frac{1}{(2\pi)^d} \int \cos\left(\mathbf{w}^\top \mathbf{x} + \phi(\mathbf{w})\right) |\mathcal{F}f(\mathbf{w})| d\mathbf{w} \end{aligned}$$

- discretization of the integral on the right hand side gives the structure of a shallow network with activation function cos()
- $\rightsquigarrow$ will be used later for approximation rates

Approximation rates for shallow networks

How well can we approximate a function in dependence on smoothness etc. ?

- smooth activation functions
- approximation rates using multivariate polynomials
- Barron's class

approximation rates for smooth activation function

- Mhaskar '96
- smooth activation function
- β-smooth function (in L^2-Sobolev sense)
- rate of approximation over all shallow networks with m units is $m^{-\beta/d}$ with d the dimension
- proof first approximates polynomials of ridge functions and then continues with polynomial approximation

approximation rates for arbitrary activation function

- Petrushev '99
- good approximation rates can be obtained for functions that are smoother than the activation function

Theorem: if activation function is s-smooth (Sobolev), optimal approximation rates are obtained for $s + (d-1)/2$-smooth functions

- $\rightsquigarrow$ effect becomes better as input dimension increases
- **proof:** reduce to ridge functions + approximation of Radon inversion + polynomial eigenbasis
- proof is constructive $\rightsquigarrow$ several interesting conclusions

remarks

- proofs always relate shallow networks to polynomials
- we could start directly with polynomials and would obtain the same approximation rates
- does not help to identify problems where neural networks perform better than other methods
- **Next:** Barron's result

Barron's approximation theorem

- for any sigmoidal activation function
- any $m \geq 1$,
- any function f
- define $C_f := \int |\mathbf{w}|_1 \mathcal{F}(f)(\mathbf{w}) d\mathbf{w}$
- there exist shallow network such that

$$\left\| f(\cdot) - f(\mathbf{0}) - \sum_{j=1}^{m} c_j \sigma(\mathbf{w}_j^\top \cdot + v_j) \right\| \leq \frac{2C_f}{(2\pi)^d \sqrt{m}},$$

Remarks:

- rate $m^{-1/2}$ does not depend on the dimension d
- do neural networks avoid curse of dimensionality?

On the rate

- **Recall:** $C_f = \int |\mathbf{w}|_1 |\mathcal{F}f(\mathbf{w})| d\mathbf{w}$
- indeed there is nothing special about neural networks here
- Candes '02 shows that truncated Fourier series achieves faster approximation rate
$$m^{-1/2-1/d}$$
for the same function class $\{f : C_f < \infty\}$
- gain is related to loss in Maurey's theorem

Up to now, no approximation problem has been found where shallow networks outperform Fourier series or polynomial approximation

statistical model

- combine approximation theory with statistical analysis
- given an i.i.d. sample $(\mathbf{X}_i, Y_i) \in \mathbb{R}^d \times \mathbb{R}$, $i = 1, \ldots, n$ with bounded responses $|Y_i| \leq 1$,
- want to recover the regression function

$$f(\mathbf{x}) = E\left[Y_i | \mathbf{X}_i = \mathbf{x}\right]$$

- covers binary classification
  $\rightsquigarrow Y_i \in \{0, 1\}$ and $f(\mathbf{x}) = P(Y_i | \mathbf{X}_i = x)$

oracle inequality

- $\widehat{f}$ be the empirical risk minimizer

$$\widehat{f} \in \operatorname{argmin}_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{n} \big(Y_i - f_{\boldsymbol{\theta}}(\mathbf{X}_i)\big)^2.$$

- standard exponential inequalities
  $\rightsquigarrow$ if Θ is a discrete set with cardinality $|\Theta|$, then

$$E_f\left[\|\widehat{f} - f\|_2^2\right] \le C \inf_{\theta\in\Theta} \|f - f_{\boldsymbol{\theta}}\|_2^2 + C\frac{\log|\Theta|}{n}$$

statistical bounds for shallow networks

- Barron '94
- discretizes network parameters
- study empirical risk minimizer
- $m(d+2)$ is the number of parameters
- $\log|\Theta| \lesssim m(d+2)\log n$
- oracle inequality + approximation theory $\rightsquigarrow$

$$E_f\left[\|\widehat{f} - f\|_2^2\right] \lesssim m^{-1} + \frac{m\log n}{n}.$$

  if $C_f = \int |\mathbf{w}|_1 |\mathcal{F}f(\mathbf{w})| d\mathbf{w} < \infty.$
- bias variance trade-off $\rightsquigarrow$ $m = \sqrt{n/\log n}$
- yields the rate

$$\sqrt{\frac{\log n}{n}}$$

summary

shallow networks:

- universal approximation
- approximation rates
- estimation risk bounds

no gain in terms of rates with respect to series estimators

next lecture discusses advantages of additional layers

advantages of additional layers

- localization
- approximation of polynomials with deep networks
- Kolmogorov-Arnold representation theorem
- advantages of deep ReLU networks

localization with Heaviside activation function

- no localization for shallow networks in dimension $d > 1$ (?)
- for commonly used activation functions, taking two hidden layers allows us to localize in arbitrary dimensions
- Heaviside activation function $\sigma_0 = \mathbf{1}(\cdot \geq 0)$,

$$\mathbf{1}(\mathbf{x} \in [-1,1]^d) = \sigma_0\Big(\sum_{i=1}^{d} \sigma_0(x_i + 1) + \sigma_0(-x_i + 1) - 2d + \frac{1}{2}\Big)$$

- $\leadsto$ outer neuron only gets activated **iff** all the inner neurons output one
- this is the case **iff** $-1 \leq x_i \leq 1$ for all $i = 1, \ldots, d$

localization by other activation functions

- for sigmoidal activation function

$$\sigma(\alpha x) \approx \sigma_0(x), \quad \text{for large } \alpha.$$

- for the ReLU $\sigma(x) = (x)_+$,

$$\sigma(\alpha x) - \sigma(\alpha x - 1) \approx \sigma_0(x), \quad \text{for large } \alpha.$$

- approximation quality depends on α
- $\rightsquigarrow$ results using neural networks with sigmoidal activation often have conditions on the speed at which $|\sigma(x)| \to 0$ for $x \to -\infty$, and $|1 - \sigma(x)| \to 0$ for $x \to +\infty$
- localization might be a useful property for approximation, being non-local might be helpful for the (stochastic) gradient descent

approximation of x^{2^k} with deep networks

- for a smooth activation function, the function $x \mapsto x^{2^k}$ lies in the closure of a shallow network with $2^k + 1$ units ($\nearrow$ previous lecture)
- stacking layers on top of each other, this can be reduced to $O(k)$ units in k layers
- rescaled finite second order differences

$$\frac{\sigma(t + 2xh) - 2\sigma(t + xh) + \sigma(t)}{\sigma''(t)h^2} \approx x^2.$$

a graphical proof

$$\begin{tikzcd}
 & \approx x^{2^k} & \\
\approx \sigma(t+2x^{2^{k-1}}h) \arrow[ur] & \approx \sigma(t+x^{2^{k-1}}h) \arrow[u] & \sigma(t) \arrow[ul] \\
 & \approx x^{2^{k-1}} \arrow[ul] \arrow[u] & \\
\vdots & \vdots & \vdots \\
 & \approx x^4 & \\
\approx \sigma(t+2x^2h) \arrow[ur] & \approx \sigma(t+x^2h) \arrow[u] & \sigma(t) \arrow[ul] \\
 & \approx x^2 \arrow[ul] \arrow[u] & \\
\sigma(t+2xh) \arrow[ur] & \sigma(t+xh) \arrow[u] & \sigma(t) \arrow[ul] \\
 & x \arrow[ul] \arrow[u] &
\end{tikzcd}$$

$$\begin{tikzcd}
 & \approx x^{2^k} & \\
\approx \sigma(t+2x^{2^{k-1}}h) \arrow[ur] & \approx \sigma(t+x^{2^{k-1}}h) \arrow[u] & \sigma(t) \arrow[ul] \\
\approx \sigma(t+2x^{2^{k-2}}h) \arrow[u] \arrow[ur] & \approx \sigma(t+x^{2^{k-2}}h) \arrow[ul] \arrow[u] & \sigma(t) \arrow[ul] \arrow[ull] \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\approx \sigma(t+2x^2h) \arrow[u] \arrow[ur] & \approx \sigma(t+x^2h) \arrow[ul] \arrow[u] & \sigma(t) \arrow[ul] \arrow[ull] \\
\sigma(t+2xh) \arrow[u] \arrow[ur] & \sigma(t+xh) \arrow[ul] \arrow[u] & \sigma(t) \arrow[ul] \arrow[ull] \\
 & x \arrow[ul] \arrow[u] &
\end{tikzcd}$$

improved representation theorems

- Kolmogorov-Arnold approximation theorem shows that every continuous function can be represented by a specific two-layer network
- very different structure if compared with the universal approximation theorem for shallow networks
- indicates that additional layers can lead to new features of network functions

Theorem (Braun '09): Fix $d \geq 2$. There are real numbers a, b_p, c_q and a continuous and monotone function $\psi : \mathbb{R} \to \mathbb{R}$, such that for any continuous function $f : [0,1]^d \to \mathbb{R}$, there exists a continuous function $g : \mathbb{R} \to \mathbb{R}$ with

$$f(x_1, \ldots, x_d) = \sum_{q=0}^{2d} g\Big(\sum_{p=1}^{d} b_p \psi(x_p + qa) + c_q \Big).$$

remarks

$$f(x_1,\ldots,x_d) = \sum_{q=0}^{2d} g\Big(\sum_{p=1}^{d} b_p \psi(x_p + qa) + c_q\Big).$$

- one inner function ψ and one outer function g
- inner function is independent of f
- q-dependence in the first layer comes through the shifts qa.
- right hand side can be realized by a network with two hidden layers, architecture $\mathbf{p} = (d, d, 2d+1, 1)$, and ψ being the activation function in the first layer.

link to pre-training

- inner function in the Kolmogorov-Arnold representation theorem is independent of the represented function f
- in deep learning it has been observed that the first layers build function systems which can be used for other classification problems
- exploited in pre-training where a trained deep network from a possibly completely different classification problem is taken and only the last layer is learned by the new dataset
- fact that pre-training works shows that deep networks build generic function systems in the first layers.

deep ReLU networks

we discuss several advantages of deep ReLU networks

- representation of identity
- growth of number of linear pieces
- approximation by ReLU networks with small parameters

deep ReLU networks can learn skip connections

$$\sigma(x) = \max(x, 0)$$

- projection property

$$\sigma \circ \sigma = \sigma$$

- $\rightsquigarrow$ pass a signal without change through several layers in the network
- $\rightsquigarrow$ network synchronization by adding hidden layers
- related to skip connections and ResNets
- for other activation functions it is much harder to approximate the identity

number of linear pieces of deep ReLU networks

- deep ReLU networks are piecewise linear functions of the input
- adding layers $\rightsquigarrow$ highly oscillating functions with few parameters
- consider ReLU network with two hidden layers and width vector $(1, m, 1, 1)$ of the form

$$\Big(\sum_{j=1}^{m} c_j (w_j x + v_j)_+\Big)_+$$

- $\rightsquigarrow$ number of added pieces by outer ReLU is proportional to number of zero crossings of inner function
- any ReLU network with width vector $(1, p_1, \ldots, p_L, 1)$ has at most

$$\Big(\frac{3}{2}\Big)^L \prod_{j=1}^{L} (p_j + 1)$$

pieces

example of a highly oscillating function

Functions:

- let $T : [0,1] \to [1]$,

$$T(x) := (2x) \wedge (1 - 2x) = (2x)_+ - (4x - 2)_+$$

- can be realized by shallow network with two units
- $R^k : [0,1] \to [0,1]$,

$$R^k := \underbrace{T \circ T \circ \dots T}_{k \text{ times}}$$

network representation

$$\begin{tikzcd}
& T^k(x) & \\
(2T^{k-1}(x))_+ \arrow[ur, "1"] & & (4T^{k-1}(x)-2)_+ \arrow[ul, "-1"'] \\
& T^{k-1}(x) \arrow[ul, "2"] \arrow[ur, "4"'] & \\
\vdots \arrow[ur] & & \vdots \arrow[ul] \\
\vdots & & \vdots \\
& T^2(x) \arrow[ul] \arrow[ur] & \\
(2T(x))_+ \arrow[ur, "1"] & & (4T(x)-2)_+ \arrow[ul, "-1"'] \\
& T(x) \arrow[ul, "2"] \arrow[ur, "4"'] & \\
(2x)_+ \arrow[ur, "1"] & & (4x-2)_+ \arrow[ul, "-1"'] \\
& x \arrow[ul, "2"] \arrow[ur, "4"'] &
\end{tikzcd}$$

$$\begin{tikzcd}
& T^k(x) & \\
(2T^{k-1}(x))_+ \arrow[ur, "1"] & & (4T^{k-1}(x)-2)_+ \arrow[ul, "-1"'] \\
(2T^{k-2}(x))_+ \arrow[u, "2"] \arrow[urr, "4"] & & (4T^{k-2}(x)-2)_+ \arrow[u, "-4"'] \arrow[ull, "-2"] \\
\vdots \arrow[u] \arrow[urr] & & \vdots \arrow[u] \arrow[ull] \\
\vdots & & \vdots \\
(2T^2(x))_+ \arrow[u] \arrow[urr] & & (4T^2(x)-2)_+ \arrow[u] \arrow[ull] \\
(2T(x))_+ \arrow[u, "2"] \arrow[urr, "4"] & & (4T(x)-2)_+ \arrow[u, "-4"'] \arrow[ull, "-2"] \\
(2x)_+ \arrow[u, "2"] \arrow[urr, "4"] & & (4x-2)_+ \arrow[u, "-4"'] \arrow[ull, "-2"] \\
& x \arrow[ul, "2"] \arrow[ur, "4"'] &
\end{tikzcd}$$

multiplication

how can we (approximately) multiply two inputs with a network?

- crucial problem for approximation theory
- for deep networks this can be reduced to approximation of square function $x \mapsto x^2$ via

$$xy = \left(\frac{x+y}{2}\right)^2 - \left(\frac{x-y}{2}\right)^2$$

- has a surprising answer for ReLU networks

approach from Lecture 2

network approximation of the function $x \mapsto x^2$ is very important!

- for twice differentiable activation function, we used

$$\frac{\sigma(t+2xh) - 2\sigma(t+xh) + \sigma(xh)}{h^2\sigma''(t)} \to x^2 \text{ for } h \to 0$$

- $\leadsto$ network parameters become large
- for deep ReLU networks we use a different construction

ReLU approximation of the square function

Functions:

- let $T^k : [0, 2^{2-2k}] \to [0, 2^{-2k}]$,

$$T^k(x) := (x/2) \wedge (2^{1-2k} - x/2) = (x/2)_+ - (x - 2^{1-2k})_+$$

- $R^k : [0,1] \to [0, 2^{-2k}]$,

$$R^k := T^k \circ T^{k-1} \circ \dots T^1.$$

Lemma (Telgarsky '16, Yarotski '18, SH '17):

$$\left| x(1-x) - \sum_{k=1}^{m} R^k(x) \right| \leq 2^{-m}.$$

rewriting approximation as network

$$\left| x(1-x) - \sum_{k=1}^{m} R^k(x) \right| \leq 2^{-m}.$$

deep ReLU approximation:

- m hidden layers
- $O(m)$ network parameters
- bounded parameters
- approximation 2^{-m}

shallow ReLU network

- for $x(1-x)$ a shallow ReLU network needs at least $O(2^{m/2})$ parameters to achieve approximation error 2^{-m}

multiplication with deep ReLU networks

Lemma: There exists a network Mult_m with $m+4$ hidden layers, width vector $(2, 6, 6, \ldots, 6, 1)$ and all network parameters bounded by one, such that

$$\left|\mathrm{Mult}_m(x, y) - xy\right| \leq 2^{-m}, \quad \text{for all } x, y \in [0, 1].$$

Proof:

- use polarization identity

$$xy = \left(\frac{x+y}{2}\right)^2 - \left(\frac{x-y}{2}\right)^2$$

- separation of positive and negative part
- compute $(x+y)/2$ and $(x-y)/2$ in first layer (non-negativity!)
- square network has to be incorporated twice (inefficient)

a step in the proof

$$\begin{tikzcd}
 & \sum_{k=1}^{m+1} R^k(u) + h(u) & \\
T_+(R^m(u)) \arrow[ur] & \sum_{k=1}^{m} R^k(u) + h(u) \arrow[u] & T_-^{m+1}(R^m(u)) \arrow[ul] \\
\vdots \arrow[u] \arrow[ur] \arrow[urr] & \vdots \arrow[u] & \vdots \arrow[u] \arrow[ul] \arrow[ull] \\
\vdots & \vdots & \vdots \\
T_+(R^1(u)) \arrow[u] \arrow[ur] \arrow[urr] & R^1(u) + h(u) \arrow[u] & T_-^2(R^1(u)) \arrow[u] \arrow[ul] \arrow[ull] \\
T_+(u) \arrow[u] \arrow[ur] \arrow[urr] & h(u) \arrow[u] & T_-^1(u) \arrow[u] \arrow[ul] \arrow[ull]
\end{tikzcd}$$

localization and approximation

- we have seen that with two hidden layers we can localize
- how can this be done for ReLU networks?
- goes back to Yarotsky '18
- define $\mathbf{D}(M)$ as all grid points on the grid

$$\left\{(\ell_j/M)_{j=1,\ldots,r} : \boldsymbol{\ell} = (\ell_1, \ldots, \ell_r) \in \{0, 1, \ldots, M\}^r\right\}$$

- partition of unity on unit cube

$$\sum_{\mathbf{x}_{\boldsymbol{\ell}} \in \mathbf{D}(M)} \underbrace{\prod_{j=1}^{r} (1 - M|x_j - x_j^{\boldsymbol{\ell}}|)_+}_{\text{localized functions}} = \prod_{j=1}^{r} \sum_{\ell=0}^{M} (1 - M|x_j - \ell/M|)_+ = 1,$$

local Taylor approximation

- on each localized bit ($\mathbf{a} \in \mathbf{D}(M)$) do a Taylor approximation

$$f(\mathbf{x}) \approx P_{\mathbf{a}}^{\beta} f(\mathbf{x}) := \sum_{0 \le |\boldsymbol{\alpha}| < \beta} (\partial^{\boldsymbol{\alpha}} f)(\mathbf{a}) \frac{(\mathbf{x} - \mathbf{a})^{\boldsymbol{\alpha}}}{\boldsymbol{\alpha}!} = \sum_{0 \le |\boldsymbol{\gamma}| < \beta} \mathbf{x}^{\boldsymbol{\gamma}} c_{\boldsymbol{\gamma}}$$

- this can be approximately realized by a deep ReLU network
- many technicalities occur (see the article SH '17)

approximation rate

Theorem: For any β-smooth function $f : [0,1]^r \to \mathbb{R}$ and any integers $m, N \geq 1$, there exists a ReLU network with

- depth $L \asymp m$
- width in each layer bounded by $\lesssim N$
- number of non-zero network parameters $s \lesssim Nm$

such that

$$\|\widetilde{f} - f\|_{L^\infty([0,1]^r)} \lesssim \underbrace{N2^{-m}}_{\text{small for deep networks}} + \underbrace{N^{-\frac{\beta}{r}}}_{\text{approx. rate}} .$$

remarks

$$\|\widetilde{f} - f\|_{L^\infty([0,1]^r)} \lesssim \underbrace{N2^{-m}}_{\text{small for deep networks}} + \underbrace{N^{-\frac{\beta}{r}}}_{\text{approx. rate}} .$$

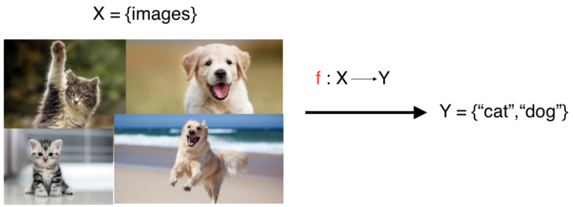
- for deep networks first term is of smaller order
- second term becomes suboptimal for large depth
- trade-off
- sparse networks

risk bounds for deep ReLU networks

Framework:

- we now study a statistical problem
- requires that we first need to specify a statistical model
- we study nonparametric regression

mathematical problem



The data are used to fit a network, i.e. estimate the network parameters.

How fast does the estimated network converge to the truth f as sample size increases?

statistical analysis

- we observe n i.i.d. copies $(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)$,

$$Y_i = f(\mathbf{X}_i) + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0,1)$$

    - $\mathbf{X}_i \in \mathbb{R}^d$, $Y_i \in \mathbb{R}$,
    - goal is to reconstruct the function $f : \mathbb{R}^d \to \mathbb{R}$
- has been studied extensively
  (kernel smoothing, wavelets, splines, ...)

the estimator

- choose network architecture $(L, \mathbf{p})$ and sparsity s
- denote by $\mathcal{F}(L, \mathbf{p}, s)$ the class of all networks with
    - architecture $(L, \mathbf{p})$
    - number of active (e.g. non-zero) parameters is s
- our theory applies to any estimator $\widehat{f}_n$ taking values in $\mathcal{F}(L, \mathbf{p}, s)$
- prediction error

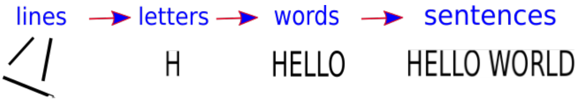
$$R(\widehat{f}_n, f) := E_f\big[\big(\widehat{f}_n(\mathbf{X}) - f(\mathbf{X})\big)^2\big],$$

  with $\mathbf{X} \overset{\mathcal{D}}{=} \mathbf{X}_1$ being independent of the sample
- study the dependence of n on $R(\widehat{f}_n, f)$

function class

- classical idea: assume that regression function is β-smooth
- optimal nonparametric estimation rate is $n^{-2\beta/(2\beta+d)}$
- suffers from curse of dimensionality
- to understand deep learning this setting is therefore useless
- $\rightsquigarrow$ make a good structural assumption on f

hierarchical structure



- Important: Only few objects are combined on deeper abstraction level
  - few letters in one word
  - few words in one sentence

function class

- We assume that

$$f = g_q \circ \ldots \circ g_0$$

  with
  - $g_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_{i+1}}$.
  - each of the d_{i+1} components of g_i is β_i-smooth and depends only on t_i variables
  - t_i can be much smaller than d_i
  - effective smoothness

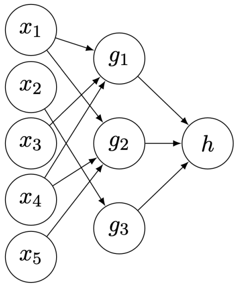
$$\beta_i^* := \beta_i \prod_{\ell=i+1}^{q} (\beta_\ell \wedge 1).$$

  - we show that **the rate depends on the pairs**

$$(t_i, \beta_i^*), \quad i = 0, \ldots, q.$$

  - similar conditions have been proposed by Horowitz & Mammen (2007), Kohler & Kryzak (2017), Bauer & Kohler (2017), Kohler & Langer (2018)

example



$$f(x_1, \ldots, x_5) = h\Big(g_1(x_1, x_3, x_4), g_2(x_1, x_4, x_5), g_3(x_2)\Big)$$

here: $q = 1$, $(d_0, d_1, d_2) = (5, 3, 1)$, $t_0 = t_1 = 3$.

main result

Theorem: If

(i) depth $\asymp \log n$

(ii) width $\geq$ network sparsity $\asymp \max_{i=0,\dots,q} n^{\frac{t_i}{2\beta_i^*+t_i}} \log n$

Then, for any network reconstruction method $\widehat{f}_n$,

$$\text{prediction error} \asymp \phi_n + \Delta_n$$

(up to $\log n$-factors) with

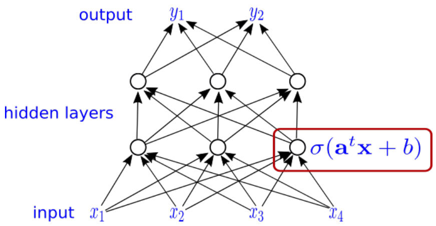
$$\Delta_n := E\Big[\frac{1}{n}\sum_{i=1}^n (Y_i - \widehat{f}_n(\mathbf{X}_i))^2 - \inf_{f\in\mathcal{F}(L,\mathbf{p},s)} \frac{1}{n}\sum_{i=1}^n (Y_i - f(\mathbf{X}_i))^2\Big]$$

and

$$\phi_n := \max_{i=0,\dots,q} n^{-\frac{2\beta_i^*}{2\beta_i^*+t_i}}.$$

Statistical theory for deep neural networks

Lecture 3 & 4



Johannes Schmidt-Hieber

advantages of additional layers

- localization
- approximation of polynomials with deep networks
- Kolmogorov-Arnold representation theorem
- advantages of deep ReLU networks

improved representation theorems

- Kolmogorov-Arnold approximation theorem shows that every continuous function can be represented by a specific two-layer network
- very different structure if compared with the universal approximation theorem for shallow networks
- indicates that additional layers can lead to new features of network functions

Theorem (Braun '09): Fix $d \geq 2$. There are real numbers a, b_p, c_q and a continuous and monotone function $\psi : \mathbb{R} \to \mathbb{R}$, such that for any continuous function $f : [0,1]^d \to \mathbb{R}$, there exists a continuous function $g : \mathbb{R} \to \mathbb{R}$ with

$$f(x_1, \ldots, x_d) = \sum_{q=0}^{2d} g\Big(\sum_{p=1}^{d} b_p \psi(x_p + qa) + c_q\Big).$$

remarks

$$f(x_1, \ldots, x_d) = \sum_{q=0}^{2d} g\Big(\sum_{p=1}^{d} b_p \psi(x_p + qa) + c_q\Big).$$

- one inner function ψ and one outer function g
- inner function is independent of f
- q-dependence in the first layer comes through the shifts qa.
- right hand side can be realized by a network with two hidden layers, architecture $\mathbf{p} = (d, d, 2d + 1, 1)$, and ψ being the activation function in the first layer.

link to pre-training

- inner function in the Kolmogorov-Arnold representation theorem is independent of the represented function f
- in deep learning it has been observed that the first layers build function systems which can be used for other classification problems
- exploited in pre-training where a trained deep network from a possibly completely different classification problem is taken and only the last layer is learned by the new dataset
- fact that pre-training works shows that deep networks build generic function systems in the first layers.

deep ReLU networks

we discuss several advantages of deep ReLU networks

- representation of identity
- growth of number of linear pieces
- approximation by ReLU networks with small parameters

deep ReLU networks can learn skip connections

$$\sigma(x) = \max(x, 0)$$

- projection property

$$\sigma \circ \sigma = \sigma$$

- $\rightsquigarrow$ pass a signal without change through several layers in the network
- $\rightsquigarrow$ network synchronization by adding hidden layers
- related to skip connections and ResNets
- for other activation functions it is much harder to approximate the identity

number of linear pieces of deep ReLU networks

- deep ReLU networks are piecewise linear functions of the input
- adding layers $\rightsquigarrow$ highly oscillating functions with few parameters
- consider ReLU network with two hidden layers and width vector $(1, m, 1, 1)$ of the form

$$\Big(\sum_{j=1}^{m} c_j (w_j x + v_j)_+\Big)_+$$

- $\rightsquigarrow$ number of added pieces by outer ReLU is proportional to number of zero crossings of inner function
- any ReLU network with width vector $(1, p_1, \ldots, p_L, 1)$ has at most

$$\Big(\frac{3}{2}\Big)^L \prod_{j=1}^{L} (p_j + 1)$$

pieces

example of a highly oscillating function

Functions:

- let $T : [0,1] \to [1]$,

$$T(x) := (2x) \wedge (1-2x) = (2x)_+ - (4x-2)_+$$

- can be realized by shallow network with two units
- $R^k : [0,1] \to [0,1]$,

$$R^k := \underbrace{T \circ T \circ \dots T}_{k \text{ times}}$$

network representation

$$\begin{tikzcd}
 & T^k(x) & \\
(2T^{k-1}(x))_+ \arrow[ur, "1"] & & (4T^{k-1}(x)-2)_+ \arrow[ul, "-1"'] \\
 & T^{k-1}(x) \arrow[ul, "2"] \arrow[ur, "4"'] & \\
\vdots \arrow[ur] & & \vdots \arrow[ul] \\
\vdots & & \vdots \\
 & T^2(x) \arrow[ul] \arrow[ur] & \\
(2T(x))_+ \arrow[ur, "1"] & & (4T(x)-2)_+ \arrow[ul, "-1"'] \\
 & T(x) \arrow[ul, "2"] \arrow[ur, "4"'] & \\
(2x)_+ \arrow[ur, "1"] & & (4x-2)_+ \arrow[ul, "-1"'] \\
 & x \arrow[ul, "2"] \arrow[ur, "4"'] &
\end{tikzcd}$$

$$\begin{tikzcd}
 & T^k(x) & \\
(2T^{k-1}(x))_+ \arrow[ur, "1"] & & (4T^{k-1}(x)-2)_+ \arrow[ul, "-1"'] \\
(2T^{k-2}(x))_+ \arrow[u, "2"] \arrow[urr, "4"] & & (4T^{k-2}(x)-2)_+ \arrow[u, "-4"'] \arrow[ull, "-2"] \\
\vdots \arrow[u] \arrow[urr] & & \vdots \arrow[u] \arrow[ull] \\
\vdots & & \vdots \\
(2T^2(x))_+ \arrow[u] \arrow[urr] & & (4T^2(x)-2)_+ \arrow[u] \arrow[ull] \\
(2T(x))_+ \arrow[u, "2"] \arrow[urr, "4"] & & (4T(x)-2)_+ \arrow[u, "-4"'] \arrow[ull, "-2"] \\
(2x)_+ \arrow[u, "2"] \arrow[urr, "4"] & & (4x-2)_+ \arrow[u, "-4"'] \arrow[ull, "-2"] \\
 & x \arrow[ul, "2"] \arrow[ur, "4"'] &
\end{tikzcd}$$

multiplication

how can we (approximately) multiply two inputs with a network?

- crucial problem for approximation theory
- for deep networks this can be reduced to approximation of square function $x \mapsto x^2$ via

$$xy = \left(\frac{x+y}{2}\right)^2 - \left(\frac{x-y}{2}\right)^2$$

- has a surprising answer for ReLU networks

approach from Lecture 2

network approximation of the function $x \mapsto x^2$ is very important!

- for twice differentiable activation function, we used

$$\frac{\sigma(t+2xh) - 2\sigma(t+xh) + \sigma(xh)}{h^2\sigma''(t)} \to x^2 \text{ for } h \to 0$$

- $\rightsquigarrow$ network parameters become large
- for deep ReLU networks we use a different construction

ReLU approximation of the square function

Functions:

- let $T^k : [0, 2^{2-2k}] \to [0, 2^{-2k}]$,
$$T^k(x) := (x/2) \wedge (2^{1-2k} - x/2) = (x/2)_+ - (x - 2^{1-2k})_+$$
- $R^k : [0,1] \to [0, 2^{-2k}]$,
$$R^k := T^k \circ T^{k-1} \circ \dots T^1.$$

Lemma (Telgarsky '16, Yarotski '18, SH '17):

$$\left| x(1-x) - \sum_{k=1}^{m} R^k(x) \right| \leq 2^{-m}.$$

rewriting approximation as network

$$\left| x(1-x) - \sum_{k=1}^{m} R^k(x) \right| \leq 2^{-m}.$$

deep ReLU approximation:

- m hidden layers
- $O(m)$ network parameters
- bounded parameters
- approximation 2^{-m}

shallow ReLU network

- for $x(1-x)$ a shallow ReLU network needs at least $O(2^{m/2})$ parameters to achieve approximation error 2^{-m}

multiplication with deep ReLU networks

Lemma: There exists a network Mult_m with $m+4$ hidden layers, width vector $(2, 6, 6, \ldots, 6, 1)$ and all network parameters bounded by one, such that

$$\left|\mathrm{Mult}_m(x,y) - xy\right| \leq 2^{-m}, \quad \text{for all } x, y \in [0,1].$$

Proof:

- use polarization identity

$$xy = \left(\frac{x+y}{2}\right)^2 - \left(\frac{x-y}{2}\right)^2$$

- separation of positive and negative part
- compute $(x+y)/2$ and $(x-y)/2$ in first layer (non-negativity!)
- square network has to be incorporated twice (inefficient)

a step in the proof

$$\begin{tikzcd}
 & \sum_{k=1}^{m+1} R^k(u) + h(u) & \\
T_+(R^m(u)) \arrow[ur] & \sum_{k=1}^{m} R^k(u) + h(u) \arrow[u] & T_-^{m+1}(R^m(u)) \arrow[ul] \\
\vdots \arrow[u] \arrow[ur] \arrow[urr] & \vdots \arrow[u] & \vdots \arrow[u] \arrow[ul] \arrow[ull] \\
\vdots & \vdots & \vdots \\
T_+(R^1(u)) \arrow[u] \arrow[ur] \arrow[urr] & R^1(u) + h(u) \arrow[u] & T_-^2(R^1(u)) \arrow[u] \arrow[ul] \arrow[ull] \\
T_+(u) \arrow[u] \arrow[ur] \arrow[urr] & h(u) \arrow[u] & T_-^1(u) \arrow[u] \arrow[ul] \arrow[ull]
\end{tikzcd}$$

localization and approximation

- we have seen that with two hidden layers we can localize
- how can this be done for ReLU networks?
- goes back to Yarotsky '18
- define $\mathbf{D}(M)$ as all grid points on the grid

$$\left\{(\ell_j/M)_{j=1,\dots,r} : \boldsymbol{\ell} = (\ell_1, \dots, \ell_r) \in \{0, 1, \dots, M\}^r\right\}$$

- partition of unity on unit cube

$$\sum_{\mathbf{x}_{\boldsymbol{\ell}} \in \mathbf{D}(M)} \underbrace{\prod_{j=1}^{r} (1 - M|x_j - x_j^{\boldsymbol{\ell}}|)_+}_{\text{localized functions}} = \prod_{j=1}^{r} \sum_{\ell=0}^{M} (1 - M|x_j - \ell/M|)_+ = 1,$$

local Taylor approximation

- on each localized bit ($\mathbf{a} \in \mathbf{D}(M)$) do a Taylor approximation

$$f(\mathbf{x}) \approx P_{\mathbf{a}}^{\beta} f(\mathbf{x}) := \sum_{0 \le |\boldsymbol{\alpha}| < \beta} (\partial^{\boldsymbol{\alpha}} f)(\mathbf{a}) \frac{(\mathbf{x} - \mathbf{a})^{\boldsymbol{\alpha}}}{\boldsymbol{\alpha}!} = \sum_{0 \le |\boldsymbol{\gamma}| < \beta} \mathbf{x}^{\boldsymbol{\gamma}} c_{\boldsymbol{\gamma}}$$

- this can be approximately realized by a deep ReLU network
- many technicalities occur (see the article SH '17)

approximation rate

Theorem: For any β-smooth function $f : [0,1]^r \to \mathbb{R}$ and any integers $m, N \geq 1$, there exists a ReLU network with

- depth $L \asymp m$
- width in each layer bounded by $\lesssim N$
- number of non-zero network parameters $s \lesssim Nm$

such that

$$\|\widetilde{f} - f\|_{L^\infty([0,1]^r)} \lesssim \underbrace{N2^{-m}}_{\text{small for deep networks}} + \underbrace{N^{-\frac{\beta}{r}}}_{\text{approx. rate}} .$$

remarks

$$\|\widetilde{f} - f\|_{L^\infty([0,1]^r)} \lesssim \underbrace{N2^{-m}}_{\text{small for deep networks}} + \underbrace{N^{-\frac{\beta}{r}}}_{\text{approx. rate}} .$$

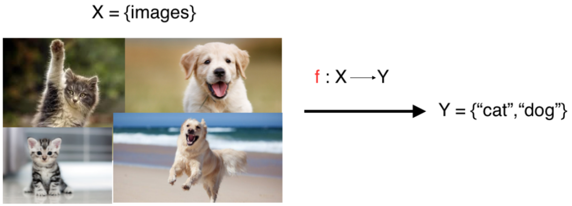
- for deep networks first term is of smaller order
- second term becomes suboptimal for large depth
- trade-off
- sparse networks

risk bounds for deep ReLU networks

Framework:

- we now study a statistical problem
- requires that we first need to specify a statistical model
- we study nonparametric regression

mathematical problem



The data are used to fit a network, i.e. estimate the network parameters.

How fast does the estimated network converge to the truth f as sample size increases?

statistical analysis

- we observe n i.i.d. copies $(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)$,

$$Y_i = f(\mathbf{X}_i) + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0,1)$$

  - $\mathbf{X}_i \in \mathbb{R}^d$, $Y_i \in \mathbb{R}$,
  - goal is to reconstruct the function $f : \mathbb{R}^d \to \mathbb{R}$
- has been studied extensively
  (kernel smoothing, wavelets, splines, . . .)

the estimator

- choose network architecture $(L, \mathbf{p})$ and sparsity s
- denote by $\mathcal{F}(L, \mathbf{p}, s)$ the class of all networks with
    - architecture $(L, \mathbf{p})$
    - number of active (e.g. non-zero) parameters is s
- our theory applies to any estimator $\widehat{f}_n$ taking values in $\mathcal{F}(L, \mathbf{p}, s)$
- prediction error

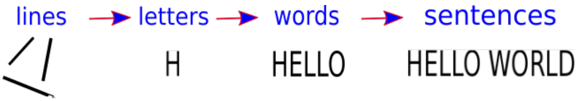
$$R(\widehat{f}_n, f) := E_f\big[\big(\widehat{f}_n(\mathbf{X}) - f(\mathbf{X})\big)^2\big],$$

  with $\mathbf{X} \overset{\mathcal{D}}{=} \mathbf{X}_1$ being independent of the sample
- study the dependence of n on $R(\widehat{f}_n, f)$

function class

- classical idea: assume that regression function is β-smooth
- optimal nonparametric estimation rate is $n^{-2\beta/(2\beta+d)}$
- suffers from curse of dimensionality
- to understand deep learning this setting is therefore useless
- $\rightsquigarrow$ make a good structural assumption on f

hierarchical structure



- Important: Only few objects are combined on deeper abstraction level
    - few letters in one word
    - few words in one sentence

function class

- We assume that

$$f = g_q \circ \ldots \circ g_0$$

  with
  - $g_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_{i+1}}$.
  - each of the d_{i+1} components of g_i is β_i-smooth and depends only on t_i variables
  - t_i can be much smaller than d_i
  - effective smoothness

$$\beta_i^* := \beta_i \prod_{\ell=i+1}^{q} (\beta_\ell \wedge 1).$$

  - we show that **the rate depends on the pairs**

$$(t_i, \beta_i^*), \quad i = 0, \ldots, q.$$

  - similar conditions have been proposed by Horowitz & Mammen (2007), Kohler & Kryzak (2017), Bauer & Kohler (2017), Kohler & Langer (2018)

example

$$f_0(x_1, x_2, x_3) = g_{11}\Big(g_{01}(x_3), g_{02}(x_2)\Big)$$

- $f_0 = g_1 \circ g_0$
- $d_0 = 3,\ t_0 = 1,\ d_1 = t_1 = 2,\ d_2 = 1$

main result

Theorem: If

(i) depth $\asymp \log n$

(ii) width $\geq$ network sparsity $\asymp \max_{i=0,\ldots,q} n^{\frac{t_i}{2\beta_i^*+t_i}} \log n$

Then, for any network reconstruction method $\widehat{f}_n$,

$$\text{prediction error} \asymp \phi_n + \Delta_n$$

(up to $\log n$-factors) with

$$\Delta_n := E\Big[\frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{f}_n(\mathbf{X}_i))^2 - \inf_{f\in\mathcal{F}(L,\mathbf{p},s)} \frac{1}{n}\sum_{i=1}^{n}(Y_i - f(\mathbf{X}_i))^2\Big]$$

and

$$\phi_n := \max_{i=0,\ldots,q} n^{-\frac{2\beta_i^*}{2\beta_i^*+t_i}}.$$

main result

Theorem: If

(i) depth $\asymp \log n$

(ii) width $\geq$ network sparsity $\asymp \max_{i=0,\ldots,q} n^{\frac{t_i}{2\beta_i^*+t_i}} \log n$

Then, for any network reconstruction method $\widehat{f}_n$,

$$\text{prediction error} \asymp \phi_n + \Delta_n$$

(up to $\log n$-factors) with

$$\Delta_n := E\Big[\frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{f}_n(\mathbf{X}_i))^2 - \inf_{f\in\mathcal{F}(L,\mathbf{p},s)} \frac{1}{n}\sum_{i=1}^{n}(Y_i - f(\mathbf{X}_i))^2\Big]$$

and

$$\phi_n := \max_{i=0,\ldots,q} n^{-\frac{2\beta_i^*}{2\beta_i^*+t_i}}.$$

consequences

- empirical risk minimizer is optimal in this class
- problem is high-dimensional (no upper bound on the width)
- network sparsity induces regularization
- the assumption that depth $\asymp \log n$ appears naturally
- in particular the depth scales with the sample size

important for statistical performance is not the size of the network but the amount of regularization

consequences (ctd.)

paradox:

- good rate for all smoothness indices
- existing piecewise linear methods only give good rates up to smoothness two
- Here the non-linearity of the function class helps

$\rightsquigarrow$ **non-linearity is essential!!!**

additive models

- functions are of the form

$$f(x_1, \dots, x_d) = f_1(x_1) + \dots + f_d(x_d)$$

- f_i are β-smooth
- $f = g_1 \circ g_0$ with

$$g_0(\mathbf{x}) = (f_1(x_1), \dots, f_d(x_d))^\top \text{ and } \; g_1(\mathbf{y}) = \sum_{j=1}^{d} y_j$$

- $\rightsquigarrow d_0 = d, \; t_0 = 1, \; d_1 = t_1 = d, \; d_2 = 1$

rate achieved by a neural network

$$R(\widehat{f}_n, f_0) \lesssim n^{-\frac{2\beta}{2\beta+1}} \log^3 n + \Delta(\widehat{f}_n, f_0).$$

on the proof

- oracle inequality (roughly)

$$R(\widehat{f}, f) \lesssim \inf_{f^* \in \mathcal{F}(L, \mathbf{p}, s)} \left\| f^* - f \right\|_\infty^2 + \frac{\log \mathcal{N}_n}{n}.$$

  - $\log \mathcal{N}_n$ denotes the covering entropy
  - shows the trade-off between approximation and model size

- for networks we obtain a bound of the type

$$\log \mathcal{N}_n \lesssim sL \log(n)$$

- $\rightsquigarrow$ trade-off between approximation and network sparsity

lower bounds on the network sparsity

the convergence theorem implies a deterministic lower bound on the network sparsity required to approximate β-smooth functions on $[0,1]^d$

Result:

- if for $\varepsilon > 0$,

$$s \lesssim \frac{\varepsilon^{-d/\beta}}{L \log(1/\varepsilon)}$$

  then

$$\sup_{f_0 \text{ is } \beta-\text{Hölder}} \ \inf_{f \text{ a } s-\text{sparse network}} \|f - f_0\|_\infty \geq \varepsilon.$$

- has been proved via a different technique in Bölcskei et al. ’17

sparsely connected networks

Network sparsity is crucial in the proof but classical deep learning produces dense networks. Recently many new methods have been proposed generating sparsely connected networks.

- sparsifying as post-processing step $\rightsquigarrow$ compression
- starting with sparse network topology
- evolutionary methods inspired by human brain

other statistical results

- piecewise smooth functions, Imaizumi and Fukumizu ’18
- binary classification with hinge loss, Kim, Ohn, Kim ’18
- causal models, Farell, Liang, Misra ’18
- deep Q-learning, Fan et al. ’20
- and others ...

suboptimality of wavelet estimators

- $f(\mathbf{x}) = h(x_1 + \ldots + x_d)$
- for some α-smooth function h
- Rate for DNNs $\lesssim n^{-\alpha/(2\alpha+1)}$ (up to logarithmic factors)
- Rate for best wavelet thresholding estimator $\gtrsim n^{-\alpha/(2\alpha+d)}$
- Reason: Low-dimensional structure does not affect the decay of the wavelet coefficients

MARS

- consider products of ramp functions

$$h_{I,\boldsymbol{t}}(x_1, \ldots, x_d) = \prod_{j \in I} \big(\pm (x_j - t_j) \big)_+$$

- piecewise constant in each component
- MARS (multivariate adaptive regression splines) fits linear combinations of such functions to data
- greedy algorithm
- has depth and width type parameters

Comparison with MARS

- how does MARS compare to ReLU networks?
- functions that can be represented by s parameters with respect to the MARS function system can be represented by $s \log(1/\varepsilon)$-sparse DNNs up to sup-norm error ε

Comparison with MARS (ctd.)

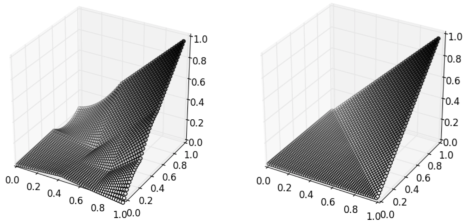


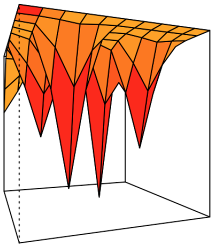
Figure: Reconstruction using MARS (left) and networks (right)

- the opposite is not true, one counterexample is

$$f(x_1, x_2) = (x_1 + x_2 - 1)_+$$

- we need $\gtrsim \varepsilon^{-1/2}$ many parameters to get ε-close with MARS functions
- $\rightsquigarrow$ conclusion: DNNs work better for correlated design

energy landscape



Definition:

- data $(X, Y) \in (\mathcal{X}, \mathcal{Y})$
- class of functions $F_\theta : \mathcal{X} \to \mathcal{Y}$
- loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$

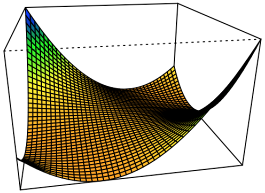
the energy landscape/loss surface is the function

$$\theta \mapsto L\big(Y, F_\theta(X)\big).$$

critical points of the energy landscape

- local/global minima
- saddle points
- (bad) saddle points (Hessian vanishes)

linear activation function



- fit a linear regression line $f(y) = abx$ to data $(X_i, Y_i)_{i=1,\dots,n}$
- a, b parameters
- energy landscape for squared loss

$$(a,b) \mapsto \sum_{i=1}^{n} \big(Y_i - abX_i\big)^2.$$

- saddle point for $a = b = 0$
- global minimum whenever $ab =$ least squares solution

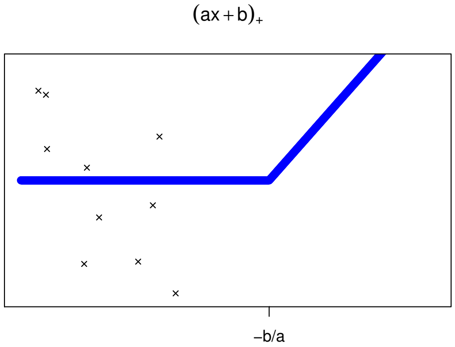
extensions

- same setting as before but now we consider $f(x) = abcx$
- $a = b = c = 0$ is a bad saddle point

Kawaguchi '16:

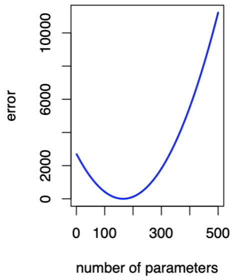
- $f(\mathbf{x}) = W_L W_{L-1} \dots W_0 \mathbf{x}$
- every local minimum is a global minimum
- saddle points exist (if $L > 1$, there exist bad saddle points)
- proof is based on studying local perturbations

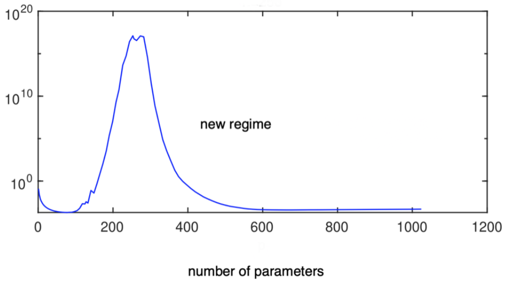
ReLU activation function



- possible many local minima that are not global minima
- happens in practice
- very dependent on initialization

double descent and implicit regularization





overparametrization generalizes well $\rightsquigarrow$ implicit regularization

overfitting

- training error $= 0$ implies that $\Delta_n = 0$
- Δ_n does not fully characterize the statistical properties anymore
- because of implicit regularization, SGD will pick interpolant with good statistical properties

can implicit regularization avoid sparsity?

we conjecture the answer is **no** for the regression problem!

interpolation properties

- consider continuous activation function that is not a polynomial
- given data $(\mathbf{X}_k, Y_k) \in \mathbb{R}^d \times \mathbb{R}$ with distinct design vectors $\mathbf{X}_k$
- shallow networks: one can perfectly interpolate n data points with n units in the hidden layer
- related to the universal approximation theorem (therefore same condition appears)

theory for vanishing training error

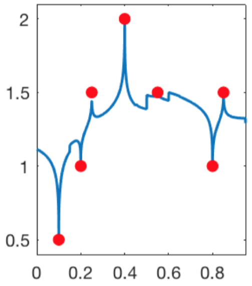
smooth activation function

- Du et al. '18 consider highly over-parametrized setting
- number of units in each layer has to be of some (unspecified ?) polynomial order in the sample size
- setup is regression with least-squares loss
- show that gradient descent with randomly initialization converges to zero training error

ReLU networks

- Allen-Zhu et al. '18 shows a similar result
- one assumptions is that the network width scales at least with the 30-th power of the sample size

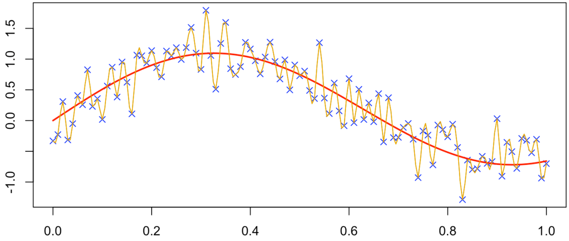
does data interpolation contradict statistical optimality?



Source: Belkin, Rakhlin, Tsybakov, 2018

in principle it is possible to interpolate and to denoise simultaneously

more details



we can show that for a simplified model and properly chosen learning rate, SGD converges to natural cubic spline interpolant

$\rightsquigarrow$ inconsistent estimator

main idea

A shallow network ($L = 1$) can be written as

$$x \mapsto \sum_{j=1}^{m} a_j (b_j x - c_j)_+, \quad a_j, b_j, c_j \in \mathbb{R}.$$

Taylor expansion in one dimension

$$g(x) = g(0) + x g'(0) + \int g''(u)(x-u)_+ du$$

If, say $g(0) = g'(0) = 0$, we have that approximately

$$g(x) \approx \frac{1}{m} \sum_{j=1}^{m} g''\left(\frac{j}{m}\right)\left(x - \frac{j}{m}\right)_+.$$

denoising vs. interpolation

- implicit regularization is not sufficient to do denoising
- it still works in practice because standard datasets have a lot of structure in common (classification with few misclassified data points)

All statements that start with "In deep learning . . ." are wrong, what matters is the structure of the data. To describe for which data structures such claims are true is a major challenge for research in statistics.

outlook

deep networks are an exciting field with many open problems

- new phenomena, . . .
- network types: CNNs, RNNs, autoencoders, . . .
- generative adversarial networks (GANs)

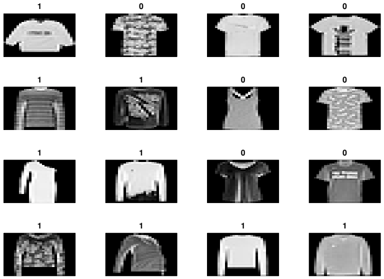
Thank you for your attention!

open positions at the University of Twente

- we have constantly openings for PhD and postdoc positions
- mainly theoretical topics (theory for machine learning methods, privacy, time series, . . .)
- 4-year PhD positions
- fully paid (gross salary ranging from 2400 Euro (1st year) to 3000 Euro (4th year))

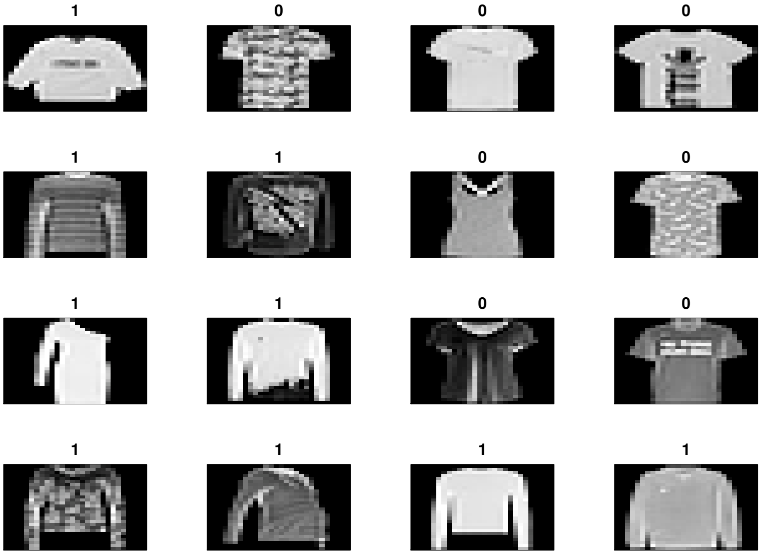
Statistical theory for deep neural networks

Tutorial



Johannes Schmidt-Hieber

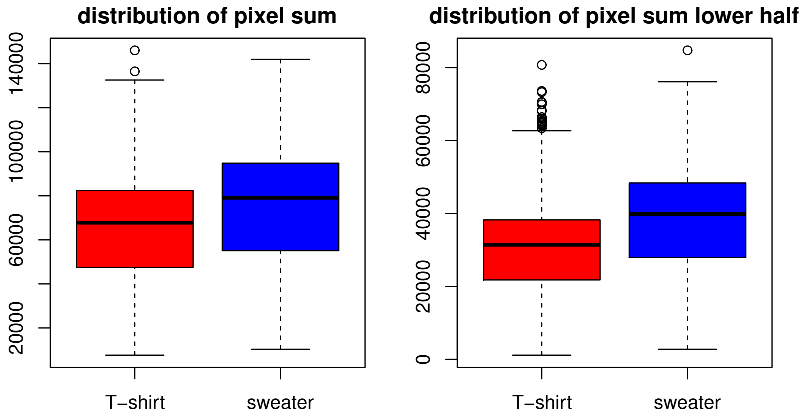
t-shirt or sweater



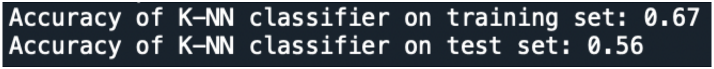
data

- 12000 28×28 pixel images
- k-th image is stored in k-th row
- t-shirt=0, sweater=1

simple classifier

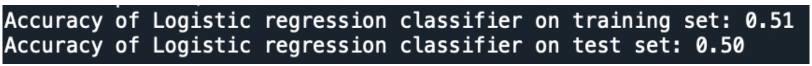


10-nearest neighbors



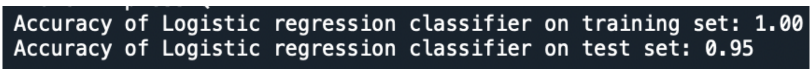
```
Accuracy of K-NN classifier on training set: 0.67
Accuracy of K-NN classifier on test set: 0.56
```

logistic regression with sum of pixels



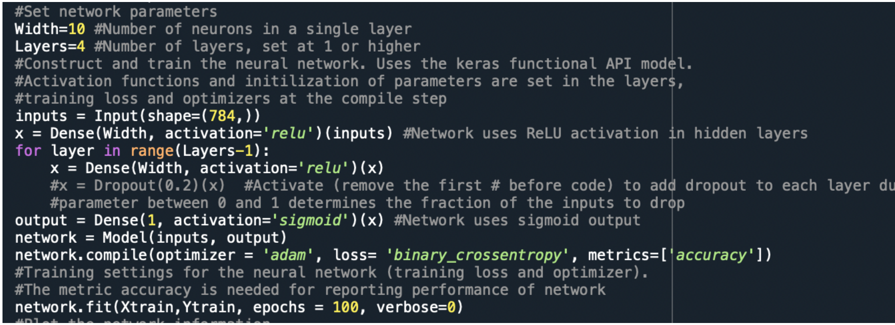
```
Accuracy of Logistic regression classifier on training set: 0.51
Accuracy of Logistic regression classifier on test set: 0.50
```

logistic regression with all pixels



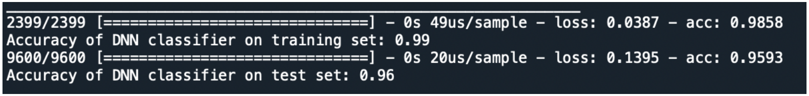
```
Accuracy of Logistic regression classifier on training set: 1.00
Accuracy of Logistic regression classifier on test set: 0.95
```

neural network



```
#Set network parameters
Width=10 #Number of neurons in a single layer
Layers=4 #Number of layers, set at 1 or higher
#Construct and train the neural network. Uses the keras functional API model.
#Activation functions and initilization of parameters are set in the layers,
#training loss and optimizers at the compile step
inputs = Input(shape=(784,))
x = Dense(Width, activation='relu')(inputs) #Network uses ReLU activation in hidden layers
for layer in range(Layers-1):
    x = Dense(Width, activation='relu')(x)
    #x = Dropout(0.2)(x)  #Activate (remove the first # before code) to add dropout to each layer du
    #parameter between 0 and 1 determines the fraction of the inputs to drop
output = Dense(1, activation='sigmoid')(x) #Network uses sigmoid output
network = Model(inputs, output)
network.compile(optimizer = 'adam', loss= 'binary_crossentropy', metrics=['accuracy'])
#Training settings for the neural network (training loss and optimizer).
#The metric accuracy is needed for reporting performance of network
network.fit(Xtrain,Ytrain, epochs = 100, verbose=0)
```

neural network



```
2399/2399 [==============================] - 0s 49us/sample - loss: 0.0387 - acc: 0.9858
Accuracy of DNN classifier on training set: 0.99
9600/9600 [==============================] - 0s 20us/sample - loss: 0.1395 - acc: 0.9593
Accuracy of DNN classifier on test set: 0.96
```

Questions:

1. How does the test error behave for wide and/or deep networks?
2. How does the test error behave as you change the learning rate?
3. For the best network architecture, how does the test error change if you do (not) use batch normalization and/or dropout ?
4. Try different network initialization strategies. See `https://keras.io/api/layers/initializers/` for details.
5. make a histogram of the learned network weights.