Deep Learning meets statistics: Improving neural networks with statistical theory

TECHNISCHE UNIVERSITÄT DARMSTADT

Sophie Langer

Algorithms & Computationally Intensive Inference seminar, Warwick Statistics | June 2021

Motivation

Motivation

Successful, but why?

Motivation

Successful applications

...**but** lack of mathematical understanding

Aim

Understanding theoretical properties of deep neural networks

Research field

Deep neural networks in nonparametric regression
→ Least squares regression estimates using deep neural networks

Motivation

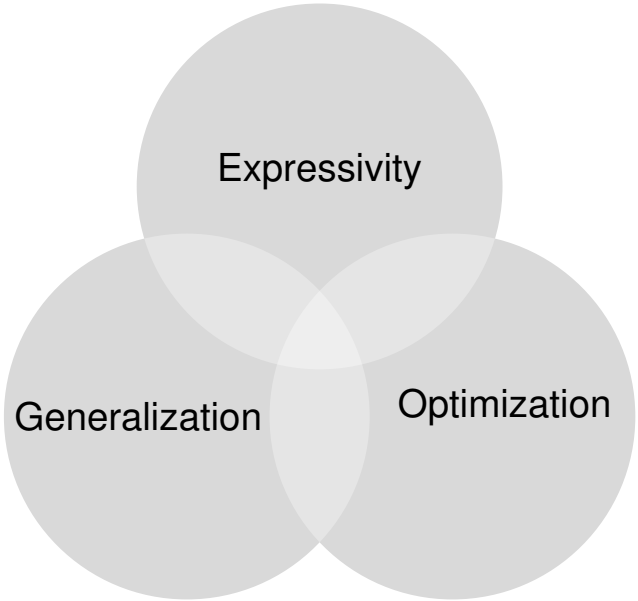


Figure: Fundamental research topics of Deep Learning

- Problem: Describing the procedure of DNNs by hand is a highly complex task
- Theoretical results focus either on approximation, generalization or optimization of DNNs
- But: Results did not take into account all three aspects of Deep Learning, simultaneously

$\hookrightarrow$ Cannot be used to improve estimators in practice

Should it not be the aim of statistical theory to not only understand but also improve estimators in practice?

A promising result

- Braun et al. (2021): Analysis of shallow networks learned by gradient descent

$\hookrightarrow$ Good rate of convergence results

$\hookrightarrow$ Improved performance on simulated data

$\hookrightarrow$ Can be considered as the *smoking gun* of deep learning theory, as it shows for the first time that statistical theory can be used to improve estimators in practice

Nonparametric regression

Aim

Study the **statistical** performance of neural networks
$\rightsquigarrow$ Analyze neural networks in the context of nonparametric regression

Prediction problem

Given $\mathbb{R}^d \times \mathbb{R}$-valued random vector $(\mathbf{X}, Y)$

Try to predict Y by $f^*(\mathbf{X})$ from some $f^* : \mathbb{R}^d \to \mathbb{R}$

$\rightsquigarrow$ Find $f^* : \mathbb{R}^d \to \mathbb{R}$ such that

$$\mathbf{E}\{|f^*(\mathbf{X}) - Y|^2\} = \min_{f:\mathbb{R}^d \to \mathbb{R}} \mathbf{E}\{|f(\mathbf{X}) - Y|^2\}.$$

It can be shown, that $f^*(\mathbf{x}) = m(\mathbf{x}) := \mathbf{E}\{Y|\mathbf{X} = \mathbf{x}\}$.
$\rightsquigarrow$ $m : \mathbb{R}^d \to \mathbb{R}$ is the so-called **regression function**

Nonparametric regression

- **Problem:** Distribution of $(\mathbf{X}, Y)$ is unknown
- Given a set of n of i.i.d. observations $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ of $(\mathbf{X}, Y)$
- Reconstruct m by an estimator

$$m_n(\cdot) = m_n(\cdot, \mathcal{D}_n) : \mathbb{R}^d \to \mathbb{R}$$

such that

$$\int |m_n(\mathbf{x}) - m(\mathbf{x})|^2 \mathbf{P}_{\mathbf{X}}(d\mathbf{x})$$

is *small*.

Barron's result

Define

$$\mathcal{F}_n = \left\{ \sum_{k=1}^{\lceil \sqrt{n} \rceil} \alpha_k \cdot \sigma(\beta_k \cdot \mathbf{x} + \gamma_k) : \alpha_k, \gamma_k \in \mathbb{R}, \beta_k \in \mathbb{R}^d, \sum_{k=0}^{K_n} |\alpha_k| \leqslant L_n \right\},$$

where $\sigma(u) = 1/(1 + \exp(-u))$ $(u \in \mathbb{R})$ and let

$$m_n(\cdot) = \arg\min_{f \in \mathcal{F}_n} \frac{1}{n} \sum_{i=1}^{n} |Y_i - f(\mathbf{X}_i)|^2$$

be the corresponding least squares estimator.

Barron's result

Define

$$\mathcal{F}_n = \left\{ \sum_{k=1}^{\lceil \sqrt{n} \rceil} \alpha_k \cdot \sigma(\beta_k \cdot \mathbf{x} + \gamma_k) : \alpha_k, \gamma_k \in \mathbb{R}, \beta_k \in \mathbb{R}^d, \sum_{k=0}^{K_n} |\alpha_k| \leqslant L_n \right\},$$

where $\sigma(u) = 1/(1 + \exp(-u))$ $(u \in \mathbb{R})$ and let

$$m_n(\cdot) = \arg\min_{f \in \mathcal{F}_n} \frac{1}{n} \sum_{i=1}^{n} |Y_i - f(\mathbf{X}_i)|^2$$

be the corresponding least squares estimator. Then

$$\mathbf{E} \int |m_n(\mathbf{x}) - m(\mathbf{x})|^2 \, \mathbf{P}_{\mathbf{X}}(d\mathbf{x}) \leqslant c_1 \cdot (\log n)^5 \cdot \frac{1}{\sqrt{n}}$$

holds whenever the Fourier transform of the regression function has a finite first moment.

An estimator learned by gradient descent

We study the rate of convergence of a neural network estimators learned by gradient descent

An estimator learned by gradient descent

We study the rate of convergence of a neural network estimators learned by gradient descent
We need the following definitions:

$$\sigma(u) = 1/(1 + \exp(-u)) \quad (u \in \mathbb{R}),$$

An estimator learned by gradient descent

We study the rate of convergence of a neural network estimators learned by gradient descent
We need the following definitions:

$$\sigma(u) = 1/(1 + \exp(-u)) \quad (u \in \mathbb{R}),$$

$$f_{net,\mathbf{w}}(\mathbf{x}) = \alpha_0 + \sum_{j=1}^{K_n} \alpha_j \cdot \sigma(\beta_j^T \cdot \mathbf{x} + \gamma_j),$$

where

$$\mathbf{w} = (\alpha_0, \alpha_1, \ldots, \alpha_{K_n}, \beta_1, \ldots, \beta_{K_n}, \gamma_1, \ldots, \gamma_{K_n}),$$

An estimator learned by gradient descent

We study the rate of convergence of a neural network estimators learned by gradient descent
We need the following definitions:

$$\sigma(u) = 1/(1 + \exp(-u)) \quad (u \in \mathbb{R}),$$

$$f_{net,\mathbf{w}}(\mathbf{x}) = \alpha_0 + \sum_{j=1}^{K_n} \alpha_j \cdot \sigma(\beta_j^T \cdot \mathbf{x} + \gamma_j),$$

where

$$\mathbf{w} = (\alpha_0, \alpha_1, \ldots, \alpha_{K_n}, \beta_1, \ldots, \beta_{K_n}, \gamma_1, \ldots, \gamma_{K_n}),$$

and

$$F(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} |Y_i - f_{net,\mathbf{w}}(\mathbf{X}_i)|^2 + \frac{c_2}{K_n} \cdot \sum_{k=0}^{K_n} \alpha_k^2.$$

An estimator learned by gradient descent

- Initial weights:

$$\mathbf{w}(0) = (\alpha_0(0), \ldots, \alpha_{K_n}(0), \beta_1(0), \ldots, \beta_{K_n}(0), \gamma_1(0), \ldots, \gamma_{K_n}(0))$$

such that

$$\alpha_0(0) = \alpha_1(0) = \cdots = \alpha_{K_n}(0) = 0$$

An estimator learned by gradient descent

- Initial weights:

$$\mathbf{w}(0) = (\alpha_0(0), \ldots, \alpha_{K_n}(0), \beta_1(0), \ldots, \beta_{K_n}(0), \gamma_1(0), \ldots, \gamma_{K_n}(0))$$

such that

$$\alpha_0(0) = \alpha_1(0) = \cdots = \alpha_{K_n}(0) = 0$$

and $\beta_1(0), \ldots, \beta_{K_n}(0), \gamma_1(0), \ldots, \gamma_{K_n}(0)$ independently randomly chosen such that

- $\beta_k(0)$ are uniformly distributed on a sphere with radius B_n
- $\gamma_j(0)$ are uniformly distributed on $[-B_n \cdot \sqrt{d}, B_n \cdot \sqrt{d}]$.

An estimator learned by gradient descent

- Initial weights:

$$\mathbf{w}(0) = (\alpha_0(0), \ldots, \alpha_{K_n}(0), \beta_1(0), \ldots, \beta_{K_n}(0), \gamma_1(0), \ldots, \gamma_{K_n}(0))$$

  such that

$$\alpha_0(0) = \alpha_1(0) = \cdots = \alpha_{K_n}(0) = 0$$

  and $\beta_1(0), \ldots, \beta_{K_n}(0), \gamma_1(0), \ldots, \gamma_{K_n}(0)$ independently randomly chosen such that

  - $\beta_k(0)$ are uniformly distributed on a sphere with radius B_n
  - $\gamma_j(0)$ are uniformly distributed on $[-B_n \cdot \sqrt{d}, B_n \cdot \sqrt{d}]$.

- t_n gradient descent steps:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \lambda_n \cdot \nabla_{\mathbf{w}} F(\mathbf{w}(t)) \quad (t = 0, \ldots, t_n - 1).$$

An estimator learned by gradient descent

- The estimator:

$$\tilde{m}_n(\cdot) = f_{net,\mathbf{w}(t_n)}(\cdot) \quad \text{and} \quad m_n(\mathbf{x}) = T_{c_1 \cdot \log n} \tilde{m}_n(\mathbf{x})$$

where $T_L z = \max\{\min\{z, L\}, -L\}$ for $z \in \mathbb{R}$ and $L \geqslant 0$.

An estimator learned by gradient descent

- The estimator:

$$\tilde{m}_n(\cdot) = f_{net,\mathbf{w}(t_n)}(\cdot) \quad \text{and} \quad m_n(\mathbf{x}) = T_{c_1 \cdot \log n} \tilde{m}_n(\mathbf{x})$$

  where $T_L z = \max\{\min\{z, L\}, -L\}$ for $z \in \mathbb{R}$ and $L \geqslant 0$.
- Main assumption: Fourier transform

$$\mathcal{F} m(\omega) = \frac{1}{(2\pi)^{d/2}} \cdot \int_{\mathbb{R}^d} e^{-i \cdot \omega^T \mathbf{x}} \cdot m(\mathbf{x}) \, d\mathbf{x}$$

  of the regression function satisfies

$$|\mathcal{F} m(\omega)| \leqslant \frac{c_3}{\|\omega\|^{d+1+\epsilon}} \quad (\omega \in \mathbb{R}^d \backslash \{0\}) \tag{1}$$

  for some $\epsilon \in (0, 1]$ and some $c_3 > 0$.

An estimator learned by gradient descent

Theorem: If

- Fourier transform $\mathcal{F}m$ satisfies (1)
- number of neurons $K_n \approx \sqrt{n}$
- $B_n \approx n^{5/2}$
- learning rate $\lambda_n \approx n^{-1.25}$
- gradient descent steps $t_n \approx n^{1.75}$

Then

$$\mathbf{E} \int |m_n(\mathbf{x}) - m(\mathbf{x})|^2 \mathbf{P}_{\mathbf{X}}(d\mathbf{x}) \leqslant c_4 \cdot (\log n)^4 \cdot \frac{1}{\sqrt{n}}.$$

On the proof

Set $\tilde{K}_n = \lceil K_n/(\log n)^4 \rceil$. **In the proof** we show that with high probability

$$\mathbf{w}(0) = (\alpha_0(0), \ldots, \alpha_{K_n}(0), \beta_1(0), \ldots, \beta_{K_n}(0), \gamma_1(0), \ldots, \gamma_{K_n}(0))$$

is chosen such that

$$\int \left| \sum_{k=1}^{\tilde{K}_n} \bar{\alpha}_{i_k} \cdot \sigma(\beta_{i_k}(0)^T \cdot \mathbf{x} + \gamma_{i_k}(0)) - m(\mathbf{x}) \right|^2 \mathbf{P}_{\mathbf{X}}(d\mathbf{x})$$

is small for some (random) $1 \leqslant i_1 < \cdots < i_{\tilde{K}_n}$ and some (random) $\bar{\alpha}_{i_1}, \ldots, \bar{\alpha}_{i_{\tilde{K}_n}} \in \mathbb{R}$,

On the proof

Set $\tilde{K}_n = \lceil K_n/(\log n)^4 \rceil$. **In the proof** we show that with high probability

$$\mathbf{w}(0) = (\alpha_0(0), \ldots, \alpha_{K_n}(0), \beta_1(0), \ldots, \beta_{K_n}(0), \gamma_1(0), \ldots, \gamma_{K_n}(0))$$

is chosen such that

$$\int \left| \sum_{k=1}^{\tilde{K}_n} \bar{\alpha}_{i_k} \cdot \sigma(\beta_{i_k}(0)^T \cdot \mathbf{x} + \gamma_{i_k}(0)) - m(\mathbf{x}) \right|^2 \mathbf{P}_{\mathbf{X}}(d\mathbf{x})$$

is small for some (random) $1 \leqslant i_1 < \cdots < i_{\tilde{K}_n}$ and some (random) $\bar{\alpha}_{i_1}, \ldots, \bar{\alpha}_{i_{\tilde{K}_n}} \in \mathbb{R}$, and that during the gradient descent the inner weights

$$\beta_{i_1}(0), \gamma_{i_1}(0), \ldots, \beta_{i_{\tilde{K}_n}}(0), \gamma_{i_{\tilde{K}_n}}(0)$$

change only slightly.

A lower bound

Under the above assumption a much better rate of convergence than $1/\sqrt{n}$ is not possible:

A lower bound

Under the above assumption a much better rate of convergence than $1/\sqrt{n}$ is not possible:

Theorem: Let $\mathcal{D}$ be the class of all distributions of $(\mathbf{X}, Y)$ which satisfy the assumptions of the above Theorem. Then

$$\inf_{\hat{m}_n} \sup_{(X,Y)\in\mathcal{D}} \mathbf{E} \int |\hat{m}_n(\mathbf{x}) - m(\mathbf{x})|^2 \mathbf{P}_{\mathbf{X}}(d\mathbf{x}) \geqslant c_5 \cdot n^{-\frac{1}{2}-\frac{1}{d+1}},$$

where the infimum is taken with respect to all estimates $\hat{m}_n$, i.e., all measurable functions of the data.

A simplified estimator

Insights in our statistical analysis help us simplify our estimate as follows:

- Choose
  - $\beta_1, \ldots, \beta_{K_n}, \gamma_1, \ldots, \gamma_{K_n}$ i.i.d.
  - $\beta_1, \ldots, \beta_{K_n}$ uniformly distributed on $\{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\| = B_n\}$
  - $\gamma_1, \ldots, \gamma_{K_n}$ uniformly distributed on $[-B_n \cdot \sqrt{d}, B_n \cdot \sqrt{d}]$
- Denote the linear function space by

$$\mathcal{F}_n = \left\{ f : \mathbb{R}^d \to \mathbb{R} : f(\mathbf{x}) = \alpha_0 + \sum_{j=1}^{K_n} \alpha_j \cdot \sigma\left(\beta_j^T \cdot \mathbf{x} + \gamma_j\right) \right.$$

$$\left. \text{for some } \alpha_0, \ldots, \alpha_{K_n} \in \mathbb{R} \right\}$$

A simplified estimator

- Choose the estimate according to the principle of least squares

$$\tilde{m}_n = \arg\min_{f \in \mathcal{F}_n} \frac{1}{n} \sum_{i=1}^{n} |Y_i - f(\mathbf{X}_i)|^2.$$

A simplified estimator

- Choose the estimate according to the principle of least squares

$$\tilde{m}_n = \arg\min_{f \in \mathcal{F}_n} \frac{1}{n} \sum_{i=1}^{n} |Y_i - f(\mathbf{X}_i)|^2.$$

- Truncate it on some level $\beta_n = c_1 \cdot \log n$

$$m_n = T_{\beta_n} \tilde{m}_n,$$

where $T_L z = \max\{\min\{z, L\}, -L\}$ for $z \in \mathbb{R}$ and $L \geqslant 0$.

A simplified estimator

Theorem: If

- the Fourier transform $\mathcal{F}m$ satisfies (1)
- number of summands $K_n \approx \sqrt{n}$
- $B_n = \frac{1}{\sqrt{d}} \cdot (\log n)^2 \cdot K_n \cdot n^2$.

Then

$$\mathbf{E} \int |m_n(\mathbf{x}) - m(\mathbf{x})|^2 \mathbf{P}_{\mathbf{X}}(d\mathbf{x}) \leqslant c_6 \cdot (\log n)^4 \cdot \frac{1}{\sqrt{n}}.$$

A simplified estimator

- Same rate as for the neural network estimate learned by gradient descent, **but** much faster in computation
- Ability to learn a good hierarchical representation of the data is considered as a key factor of Deep Learning
  $\rightsquigarrow$ So-called representation learning (see Goodfellow et al. (2016))
  Suprisingly: In our estimate it is much more a representation **guessing**

Application to simulated data

The setting:

- Generate $n \in \{200, 400\}$ independent observations from

$$Y = m_i(\mathbf{X}) + \sigma_j \cdot \gamma_i \cdot \epsilon \quad (i \in \{1, \ldots, 6\}, j \in \{1, 2\})$$

  with
  - $\mathbf{X}$ uniformly distributed on $[0, 1]^d$
  - ϵ standard normally distributed
  - $\sigma_j \in \{0.05, 0.2\}$
  - γ_i covers the range of m_i on $\mathbf{X}$

Application to simulated data

Implementation of the estimate:

- Choose
  - $\beta_1, \ldots, \beta_{K_n}$ uniformly distributed on $\{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\| = B_n\}$
  - $\gamma_1, \ldots, \gamma_{K_n}$ uniformly distributed on $[-B_n \cdot \sqrt{d}, B_n \cdot \sqrt{d}]$
- Solve a linear equation system to compute the values of $\alpha_0, \ldots, \alpha_{K_n}$

Application to simulated data

Implementation of the estimate:

- Choose
  - $\beta_1, \ldots, \beta_{K_n}$ uniformly distributed on $\{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\| = B_n\}$
  - $\gamma_1, \ldots, \gamma_{K_n}$ uniformly distributed on $[-B_n \cdot \sqrt{d}, B_n \cdot \sqrt{d}]$
- Solve a linear equation system to compute the values of $\alpha_0, \ldots, \alpha_{K_n}$
- Choose the values of B_n and K_n by the splitting of the sample with

$$n_{train} = 0.8 \cdot n \text{ and } n_{test} = 0.2 \cdot n$$

  out of the sets
  - $K_n \in \{1, 2, \ldots, 6, 8, 16, \ldots, 131072\}$
  - $B_n \in \{1, 2, \ldots, 6, 8, 16, \ldots, 524288\}$

Application to simulated data

Competitive approaches:

- **Standard nets** of the Deep Learning framework of *tensorflow* and *keras* with
  - ReLU activation function (abbrv. *relu-net*)
  - Sigmoid activation function (abbrv. *sig-net*)

  and
  - 1 hidden layer (abbrv. *net-1*)
  - 3 hidden layers (abbrv. *net-3*)
  - 6 hidden layers (abbrv. *net-6*)
- Choose number of neurons with the splitting of the sample procedure out of the set $\{4, 8, 16, 32, 64\}$

Application to simulated data

Test functions:

$$m_1(\mathbf{x}) = \exp\left(0.5 \cdot \sum_{i=1}^{7} x_i^2\right),$$

$$m_2(\mathbf{x}) = \frac{1}{4000} \cdot \sum_{i=1}^{7} x_i^2 - \prod_{i=1}^{7} \cos\left(\frac{x_i}{\sqrt{i-1}}\right)$$

$$m_3(\mathbf{x}) = \sum_{i=1}^{6} 100 \cdot \left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2,$$

$$m_4(\mathbf{x}) = \frac{1}{1 + \frac{\|\mathbf{x}\|}{4}} + x_7^2 + x_4 \cdot x_5 \cdot x_2,$$

4. Application to simulated data

The test functions:

$$m_5(\mathbf{x}) = \tanh(0.2x_1 + 0.9x_2 + x_3 + x_4 + 0.2 \cdot x_5 + 0.6 \cdot x_6).$$

$$m_6(\mathbf{x}) = \cot\left(\frac{\pi}{1 + \exp(x_1^2 + 2 \cdot x_2 + \sin(6 \cdot x_4^2 - 3))}\right) + \exp(3 \cdot x_3 + 2 \cdot x_4 - 5 \cdot x_5 + \sqrt{x_6 + 0.9 \cdot x_7 + 0.1}),$$

Application to simulated data

Evaluation:

- Quality of each of the estimates is determined by the empirical L_2-error, i.e. , by

$$\epsilon_{L_2,N} = \frac{1}{N}\sum_{k=1}^{N}(m_{n,i}(\mathbf{X}_{n+k}) - m_i(\mathbf{X}_{n+k})^2),$$

- Input values $\mathbf{X}_{n+1}, \mathbf{X}_{n+2}, \ldots, \mathbf{X}_{n+N}$ are newly generated independent realizations of the random value $\mathbf{X}$
- $N = 10^5$
- Each estimation is performed 50 times with different values of $(\mathbf{X}, \epsilon)$

Application to simulated data

Results:

$$m_1(\mathbf{x}) = \exp\left(0.5 \cdot \sum_{i=1}^{7} x_i^2\right)$$

noise	5%		20%	
sample size	$n = 200$	$n = 400$	$n = 200$	$n = 400$
$\bar{\epsilon}_{L_2,N}(avg)$	0.6823	0.6823	0.6820	0.6825
relu-net-1	0.0554(0.0126)	0.0309(0.0132)	0.0586(0.0099)	0.0325(0.0137)
relu-net-3	0.0632(0.0110)	0.0428(0.0100)	0.0645(0.0094)	0.0449(0.0135)
relu-net-6	0.0669(0.0108)	0.0518(0.0143)	0.0676(0.0125)	0.0489(0.0134)
sig-net-1	0.0763(0.0088)	0.0615(0.0038)	0.0768(0.0087)	0.0623(0.0033)
sig-net-3	0.0967(0.0150)	0.0683(0.0036)	0.1006(0.0140)	0.0685(0.0042)
sig-net-6	0.1335(0.0808)	0.0676(0.0054)	0.1424(0.0712)	0.0684(0.0047)
lsq-est	**0.0014**(0.0009)	**0.0006**(0.0003)	**0.0079**(0.0025)	**0.0020**(0.0006)

Application to simulated data

Results:

$$m_2(\mathbf{x}) = \frac{1}{4000} \cdot \sum_{i=1}^{7} x_i^2 - \prod_{i=1}^{7} \cos\left(\frac{x_i}{\sqrt{i-1}}\right)$$

noise	5%		20%	
sample size	$n = 200$	$n = 400$	$n = 200$	$n = 400$
$\bar{\epsilon}_{L_2,N}(avg)$	2.2190	2.2154	2.2159	2.2151
relu-net-1	0.1123(0.0267)	0.0582(0.0070)	0.1074(0.0314)	0.0561(0.0106)
relu-net-3	0.1015(0.0252)	0.0607(0.0104)	0.0977(0.0266)	0.0629(0.0144)
relu-net-6	0.0950(0.0261)	0.0642(0.0084)	0.0902(0.0292)	0.0630(0.0144)
sig-net-1	0.2278(0.0821)	0.0918(0.0266)	0.2117(0.0932)	0.0920(0.0273)
sig-net-3	0.3058(0.4057)	0.0745(0.0117)	0.2487(0.5568)	0.0736(0.0123)
sig-net-6	0.9654(0.6720)	0.1045(0.0420)	0.8871(0.6894)	0.0943(0.0322)
lsq-est	**0.0303**(0.0095)	**0.0167**(0.0043)	**0.0294**(0.0113)	**0.0169**(0.0052)

Application to simulated data

Results:

$$m_3(\mathbf{x}) = \sum_{i=1}^{6} 100 \cdot \left(x_{i+1} - (x_i)^2\right)^2 + (x_i - 1)^2$$

noise	5%		20%	
sample size	$n = 200$	$n = 400$	$n = 200$	$n = 400$
$\bar{\epsilon}_{L_2,N}(avg)$	25.4803	25.4734	25.4906	25.4864
relu-net-1	0.6428(0.2389)	0.2083(0.0359)	0.7063(0.1625)	0.2621(0.0773)
relu-net-3	0.6845(0.2401)	0.1846(0.0455)	0.6757(0.3474)	0.2567(0.0660)
relu-net-6	0.5204(0.2810)	0.2114(0.1316)	0.6455(0.3179)	0.2878(0.0960)
sig-net-1	0.9820(0.0297)	0.9442(0.0350)	0.9851(0.0354)	0.9516(0.0397)
sig-net-3	1.0175(0.0322)	0.9934(0.0358)	1.0047(0.0248)	1.0025(0.0324)
sig-net-6	1.1014(0.0275)	1.0016(0.0253)	1.0064(0.0271)	1.0058(0.0247)
lsq-est	**0.2181**(0.0577)	**0.1024**(0.0161)	**0.3970**(0.1589)	**0.1778**(0.0447)

Application to simulated data

Results:

$$m_4(\mathbf{x}) = \frac{1}{1 + \frac{\|\mathbf{x}\|}{4}} + x_7^2 + x_4 \cdot x_5 \cdot x_2$$

noise	5%		20%	
sample size	$n = 200$	$n = 400$	$n = 200$	$n = 400$
$\bar{\epsilon}_{L_2,N}(avg)$	0.0049	0.0049	0.0049	0.0049
relu-net-1	0.2069(0.1520)	0.0290(0.0172)	0.1889(0.1260)	0.0475(0.0219)
relu-net-3	0.2531(0.2178)	0.0245(0.0283)	0.2292(0.1971)	0.0427(0.0256)
relu-net-6	0.0760(0.2178)	0.0276(0.0290)	0.1058(0.1280)	0.0349(0.0386)
sig-net-1	0.1245(0.0188)	0.0038(0.0077)	0.1243(0.0597)	0.0100(0.0121)
sig-net-3	0.1375(0.5441)	**0.0032**(0.0026)	0.0831(0.2748)	**0.0055**(0.0058)
sig-net-6	0.1766(0.8157)	0.0147(0.0284)	0.2949(0.9867)	0.0117(0.0195)
lsq-est	**0.0188**(0.0106)	0.0077(0.0046)	**0.0469**(0.0187)	0.0230(0.0079)

Application to simulated data

Results:

$$m_5(\mathbf{x}) = \tanh(0.2 \cdot x_1 + 0.9 \cdot x_2 + x_3 + x_4 + 0.2 \cdot x_5 + 0.6 \cdot x_6).$$

noise	5%		20%	
sample size	$n = 200$	$n = 400$	$n = 200$	$n = 400$
$\bar{\epsilon}_{L_2,N}(avg)$	0.0135	0.0135	0.1345	0.0135
relu-net-1	0.1989(0.0737)	0.1019(0.0264)	0.2846(0.1094)	0.1768(0.0618)
relu-net-3	0.1826(0.0667)	0.1061(0.0454)	0.3182(0.1858)	0.2230(0.0808)
relu-net-6	0.1744(0.1099)	0.1066(0.0264)	0.3519(0.1995)	0.2005(0.0621)
sig-net-1	0.0971(0.0183)	0.0784(0.0071)	**0.1139**(0.0202)	**0.0880**(0.0131)
sig-net-3	**0.0935**(0.0202)	0.0796(0.0049)	0.1190(0.0434)	**0.0880**(0.0138)
sig-net-6	0.1396(0.0547)	0.0850(0.0087)	0.1816(0.0434)	0.0983(0.0207)
lsq-est	0.1025(0.0343)	**0.0516**(0.0118)	0.1682(0.0434)	0.0981(0.0181)

Application to simulated data

Results:

$$m_6(\mathbf{x}) = \cot\left(\frac{\pi}{1+\exp(x_1^2 + 2\cdot x_2 + \sin(6\cdot x_4^2 - 3)}\right) + \exp(3\cdot x_3 + 2\cdot x_4 - 5\cdot x_5 + \sqrt{x_6 + 0.9\cdot x_7 + 0.1})$$

noise	5%		20%	
sample size	$n = 200$	$n = 400$	$n = 200$	$n = 400$
$\bar{\epsilon}_{L_2,N}(avg)$	591.77	592.74	591.10	589.81
relu-net-1	0.1897(0.0913)	0.0552(0.0346)	0.2195(0.0876)	0.0845(0.0378)
relu-net-3	**0.1349**(0.0828)	0.0398(0.0315)	0.1383(0.0828)	0.0467(0.0363)
relu-net-6	0.1075(0.0856)	**0.0385**(0.0247)	**0.1073**(0.0657)	**0.0434**(0.0249)
sig-net-1	0.5733(0.1122)	0.2027(0.0539)	0.5572(0.0852)	0.1877(0.1123)
sig-net-3	0.8355(0.1451)	0.3905(0.1181)	0.8697(0.1728)	0.3805(0.0822)
sig-net-6	1.0058(0.0728)	0.7017(0.1435)	0.9955(0.0912)	0.7332(0.1188)
lsq-est	0.3070(0.1534)	0.1784(0.0758)	0.2643(0.1951)	0.1655(0.0570)

Application to simulated data

Improvement of the results:

- Choose a combined estimator *comb-new*, that chooses between the estimators
  - New least squares estimator
  - Standard nets

  the one with the smallest empirical L_2-error on the dataset x_{test}
- Compare this estimator with a classical combined estimator *comb-class*, that chooses the best standard net according to the smallest empirical L_2-error on the test sample

Application to simulated data

Results:

$$m_1(\mathbf{x}) = \exp\left(0.5 \cdot \sum_{i=1}^{7} x_i^2\right)$$

noise	5%		20%	
sample size	$n = 200$	$n = 400$	$n = 200$	$n = 400$
$\bar{\epsilon}_{L_2,N}(avg)$	0.6823	0.6823	0.6820	0.6825
relu-net-1	0.0554(0.0126)	0.0309(0.0132)	0.0586(0.0099)	0.0325(0.0137)
relu-net-3	0.0632(0.0110)	0.0428(0.0100)	0.0645(0.0094)	0.0449(0.0135)
relu-net-6	0.0669(0.0108)	0.0518(0.0143)	0.0676(0.0125)	0.0489(0.0134)
sig-net-1	0.0763(0.0088)	0.0615(0.0038)	0.0768(0.0087)	0.0623(0.0033)
sig-net-3	0.0967(0.0150)	0.0683(0.0036)	0.1006(0.0140)	0.0685(0.0042)
sig-net-6	0.1335(0.0808)	0.0676(0.0054)	0.1424(0.0712)	0.0684(0.0047)
comb-classic	0.0552(0.0127)	0.0308(0.0127)	0.0572(0.0098)	0.0319(0.0113)
lsq-est	**0.0014**(0.0009)	**0.0006**(0.0003)	**0.0079**(0.0025)	**0.0020**(0.0006)
comb-new	**0.0014**(0.0009)	**0.0006**(0.0003)	**0.0079**(0.0025)	**0.0020**(0.0006)

Application to simulated data

Results:

$$m_2(\mathbf{x}) = \frac{1}{4000} \cdot \sum_{i=1}^{7} x_i^2 - \prod_{i=1}^{7} \cos\left(\frac{x_i}{\sqrt{i-1}}\right)$$

noise	5%		20%	
sample size	$n = 200$	$n = 400$	$n = 200$	$n = 400$
$\bar{\epsilon}_{L_2,N}(avg)$	2.2190	2.2154	2.2159	2.2151
relu-net-1	0.1123(0.0267)	0.0582(0.0070)	0.1074(0.0314)	0.0561(0.0106)
relu-net-3	0.1015(0.0252)	0.0607(0.0104)	0.0977(0.0266)	0.0629(0.0144)
relu-net-6	0.0950(0.0261)	0.0642(0.0084)	0.0902(0.0292)	0.0630(0.0144)
sig-net-1	0.2278(0.0821)	0.0918(0.0266)	0.2117(0.0932)	0.0920(0.0273)
sig-net-3	0.3058(0.4057)	0.0745(0.0117)	0.2487(0.5568)	0.0736(0.0123)
sig-net-6	0.9654(0.6720)	0.1045(0.0420)	0.8871(0.6894)	0.0943(0.0322)
comb-classic	0.0910(0.0238)	0.0560(0.0108)	0.0893(0.0200)	0.0534(0.0111)
lsq-est	**0.0303**(0.0095)	**0.0167**(0.0043)	**0.0294**(0.0113)	**0.0169**(0.0052)
comb-new	**0.0303**(0.0095)	**0.0167**(0.0043)	**0.0294**(0.0113)	**0.0169**(0.0052)

Application to simulated data

Results:

$$m_3(\mathbf{x}) = \sum_{i=1}^{6} 100 \cdot \left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2$$

noise	5%		20%	
sample size	$n = 200$	$n = 400$	$n = 200$	$n = 400$
$\bar{\epsilon}_{L_2,N}(avg)$	25.4803	25.4734	25.4906	25.4864
relu-net-1	0.6428(0.2389)	0.2083(0.0359)	0.7063(0.1625)	0.2621(0.0773)
relu-net-3	0.6845(0.2401)	0.1846(0.0455)	0.6757(0.3474)	0.2567(0.0660)
relu-net-6	0.5204(0.2810)	0.2114(0.1316)	0.6455(0.3179)	0.2878(0.0960)
sig-net-1	0.9820(0.0297)	0.9442(0.0350)	0.9851(0.0354)	0.9516(0.0397)
sig-net-3	1.0175(0.0322)	0.9934(0.0358)	1.0047(0.0248)	1.0025(0.0324)
sig-net-6	1.1014(0.0275)	1.0016(0.0253)	1.0064(0.0271)	1.0058(0.0247)
comb-classic	0.4630(0.2149)	0.1698(0.0386)	0.5341(0.2286)	0.2427(0.0456)
lsq-est	**0.2181**(0.0577)	**0.1024**(0.0161)	**0.3970**(0.1589)	**0.1778**(0.0447)
comb-new	**0.2182**(0.0577)	**0.1024**(0.0161)	0.4094(0.1472)	0.1783(0.0486)

Application to simulated data

Results:

$$m_4(\mathbf{x}) = \frac{1}{1 + \frac{\|x\|}{4}} + x_7^2 + x_4 \cdot x_5 \cdot x_2$$

noise	5%		20%	
sample size	$n = 200$	$n = 400$	$n = 200$	$n = 400$
$\bar{\epsilon}_{L_2,N}(avg)$	0.0049	0.0049	0.0049	0.0049
relu-net-1	0.2069(0.1520)	0.0290(0.0172)	0.1889(0.1260)	0.0475(0.0219)
relu-net-3	0.2531(0.2178)	0.0245(0.0283)	0.2292(0.1971)	0.0427(0.0256)
relu-net-6	0.0760(0.2178)	0.0276(0.0290)	0.1058(0.1280)	0.0349(0.0386)
sig-net-1	0.1245(0.0188)	0.0038(0.0077)	0.1243(0.0597)	0.0100(0.0121)
sig-net-3	0.1375(0.5441)	0.0032(0.0026)	0.0831(0.2748)	0.0055(0.0058)
sig-net-6	0.1766(0.8157)	0.0147(0.0284)	0.2949(0.9867)	0.0117(0.0195)
comb-classic	0.0445(0.0455)	**0.0028**(0.0033)	0.0594(0.0620)	**0.0052**(0.0067)
lsq-est	**0.0188**(0.0106)	0.0077(0.0046)	0.0469(0.0187)	0.0230(0.0079)
comb-new	**0.0184**(0.0142)	**0.0028**(0.0033)	**0.0437**(0.0262)	**0.0052**(0.0050)

Application to simulated data

Results:

$$m_5(\mathbf{x}) = \tanh(0.2 \cdot x_1 + 0.9 \cdot x_2 + x_3 + x_4 + 0.2 \cdot x_5 + 0.6 \cdot x_6).$$

noise	5%		20%	
sample size	$n = 200$	$n = 400$	$n = 200$	$n = 400$
$\bar{\epsilon}_{L_2,N}(avg)$	0.0135	0.0135	0.1345	0.0135
relu-net-1	0.1989(0.0737)	0.1019(0.0264)	0.2846(0.1094)	0.1768(0.0618)
relu-net-3	0.1826(0.0667)	0.1061(0.0454)	0.3182(0.1858)	0.2230(0.0808)
relu-net-6	0.1744(0.1099)	0.1066(0.0264)	0.3519(0.1995)	0.2005(0.0621)
sig-net-1	0.0971(0.0183)	0.0784(0.0071)	0.1139(0.0202)	0.0880(0.0131)
sig-net-3	0.0935(0.0202)	0.0796(0.0049)	0.1190(0.0434)	0.0880(0.0138)
sig-net-6	0.1396(0.0547)	0.0850(0.0087)	0.1816(0.0434)	0.0983(0.0207)
comb-class	0.0948(0.0187)	0.0794(0.0869)	**0.1121**(0.0210)	**0.0870**(0.0116)
lsq-est	0.1025(0.0343)	**0.0516**(0.0118)	0.1682(0.0434)	0.0981(0.0181)
comb-new	**0.0923**(0.0198)	**0.0516**(0.0118)	0.1162(0.0392)	0.0918(0.0131)

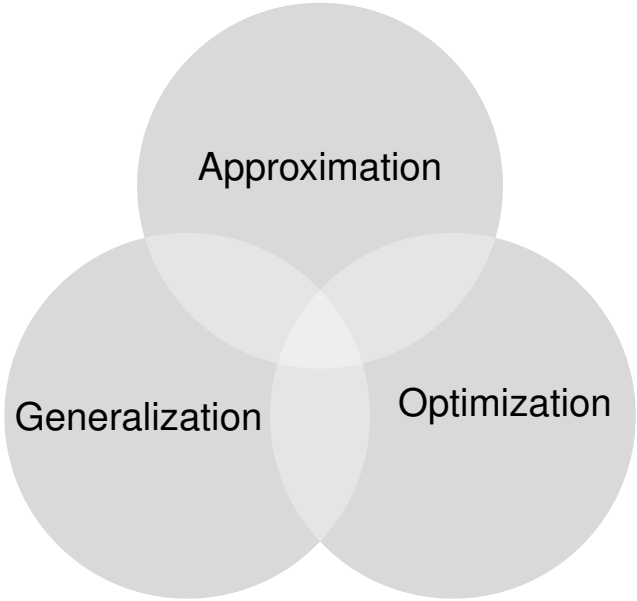
Application to simulated data

Results: $$m_6(\mathbf{x}) = \cot\left(\frac{\pi}{1+\exp(x_1^2 + 2\cdot x_2 + \sin(6\cdot x_4^2 - 3)}\right) + \exp(3\cdot x_3 + 2\cdot x_4 - 5\cdot x_5 + \sqrt{x_6 + 0.9\cdot x_7 + 0.1})$$

noise	5%		20%	
sample size	$n = 200$	$n = 400$	$n = 200$	$n = 400$
$\bar{\epsilon}_{L_2,N}(avg)$	591.77	592.74	591.10	589.81
relu-net-1	0.1897(0.0913)	0.0552(0.0346)	0.2195(0.0876)	0.0845(0.0378)
relu-net-3	0.1349(0.0828)	0.0398(0.0315)	0.1383(0.0828)	0.0467(0.0363)
relu-net-6	0.1075(0.0856)	0.0385(0.0247)	**0.1073**(0.0657)	0.0434(0.0249)
sig-net-1	0.5733(0.1122)	0.2027(0.0539)	0.5572(0.0852)	0.1877(0.1123)
sig-net-3	0.8355(0.1451)	0.3905(0.1181)	0.8697(0.1728)	0.3805(0.0822)
sig-net-6	1.0058(0.0728)	0.7017(0.1435)	0.9955(0.0912)	0.7332(0.1188)
comb-classic	**0.0991**(0.0749)	**0.0331**(0.0242)	0.1173(0.0842)	**0.0352**(0.0229)
lsq-est	0.3070(0.1534)	0.1784(0.0758)	0.2643(0.1951)	0.1655(0.0570)
comb-new	0.1063(0.0803)	**0.0331**(0.0394)	0.1214(0.0902)	**0.0352**(0.0229)

Summary

- In the analysis of our first estimate all three aspects of Deep Learning, namely approximation, generalization and optimization, were considered simultaneously
- Statistical insights helped us to construct a simplified estimate, which can be much faster computed in applications
- The combined estimate outperforms all others in our simulation study



Eliezer Yudkowsky *„By far, the greatest danger for artificial intelligence is that people conclude too early that they understand it.“*

Thank you for your attention!