Deep Learning as a Mathematician

Philipp Grohs

universität wien

Faculty of Mathematics

September 12th 2017

Syllabus

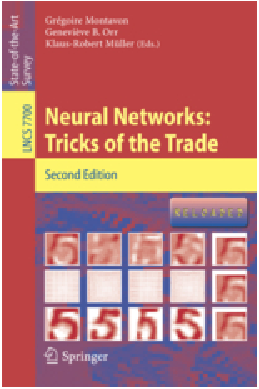
1 Why Mathematical Understanding?

Curiosity

It is the guiding principle of many applied mathematicians that if something mathematical works really well, there must be a good underlying mathematical reason for it, and we ought to be able to understand it.

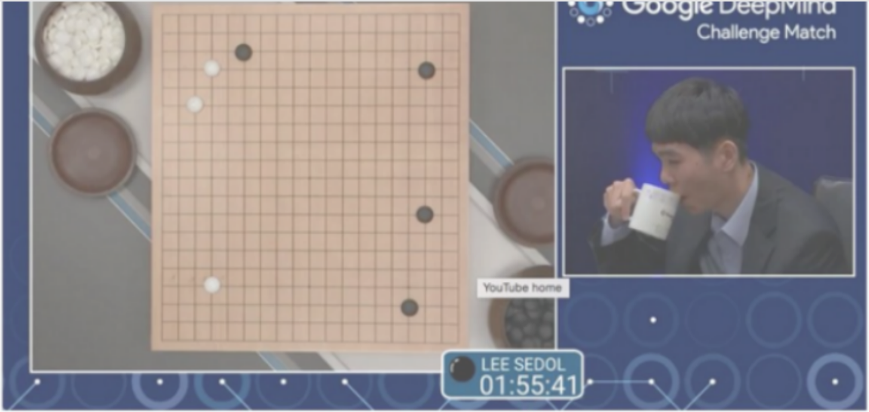
[Ingrid Daubechies. *Big Data's Mathematical Myteries*, Quanta Magazine (2015)]

Improve Usability/Availability



$\sim$ 800 page book explaining various ad-hoc tricks, which are necessary for good performance.

Be More Efficient



AlphaGO	Lee Se-dol
1202 CPUs, 176 GPUs, 100+ Scientists.	1 Human Brain, 1 Coffee.

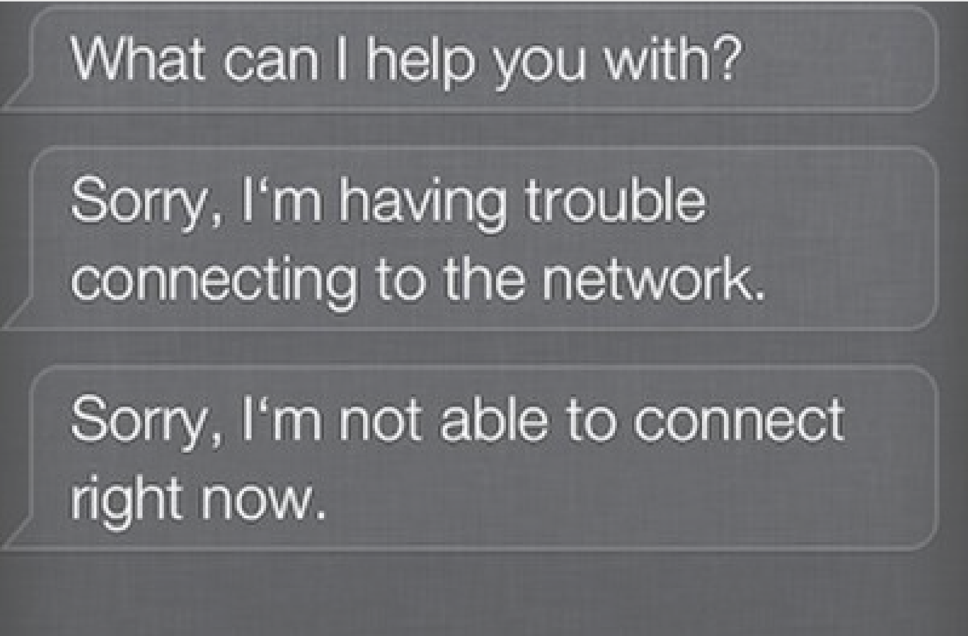
Be More Efficient

Electricity bill for one game of AlphaGo:

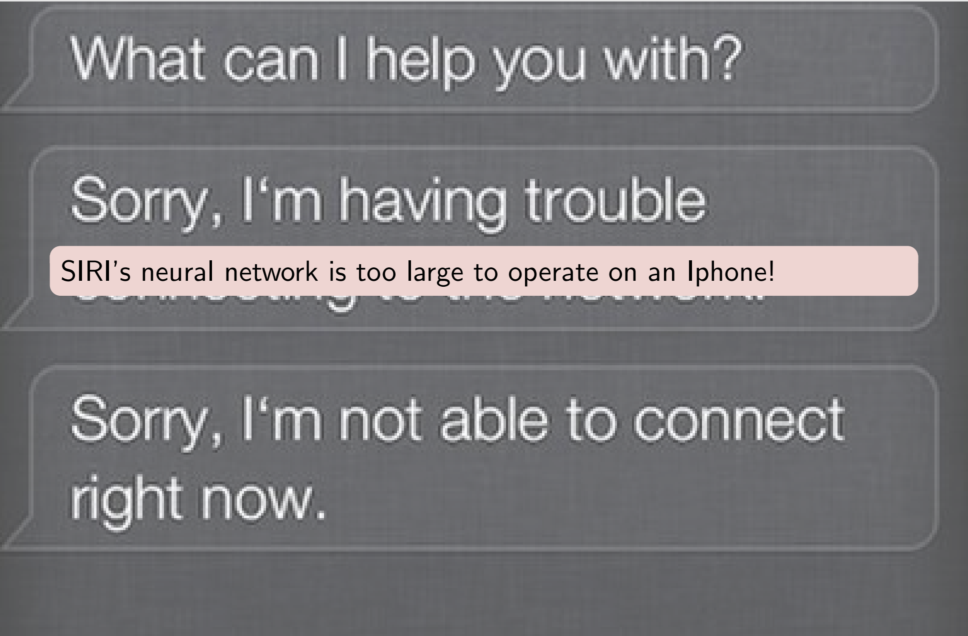
Be More Efficient

Electricity bill for one game of AlphaGo: $\sim$ 3000 USD

Be More Efficient



Be More Efficient



SIRI's neural network is too large to operate on an Iphone!

Today's Focus: Vanilla Neural Networks Regression

Today's Focus: Vanilla Neural Networks Regression

Neural Network Hypothesis Class

Given $d, L, N_1, \ldots, N_L$ and σ define the associated hypothesis class

$$\mathcal{H}_{[d,N_1,\ldots,N_L],\sigma} :=$$
$$\left\{ A_L\sigma\left(A_{L-1}\sigma\left(\ldots\sigma\left(A_1(x)\right)\right)\right) : \; A_\ell : \mathbb{R}^{N_{\ell-1}} \to \mathbb{R}^{N_\ell} \text{ affine linear } \right\}.$$

Today's Focus: Vanilla Neural Networks Regression

Neural Network Hypothesis Class

Given $d, L, N_1, \ldots, N_L$ and σ define the associated hypothesis class

$$\mathcal{H}_{[d,N_1,\ldots,N_L],\sigma} :=$$
$$\left\{ A_L \sigma\left(A_{L-1}\sigma\left(\ldots\sigma\left(A_1(x)\right)\right)\right) : \ A_\ell : \mathbb{R}^{N_{\ell-1}} \to \mathbb{R}^{N_\ell} \text{ affine linear } \right\}.$$

Simplest Regression/Classification Task

Given data $\mathbf{z} = ((x_i, y_i))_{i=1}^m \subset \mathbb{R}^d \times \mathbb{R}^{N_L}$, find the empirical regression function

$$f_{\mathbf{z}} \in \operatorname{argmin}_{f \in \mathcal{H}_{[d,N_1,\ldots,N_L],\sigma}} \sum_{i=1}^m \mathcal{L}(f, x_i, y_i),$$

where $\mathcal{L} : C(\mathbb{R}^d) \times \mathbb{R}^d \times \mathbb{R}^{N_L} \to \mathbb{R}_+$ is the *loss function* (in least squares problems we have $\mathcal{L}(f, x, y) = |f(x) - y|^2$).

Example: Handwritten Digits

MNIST Database for handwritten digit recognition http://yann.lecun.com/exdb/mnist/

Example: Handwritten Digits

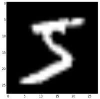
MNIST Database for hand-written digit recognition http://yann.lecun.com/exdb/mnist/

- Every image is given as a 28×28 matrix $x \in \mathbb{R}^{28\times 28} \sim \mathbb{R}^{784}$:

Example: Handwritten Digits

MNIST Database for handwritten digit recognition `http://yann.lecun.com/exdb/mnist/`

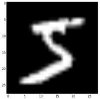
- Every image is given as a 28×28 matrix $x \in \mathbb{R}^{28\times 28} \sim \mathbb{R}^{784}$:



Example: Handwritten Digits

MNIST Database for handwritten digit recognition http://yann.lecun.com/exdb/mnist/

- Every image is given as a 28×28 matrix $x \in \mathbb{R}^{28\times 28} \sim \mathbb{R}^{784}$:

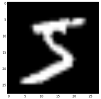


- Every label is given as a 10-dim vector $y \in \mathbb{R}^{10}$ describing the 'probability' of each digit

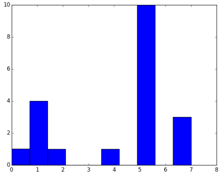
Example: Handwritten Digits

MNIST Database for handwritten digit recognition `http://yann.lecun.com/exdb/mnist/`

- Every image is given as a 28×28 matrix $x \in \mathbb{R}^{28\times 28} \sim \mathbb{R}^{784}$:

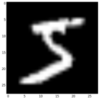


- Every label is given as a 10-dim vector $y \in \mathbb{R}^{10}$ describing the 'probability' of each digit

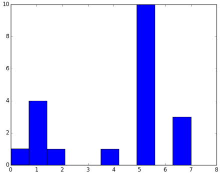


Example: Handwritten Digits

- Every image is given as a 28×28 matrix $x \in \mathbb{R}^{28\times 28} \sim \mathbb{R}^{784}$:

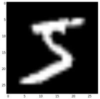


- Every label is given as a 10-dim vector $y \in \mathbb{R}^{10}$ describing the 'probability' of each digit

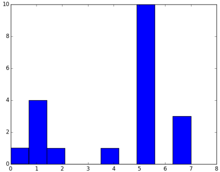


Example: Handwritten Digits

- Every image is given as a 28×28 matrix $x \in \mathbb{R}^{28\times 28} \sim \mathbb{R}^{784}$:



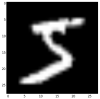
- Every label is given as a 10-dim vector $y \in \mathbb{R}^{10}$ describing the 'probability' of each digit



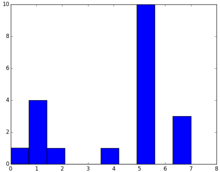
- Given labeled *training data* $(x_i, y_i)_{i=1}^{m} \subset \mathbb{R}^{784} \times \mathbb{R}^{10}$.

Example: Handwritten Digits

- Every image is given as a 28×28 matrix $x \in \mathbb{R}^{28\times 28} \sim \mathbb{R}^{784}$:



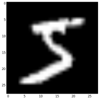
- Every label is given as a 10-dim vector $y \in \mathbb{R}^{10}$ describing the ‘probability’ of each digit



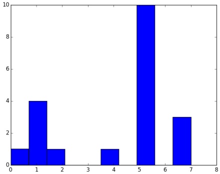
- Given labeled *training data* $(x_i, y_i)_{i=1}^{m} \subset \mathbb{R}^{784} \times \mathbb{R}^{10}$.
- Fix network topology, e.g., number of layers (for example $L = 3$) and numbers of neurons ($N_1 = 20, \; N_2 = 20$).

Example: Handwritten Digits

- Every image is given as a 28×28 matrix $x \in \mathbb{R}^{28\times 28} \sim \mathbb{R}^{784}$:



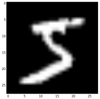
- Every label is given as a 10-dim vector $y \in \mathbb{R}^{10}$ describing the ‘probability’ of each digit



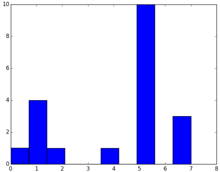
- Given labeled *training data* $(x_i, y_i)_{i=1}^{m} \subset \mathbb{R}^{784} \times \mathbb{R}^{10}$.
- Fix network topology, e.g., number of layers (for example $L = 3$) and numbers of neurons $(N_1 = 20,\ N_2 = 20)$.
- The learning goal is to find the empirical regression function $f_{\mathbf{z}} \in \mathcal{H}_{[784,20,20,10],\sigma}$.

Example: Handwritten Digits

- Every image is given as a 28×28 matrix $x \in \mathbb{R}^{28\times 28} \sim \mathbb{R}^{784}$:



- Every label is given as a 10-dim vector $y \in \mathbb{R}^{10}$ describing the 'probability' of each digit



- Given labeled *training data* $(x_i, y_i)_{i=1}^m \subset \mathbb{R}^{784} \times \mathbb{R}^{10}$.
- Fix network topology, e.g., number of layers (for example $L = 3$) and numbers of neurons ($N_1 = 20,\ N_2 = 20$).
- The learning goal is to find the empirical regression function $f_{\mathbf{z}} \in \mathcal{H}_{[784,20,20,10],\sigma}$.
- Typically solved by stochastic first order approximation methods.

How Can Mathematics Contribute?

How Can Mathematics Contribute?

1. Approximation Power of (Convolutional) Neural Networks

How Can Mathematics Contribute?

1. Approximation Power of (Convolutional) Neural Networks
2. Convergence Properties of Stochastic Optimization Algorithms

How Can Mathematics Contribute?

1. Approximation Power of (Convolutional) Neural Networks
2. Convergence Properties of Stochastic Optimization Algorithms
3. Generalization of Neural Networks

How Can Mathematics Contribute?

1. Approximation Power of (Convolutional) Neural Networks
2. Convergence Properties of Stochastic Optimization Algorithms
3. Generalization of Neural Networks
4. Invariances and Discriminatory Properties

This Talk

1. Approximation Power of (Convolutional) Neural Networks
2. Convergence Properties of Stochastic Optimization Algorithms
3. Generalization of Neural Networks
4. Invariances and Discriminatory Properties

1. Approximation Power

Universal Approximation Theorem

Theorem [Cybenko (1989), Hornik (1991)]

Suppose that $\sigma : \mathbb{R} \to \mathbb{R}$ continuous is not a polynomial and fix $d \geq 1, L \geq 2, N_L \geq 1 \in \mathbb{N}$ and a compact subset $K \subset \mathbb{R}^d$. Then for any continuous $f : \mathbb{R}^d \to \mathbb{R}^{N_L}$ and any $\varepsilon > 0$ there exist $N_1, \ldots, N_{L-1} \in \mathbb{N}$ and affine linear maps $A_\ell : \mathbb{R}^{N_{\ell-1}} \to \mathbb{R}^{N_\ell}$, $1 \leq \ell \leq L$ such that the neural network

$$\Phi(x) = A_L \sigma \left(A_{L-1} \sigma \left(\ldots \sigma \left(A_1(x) \right) \right) \right), \quad x \in \mathbb{R}^d,$$

approximates f to within accuracy ε, i.e.,

$$\sup_{x \in K} |f(x) - \Phi(x)| \leq \varepsilon.$$

Universal Approximation Theorem

Theorem [Cybenko (1989), Hornik (1991)]

Suppose that $\sigma : \mathbb{R} \to \mathbb{R}$ continuous is not a polynomial and fix $d \geq 1, L \geq 2, N_L \geq 1 \in \mathbb{N}$ and a compact subset $K \subset \mathbb{R}^d$. Then for any continuous $f : \mathbb{R}^d \to \mathbb{R}^{N_L}$ and any $\varepsilon > 0$ there exist $N_1, \ldots, N_{L-1} \in \mathbb{N}$ and affine linear maps $A_\ell : \mathbb{R}^{N_{\ell-1}} \to \mathbb{R}^{N_\ell}$, $1 \leq \ell \leq L$ such that the neural network

$$\Phi(x) = A_L \sigma\left(A_{L-1} \sigma\left(\ldots \sigma\left(A_1(x)\right)\right)\right), \quad x \in \mathbb{R}^d,$$

approximates f to within accuracy ε, i.e.,

$$\sup_{x \in K} |f(x) - \Phi(x)| \leq \varepsilon.$$

Does not imply any quantitative results (e.g., how many nodes to achieve a desired accuracy?).

A Quantitative Universal Approximation Theorem

Theorem [Maiorov - Pinkus (1999)]

There exists an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ that is smooth, monotone increasing and sigmoidal ($\lim_{t\to\infty} \sigma(t) = 1$ and $\lim_{t\to-\infty} \sigma(t) = 0$) with the following property: For any $\varepsilon > 0$, any $d \geq 1$, any compact subset $K \subset \mathbb{R}^d$ and any continuous $f : \mathbb{R}^d \to \mathbb{R}^{N_L}$ there exist affine linear maps $A_1 : \mathbb{R}^d \to \mathbb{R}^{3d}$, $A_2 : \mathbb{R}^{3d} \to \mathbb{R}^{6d+3}$, $A_2 : \mathbb{R}^{6d+3} \to \mathbb{R}^{N_L}$ such that the neural network

$$\Phi(x) = A_3\sigma\left(A_2\sigma\left(A_1(x)\right)\right), \quad x \in \mathbb{R}^d,$$

approximates f to within accuracy ε, i.e.,

$$\sup_{x\in K} |f(x) - \Phi(x)| \leq \varepsilon.$$

A Quantitative Universal Approximation Theorem

Theorem [Maiorov - Pinkus (1999)]

There exists an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ that is smooth, monotone increasing and sigmoidal ($\lim_{t\to\infty} \sigma(t) = 1$ and $\lim_{t\to-\infty} \sigma(t) = 0$) with the following property: For any $\varepsilon > 0$, any $d \geq 1$, any compact subset $K \subset \mathbb{R}^d$ and any continuous $f : \mathbb{R}^d \to \mathbb{R}^{N_L}$ there exist affine linear maps $A_1 : \mathbb{R}^d \to \mathbb{R}^{3d}$, $A_2 : \mathbb{R}^{3d} \to \mathbb{R}^{6d+3}$, $A_2 : \mathbb{R}^{6d+3} \to \mathbb{R}^{N_L}$ such that the neural network

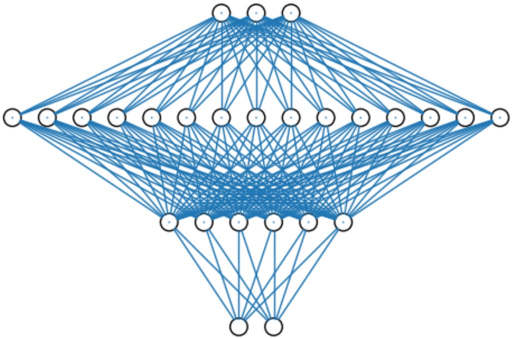
$$\Phi(x) = A_3\sigma\left(A_2\sigma\left(A_1(x)\right)\right), \quad x \in \mathbb{R}^d,$$

approximates f to within accuracy ε, i.e.,

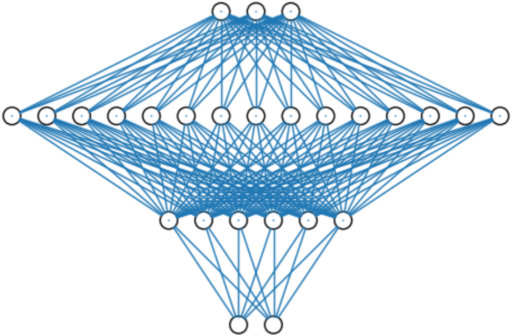
$$\sup_{x\in K} |f(x) - \Phi(x)| \leq \varepsilon.$$

In other words, we can approximate *any* function up to *any* accuracy with a *fixed* number of coefficients????

A Universal Architecture



A Universal Architecture



Where is the catch?

A Meaningful Notion of Approximation

Definition

Let $K \subset \mathbb{R}^d$ be compact. Any $\mathcal{C} \subset L^2(K)$ that is relatively compact is called a *regression problem class*.

A Meaningful Notion of Approximation

Definition

Let $K \subset \mathbb{R}^d$ be compact. Any $\mathcal{C} \subset L^2(K)$ that is relatively compact is called a *regression problem class*.

Definition

Let $\sigma : \mathbb{R} \to \mathbb{R}$ be an activation function and denote $\mathcal{NN}_{M,\sigma,R}$ the set of neural networks with activation function σ and at most M coefficients which are all bounded by R.

A Meaningful Notion of Approximation

Definition

Let $K \subset \mathbb{R}^d$ be compact. Any $\mathcal{C} \subset L^2(K)$ that is relatively compact is called a *regression problem class*.

Definition

Let $\sigma : \mathbb{R} \to \mathbb{R}$ be an activation function and denote $\mathcal{NN}_{M,\sigma,R}$ the set of neural networks with activation function σ and at most M coefficients which are all bounded by R.

Definition [Bölcskei-G-Kutyniok-Petersen (2017)]

A regression problem class $\mathcal{C}$ has effective approximation rate γ if there exists a constant $C > 0$ and a polynomial π with

$$\sup_{f \in \mathcal{C}} \inf_{\Phi \in \mathcal{NN}_{M,\sigma,\pi(M)}} \|f - \Phi\|_{L^2(K)} \leq C \cdot M^{-\gamma}.$$

Why is this Meaningful?

Effective approximation rate implies efficient storage!

Why is this Meaningful?

Effective approximation rate implies efficient storage!

Theorem ([Bölcskei-G-Kutyniok-Petersen (2017)] Informal Version)

If $\mathcal{C}$ has effective approximation rate γ, then every $f \in \mathcal{C}$ can be approximated to within error $N^{-\gamma}$ by a neural network whose topology and quantized coefficients can be stored with $C \cdot N \cdot \log(N)$ bits.

Why is this Meaningful?

Effective approximation rate implies efficient storage!

Theorem ([Bölcskei-G-Kutyniok-Petersen (2017)] Informal Version)

If $\mathcal{C}$ has effective approximation rate γ, then every $f \in \mathcal{C}$ can be approximated to within error $N^{-\gamma}$ by a neural network whose topology and quantized coefficients can be stored with $C \cdot N \cdot \log(N)$ bits.

Effective approximation rate correlates with complexity of regression problem class!

Why is this Meaningful?

Effective approximation rate implies efficient storage!

Theorem ([Bölcskei-G-Kutyniok-Petersen (2017)] Informal Version)

If $\mathcal{C}$ has effective approximation rate γ, then every $f \in \mathcal{C}$ can be approximated to within error $N^{-\gamma}$ by a neural network whose topology and quantized coefficients can be stored with $C \cdot N \cdot \log(N)$ bits.

Effective approximation rate correlates with complexity of regression problem class!

Theorem ([Bölcskei-G-Kutyniok-Petersen (2017)] Informal Version)

Let $s(\mathcal{C})$ be the Kolmogorov entropy of $\mathcal{C}$ (a measure of complexity that can be computed). Then $\gamma \leq 1/s(\mathcal{C})$. In particular, this scaling between accuracy and complexity has to be obeyed by all learning algorithms!

...or more formally...

Theorem [Bölcskei-G-Kutyniok-Petersen (2017)]

Consider *any* learning algorithm **Learn** : $(0,1) \times \mathcal{C} \to \mathcal{NN}$ ($\mathcal{NN}$ being the class of neural networks) which satisfies

$$\sup_{F \in \mathcal{C}} \|F - \textbf{Learn}(\epsilon, F)\| \leq \epsilon$$

and the weights of **Learn**(ϵ, F) at most growing polynomially in ϵ^{-1}.

...or more formally...

Theorem [Bölcskei-G-Kutyniok-Petersen (2017)]

Consider *any* learning algorithm **Learn** : $(0,1) \times \mathcal{C} \to \mathcal{NN}$ ($\mathcal{NN}$ being the class of neural networks) which satisfies

$$\sup_{F \in \mathcal{C}} \|F - \mathbf{Learn}(\epsilon, F)\| \leq \epsilon$$

and the weights of $\mathbf{Learn}(\epsilon, F)$ at most growing polynomially in ϵ^{-1}. Then, for each $\gamma < \gamma^*(\mathcal{C})$ there exists a regression problem $F \in \mathcal{C}$ with

$$\sup_{\epsilon \in (0,1)} \epsilon^{\gamma} \cdot \text{size}(\mathbf{Learn}(\epsilon, F)) = \infty.$$

...or more formally...

Theorem [Bölcskei-G-Kutyniok-Petersen (2017)]

Consider *any* learning algorithm **Learn** : $(0,1) \times \mathcal{C} \to \mathcal{NN}$ ($\mathcal{NN}$ being the class of neural networks) which satisfies

$$\sup_{F \in \mathcal{C}} \|F - \mathbf{Learn}(\epsilon, F)\| \leq \epsilon$$

and the weights of $\mathbf{Learn}(\epsilon, F)$ at most growing polynomially in ϵ^{-1}. Then, for each $\gamma < \gamma^*(\mathcal{C})$ there exists a regression problem $F \in \mathcal{C}$ with

$$\sup_{\epsilon \in (0,1)} \epsilon^{\gamma} \cdot \mathrm{size}(\mathbf{Learn}(\epsilon, F)) = \infty.$$

Coefficients in Pinkus' network are so large that they cannot be stored!

...we can finally ask a meaningful question

Definition [Bölcskei-G-Kutyniok-Petersen (2017)]

Neural networks are optimal for a regression problem class $\mathcal{C}$ if $\mathcal{C}$ has effective approximation rate γ for all $\gamma < 1/s(\mathcal{C})$.

...we can finally ask a meaningful question

Definition [Bölcskei-G-Kutyniok-Petersen (2017)]

Neural networks are optimal for a regression problem class $\mathcal{C}$ if $\mathcal{C}$ has effective approximation rate γ for all $\gamma < 1/s(\mathcal{C})$.

Questions

...we can finally ask a meaningful question

Definition [Bölcskei-G-Kutyniok-Petersen (2017)]

Neural networks are optimal for a regression problem class $\mathcal{C}$ if $\mathcal{C}$ has effective approximation rate γ for all $\gamma < 1/s(\mathcal{C})$.

Questions

- Characterize the regression problem classes for which neural networks are optimal!

...we can finally ask a meaningful question

Definition [Bölcskei-G-Kutyniok-Petersen (2017)]

Neural networks are optimal for a regression problem class $\mathcal{C}$ if $\mathcal{C}$ has effective approximation rate γ for all $\gamma < 1/s(\mathcal{C})$.

Questions

- Characterize the regression problem classes for which neural networks are optimal!
- What architectures (deep, shallow,...) are good for which regression problem classes?

What is Known

Theorem ([Bölcskei-G-Kutyniok-Petersen (2017)] Informal Version)

Let $\mathcal{C}$ be a ball of any classical approximation space (for example Sobolev, Besov, Shearlet, Kernel Approximation Space, piecewise smooth functions on submanifolds, ...). Then neural networks are optimal for $\mathcal{C}$.

What is Known

Theorem ([Bölcskei-G-Kutyniok-Petersen (2017)] Informal Version)

Let $\mathcal{C}$ be a ball of any classical approximation space (for example Sobolev, Besov, Shearlet, Kernel Approximation Space, piecewise smooth functions on submanifolds, ...). Then neural networks are optimal for $\mathcal{C}$.

- This means neural networks are as good as all classical 'linear' methods combined!

What is Known

Theorem ([Bölcskei-G-Kutyniok-Petersen (2017)] Informal Version)

Let $\mathcal{C}$ be a ball of any classical approximation space (for example Sobolev, Besov, Shearlet, Kernel Approximation Space, piecewise smooth functions on submanifolds, ...). Then neural networks are optimal for $\mathcal{C}$.

- This means neural networks are as good as all classical 'linear' methods combined!
- They are even better: Result remains true if signal class is defined on a submanifold and/or is warped by a smooth diffeomorphism.

A Detour: Sparse Coding

- Given a dictionary $\mathcal{D} = (\varphi_i)_{i \in \mathbb{N}} \subset L^2(\Gamma)$, approximate every $F \in \mathcal{C}$ by *optimally sparse* linear combinations of $\mathcal{D}$, i.e.

$$\sum_{i \in J} c_i \varphi_i$$

with $|J|$ as small as possible.

A Detour: Sparse Coding

- Given a dictionary $\mathcal{D} = (\varphi_i)_{i\in\mathbb{N}} \subset L^2(\Gamma)$, approximate every $F \in \mathcal{C}$ by *optimally sparse* linear combinations of $\mathcal{D}$, i.e.

$$\sum_{i\in J} c_i \varphi_i$$

  with $|J|$ as small as possible.
- For most known $\mathcal{C}$ there exists a specific *optimal* dictionary that achieves the optimal tradeoff between sparsity ($\sim$ codelength!) and approximation error, as dictated by $\gamma^*(\mathcal{C})$.

A Detour: Sparse Coding

- Given a dictionary $\mathcal{D} = (\varphi_i)_{i \in \mathbb{N}} \subset L^2(\Gamma)$, approximate every $F \in \mathcal{C}$ by *optimally sparse* linear combinations of $\mathcal{D}$, i.e.

$$\sum_{i \in J} c_i \varphi_i$$

  with $|J|$ as small as possible.
- For most known $\mathcal{C}$ there exists a specific *optimal* dictionary that achieves the optimal tradeoff between sparsity ($\sim$ codelength!) and approximation error, as dictated by $\gamma^*(\mathcal{C})$.
- Examples: Textures $\leftrightarrow$ Gabor frames (JPEG), point singularities $\leftrightarrow$ wavelets (JPEG2000), line/hyperplane singularities $\leftrightarrow$ ridgelets, curved/hypersurface singularities $\leftrightarrow$ (α-)curvelets.

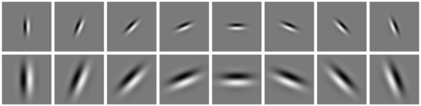
A Detour: Sparse Coding

- Given a dictionary $\mathcal{D} = (\varphi_i)_{i\in\mathbb{N}} \subset L^2(\Gamma)$, approximate every $F \in \mathcal{C}$ by *optimally sparse* linear combinations of $\mathcal{D}$, i.e.

$$\sum_{i\in J} c_i\varphi_i$$

  with $|J|$ as small as possible.

- For most known $\mathcal{C}$ there exists a specific *optimal* dictionary that achieves the optimal tradeoff between sparsity ($\sim$ codelength!) and approximation error, as dictated by $\gamma^*(\mathcal{C})$.
- Examples: Textures $\leftrightarrow$ Gabor frames (JPEG), point singularities $\leftrightarrow$ wavelets (JPEG2000), line/hyperplane singularities $\leftrightarrow$ ridgelets, curved/hypersurface singularities $\leftrightarrow$ (α-)curvelets.



Transferring Optimality

Definition [Bölcskei-G-Kutyniok-Petersen (2017)]

A dictionary $\mathcal{D} = (\varphi_i)_{i \in \mathbb{N}}$ is representable by neural networks if for all $\epsilon > 0$ and $i \in \mathbb{N}$ there is $\Phi_{i,\epsilon} \in \mathcal{NN}$ with $O(1)$ nonzero weights, growing at most polynomially in $i \cdot \epsilon^{-1}$ with

$$\|\varphi_i - \Phi_{i,\epsilon}\| \leq \epsilon.$$

Transferring Optimality

Definition [Bölcskei-G-Kutyniok-Petersen (2017)]

A dictionary $\mathcal{D} = (\varphi_i)_{i\in\mathbb{N}}$ is representable by neural networks if for all $\epsilon > 0$ and $i \in \mathbb{N}$ there is $\Phi_{i,\epsilon} \in \mathcal{NN}$ with $O(1)$ nonzero weights, growing at most polynomially in $i \cdot \epsilon^{-1}$ with

$$\|\varphi_i - \Phi_{i,\epsilon}\| \leq \epsilon.$$

Theorem [Bölcskei-G-Kutyniok-Petersen (2017)]

Suppose that the dictionary $\mathcal{D}$ is optimal for the class $\mathcal{C}$ and suppose that $\mathcal{D}$ is representable by neural networks. Then neural networks are optimal for the regression class $\mathcal{C}$.

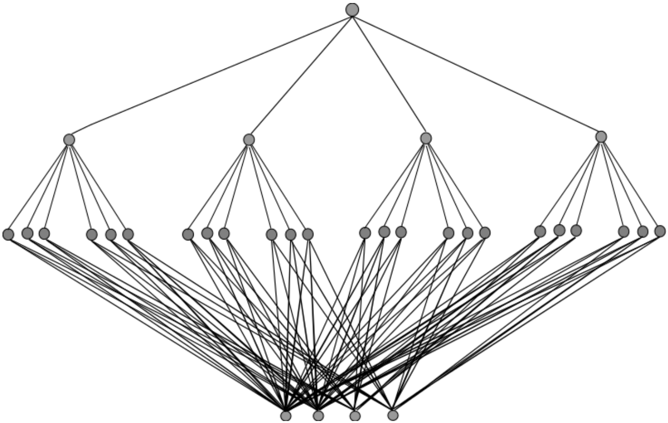
Idea

Idea

Suppose that $\sum_{i=1}^{4} c_i \varphi$ is a sparse approximation of F in $\mathcal{D}$.

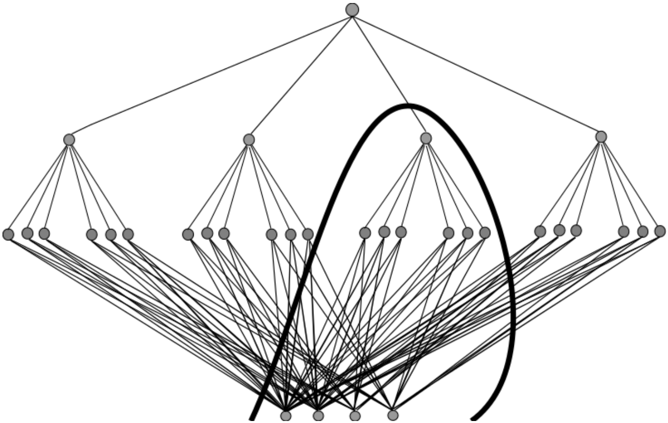
Idea

Suppose that $\sum_{i=1}^{4} c_i \varphi$ is a sparse approximation of F in $\mathcal{D}$.



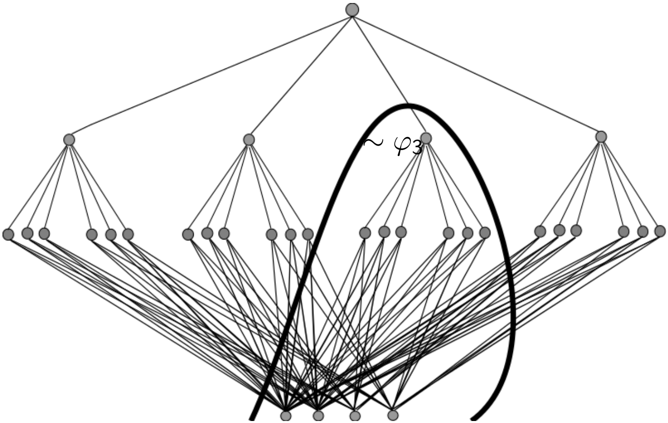
Idea

Suppose that $\sum_{i=1}^{4} c_i \varphi_i$ is a sparse approximation of $F \in \mathcal{D}$.



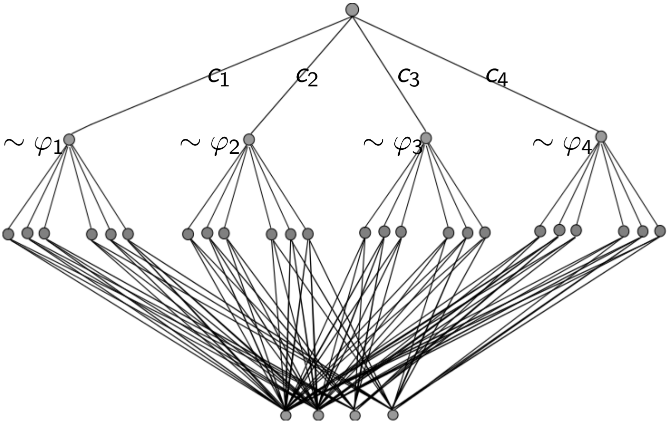
Idea

Suppose that $\sum_{i=1}^{4} c_i \varphi_i$ is a sparse approximation of $F \in \mathcal{D}$.



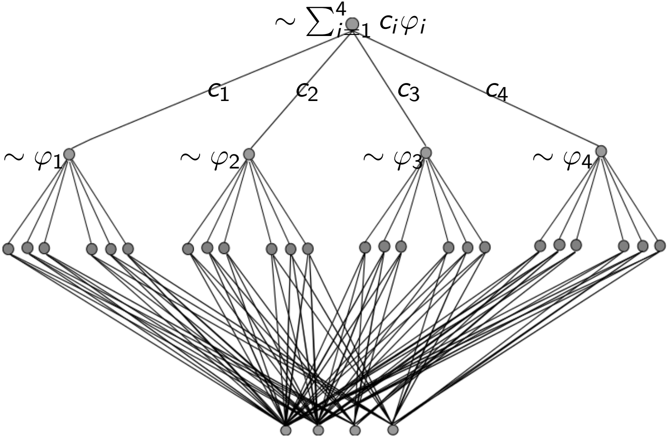
Idea

Suppose that $\sum_{i=1}^{4} c_i \varphi_i$ is a sparse approximation of $F \in \mathcal{D}$.



Idea

Suppose that $\sum_{i=1}^{4} c_i \varphi_i$ is a sparse approximation of $F \in \mathcal{D}$.



The Good News

Theorem ([Bölcskei-G-Kutyniok-Petersen (2017)] informal version)

All known (affine) dictionaries are representable by (shallow) neural networks (under weak assumptions on the activation function).

The Good News

Theorem ([Bölcskei-G-Kutyniok-Petersen (2017)] informal version)

All known (affine) dictionaries are representable by (shallow) neural networks (under weak assumptions on the activation function).

 Neural network regression is as powerful as regression with all known dictionaries, combined and in particular optimal for all corresponding problem classes!

Two Big Questions

Two Big Questions

Our approximation result does not require the approximating network to be very deep!

Two Big Questions

Our approximation result does not require the approximating network to be very deep!

Big Question 1

What do we gain by using *deep* networks?

Two Big Questions

Our approximation result does not require the approximating network to be very deep!

Big Question 1

What do we gain by using *deep* networks?

Our result implies that classical approximation results can be emulated by neural networks. However, this only partially explains their success in approximating high-dimensional regression/classification problems!

Two Big Questions

Our approximation result does not require the approximating network to be very deep!

Big Question 1

What do we gain by using *deep* networks?

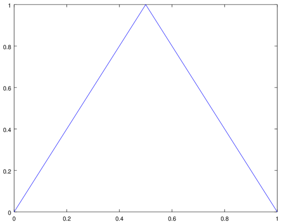
Our result implies that classical approximation results can be emulated by neural networks. However, this only partially explains their success in approximating high-dimensional regression/classification problems!

Big Question 2

Why are neural networks so good at approximating high-dimensional functions?

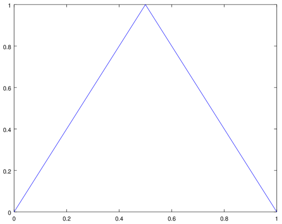
A Simple Example

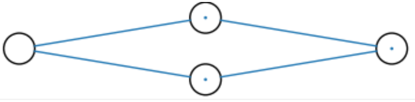
A Simple Example



$$g(x) = \begin{cases} 2x & 0 < x < 1/2 \\ 2(1-x) & 1/2 \leq x < 1 \\ 0 & \text{else} \end{cases}$$

A Simple Example

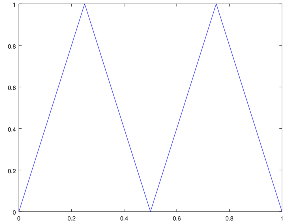




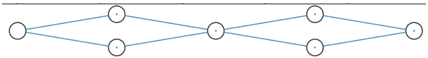
$$g(x) = \begin{cases} 2x & 0 < x < 1/2 \\ 2(1-x) & 1/2 \leq x < 1 \\ 0 & \text{else} \end{cases}$$

$$g(x) = ReLU\left(2 \cdot ReLU(x) - 4 \cdot ReLU\left(x - \frac{1}{2}\right)\right)$$

A Simple Example

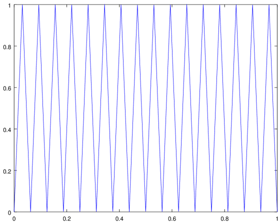


$g \circ g(x)$



"deep" ReLU net

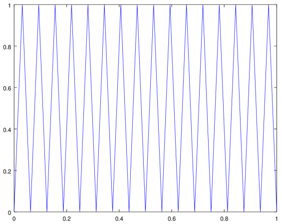
A Simple Example



$g \circ g \circ g \circ g \circ g(x)$

"deep" ReLU net

A Simple Example

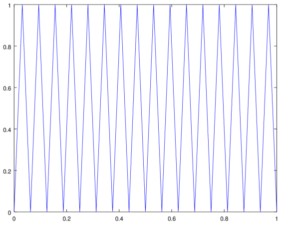


$g \circ g \circ g \circ g \circ g(x)$

"deep" ReLU net

High-frequency oscillations can be represented by deep neural networks!

A Simple Example



$g \circ g \circ g \circ g \circ g(x)$

"deep" ReLU net

High-frequency oscillations can be represented by deep neural networks!

Exercise (see also [Telgarsky (2015)])

Shallow networks cannot represent high-frequency oscillations: one needs $\gtrsim j$ layers to capture frequencies oscillating at scale 2^{-j}

Related Results

[Montufar (2014)] shows that the number of linear regions in ReLU networks grows exponentially for deep networks and only polynomially for shallow networks

Related Results

[Montufar (2014)] shows that the number of linear regions in ReLU networks grows exponentially for deep networks and only polynomially for shallow networks

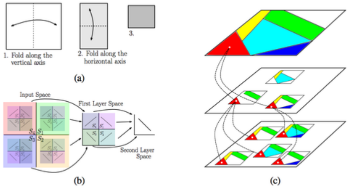


Figure 2: (a) Space folding of 2-D Euclidean space along the two coordinate axes. (b) An illustration of how the top-level partitioning (on the right) is replicated to the original input space (left). (c) Identification of regions across the layers of a deep model.

Related Results

[Montufar (2014)] shows that the number of linear regions in ReLU networks grows exponentially for deep networks and only polynomially for shallow networks

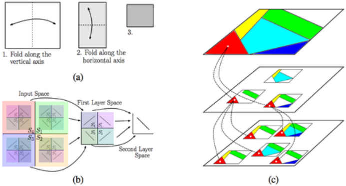


Figure 2: (a) Space folding of 2-D Euclidean space along the two coordinate axes. (b) An illustration of how the top-level partitioning (on the right) is replicated to the original input space (left). (c) Identification of regions across the layers of a deep model.

- [Bianchini, Scarselli (2014)] showed analogous results for Betti numbers of level sets.

Related Results

[Montufar (2014)] shows that the number of linear regions in ReLU networks grows exponentially for deep networks and only polynomially for shallow networks

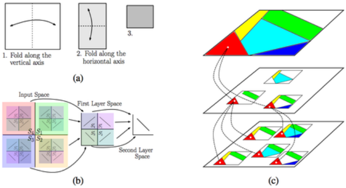


Figure 2: (a) Space folding of 2-D Euclidean space along the two coordinate axes. (b) An illustration of how the top-level partitioning (on the right) is replicated to the original input space (left). (c) Identification of regions across the layers of a deep model.

- [Bianchini, Scarselli (2014)] showed analogous results for Betti numbers of level sets.

It seems that deep networks are better at approximating highly oscillating textures with fractal structure, but no precise characterization yet!

High-Dimensional PDEs

High-Dimensional PDEs

Consider for example the Black-Scholes equation where we want to compute the prize $V(0, x)$ of an option depending on a financial portfolio $x \in \mathbb{R}^d$ subject to

$$V_t + div(A(x) \cdot \nabla V) + b(x) \cdot \nabla V - f(V, \nabla V) = 0, \quad V(T, x) = g(x),$$

where g models the prize of of the option g at terminal time T.

High-Dimensional PDEs

Consider for example the Black-Scholes equation where we want to compute the prize $V(0,x)$ of an option depending on a financial portfolio $x \in \mathbb{R}^d$ subject to

$$V_t + div(A(x)\cdot\nabla V) + b(x)\cdot\nabla V - f(V,\nabla V) = 0, \quad V(T,x) = g(x),$$

where g models the prize of of the option g at terminal time T. Typical options (maximum call) are of the form

$$g(x) = \max(\max_{i=1}^{d} x_i - X, 0).$$

Deep DBSE Solver

[E, Han, Jentzen (2017)] solve high-dimensional ($>$100d) parabolic PDEs using deep neural networks, essentially using a vanilla tensorflow implementation and achieving efficiency beyond the current state-of-the-art!

Deep DBSE Solver

[E, Han, Jentzen (2017)] solve high-dimensional ($>$100d) parabolic PDEs using deep neural networks, essentially using a vanilla tensorflow implementation and achieving efficiency beyond the current state-of-the-art!

Question

Why does this work so well? Can we establish a neural network approximation theory for solutions of PDEs?

A Hint

Observation

The maximum call option can be expressed by neural networks with $\sim \log_2(d)$ and nodes!

A Hint

Observation

The maximum call option can be expressed by neural networks with $\sim \log_2(d)$ and nodes!

Proof:

$$\max(x_1, x_2, x_3, x_4) = \max(\max(x_1, x_2), \max(x_3, x_4))$$

and

$$\max(x, y) = x + ReLU(y - x).$$

A Hint

Observation

The maximum call option can be expressed by neural networks with $\sim \log_2(d)$ and nodes!

Proof:

$$\max(x_1, x_2, x_3, x_4) = \max(\max(x_1, x_2), \max(x_3, x_4))$$

and

$$\max(x, y) = x + ReLU(y - x).$$

Depth is Necessary!

One can show that high-dimensional options cannot be approximated well by shallow networks (related methods for circuits in [Hastad (1986)] and [Kane-Williams (2016)])!

Related Work

[Mhaskar, Liao, Poggio (2014)] consider *compositional functions* for example of the form

$$f(x_1, \ldots, x_8) = \\ h_3\left(h_{21}\left(h_{11}(x_1, x_2), h_{12}(x_3, x_4)\right), h_{22}\left(h_{13}(x_5, x_6), h_{14}(x_7, x_8)\right)\right),$$

with h_{ij} smooth.

Summary

What we Know

Deep Networks are better than shallow networks if the function to be approximated is

Summary

What we Know

Deep Networks are better than shallow networks if the function to be approximated is

- fractal, texture-like

Summary

What we Know

Deep Networks are better than shallow networks if the function to be approximated is

- fractal, texture-like
- high-dimensional with compositional structure.

Summary

What we Know

Deep Networks are better than shallow networks if the function to be approximated is

- fractal, texture-like
- high-dimensional with compositional structure.

Open Questions

Summary

What we Know

Deep Networks are better than shallow networks if the function to be approximated is

- fractal, texture-like
- high-dimensional with compositional structure.

Open Questions

- Build general framework/theory!

Summary

What we Know

Deep Networks are better than shallow networks if the function to be approximated is

- fractal, texture-like
- high-dimensional with compositional structure.

Open Questions

- Build general framework/theory!
- Which high-dimensional problems have compositional structure?

Summary

What we Know

Deep Networks are better than shallow networks if the function to be approximated is

- fractal, texture-like
- high-dimensional with compositional structure.

Open Questions

- Build general framework/theory!
- Which high-dimensional problems have compositional structure?
- PDE Regularity theory for neural networks?

Summary

What we Know

Deep Networks are better than shallow networks if the function to be approximated is

- fractal, texture-like
- high-dimensional with compositional structure.

Open Questions

- Build general framework/theory!
- Which high-dimensional problems have compositional structure?
- PDE Regularity theory for neural networks?
- How to approximate a large neural network by a smaller one?

Summary

What we Know

Deep Networks are better than shallow networks if the function to be approximated is

- fractal, texture-like
- high-dimensional with compositional structure.

Open Questions

- Build general framework/theory!
- Which high-dimensional problems have compositional structure?
- PDE Regularity theory for neural networks?
- How to approximate a large neural network by a smaller one?
- How about convolutional networks?

Summary

What we Know

Deep Networks are better than shallow networks if the function to be approximated is

- fractal, texture-like
- high-dimensional with compositional structure.

Open Questions

- Build general framework/theory!
- Which high-dimensional problems have compositional structure?
- PDE Regularity theory for neural networks?
- How to approximate a large neural network by a smaller one?
- How about convolutional networks?
- Why do neural networks with SGD generalize despite their huge capacity?

2. Stochastic Optimization Algorithms

Vanilla SGD

Vanilla SGD

- Let f_θ be a neural network with parameters θ.

Vanilla SGD

- Let f_θ be a neural network with parameters θ.
- Empirical Risk minimization seeks the minimizer θ_* of

$$\theta \mapsto \frac{1}{n} \sum_{i=1}^{m} |f_\theta(x_i) - y_i|^2.$$

Vanilla SGD

- Let f_θ be a neural network with parameters θ.
- Empirical Risk minimization seeks the minimizer θ_* of

$$\theta \mapsto \frac{1}{n} \sum_{i=1}^{m} |f_\theta(x_i) - y_i|^2.$$

- SGD computes updates as

$$\theta_{n+1} := \theta_n - \nu_{n+1} \nabla_\theta \left(|f_\theta(x_{i_n}) - y_{i_n}|^2 \right),$$

where ν_{n+1} is the *learning rate* and the indices i_n are chosen uniformly and independently.

Vanilla SGD

- Let f_θ be a neural network with parameters θ.
- Empirical Risk minimization seeks the minimizer θ_* of

$$\theta \mapsto \frac{1}{n}\sum_{i=1}^{m} |f_\theta(x_i) - y_i|^2.$$

- SGD computes updates as

$$\theta_{n+1} := \theta_n - \nu_{n+1} \nabla_\theta \left(|f_\theta(x_{i_n}) - y_{i_n}|^2 \right),$$

  where ν_{n+1} is the *learning rate* and the indices i_n are chosen uniformly and independently.

🙁 In general nothing can be said about global convergence.

What is Known

What is Known

Strong Convergence

For SGD applied to a convex problem we have, for appropriate learning rates

$$\mathbb{E}|\theta_n - \theta_*| \lesssim \frac{1}{n^{1/2-\epsilon}},$$

and this cannot be improved.

What is Known

Strong Convergence

For SGD applied to a convex problem we have, for appropriate learning rates

$$\mathbb{E}|\theta_n - \theta_*| \lesssim \frac{1}{n^{1/2-\epsilon}},$$

and this cannot be improved.

Weak Convergence [G-Jentzen (2017)]

For SGD applied to a convex problem we have, for appropriate learning rates and every C^2 function ψ

$$|\mathbb{E}\psi(\theta_n) - \psi(\theta_*)| \lesssim \frac{1}{n^{1-\epsilon}},$$

and this cannot be improved.

What is Known

Strong Convergence

For SGD applied to a convex problem we have, for appropriate learning rates

$$\mathbb{E}|\theta_n - \theta_*| \lesssim \frac{1}{n^{1/2-\epsilon}},$$

and this cannot be improved.

Weak Convergence [G-Jentzen (2017)]

For SGD applied to a convex problem we have, for appropriate learning rates and every C^2 function ψ

$$|\mathbb{E}\psi(\theta_n) - \psi(\theta_*)| \lesssim \frac{1}{n^{1-\epsilon}},$$

and this cannot be improved.

Of course neural network ERM is highly non-convex!

Can we Leverage Convexity?

Can we Leverage Convexity?

- Non-convexity stems from the parameterization $\Pi(\theta) := f_\theta$.

Can we Leverage Convexity?

- Non-convexity stems from the parameterization $\Pi(\theta) := f_\theta$.
- The function $F(f) := \frac{1}{m} \sum_{i=1}^{m} (f(x_i) - y_i)^2$ *is* convex.

Can we Leverage Convexity?

- Non-convexity stems from the parameterization $\Pi(\theta) := f_\theta$.
- The function $F(f) := \frac{1}{m}\sum_{i=1}^{m}(f(x_i) - y_i)^2$ *is* convex.
- Now suppose for a moment that the set $\mathcal{NN}$ of neural networks over which we optimize is convex.

Can we Leverage Convexity?

- Non-convexity stems from the parameterization $\Pi(\theta) := f_\theta$.
- The function $F(f) := \frac{1}{m}\sum_{i=1}^{m}(f(x_i) - y_i)^2$ *is* convex.
- Now suppose for a moment that the set $\mathcal{N}\mathcal{N}$ of neural networks over which we optimize is convex.
- Let θ_* be a local minimum. Then (if Π is nice) f_{θ_*} is a local minimum of the convex function F over the convex set $\mathcal{N}\mathcal{N}$.

Can we Leverage Convexity?

- Non-convexity stems from the parameterization $\Pi(\theta) := f_\theta$.
- The function $F(f) := \frac{1}{m}\sum_{i=1}^{m}(f(x_i) - y_i)^2$ *is* convex.
- Now suppose for a moment that the set $\mathcal{N}\mathcal{N}$ of neural networks over which we optimize is convex.
- Let θ_* be a local minimum. Then (if Π is nice) f_{θ_*} is a local minimum of the convex function F over the convex set $\mathcal{N}\mathcal{N}$.
- Consequently, f_{θ_*} is a global minimum of F!

Can we Leverage Convexity?

- Non-convexity stems from the parameterization $\Pi(\theta) := f_\theta$.
- The function $F(f) := \frac{1}{m}\sum_{i=1}^{m}(f(x_i) - y_i)^2$ *is* convex.
- Now suppose for a moment that the set $\mathcal{NN}$ of neural networks over which we optimize is convex.
- Let θ_* be a local minimum. Then (if Π is nice) f_{θ_*} is a local minimum of the convex function F over the convex set $\mathcal{NN}$.
- Consequently, f_{θ_*} is a global minimum of F!

Of course the hypothesis class $\mathcal{NN}$ is not convex and we don't know anything about Π!

Can we Leverage Convexity?

- Non-convexity stems from the parameterization $\Pi(\theta) := f_\theta$.
- The function $F(f) := \frac{1}{m}\sum_{i=1}^{m}(f(x_i) - y_i)^2$ *is* convex.
- Now suppose for a moment that the set $\mathcal{NN}$ of neural networks over which we optimize is convex.
- Let θ_* be a local minimum. Then (if Π is nice) f_{θ_*} is a local minimum of the convex function F over the convex set $\mathcal{NN}$.
- Consequently, f_{θ_*} is a global minimum of F!

🙁 Of course the hypothesis class $\mathcal{NN}$ is not convex and we don't know anything about Π!

Question

Is every local minimum of the neural network ERM problem also a global minimum?

What is Known

What is Known

Theorem [Kawaguchi (2016)]

The answer is 'yes' for linear activation functions.

What is Known

Theorem [Kawaguchi (2016)]

The answer is 'yes' for linear activation functions.

Observation

If $\mathcal{NN}$ is nearly convex (meaning that convex combinations of neural networks can be very well approximated by neural networks) then every local minimum is almost a global minimum.

What is Known

Theorem [Kawaguchi (2016)]

The answer is 'yes' for linear activation functions.

Observation

If $\mathcal{NN}$ is nearly convex (meaning that convex combinations of neural networks can be very well approximated by neural networks) then every local minimum is almost a global minimum.

Question

How well can convex combinations of neural networks be approximated by neural networks of the same size? Does "almost convexity" improve with the size?

Understanding the Parameterization Π

Question

How are the parameters θ and the function $\Pi(\theta)$ related?

Understanding the Parameterization Π

Question

How are the parameters θ and the function $\Pi(\theta)$ related?

Theorem [Fefferman (1992)]

For a sigmoidal activation function, the parameters (i.e., the architecture *and* the coefficients) are uniquely determined by f_θ, up to trivial symmetries and for almost all networks.

Understanding the Parameterization Π

Question

How are the parameters θ and the function $\Pi(\theta)$ related?

Theorem [Fefferman (1992)]

For a sigmoidal activation function, the parameters (i.e., the architecture *and* the coefficients) are uniquely determined by f_θ, up to trivial symmetries and for almost all networks.

Proof is deep and beautiful.

Understanding the Parameterization Π

Question

How are the parameters θ and the function $\Pi(\theta)$ related?

Theorem [Fefferman (1992)]

For a sigmoidal activation function, the parameters (i.e., the architecture *and* the coefficients) are uniquely determined by f_θ, up to trivial symmetries and for almost all networks.

Proof is deep and beautiful. But it is doubtful that it can be cast into a stable algorithm.

Understanding the Parameterization Π

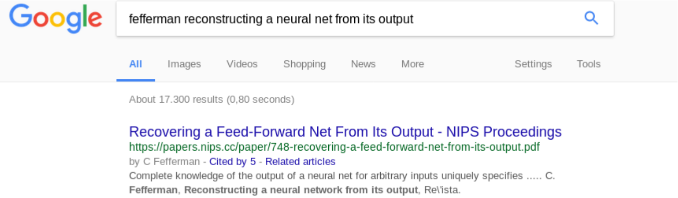
Question

How are the parameters θ and the function $\Pi(\theta)$ related?

Theorem [Fefferman (1992)]

For a sigmoidal activation function, the parameters (i.e., the architecture *and* the coefficients) are uniquely determined by f_θ, up to trivial symmetries and for almost all networks.

Proof is deep and beautiful. But it is doubtful that it can be cast into a stable algorithm.



Understanding the Parameterization Π

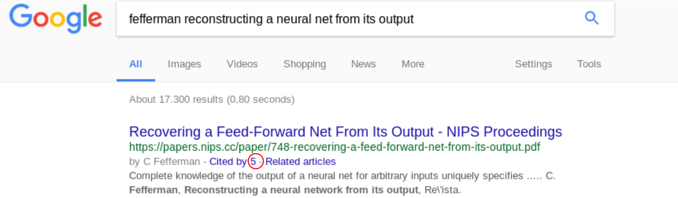
Question

How are the parameters θ and the function $\Pi(\theta)$ related?

Theorem [Fefferman (1992)]

For a sigmoidal activation function, the parameters (i.e., the architecture *and* the coefficients) are uniquely determined by f_θ, up to trivial symmetries and for almost all networks.

Proof is deep and beautiful. But it is doubtful that it can be cast into a stable algorithm.



Summary

- Deep Learning is a great field of research for mathematicians:

Summary

- Deep Learning is a great field of research for mathematicians:
    - it is highly relevant, exciting and fun

Summary

- Deep Learning is a great field of research for mathematicians:
  - it is highly relevant, exciting and fun
  - its problems involve deep mathematics

Summary

- Deep Learning is a great field of research for mathematicians:
  - it is highly relevant, exciting and fun
  - its problems involve deep mathematics
  - most problems are completely open

Summary

- Deep Learning is a great field of research for mathematicians:
  - it is highly relevant, exciting and fun
  - its problems involve deep mathematics
  - most problems are completely open
  - it is interdisciplinary

Summary

- Deep Learning is a great field of research for mathematicians:
  - it is highly relevant, exciting and fun
  - its problems involve deep mathematics
  - most problems are completely open
  - it is interdisciplinary
- Possible contributions include:

Summary

- Deep Learning is a great field of research for mathematicians:
    - it is highly relevant, exciting and fun
    - its problems involve deep mathematics
    - most problems are completely open
    - it is interdisciplinary
- Possible contributions include:
    - more informed design choices for network architectures

Summary

- Deep Learning is a great field of research for mathematicians:
  - it is highly relevant, exciting and fun
  - its problems involve deep mathematics
  - most problems are completely open
  - it is interdisciplinary
- Possible contributions include:
  - more informed design choices for network architectures
  - new algorithmic paradigms (for example optimal network compression or better optimization algorithms)

Summary

- Deep Learning is a great field of research for mathematicians:
  - it is highly relevant, exciting and fun
  - its problems involve deep mathematics
  - most problems are completely open
  - it is interdisciplinary
- Possible contributions include:
  - more informed design choices for network architectures
  - new algorithmic paradigms (for example optimal network compression or better optimization algorithms)
- interaction between different fields will be crucial!

Thank You!

Questions?