Efficient Algorithms for Deep Learning

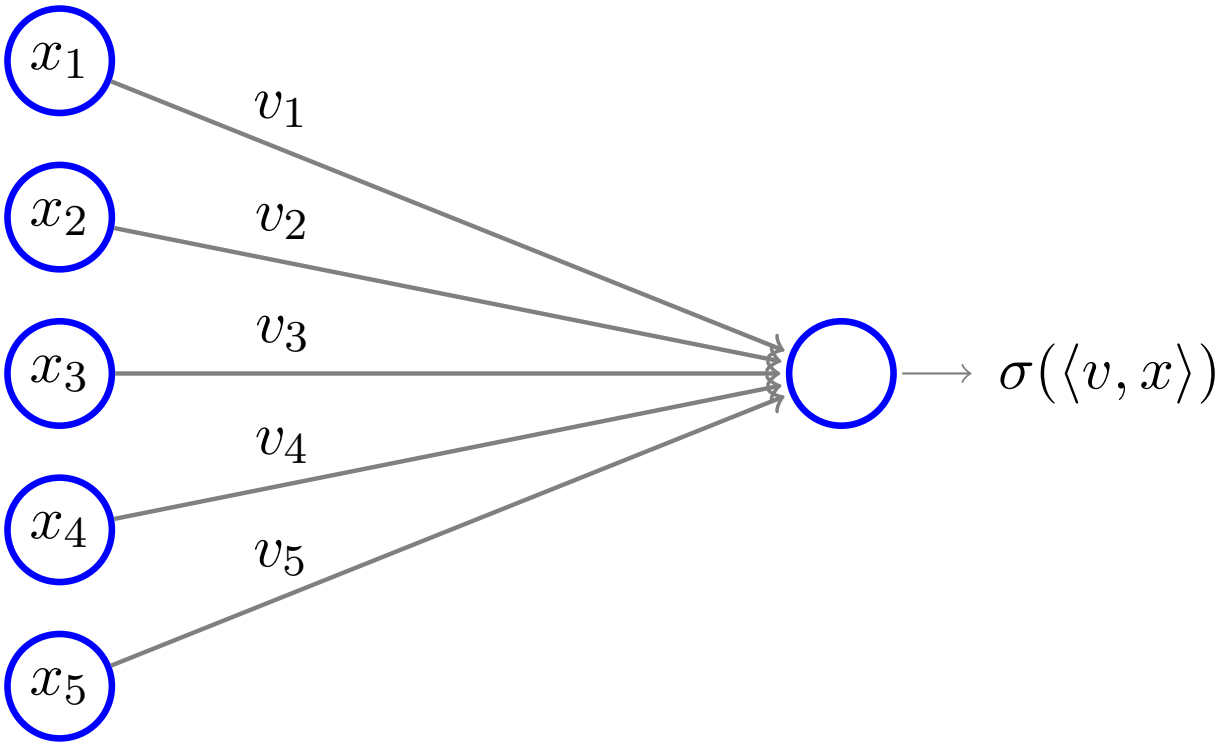
Shai Shalev-Shwartz

School of CS and Engineering,
The Hebrew University of Jerusalem

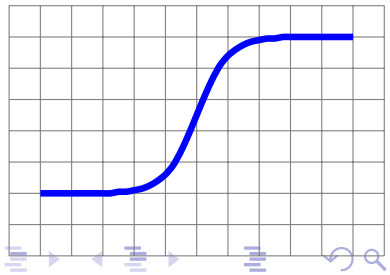
"Simons Institute",
Berkeley 2013

Neural Networks

- A single neuron with activation function $\sigma : \mathbb{R} \to \mathbb{R}$

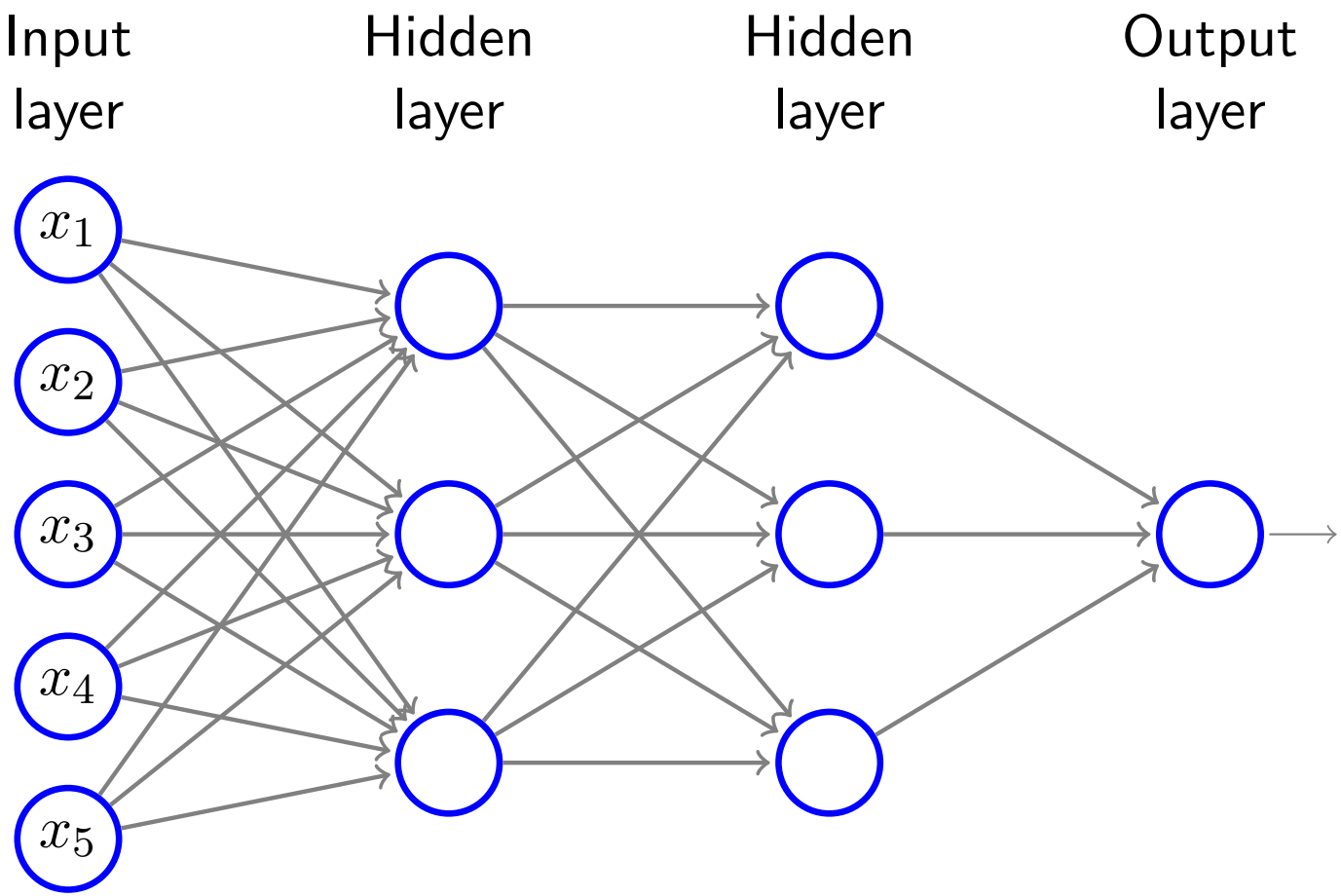


- Usually, σ is taken to be a sigmoidal function



Neural Networks

- A multilayer neural network of depth 3 and size 6



Why Deep Neural Networks are Great?

- Because "A" uses it to do "B"

Why Deep Neural Networks are Great?

- Because "A" uses it to do "B"
- Classic explanation: Neural Networks are *universal approximators* — every Lipschitz function $f : [-1, 1]^d \to [-1, 1]$ can be approximated by a neural network

Why Deep Neural Networks are Great?

- Because "A" uses it to do "B"
- Classic explanation: Neural Networks are *universal approximators* — every Lipschitz function $f : [-1, 1]^d \to [-1, 1]$ can be approximated by a neural network
- Not convincing because
  - It can be shown that the size of the network must be exponential in d, so why should we care about such large networks ?
  - Many other universal approximators exist (nearest neighbor, boosting with decision stumps, SVM with RBF kernels), so why should we prefer neural networks?

Why Deep Neural Networks are Great? A Statistical Learning Perspective

- Goal: Learn a function $h : \mathcal{X} \to \mathcal{Y}$ based on training examples $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$
- No-Free-Lunch Theorem: For any algorithm A, and any sample size m, there exists a distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$ and a function h^* such that h^* is perfect w.r.t. $\mathcal{D}$ but with high probability over $S \sim \mathcal{D}^m$, the output of A is very bad
- Prior knowledge: We must bias the learner toward "reasonable" functions — hypothesis class $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$
- What should be $\mathcal{H}$?

Why Deep Neural Networks are Great? A Statistical Learning Perspective

- First idea: Let $\mathcal{H}_{++}$ be all functions we can implement in C++ using code length of at most b bits
- With sufficiently large b, $\mathcal{H}_{++}$ contains all functions we would ever want to learn
- Sample complexity of learning $\mathcal{H}_{++}$ to accuracy ϵ is b/ϵ^2
- Learning algorithm is very simple: Empirical Risk Minimization (ERM) — find $h \in \mathcal{H}_{++}$ that has minimal error on S
- End of story ?

Why Deep Neural Networks are Great? A Statistical Learning Perspective

- First idea: Let $\mathcal{H}_{++}$ be all functions we can implement in C++ using code length of at most b bits
- With sufficiently large b, $\mathcal{H}_{++}$ contains all functions we would ever want to learn
- Sample complexity of learning $\mathcal{H}_{++}$ to accuracy ϵ is b/ϵ^2
- Learning algorithm is very simple: Empirical Risk Minimization (ERM) — find $h \in \mathcal{H}_{++}$ that has minimal error on S
- End of story ?
- The computational barrier: But, how do we implement ERM?

Why Deep Neural Networks are Great? A Statistical Learning Perspective

- Second idea: Consider all functions over $\{0,1\}^d$ that can be executed in time at most $T(d)$
- Theorem: The class $\mathcal{H}_{NN}$ of neural networks of depth $O(T(d))$ and size $O(T(d)^2)$ contains all functions that can be executed in time at most $T(d)$
- A great hypothesis class:
  - With sufficiently large network depth and size, we can express all functions we would ever want to learn
  - Sample complexity behaves nicely and is well understood (see Anthony & Bartlett 1999)

Why Deep Neural Networks are Great? A Statistical Learning Perspective

- Second idea: Consider all functions over $\{0,1\}^d$ that can be executed in time at most $T(d)$
- Theorem: The class $\mathcal{H}_{NN}$ of neural networks of depth $O(T(d))$ and size $O(T(d)^2)$ contains all functions that can be executed in time at most $T(d)$
- A great hypothesis class:
  - With sufficiently large network depth and size, we can express all functions we would ever want to learn
  - Sample complexity behaves nicely and is well understood (see Anthony & Bartlett 1999)
- The computational barrier: But, how do we train neural networks ?

Neural Networks — The computational barrier

- It is NP hard to implement ERM for a depth 2 network with $k \geq 3$ hidden layers whose activation function is sigmoidal or sign (Blum and Rivest 1992, Bartlett and Ben-David 2002)
- Current approaches: Back propagation, possibly with unsupervised pre-training and other bells and whistles
- No theoretical guarantees, and often requires manual tweaking

Outline

How to circumvent hardness?

1. Over-specification
   - Extreme over-specification eliminate local (non-global) minima
   - Hardness of improperly learning a two layers network with $k = \omega(1)$ hidden neurons
2. Change the activation function (sum-product networks)
   - An efficient algorithm for learning sum-product networks of depth 2 and small size using over-specification
   - Hardness of learning deep sum-product networks
3. Distributional assumptions
   - Learning of algebraic sets

Circumventing Hardness using Over-specification

- Yann LeCun:
  - Fix a network architecture and generate data according to it
  - Backpropagation fails to recover parameters
  - However, if we enlarge the network size, backpropagation works just fine

Circumventing Hardness using Over-specification

- Yann LeCun:
  - Fix a network architecture and generate data according to it
  - Backpropagation fails to recover parameters
  - However, if we enlarge the network size, backpropagation works just fine
  - Maybe we can efficiently learn neural network using over-specification?

Extremely over-specified Networks have no local (non-global) minima

- Let $X \in \mathbb{R}^{d,m}$ be a data matrix of m examples
- Consider a network with:
  - N internal neurons
  - v be the weights of all but the last layer
  - $F(v; X)$ be evaluations of internal neurons over data matrix X
  - w be weights connecting internal neurons to the output neuron
  - The output of the network is $w^\top F(v; X)$
- Theorem: If $N \geq m$, and under mild conditions on F, the optimization problem $\min_{w,v} \|w^\top F(v; X) - y\|^2$ has no local (non-global) minima

Extremely over-specified Networks have no local (non-global) minima

- Let $X \in \mathbb{R}^{d,m}$ be a data matrix of m examples
- Consider a network with:
  - N internal neurons
  - v be the weights of all but the last layer
  - $F(v; X)$ be evaluations of internal neurons over data matrix X
  - w be weights connecting internal neurons to the output neuron
  - The output of the network is $w^\top F(v; X)$
- Theorem: If $N \geq m$, and under mild conditions on F, the optimization problem $\min_{w,v} \|w^\top F(v; X) - y\|^2$ has no local (non-global) minima
- Proof idea: W.h.p. over perturbation of v, $F(v; X)$ has full rank. For such v, if we're not at global minimum, just by changing w we can decrease the objective

Is over-specification enough ?

- But, such large networks will lead to overfitting
- Maybe there's a clever trick that circumvent overfitting (regularization, dropout, ...) ?

Is over-specification enough ?

- But, such large networks will lead to overfitting
- Maybe there's a clever trick that circumvent overfitting (regularization, dropout, ...) ?
- Theorem (Daniely, Linial, S.) Even if the data is perfectly generated by a neural network of depth 2 and with only $k = \omega(1)$ neurons in the hidden layer, there is no algorithm that can achieve small test error
- Corollary: over-specification alone is not enough for efficient learnability

Proof Idea: Hardness of Improper Learning

- Improper learning: Learner tries to learn some hypothesis $h^* \in \mathcal{H}$ but is not restricted to output a hypothesis from $\mathcal{H}$

Proof Idea: Hardness of Improper Learning

- Improper learning: Learner tries to learn some hypothesis $h^* \in \mathcal{H}$ but is not restricted to output a hypothesis from $\mathcal{H}$
- How to show hardness?

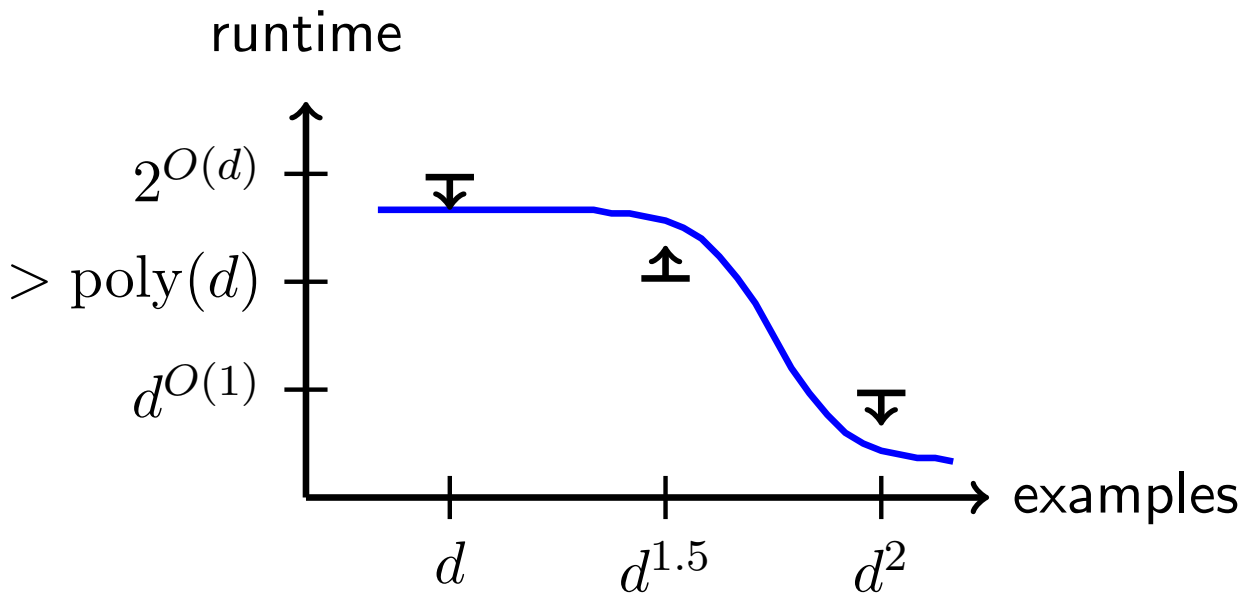
Proof Idea: Hardness of Improper Learning

- Improper learning: Learner tries to learn some hypothesis $h^* \in \mathcal{H}$ but is not restricted to output a hypothesis from $\mathcal{H}$
- How to show hardness?
- Technical novelty: A new method for deriving lower bounds for improper learning
- Technique yields new hardness results for improper learning of:
  - DNFs
    (open problem since Kearns&Valiant'1989)
  - Intersection of $\omega(1)$ halfspaces
    (Klivans&Sherstov'2006 showed hardness for d^c halfspaces)
  - Constant approximation ratio for agnostically learning halfspaces
    (previously, only hardness of exact learning was known)

Computational-Statistical Tradeoffs

- Daniely, Linial, S. To appear in NIPS'13

For agnostically learning halfspaces over 3-sparse vectors:



- Most previous work either rely on upper bounds or deal with synthetic hypothesis classes

Outline

How to circumvent hardness?

1. Over-specification
   - Extreme over-specification eliminate local (non-global) minima
   - Hardness of improperly learning a two layers network with $k = \omega(1)$ hidden neurons

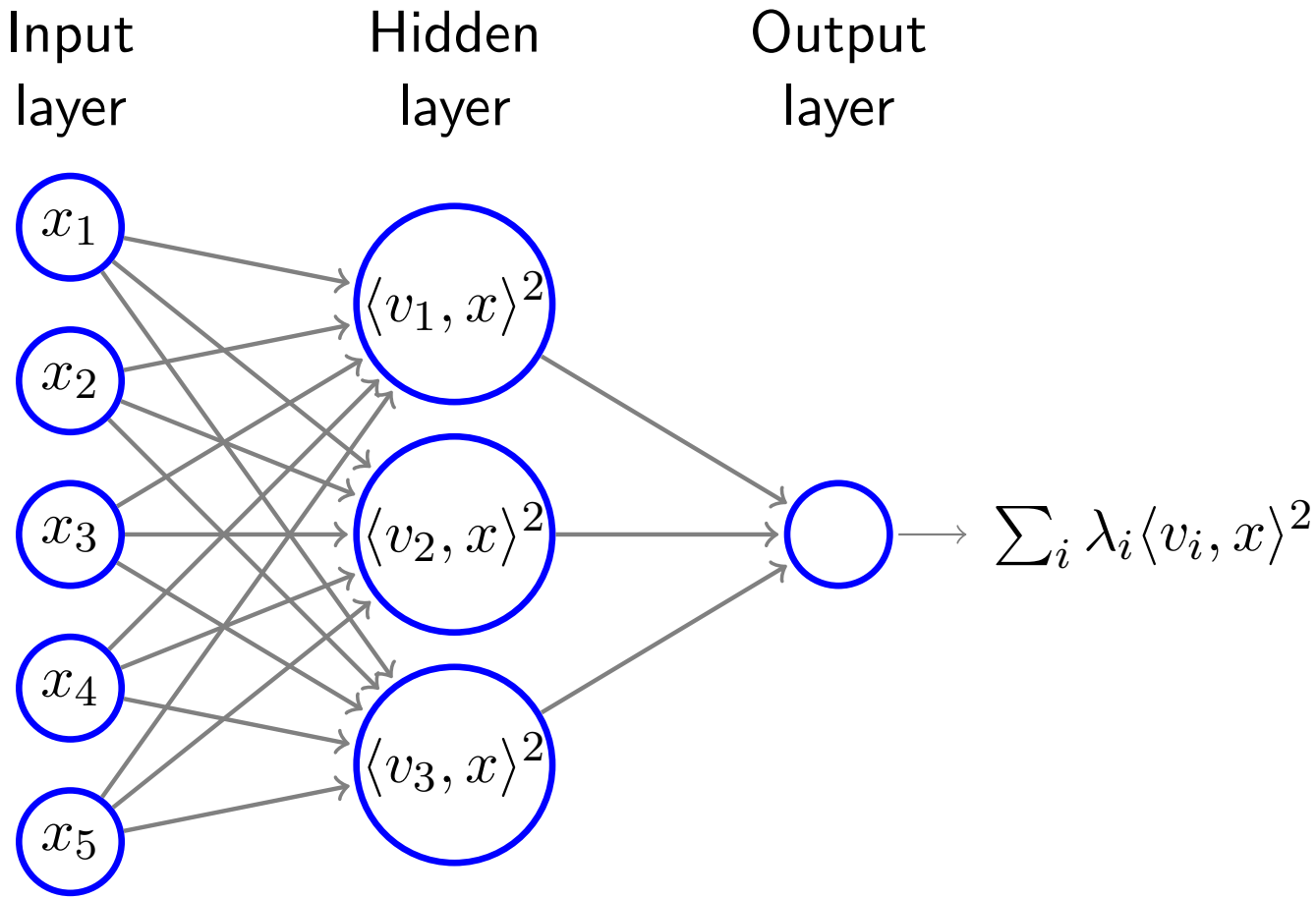
2. Change the activation function (sum-product networks)
   - An efficient algorithm for learning sum-product networks of depth 2 and small size using over-specification
   - Hardness of learning deep sum-product networks

3. Distributional assumptions
   - Learning of algebraic sets

Circumventing hardness — sum-product networks

- Simpler non-linearity — replace sigmoidal activation function by the square function $\sigma(a) = a^2$
- Network implements polynomials, where the depth corresponds to degree
- The size of the network (number of neurons) determines generalization properties and evaluation time
- Can we efficiently learn the class of polynomial networks of small size?

Depth 2 polynomial network



Depth 2 polynomial networks

- Corresponding hypothesis class:

$$\mathcal{H} = \left\{ x \mapsto \sum_{i=1}^{r} \lambda_i \langle v_i, x \rangle^2 \;:\; \lambda_i \in \mathbb{R}, v_i \in \mathbb{R}^d \right\} .$$

Depth 2 polynomial networks

- Corresponding hypothesis class:

$$\mathcal{H} = \left\{ x \mapsto \sum_{i=1}^{r} \lambda_i \langle v_i, x \rangle^2 \;:\; \lambda_i \in \mathbb{R}, v_i \in \mathbb{R}^d \right\} .$$

- ERM is still NP hard
- But, here, over-specification works !

Depth 2 polynomial networks

- Corresponding hypothesis class:

$$\mathcal{H} = \left\{ x \mapsto \sum_{i=1}^{r} \lambda_i \langle v_i, x \rangle^2 \;:\; \lambda_i \in \mathbb{R}, v_i \in \mathbb{R}^d \right\} .$$

- ERM is still NP hard
- But, here, over-specification works !
- Using d^2 hidden neurons suffices (trivial)
- Can we do better?

Learning depth 2 polynomial networks using GECO

Greedy Efficient Component Optimization (GECO):

- Initialize $V = [\,]$, $\lambda = []$
- For $t = 1, 2, \ldots, T$
  - Let $M = \mathbb{E}_{(x,y)}(\sum_i \lambda_i(\langle v_i, x\rangle)^2 - y)xx^\top$
  - $V = [V\ v]$ where v is a leading eigenvector of M
  - Let $B = \operatorname{argmin}_B \mathbb{E}_{(x,y)}((Vx)^\top B(Vx) - y)^2$
  - Update $\lambda = \text{eigenvalues}(B)$ and $V = V\text{eigenvectors}(B)$

Learning depth 2 polynomial networks using GECO

Greedy Efficient Component Optimization (GECO):

- Initialize $V = [\,]$, $\lambda = []$
- For $t = 1, 2, \ldots, T$
  - Let $M = \mathbb{E}_{(x,y)}(\sum_i \lambda_i(\langle v_i, x\rangle)^2 - y)xx^\top$
  - $V = [V\ v]$ where v is a leading eigenvector of M
  - Let $B = \operatorname{argmin}_B \mathbb{E}_{(x,y)}((Vx)^\top B(Vx) - y)^2$
  - Update $\lambda = \text{eigenvalues}(B)$ and $V = V\text{eigenvectors}(B)$

Analysis:

- For every $\lambda_1, \ldots, \lambda_r$ and $v_1, \ldots, v_r$ s.t. $\|v_i\| = 1$ and $|\lambda_i| = O(1)$
- If $T \geq \Omega(r^2/\epsilon^2)$ then the output of GECO is ϵ-accurate
- Over-specification helps !

High degree polynomials ?

- Learning sigmoidal networks is hard even of depth 2 and $\omega(1)$ hidden neurons, and even if we allow over-specification
- Learning polynomial networks of depth 2 is tractable if we allow over-specification
- What about higher degrees?

Theorem (Livni, Shamir, S.): It is hard to learn polynomial networks of depth $\mathrm{poly}(d)$ even if their size is $\mathrm{poly}(d)$.
Proof idea: It is possible to approximate the sigmoid function with a polynomial of degree $\mathrm{poly}(d)$

- What about depth 3 and constant number of hidden neurons?

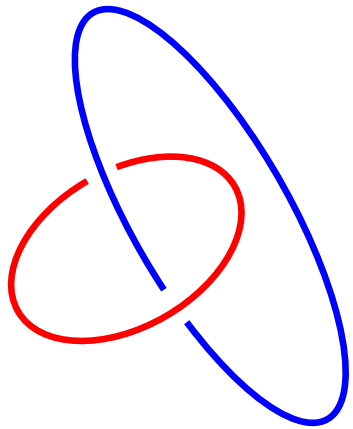
Outline

How to circumvent hardness?

1. Over-specification
   - Extreme over-specification eliminate local (non-global) minima
   - Hardness of improperly learning a two layers network with $k = \omega(1)$ hidden neurons
2. Change the activation function (sum-product networks)
   - An efficient algorithm for learning sum-product networks of depth 2 and small size using over-specification
   - Hardness of learning deep sum-product networks
3. Distributional assumptions
   - Learning of algebraic sets

Distributional Assumptions

- A set of points is an algebraic set if it is the set of solutions to a set of polynomial equations
- Assume that the positive and negative examples lie on different algebraic sets
- Can we efficiently train a network that classifies the data?



Vanishing Component Analysis (VCA)

- The vanishing ideal: $I(S)$, for $S \subset \mathbb{R}^d$, is the set of all polynomials p s.t. $\forall x \in S, p(x) = 0$
- Generators: $f_1, \ldots, f_k$ are generators of ideal I if every $f \in I$ can be written as $f = \sum_{i=1}^{k} g_i f_i$, for g_i being polynomials
- Hilbert's basis theorem: Every ideal is generated by a finite set of polynomials
- Goal: Given a finite set of points, $S \subset \mathbb{R}^d$, efficiently find a small set of polynomials that generates $I(S)$

Vanishing Component Analysis (VCA)

Main ideas:

- Given p and $S = (x_1, \ldots, x_m)$ define $p(S) = (p(x_1), \ldots, p(x_m))$
- Every linear operation on $p(S)$ has an analogue on p
- Let $C_1 = [x_1(S) \ \ldots \ x_d(S)]$.
  - Perform SVD on C_1
  - Non-vanishing eigenvectors go to F_1
  - Vanishing eigenvectors go to V_1
- Induction step
  - Assume $F_1, \ldots, F_t$ spans non-vanishing polynomials of degree at most t and $V_1, \ldots, V_t$ generates vanishing polynomials of degree at most t
  - Grading property: Every polynomial f of degree $t+1$ can be written as $q + \sum_i g_i h_i$ where q is of degree at most t, all h_i are of degree t and all g_i are of degree 1
  - Let $C_{t+1} = [\ g(S)h(S) \ : \ g \in F_t, h \in F_1\ ]$
  - Obtain F_{t+1}, V_{t+1} by SVD'ing C_{t+1}

Vanishing Component Analysis (VCA)

Analysis

- Correctness: For every t, for every p of degree t, we can write $p = g + h$ where $g \in \text{span}(F_1, \ldots, F_t)$ and h is in the ideal generated by $V_1, \ldots, V_t$
- Usefulness: If negative and positive examples are on different algebraic set, using F, V as features yields linearly separable data
- Efficiency: Number of polynomials and their evaluation time is polynomial in m, d
- What about statistical usefulness ?

Comparing VCA to Polynomial Kernels

- Polynomial kernels also rely on a distributional assumption: large margin in the feature space
- VCA relies on a different distributional assumption
- Which assumption is more natural / realistic?

Summary

- Deep networks are great statistically but cannot be trained efficiently
- Main open problem: Find a combination of network architecture and distributional assumptions that are useful in practice and lead to efficient algorithms

Thanks!

Collaborators

- Seek of efficient algorithms for deep learning: **Ohad Shamir**
- GECO: Alon Gonen and Ohad Shamir
  Based on a previous paper with Tong Zhang and Nati Srebro
- VCA: Roi Livni, David Lehavi, Hila Nachlieli, Sagi Schein, Amir Globerson
- Lower bounds: Amit Daniely and Nati Linial