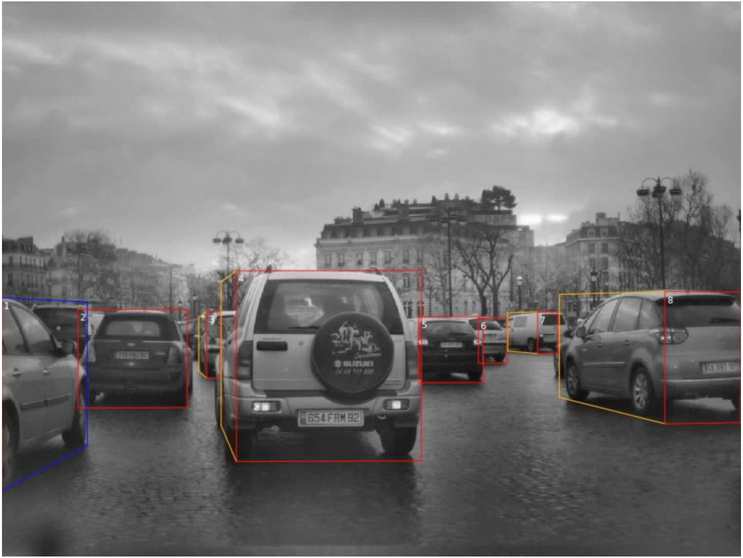
Failures of Gradient-Based Deep Learning

Shai Shalev-Shwartz, Shaked Shammah, Ohad Shamir

The Hebrew University and Mobileye

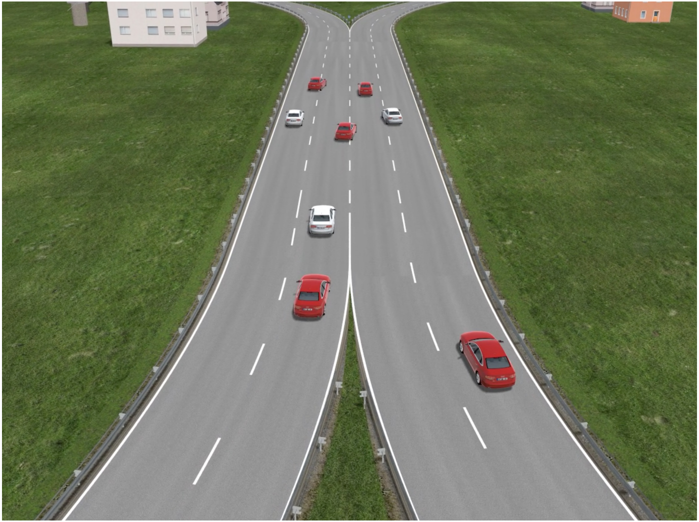
Representation Learning Workshop
Simons Institute, Berkeley, 2017

Deep Learning is Amazing ...



Deep Learning is Amazing ...

Deep Learning is Amazing ...



Deep Learning is Amazing ...

Deep Learning is Amazing ...

INTEL IS BUYING MOBILEYE FOR $15.3 BILLION IN BIGGEST ISRAELI HI-TECH DEAL EVER

This Talk

Simple problems where standard deep learning either

- Does not work well
  - Requires prior knowledge for better architectural/algorithmic choices
  - Requires other than gradient update rule
  - Requiers to decompose the problem and add more supervision
- Does not work at all
  - No “local-search” algorithm can work
  - Even for “nice” distributions and well-specified models
  - Even with over-parameterization (a.k.a. improper learning)

Mix of theory and experiments

Outline

Piecewise-linear Curves: Motivation

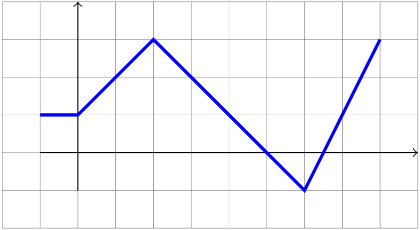
Piecewise-linear Curves

Problem: Train a piecewise-linear curve detector

Input: $\mathbf{f} = (f(0), f(1), \ldots, f(n-1))$ where

$$f(x) = \sum_{r=1}^{k} a_r [x - \theta_r]_+ \quad , \quad \theta_r \in \{0, \ldots, n-1\}$$

Output: Curve parameters $\{a_r, \theta_r\}_{r=1}^{k}$

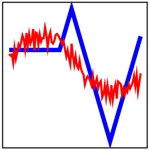


First try: Deep AutoEncoder

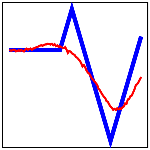
- Encoding network, E_{w_1}: Dense(500,relu)-Dense(100,relu)-Dense($2k$)
- Decoding network, D_{w_2}: Dense(100,relu)-Dense(100,relu)-Dense(n)
- Squared Loss: $(D_{w_2}(E_{w_1}(\mathbf{f})) - \mathbf{f})^2$

First try: Deep AutoEncoder

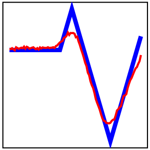
- Encoding network, E_{w_1}: Dense(500,relu)-Dense(100,relu)-Dense($2k$)
- Decoding network, D_{w_2}: Dense(100,relu)-Dense(100,relu)-Dense(n)
- Squared Loss: $(D_{w_2}(E_{w_1}(\mathbf{f})) - \mathbf{f})^2$
- Doesn't work well ...



500 iterations



10,000 iterations



50,000 iterations

Second try: Pose as a Convex Objective

Problem: Train a piecewise-linear curve detector

Input: $\mathbf{f} \in \mathbb{R}^n$ where $f(x) = \sum_{r=1}^{k} a_r [x - \theta_r]_+$
Output: Curve parameters $\{a_r, \theta_r\}_{r=1}^{k}$

Convex Formulation

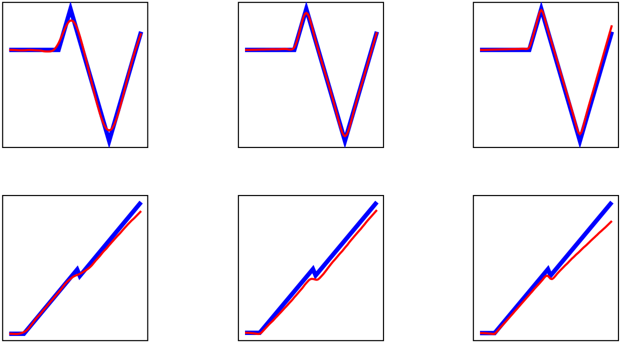
- Let $\mathbf{p} \in \mathbb{R}^{n,n}$ be a k-sparse vector whose k max/argmax elements are $\{a_r, \theta_r\}_{r=1}^{k}$
- Observe: $\mathbf{f} = W\mathbf{p}$ where $W \in \mathbb{R}^{n,n}$ is s.t. $W_{i,j} = [i - j + 1]_+$
- Learning approach — linear regression: Train a one-layer fully connected network on $(\mathbf{f}, \mathbf{p})$ examples:

$$\min_{U} \mathbb{E}\left[(U\mathbf{f} - \mathbf{p})^2\right] = \mathbb{E}\left[(U\mathbf{f} - W^{-1}\mathbf{f})^2\right]$$

Second try: Pose as a Convex Objective

$$\min_U \mathbb{E}\left[(U\mathbf{f}-\mathbf{p})^2\right] = \mathbb{E}\left[(U\mathbf{f}-W^{-1}\mathbf{f})^2\right]$$

- Convex; Realizable; but still doesn't work well ...



500 iterations 10,000 iterations 50,000 iterations

What went wrong?

Theorem

The convergence of SGD is governed by the condition number of $W^\top W$*, which is large:*

$$\frac{\lambda_{\max}(W^\top W)}{\lambda_{\min}(W^\top W)} = \Omega(n^{3.5})$$

$\Rightarrow$ *SGD requires* $\Omega(n^{3.5})$ *iterations to reach* U *s.t.* $\left\|\mathbb{E}[U] - W^{-1}\right\| < 1/2$

- Note: Adagrad/Adam doesn't work because they perform diagonal conditioning

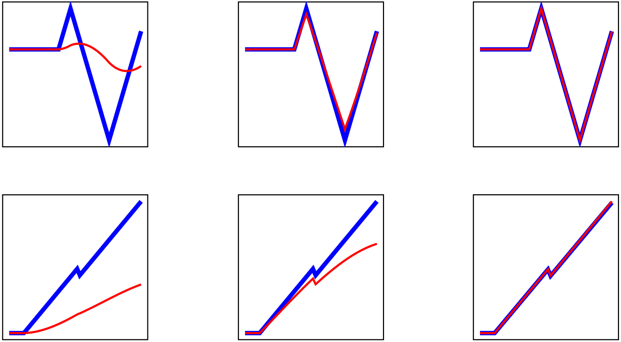
3rd Try: Pose as a Convolution

- $\mathbf{p} = W^{-1}\mathbf{f}$
- Observation:

$$W^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots \\ -2 & 1 & 0 & 0 & \cdots \\ 1 & -2 & 1 & 0 & \cdots \\ 0 & 1 & -2 & 1 & \cdots \\ 0 & 0 & 1 & -2 & \cdots \\ \vdots & \vdots & \vdots & & \end{pmatrix}$$

- $W^{-1}\mathbf{f}$ is 1D convolution of $\mathbf{f}$ with "2nd derivative" filter $(1, -2, 1)$
- Can train a one-layer convnet to learn filter (problem in $\mathbb{R}^3$!)

Better, but still doesn't work well ...



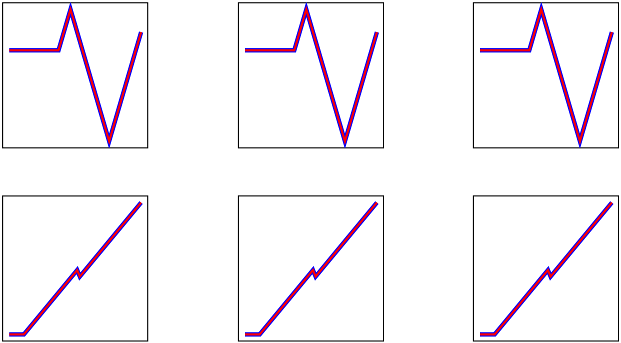
500 iterations 10,000 iterations 50,000 iterations

- Theorem: Condition number reduced to $\Theta(n^3)$. Convolutions aid geometry!
- But, $\Theta(n^3)$ is very disappointing for a problem in $\mathbb{R}^3$ ...

4th Try: Convolution + Preconditioning

- ConvNet is equivalent to solving $\min_{\mathbf{x}} \mathbb{E}\left[(F\mathbf{x}-\mathbf{p})^2\right]$ where F is $n \times 3$ matrix with $(f(i-1), f(i), f(i+1))$ at row i
- Observation: Problem is now low-dimensional, so can easily precondition
  - Compute empirical approximation C of $\mathbb{E}[F^\top F]$
  - Solve $\min_{\mathbf{x}} \mathbb{E}[(FC^{-1/2}\mathbf{x}-\mathbf{p})^2]$
  - Return $C^{1/2}\mathbf{x}$
- Condition number of $FC^{-1/2}$ is close to 1

Finally, it works ...



500 iterations | 10,000 iterations | 50,000 iterations

Remark:

- Use of convnet allows for efficient preconditioning
- Estimating and manipulating 3×3 rather than $n \times n$ matrices.

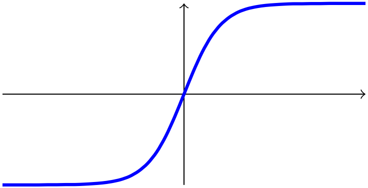
Lesson Learned

- SGD might be extremely slow
- Prior knowledge allows us to:
  - Choose a better architecture (not for expressivity, but for a better geometry)
  - Choose a better algorithm (preconditioned SGD)

Outline

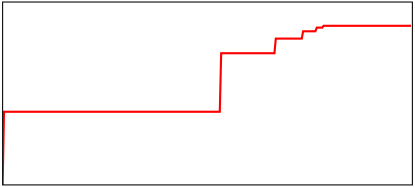
Flat Activations

Vanishing gradients due to saturating activations (e.g. in RNN's)



Flat Activations

Problem: Learning $\mathbf{x} \mapsto u(\langle \mathbf{w}^\star, \mathbf{x} \rangle)$ where u is a fixed step function



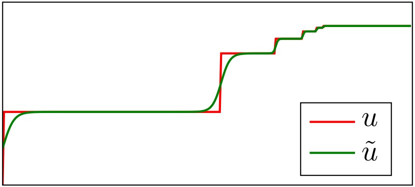
Optimization problem:

$$\min_{\mathbf{w}} \mathop{\mathbb{E}}_{\mathbf{x}}[(u(N_{\mathbf{w}}(\mathbf{x})) - u(\mathbf{w}^{\star\top}\mathbf{x}))^2]$$

$u'(z) = 0$ almost everywhere $\rightarrow$ can't apply gradient-based methods

Flat Activations: Smooth approximation

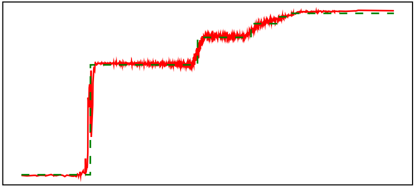
Smooth approximation: replace u with $\tilde{u}$ (similar to using sigmoids as gates in LSTM's)



Sometimes works, but slow and only approximate result. Often completely fails

Approach: End-to-end

$$\min_{\mathbf{w}} \mathbb{E}_{\mathbf{x}}[(N_{\mathbf{w}}(\mathbf{x}) - u(\mathbf{w}^{\star\top}\mathbf{x}))^2]$$



(3 ReLU + 1 linear layers; 10000 iterations)

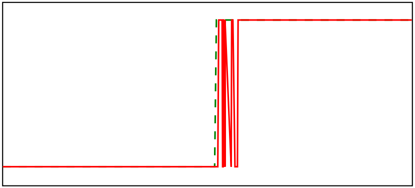
Slow train+test time; curve not captured well

Flat Activations: Multiclass

Approach: Multiclass

$$\min_{\mathbf{w}} \mathbb{E}_{\mathbf{x}}[\ell(N_{\mathbf{w}}(\mathbf{x}), y(\mathbf{x}))]$$

$N_{\mathbf{w}}(\mathbf{x})$: to which step does $\mathbf{x}$ belong



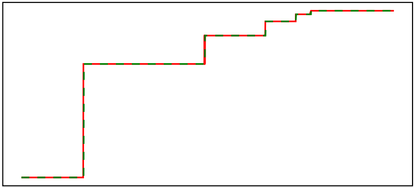
Problem capturing boundaries

Flat Activations: "Forward only" backpropagation

Different approach (Kalai & Sastry 2009, Kakade, Kalai, Kanade, Shamir 2011): Gradient descent, but replace gradient with something else

- Objective: $\min_{\mathbf{w}} \mathbb{E}_{\mathbf{x}} \left[\frac{1}{2} \left((u(\mathbf{w}^\top \mathbf{x})) - u(\mathbf{w}^{\star\top} \mathbf{x}) \right)^2 \right]$
- Gradient: $\nabla = \mathbb{E}_{\mathbf{x}} \left[\left(u(\mathbf{w}^\top \mathbf{x}) - u(\mathbf{w}^{\star\top} \mathbf{x}) \right) \cdot u'(\mathbf{w}^\top \mathbf{x}) \cdot \mathbf{x} \right]$
- Non-gradient direction: $\tilde{\nabla} = \mathbb{E}_{\mathbf{x}} \left[(u(\mathbf{w}^\top \mathbf{x}) - u(\mathbf{w}^{\star\top} \mathbf{x}))\mathbf{x} \right]$
- Interpretation: "Forward only" backpropagation

Flat Activation



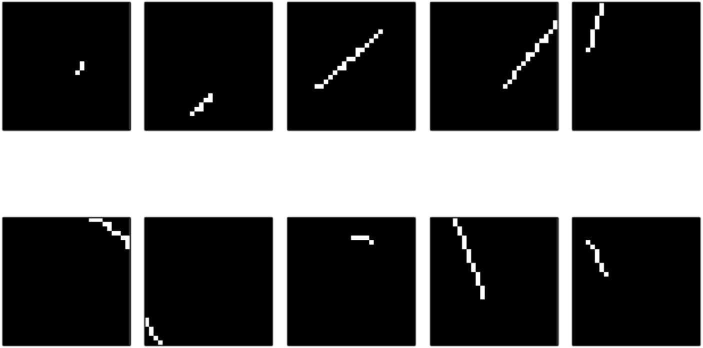
(linear; 5000 iterations)

- Best results, and smallest train+test time
- Analysis (KS09, KKKS11): Needs $\mathcal{O}(L^2/\epsilon^2)$ iterations if u is L-Lipschitz
- Lesson learned: Local search works, but not with the gradient ...

Outline

End-to-End vs. Decomposition

- Input $\mathbf{x}$: k-tuple of images of random lines
- $f_1(\mathbf{x})$: For each image, whether slope is negative/positive
- $f_2(\mathbf{x})$: return parity of slope signs
- Goal: Learn $f_2(f_1(\mathbf{x}))$

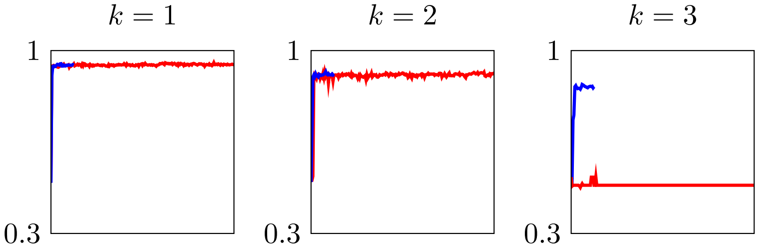


End-to-End vs. Decomposition

- Architecture: Concatenation of Lenet and 2-layer ReLU, linked by sigmoid
- End-to-end approach: Train overall network on primary objective
- Decomposition approach: Augment objective with loss specific to first net, using per-image labels

End-to-End vs. Decomposition

- Architecture: Concatenation of Lenet and 2-layer ReLU, linked by sigmoid
- End-to-end approach: Train overall network on primary objective
- Decomposition approach: Augment objective with loss specific to first net, using per-image labels



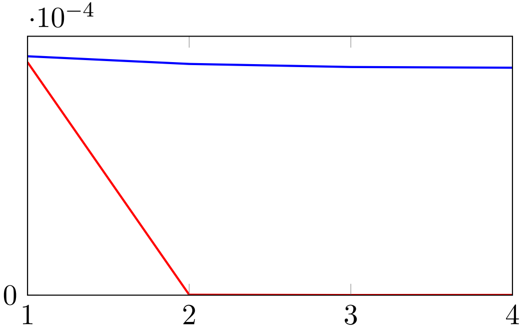
20000 iterations

End-to-End vs. Decomposition

- Why end-to-end training doesn't work ?
  - Similar experiment by Gulcehre and Bengio, 2016
  - They suggest "local minima" problems
  - We show that the problem is different

End-to-End vs. Decomposition

- Why end-to-end training doesn't work ?
  - Similar experiment by Gulcehre and Bengio, 2016
  - They suggest "local minima" problems
  - We show that the problem is different
- Signal-to-Noise Ratio (SNR) for random initialization:
  - End-to-end (red) vs. decomposition (blue), as a function of k
  - SNR of end-to-end for $k \geq 3$ is below the precision of float32



Outline

Learning Many Orthogonal Functions

- Let $\mathcal{H}$ be a hypothesis class of orthonormal functions: $\forall h, h' \in \mathcal{H}$, $\mathbb{E}[h(\mathbf{x})h'(\mathbf{x})] = 0$

Learning Many Orthogonal Functions

- Let $\mathcal{H}$ be a hypothesis class of orthonormal functions: $\forall h, h' \in \mathcal{H}$, $\mathbb{E}[h(\mathbf{x})h'(\mathbf{x})] = 0$
- (Improper) learning of $\mathcal{H}$ using gradient-based deep learning:
  - Learn the parameter vector, $\mathbf{w}$, of some architecture, $p_{\mathbf{w}} : X \to \mathbb{R}$
  - For every target $h \in \mathcal{H}$, the learning task is to solve:

$$\min_{\mathbf{w}} F_h(\mathbf{w}) := \mathbb{E}_{\mathbf{x}}[\ell(p_{\mathbf{w}}(\mathbf{x}), h(\mathbf{x})]$$

  - Start with a random $\mathbf{w}$ and update the weights based on $\nabla F_h(\mathbf{w})$

Learning Many Orthogonal Functions

- Let $\mathcal{H}$ be a hypothesis class of orthonormal functions: $\forall h, h' \in \mathcal{H}$, $\mathbb{E}[h(\mathbf{x})h'(\mathbf{x})] = 0$
- (Improper) learning of $\mathcal{H}$ using gradient-based deep learning:
  - Learn the parameter vector, $\mathbf{w}$, of some architecture, $p_{\mathbf{w}} : X \to \mathbb{R}$
  - For every target $h \in \mathcal{H}$, the learning task is to solve:

$$\min_{\mathbf{w}} F_h(\mathbf{w}) := \mathbb{E}_{\mathbf{x}}[\ell(p_{\mathbf{w}}(\mathbf{x}), h(\mathbf{x})]$$

  - Start with a random $\mathbf{w}$ and update the weights based on $\nabla F_h(\mathbf{w})$
- Analysis tool: How much $\nabla F_h(\mathbf{w})$ tells us about the identity of h ?

Analysis: how much information in the gradient?

$$\min_{\mathbf{w}} F_h(\mathbf{w}) := \mathop{\mathbb{E}}_{\mathbf{x}}[\ell(p_{\mathbf{w}}(\mathbf{x}), h(\mathbf{x})]$$

- Theorem: For every $\mathbf{w}$, there are many pairs $h, h' \in \mathcal{H}$ s.t. $\mathbb{E}_{\mathbf{x}}[h(\mathbf{x})h'(\mathbf{x})] = 0$ while

$$\|\nabla F_h(\mathbf{w}) - \nabla F_{h'}(\mathbf{w})\|^2 = O\left(\frac{1}{|\mathcal{H}|}\right)$$

Analysis: how much information in the gradient?

$$\min_{\mathbf{w}} F_h(\mathbf{w}) := \mathbb{E}_{\mathbf{x}}[\ell(p_{\mathbf{w}}(\mathbf{x}), h(\mathbf{x})]$$

- Theorem: For every $\mathbf{w}$, there are many pairs $h, h' \in \mathcal{H}$ s.t. $\mathbb{E}_{\mathbf{x}}[h(\mathbf{x})h'(\mathbf{x})] = 0$ while

$$\|\nabla F_h(\mathbf{w}) - \nabla F_{h'}(\mathbf{w})\|^2 = O\left(\frac{1}{|\mathcal{H}|}\right)$$

- Proof idea: show that if the functions in $\mathcal{H}$ are orthonormal then, for every $\mathbf{w}$,

$$\mathrm{Var}(\mathcal{H}, F, \mathbf{w}) := \mathbb{E}_h \left\|\nabla F_h(\mathbf{w}) - \mathbb{E}_{h'} \nabla F_{h'}(\mathbf{w})\right\|^2 = O\left(\frac{1}{|\mathcal{H}|}\right)$$

- To do so, express every coordinate of $\nabla p_{\mathbf{w}}(\mathbf{x})$ using the orthonormal functions in $\mathcal{H}$

Proof idea

Assume the squared loss, then

$$\begin{aligned}\nabla F_h(\mathbf{w}) &= \mathop{\mathbb{E}}_{\mathbf{x}}[(p_{\mathbf{w}}(\mathbf{x}) - h(\mathbf{x})\,\nabla p_{\mathbf{w}}(\mathbf{x})] \\ &= \underbrace{\mathop{\mathbb{E}}_{\mathbf{x}}[p_{\mathbf{w}}(\mathbf{x})\,\nabla p_{\mathbf{w}}(\mathbf{x})]}_{\text{independent of } h} - \mathop{\mathbb{E}}_{\mathbf{x}}[h(\mathbf{x})\,\nabla p_{\mathbf{w}}(\mathbf{x})]\end{aligned}$$

Proof idea

Assume the squared loss, then

$$\nabla F_h(\mathbf{w}) = \mathbb{E}_{\mathbf{x}}[(p_{\mathbf{w}}(\mathbf{x}) - h(\mathbf{x})\,\nabla p_{\mathbf{w}}(\mathbf{x})]$$
$$= \underbrace{\mathbb{E}_{\mathbf{x}}[p_{\mathbf{w}}(\mathbf{x})\,\nabla p_{\mathbf{w}}(\mathbf{x})]}_{\text{independent of } h} - \mathbb{E}_{\mathbf{x}}[h(\mathbf{x})\,\nabla p_{\mathbf{w}}(\mathbf{x})]$$

- Fix some j and denote $g(\mathbf{x}) = \nabla_j p_{\mathbf{w}}(\mathbf{x})$
- Can expand $g = \sum_{i=1}^{|\mathcal{H}|} \langle h_i, g\rangle h_i + \text{orthogonal component}$
- Therefore, $\mathbb{E}_h\left(\mathbb{E}_{\mathbf{x}}[h(\mathbf{x})\,g(\mathbf{x})]\right)^2 \leq \frac{\mathbb{E}_{\mathbf{x}}[g(\mathbf{x})^2]}{|\mathcal{H}|}$
- It follows that

$$\mathrm{Var}(\mathcal{H}, F, \mathbf{w}) \;\leq\; \frac{\mathbb{E}_{\mathbf{x}}[\|\nabla p_{\mathbf{w}}(\mathbf{x})\|^2]}{|\mathcal{H}|}$$

Example: Parity Functions

- $\mathcal{H}$ is the class of parity functions over $\{0,1\}^d$ and $\mathbf{x}$ is uniformly distributed
- There are 2^d orthonormal functions, hence there are many pairs $h, h' \in \mathcal{H}$ s.t. $\mathbb{E}_{\mathbf{x}}[h(\mathbf{x})h'(\mathbf{x})] = 0$ while $\|\nabla F_h(\mathbf{w}) - \nabla F_{h'}(\mathbf{w})\|^2 = O(2^{-d})$

Example: Parity Functions

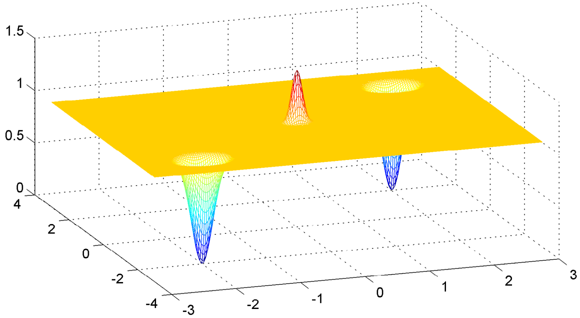
- $\mathcal{H}$ is the class of parity functions over $\{0,1\}^d$ and $\mathbf{x}$ is uniformly distributed
- There are 2^d orthonormal functions, hence there are many pairs $h, h' \in \mathcal{H}$ s.t. $\mathbb{E}_{\mathbf{x}}[h(\mathbf{x})h'(\mathbf{x})] = 0$ while $\|\nabla F_h(\mathbf{w}) - \nabla F_{h'}(\mathbf{w})\|^2 = O(2^{-d})$

Remark:

- Similar hardness result can be shown by combining existing results:
  - Parities on uniform distribution over $\{0,1\}^d$ is difficult for statistical query algorithms (Kearns, 1999)
  - Gradient descent with approximate gradients can be implemented with statistical queries (Feldman, Guzman, Vempala 2015)

Visual Illustration: Linear-Periodic Functions

$F_h(\mathbf{w}) = \mathbb{E}_{\mathbf{x}}[(\cos(\mathbf{w}^\top \mathbf{x}) - h(\mathbf{x}))^2]$ for $h(\mathbf{x}) = \cos([2,2]^\top \mathbf{x})$, in 2 dimensions, $\mathbf{x} \sim \mathcal{N}(0, I)$:



- No local minima/saddle points
- However, extremely flat unless very close to optimum
  $\Rightarrow$ difficult for gradient methods, especially stochastic
- In fact, difficult for any local-search method

Summary

- Cause of failures: optimization can be difficult for geometric reasons other than local minima / saddle points
  - Condition number, flatness
  - Using bigger/deeper networks doesn't always help
- Remedies: prior knowledge can still be important
  - Convolution can improve geometry (and not just sample complexity)
  - "Other than gradient" update rule
  - Decomposing the problem and adding supervision can improve geometry
- Understanding the limitations: While deep learning is great, understanding the limitations may lead to better algorithms and/or better theoretical guarantees
- For more information:
  - "Failures of Deep Learning": arxiv 1703.07950
  - `github.com/shakedshammah/failures_of_DL`