EE270
Large Scale Matrix Computation, Optimization and Learning

Lecture 20

Instructor : Mert Pilanci

March 16, 2021

Stanford University

Outline

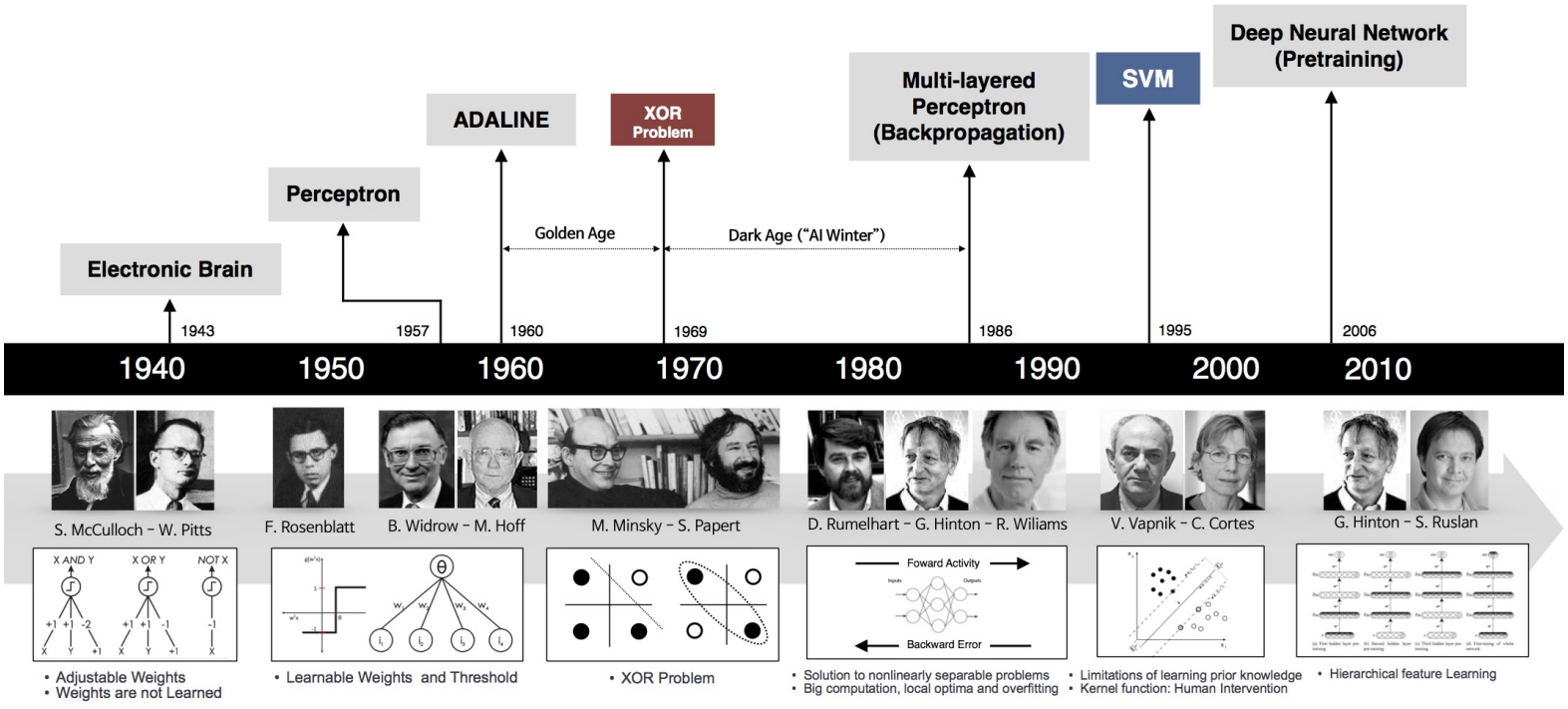
Neural Networks (NNs)

Optimizing NNs

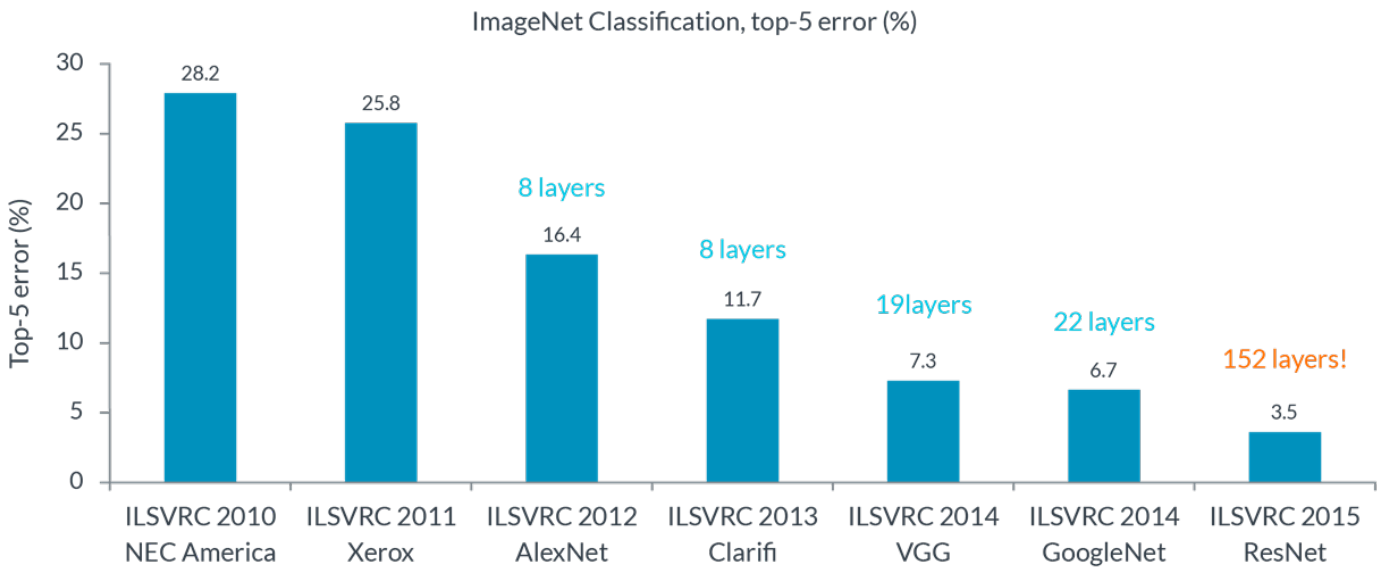
Regularized Least Squares

Convex Optimization for NNs

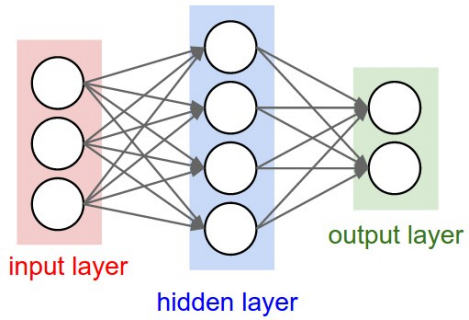
Neural Network Timeline



Deep learning revolution



Neural Networks



The output predicted by the network given input x is

$$f_1(x), f_2(x), ...f_k(x)$$

$y_k = f_k(x) = g_k(\beta_k^T z) \quad \forall k$ and $z_m = \sigma(\alpha_m^T x)$

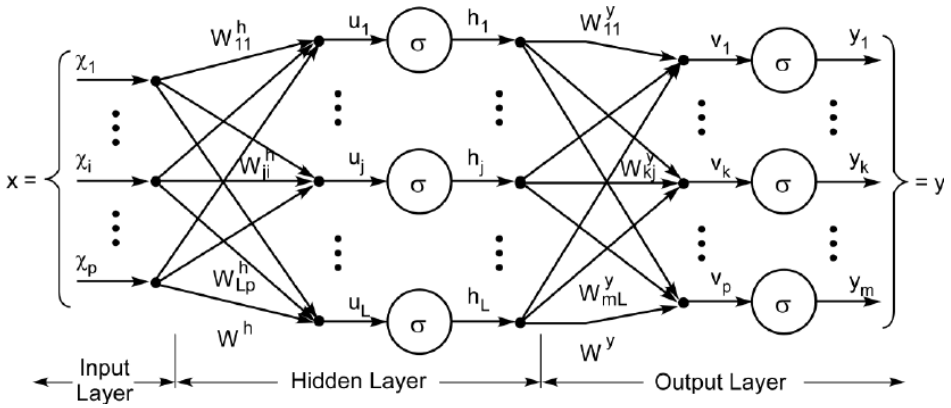
$\sigma(\cdot)$ and $g_1, ...g_K$ are nonlinear functions

Regression (continuous output): $g_t(u) = u$ and $f(x) = \beta^T z$

Multi-class classification $g_k(u_1, ..., u_K) = \frac{e^{u_k}}{\sum_{j=1}^{K} e^{u_j}}$

One-hot encoding: x is class $k \quad y = e_k$ (kth ordinary basis)

Multilayer Neural Networks



$$z^{(0)} = x \quad \text{(input)}$$

$$a_j^{(l)} = \sum_i W_{ij}^{(l)} z_i^{(l-1)} \qquad l = 1, ..., L$$

$$z_j^{(l)} = \sigma(a_k^l) \qquad l = 1, ..., L$$

Training Multilayer Neural Networks

Training: Parameters $\Theta = (W^{(1)}, W^{(2)}, ..., W^{(L)})$

regression: (squared loss) vs classification (cross-entropy loss)

$$\min_{\Theta} \sum_{n=1}^{N} \underbrace{(y_n - f(x_n))^2}_{R_n(\Theta)} \qquad \min_{\Theta} - \sum_{n=1}^{N} \underbrace{\sum_{k=1}^{K} y_{nk} \log f_k(x_n)}_{R_n(\Theta)}$$

Gradient Descent

$\Theta_{t+1} = \Theta_{t+1} - \sum_{i=1}^{n} \frac{\partial}{\partial \Theta} R_n(\Theta)$

Stochastic Gradient Descent

$\Theta_{t+1} = \Theta_{t+1} - \frac{\partial}{\partial \Theta} R_{n_t}(\Theta)$

where n_t is a random index

non-convex optimization problem

Computing derivatives: Backpropagation Algorithm

$$z^{(0)} = x \quad \text{(input)}$$

$$a_j^{(l)} = \sum_i W_{ij}^{(l)} z_i^{(l-1)} \qquad l = 1, ..., L$$

$$z_j^{(l)} = \sigma(a_k^l) \qquad l = 1, ..., L$$

$$\min_{\Theta} \sum_{n=1}^{N} \underbrace{(y_n - f(x_n))^2}_{R_n(\Theta)}$$

$$\frac{\partial R_n(\Theta)}{\partial W_{ij}^{(l)}} = \frac{\partial R_n(\Theta)}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial W_{ij}^{(l)}} = \delta_{nj}^{(l)} z_i^{(l-1)}$$

where we defined $\delta_{nj}^{(l)} \triangleq \frac{\partial R_n(\Theta)}{\partial a_j^{(l)}}$

Computing derivatives: Backpropagation Algorithm

$$\delta_{nj}^{(l)} \triangleq \frac{\partial R_n(\Theta)}{\partial a_j^{(l)}} = \sum_k \frac{\partial R_n(\Theta)}{\partial a_j^{(l+1)}} \frac{\partial a_j^{(l+1)}}{\partial a_j^{(l)}}$$
$$= \sum_k \delta_{nk}^{(l+1)} W_{jk}^{(l+1)} \sigma'(a_j^{(l)})$$

last term follows from:

$$a_k^{(l+1)} = \sum_r W_{rk}^{(l+1)} z_r^{(l)} = \sum_r W_{rk}^{(l+1)} \sigma(a_r^{(l)})$$

output layer: (e.g. $R_n(\Theta) = ||a^{(L)} - y_n||^2$) $\delta_{nj}^{(L)} = 2(a^{(L)} - y_n)$

Computing the gradient involves lots of matrix-vector products

Training Neural Networks

- SGD with momentum

$$d_{t+1} = \rho d_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha d_{t+1}$$

  α is the step size (learning rate)

  ρ is the momentum parameter, e.g., $\rho = 0.9$

- $\nabla f(x_t)$ can be replaced with a subgradient for non-differentiable functions (stochastic subgradient descent)
- slow progress when the condition number is high

Diagonal Hessian Approximations

- H_t: a diagonal approximation of the Hessian $\nabla^2 f(x)$

$$x_{t+1} = x_t - \alpha H_t^{-1} \nabla f(x_t)$$
$$H_{t+1} = \text{ update using previous gradients}$$

- AdaGrad - adaptive subgradient method (Duchi et al., 2011)

  $[H_t]_{jj} = \text{diag}\Big(\big(\sum_{i=1}^{t} g_j^2 \big)^{1/2} + \delta \big)$

  where $g_j := [\nabla f(x_t)]_j$, and $\delta > 0$ small to avoid numerical issues in inversion, e.g., $\delta = 10^{-7}$

  - ▷ effectively uses different learning rates for each coordinate

    progress along flat directions is accelerated

    progress along steep directions is damped

Other Variations of Diagonal Hessian Approximations

$$x_{t+1} = x_t - \alpha H_t^{-1} \nabla f(x_t)$$

- RMSProp, Tieleman and Hinton, 2012

$$H_{t+1} = \mathrm{diag}((s_{t+1} + \delta)^{1/2})$$

weighted gradient squared update

$s_{t+1} = \gamma s_t + (1-\gamma) g_t^2$ where $g_j := [\nabla f(x_t)]_j$

- ADAM, Kingma and Ba, 2015

adds momentum and keeps a weighted sum of $[\nabla f(x_t)]_j^2$ and $[\nabla f(x_t)]_j$

Second Order Non-convex Optimization Methods

$$\min_x \sum_{i=1}^{n} (f_x(a_i) - y_i)^2$$

- Gauss-Newton method

$$x_{t+1} = \arg\min_x \quad \| \underbrace{f_{x_t}(A) + J_t x}_{\text{Taylor's approx for } f_x} - y \|_2^2 = J_t^\dagger (y - f_{x_t}(A))$$

where $(J_t)_{ij} = \frac{\partial}{\partial x_j} f_x(a_i)$ is the Jacobian matrix

Jacobians can be approximated:

- Block-diagonal approximation
- Kronecker-factored Approximate Curvature (KFAC)

  Martens and Grosse, 2015.
- Sampled uniformly/non-uniformly or sketched
- Conjugate Gradient can be used to approximate the solution

Limits and challenges of deep learning

deep learning models

- often provide the best performance due to their large capacity
- $\rightarrow$ **challenging to train**

Limits and challenges of deep learning

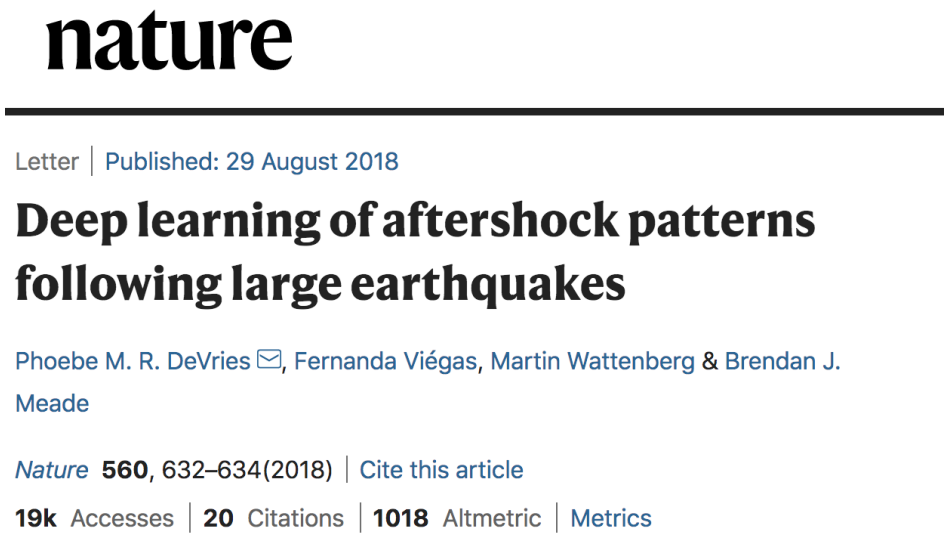
deep learning models

- often provide the best performance due to their large capacity
  $\rightarrow$ **challenging to train**
- are complex black-box systems based on non-convex optimization
  $\rightarrow$ **hard to interpret what the model is actually learning**

Limits and challenges of deep learning

deep learning models

- often provide the best performance due to their large capacity
  → **challenging to train**
- are complex black-box systems based on non-convex optimization
  → **hard to interpret what the model is actually learning**



nature

Letter | Published: 29 August 2018

Deep learning of aftershock patterns following large earthquakes

Phoebe M. R. DeVries ✉, Fernanda Viégas, Martin Wattenberg & Brendan J. Meade

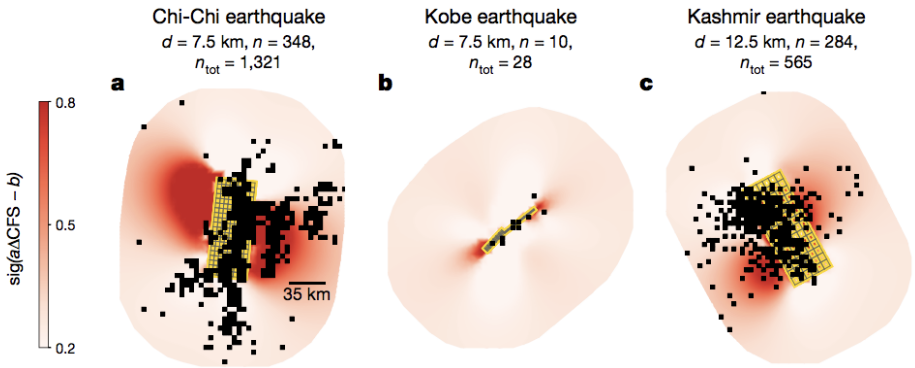
Nature **560**, 632–634(2018) | Cite this article

19k Accesses | **20** Citations | **1018** Altmetric | Metrics

Limits and challenges of deep learning

deep learning models

- often provide the best performance due to their large capacity
  → **challenging to train**
- are complex black-box systems based on non-convex optimization
  → **hard to interpret what the model is actually learning**

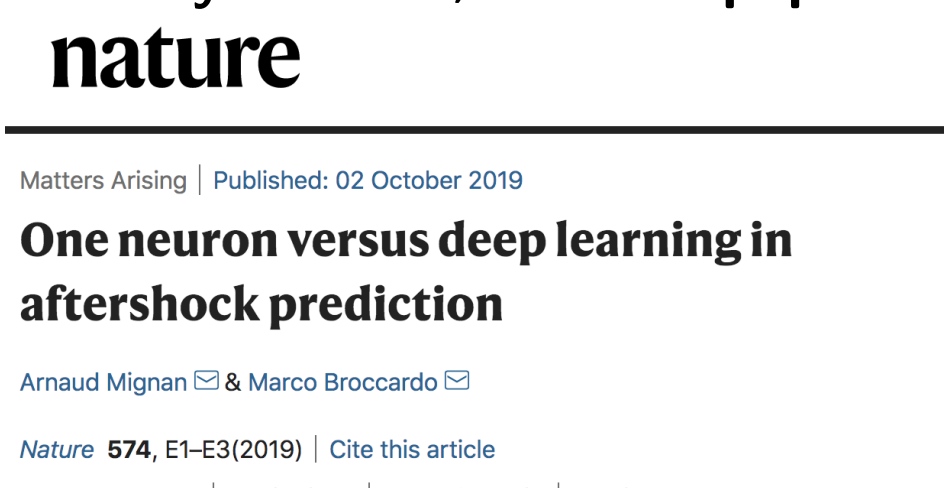


Limits and challenges of deep learning

deep learning models

- often provide the best performance due to their large capacity
  → **challenging to train**
- are complex black-box systems based on non-convex optimization
  → **hard to interpret what the model is actually learning**

one year later, another paper



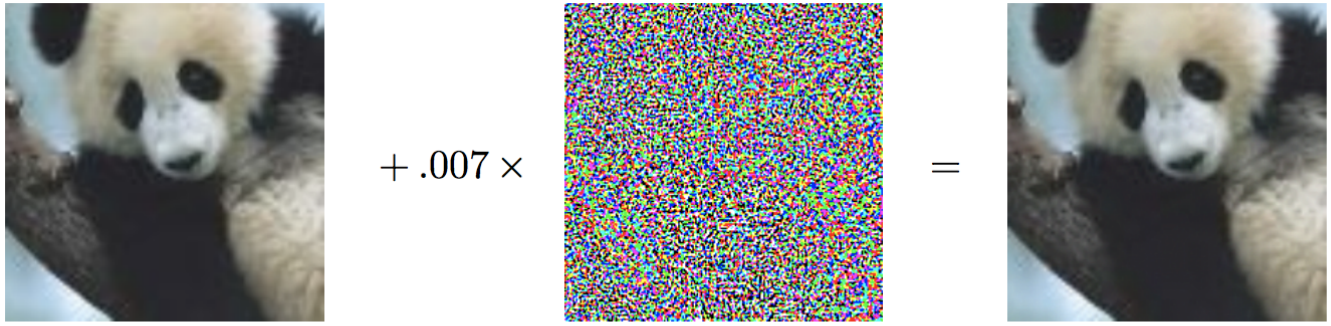
Limits and challenges of deep learning

deep learning models

- often provide the best performance due to their large capacity
  $\rightarrow$ **challenging to train**
- are complex black-box systems based on non-convex optimization
  $\rightarrow$ **hard to interpret what the model is actually learning**

logistic regression (1 layer) has the same performance as the 6 layer NN for this task

Adversarial Examples



"panda"
57.7% confidence

"nematode"
8.2% confidence

"gibbon"
99.3 % confidence

adversarial examples, Szegedy et al., 2014, Goodfellow et al., 2015

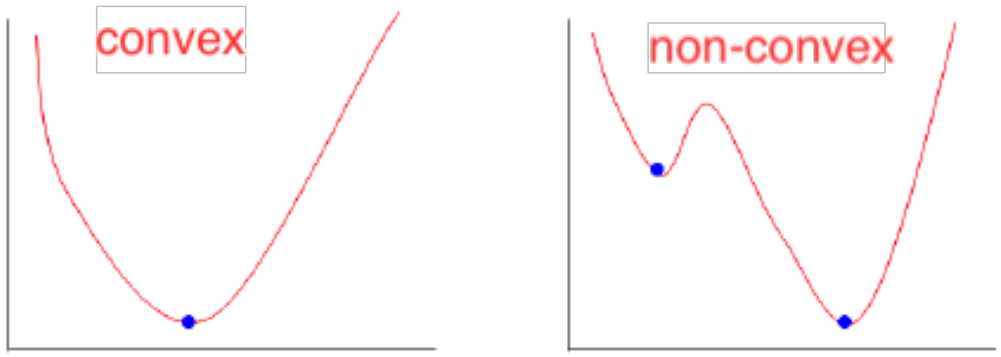
stop sign recognized as speed limit sign, Evtimov et al, 2017

Limitations of Neural Networks

How to understand what neural network models are learning? Are they automatically finding the best features?

How to train them more efficiently?

How neural networks work?



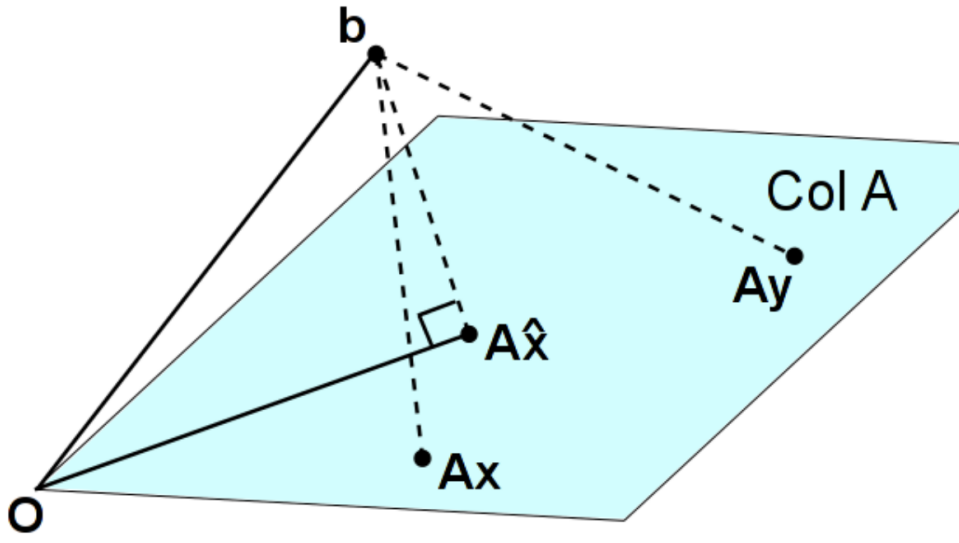
Least-Squares, Logistic Regression, Support Vector Machines etc. are

understood extremely well

Insightful theorems for neural networks?

Least Squares

$$\min_x \|Ax - b\|_2^2$$



ℓ_2 regularized form:

$$\|Ax - b\|_2^2 + \lambda\|x\|_2^2$$

algorithms: Conjugate Gradient, QR, Cholesky, SVD

left-sketched $\min_x \|SAx - b\|_2^2 + \lambda\|x\|_2^2$

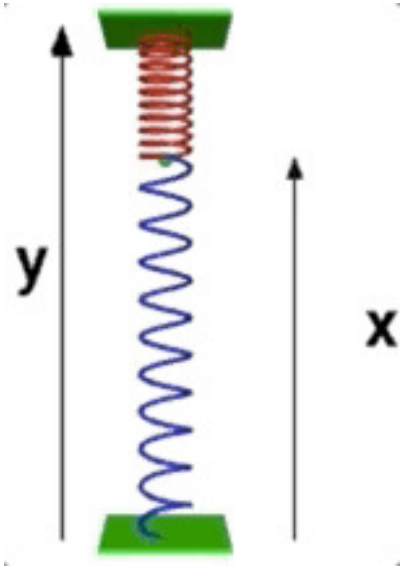
right-sketched $\min_z \|ASz - b\|_2^2 + \lambda\|Sz\|_2^2$

L2 regularization: mechanical model

$$\min_x \quad \underbrace{\frac{1}{2}(x-y)^2}_{\text{elastic energy}} \quad + \quad \underbrace{\frac{1}{2}\lambda x^2}_{\text{elastic energy}}$$

red spring constant $=1$

blue spring constant $= \lambda$

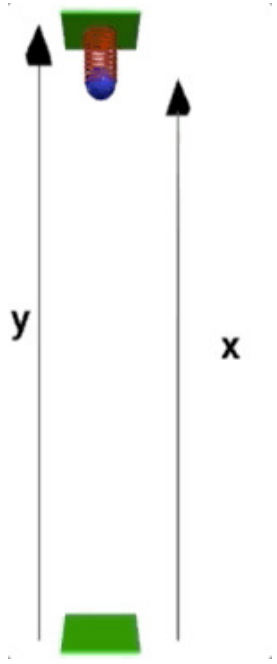


L1 regularization: mechanical model

$$\min_x \quad \underbrace{\frac{1}{2}(x-y)^2}_{\text{elastic energy}} \quad + \quad \underbrace{\lambda|x|}_{\text{potential energy}}$$

red spring constant $=1$

blue ball mass $= \lambda$ (small)

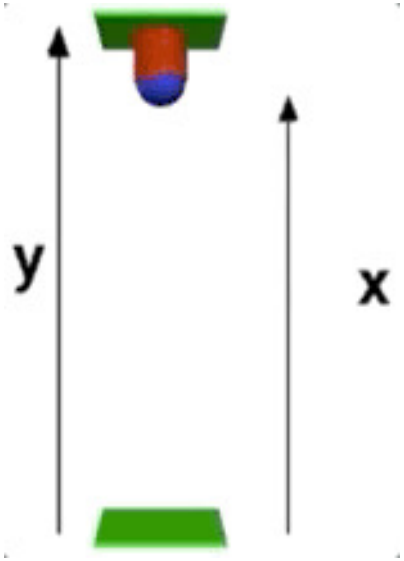


L1 regularization: mechanical model with large λ

$$\min_x \quad \underbrace{\frac{1}{2}(x-y)^2}_{\text{elastic energy}} \quad + \quad \underbrace{\lambda|x|}_{\text{potential energy}}$$

red spring constant =1

blue ball mass = λ (large)



Least Squares with L1 regularization

$$\min_x \|Ax - y\|_2^2 + \lambda \|x\|_1$$

L1 norm $\|x\|_1 = \sum_{i=1}^{d} |x_i|$

encourages solution x^* to be sparse

Least Squares with L1 regularization

$$\min_x \|Ax - y\|_2^2 + \lambda \|x\|_1$$

L1 norm $\|x\|_1 = \sum_{i=1}^d |x_i|$

encourages solution x^* to be sparse

algorithms: proximal gradient, interior point, ADMM (EE364a/b)

left-sketched $\min_x \|SAx - b\|_2^2 + \lambda \|x\|_1$

right-sketching can be done via the dual problem $\max_{\|A^T z\|_\infty \leq \lambda} -\|z - b\|_2^2$ using log-barrier (see EE364a/b)

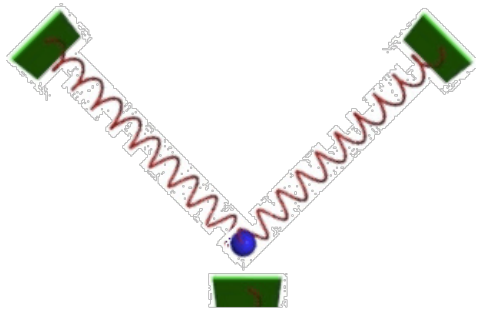
Least Squares with group L1 regularization

$$\min_x \| \sum_{i=1}^{k} A_i x_i - y \|_2^2 + \lambda \sum_{i=1}^{k} \|x_i\|_2$$

$\|x_i\|_2 = \sqrt{\sum_{j=1}^{d} x_{ij}^2}$

encourages solution x^* to be group sparse, i.e., most blocks x_i are zero

convex optimization and convex regularization methods are well understood



Two-Layer Neural Networks with Rectified Linear Unit (ReLU) activation

$$
\begin{aligned}
p_{\text{non-convex}} := \quad & \text{minimize} \quad L\left(\phi(XW_1)W_2, y\right) + \lambda\left(\|W_1\|_F^2 + \|W_2\|_F^2\right) \\
& W_1 \in \mathbb{R}^{d\times m} \\
& W_2 \in \mathbb{R}^{m\times 1}
\end{aligned}
$$

where $\phi(u) = \text{ReLU}(u) = \max(0, u)$

Neural Networks are Convex Regularizers

$$
\begin{aligned}
p_{\text{non-convex}} := \quad & \text{minimize} \quad L\left(\phi(XW_1)W_2, y\right) + \lambda\left(\|W_1\|_F^2 + \|W_2\|_F^2\right) \\
& W_1 \in \mathbb{R}^{d\times m} \\
& W_2 \in \mathbb{R}^{m\times 1} \\
p_{\text{convex}} := \quad & \text{minimize} \quad L\left(Z, y\right) + \lambda \underbrace{R(Z)}_{\substack{\text{convex regularization}}} \\
& Z \in \mathcal{K} \subseteq \mathbb{R}^{d\times p}
\end{aligned}
$$

$$p_{\text{non-convex}} := \begin{array}{ll} \text{minimize} & L\left(\phi(XW_1)W_2, y\right) + \lambda\left(\|W_1\|_F^2 + \|W_2\|_F^2\right) \\ & W_1 \in \mathbb{R}^{d\times m} \\ & W_2 \in \mathbb{R}^{m\times 1} \end{array}$$

$$p_{\text{convex}} := \begin{array}{ll} \text{minimize} & L\left(Z, y\right) + \lambda R(Z) \\ & Z \in \mathcal{K} \subseteq \mathbb{R}^{d\times p} \end{array}$$

Theorem $p_{\text{non-convex}} = p_{\text{convex}}$, and an optimal solution to $p_{\text{non-co}}$ can be obtained from an optimal solution to p_{convex}.

M. Pilanci, T. Ergen **Neural Networks are Convex Regularizers: Exact Polynomial-time Convex Optimization Formulations for Two-Layer Networks. ICML 2020**

Squared Loss

data matrix $X \in \mathbb{R}^{n \times d}$ and label vector $y \in \mathbb{R}^n$

$$X = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

$$p_{\text{non-convex}} = \min_{W_1, W_2} \left\| \sum_{j=1}^{m} \phi(XW_{1j}) W_{2j} - y \right\|_2^2 + \lambda \left(\|W_1\|_F^2 + \|W_2\|_F^2 \right)$$

$$p_{\text{convex}} = \min_{u_i, v_i \in \mathcal{K}} \left\| \sum_{i=1}^{p} D_i X (u_i - v_i) - y \right\|_2^2 + \lambda \sum_{i=1}^{p} \|u_i\|_2 + \|v_i\|_2$$

$D_1, ..., D_p$ are fixed diagonal matrices

Theorem $p_{\text{non-convex}} = p_{\text{convex}}$, and an optimal solution to $p_{\text{non-convex}}$ can be recovered from optimal non-zero u_i^*, v_i^* as $W_{1j}^* = \frac{u_i^*}{\sqrt{\|u_i^*\|_2}}$, $W_{2j} = \sqrt{\|u_i^*\|_2}$ or $W_{1j}^* = \frac{v_i^*}{\sqrt{\|v_i^*\|_2}}$, $W_{2j} = -\sqrt{\|v_i^*\|_2}$.

Regularization path

$$p_{\text{convex}} = \min_{u_1, v_1 \ldots u_p, v_p \in \mathcal{K}} \left\| \sum_{i=1}^{p} D_i X (u_i - v_i) - y \right\|_2^2 + \lambda \sum_{i=1}^{p} \|u_i\|_2 + \|v_i\|_2$$

As $\lambda \in (0, \infty)$ increases, the number of non-zeros in the solution decreases

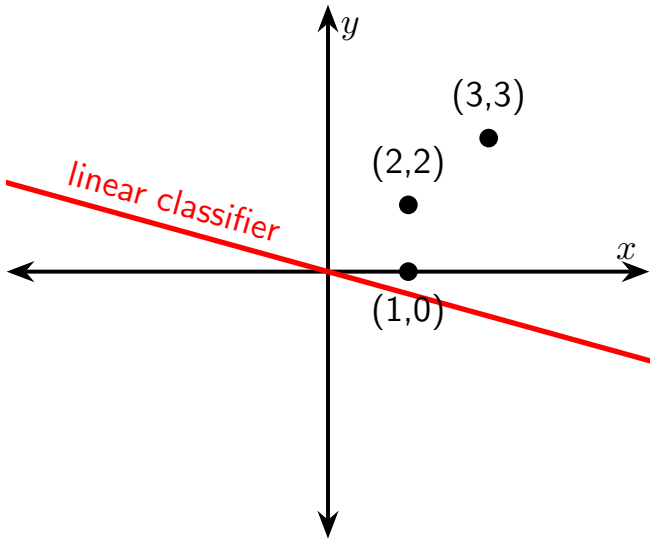
Theorem

Optimal solutions of p_{convex} generates the entire set of optimal architectures $f(x) = W_2 \phi(W_1 x)$ with m neurons for $m = 1, 2, ...,$

where $W_1 \in \mathbb{R}^{d \times m}$, $W_2 \in \mathbb{R}^{m \times 1}$

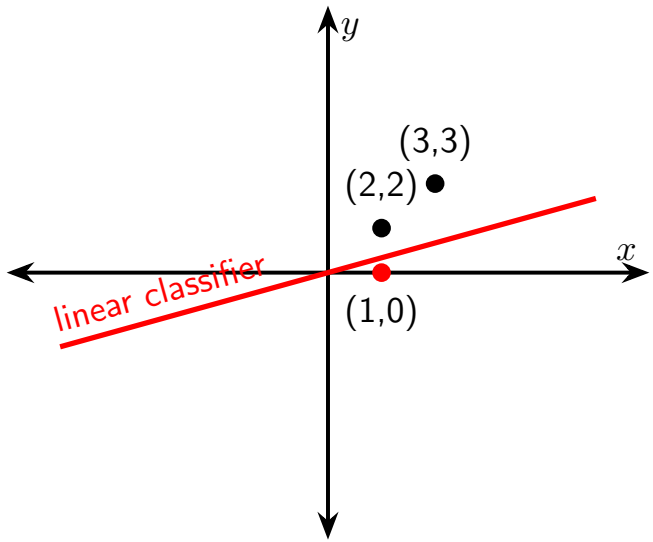
non-convex NN models actually correspond to regularized convex models

$n = 3$ samples in $\mathbb{R}^d$, $d = 2$ $\quad X = \begin{bmatrix} x_1^T \\ x_2^T \\ x_3^T \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 3 & 3 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$



$$D_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} X = \begin{bmatrix} 2 & 2 \\ 3 & 3 \\ 1 & 0 \end{bmatrix}$$

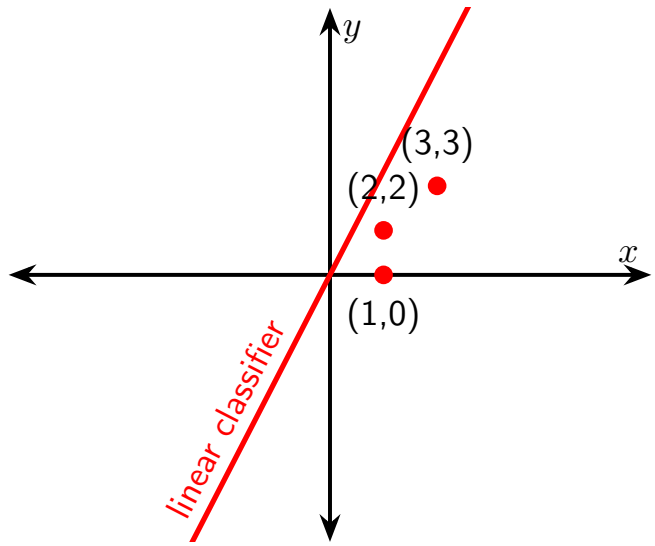
$n = 3$ samples in $\mathbb{R}^d$, $d = 2$ $\quad X = \begin{bmatrix} x_1^T \\ x_2^T \\ x_3^T \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 3 & 3 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$



$$D_1 X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} X = \begin{bmatrix} 2 & 2 \\ 3 & 3 \\ 1 & 0 \end{bmatrix}$$

$$D_2 X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} X = \begin{bmatrix} 2 & 2 \\ 3 & 3 \\ 0 & 0 \end{bmatrix}$$

$n = 3$ samples in $\mathbb{R}^d$, $d = 2$ $\quad X = \begin{bmatrix} x_1^T \\ x_2^T \\ x_3^T \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 3 & 3 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$

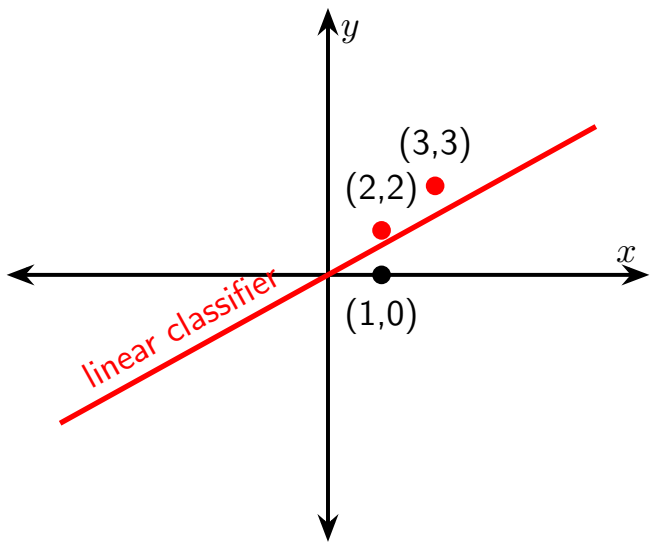


$$D_1 X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} X = \begin{bmatrix} 2 & 2 \\ 3 & 3 \\ 1 & 0 \end{bmatrix}$$

$$D_2 X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} X = \begin{bmatrix} 2 & 2 \\ 3 & 3 \\ 0 & 0 \end{bmatrix}$$

$$D_3 X = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} X = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$n = 3$ samples in $\mathbb{R}^d$, $d = 2$ $\quad X = \begin{bmatrix} x_1^T \\ x_2^T \\ x_3^T \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 3 & 3 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$



$$D_1 X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} X = \begin{bmatrix} 2 & 2 \\ 3 & 3 \\ 1 & 0 \end{bmatrix}$$

$$D_2 X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} X = \begin{bmatrix} 2 & 2 \\ 3 & 3 \\ 0 & 0 \end{bmatrix}$$

$$D_4 X = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} X = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}$$

Example: Convex Program for $n = 3, d = 2$

$$n = 3 \text{ samples} \quad X = \begin{bmatrix} x_1^T \\ x_2^T \\ x_3^T \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$$\min \left\| \begin{bmatrix} x_1^T \\ x_2^T \\ x_3^T \end{bmatrix} (u_1 - v_1) + \begin{bmatrix} x_1^T \\ x_2^T \\ 0 \end{bmatrix} (u_2 - v_2) + \begin{bmatrix} 0 \\ 0 \\ x_3^T \end{bmatrix} (u_3 - v_3) - y \right\|_2^2$$

subject to

$$+ \lambda \Big(\sum_{i=1}^{3} \|u_i\|_2 + \|v_i\|_2 \Big)$$

$$D_1 X u_1 \geq 0, D_1 X v_1 \geq 0$$

$$D_2 X u_2 \geq 0, D_2 X v_2 \geq 0$$

$$D_4 X u_3 \geq 0, D_4 X v_3 \geq 0$$

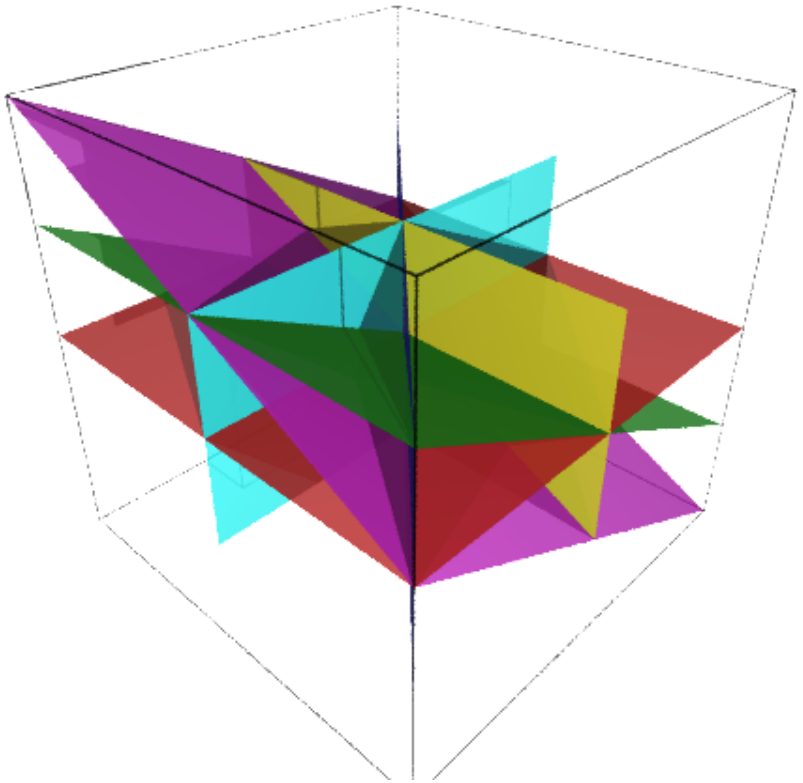
equivalent to the non-convex two-layer NN problem

Hyperplane Arrangements

Let $X \in \mathbb{R}^{n \times d}$

$$\{\mathbf{sign}(Xw) : w \in \mathbb{R}^d\}$$

at most $2\sum_{k=0}^{r-1} \binom{n}{k} \leq O\left(\left(\frac{n}{r}\right)^r\right)$ patterns where $r = \mathbf{rank}(X)$.

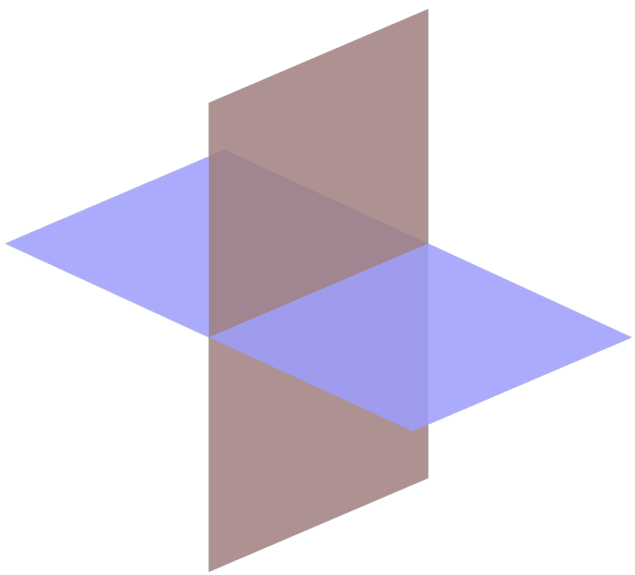


Convolutional Hyperplane Arrangements

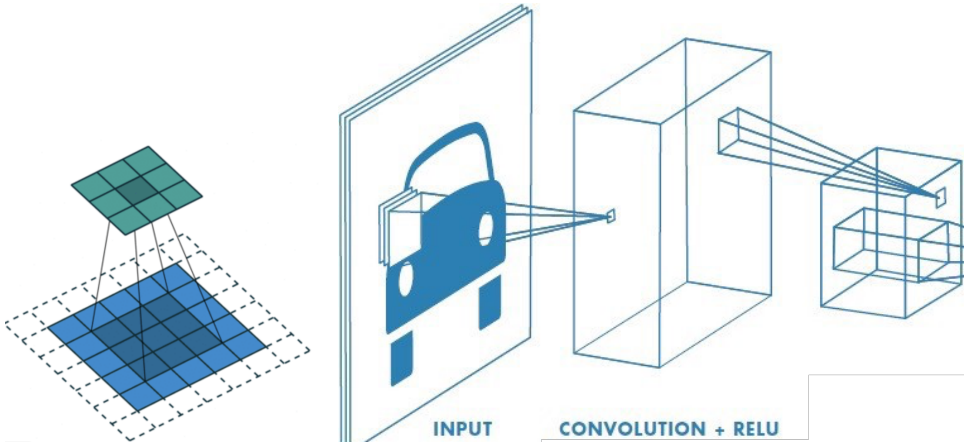
Let $X \in \mathbb{R}^{n \times d}$ be partitioned into patch matrices $X = [X_1, ..., X_K]$ where $X_k \in \mathbb{R}^{n \times h}$

$$\{\mathbf{sign}(X_k w) : w \in \mathbb{R}^h\}_{k=1}^K$$

at most $O\big((\frac{nK}{h})^h\big)$ patterns where h is the filter size.



Convolutional Neural Networks can be optimized in fully polynomial time



$f(x) = W_2\phi(W_1 x)$, $W_1 \in \mathbb{R}^{d\times m}$, $W_2 \in \mathbb{R}^{m\times 1}$

m filters (neurons), h filter size

typical example: 1024 filters of size 3×3 $(m = 1024, h = 9)$

convex optimization complexity: polynomial in all parameters n, m and d

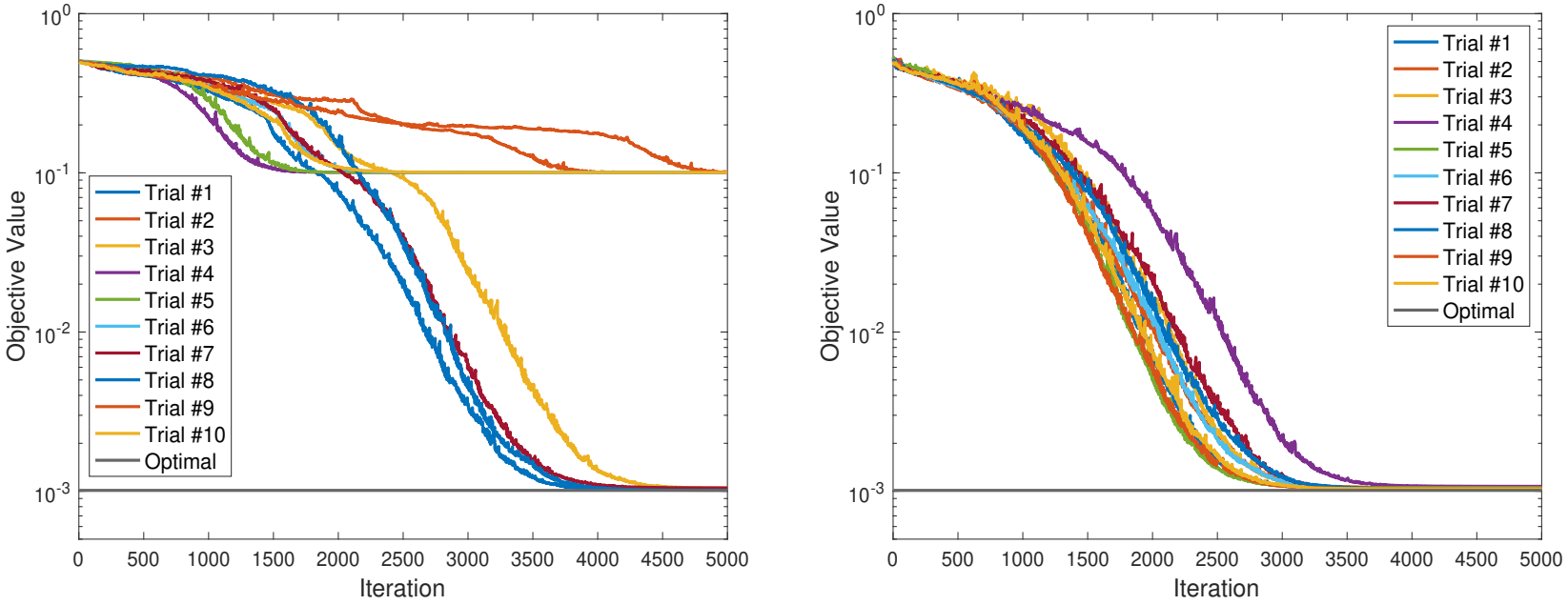
Approximating the Convex Program

$$\min_{u_1,v_1 \ldots u_p,v_p \in \mathcal{K}} \Big\| \sum_{i=1}^{p} D_i X(u_i - v_i) - y \Big\|_2^2 + \lambda \big(\sum_{i=1}^{p} \|u_i\|_2 + \|v_i\|_2 \big)$$

- Sample $D_1, ..., D_p$ as Diag$(Xu \geq 0)$ where $u \sim N(0, I)$
- Right-sketching applied to the original problem with the data matrix $[D_1 X, D_2 X, ..., D_P X]$ using non-uniform sampling probabilities
- Backpropagation (gradient descent) on the non-convex loss

  is a **heuristic** for the convex program

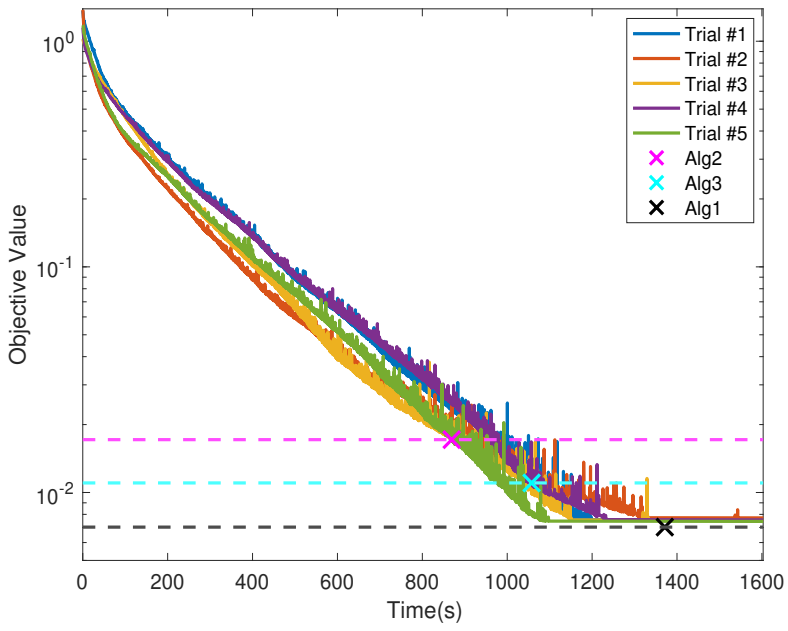
Numerical Results: Interior Point (Log Barrier) Method for Two-Layer Fully Connected ReLU



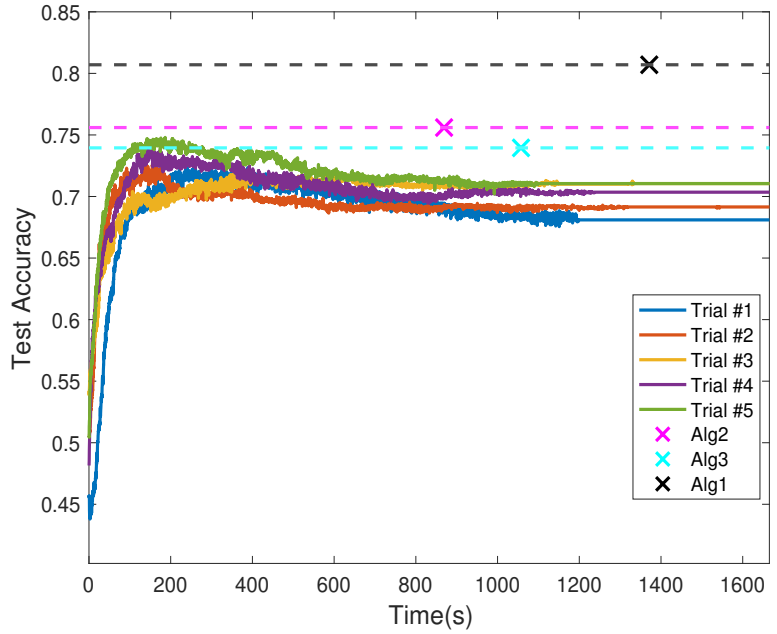
(a) $m = 8$ **(b)** $m = 50$

Figure 1: Training cost of a two-layer ReLU network trained with SGD (10 initialization trials) and the convex program solved with interior point method (Optimal) on a small dataset ($d = 2$)

Two-Layer Fully Connected ReLU on CIFAR-10



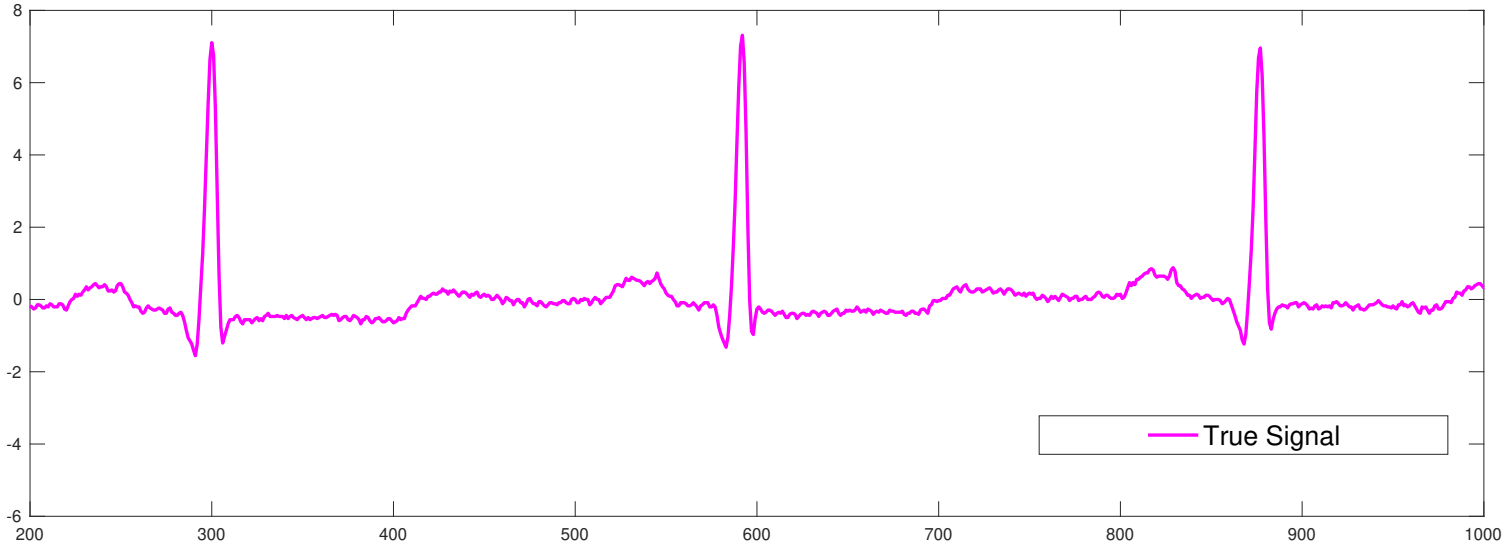
(a) $m = 8$



(b) $m = 50$

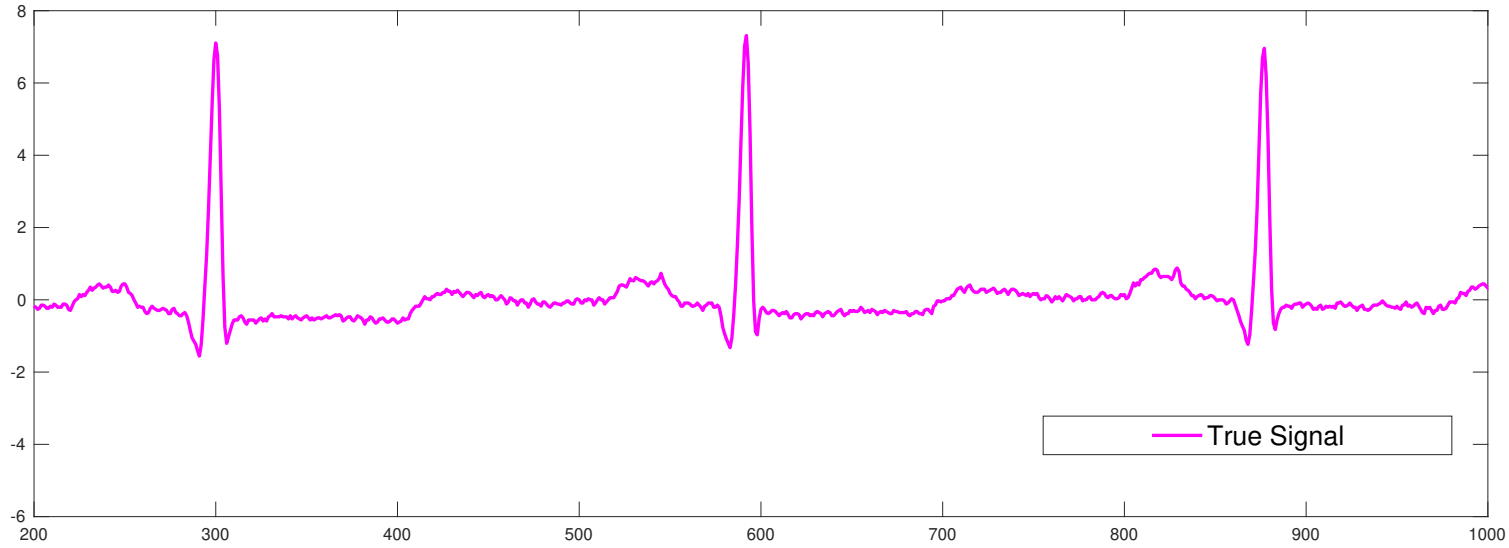
Figure 2: Two-layer ReLU network trained with SGD (10 initialization trials) and the convex program solved with interior point method on CIFAR-10 for binary classification ($n = 200$)

Application: Electrocardiogram (ECG) Prediction



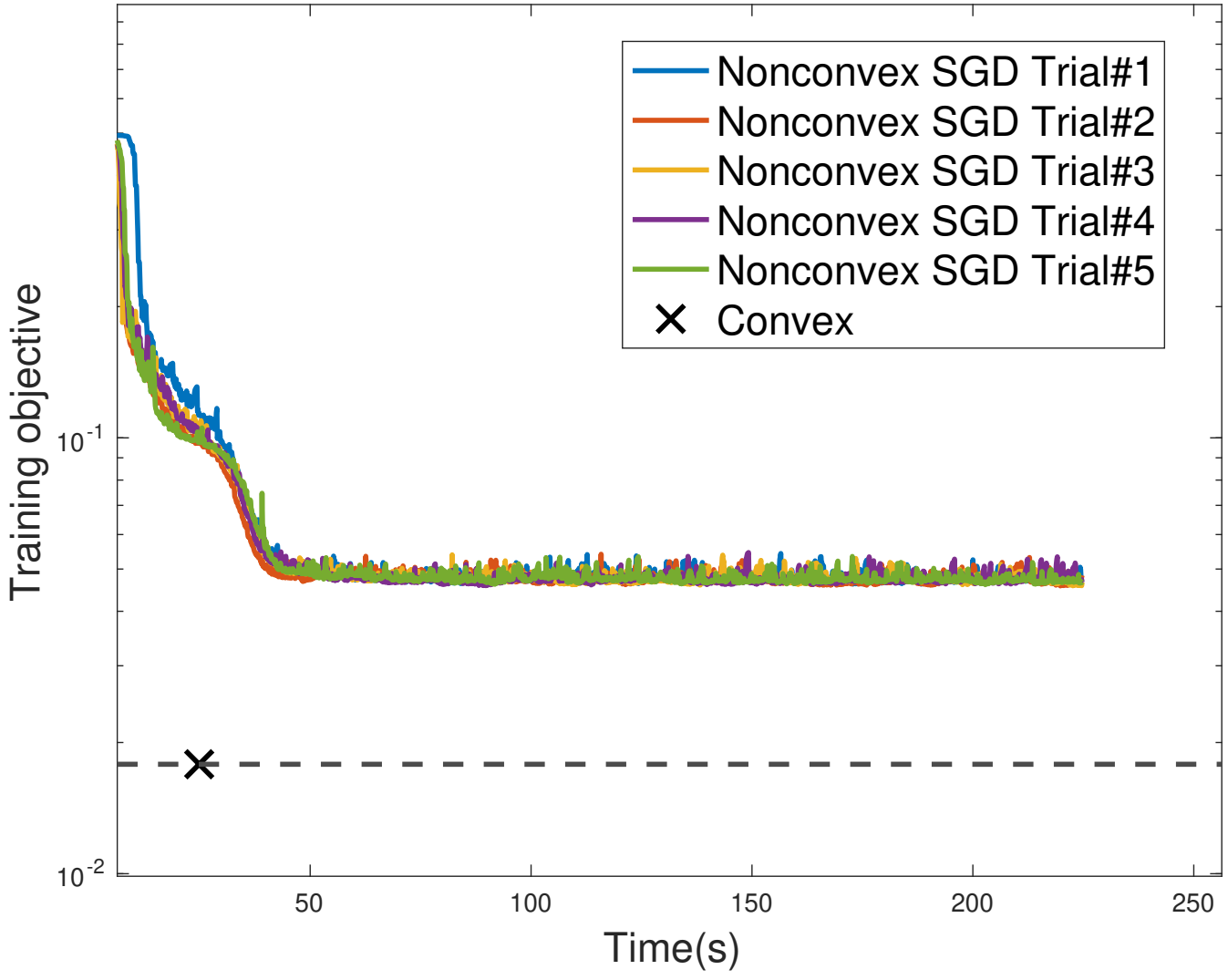
window size: 15 samples

training and test set

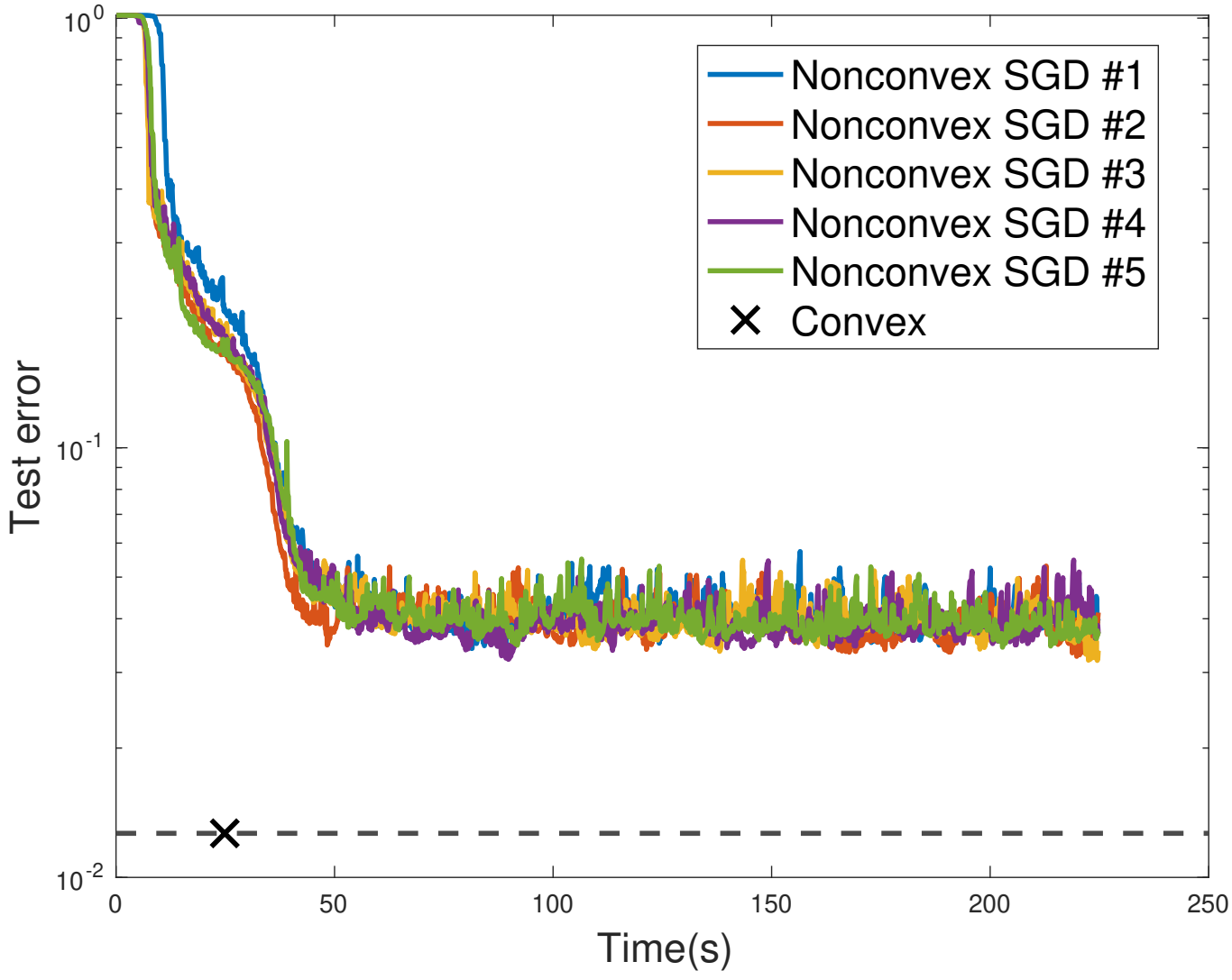


$$X = \begin{bmatrix} x[1] & \dots & x[d] \\ x[2] & \dots & x[d+1] \\ \vdots & & \\ x[n] & \dots & x[d+n-1] \end{bmatrix}, \quad y = \begin{bmatrix} x[d+1] \\ x[d+2] \\ \vdots \\ x[d+n] \end{bmatrix}$$

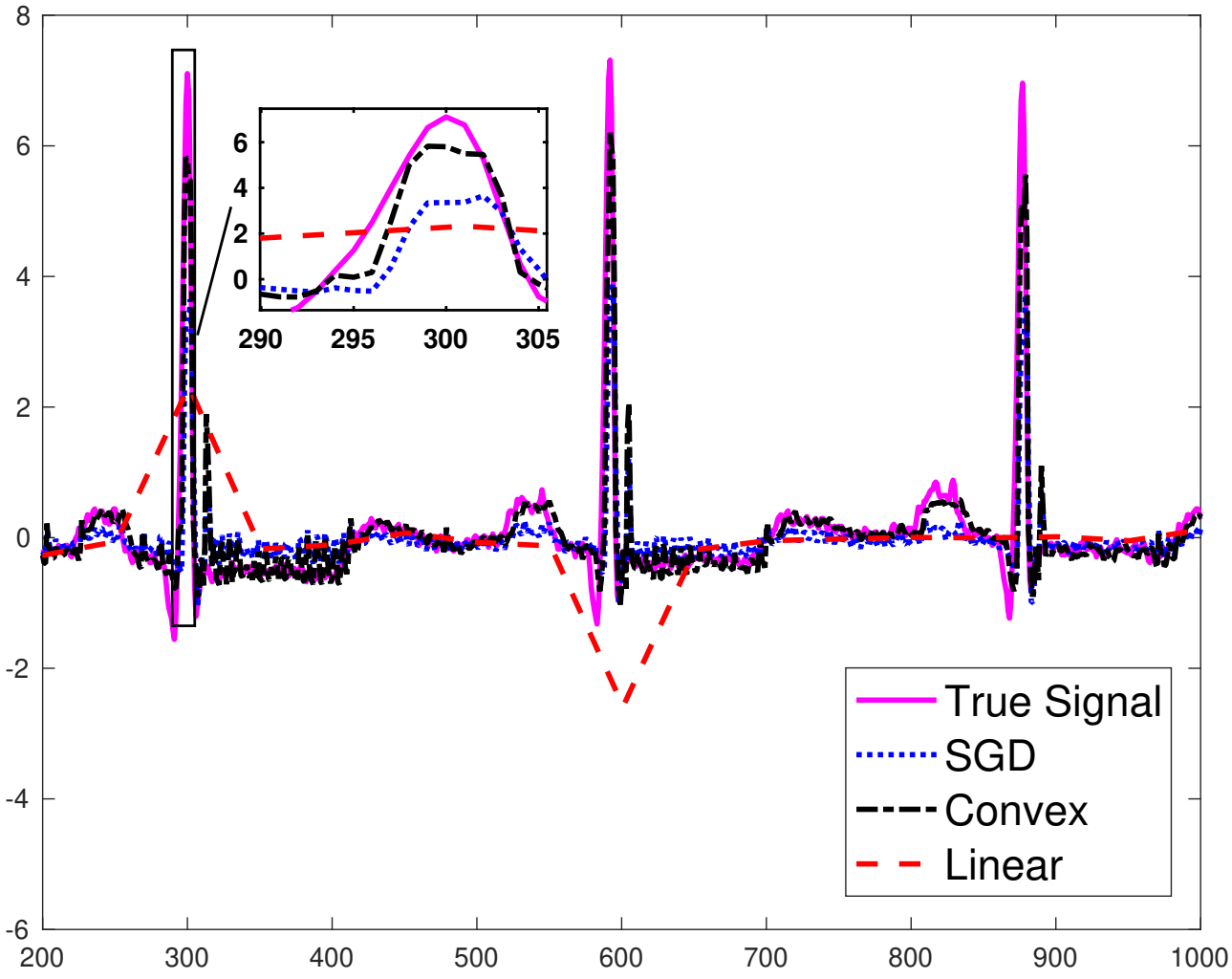
Signal Prediction: Training



Signal Prediction: Test Accuracy

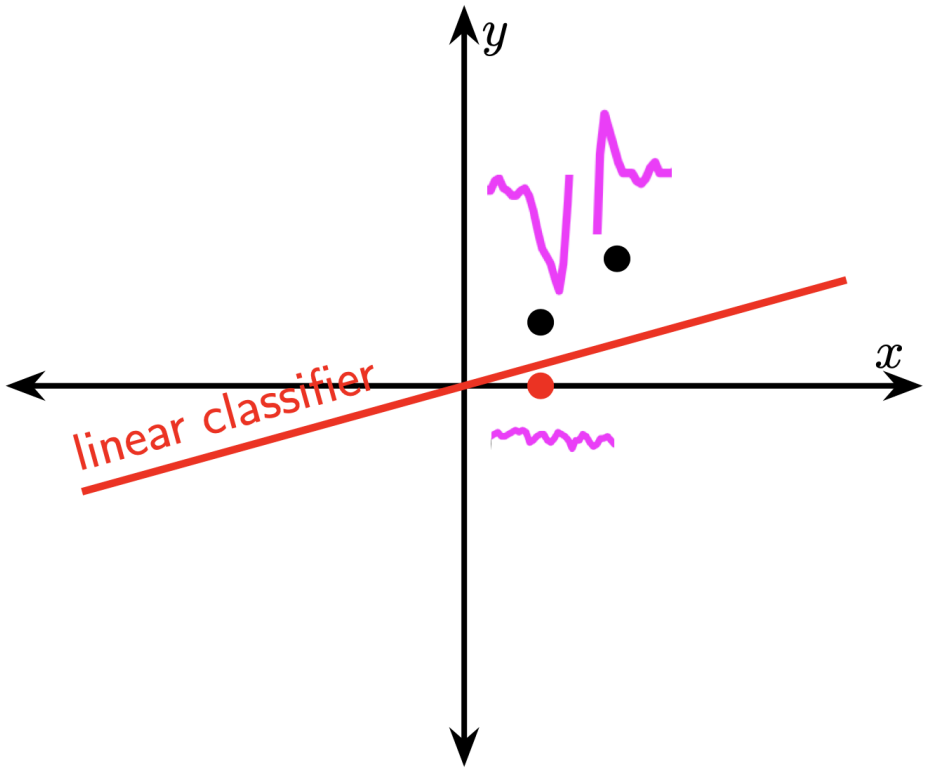


Signal Prediction: Test Accuracy



Neural Networks are Explainable

$$\min_{u_1,v_1 \ldots u_p,v_p \in \mathcal{K}} \left\| \sum_{i=1}^{p} D_i X (u_i - v_i) - y \right\|_2^2 + \lambda \sum_{i=1}^{p} \|u_i\|_2 + \|v_i\|_2$$



Convex Optimization for Neural Networks

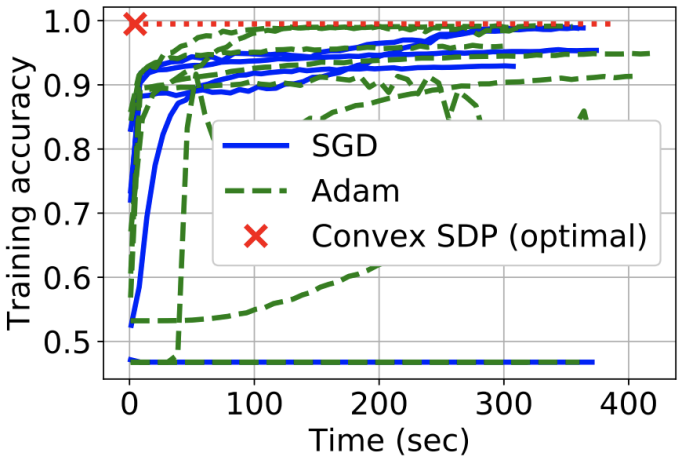
Polynomial activation function $\phi(t) = at^2 + bt + c$

$$
\begin{aligned}
p_{\text{non-convex}} := \quad & \text{minimize}_{\|W_{1i}\|_2=1,\forall i} \quad L\left(\phi(XW_1)W_2, y\right) + \lambda\|W_2\|_1 \\
& W_1 \in \mathbb{R}^{d\times m} \\
& W_2 \in \mathbb{R}^{m\times 1} \\
p_{\text{convex}} := \quad & \text{minimize}_{Z} \quad L\left(Z, y\right) + \lambda \underbrace{R(Z)}_{\text{convex regularization}} \\
& Z \in \mathcal{K} \subseteq \mathbb{R}^{d\times p}
\end{aligned}
$$

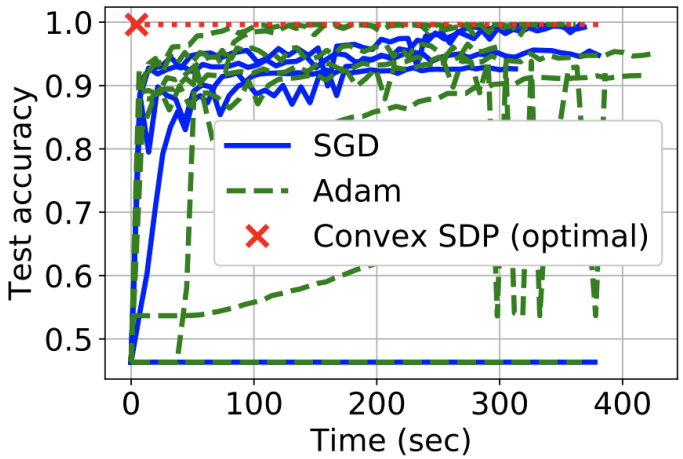
convex semidefinite program, can be solved in time polynomial in all problem parameters (n, d, m)

- B. Bartan, M. Pilanci *Neural Spectrahedra and Semidefinite Lifts: Global Convex Optimization of Polynomial Activation Neural Networks, 2021*

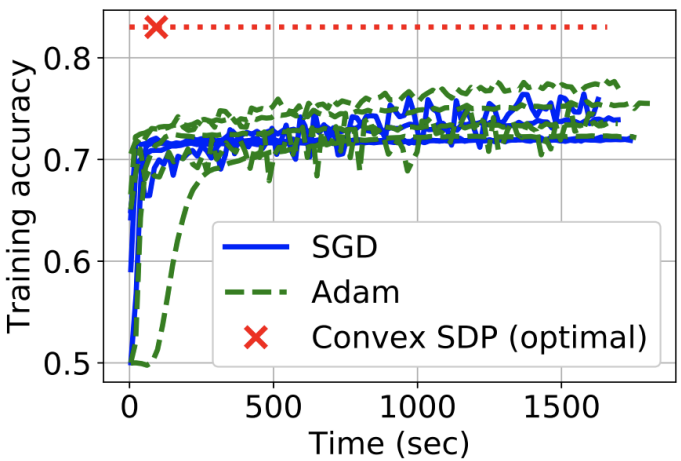
Numerical Results: Polynomial Activation

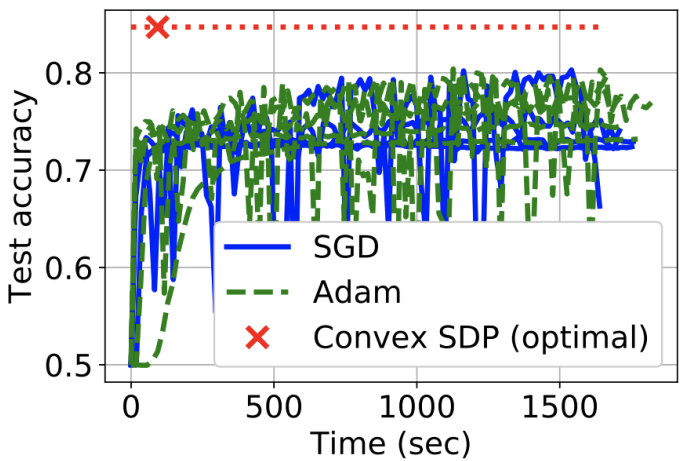


(a) CNN, MNIST, training accuracy



(b) CNN, MNIST, test accuracy





Other Relevant Courses

EE364a - Convex Optimization I (Winter)

EE364b - Convex Optimization II (Spring)

CS265 - Randomized Algorithms and Probabilistic Analysis

CS 229 - Machine Learning

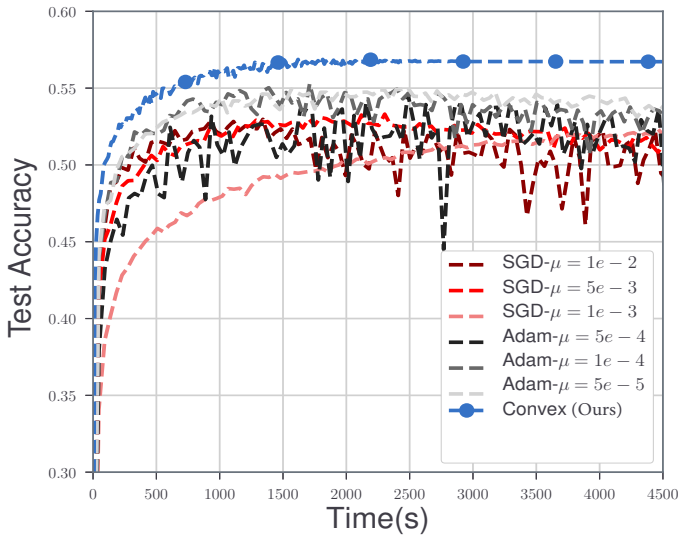
CS 230 - Deep Learning

CS369G: Algorithmic Techniques for Big Data

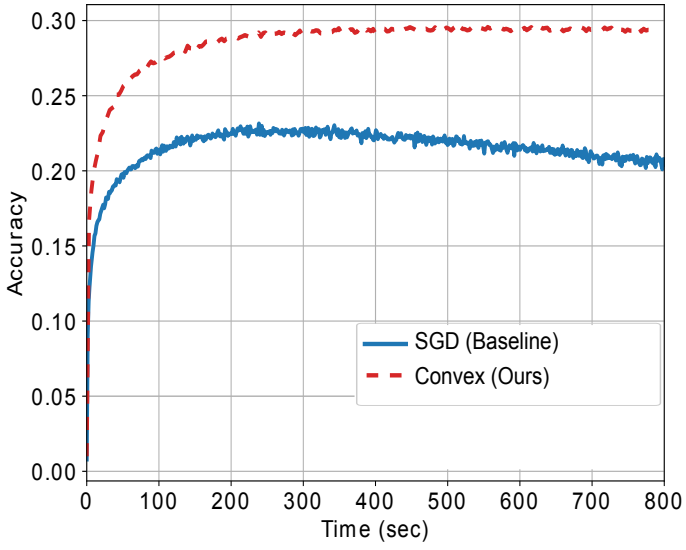
Math 301: Advanced Topics in Convex Optimization

EE269 - Signal Processing for Machine Learning (Fall)

SGD for Convex vs Non-convex Neural Networks on CIFAR-10 and CIFAR-100



(a) CIFAR-10



(b) CIFAR-100

Figure 3: Test accuracy of two-layer ReLU networks: Nonconvex vs convex formulation on 10 class and 100 class image classification problems (50,000 samples in dimension 3072)