Convex Optimization for Neural Networks

- neural networks
- convex optimization formulations of neural networks
- semi-infinite optimization problems
- numerical examples
Neural Network Timeline

- **1940**: Perceptron
- **1950**: ADALINE
- **1960**: XOR Problem
- **1970**: Golden Age
- **1980**: Multi-layered Perceptron (Backpropagation)
- **1990**: SVM
- **2000**: Deep Neural Network (Pretraining)

**Key Events and Innovations**

- **1943**: S. McCulloch – W. Pitts
- **1957**: F. Rosenblatt
- **1960**: B. Widrow – M. Hoff
- **1969**: M. Minsky – S. Papert
- **1986**: D. Rumelhart – G. Hinton – R. Williams
- **1995**: V. Vapnik – C. Cortes
- **2006**: G. Hinton – S. Ruslan

**Innovations**

- Adjusted Weights
- Learnable Weights and Threshold
- XOR Problem
- Solution to non-linearly separable problems
- Big computation, local optima and overfitting
- Limitations of learning prior knowledge
- Hierarchical feature learning
- Kernel function: Human Intervention
Deep learning revolution

ImageNet Classification, top-5 error (%)

ILSVRC 2010 NEC America: 28.2
ILSVRC 2011 Xerox: 25.8
ILSVRC 2012 AlexNet: 16.4
8 layers
ILSVRC 2013 Clarifi: 11.7
8 layers
ILSVRC 2014 VGG: 7.3
19 layers
ILSVRC 2014 GoogleNet: 6.7
22 layers
ILSVRC 2015 ResNet: 3.5
152 layers!
Multilayer Neural Networks

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

\[ a_{j}^{(l)} = \sum_{i} W_{i,j}^{(l)} z_{i}^{(l-1)} \quad l = 1, \ldots, L \]

\[ z_{j}^{(l)} = \sigma(a_{j}^{l}) \quad l = 1, \ldots, L \]

• \( \sigma(\cdot) \): activation function, \( a_{j}^{l} \): pre-activation of neuron \( j \) at layer \( l \)
Training Multilayer Neural Networks

- parameters $\Theta = (W^{(1)}, W^{(2)}, \ldots, W^{(L)})$

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

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

$$\min_{\Theta} - \sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \log f_k(x_n)$$

- (Stochastic) Gradient Descent

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

- non-convex optimization problem
Computing derivatives: Backpropagation Algorithm

\[
\min_{\Theta} \sum_{n=1}^{N} \left( y_n - f(x_n) \right)^2
\]

define \( \delta^{(l)}_{n,j} \triangleq \frac{\partial R_n(\Theta)}{\partial a_j^{(l)}} \), which are the derivatives of the loss with respect to the pre-activations

then gradients can be computed from

\[
\frac{\partial R_n(\Theta)}{\partial W_{ij}^{(l)}} = \frac{\partial R_n(\Theta)}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial W_{ij}^{(l)}} = \delta_{n,j}^{(l)} z_i^{(l-1)}
\]
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 the definition

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

• at the output layer \( \delta_{nj}^{(L)} = 2(a^{(L)} - y_n) \) since \( R_n(\Theta) = \|a^{(L)} - y_n\|^2 \)
Other Optimization Methods

▶ Stochastic Gradient Descent with momentum

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

- \( \alpha \) is the step size (learning rate), e.g., \( \alpha = 0.1 \) and \( \rho \) is the momentum parameter, e.g., \( \rho = 0.9 \)

- \( \nabla f(x_t) \) can be replaced with a subgradient for non-differentiable functions

- 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)

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

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
Other Variations of Diagonal Hessian Approximations

- **RMSProp**, Tieleman and Hinton, 2012

  \[ H_{t+1} = \text{diag}((s_{t+1} + \delta)^{1/2}) \]

  weighted gradient squared update

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

- **ADAM**, Kingma and Ba, 2015

  includes 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

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

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

• Block-diagonal approximations

• Kronecker-factored Approximate Curvature (KFAC), Martens and Grosse, 2015

• Uniform or weighted sampling

• Conjugate Gradient can be used to approximate the Gauss-Newton step
Limitations of Neural Networks and Non-convex Training

- sensitive to initialization, step-sizes, mini-batching, and the choice of the optimizer
- challenging to train and requires babysitting
- neural networks are complex black-box systems
- hard to interpret what the model is actually learning
Advantages of Convex Optimization

- Convex optimization provides a globally optimal solution
- Reliable and efficient solvers
- Specific solvers and internal parameters, e.g., initialization, step-size, batch-size does not matter
- We can check global optimality via KKT conditions
- Dual problem provides a lower-bound and an optimality gap
- Distributed and decentralized methods are well-studied
Example: Least Squares

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

- well-studied convex optimization problem
- many efficient numerical procedures exist: Conjugate Gradient (CG), Preconditioned CG, QR, Cholesky, SVD
- regularized form $\min_x \|Ax - b\|_2^2 + \lambda \|x\|_2^2$, i.e., Ridge Regression is widely used
L2 regularization: mechanical model

\[
\min_x \frac{1}{2} (x - y)^2 + \frac{1}{2} \lambda x^2
\]

- \text{elastic energy}
- \text{elastic energy}

\text{red spring constant} = 1

\text{blue spring constant} = \lambda
**L1 regularization: mechanical model**

\[
\min_x \frac{1}{2} (x - y)^2 + \frac{\lambda}{2} |x|
\]

- **elasticsearch energy**
- **potential energy**

- **red spring constant** = 1
- **blue ball mass** = \( \lambda \) (small)
**L1 regularization: mechanical model with large $\lambda$**

$$\min_x \frac{1}{2} (x - y)^2 + \lambda |x|$$

- elastic energy
- potential energy

- red spring constant $= 1$
- blue ball mass $= \lambda$ (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 sparsity of the solution $x^*$

- many efficient algorithms exist: proximal gradient (PG), accelerated PG, interior point, ADMM
Least Squares with group L1 regularization

\[
\min_x \left\| \sum_{i=1}^k A_i x_i - y \right\|^2_2 + \lambda \sum_{i=1}^k \|x_i\|^2_2
\]

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

encourages group sparsity in the solution \( x^* \), i.e., most blocks \( x_i \) are zero

- convex optimization and convex regularization methods are well understood and widely used in machine learning and statistics
Two-Layer Neural Networks with Rectified Linear Unit (ReLU) activation

\[ p_{\text{non-convex}} := \min \quad L(\sigma(XW_1)W_2, y) + \lambda \left( \|W_1\|_F^2 + \|W_2\|_F^2 \right) \]

\[ W_1 \in \mathbb{R}^{d \times m} \]

\[ W_2 \in \mathbb{R}^{m \times 1} \]

\[ \sigma(u) = \text{ReLU}(u) = \max(0, u) \]
Neural Networks are Convex Regularizers

\[ p_{\text{non-convex}} := \min \ L(\sigma(XW_1)W_2, y) + \lambda \left( \|W_1\|_F^2 + \|W_2\|_F^2 \right) \]

\[ W_1 \in \mathbb{R}^{d \times m} \]

\[ W_2 \in \mathbb{R}^{m \times 1} \]

\[ p_{\text{convex}} := \min \ L(Z, y) + \lambda \underbrace{R(Z)}_{\text{convex regularization}} \]

\[ Z \in \mathcal{K} \subseteq \mathbb{R}^{d \times p} \]
Neural Networks are Convex Regularizers

\[ p_{\text{non-convex}} := \min_{W_1, W_2} L(\sigma(XW_1)W_2, y) + \lambda (\|W_1\|_F^2 + \|W_2\|_F^2) \]

\[ W_1 \in \mathbb{R}^{d \times m} \]
\[ W_2 \in \mathbb{R}^{m \times 1} \]

\[ p_{\text{convex}} := \min_{Z} L(Z, y) + \lambda R(Z) \]
\[ Z \in \mathcal{K} \subseteq \mathbb{R}^{d \times p} \]

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

M. Pilanci, T. Ergen Neural Networks are Convex Regularizers: Exact Polynomial-time Convex Optimization Formulations..., ICML 2020
Two Layer Networks Trained with Squared Loss

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

$$p_{\text{non-convex}} = \min_{W_1, W_2} \left\| \sum_{j=1}^{m} \sigma(XW_1 j)W_2 j - y \right\|_2^2 + \lambda \left( \|W_1\|_F^2 + \|W_2\|_F^2 \right)$$

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

here $D_1, ..., D_p$ 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

$$W_{1i}^* = \frac{u_i^*}{\sqrt{\|u_i^*\|_2}}$$

$$W_{2i}^* = \frac{v_i^*}{\sqrt{\|v_i^*\|_2}}$$

or

$$W_{1i}^* = \frac{v_i^*}{\sqrt{\|v_i^*\|_2}}$$

$$W_{2i}^* = -\frac{u_i^*}{\sqrt{\|u_i^*\|_2}}$$

EE346b, Stanford University 23
Regularization path

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

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

- optimal solutions of \( p_{\text{convex}} \) generates the entire set of optimal architectures \( f(x) = W_2 \sigma (W_1 x) \) with \( m \) neurons for \( m = 1, 2, \ldots \),

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

- non-convex NN models correspond to regularized convex models!
Simple Example

\[ n = 3 \text{ samples in } \mathbb{R}^d, \ d = 2 \]

\[ X = \begin{bmatrix}
  x_1^T \\
  x_2^T \\
  x_3^T
\end{bmatrix} = \begin{bmatrix}
  2 & 2 \\
  3 & 3 \\
  1 & 0
\end{bmatrix}, \quad y = \begin{bmatrix}
  y_1 \\
  y_2 \\
  y_3
\end{bmatrix} \]

\[ D_1 = \begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix} \]

\[ X = \begin{bmatrix}
  2 & 2 \\
  3 & 3 \\
  1 & 0
\end{bmatrix} \]
Simple Example

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

\[
D_1X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} X = \begin{bmatrix} 2 & 2 \\ 3 & 3 \\ 1 & 0 \end{bmatrix}
\]

\[
D_2X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} X = \begin{bmatrix} 2 & 2 \\ 3 & 3 \\ 0 & 0 \end{bmatrix}
\]
Simple Example

\[
D_1 X = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix} \quad X = \begin{bmatrix}
2 & 2 \\
3 & 3 \\
1 & 0 \\
\end{bmatrix}
\]

\[
D_2 X = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \quad X = \begin{bmatrix}
2 & 2 \\
3 & 3 \\
0 & 0 \\
\end{bmatrix}
\]

\[
D_3 X = \begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \quad X = \begin{bmatrix}
0 & 0 \\
0 & 0 \\
\end{bmatrix}
\]
Simple Example

\[
D_1 X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad X = \begin{bmatrix} 2 & 2 \\ 3 & 3 \\ 1 & 0 \end{bmatrix}
\]

\[
D_2 X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad X = \begin{bmatrix} 2 & 2 \\ 3 & 3 \\ 0 & 0 \end{bmatrix}
\]

\[
D_3 X = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad X = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}
\]

\[
D_4 X = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad X = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}
\]
Convex Program for $n = 3, d = 2$

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

subject to

\[D_1 X u_1 \geq 0, D_1 X v_1 \geq 0\]
\[D_2 X u_2 \geq 0, D_2 X v_2 \geq 0\]
\[D_4 X u_3 \geq 0, D_4 X v_3 \geq 0\]

equivalent to the non-convex two-layer NN problem
Hyperplane Arrangements

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

• $D_1, \ldots, D_P$ are diagonal $0-1$ matrices that encode patterns

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

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

ReLU neural networks with \( m \) neurons

\[
f(x) = \sum_{j=1}^{m} W_{2j} \phi(W_{j1}x)
\]

Previous results:

- Combinatorial \( O(2^m n^d m) \) (Arora et al., ICLR 2018)
- Approximate \( O(2^{\sqrt{m}}) \) (Goel et al., COLT 2017)

Convex program \( O\left((\frac{n}{r})^r\right) \) where \( r = \text{rank}(X) \)

\( n \): number of samples, \( d \): dimension

(i) polynomial in \( n \) and \( m \) for fixed rank \( r \)

(ii) exponential in \( d \) for full rank data \( r = d \). This can not be improved unless \( P = NP \) even for \( m = 1 \).
Convolutional Hyperplane Arrangements

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

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

at most $O\left((\frac{nK}{h})^h\right)$ patterns where $h$ is the filter size.
Convolutional Neural Networks can be optimized in fully polynomial time

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

- \( m \) filters (neurons), \( h \) filter size, e.g., 1024 filters of size \( 3 \times 3 \) (\( m = 1024, h = 9 \))

- convex optimization complexity is polynomial in all parameters \( n, m \) and \( d \)
Approximating the Convex Program

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

• sample \( D_1, ..., D_p \) as \( \text{Diag}(Xu \geq 0) \) where \( u \sim N(0, I) \)

• Backpropagation (gradient descent) on the non-convex loss

    is a **heuristic** for the convex program
Numerical Results

- backpropagation converges to a stationary point of the loss
- convex optimization formulation returns the globally optimal neural network
- note that the number of variables is larger in the convex formulation
- interior point method, proximal gradient and ADMM are very effective
- proximal map of the group $\ell_1$ regularizer is closed-form
Interior Point Method vs Non-convex SGD

**Figure 1:** $m = 8$
SGD (10 different initializations) vs the convex program solved with interior point method (optimal) on a toy dataset ($d = 2$)

**Figure 2:** $m = 50$

Convex SGD vs Non-convex SGD and ADAM

Figure 3: CIFAR-10
CIFAR image classification task \((n = 50000, d = 3072)\)

Figure 4: CIFAR-100
Polynomial Activation Networks

- polynomial activation function $\sigma(t) = at^2 + bt + c$

$$p_{\text{non-convex}} := \min_{\|W_1\|_2=1, \forall i} L(\sigma(XW_1)W_2, y) + \lambda\|W_2\|_1$$

$W_1 \in \mathbb{R}^{d \times m}$

$W_2 \in \mathbb{R}^{m \times 1}$

$$p_{\text{convex}} := \min_{Z} L(Z, y) + \lambda \left\langle R(Z) \right\rangle_{\text{convex regularization}}$$

$Z \in \mathcal{K} \subseteq \mathbb{R}^{d \times p}$

- **Theorem:** $p_{\text{convex}} = p_{\text{non-convex}}$ and can be solved via a convex semidefinite program in polynomial-time with respect to $(n, d, m)$.

B. Bartan, M. Pilanci Neural Spectrahedra and Semidefinite Lifts 2021
Polynomial Activation Networks

• special case: quadratic activation \( \sigma(t) = t^2 \)

\[
p_{\text{convex}} := \min_Z L(Z, y) + \lambda \|Z\|_* \quad \quad Z \in \mathcal{K} \subseteq \mathbb{R}^{d \times p}
\]

• \( \|Z\|_* \) is the nuclear norm

• promotes low rank solutions

• first and second layer weights can be recovered via Eigenvalue Decomposition \( Z = \sum_{i=1}^{m} \alpha_i u_i u_i^T \)
Polynomial Activation Networks

- polynomial activation function

\[ \phi(t) = at^2 + bt + c \]

\[
\begin{align*}
\min_Z & \quad L(\hat{y}, y) + \lambda Z_4 \\
\text{s.t.} & \quad \hat{y}_i = ax_i^T Z_1 x_i + bx_i^T Z_2 + c Z_4, i \in [n] \\
Z & = \begin{bmatrix} Z_1 & Z_2 \\ Z_2^T & Z_4 \end{bmatrix} \succeq 0, \ \text{tr}(Z_1) = Z_4,
\end{align*}
\]
Numerical Results: Quadratic Activation

- toy dataset $n = 100$, $d = 10$

- $m = 10$ planted neurons

- red cross marker shows the time taken by the convex solver
Numerical Results: Polynomial Activation

(a) CNN, MNIST, training accuracy

(b) CNN, MNIST, test accuracy

(c) CNN, CIFAR, training accuracy

(d) CNN, CIFAR, test accuracy
Deriving the convex program

- ReLU activation $\sigma(t) = (t)_+$ and weight decay regularization

$$p_{\text{non-convex}} = \min_{W_1} \min_{W_2} L(\sigma(XW_1)W_2, y) + \lambda \left( \|W_1\|_F^2 + \|W_2\|_F^2 \right)$$

- nested minimization problems

- inner minimization over $W_2$ is convex for fixed $W_1$

- not jointly convex in $(W_1, W_2)$
Scaling Variables

- **Lemma:** The weight decay ($\ell^2_2$) regularized non-convex program is equivalent to an $\ell_1$ penalized non-convex program

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

$$= \min_{\|W_{1j}\|_2=1, W_2} \frac{1}{2} \left\| \sum_{j=1}^{m} \phi(X W_{1j}) W_{2j} - y \right\|^2_2 + \lambda \sum_{j=1}^{m} |W_{2j}|$$
Scaling Variables

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

- we define

\[ \tilde{W}_{1j} := \frac{W_{1j}}{\alpha_j} \]
\[ \tilde{W}_{2j} := W_{2j} \alpha_j \]

- neural network output does not change, regularization term changes
Scaling Variables

• plugging-in we get

$$\min_{\alpha_j} \min_{W_1, W_2} \frac{1}{2} \left\| \sum_{j=1}^{m} \phi(XW_1^j)W_2^j - y \right\|_2^2 + \lambda \left( \sum_{j=1}^{m} \frac{\|W_1^j\|_2^2}{\alpha_j^2} + |W_2^j| \alpha_j^2 \right)$$

• optimize with respect to $\alpha_1, \ldots, \alpha_m$

• we obtain

$$\min_{\|W_1^j\|_2=1, W_2} \frac{1}{2} \left\| \sum_{j=1}^{m} \phi(XW_1^j)W_2^j - y \right\|_2^2 + \lambda \sum_{j=1}^{m} |W_2^j|$$
Convex Duality of Neural Networks

- Replace the inner minimization problem by its convex dual

\[
p_{\text{non-convex}} = \min_{\|W_1\|_2=1} \min_{W_2} \frac{1}{2} \left\| \sum_{j=1}^{m} \phi(XW_1j)W_2j - y \right\|^2_2 + \lambda \sum_{j=1}^{m} |W_{2j}|
\]

= \min_{\|W_1\|_2=1} \max_{v: |v^T(XW_1j)_+| \leq \lambda \forall j} \left( -\frac{1}{2} \left\| v - y \right\|^2_2 \right)

= \min_{\|W_1\|_2=1} \max_{v} \left( -\frac{1}{2} \left\| v - y \right\|^2_2 + I\left( |v^T(XW_1j)_+| \leq \lambda \forall j \right) \right)

- \( I(\cdot) \) is the \(-\infty/0\) valued indicator function

- interchange the order of \( \min \) and \( \max \)
Convex Duality of Neural Networks

• by weak duality

\[
p_{\text{non-convex}} \geq \max_{v: v^T (XW_1) + |\leq \lambda} \forall W_1: \|W_1\|_2 = 1, \forall j \frac{1}{2} \|v - y\|_2
\]

\[
= \max_{v: v^T (XW) + |\leq \lambda} \forall W: \|W\|_2 = 1 \frac{1}{2} \|v - y\|_2
\]

• note that this is a convex optimization problem

• semi-infinite program: infinitely many constraints and finitely many variables

• it turns out that strong duality holds (Pilanci and Ergen, ICML 2020), i.e., the inequality is in fact an equality
Representing constraints

• finally, we can represent the constraints as

\[ |v^T (XW)_+| \leq \lambda \iff |v^T D_k X W_k| \leq \lambda \]

\[ D_k X W_k \geq 0 \]

\[ (I - D_k) X W_k \geq 0 \]

\(D_k\) are diagonal 0/1 matrices that encode hyperplane arrangements, i.e., sign patterns that can be obtained by \(\text{sign}(XW)\) for \(W \in \mathbb{R}^d\)
Bidual Problem

• the dual of the dual yields claimed convex neural network problem

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

• exact representation since strong duality holds

\[
p_{\text{convex}} = p_{\text{non-convex}}
\]
Conclusions

• neural networks can be trained via convex optimization

• global optimality is guaranteed

• higher-dimensional optimization problems

• convex models are transparent and interpretable
Extra Slides

• convex neural networks for signal processing

• interpreting and explaining neural networks based on the convex formulations

• batch normalization

• convex three-layer ReLU neural networks

• convex Generative Adversarial Networks (GANs)
Electrocardiogram (ECG) Prediction

- window size: 15 samples
- training and test set
\[
X = \begin{bmatrix}
x[1] & \ldots & x[d] \\
x[2] & \ldots & x[d+1] \\
\vdots \\
x[n] & \ldots & x[d+n-1]
\end{bmatrix}, \quad y = \begin{bmatrix}
x[d+1] \\
x[d+2] \\
\vdots \\
x[d+n]
\end{bmatrix}
\]
Signal Prediction: Training

![Graph showing training objective over time for different nonconvex SGD trials and a convex SGD trial. The graph indicates the training objective decreases over time for all trials, with the nonconvex SGD trials showing variations in convergence rates and the convex SGD trial showing a consistently lower objective.]
Signal Prediction: Test Accuracy

![Graph showing test accuracy over time for different nonconvex SGD methods and a convex method.](image)

- Test error is plotted on a logarithmic scale.
- The x-axis represents time in seconds.
- Different nonconvex SGD methods are differentiated by color.
- The convex method is indicated by a cross symbol.

EE364b, Stanford University 56
Signal Prediction: Test Accuracy

![Graph showing signal prediction test accuracy with different methods: True Signal, SGD, Convex, Linear. The x-axis represents the range from 200 to 1000, and the y-axis represents the accuracy values from -6 to 8.Insets show zoomed areas highlighting the accuracy differences.]
Interpreting Neural Networks

\[
\min_{u_1, v_1 \ldots u_p, v_p \in \mathcal{K}} \left\| \sum_{i=1}^{p} D_i X (u_i - v_i) - y \right\|_2^2 + \lambda \sum_{i=1}^{p} \|u_i\|_2 + \|v_i\|_2
\]
ReLU Networks with Batch Normalization (BN)

- BN layer transforms a batch of input data to have a mean of zero and a standard deviation of one and has two trainable parameters $\alpha, \gamma$

$$\text{BN}_{\alpha,\gamma}(x) = \frac{(I - \frac{1}{n}11^T)x}{\| (I - \frac{1}{n}11^T)x \|_2} \gamma + \alpha$$

$$p_{\text{non-convex}} = \min_{W_1, W_2, \alpha, \gamma} \left\| \text{BN}_{\alpha,\gamma}(\phi(XW_1))W_2 - y \right\|_2^2 + \lambda \left( \|W_1\|_F^2 + \|W_2\|_F^2 \right)$$

$$p_{\text{convex}} = \min_{w_1, v_1 \ldots w_p, v_p \in K} \left\| \sum_{i=1}^p U_i(w_i - v_i) - y \right\|_2^2 + \lambda \left( \sum_{i=1}^p \|w_i\|_2 + \|v_i\|_2 \right)$$

- where $U_i \Sigma_i V_i^T = D_i X$ is the SVD of $DX_i$, i.e., BatchNorm extracts singular vectors (T. Ergen et al. Demystifying Batch Normalization in ReLU Networks, 2021)
Three layer ReLU Networks

\[
\min_{\{W_j, \tilde{u}_j, \tilde{w}_{1j}, w_{2j}\}_{j=1}^m} \left\| \sum_{j=1}^m \left( (X\tilde{W}_j) + \tilde{w}_{1j} \right) + w_{2j} - \tilde{y} \right\|_2^2 + \beta \sum_{j=1}^m \|\tilde{W}_j\|_F^2 + \|\tilde{w}_{1j}\|_2^2 + w_{2j}^2
\]

- the equivalent convex problem is

\[
\min_{\{\tilde{W}_i, \tilde{W}_i'\}_{i=1}^P} \frac{1}{2} \left\| \sum_{i=1}^P \sum_{j=1}^P D_i D_j X \left( \tilde{W}_i' - \tilde{W}_i \right) - \tilde{y} \right\|_2^2 + \frac{\beta}{2} \sum_{i,j=1}^P \|\tilde{W}_{ij}\|_F + \|\tilde{W}_{ij}'\|_F
\]

T. Ergen, M. Pilanci, Convex Optimization of Two- and Three-Layer Networks in Polynomial Time, ICLR 2021
Convex Generative Adversarial Networks (GANs)

- Wasserstein GAN

\[
p^* = \min_{\theta_g} \max_{\theta_d} \mathbb{E}_{\bar{x} \sim p_x}[D_{\theta_d}(\bar{x})] - \mathbb{E}_{\bar{z} \sim p_z}[D_{\theta_d}(G_{\theta_g}(\bar{z}))]. \tag{1}
\]

- Generative model for the data

- Discriminator and generator are neural networks
• consider a two-layer ReLU-activation generator $G_{\theta_g}(\vec{Z}) = (\vec{Z}\vec{W}_1) + \vec{W}_2$
and quadratic activation discriminator $D_{\theta_d}(\vec{X}) = (\vec{X}\vec{V}_1)^2\vec{V}_2$

• Wasserstein GAN problem is equivalent to a convex-concave game

• can be solved via convex optimization as follows

$$\tilde{G}^* = \arg\min_{\tilde{G}} \|\tilde{G}\|^2_F \text{ s.t. } \|\vec{X}^\top\vec{X} - \tilde{G}^\top\tilde{G}\|_2 \leq \beta_d$$

$$\tilde{W}_1^*, \tilde{W}_2^* = \arg\min_{\tilde{W}_1, \tilde{W}_2} \|\tilde{W}_1\|^2_F + \|\tilde{W}_2\|^2_F \text{ s.t. } \tilde{G}^* = (\vec{Z}\vec{W}_1) + \tilde{W}_2,$$
• the first problem can be solved via singular value thresholding as
  \[ \tilde{G}^* = (\tilde{\Sigma}^2 - \beta_d I)^{1/2} \tilde{V}^\top \] where \( \tilde{X} = \tilde{U} \tilde{\Sigma} \tilde{V}^\top \) is the SVD of \( \tilde{X} \).

• the second problem can be solved via convex optimization as shown earlier
Progressive GANs

- deeper architectures can be trained layerwise
Numerical Results

• fake faces generated from CelebA dataset

• two-layer quadratic activation discriminator and linear generator

• convex optimization with closed form optimal solution