Neurodynamic Optimization: Models and Applications

Jun Wang
jwang@mae.cuhk.edu.hk

Department of Mechanical & Automation Engineering
The Chinese University of Hong Kong
Shatin, New Territories, Hong Kong
http://www.mae.cuhk.edu.hk/~jwang
Outline

• Introduction
Outline

• Introduction
• Problem Formulation
Outline

• Introduction
• Problem Formulation
• Dynamic Optimization
Outline

• Introduction
• Problem Formulation
• Dynamic Optimization
• Exiting Approaches
Outline

• Introduction
• Problem Formulation
• Dynamic Optimization
• Exiting Approaches
• Neurodynamic Models
Outline

• Introduction
• Problem Formulation
• Dynamic Optimization
• Exiting Approaches
• Neurodynamic Models
• Design Procedure
Outline

• Introduction
• Problem Formulation
• Dynamic Optimization
• Exiting Approaches
• Neurodynamic Models
• Design Procedure
• Winners Take All
Outline

• Introduction
• Problem Formulation
• Dynamic Optimization
• Exiting Approaches
• Neurodynamic Models
• Design Procedure
• Winners Take All
• Linear Assignment
Outline

- Introduction
- Problem Formulation
- Dynamic Optimization
- Exiting Approaches
- Neurodynamic Models
- Design Procedure
- Winners Take All
- Linear Assignment
- Machine Learning
Outline

• Introduction
• Problem Formulation
• Dynamic Optimization
• Exiting Approaches
• Neurodynamic Models
• Design Procedure
• Winners Take All
• Linear Assignment
• Machine Learning
• Robot Control
Outline

- Introduction
- Problem Formulation
- Dynamic Optimization
- Exiting Approaches
- Neurodynamic Models
- Design Procedure
- Winners Take All
- Linear Assignment
- Machine Learning
- Robot Control
- Concluding Remarks
Outline

• Introduction
• Problem Formulation
• Dynamic Optimization
• Exiting Approaches
• Neurodynamic Models
• Design Procedure
• Winners Take All
• Linear Assignment
• Machine Learning
• Robot Control
• Concluding Remarks
• Future Works
Introduction

Optimization is ubiquitous in nature and society.
Introduction

Optimization is ubiquitous in nature and society.

Optimization arises in a wide variety of scientific problems.
**Introduction**

Optimization is ubiquitous in nature and society.

Optimization arises in a wide variety of scientific problems.

Optimization is an important tool for design, planning, control, operation, and management of engineering systems.
Problem Formulation

Consider a general optimization problem:

\[
\text{OP}_1 : \text{Minimize} \quad f(x) \\
\text{subject to} \quad c(x) \leq 0, \\
\quad d(x) = 0,
\]

where \( x \in \mathbb{R}^n \) is the vector of decision variables, \( f(x) \) is an objective function, \( c(x) = [c_1(x), \ldots, c_m(x)]^T \) is a vector-valued function, and \( d(x) = [d_1(x), \ldots, d_p(x)]^T \) a vector-valued function.
Problem Formulation

Consider a general optimization problem:

\[
\text{OP}_1 : \text{Minimize} \quad f(x) \\
\text{subject to} \quad c(x) \leq 0, \\
d(x) = 0,
\]

where \( x \in \mathbb{R}^n \) is the vector of decision variables, \( f(x) \) is an objective function, \( c(x) = [c_1(x), \ldots, c_m(x)]^T \) is a vector-valued function, and \( d(x) = [d_1(x), \ldots, d_p(x)]^T \) a vector-valued function.

If \( f(x) \) and \( c(x) \) are convex and \( d(x) \) is affine, then \( \text{OP} \) is a convex programming problem \( \text{CP} \). Otherwise, it is a nonconvex program.
Quadratic Programs

\[ \text{QP}_1: \text{ minimize } \frac{1}{2} x^T Q x + q^T x \]
subject to \( A x = b, \)
\( l \leq C x \leq h, \)

where \( Q \in \mathbb{R}^{n \times n}, q \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, \)
\( b \in \mathbb{R}^m, C \in \mathbb{R}^{n \times n}, l \in \mathbb{R}^n, h \in \mathbb{R}^n. \)
Quadratic Programs

QP₁: minimize \( \frac{1}{2} x^T Q x + q^T x \)
subject to \( Ax = b, \)
\( l \leq C x \leq h, \)

where \( Q \in \mathbb{R}^{n \times n}, q \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, \)
\( b \in \mathbb{R}^m, C \in \mathbb{R}^{n \times n}, l \in \mathbb{R}^n, h \in \mathbb{R}^n. \)

When \( l = 0, h = \infty, C = I, \) QP₁ becomes a standard QP:

QP₂: minimize \( \frac{1}{2} x^T Q x + q^T x \)
subject to \( Ax = b, x \geq 0 \)
Linear Programs

When $Q = 0$, and $C = I$, $QP_1$ becomes a linear program with bound constraints:

$$LP_1: \text{minimize } q^T x$$
$$\text{subject to } Ax = b,$$
$$l \leq x \leq h$$
Linear Programs

When $Q = 0$, and $C = I$, $QP_1$ becomes a linear program with bound constraints:

$$\text{LP}_1 : \text{minimize} \quad q^T x$$
$$\text{subject to} \quad Ax = b,$$
$$l \leq x \leq h$$

In addition, when $l = 0$, and $h = +\infty$, $\text{LP}_1$ becomes a standard linear program:

$$\text{LP}_2 : \text{minimize} \quad q^T x$$
$$\text{subject to} \quad Ax = b,$$
$$x \geq 0$$
Dynamic Optimization

In many applications (e.g., online pattern recognition, robot motion control, and onboard signal processing), real-time solutions to optimization problems are necessary or desirable.
Dynamic Optimization

In many applications (e.g., online pattern recognition, robot motion control, and onboard signal processing), real-time solutions to optimization problems are necessary or desirable. For such applications, classical optimization techniques may not be competent due to the problem dimensionality and stringent requirement on computational time.
Dynamic Optimization

In many applications (e.g., online pattern recognition, robot motion control, and onboard signal processing), real-time solutions to optimization problems are necessary or desirable. For such applications, classical optimization techniques may not be competent due to the problem dimensionality and stringent requirement on computational time. It is computationally challenging when optimization procedures have to be performed in real time to optimize the performance of dynamical systems.
Dynamic Optimization

In many applications (e.g., online pattern recognition, robot motion control, and onboard signal processing), real-time solutions to optimization problems are necessary or desirable. For such applications, classical optimization techniques may not be competent due to the problem dimensionality and stringent requirement on computational time. It is computationally challenging when optimization procedures have to be performed in real time to optimize the performance of dynamical systems. One very promising approach to dynamic optimization is to apply artificial neural networks.
Neurodynamic Optimization

Because of the inherent nature of parallel and distributed information processing in neural networks, the convergence rate of the solution process is not decreasing as the size of the problem increases.
Neurodynamic Optimization

Because of the inherent nature of parallel and distributed information processing in neural networks, the convergence rate of the solution process is not decreasing as the size of the problem increases.

Neural networks can be implemented physically in designated hardware such as ASICs where optimization is carried out in a truly parallel and distributed manner.
Neurodynamic Optimization

Because of the inherent nature of parallel and distributed information processing in neural networks, the convergence rate of the solution process is not decreasing as the size of the problem increases.

Neural networks can be implemented physically in designated hardware such as ASICs where optimization is carried out in a truly parallel and distributed manner.

This feature is particularly desirable for dynamic optimization in decentralized decision-making situations.
Existing Approaches

In their seminal work, Tank and Hopfield (1985, 1986) applied the Hopfield networks for solving a linear program and the traveling salesman problem.
Existing Approaches

In their seminal work, Tank and Hopfield (1985, 1986) applied the Hopfield networks for solving a linear program and the traveling salesman problem.

Kennedy and Chua (1988) developed a neural network for nonlinear programming, which contains finite penalty parameters and thus its equilibrium points correspond to approximate optimal solutions only.
Existing Approaches

In their seminal work, Tank and Hopfield (1985, 1986) applied the Hopfield networks for solving a linear program and the traveling salesman problem.

Kennedy and Chua (1988) developed a neural network for nonlinear programming, which contains finite penalty parameters and thus its equilibrium points correspond to approximate optimal solutions only.

Existing Approaches

In their seminal work, Tank and Hopfield (1985, 1986) applied the Hopfield networks for solving a linear program and the traveling salesman problem.

Kennedy and Chua (1988) developed a neural network for nonlinear programming, which contains finite penalty parameters and thus its equilibrium points correspond to approximate optimal solutions only.


Existing Approaches (cont’d)

A recurrent neural network for quadratic optimization with bounded variables only by Bouzerdoum and Pattison (1993).
Existing Approaches (cont’d)

A recurrent neural network for quadratic optimization with bounded variables only by Bouzerdoum and Pattison (1993).

Existing Approaches (cont’d)

A recurrent neural network for quadratic optimization with bounded variables only by Bouzerdoum and Pattison (1993).


The primal-dual networks for linear and quadratic programming by Xia (1996, 1997).
Existing Approaches (cont’d)

A recurrent neural network for quadratic optimization with bounded variables only by Bouzerdoum and Pattison (1993).


The primal-dual networks for linear and quadratic programming by Xia (1996, 1997).

Existing Approaches (cont’d)

Existing Approaches (cont’d)


A two-layer network for convex programming subject to nonlinear inequality constraints by Xia and Wang (2004).
Existing Approaches (cont’d)


A two-layer network for convex programming subject to nonlinear inequality constraints by Xia and Wang (2004).

Existing Approaches (cont’d)


A two-layer network for convex programming subject to nonlinear inequality constraints by Xia and Wang (2004).


General Design Procedure

A design procedure begins with a given objective function and constraint(s).

The next step involves the derivation of a neurodynamic equation which prescribes the motion of the activation states of the neural network.
General Design Procedure

A design procedure begins with a given objective function and constraint(s).

The next step involves the derivation of a neurodynamic equation which prescribes the motion of the activation states of the neural network.

The derivation of a neurodynamic equation is crucial for success of the neural network approach to optimization.
General Design Procedure

A design procedure begins with a given objective function and constraint(s).

The next step involves the derivation of a neurodynamic equation which prescribes the motion of the activation states of the neural network.

The derivation of a neurodynamic equation is crucial for success of the neural network approach to optimization.

A properly derived neurodynamic equation can ensure that the state of neural network reaches an equilibrium and the equilibrium satisfies the constraints and optimizes the objective function.
In general, there are two approaches to design neurodynamic equations.
General Design Procedure (cont’d)

In general, there are two approaches to design neurodynamic equations.

The first approach is based on an defined energy function.
General Design Procedure (cont’d)

In general, there are two approaches to design neurodynamic equations.

The first approach is based on an defined energy function.

The second approach is based on the existing optimality conditions.
General Design Procedure

The first approach starts with the formulation of an energy function based on a given objective function and constraints
General Design Procedure

The first approach starts with the formulation of an energy function based on a given objective function and constraints.

It plays an important role in neurodynamic optimization.
General Design Procedure

The first approach starts with the formulation of an energy function based on a given objective function and constraints.

It plays an important role in neurodynamic optimization.

Ideally, the minimum of a formulated energy function corresponds to the optimal solution of the original optimization problem.
General Design Procedure

The first approach starts with the formulation of an energy function based on a given objective function and constraints.

It plays an important role in neurodynamic optimization.

Ideally, the minimum of a formulated energy function corresponds to the optimal solution of the original optimization problem.

For constrained optimization, the minimum of the energy function has to satisfy a set of constraints.
General Design Procedure (cont’d)

The majority of the existing approaches formulates an energy function by incorporating objective function and constraints through functional transformation and numerical weighting.
The majority of the existing approaches formulates an energy function by incorporating objective function and constraints through functional transformation and numerical weighting.

Functional transformation is usually used to convert constraints to a penalty function to penalize the violation of constraints; e.g.,

\[ p(x) = \sum_{i=1}^{m} \left\{ [ -c_i(x) ]^+ \right\}^2 + \sum_{j=1}^{p} [ d_j(x) ]^2, \text{ where} \]

\[ [y]^+ = \max\{0, y\}. \]
General Design Procedure (cont’d)

The majority of the existing approaches formulates an energy function by incorporating objective function and constraints through functional transformation and numerical weighting.

Functional transformation is usually used to convert constraints to a penalty function to penalize the violation of constraints; e.g.,

\[ p(x) = \sum_{i=1}^{m} \{ [-c_i(x)]^+ \}^2 + \sum_{j=1}^{p} [d_j(x)]^2, \]  

where \([y]^+ = \max\{0, y\}\).

Numerical weighting is often used to balance constraint satisfaction and objective optimization; e.g., \( E(x) = f(x) + wp(x) \) where \( w \) is a positive weight.
The majority of the existing approaches formulates an energy function by incorporating objective function and constraints through functional transformation and numerical weighting.

Functional transformation is usually used to convert constraints to a penalty function to penalize the violation of constraints; e.g.,

\[ p(x) = \sum_{i=1}^{m} \{[-c_i(x)]^+\}^2 + \sum_{j=1}^{p} [d_j(x)]^2, \]

where \([y]^+ = \max\{0, y\}\).

Numerical weighting is often used to balance constraint satisfaction and objective optimization; e.g., \(E(x) = f(x) + wp(x)\) where \(w\) is a positive weight.
Neurodynamic equations are usually derived as the negative gradient of the energy function:

$$\frac{dx(t)}{dt} \propto -\nabla E(x(t)).$$
Neurodynamic equations are usually derived as the negative gradient of the energy function:

\[
\frac{dx(t)}{dt} \propto -\nabla E(x(t)).
\]

If the energy function is bounded below, the stability of the neurodynamics can be ensured.
Second approach: Neurodynamic equations of some recent neural networks for optimization are derived based on optimality conditions (e.g., Karush-Kuhn-Tucker condition) and projection equations.
Second approach: Neurodynamic equations of some recent neural networks for optimization are derived based on optimality conditions (e.g., Karush-Kuhn-Tucker condition) and projection equations.

Stability analysis is needed explicitly to ensure that resulting neural network is stable.
Second approach: Neurodynamic equations of some recent neural networks for optimization are derived based on optimality conditions (e.g., Karush-Kuhn-Tucker condition) and projection equations.

Stability analysis is needed explicitly to ensure that resulting neural network is stable.

All equilibria of a stable neural network satisfy the optimality condition.
Second approach: Neurodynamic equations of some recent neural networks for optimization are derived based on optimality conditions (e.g., Karush-Kuhn-Tucker condition) and projection equations.

Stability analysis is needed explicitly to ensure that resulting neural network is stable.

All equilibria of a stable neural network satisfy the optimality condition.

If the problem is a convex program, an equilibrium point represents an optimal solution.
The next step is to determine the architecture of the neural network in terms of the neurons and connections based on the derived dynamical equation.
General Design Procedure (cont’d)

The next step is to determine the architecture of the neural network in terms of the neurons and connections based on the derived dynamical equation.

An activation function models important characteristics of a neuron.
General Design Procedure (cont’d)

The next step is to determine the architecture of the neural network in terms of the neurons and connections based on the derived dynamical equation.

An activation function models important characteristics of a neuron.

The range of an activation function usually prescribes the state space of the neural network.
The next step is to determine the architecture of the neural network in terms of the neurons and connections based on the derived dynamical equation.

An activation function models important characteristics of a neuron.

The range of an activation function usually prescribes the state space of the neural network.

The activation function depends on the feasible region delimited by the constraints.
The next step is to determine the architecture of the neural network in terms of the neurons and connections based on the derived dynamical equation.

An activation function models important characteristics of a neuron.

The range of an activation function usually prescribes the state space of the neural network.

The activation function depends on the feasible region delimited by the constraints.

Specifically, it is necessary for the state space to include the feasible region.
General Design Procedure (cont’d)

Any explicit bounds on decision variables can be realized by properly selecting the range of activation functions.
General Design Procedure (cont’d)

Any explicit bounds on decision variables can be realized by properly selecting the range of activation functions.

If the gradient-based method is adopted in deriving the dynamical equation, then the convex energy function results in an increasing activation function.
General Design Procedure (cont’d)

Any explicit bounds on decision variables can be realized by properly selecting the range of activation functions.

If the gradient-based method is adopted in deriving the dynamical equation, then the convex energy function results in an increasing activation function.

Precisely, if the steepest descent method is used, the activation function is equal to the derivative of the energy function.
General Design Procedure (cont’d)

Any explicit bounds on decision variables can be realized by properly selecting the range of activation functions.

If the gradient-based method is adopted in deriving the dynamical equation, then the convex energy function results in an increasing activation function.

Precisely, if the steepest descent method is used, the activation function is equal to the derivative of the energy function.

The last step is usually devoted to simulation to test the performance of the neural network numerically or physically.
Kennedy-Chua Network

The Kennedy-Chua network for solving OP\(^a\):

\[
\epsilon \frac{dx}{dt} = -\nabla f(x) - s \cdot h(c(x))^T \nabla c(x) - s \cdot d(x)^T \nabla d(x),
\]

where \(\epsilon > 0\) is a scaling parameter, \(x \in \mathbb{R}^n\) is the state vector, \(s > 0\) is a penalty parameter, \(h(r) = (h(r_1), ..., h(r_n))^T\), and \(h(r_i) = \max\{0, r_i\}\).

Kennedy-Chua Network

The Kennedy-Chua network for solving OP\(^a\):\n
\[
\epsilon \frac{dx}{dt} = -\nabla f(x) - s \cdot h(c(x))^T \nabla c(x) - s \cdot d(x)^T \nabla d(x),
\]

where \(\epsilon > 0\) is a scaling parameter, \(x \in \mathbb{R}^n\) is the state vector, \(s > 0\) is a penalty parameter, \(h(r) = (h(r_1), \ldots, h(r_n))^T\), and \(h(r_i) = \max\{0, r_i\}\).

With a finite penalty parameter \(s\), the network is globally convergent to a near-optimal solution to an OP even though CP.

---

Deterministic Annealing Network

The deterministic annealing network for solving OP$^a$:

$$
\epsilon \frac{dx}{dt} = -T(t) \nabla f(x) - h(c(x))^T \nabla c(x) - d(x)^T \nabla d(x),
$$

where $\epsilon > 0$ is a scaling parameter, $x \in \mathbb{R}^n$ is the state vector, $T(t) \geq 0$ is a temperature parameter, $h(r) = (h(r_1), \ldots, h(r_n))^T$, and $h(r_i) = \max\{0, r_i\}$.

---

Deterministic Annealing Network

The deterministic annealing network for solving OP\textsuperscript{a}:

\[
\epsilon \frac{dx}{dt} = -T(t) \nabla f(x) - h(c(x))^T \nabla c(x) - d(x)^T \nabla d(x),
\]

where \( \epsilon > 0 \) is a scaling parameter, \( x \in \mathbb{R}^n \) is the state vector, \( T(t) \geq 0 \) is a temperature parameter, \( h(r) = (h(r_1), ..., h(r_n))^T \), and \( h(r_i) = \max\{0, r_i\} \).

If \( \lim_{t \to \infty} T(t) = 0 \), then the network is globally convergent to a feasible near-optimal solution to CP.

Deterministic Annealing Network

The deterministic annealing network for solving \( \text{OP}^{\text{a}} \):

\[
\epsilon \frac{dx}{dt} = -T(t) \nabla f(x) - h(c(x))^T \nabla c(x) - d(x)^T \nabla d(x),
\]

where \( \epsilon > 0 \) is a scaling parameter, \( x \in \mathbb{R}^n \) is the state vector, \( T(t) \geq 0 \) is a temperature parameter, \( h(r) = (h(r_1), ..., h(r_n))^T \), and \( h(r_i) = \max \{0, r_i\} \).

If \( \lim_{t \to \infty} T(t) = 0 \), then the network is globally convergent to a feasible near-optimal solution to \( \text{CP} \).

If \( T(t) \) decreases gradually to 0, then the network is globally convergent to an optimal solution to \( \text{CP} \).

Primal-Dual Network

The primal-dual network for solving LP$_2$: 

\[
\epsilon \frac{dx}{dt} = -(q^T x - b^T y)q - A^T(Ax - b) + x^+, \\
\epsilon \frac{dy}{dt} = -(q^T x - b^T y)b,
\]

where $\epsilon > 0$ is a scaling parameter, $x \in \mathbb{R}^n$ is the primal state vector, $y \in \mathbb{R}^m$ is the dual (hidden) state vector, $x^+ = (x_1^+, \ldots, x_n^+)^T$, and $x_i^+ = \max\{0, x_i\}$.

---

Primal-Dual Network

The primal-dual network for solving LP

\[
\begin{align*}
\epsilon \frac{dx}{dt} &= -(q^T x - b^T y)q - A^T (Ax - b) + x^+, \\
\epsilon \frac{dy}{dt} &= -(q^T x - b^T y)b,
\end{align*}
\]

where \( \epsilon > 0 \) is a scaling parameter, \( x \in \mathbb{R}^n \) is the primal state vector, \( y \in \mathbb{R}^m \) is the dual (hidden) state vector, \( x^+ = (x_1^+, \ldots, x_n^+)^T \), and \( x_i^+ = \max\{0, x_i\} \).

The network is globally convergent to an optimal solution to LP.

\(^a\) Y. Xia, “A new neural network for solving linear and quadratic programming problems,” *IEEE Transactions on Neural Networks*, vol. 7, no. 6, 1544-1548, 1996.
Lagrangian Network for QP

If $C = 0$ in QP$_1$:

$$\epsilon \frac{d}{dt} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -Qx(t) - A^T y(t) - q, \\ Ax - b \end{pmatrix}.$$  

where $\epsilon > 0$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$.

It is globally exponentially convergent to the optimal solution$^a$.

Projection Network

A recurrent neural network called the projection network was developed for optimization with bound constraints only\(^{a,b}\)

\[
\epsilon \frac{dx}{dt} = -x + g(x - \nabla f(x)).
\]

---


Convex Program

Consider a convex programming problem without equality constraints:

\[ \text{CP}_2 : \quad \text{minimize } f(x) \]
\[ \text{subject to } c(x) \leq 0, \; x \geq 0 \]

where \( f(x) \) and \( c(x) = (c_1(x), \ldots, c_m(x))^T \) are convex, \( m \leq n \).
Equivalent Reformulation

The Karush-Kuhn-Tucker (KKT) conditions for CP:

\[
\begin{align*}
    y & \geq 0, \quad c(x) \leq 0, \quad x \geq 0 \\
    \nabla f(x) + \nabla c(x)y & \geq 0, \quad y^Tc(x) = 0
\end{align*}
\]

According to the projection method, the KKT condition is equivalent to:

\[
\begin{align*}
    h(x - \alpha(\nabla f(x) + \nabla c(x)y)) - x & = 0 \\
    h(y + \alpha c(x)) - y & = 0,
\end{align*}
\]

where \( h(r) = (h(r_1), ..., h(r_n))^T \), \( h(r_i) = \max\{0, r_i\} \), and \( \alpha \) is any positive constant.
Two-layer network

Based on an equivalent formulation, a two-layer neural network was developed for OP. It is then given by

$$\epsilon \frac{d}{dt} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -x + g(x - (\nabla f(x) + \nabla c(x)y)) \\ -y + h(y + c(x)) \end{pmatrix},$$

where \( x \in \mathbb{R}^n \) and \( y \in \mathbb{R}^m \).

---

Model Architecture
Convergence Results

For any $x(t_0)$ and $y(t_0)$, $x(t)$ and $y(t)$ are continuous and unique. $u(t) \geq 0$ if $u(t_0) \geq 0$. The equilibrium point solves CP$_2$.

If $\nabla^2 f(x) + \sum_{i=1}^{n} y_i \nabla^2 c_i(x)$ is positive definite on $\mathbb{R}^{n+m}_+$, then the two-layer neural network is globally convergent to the KKT point $(x^*, y^*)$, where $x^*$ is the optimal solution to CP$_2$. 
Two-layer Neural Network for QP

If $C = I$ in $QP_1$, let $\alpha = 1$ in the two-layer neural network for CP:

$$\epsilon \frac{d}{dt} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -x + g((I - Q)x + A^T y - q) \\ -Ax + b \end{pmatrix}.$$

where $\epsilon > 0$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, \( g(x) = [g(x_1), \ldots, g(x_n)]^T \)

\[ g(x_i) = \begin{cases} \quad l_i & x_i < l_i \\ \quad u_i & l_i \leq x_i \leq h_i \\ \quad h_i & x_i > h_i. \end{cases} \]

It is globally asymptotically convergent to the optimal solution.
Illustrative Example

minimize \[ \frac{1}{4}x_1^4 + 0.5x_1^2 + \frac{1}{4}x_2^4 + 0.5x_2^2 - 0.9x_1x_2 \]

subject to \[ Ax \leq b, \ x \geq 0 \]

where

\[ A = \begin{pmatrix} 1 & 1 \\ -1 & 1 \\ 1 & -3 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} 2 \\ 2 \\ -2 \end{pmatrix}. \]

This problem has an optimal solution \[ x^* = [0.427, 0.809]^T. \]
Simulation Results
Illustrative Example

minimize \( x_1^2 + 2x_1x_2 + x_2^2 + (x_1 - 1)^4 + (x_2 - 3) \)

subject to \( x \geq 0, \ c_i(x) \leq 0 \ (i = 1, 2, 3), \)

where

\[
\begin{align*}
 c_1(x) &= x_1^2 + x_2^2 - 64, \\
 c_2(x) &= (x_1 + 3)^2 + (x_2 + 4)^2 - 36, \\
 c_3(x) &= (x_1 - 3)^2 + (x_2 + 4)^2 - 36.
\end{align*}
\]

This problem has an optimal solution \( x^* = (0, 1.96)^T \).
Simulation Results
Illustrative Example

\[
\begin{align*}
\text{minimize} & \quad (x_1 - x_2)^2 + (x_2 - x_3)^2 + (x_3 - x_4)^4 \\
\text{subject to} & \quad x \geq 0, \quad c_i(x) \leq 0 \quad (i = 1, 2),
\end{align*}
\]

where

\[
\begin{align*}
c_1(x) &= x_1^2 + x_2^2 + x_3^2 + x_4^2 - 9 \\
c_1(x) &= (x_1 - 4)^2 + (x_2 + 4)^2 + (x_3 - 1)^2 + (x_4 + 1)
\end{align*}
\]

This problem has an optimal solution

\[
x^* = (3.013, 0, 0.766, 0)^T.
\]
Simulation Results

![Graph showing simulation results with multiple trajectories for variables \( x_1(t) \), \( x_3(t) \), \( x_2(t) \), and \( x_4(t) \) over time.](image)
Dual Network for QP\(_2\)

For strictly convex QP\(_2\), \(Q\) is invertible. The dynamic equation of the dual network:

\[
\epsilon \frac{dy(t)}{dt} = -CQ^{-1}C^Ty + g \left( CQ^{-1}C^Ty - y - Cq \right) + Cq + b,
\]

\[x(t) = Q^{-1}C^Ty - q,
\]

where \(\epsilon > 0\).

It is also globally exponentially convergent to the optimal solution\(^a\) \(^b\).


Simplified Dual Network for QP\textsubscript{1}

For strictly convex QP\textsubscript{1}, \( Q \) is invertible. The dynamic equation of the simplified dual network is:

\[
\epsilon \frac{du}{dt} = -Cx + g(Cx - u),
\]

\[
x = Q^{-1}(ATy + C^Tu - q),
\]

\[
y = (AQ^{-1}A^T)^{-1} [-AQ^{-1}C^Tu + AQ^{-1}q + b],
\]

where \( u \in \mathbb{R}^n \) is the state vector, \( \epsilon > 0 \). It is proven to be globally asymptotically convergent to the optimal solution.

---

Illustrative Example

minimize \[ 3x_1^2 + 3x_2^2 + 4x_3^2 + 5x_4^2 + 3x_1x_2 + 5x_1x_3 + x_2x_4 - 11x_1 - 5x_4 \]

subject to \[ 3x_1 - 3x_2 - 2x_3 + x_4 = 0, \]
\[ 4x_1 + x_2 - x_3 - 2x_4 = 0, \]
\[ -x_1 + x_2 \leq -1, \]
\[ -2 \leq 3x_1 + x_3 \leq 4. \]
Illustrative Example (cont’d)

\[
Q = \begin{bmatrix}
6 & 3 & 5 & 0 \\
3 & 6 & 0 & 1 \\
5 & 0 & 8 & 0 \\
0 & 1 & 0 & 10
\end{bmatrix},
q = \begin{bmatrix}
-11 \\
0 \\
0 \\
-5
\end{bmatrix},
\]

\[
A = \begin{bmatrix}
3 & -3 & -2 & 1 \\
4 & 1 & -1 & -2 \\
-1 & 1 & 0 & 0 \\
3 & 0 & 1 & 0
\end{bmatrix},
b = \begin{bmatrix}
0 \\
0
\end{bmatrix},
\]

\[
C = \begin{bmatrix}
-1 & 1 & 0 & 0 \\
3 & 0 & 1 & 0
\end{bmatrix},
l = \begin{bmatrix}
-\infty \\
-2
\end{bmatrix},
h = \begin{bmatrix}
-1 \\
4
\end{bmatrix}.
\]

The simplified dual neural network for solving this quadratic programming problem needs only two neurons, whereas the Lagrange neural network needs twelve neurons, the primal-dual neural network needs nine neurons, the dual neural network needs four neurons.
Illustrative Example (cont’d)

Transient behaviors of the state vector $u$. 
Illustrative Example (cont’d)

Transient behaviors of the output vector $x$. 
Illustrative Example (cont’d)

Trajectories of $x_1$ and $x_2$ from different initial states.
Illustrative Example (cont’d)

Trajectories of $x_3$ and $x_4$ from different initial states.
A New Model for LP

A new recurrent neural network model with a discontinuous activation function was recently developed for linear programming $LP_1$. 
A New Model for LP

A new recurrent neural network model with a discontinuous activation function was recently developed for linear programming LP$_1$.

The dynamic equation of the new model is described as follows:

$$\epsilon \frac{dx}{dt} = -Px - \sigma(I - P)g(x) + s,$$

where $g(x) = (g_1(x_1), g_2(x_2), \ldots, g_n(x_n))^T$ is the vector-valued activation function, $\epsilon$ is a positive scaling constant, $\sigma$ is a nonnegative gain parameter, $P = A^T(AA^T)^{-1}A$, and

$$s = -(I - P)q + A^T(AA^T)^{-1}b.$$
Activation Function

The following activation function is defined: For \( i = 1, 2, \ldots, n; \)

\[
g_i(x_i) = \begin{cases} 
1, & \text{if } x_i > h_i, \\
[0, 1], & \text{if } x_i = h_i, \\
0, & \text{if } x_i \in (l_i, h_i), \\
[-1, 0], & \text{if } x_i = l_i, \\
-1, & \text{if } x_i < l_i.
\end{cases}
\]

Activation Function (cont’d)

\[ g_i(x_i) \]

![Graph showing the activation function with thresholds at \( l_i \) and \( h_i \).]
Convergence Results

The neural network is globally convergent to an optimal solution of LP\(_1\) with \(C = I\), if \(\bar{\Omega} \subset \Omega\), where \(\bar{\Omega}\) is the equilibrium point set and \(\Omega = \{x | l \leq x \leq h\}\). The neural network is globally convergent to an optimal solution of LP\(_1\) with \(C = I\), if it has a unique equilibrium point and \(\sigma \geq 0\) when \((I - P)c = 0\) or one of the following conditions holds when \((I - P)c \neq 0\):

(i) \(\sigma \geq \| (I - P)c \|_p / \min_{\gamma \in X}^+ \| (I - P)\gamma \|_p\) for \(p = 1, 2, \infty\), or

(ii) \(\sigma \geq c^T (I - P)c / \min_{\gamma \in X}^+ \{ |c^T (I - P)\gamma| \}\),

where \(X = \{-1, 0, 1\}^n\).
Simulation Results

Consider the following LP problem:

\[
\begin{align*}
\text{minimize} & \quad 4x_1 + x_2 + 2x_3, \\
\text{subject to} & \quad x_1 - 2x_2 + x_3 = 2, \\
& \quad -x_1 + 2x_2 + x_3 = 1, \\
& \quad -5 \leq x_1, x_2, x_3 \leq 5.
\end{align*}
\]

According to the above condition, the lower bound of \( \sigma \) is 9
Transient behaviors of the states with four different values of \( \sigma \in \{3, 5, 9, 15\} \).
$k$-Winners Take All Operation

The $k$-winners-take-all ($k$WTA) operation is to select the $k$ largest inputs out of $n$ inputs ($1 \leq k \leq n$).
$k$ Winners Take All Operation

The $k$-winners-take-all ($k$WTA) operation is to select the $k$ largest inputs out of $n$ inputs ($1 \leq k \leq n$).

The $k$WTA operation has important applications in machine learning, such as $k$-neighborhood classification, $k$-means clustering, etc.
\( \kappa \) Winners Take All Operation

The \( \kappa \)-winners-take-all (\( \kappa \)WTA) operation is to select the \( \kappa \) largest inputs out of \( n \) inputs (\( 1 \leq \kappa \leq n \)).

The \( \kappa \)WTA operation has important applications in machine learning, such as \( \kappa \)-neighborhood classification, \( \kappa \)-means clustering, etc.

As the number of inputs increases and/or the selection process should be operated in real time, parallel algorithms and hardware implementation are desirable.
**$k$WTA Problem Formulations**

The $k$WTA function can be defined as:

$$x_i = f(u_i) = \begin{cases} 
1, & \text{if } u_i \in \{k \text{ largest elements of } u\}, \\
0, & \text{otherwise}, 
\end{cases}$$

where $u \in \mathbb{R}^n$ and $x \in \mathbb{R}^n$ is the input vector and output vector, respectively.
$k$WTA Problem Formulations

The $k$WTA function can be defined as:

$$x_i = f(u_i) = \begin{cases} 1, & \text{if } u_i \in \{k \text{ largest elements of } u \}, \\ 0, & \text{otherwise,} \end{cases}$$

where $u \in \mathbb{R}^n$ and $x \in \mathbb{R}^n$ is the input vector and output vector, respectively.

The $k$WTA solution can be determined by solving the following linear integer program:

$$\begin{align*}
\text{minimize} & \quad - \sum_{i=1}^{n} u_i x_i, \\
\text{subject to} & \quad \sum_{i=1}^{n} x_i = k, \\
& \quad x_i \in \{0, 1\}, \quad i = 1, 2, \ldots, n.
\end{align*}$$
$k$WTA Problem Formulations (cont’d)

If the $k$th and $(k + 1)$th largest elements of $u$ are different (denoted as $\bar{u}_k$ and $\bar{u}_{k+1}$ respectively), the $k$WTA problem is equivalent to the following LP or QP problems:

\[
\begin{align*}
\text{minimize} & \quad -u^T x \text{ or } \frac{a}{2} x^T x - u^T x, \\
\text{subject to} & \quad \sum_{i=1}^{n} x_i = k, \\
& \quad 0 \leq x_i \leq 1, \quad i = 1, 2, \ldots, n,
\end{align*}
\]

where $a \leq \bar{u}_k - \bar{u}_{k+1}$ is a positive constant.
The Primal-Dual Network for \( k \)WTA

The primal-dual network based on the QP formulation needs \( 3n + 1 \) neurons and \( 6n + 2 \) connections, and its dynamic equations can be written as:

\[
\begin{align*}
\frac{dx}{dt} &= -(1 + a)(x - (x + ve + w - ax + u)^+) \\
&\quad - (e^T x - k)e - x - y + e \\
\frac{dy}{dt} &= -y + (y + w)^+ - x - y + e \\
\frac{dv}{dt} &= -e^T (x - (x + ve + w - ax + u)^+) \\
&\quad + e^T x - k \\
\frac{dw}{dt} &= -x + (x + ve + w - ax + u)^+ \\
&\quad - y + (y + w)^+ + x + y - e
\end{align*}
\]

where \( x, y, w \in \mathbb{R}^n, v \in \mathbb{R}, e = (1, 1, \ldots, 1)^T \in \mathbb{R}^n \), \( x^+ = (x_1^+, \ldots, x_n^+)^T \), and \( x_i^+ = \max\{0, x_i\} \).
The Projection Network for $k$WTA

The projection neural network for $k$WTA operation based on the QP formulation needs $n + 1$ neurons and $2n + 2$ connections, which dynamic equations can be written as:

\[
\begin{align*}
\frac{dx}{dt} &= \lambda \left[ -x + f(x - \eta(ax - u - ve)) \right] \\
\frac{dv}{dt} &= \lambda (-e^T x + k).
\end{align*}
\]

where $x \in \mathbb{R}^n$, $v \in \mathbb{R}$, $\lambda$ and $\eta$ are positive constants, $f(x) = (f(x_1), \ldots, f(x_n))^T$ and

\[
f(x_i) = \begin{cases} 
0, & \text{if } x_i < 0, \\
x_i, & \text{if } 0 \leq x_i \leq 1, \\
1, & \text{if } x_i > 1.
\end{cases}
\]
The Simplified Dual Network for \( k \)WTA

The simplified dual neural network for \( k \)WTA operation based on the QP formulation needs \( n \) neurons and \( 3n \) connections, and its dynamic equation can be written as:

\[
\begin{cases}
\frac{dy}{dt} &= \lambda \left[ -My + f\left( (M - I)y - s \right) - s \right] \\
x &= My + s,
\end{cases}
\]

where \( x, y \in \mathbb{R}^n \), \( M = 2(I - ee^T/n)/\alpha \), \( s = Mu + ke/n \), \( I \) is an identity matrix, \( \lambda \) and \( f \) are defined as before.

The Simplified Dual Network for $\kappa$WTA
A Static Example

Let the inputs are \( v_i = i \) \((i = 1, 2, \cdots, n)\), \( n = 10 \), \( k = 2 \), \( \varepsilon = 10^{-8} \), and \( a = 0.25 \).
A Static Example (cont’d)

Let the inputs are $v_i = i \ (i = 1, 2, \cdots, n)$, $n = 10$, $k = 2$, $\epsilon = 10^{-8}$, and $a = 0.25$. 

![Graph showing the behavior of the system over time, with axes labeled $t$ (sec) and $x$. The graph illustrates the system's response to the given inputs.](image-url)
A Static Example (cont’d)
A Static Example (cont’d)

![Graph showing the behavior of a static example over time. The graph illustrates the relationship between time (t) and a variable u. Different curves represent various values of n (5, 10, 20, 40) with a constant value of 0.05, showing the system's response under different conditions.](image-url)
A Dynamic Example

Let inputs be 4 sinusoidal input signals (i.e., \( n = 4 \))
\[ v_i(t) = 10 \sin[2\pi(1000t + 0.2(i - 1))], \text{ and } k = 2. \]
A One-layer $k$WTA Network

The dynamic equation of a new LP-based $k$WTA network model is described as follows:

$$\epsilon \frac{dx}{dt} = -Px - \sigma(I - P)g(x) + s,$$  \hspace{1cm} (4)

where $P = ee^T/n$, $s = u - Pu + ke/n$, $\epsilon$ is a positive scaling constant, $\sigma$ is a nonnegative gain parameter, and $g(x) = (g(x_1), g(x_2), \ldots, g(x_n))^T$ is a discontinuous vector-valued activation function.
Activation Function

A discontinuous activation function is defined as follows:

\[
g(x_i) = \begin{cases} 
1, & \text{if } x_i > 1, \\
[0, 1], & \text{if } x_i = 1, \\
0, & \text{if } 0 < x_i < 1, \\
[-1, 0], & \text{if } x_i = 0, \\
-1, & \text{if } x_i < 0.
\end{cases}
\]
Activation Function (cont’d)

\[ g(x_i) \]

[Graph showing a step function with \( g(x_i) \) on the y-axis and \( x_i \) on the x-axis. The function is 1 for \( x_i \geq 0 \) and -1 for \( x_i < 0 \).]
Convergence Results

The network (4) can perform the $k$WTA operation if $\bar{\Omega} \subset \{x \in \mathbb{R}^n : 0 \leq x \leq 1\}$, where $\bar{\Omega}$ is the set of equilibrium point(s).

The network (4) can perform the $k$WTA operation if it has a unique equilibrium point and $\sigma \geq 0$ when $(I - ee^T/n)u = 0$ or one of the following conditions holds when $(I - ee^T/n)u \neq 0$:

(i) $\sigma \geq \frac{\sum_{i=1}^{n} |u_i - \sum_{j=1}^{n} u_j/n|}{2n-2}$, or

(ii) $\sigma \geq n \sqrt{\frac{\sum_{i=1}^{n} (u_i - \sum_{j=1}^{n} u_j/n)^2}{n(n-1)}}$, or

(iii) $\sigma \geq 2 \max_i |u_i - \sum_{j=1}^{n} u_j/n|$, or,

(iv) $\sigma \geq \frac{\sqrt{\sum_{i=1}^{n} (u_i - \sum_{j=1}^{n} u_j/n)^2}}{\min_{\gamma_i \in \{-1,0,1\}} \{ |\sum_{i=1}^{n} (u_i - \sum_{j=1}^{n} u_j/n) \gamma_i| \}}$. 
## Model Comparisons

<table>
<thead>
<tr>
<th>model type</th>
<th>Eqn(s.)</th>
<th>neurons</th>
<th>connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primal-dual neural network</td>
<td>??</td>
<td>$3n + 1$</td>
<td>$6n + 3$</td>
</tr>
<tr>
<td>Projection neural network</td>
<td>??</td>
<td>$n + 1$</td>
<td>$2n + 3$</td>
</tr>
<tr>
<td>Simplified dual network</td>
<td>??</td>
<td>$n$</td>
<td>$3n$</td>
</tr>
<tr>
<td>Neural network herein</td>
<td>(4)</td>
<td>$n$</td>
<td>$2n$</td>
</tr>
<tr>
<td>Neural network herein</td>
<td>??(?)</td>
<td>$n$</td>
<td>$3n$</td>
</tr>
</tbody>
</table>

Table 1: Comparison of related neural networks in terms of model complexity.
Simulation Results

Consider a $k$WTA problem with input vector $u_i = i \ (i = 1, 2, \ldots, n)$, $n = 5$, $k = 3$.

Transient behaviors of the $k$WTA network $\sigma = 6$. 
Transient behaviors of the $k$-WTA network with $\sigma = 2$. 

Convergence Results (cont’d)
Convergence behavior of the $k$WTA network with respect to different values of $n$. 
Linear Assignment Problem

The linear assignment problem is to find an optimal solution to the following linear integer programming problem:

\[
\text{minimize } \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij},
\]

subject to

\[
\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, \ldots, n,
\]

\[
\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, 2, \ldots, n,
\]

\[
x_{ij} \in \{0, 1\}, \quad i, j = 1, 2, \ldots, n.
\]
If the optimal solution to problem (71) is unique, then it is equivalent to the following linear programming problem:

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}, \\
\text{subject to} & \quad \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, \ldots, n, \\
& \quad \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, 2, \ldots, n, \\
& \quad 0 \leq x_{ij} \leq 1, \quad i, j = 1, 2, \ldots, n.
\end{align*}
\]
Simulation Results

Consider a linear assignment problem with

\[ C = \begin{pmatrix}
4 & 2 & 5 \\
3 & 1.5 & 2 \\
4 & 2.5 & 1
\end{pmatrix}. \]

A lower bound of \( \sigma \) is 13.
Simulation Results (cont’d)

Let $\epsilon = 10^{-6}$ and $\sigma = 15$. 

![Graph showing state trajectories over time](image_url)
Support Vector Machine

Consider a set of training examples

\[ \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \]

where the \( i \)-th example \( x_i \in \mathbb{R}^n \) belongs to one of two separate classes labeled by \( y_i \in \{-1, 1\} \).

A support vector machine provides an optimal partition with maximum possible margin for pattern classification.
SVM Primal Problem

\[
\min \frac{1}{2} w^T w + c \sum_{i=1}^{N} \xi_i
\]

s.t. \[
\begin{aligned}
    y_i [w^T \phi(x_i) + b] &\geq 1 - \xi_i, \quad i = 1, \cdots, N \\
    \xi_i &\geq 0, \quad i = 1, \cdots, N.
\end{aligned}
\]

where \( c > 0 \) is a regularization parameter for the tradeoff between model complexity and training error, and \( \xi_i \) measures the (absolute) difference between \( w^T z + b \) and \( y_i \).
SVM Dual Problem

\[
\text{max} \quad -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \phi(x_i)^T \phi(x_j) \alpha_i \alpha_j + \sum_{i=1}^{N} \alpha_i \\
\text{s.t.} \quad \sum_{i=1}^{N} \alpha_i y_i = 0 \quad 0 \leq \alpha_i \leq c, \quad i = 1, \ldots, N.
\]
SVM Dual Problem

For convenient computation here, let $a_i = \alpha_i y_i$. Then the SVM dual problem can be equivalently written as

$$\min \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j K(x_i, x_j) - \sum_{i=1}^{N} a_i y_i$$

s.t. $\left\{ \begin{array}{l} \sum_{i=1}^{N} a_i = 0 \\ c_i^- \leq a_i \leq c_i^+, \quad i = 1, \ldots, N. \end{array} \right.$

where $c_i^- = c \cdot sgn(1 - y_i)$ and $c_i^+ = c \cdot sgn(1 + y_i)$ for $i = 1, \ldots, N$. 
SVM Learning Network

\[
\epsilon \frac{da}{dt} = \left( -a + h(a - (Qa + e\mu - y)) \right)
\]

where \( \epsilon > 0 \), \( a \in \mathbb{R}^N \), and \( \mu \in \mathbb{R} \), \( e = (1, \ldots, 1)^T \).

\[
\epsilon \frac{d\mu}{dt} = -\sum_{k=1}^{N} a_k,
\]

where \( Q = [q_{ij}] = [K(x^{(i)}, y^{(j)})] \), \( w_{ik} = \delta_{ik} - q_{ik} \).

Network Architecture
Iris Benchmark Problem

The data of the iris problem are characterized with four attributes (i.e., the petal length and width, sepal length and width).

The dataset consists of 150 samples belonging to three classes (i.e., virginica, versicolor, setosa), each class has 50 samples.

120 samples for training and the remaining 30 for testing.

We use $c = 0.25$ and the polynomial kernel function $K(x, y) = (x^T y + 1)^p$, with $p = 2$ and $p = 4$. 
Simulation Results

Figure 1: Convergence of the SVM Learning neural network with $\epsilon = 1/150$ and $p = 2$. 
Simulation Results

Figure 2: Convergence of the proposed neural network with $\epsilon = 1/150$ and $p = 4$
Simulation Results

Figure 3: Support vectors of SVC using the proposed neural network with a polynomial kernel $p = 2$
Simulation Results

Figure 4: Support vectors of SVC using the proposed neural network with a polynomial kernel and $p = 4$.
Adult Benchmark Problem

The UCI adult benchmark task is to predict whether a household has an income greater than $50,000 based on 14 other fields in a census form.

Eight of those fields are categorical, while six are continuous. The six fields are quantized into quintile, which yields a total of 123 sparse binary features.

1605 training samples and 2000 testing samples.

Gaussian RBF kernel with width of 10 and $c = 0.5$.

Let $\epsilon = 0.1$, and the initial point $z_0 \in ^{1606}$ with the element being 1.
Adult Benchmark Problem

<table>
<thead>
<tr>
<th>Method</th>
<th>iterations</th>
<th>SVs</th>
<th>Testing accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOR</td>
<td>924</td>
<td>635</td>
<td>84.06</td>
</tr>
<tr>
<td>SMO</td>
<td>3230</td>
<td>633</td>
<td>84.06</td>
</tr>
<tr>
<td>SVM-light</td>
<td>294</td>
<td>634</td>
<td>84.25</td>
</tr>
<tr>
<td>NN</td>
<td>567</td>
<td>633</td>
<td>84.15</td>
</tr>
</tbody>
</table>

Table 2: Comparisons of results of the SOR, SMO, SVM-light, and proposed neural network algorithm
Consider the regression problem of approximating a set of data
\[
\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}
\]
with a regression function as
\[
\phi(x) = \sum_{i=1}^{N} \alpha_i \Phi_i(x) + \varsigma,
\]
where \(\Phi_i(x)(i = 1, 2, \ldots, n)\) are the feature functions defined in a high-dimensional space, \(\alpha_i(i = 1, 2, \ldots, n)\) and \(\varsigma\) are parameters of the model to be estimated.
SVR (cont’d)

By utilizing Huber loss function, the above regression function can be represented as

$$\phi(x) = \sum_{i=1}^{N} \theta_i K(x, x_i) + \varsigma,$$

where $K(x, y)$ is a kernel function satisfying $K(x, y) = \Phi(x)^T\Phi(x)$.
SVR (cont’d)

\( \theta_i \ (i = 1, 2, \ldots, N) \) can be obtained from the following quadratic program:

\[
\begin{align*}
\text{min} & \quad \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \theta_i \theta_j K(x_i, x_j) - \sum_{i=1}^{N} \theta_i y_i + \frac{\varepsilon}{2\mu} \sum_{i=1}^{N} \theta_i^2, \\
\text{s.t.} & \quad \sum_{i=1}^{N} \theta_i = 0, \\
& \quad -\mu \leq \theta_i \leq \mu, \quad i = 1, 2, \ldots, N;
\end{align*}
\]

where \( \varepsilon > 0 \) is an accuracy parameter required for the approximation, \( \mu > 0 \) is a pre-specified parameter.
SVR (cont’d)

The neural network with a discontinuous activation function for solving the above quadratic program:

\[
\frac{d\epsilon z}{dt} = -P z + \left[ PQ + \frac{\alpha}{N} ee^T \right] g(z) + q,
\]

\[
\theta = (PQ + \frac{\alpha}{N} ee^T)^{-1}(Pz - q),
\]

where \( e = [1, 1, \ldots, 1]^T \),

\( P = I - ee^T / N, \)

\( Q = \{ K(x_i, x_j) \}_{N \times N} + \epsilon I / \mu, \)

\( q = (I - ee^T / N)y \) with \( y = -(y_1, y_2, \ldots, y_n) \), and

\( h = -l = \mu e \) in the activation function.
Moreover, $\zeta$ can be obtained from

$$\zeta = -\frac{1}{N} (e^T (Q - I) \theta^* + e^T c - e^T z^*)$$

where $z^*$ is an equilibrium point and $\theta^*$ is an output vector corresponding to $z^*$.

Compared with existing neural networks for SVM learning, the existing neural networks need either two-layer structure and $n + 1$ neurons.

In contrast, the neural network herein has one-layer structure and $n$ neurons only.
SVR (cont’d)

For the SVR learning by using the proposed neural network based on titanium regression data\(^a\). Let the kernel be a Gaussian function:

\[
K(x, y) = \exp \left( -\frac{\|x - y\|^2}{2\sigma^2} \right)
\]

\(\varepsilon = 0.01, \mu = 100\) and \(\sigma = 6\).

Regression Result
Inverse Kinematics Problem

Because $\dot{\theta}$ is underdetermined in a kinematically redundant manipulator, one way to determine $\dot{\theta}(t)$ without the need for computing the pseudoinverse is to solve:

$$\text{minimize} \quad \frac{1}{2} \dot{\theta}(t)^T W \dot{\theta}(t) + c^T \dot{\theta}(t),$$

subject to

$$J(\theta(t)) \dot{\theta}(t) = \dot{x}_d(t),$$

$$\eta^- \leq \dot{\theta} \leq \eta^+$$

where $W$ is a positive-definite weighting matrix, $c$ is an column vector, and $\eta^\pm$ are upper and lower bounds of the joint velocity vector.
Lagrangian Network Dynamics

Let the state vectors of output neurons and hidden neurons be denoted by $v(t)$ and $u(t)$, representing estimated $\dot{\theta}(t)$ and estimated $\lambda(t)$, respectively.

The dynamic equation of the two-layer Lagrangian network can be expressed as:

$$\epsilon_1 \frac{dv(t)}{dt} = -Wv(t) - J(\theta(t))^Tu(t) - c,$$

$$\epsilon_2 \frac{du(t)}{dt} = J(\theta(t))v(t) - \dot{x}_d(t),$$

where $\epsilon_1 > 0$ and $\epsilon_2 > 0$. 

Lagrangian Network Architecture
7-DOF PA10 Manipulator
Coordinate system of PA10 manipulator
Circular Motion of the PA10 Manipulator
Simulation Results
Dual Network Dynamics

To reduce the number of neurons to minimum, next we propose a dual neural network with its dynamic equation and output equation defined as

\[ \frac{d}{dt} u(t) = -J(\theta(t))W^{-1}J^T(\theta)u + \dot{x}_d(t), \]

\[ v(t) = J^T(\theta(t))u(t); \]

where \( u \) is the dual state variable, \( v \) is the output variable.

The Lagrangian network contains \( n + m \) neurons. But the dual network contains only \( m \) neurons, where \( n \) is the number of joints and \( m \) is the dimension of the cartesian space (i.e., 6).
Dual Network
Simulation Results

![Graphs showing simulation results](image)
Simulation Results
Bounded Inverse Kinematics

The dual neural network with the following dynamic equation and output equation

\[
\begin{align*}
\epsilon_1 \frac{dx}{dt} &= -JW^{-1}J^T x + \dot{x}_d \\
\epsilon_2 \frac{dy}{dt} &= -W^{-1}y + g((W^{-1} - I)y) \\
v &= J^T x + y
\end{align*}
\]

where the piecewise linear activation function

\[
g_i(u_i) = \begin{cases} 
\eta_i^- , & \text{if } u_i < \eta_i^- \\
u_i , & \text{if } \eta_i^- \leq u_i \leq \eta_i^+ \\
\eta_i^+ , & \text{if } u_i > \eta_i^+
\end{cases}
\]
Piecewise Linear Activation Function
PA10 Drift-free Circular Motion
PA10 Joint Variables
Joint Velocities and Dual State Variables

(a) Joint rate variables in rad/sec
(b) Nonzero dual decision variables
Euclidean Norm vs. Infinity Norm

The Euclidean norm (or 2-norm) is widely used often because of its analytical tractability.

Minimizing the 2-norm of the joint velocities does not necessarily minimize the magnitudes of the individual joint velocities.

This is undesirable in situations where the individual joint velocities are of primary interest.

Minimizing the infinity norm of velocity variables can minimize the maximum velocity.
Inverse Kinematics Problem

Minimizing the infinity norm of $\dot{\theta}$ subject to the kinematic constraint:

\[
\min_{\dot{\theta}} \left\| \dot{\theta} \right\|_\infty = \min_{\dot{\theta}} \max_{1 \leq j \leq n} \left| e_j^T \dot{\theta} \right|
\]

s.t. \( J(\theta(t)) \dot{\theta}(t) = \dot{x}_d(t) \),

where $e_j$ is the $j$-th column of the identity matrix.
Inverse Kinematics Problem

Let

\[ s = \max_{1 \leq j \leq n} \left| e_j^T \dot{\theta} \right|. \]

The inverse kinematic problem can be written as

\[
\min_{\dot{\theta}_n} s \\
\text{s.t.} \quad \left| e_j^T \dot{\theta} \right| \leq s, \quad j = 1, 2, \ldots, n \\
J(\theta(t))\dot{\theta}(t) = \dot{x}_d(t).
\]
Inverse Kinematics Problem Re-formulation

The inverse kinematics problem can be summarized in a matrix form:

\[
\begin{align*}
\min & \quad s \\
\text{s.t.} & \quad \begin{bmatrix} -I & I_n \\ I & I_n \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ s \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
J(\theta)\dot{\theta} & = \ddot{x}_d(t),
\end{align*}
\]

where \( I_n = (1, 1, \ldots, 1)^T \in \mathbb{R}^n \) and \( I \) is the identity matrix.
Primal Inverse Kinematics

Problem Formulation

Let \( y = (y_1^T, y_2)^T \), \( y_1 = \dot{\theta} \), \( y_2 = s \), then a final form of the problem can be derived as

\[
\min c^T y \\
\text{s.t.} \quad A_1 y \geq 0, \\
\quad A_2 y = b(t),
\]

where

\[
A_1 = \begin{bmatrix} -I & I_n \\ I & I_n \end{bmatrix}, \quad A_2(t) = [J(\theta(t), 0],
\]

\[
b(t) = \dot{x}_d(t), \quad c^T = [0, 0, \ldots, 1].
\]
Dual Inverse Kinematics Problem Formulation

The dual problem of the preceding linear program is defined as follows:

\[
\begin{align*}
\text{max} & \quad b^T z_2 \\
\text{s.t.} & \quad A_1^T z_1 + A_2^T z_2 = c, \\
& \quad z_1 \geq 0,
\end{align*}
\]

where \( z = (z_1^T, z_2^T)^T \) is the dual decision variable.
Energy Function

An energy function to be minimized can be defined based on the primal and dual formulation:

\[
E(y, z) = \frac{1}{2}(c^T y - b z_2)^2 + \frac{1}{2} \| A_2 y - b \|^2 + \\
\frac{1}{2} \| A_1^T z_1 + A_2^T z_2 - c \|^2 + \\
\frac{1}{4} (A_1 y)^T (A_1 y - |A_1 y|) + \\
\frac{1}{4} z_1^T (z_1 - |z_1|).
\]
Primal-Dual Network

The dynamic equation of the primal-dual network:

\[ \epsilon_1 \dot{y} = -c(c^T y - b^T z_2) + A_1^T h(-A_1 y) + A_2^T (A_2 y - b), \]
\[ \epsilon_2 \dot{z}_1 = -h(-z_1) + A_1 (A_1^T z_1 + A_2^T z_2 - c), \]
\[ \epsilon_3 \dot{z}_2 = -b(c^T y - b^T z_2) + A_2 (A_1^T z_1 + A_2^T z_2 - c), \]

where \( y, z_1, z_2 \), are state vectors; \( h(x) = \max\{0, x\} \); and \( \epsilon_i \) are positive scaling constants.
Primal-Dual Network Architecture

Primal System

Dual System

Computational Intelligence Laboratory, CUHK – p. 119/145
Desired Position of PA10 End-Effector

![Graph showing desired end-effector velocity over time](image-url)
PA10 Circular Motion
Joint Velocities from the Lagrangian Network
Joint Velocities from the Primal-Dual Network
Infinity Norm of Joint Velocities

![Graph showing 2-norm and ∞-norm minimization of joint velocities over time.](image)

- **2-norm minimization**
- **∞-norm minimization**

Maximum joint velocity (rad/s) vs. Time (second)
Bi-criteria Kinematic Control

The bi-criteria redundancy resolution scheme subject to joint limits:

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \left\{ \alpha \| \dot{\theta} \|^2_2 + (1 - \alpha) \| \dot{\theta} \|^2_\infty \right\} \\
\text{subject to} & \quad J(\theta) \dot{\theta} = \dot{x}_d \\
& \quad \eta^- \leq \dot{\theta} \leq \eta^+
\end{align*}
\]

where \( \eta^\pm \) denote upper and lower limits of joint velocities respectively.
Problem Reformulation

With $e_j$ denoting the $j$th column of identity matrix $I$,

$$\|\dot{\theta}\|_\infty = \max\{|\dot{\theta}_1|, |\dot{\theta}_2|, \ldots, |\dot{\theta}_n|\} = \max_{1 \leq j \leq n} |e_j^T \dot{\theta}|.$$  

With $s(t) := \|\dot{\theta}(t)\|_\infty$, the term $(1 - \alpha)\|\dot{\theta}(t)\|_\infty^2/2$ equals

$$\min \left\{ \frac{1 - \alpha}{2} s^2(t) \right\} \text{ s.t. } |e_j^T \dot{\theta}| \leq s(t)$$

$$\implies \min \left\{ \frac{1 - \alpha}{2} s^2(t) \right\} \text{ s.t. } \begin{bmatrix} I & -1 \\ -I & -1 \end{bmatrix} \begin{bmatrix} \dot{\theta}(t) \\ s(t) \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
Problem Formulation

With $y := [\dot{\theta}^T, s]^T$, the bi-criteria problem becomes:

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} y^T Q y \\
\text{subject to} & \quad A y \leq b \\
& \quad C y = d \\
& \quad y^- \leq y \leq y^+
\end{align*}
\]

where $Q := \begin{bmatrix} \alpha I \\ (1 - \alpha) \end{bmatrix}$, $A := \begin{bmatrix} I & -1 \\ -I & -1 \end{bmatrix}$, $b := 0 \in \mathbb{R}^{2n}$,

$C := \begin{bmatrix} J(\theta) & 0 \end{bmatrix}$, $d := \dot{x}_d(t)$, $y^- := \begin{bmatrix} \eta^- \\ 0 \end{bmatrix}$, $y^+ := \begin{bmatrix} \eta^+ \\ \max\{\eta^\pm\} \end{bmatrix}$
Problem Formulation

Treat equality and inequality constraints as bound constraints:

$$\xi^- = \begin{bmatrix} b^- \\ d \\ y^- \end{bmatrix}, \quad \xi^+ = \begin{bmatrix} b \\ d \\ y^+ \end{bmatrix}, \quad E = \begin{bmatrix} A \\ C \\ I \end{bmatrix}$$

with $b^-$ sufficiently negative to represent $-\infty$. Then the bicriteria kinematic control problem can be rewritten as

minimize $\frac{1}{2} y^T Q y$

subject to $\xi^- \leq E y \leq \xi^+$. 
Dual Network Dynamics

\[
\frac{du(t)}{dt} = -EQ^{-1}E^T u(t) + g((EQ^{-1}E^T - I)u(t)),
\]

\[
y(t) = Q^{-1}E^T u(t).
\]
Simulation Results
Joint Velocity
Norm Comparison

\[ \| \dot{\theta} \|_\infty \]

\[ \| \dot{\theta} \|_2 \]
Grasping Force Optimization

Consider a multifingered robot hand grasping a single object in a 3-dimensional workspace with $m$ point contacts between the grasped object and the fingers.

The problem of the grasp force optimization is to find a set of contact forces such that the object is held at the desired position and external forces are compensated.

A grasping force $x_i$ is applied by each finger to hold the object without slippage and to balance any external forces.
Grasping Force Optimization

To ensure non-slipping at a contact point, the grasping force \( x_i \) should satisfy \( x_{i1}^2 + x_{i2}^2 \leq \mu_i x_{i3}^2 \), where \( \mu_i > 0 \) is the friction coefficient at finger \( i \), and \( x_{i1}, x_{i2}, \) and \( x_{i3} \) are components of contact force \( x_i \) in the contact coordinate frame.

Besides the form-closure constraints, to balance any external wrench \( f_{ext} \) to maintain a stable grasp, each finger must apply a grasping force \( x_i = [x_{i1}, x_{i2}, x_{i3}] \) to the object such that \( Gx = -f_{ext} \), where \( G \in R^{6 \times 3m} \) is the grasp transformation matrix and \( x = [x_1, \ldots, x_m]^T \in R^{3m} \) is the grasping force vector.
Grasping Force Optimization

The optimal grasping force optimization can be formulated as the following quadratic minimization problem with linear and quadratic constraints:

minimize \( f(x) = \frac{1}{2} x^T Q x \)

subject to \( c_i(x) \leq 0, \ i = 1, \ldots, m; \)
\( Gx = -f_{ext} \)

where \( q \in \mathbb{R}^{3m}, Q \) is a \( 3m \times 3m \) positive definite matrix, and \( c_i(x) = \sqrt{x_{i1}^2 + x_{i2}^2} - \mu_i x_{i3}. \)
Neurodynamic Optimization of Gasping Force

Based on the problem formulation, we develop the three-layer recurrent neural network for gasping force optimization

\[
\epsilon \frac{d}{dt} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -Qx - \nabla c(x)y + G^T z \\ -y + h(c(x) + y) \\ -Gx - f_{ext} \end{pmatrix},
\]

where \( x \in \mathbb{R}^{3m}, y \in \mathbb{R}^m, z \in \mathbb{R}^6 \), and \( \epsilon > 0 \) is a scaling parameter.

The neural network is globally convergent to the KKT point \((x^*, y^*, z^*)\), where \( x^* \) is the optimal gasping force.
Consider a minimum norm force $f(x) = \frac{1}{2} \|x\|^2$.

A polyhedral object with $M = 0.1$ kg is grasped by a three-fingered robotic hand.

Let the robotic hand move along a circular trajectory of radius $r = 0.5$ m with a constant velocity $\nu = 0.2$ m/s.

The time-varying external wrench applied to the center of mass of the object is

$$f_{ext} = [0, f_c \sin(\theta(t)), -Mg + f_c \cos(\theta(t)), 0, 0, 0]^T,$$

where $g = 9.8 (m/s^2)$, $\theta \in [0, 2\pi]$, and $f_c = M\nu^2/r$. 
Simulation Results

Figure 5: Three-finger grasp example

\[ f_c \frac{mv^2}{r} \]

\[ W \quad mg \]
Simulation Results

Figure 6: Motion of the three-finger grasp
Simulation Results
Figure 7: Convergence of the energy function with $\epsilon = 0.0001$
Simulation Results

Figure 8: Comparison of Euclidean norm of optimal forces using three different methods.
Concluding Remarks

Neurodynamic optimization has been demonstrated to be a powerful alternative approach to many optimization problems.

For convex optimization, recurrent neural networks are available with global convergence to the optimal solution.

Neurodynamic optimization approaches provide parallel distributed computational models more suitable for real-time applications.
Future Works

The existing neurodynamic optimization model can still be improved to reduce their model complexity or increase their convergence rate.

The available neurodynamic optimization model can be applied to more areas such as control, robotics, and signal processing.

Neurodynamic approaches to global optimization and discrete optimization are much more interesting and challenging.

It is more needed to develop neurodynamic models for nonconvex optimization and combinatorial optimization.