Chapter 2

Neuro Dynamic Programming Hybrid Model
2.1 Introduction

This Chapter proposes a Hybrid Network utilising Dynamic programming and Neural Network techniques to solve a class of general optimal control problems which includes the Mayer, the Bolza and Euler-Lagrange problems, when control signals are continuous. Conventional methods like Calculus of Variations, Maximum principle etc involve many difficulties in solving second or higher order nonlinear differential equations. This means that two or more boundary conditions must be specified in order to find a unique solution. Dynamic Programming originally developed by Bellman has long been recognized as an extremely powerful approach to solve optimization problems. It can be applied to fundamental problems from many different fields, including aerospace engineering, electrical engineering, chemical engineering, physics, economics and operations research. The standard computational algorithm based on dynamic programming is very desirable from a number of points of view, including the generality of problems to which it can be applied, the nature of the solution that is obtained and the ease with which it can be programmed. Despite the attractive features of the standard algorithm, its applicability thus far has been limited to relatively simple cases due to the combinatorial explosiveness of the algorithm called the “Curse of Dimensionality”. Thus, while Dynamic Programming (DP) is frequently used as analytical and conceptual tool, the computational difficulties associated with the standard algorithm have severely limited its application to large-scale optimal control problems [5, 6, 46]

Artificial Neural Networks (ANN) can achieve high computation rates by employing massive number of simple processing elements with a high degree of connectivity between the elements. Neural Networks with feedback connection provide a
computing model capable of exploiting fine grained parallelism to solve a rich class of optimization problems. Network parameters are explicitly computed based on problem specifications, to cause the network converge to an equilibrium that represents a solution. The use of Neural Networks to solve the constrained optimization problems was initiated by Hopfield and Tank [32, 33]. They proposed a Neural Network model to get good solutions to discrete combinatorial optimization problems like the Travelling Salesman Problem which proved the computational power of the network. Since then many researchers have attempted to apply the Hopfield Model to solve other types of combinatorial optimization problems.

In this chapter a Neuro Dynamic Programming Model is proposed to solve the general optimal control problem by treating it as a multistage decision making problem. The various states in the multistage problem constitute the nodes in a shortest path problem and the cost of the transition between states constitutes the link cost. The transition occurs only between two states in adjacent stages and only one state in a stage could lie on the final path. This method produces results comparable to conventional methods, and also the potential of using Neural Network hardware approach to achieve high computational speed can be made use of. The computational power of the proposed model is demonstrated by applying the methodology to the general optimal control problems.

2.2 Dynamic Programming

Dynamic Programming is concerned with solution of optimization problem, which can be formulated as a sequence of decisions. In most of the practical problems,
decisions have to be made sequentially at different points in time, at different points in space and at different levels, say, for a component, for a sub system and/or for a system. As the decisions in sequential decision problems are to be made at a number of stages, they are also referred to as multi stage decision problems. DP is well suited for the optimization of the multistage decision problems. This technique, when applied, represents and decomposes a multistage decision problem as a sequence of a single stage decision problems. Although simple problems can only be solved by direct means, Dynamic Programming often reduces the volume of calculations required quite drastically. As a result DP can be used for many types of optimal control problems such as Bolza and the Mayer problems which were found to be mathematically complicated and the conventional methods were found to be unsuitable to solve such problems. More computations are involved in solving optimal control problems using calculus of variations which can be overcome using DP. In many practical problems the system has a number of possible states, and a transition between them is required. In general, every system has a finite set of discrete states through which decision-maker must move, following one of a number of innumerable possible paths. The movement is controlled by an optimal policy, which at each state uses one of a set of feasible actions. However in certain instances, where the states of the problem are defined in terms of several parameters, the DP treatment suffers from "Explosion" and the number of states becomes prohibitive. It explodes when number of time intervals are increased to get better accuracy, which increases the complexity to trace optimal path.
2.2.1 Dynamic Programming Algorithm

Set $N$ as the number of stages and initialize the present stage $i=1$
{
Generate total cost for all feasible states at stage $i$
Increment stages $i = i+1$
{
Get all feasible states at previous stage
Find the minimum cost at this stage by adding the previous cost at stage $i$ to the present stage at $i=i+1$
}
Check the condition for stage values
If stage value is less than the number of stages $N$
Repeat the above procedure
Else
Get the optimal solution
End
}

2.2.2 Dynamic Programming Applied to Optimal Control Problems

Dynamic Programming can be regarded as a point of view for looking at optimization problems that lead to an iterative function equation that can be solved efficiently through simulations. Such an equation can be obtained for large class of problems, including many that are quite different from the typical deterministic optimal control problem. In the deterministic optimal control problem the three variables namely the state variable, control variable and the stage variable are present. The state variables provide a complete description of dynamic behaviour of the system and the control variables facilitate decisions to be made in an optimum fashion extremising the performance index. The stage variable determines the order in which controls are to applied and generally indicates increment in time.
The steps involved in solving optimal control problems are listed below [5, 6, 46]

1) The equations describing the system to be controlled are considered to be a set of nonlinear time varying differential equations given as:

\[
x = f(x, u, t)
\]  

(2.1)

where \( x = n \) dimensional state vector

\( u = m \) dimensional control vector

\( t = \) is stage variable.

2) The above differential equation is discretised as given below, where \( \Delta t \) is the time increment over which control \( u(t) \) is applied.

\[
x(t + \Delta t) = x(t) + f[x(t), u(t), t] \Delta t
\]  

(2.2)

3) The performance criterion, which determines the effectiveness of a given control function, is taken to be a cost function that is to be minimized. This cost function has the general variational form, which consists of the integral of scalar functional of the state variable, control variables and time plus a scalar functional of final state and time

\[
\Pi = \int_{t_i}^{t_f} L(x, u, t) \, dt + S[x(t_f), t_f]
\]  

(2.3)

where \( t_i = \) initial time

\( t_f = \) final time

\( L = \) Scalar functional for cost per unit time

\( S = \) scalar functional for final value cost.
4) In order to write the iterative equation the minimum cost function \( C(x,t) \) must be first defined. This function determines the minimum cost incurred by the system to reach the final time, if the present time is \( t \) and present state is \( x \). It is defined for all admissible states \( x \in X \) and for all time \( t_i \leq t \leq t_f \). The iterative equation can be derived directly from Eqs. 2.2 and 2.3 as

\[
C(x,t) = \min_{u \in U} \{ L(x,u,t)\Delta t + C[x + f(x,u,t)\Delta t, t + \Delta t] \}
\]

Equation (2.4)

The interpretation of this equation is that the minimum cost at a given state \( x \) and the present time \( t \) is found by minimizing, through the choice of the present control \( u \), the sum of \( L(x,u,t)\Delta t \), the cost over the next time interval \( \Delta t \), plus \( C[x + f(x,u,t)\Delta t, t + \Delta t] \), minimum cost of going to \( t_f \) from the resulting next state, \( (x + f(x,u,t)\Delta t) \).

5) The iterative equation is solved backwards in time because \( C(x,t) \) depends on the values of minimum cost function at future times. Consequently the iterations begin by specification of the minimum cost function at final time \( t_f \)

\[
C[(x(t_f))] = S(x, t_f)
\]
Equation (2.5)

6) The minimum cost function for all \( x \) and \( t \) can be evaluated by iteratively solving Eq. 2.4 with Eq. 2.5 as a boundary condition. The optimal control at every \( x \) and \( t \), denoted as \( u^*(x,t) \), is obtained as the value of \( u(t) \) which minimizes Eq. 2.4 for the given \( x \) and \( t \). The Fig. 2.1 illustrates the above optimization procedure.
The DP solution $u^*(x,t)$ leads to an optimal feedback control configuration. Generally a nominal trajectory based on either the last computed trajectory, the results of pre-computations, or operating experience, is used to decrease the size of the sets $X$ and $U$ over which computations are made. This leads to the fact that computational requirements increase very rapidly as the size of the time interval $\Delta t$ decreases which is called the curse of dimensionality. Hence the Recurrent Neural Network can be used to trace the optimal trajectory [56-59] for decreased time intervals as explained in the next section.

2.3 Recurrent Neural Networks

Neural Networks are of interest to researchers in many areas for different reasons. There are various points of view as to the nature of a neural net. For example, it is a specialized piece of computer hardware (say, a VLSI chip) or a computer program. A Neural Network is an information – process system that has certain performance
characteristics in common with biological neural network. High speed digital computers make the simulation of the neural process more feasible. A Neural Network is characterized by

1) Its pattern of connections between the neurons called its architecture.
2) Its method of determining the weights on the connections called its algorithm.
3) Its activation function.

A Neural Network consists of a large number of simple processing elements called neurons or units or nodes. Each neuron is connected to other neuron by means of directed communication link and each with an associated weight. Each neuron has an internal state called its activation or activity level. A Neural Network is distinguished on the basis of the signal flow direction and classified as:

1) Feed forward network
2) Feedback network

![Fig. 2.2 A Feedforward Neural Network](image)

A feed forward network shown in Fig. 2.2 is a network in which signals propagate in only one direction from an input stage through intermediate neurons to an output stage.
This network basically consists of three layers namely input layer, hidden layer, and the output layer. The input layer does not perform any computation or weighing of the input vector but distributes it to the neurons in the hidden layer. The neurons in the hidden layer respond to the accumulated effects of their input and propagate their response signal to the neurons in the output layer. The neurons in the output layer also accumulate the effects of the signal they receive and collectively produce an output vector of signals which represent the response of the network to the input vector.

Feed forward networks have no memory since the output is solely dependent on the input vector and have been applied successfully to a number of problems including sonar signal processing, speech recognition, stock market prediction, image compression, and adaptive process control [21, 26, 30].

Fig. 2.3 A Feedback Neural Network

A feedback network shown in Fig. 2.3 also referred to as recurrent network, is a network in which signals may propagate from the output of any neuron to the input of the any neuron. Recurrent networks are the networks that have closed loops in the network...
topology. Recurrent nets feedback the output of a neuron into the network as input to
other neuron and hence the self-state of the each neuron is altered as recurrent nets
receive their previous outputs as inputs and exhibit properties similar to short term
memory. The significant distinction is that recurrent network has the memory and can
therefore store a set of unit outputs or system information state. John Hopfield in 1982
[32, 33] described the dynamic behaviour of a single layer, symmetrically feedback
network of neurons as minimization for quadratic energy function. This energy function
depends on the state of the neurons and it is determined by the feedback synapses by
defining the synapse weights appropriately. It is possible to encode a quadratic
optimization problem or a discrete constraint satisfaction into the network. After relaxing
the network state into the equilibrium the individual neuron states describe the solution
that have been found. This network has been successfully applied to the difficult
optimization problem that belongs to the Non-deterministic Polynomial (NP) namely the
Travelling Salesman Problem. Since then many researchers have attempted to apply the
Hopfield model to solve other types of combinatorial optimization problems.

2.3.1 Neural Networks for Constraint Satisfaction

Neural Networks for constraint satisfaction problems are recurrent networks
based on a technical neuron model as shown in Fig. 2.4. These technical neurons (units,
processing units, processing elements) try to imitate the average spike rate (activity) of
biological neurons.
A neuron is generally defined as follows:

- The input signals ($e_s$) of a neuron are weighted by synapses ($w_s$) and are superimposed linearly to a threshold value $w_0$.
- The resulting activation ($x$) modifies the soma potential (state $u$) of the neuron with a certain time dependency ($T$).
- The output signal ($z$) is derived from the soma potential by a non-linear function $g(u)$.

This is described by a set of equations:

$$x = \sum_s w_s e_s + w_0 \quad (2.6)$$

$$u(t+\Delta t) = f_T[u(t),x(t),\Delta t] \quad (2.7)$$

$$z = g(u) \quad (2.8)$$

The weighing factors $w_s$ determine the input-output function of the neuron, and the function $f$ describes the dynamic behavior of the neuron. The non-linear function $g(u)$ is usually a threshold function or a differentiable sigmoid function.

![Figure. 2.4 General Definition of Neuron.](image)

### 2.3.2 Hopfield Neural Network Model

A nonlinear, totally interconnected, recurrent, symmetric network is often referred to as a Hopfield network. Hopfield Neural Network is used to trace the optimal path,
which is mainly based on the modified Hopfield energy function. The name suggests continuous Hopfield network, because the time is continuous as well as it represents the neuron states. The Hopfield Neural Computational Circuit is shown in Fig. 2.5.

![Hopfield Neural Network Structure](image)

**Fig. 2.5 Hopfield Neural Network Structure**

This circuit is designed so as to model the basic components of a biological Neural Networks. Each neuron is modeled as a nonlinear device with a sigmoid monotonic increasing function relating the output \( V_i \) of the \( i^{th} \) neuron to its input \( U_i \). The output \( V_i \) (\( z \) in the neuron model) is allowed to take on any value between 0 and 1. Typical sigmoid function is the logistic function given below

\[
V_i = g_i(U_i) = \frac{1}{1 + e^{-\lambda_i U_i}} \tag{2.9}
\]

where \( \lambda_i \) = steepness factor

Each neuron receives resistive connections (to model the synaptic connections) from other neurons, described as the matrix \( T = [T_{ij}] \) (weights \( w_{ij} \) in the neuron model) called connection matrix of the network. Each neuron receives an external current \( I_i \) (\( w_0 \) in the
neuron model) which could represent an actual data provided by the user to the neural network. The dynamics of the Hopfield network are described by [32, 35, 60-65].

\[
\frac{dU_i}{dt} = \sum_{j=1}^{N} T_{ij} V_j - \frac{U_i}{\tau} + I_i
\]  \hspace{1cm} (2.10)

Where \( \tau = \) the circuit time constant 
\( N = \) number of neurons

If the gain of the amplifiers is sufficiently high \((\lambda \rightarrow \infty)\) then the dynamics of the neuron follow the gradient descent quadratic energy function.

\[
E = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} T_{ij} V_i V_j - \sum_{i=1}^{N} I_i V_i
\]  \hspace{1cm} (2.11)

In terms of the energy function, the dynamics of the \(i^{th}\) neuron are described by

\[
\frac{dU_i}{dt} = -\frac{U_i}{\tau} - \frac{\partial E}{\partial V_i}
\]  \hspace{1cm} (2.12)

Networks of neurons with this basic organisation can be used to compute solutions to specific optimization problems by first choosing connectives \(T_{ij}\) and input currents \(I_i\) which approximates the function to be minimized.

### 2.3.2.1 Modified Energy Function

To formulate the Shortest Path (SP) problem and later to adapt to DP problem in terms of the Hopfield neural model, a suitable representation scheme must be found so that the shortest path (minimum cost path) can be decoded from the final stable state of the neural network. The proposed model is organized in an \((n \times n)\) matrix, with all diagonal elements removed, since they are not needed as there is no self connections.
Each element in the matrix is represented by a neuron which is described by double indices \((x, i)\), where the row subscript \(x\), and the column subscript \(i\) denote the node numbers. Therefore the computational network requires \(n(n-1)\) neurons, and a neuron at location \((x,i)\) is characterized by its output \(V_{xi}\), defined as follows:

\[
V_{xi} \begin{cases} 
1 & \text{if the arc from the node } x \text{ to node } i \text{ is in the shortest path.} \\
0 & \text{Otherwise.}
\end{cases}
\]

Also define

\[
\rho_{xi} \begin{cases} 
1 & \text{if the arc from node } x \text{ to node } i \text{ does not exist in the shortest path.} \\
0 & \text{Otherwise.}
\end{cases}
\]

In addition the cost of an arc from node \(x\) to node \(i\) will be denoted by \(C_{xi}\), a finite real positive number which can be determined from the DP algorithm. For non-existing arcs this cost will be assumed to zero, i.e, all non-existing arcs will be eliminated from the solution. In order to solve the shortest path problem using the Hopfield model, an energy function whose minimization process drives the Neural Network into the lowest energy state has to be defined. This stable state shall correspond to the SP solution. The energy function must favour states that correspond to valid paths between the specified origin-destination pair. Suitable energy function that satisfies such requirements is found \([32, 35]\) and is given below

\[
E = \frac{\mu_1}{2} \sum_{x=1}^{n} \sum_{i=1}^{n} C_{xi} \cdot V_{xi} + \frac{\mu_2}{2} \sum_{x=1}^{n} \sum_{i=1}^{n} \rho_{xi} \cdot V_{xi} + \frac{\mu_3}{2} \sum_{x=1}^{n} \left\{ \sum_{i=1}^{n} V_{xi} \cdot \sum_{i=1}^{n} V_{ix} \right\}^2 \\
+ \frac{\mu_4}{2} \sum_{x=1}^{n} \sum_{i=1}^{n} \cdot V_{xi} \left( 1 - V_{xi} \right) + \frac{\mu_5}{2} \left( 1 - V_{ds} \right) \tag{2.13}
\]
In the above energy function $\mu_1$ term minimizes the total cost of a path by taking into account the cost of existing links. The $\mu_2$ term prevents non-existent links being included in the chosen path. The $\mu_3$ term is zero if for every node in the solution, the number of incoming arcs equals the number of outgoing arcs. This makes sure that if a node has been entered it will also be excited by a path. The $\mu_4$ term pushes the state of the neural networks to converge to one of the corners $2^{n^2-n}$ of the hyper cube defined by $V_{x_i\in \{0,1\}}$. The $\mu_5$ term is zero when the output of the neuron at location $(d,s)$ settles to one. Although the link from $d$ to $s$ is not part of the solution, it is introduced to enforce the construction of a part, which must originate at $s$ and terminate at $d$. This makes sure that final solution contains the arc from $d$ to $s$ and therefore both source and destination nodes will be in the solution. Thus the final solution will always be a loop with nodes $s$ and $d$ included. This loop consists of two parts: A directed path from $s$ to $d$ and arc from $d$ to $s$. If there are no zero length loops in the network, then the $\mu_1$ and $\mu_3$ terms ensure that there will be almost a single one number at each row and each column. This guarantees that there will be one to one relationship between the paths and the Neural Networks representations.

### 2.3.2.2 The Connection Matrix and the Biases

Now, rewriting Eqs. 2.9, 2.10 and 2.12 in such a way as to take into account the representation of the neurons with double subscripts

\[
\frac{dU_{xi}}{dt} = \sum_{y=1}^{n} \sum_{j=1}^{n} T_{xi,yj} V_{yj} \frac{U_{xi}}{\tau} + I_{xi}
\]

\[j \neq y\]
\[
\frac{dU_{xi}}{dt} = -\frac{U_{xi}}{\tau} - \frac{\partial E}{\partial V_{xi}} \tag{2.15}
\]

\[
V_{xi} = g_{xi}(U_{xi}) = \frac{1}{1 + e^{-\lambda_{xi}U_{xi}}} \quad \forall(x, i) \in \overline{N x N} / x \neq i. \tag{2.16}
\]

By substituting Eq 2.13 in Eq 2.15 the equation of motion of the Neural Network is readily obtained as

\[
\frac{dU_{xi}}{dt} = -\frac{U_{xi}}{\tau} - \frac{\mu_1}{2} C_{xi}(1 - \partial_{xd} \partial_{is}) - \frac{\mu_2}{2} \rho_{xi}(1 - \partial_{xd} \partial_{is})
- \mu_3 \sum_{y=1}^{n} (V_{xy} - V_{yx}) + \mu_4 \sum_{y=1}^{n} (V_{iy} - V_{yi}) - \frac{\mu_4}{2} (1 - 2V_{xi}) + \frac{\mu_5}{2} \partial_{xd} \partial_{is}
\]

\[
\forall(x, i) \in \overline{N x N} / x \neq i. \tag{2.17}
\]

Where \( \partial \) is the Kronecker delta defined by

\[
\partial_{ab} = \begin{cases} 
1 & \text{if } a = b \\
0 & \text{otherwise}
\end{cases} \tag{2.18}
\]

By comparing the corresponding coefficients in Eqs. 2.14 and 2.17 the connection strengths are derived as

\[
T_{xi,yj} = \mu_4 \partial_{xy} \partial_{ij} - \mu_2 \partial_{xy} - \mu_3 \partial_{ij} + \mu_3 \partial_{jx} + \mu_3 \partial_{iy} \tag{2.19}
\]

\[
I_{xi} = -\frac{\mu_1}{2} C_{xi}(1 - \partial_{xd} \partial_{is}) - \frac{\mu_2}{2} \rho_{xi}(1 - \partial_{xd} \partial_{is}) - \frac{\mu_4}{2} + \frac{\mu_5}{2} \partial_{xd} \partial_{is} \tag{2.20}
\]

39
The first term in Eq. 2.19 represents excitatory self-feedbacks, and the second and third terms represent local inhibitory connections among neurons in the same row and in the same column, respectively. The last two terms represent excitatory cross-connections among neurons. The advantage of the proposed model is that it maps the data into biases rather than into neural connections.

For a valid neural output, one neuron corresponding to an existing arc, changes its output from 1 to 0 and for the algorithm to converge successfully, the coefficients selected are $\mu_1 = 210, \mu_2 = 700, \mu_3 = 500, \mu_4 = 78, \mu_5 = 70$. For the network shown in Fig. 2.6, the shortest path assumed is $P^sd = (1,2,5,6)$. These arcs form a loop thereafter referred to as a primary loop. Correspondingly the neural output will be represented as shown in Table 2.1, where each node included in the shortest path has a single 1 in its corresponding row/column, and other nodes having a zero in its corresponding row/column and not considering the distance between the same node, hence it is blackened as shown in Table 2.1.
Table 2.1

Neural Output Representation

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>✗</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>✗</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>✗</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 2.6 Shortest Path Illustration Network

In the representation since nodes three and four are not part of the shortest path they have all zero entities in their corresponding rows/columns. Therefore path $P^{sd} = (1,2,5,6)$ will have one and only one corresponding neural representation as shown in Table 2.1

2.3.2.3 Adaptation of Dynamic Programming Problem as Shortest Path Problem

Considering a N stage and S state problem, it is equivalent to a (N*S) node problem requiring (N*S) *(N*S) nodes of the HNN network. The following should be considered while applying the model to the DP problem shown in Fig. 2.7.

1) Connectivity exists only between states of adjacent stages.
2) Only one state in each stage can be a part of the final optimal path
3) Only those transitions between stages that obey all constraints have valid connectivity.
4) While calculating \((N*S) \times (N*S)\) cost matrix required for the network the cost of the arcs with valid connection are got from the transition between the respective states of adjacent stages and the rest of the costs are assumed to be zero since they have no connection.

With these changes the HNN Algorithm can be used to find the optimum path for any stage variable and the results obtained were found to be comparable to that obtained by conventional DP.

![Diagram of shortest path adaptation to Dynamic Programming](image)

**Fig. 2.7 Adaptation of Shortest Path to Dynamic Programming**

The modified HNN can be used to solve the shortest path problems when the cost between the two states at every stage is known in advance. These costs can be calculated using DP algorithm for optimal control problems for any levels of discretisation. When the number of stages becomes fairly large which has to be given to provide accuracy to reach the global minimum/maximum in comparison with calculation of variations, the
problem becomes explosive in nature. Hence the shortest path (SP) algorithm is applied to the cost matrix obtained by DP to trace the optimal path. The proposed Hybrid model combines many features, such as a very good convergence and scaling properties, a relatively low programming complexity and an ability to operate in real time and to adapt to changes in network topology and link costs.

2.4 Neuro Dynamic Programming Hybrid Model

The optimal control problem can be initially treated as a DP problem consisting of N stages and S states. The cost between state transitions is calculated as \( C_{xi} \) (where \( x \) denotes the state and \( i \) denotes the stage). Using \( C_{xi} \) and \( \rho_{xi} \) as inputs, \( T_{xi,yj} \) and \( I_{xi} \) are calculated using Eqs 2.19 and 2.20 from which \( U_{xi} \) can be found using Eq. 2.14 and hence \( V_{xi} \) from Eq. 2.17 which is the modified HNN algorithm adopted. The proposed Neuro Dynamic Programming Model is shown in Fig. 2.8

![Fig. 2.8 Neuro Dynamic Programming Hybrid Model](image-url)
2.4.1 Neuro Dynamic Programming Hybrid Algorithm: NN_DP

**Algorithm NN_DP( )**

**COST MATRIX**

Set \( N \) — number of stages
Set \( S \) — nume of states
Set \( n \) — number of nodes = \( N \times S \)
Set cost matrix \( \text{costfnj} \) to zero
Initialize the present stage \( i = 1 \), present state \( s = 1 \)

while \( i \leq N \)

\{

  For all states \( s = 1 \) to \( S \)

  \{

    Find cost to reach this state \( s \) from all states \( s_{i-1} \) in previous stage
    \[
    \text{cost}[(i-2)*s + s_{i-1}]/[(i-1)*s + s]
    \]

  \}

  increment \( i \)

\}

**HNN ALGORITHM (modified)**

Set all elements of \( u_{xi} \) to zero. Each element is one output of a neuron described by double indices \((x,i)\)
where \( x = 1, 2, \ldots, N \times S \)
\( i = 1, 2, \ldots, N \times S \)
Set \( u_{x_i}^0 = 0 \) for all units \( x = 1, 2, \ldots, S \times N \)
\( i = 1, 2, \ldots, S \times N \)

Introduce Random noise \( -0.0002 \leq \delta u_{x_i} \leq +0.0002 \)
Set \( \tau = 1; \ \delta \alpha = 10^{-5}; \ \varepsilon = 10^{-3} \)
Find the cost matrix \( C_{xi} \) using FINDCOST
Set \( k = 1 \)

\{

  Find \( u_{x_i}^k \) for all units using

  \[
  u_{x_i}^{k+1} = \left[ -\frac{u_{x_i}^k}{2} - \frac{\partial E}{\partial V_{x_i}} \right] dt + u_{x_i}^k
  \]

  where \( V_{x_i} = \frac{1}{1 + e^{-\lambda u_{x_i}}} \)

  \( k = i + 1 \)

\}

while ( \( \Delta V_{x_i} < \varepsilon \) for all \( x_i \) )
Set for all \( x_i \);
\( V_{x_i}^k = 1 \) if \( V_{x_i} > 0.5 \)
\( V_{x_i}^k = 0 \) if \( V_{x_i} < 0.5 \)

**FIX PATH**

Get \( V_{x_i} \) from HNN ALGO
For all \( x_i \)

\{

  if \( V_{x_i} = 1 \)
  The \( x \) mod \( N \) state of \( N \)th stage and \( i \) mod \( N \) state of \( i \) mod \( N \)th stage are present in the optimal path.

\}
2.5 Application

The proposed Neuro DP model has been applied to the general optimal control problem with states and control inputs under constraints and satisfying the different boundary conditions. The results were compared with the values obtained by calculus of variations.

Consider a system described by the first order differential equation [46]

\[
\frac{dx(t)}{dt} = ax(t) + bu(t)
\]  

(2.22)

The performance criterion to be minimised is

\[
\mathcal{P}_I = \int_0^5 [x^2(t) + u^2(t)] dt + 2.5[x(t_f) - 2]^2
\]  

(2.23)

The corresponding system difference equation is

\[
x(t + \Delta t) = (1 + a \Delta t)x(t) + b \Delta t u(t)
\]

with \(a = 0\), \(b = 1\), the difference equation becomes

\[
x(k + 1) = x(k) + u(k)
\]  

(2.24)

In a similar way the performance index becomes

\[
\mathcal{P}_I = \sum_{k=0}^4 [x^2(k) + u^2(k)] + 2.5[x(5) - 2]^2
\]  

(2.25)

The physical limits of the state and control variable are given as

\[
0 \leq x(t) \leq 2
\]

\[
-1 \leq u(t) \leq 1
\]

Boundary conditions are given as \(x(0) = 2\) and \(x(5)\) is not specified.

The solution for the optimal state trajectory obtained by Euler method for the prescribed boundary condition is
The optimum cost obtained for the prescribed bounds from Euler's is 7.7122.

The grid of quantised values of $x$ and $k$ at which computations are made along with the cost matrix is shown in Table 2.2. Using the cost matrix obtained from DP given in Table 2.2 and the HNN Algorithm, which utilises Eq. 2.13, 2.15 and 2.16, the shortest path is obtained as given in Table 2.3. The optimal sequence is traced from Table 2.3 and given in Table 2.4. Table 2.5 gives the optimum cost as $\Delta t$ is decreased to a value of 0.1.

The above problem requires (18x18) neuron representation as shown in Table 2.3. The shortest path from HNN output is between neurons 1-5-9-12-14-16 which is same as obtained from Dynamic Programming when time interval $\Delta t = 1.0$. When $\Delta t = 0.5$, then the number of neurons is (36x36) and thus the number of neurons increases with the number of time intervals. The algorithm suffered when the size of the time interval was decreased as the number of neurons depends on the size of the time interval. The optimal state trajectory and control trajectory is drawn using Table 2.4 and is shown in Fig. 2.8.

$$x(t) = \frac{2 \cdot (1 - e^{-4})}{e^4 - e^{-4}} e^t - \frac{2 \cdot (1 - e^4)}{e^4 - e^{-4}} e^{-t}$$

(2.26)
### Table 2.2

Cost Matrix from DP for $\Delta t = 1$

<table>
<thead>
<tr>
<th>$x/k$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2.5</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3.5</td>
<td>10</td>
</tr>
</tbody>
</table>

### Table 2.4

Optimal sequence from HNN for $x(0) = 2$

<table>
<thead>
<tr>
<th>Stage $k$</th>
<th>State $x(k)$</th>
<th>$u$</th>
<th>Accumulated cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>-1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 2.3

Optimal Trajectory (1-5-9-12-14-16)

```
0000100000000000
0000000000000000
0000000000000000
0000011000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
```

### Table 2.5

Optimum Cost for Different Time Intervals

<table>
<thead>
<tr>
<th>Time Interval</th>
<th>Optimum Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>10.0</td>
</tr>
<tr>
<td>0.5</td>
<td>8.875</td>
</tr>
<tr>
<td>0.25</td>
<td>8.75</td>
</tr>
<tr>
<td>0.2</td>
<td>8.6399</td>
</tr>
<tr>
<td>0.1</td>
<td>8.62</td>
</tr>
</tbody>
</table>
2.6 Summary

The hybrid model technique has been applied to the general optimal control problems including all the boundary conditions. The proposed model combines many features such as relatively low programming complexity requirements, ability to operate in the real time and to adapt to changes in link costs. Hence it can be summarised that this proposed algorithm can be accepted as another optimization procedure which can be used to solve optimal control problems as it gives the results comparable to the conventional methods. The algorithm has been coded in C++ and though the results were found to be comparable with DP, the HNN algorithm involved a large memory allocation for the above representation scheme. Hence the above proposed model can be concluded as another optimisation procedure for optimal control problems. The model was limited up to second order systems as DP for higher order systems becomes too cumbersome.