Neural Networks for Control

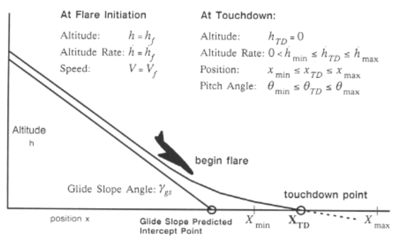
15-486/782: Artificial Neural Networks
David S. Touretzky

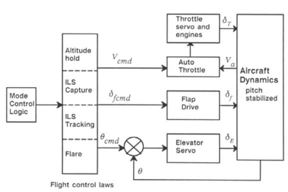
Fall 2006

Examples of Control Problems

- Getting a robot arm to pick up a soda can.
- Parallel parking a car.
- Getting a chemical plant to produce a steady flow of a desired product:
  - Control temperature, rate at which reactants are added, and pressure inside the reaction vessel.
  - Faulty controller? Plant blows up!

Landing An Airplane





Jorgensen & Schley (1990): A neural network baseline problem for control of aircraft flare and touchdown.

Control Theory

Control theory includes:
- Design of controllers
- Stability issues
- Adaptive control

Many controllers are designed by hand.

Neural nets provide a way to “learn” the controller.

The “Plant”

The thing being controlled is called the “plant”.

The control problem: how to find the right u(t) to produce the desired behavior y*(t)?

The plant is a black box.

5

The System Identification Problem

What does the plant do in response to input u(t) ?

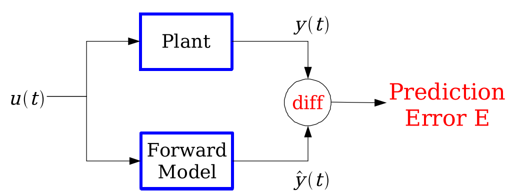
A model of the plant that maps $u(t) \rightarrow \hat{y}(t)$ *is called a* ***forward model.***

$y(t)$ = *actual plant behavior*
$\hat{y}(t)$ = *predicted behavior*

Prediction error = $y(t) - \hat{y}(t)$

6

Training a Forward Model



Generate random control signals $u(t)$.

Observe plant behavior $y(t)$, *measure prediction error E.*

Back propagate error to adjust weights in forward model.

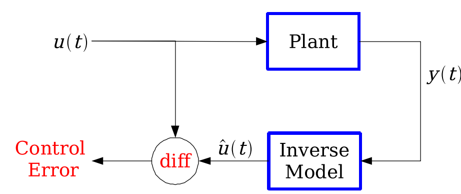
7

Inverse Models

An **inverse model** maps plant outputs back to the control signals that produced them.

Can use an inverse model to control the plant.

Training a **direct** inverse model:

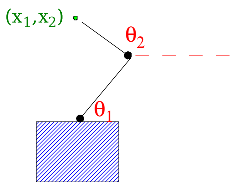


8

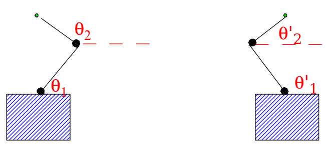
Robot Arm Control: Kinematics

Desired behavior $y^* = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

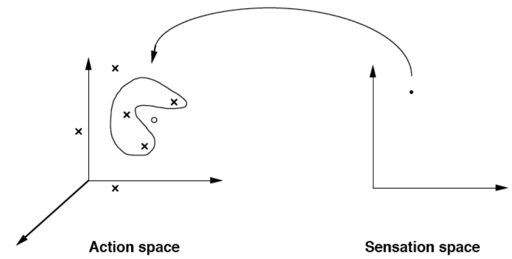
Control signal $u = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$

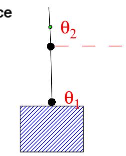


Two possible solutions:



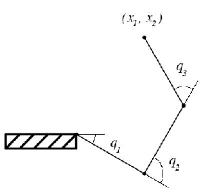
The Problem of Degeneracy



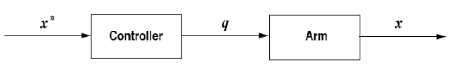


When the solution space is not convex, the “average” solution might not **be** a solution!

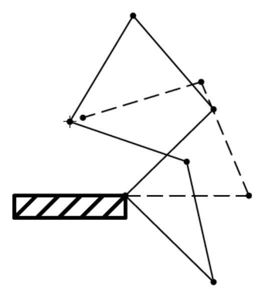
Three Joint Planar Arm



$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} l_1\cos(q_1)+l_2\cos(q_1+q_2)+l_3\cos(q_1+q_2+q_3) \\ l_1\sin(q_1)+l_2\sin(q_1+q_2)+l_3\sin(q_1+q_2+q_3) \end{bmatrix}$$

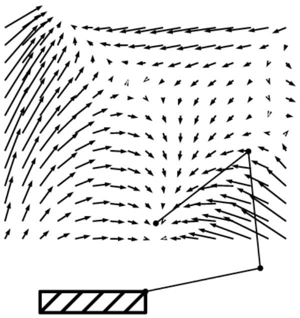


Inverse Kinematics is Non-Convex



Dashed line (average of the joint angles) does not reach the target.

Direct Inverse Model Makes Errors

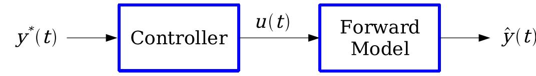


Train a backprop network as an inverse model by randomly sampling positions in joint space.

Learning asymptotes after 50,000 trials.

Performance error remains high due to non-convexity of the solution space.

Distal Learning Approach

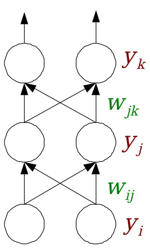


Train forward model backprop net, then freeze it.

Use backpropagated error signal to train the controller or inverse model.

Train controller using $y^* - \hat{y}$, *the predicted performance error.*

Back-Propagating the Error



Normal backprop calculates:

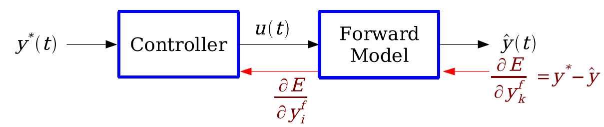
$$\delta_k = \frac{\partial E}{\partial y_k} = (y_k - t_k)$$

$$\delta_j = \frac{\partial E}{\partial y_j} = \sum_k \left(\frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j} \right)$$

For distal learning, also need:

$$\delta_i = \frac{\partial E}{\partial y_i} = \sum_j \left(\frac{\partial E}{\partial net_j} \cdot \frac{\partial net_j}{\partial y_i} \right)$$

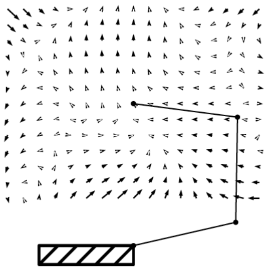
Error Signal From Forward Model



$\frac{\partial E}{\partial y_i^f}$ *acts like* $y_k^c - t_k^c$, *even though we don't know* $t_k^c = u^*$

Can introduce additional error terms to get solutions that minimize arm travel, maximize smoothness, etc.

Distally Trained Inverse Model



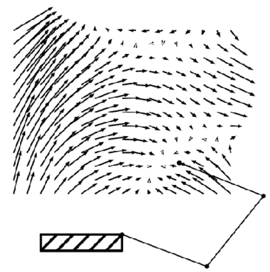
Train forward model first along with direct inverse model.

Freeze forward model.

Retrain inverse model using forward model as distal teacher.

Lower peformance error than the directly trained inverse model.

Goal-Directed Learning

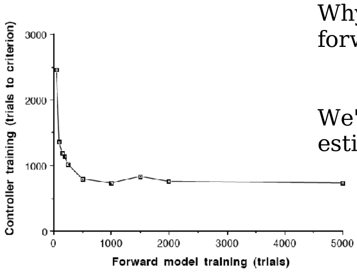


Distal learning is “goal directed”: concentrate on the regions of state space you care about.

Don't have to sample the whole space.

Error only needs to be low in the region of state space you care about.

Forward Model Needn't Be Precise



Why don't we need a precise forward model?

We're only using it to estimate the error gradient.

50 trials of training the forward model was “good enough” for successful learning.

Static vs. Dynamic Environments

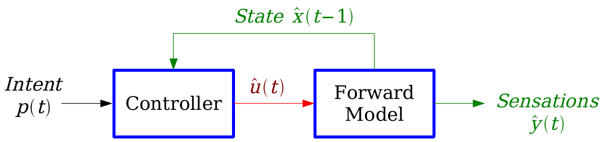
Static: actions have fixed effects. Current “state” does not determine the outcome of an action.

> Example: controlling an arm by specifying the θ_1 and θ_2 angles directly.

Dynamic environments: state matters. Actions have different effects in different states.

> Example: controlling an arm by specifying torques τ_1 and τ_2 for the motors.
>
> State $x(t) = (\theta_1, \theta_2)$
>
> Action $u(t) = (\tau_1, \tau_2)$

Distal Teacher with State



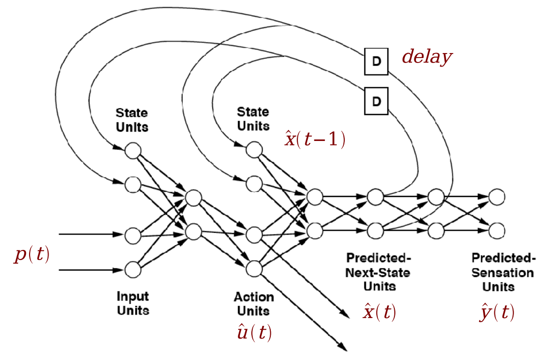
The forward model predicts both the next state $\hat{x}(t)$ *and the sensations associated with it,* $\hat{y}(t)$.

Controller function:

$$u(t) = f(p(t), x(t-1), W)$$

where W *denotes the learned weights.*

Recurrent Network with Forward Model



Training A Dynamic Model

Define performance error:

$$J = \frac{1}{2} E\{(y^* - y)^T (y^* - y)\}$$

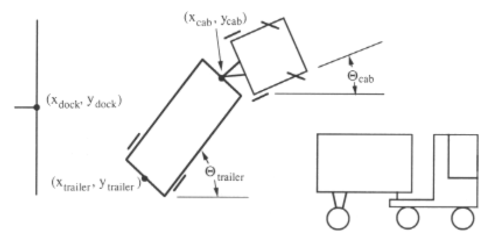
Let $u = h(x, y^*, W)$

Calculate the gradient:

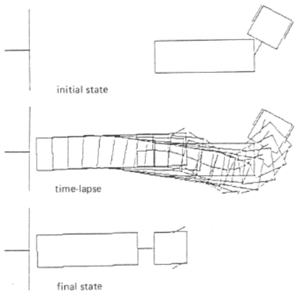
$$\nabla_W J = \frac{\partial J}{\partial y} \frac{\partial y^T}{\partial x} \frac{\partial x^T}{\partial u} \frac{\partial u^T}{\partial W}$$

$$= (y^* - y) \frac{\partial y^T}{\partial x} \frac{\partial x^T}{\partial u} \frac{\partial u^T}{\partial W}$$

Nguyen and Widrow's Truck Backer-Upper



Truck Backer-Upper



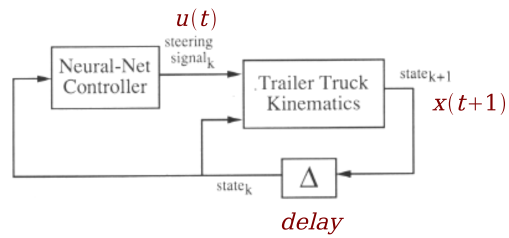
Desired state:

$$x_{trailer} = x_{dock}$$
$$y_{trailer} = y_{dock}$$
$$\theta_{trailer} = 0$$
$$\theta_{cab} = \text{(don't care)}$$

Control signal:
steering direction

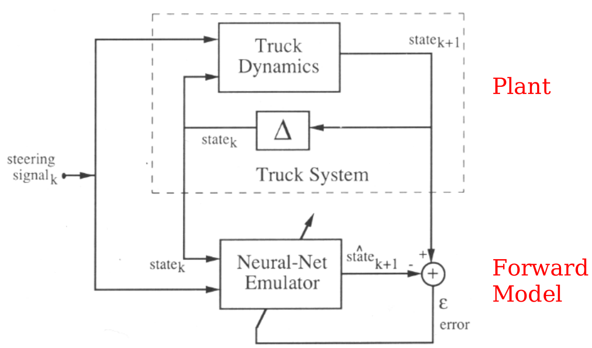
25

System Overview



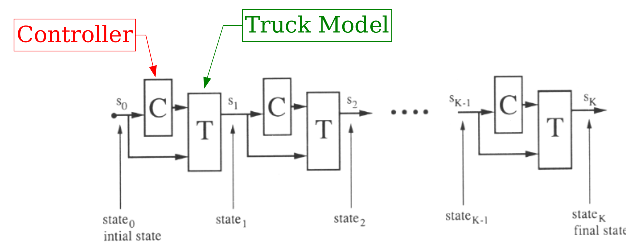
26

Train the Neural Net Truck Emulator



27

Backprop Through Time



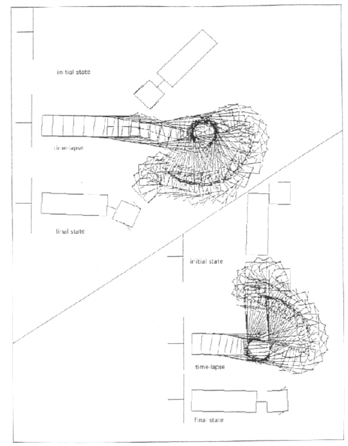
28

Training the Truck Backer-Upper

- Sixteen “lessons”.
- Start with easy problems:
  - Close to the dock
  - Heading correct within 30º
- Introduce harder cases:
  - Longer paths
  - Worse headings
- Total training: ~ 20,000 trials

29

Some Harder Training Problems

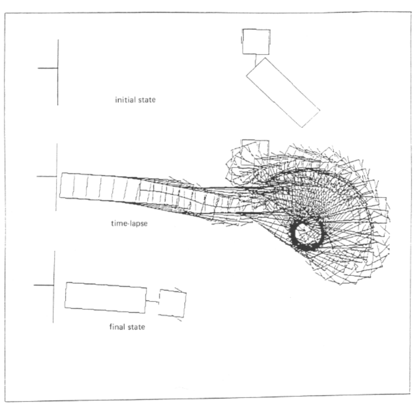


Facing wrong way

Jack-knifed

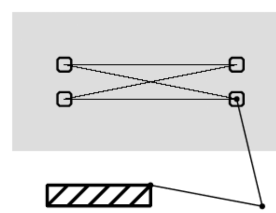
30

Results of Training



31

Modeling Arm Dynamics (Jordan)



32

Dynamic Arm Model

$$\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} l_1 \cos q_1(t) + l_2 \cos(q_1(t)+q_2(t)) \\ l_1 \sin q_1(t) + l_2 \sin(q_1(t)+q_2(t)) \end{bmatrix}$$

q	=	*joint position*	*state*
$\dot{q}$	=	*joint velocity*	*state*
$\ddot{q}$	=	*joint acceleration*	*desired behavior*
τ	=	*torque*	*control signal*

$$M(q)\ddot{q} \; + \; C(q,\dot{q})\dot{q} \; + \; G(q) \; = \; \tau$$

M = *inertia matrix*
C = *coriolis & centripetal terms*
G = *gravity term*

Learning Arm Dynamics

Assume sensors supply $\dot{q}$ *and* $\ddot{q}$

Forward dynamics:

$$\ddot{q} \; = \; M^{-1}(q)\left[\tau - C(q,\dot{q})\dot{q} - G(q)\right]$$

Inverse dynamics:
What torque τ *is needed to produce* $\ddot{q}$ *given* q *and* $\dot{q}$*?*

Tough problem: can't sample the state space randomly.

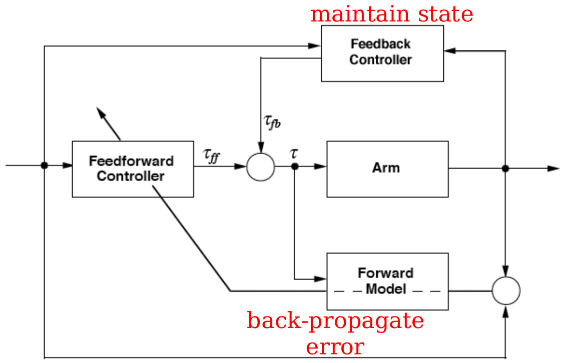
Forward Model Training Heuristics

1) Define random **equilibrium positions** for the arm, instead of generating random torques.

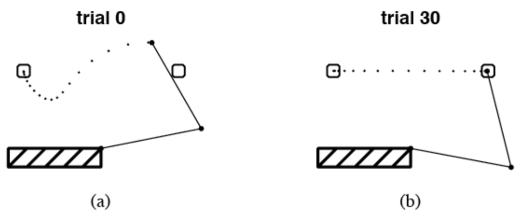
$$M(q)\ddot{q} \; + \; C(q,\dot{q})\dot{q} \; + \; G(q) \; = \; k_v(\dot{q} - \dot{u}) \; + \; k_p(q - u)$$

2) Use the target trajectories as controls to train the forward model.

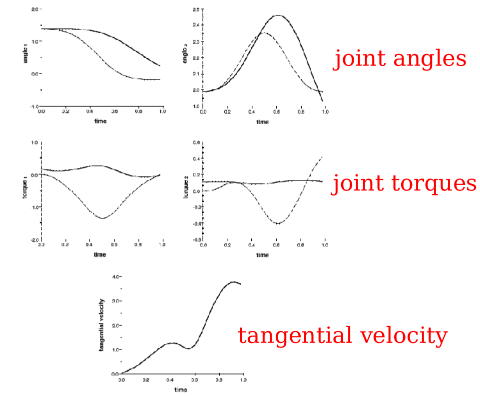
Composite Control System



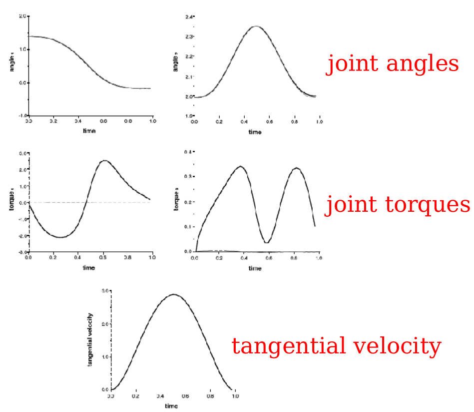
Performance Improves w/Learning



Before Learning



After Learning



Simultaneous Training

Can train forward and inverse models simultaneously:

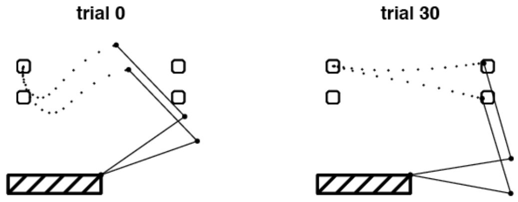
Train forward model on prediction error $y - \hat{y}$.

Train inverse model on performance error $y^* - y$.

Takes more trials to learn, but you don't have to train the forward model on the whole state space.

Focus only on desired trajectories.

Simultaneous Training of Forward Model and Controller



(a) (b)

Summary

- Difference between “supervised” and “unsupervised” learning is not so sharp.
  - The “teacher” is the forward model.
- Environment acts as a teacher for a forward model.
- Forward model can then supply teaching signal for training an inverse model.
  - Works even when the inverse is not unique.
- Can be extended to dynamic models, including those with unknown time delays.