16 A Cerebellar Model of Sensorimotor Adaptation

J. Spoelstra

16.1 Introduction

This chapter describes a neural network model of adaptation, based on the Martin *et al.* (1995) study of normal subjects and cerebellar patients throwing at a target after donning 30° prisms. The prisms caused subjects to miss the target by an angle corresponding to the prism deflection angle. With subsequent throws, however, normal subjects adjusted until they were once again throwing on target. After doffing the glasses the prism gaze-throw calibration remained and subjects made corresponding errors in the opposite direction. A cartoon sketch of the experiment is shown in figure 16.1.

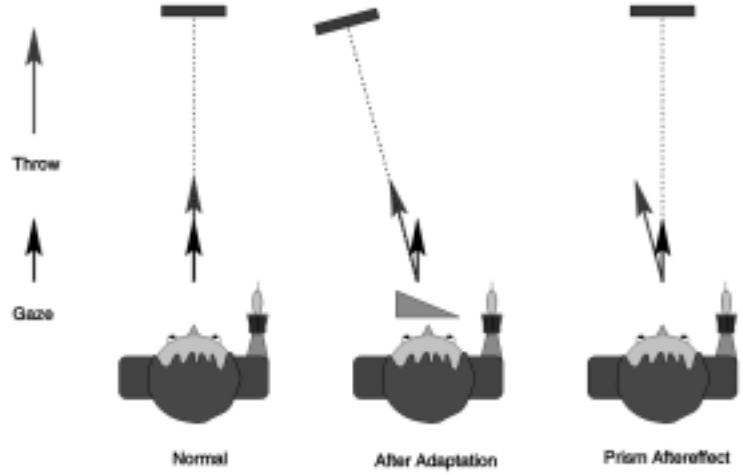

Figure 16.1
The experiment done by Martin et al. (1995). Subjects throw at a visual target while wearing prism glasses. Donning the glasses cause subjects to miss the target, but normal subjects adjust until they once again throw on target. After doffing the prism glasses, subjects make errors in the opposite direction and have to readjust their normal throwing.

From a modeling perspective, three results were particularly interesting:

- The calibration was throw-strategy specific: What was learned was not a general sensorimotor transformation; over- and under-hand throwing required independent adaptation.
- After a number of weeks of training subjects acquired the ability to throw accurately from the first throw, both with and without the prism glasses.
- Patients with lesions in the intermediate and medial cerebellum could not learn to throw accurately while wearing prisms, implicating the part of the cerebellum projecting "downstream" to the brainstem and spinal cord.

16.2 Model Description

Ito (1984) defined the basic building block of the cerebellar cortex and underlying nuclei as the microcomplex, shown in figure 16.2. Inputs arrive via mossy fibers (MF) to the granule cells (GC) whose axons bifurcate to form parallel fibers (PF) in the cerebellar cortex. Each Purkinje cell (PC) receives input from a large number of parallel fibers and one climbing fiber originating in the inferior olive (IO). Purkinje cells are the sole output from the cortex and inhibit the nuclear cells (NUC). Nuclear cell axons connect the cerebellum to the rest of the motor system, but have also been shown to produce an inhibitory effect on the same inferior olive cells that project to the overlying Purkinje cells to complete the loop. Learning occurs as long term depression (LTD) of parallel fiber-Purkinje dendrites after coactivation of parallel- and climbing fibers (Marr 1969; Albus 1971).

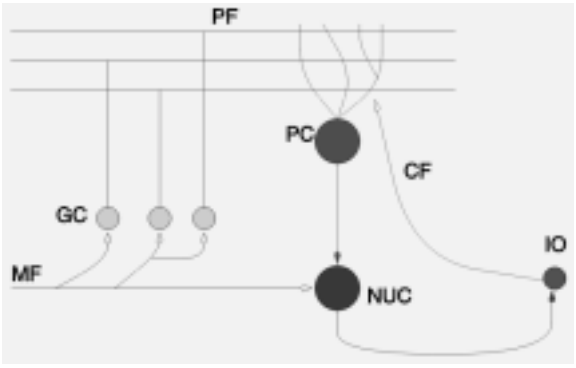

Figure 16.2
The major cell types and circuitry of the cerebellar cortex, also showing the loop made with the underlying nuclear cells and the inferior olive.

16.3 Model Implementation

Figure 16.3 shows the overall structure of the NSL implementation, including modules, submodules and input/output ports. The naming convention is that modules representing neuron populations are named xxx_layer, whereas high-level modules and other model systems "boxes" are named xxx_module.

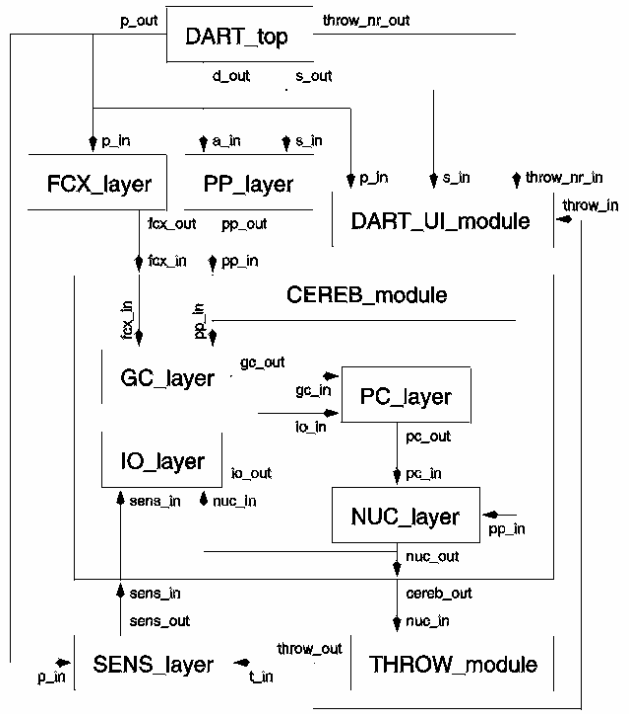

Figure 16.3
A box diagram of the NSL simulation code showing the different NSL modules with their inputs and outputs.

Neuron Populations

All neurons are modeled as having firing rate f computed from the membrane potential p using the sigmoid function:

$$f(p) = F_{\max} \frac{1}{1+\exp(-\alpha(p-\beta))} \tag{16.1}$$

with $F_{\max}$ the maximum firing rate, α determining the slope and β the offset of the sigmoid. The membrane potential is simply the weighted sum of the inputs to the neuron

$$p = \sum_{j \in A} I_j \tag{16.2}$$

with I_j the current synaptic input from neuron j and A denoting the set of projecting neurons.

PP_layer

The cerebellar granule cell layer receives input from two other layers, both containing a coarse coding of physical variables. The first input, putatively called posterior parietal (PP), is a 10x10 array with coordinates ranging from (0,0) to (9,9) and codes the arm configuration at the end of the throw. Because we are only interested in the horizontal throw direction, only the arm yaw angle relative to the head direction is represented. We also want to distinguish between over- and underhand throwing, so the PP layer arbitrarily codes both aiming angle (where the target appears visually) and throw strategy (over-/underhand).

A group of cells in a circular region with diameter of roughly 6 grid units were activated simultaneously, with activity maximal in the center and tapering off from there. As displayed by the NSL system, throw direction was coded on the Y-axis, with strategy on the X-axis. Using our coding convention, planning an overhand throw at a target centered in the visual field would cause a bump of activity centered at (3,4.5). If an underhand throw is planned the activity would be centered at (7,4.5).

The NSL code for generating this input is shown below. The parameter pp_sep determines the separation between the activity bumps for overarm and underarm throwing respectively, while pp_noise determines what portion of the signal will be generated by random to simulate noise. The inputs s_in and a_in represent throw strategy (0 for overarm, 1 for underarm) and aim direction respectively.

```
public void simRun(){
   int i,j;
   double mx, my;
   double dx,dy;

   if(s_in < .5)   // throw = over
      my = 4.5 - pp_sep/2.;
   else            // throw = under
      my = 4.5 + pp_sep/2.;
   mx = 4.5 + 4.5*a_in/30.; // Fit [-30:30] in [0:9]
   for(i=0;i<10;i++){
      dx = mx - i;
      for(j=0;j<10;j++){
         dy = my - j;
         pp_out[i][j] = pp_noise * nslRandom() +
           (1. – pp_noise) *
         nslExp(-1.*(dx*dx/sx2 + dy*dy/sy2));
      }
   }
}
```

FCX_layer

The second input layer contains a population code indicating an awareness of wearing prism glasses. Given enough time subjects could learn to throw accurately both with and without the prism glasses, indicating that the cerebellum received some information, possibly from the frontal cortex (FCX), telling it whether this was a prism-on or prism-

off trial. In the model a 10x4 array was used with the Gaussian bump of activity centered around (2.5,2) normally, and at (2.5,7) when prism glasses were worn.

In the NSL code below p_in is the prism angle while the parameter fcx_noise determines the noise level as for the PP input.

```
public void simRun(){
    int i,j;
    double mx, my, dx, dy;

    mx = 1. + 9.*p_in/50.; // Fit [0:50] in [1:10]
    my = 1.5;
    for(i=0;i<10;i++){
        dx = mx-i;
        for(j=0;j<4;j++){
               dy = my - j;
               fcx_out[i][j] = fcx_noise*nslRandom() +
               (1.-fcx_noise)*nslExp(-1.*(dx*dx/sx2 +
                  dy*dy/sy2));
        }
    }
}
```

GC_layer

Input arrives at the cerebellum via mossy fibers from the two input regions PP and FCX. Granule cells provide the input to the cerebellar cortex and are represented by a 30x30 array. In the real cerebellum each granule cell synapses with on average four mossy fibers—in the model four inputs were randomly selected with varying probability from the two input regions. The result is that the granule cell layer in a sense acts as the hidden layer in a multi-layer perceptron artificial neural network by providing nonlinear combinations of the raw inputs.

In order to produce this random mapping from the two input matrices onto the 30x30 GC grid, 5 arrays were set up in the initModule procedure: For each of the 3600 synapses (30x30x4) an input is selected randomly from the two input matrices. Vectors Xo and Yo record the coordinates on the input matrix; Xd and Yd record the coordinates on the GC matrix; and src records which of the two inputs was chosen.

During execution of the model the simRun method uses these vectors to map elements of the input matrices onto the GC inputs. The model is sensitive to GC parameters, so a number were made a available to the user for experimentation: The number of inputs each granule cell receives is determined by gc_nd; gc_dist determines the fraction of PP inputs versus FCX inputs chosen; gc_offset and gc_slope determine cell properties as described above.

```
public void initModule(){
   int gx,gy,i,x,y;
   double td;
   w = 1./((double)gc_nd);
   // Create mapping function
   NC = 0;
   for(gx=0;gx<30;gx++){
      for(gy=0;gy<30;gy++){
      for(i=0;i<gc_nd;i++){
         Xd[NC] = gx;
         Yd[NC] = gy;
         if(NslRandom() < gc_dist){ // PP input
            src[NC] = 0;
            td = (NslRandom()*5. + 3.);
            Xo[NC] = (int)td;
            td = (NslRandom()*10.);
            Yo[NC] = (int)td;
         } else { // FCX input
                src[NC] = 1;
                td = (NslRandom()*10.);
                Xo[NC] = (int)td;
                td = (NslRandom()*2. + 1);
                Yo[NC] = (int)td;
            }
            NC++;
         }
      }
   }
}
```

```
public void simRun(){
   int i,j;
   int mx,my,ix,iy;

   // Map inputs onto 30x30 array using mapping function
   gc_mp = 0.;
   for(i=0;i<NC;i++){
      mx = Xd[i];
      my = Yd[i];
      ix = Xo[i];
      iy = Yo[i];
      if(src[i]==0)
         gc_mp[mx][my] = gc_mp[mx][my] + pp_in[ix][iy];
      else
         gc_mp[mx][my] = gc_mp[mx][my] + fcx_in[ix][iy];
   }

   gc_mp = w * gc_mp;
   for(i=0;i<30;i++){
      for(j=0;j<30;j++){
         gc_out[i][j] =
            f_max/(1.+nslExp(gc_slope*(gc_offset-
               gc_mp[i][j])));
      }
   }
}
```

PC_layer

Granule cell axons bifurcate to give rise to the parallel fibers in the cortex that synapse with the Purkinje cell (PC) dendrites. The PCs are modeled as a 2x5 array. Parallel fibers run parallel to the X-direction and are modeled to span the entire width of the cerebellar patch modeled. Thus, if a PC receives input from one granule cell in a row, it receives input from all the granule cells in that row. The synaptic weights are excitatory and modifiable.

The section of NSL code below, taken from the simRun method shows how GC inputs are mapped onto the PC layer so that each PC receives input from a beam of GCs comprising one third of the total GC population. All the weights are stored in a single large vector.

```
// GC inputs
    pc_mp = 0.;
    wc = 0;
    for(px=0;px<2;px++){
        for(py=0;py<5;py++){
            beam_start = py*30/5;
            for(gx=0;gx<30;gx++){
                for(y=0;y<10;y++){
                    gy = (beam_start + y)%30;
                    pc_mp[px][py] = pc_mp[px][py]
                        + w[wc] * gc_in[gx][gy];
                    wc++;
                }
            }
        }
    }
```

We follow the current thinking that learning in the cerebellum occurs at the parallel fiber-Purkinje synapses and specifically that long term depression (LTD) of synaptic weights occur with simultaneous granule (pre synaptic), Purkinje (post synaptic) and climbing fiber activity. In order to prevent all weights systematically decreasing to zero, it is postulated that long term potentiation (LTP) will occur if pre- and post-synaptic activity is paired without climbing fiber activity.

Climbing fibers originate in the inferior olive (IO) and project topographically to the Purkinje layer: Each PC receives only one climbing fiber from the IO (Ito 1984). In this model we do not address the real-time role of climbing fiber activity on the firing rate of PCs; the inputs from the inferior olive (IO) are used solely as training signals.

The learning rule can be formalized as:

$$\Delta w = -\alpha F_g F_p \left(F_{io} - F_{io}^{back} \right) \tag{16.3}$$

with w the synaptic efficacy at one of the parallel fiber-Purkinje synapses, α some constant, F_g the firing rate of the granule cell, F_p the firing rate of the Purkinje cell, F_{io} the climbing fiber activity and F_{io}^{back} the tonic activity rate of the IO cells. IO activity below the tonic rate will result in LTP while any activity higher than the tonic rate will cause LTD.

In the NSL implementation below the same loop structure is used as above to make clear which PC, GC and IO cells are used when updating a specific weight. The test against getCurTime is made to ensure that all the inputs have filtered through the various stages of the process.

```
// Learning
if(system.getCurTime()>.055){ // give others time to settle
   (dart to fly)
   wc = 0;
   for(px=0;px<2;px++){
      for(py=0;py<5;py++){
         beam_start = py*30/5;
         for(gx=0;gx<30;gx++){
            for(y=0;y<10;y++){
               gy = (beam_start + y)%30;
               w[wc] = w[wc]
                  + alpha * (gc_in[gx][gy]*.01) *
                     (io_in[px] - 2.);
               if(w[wc] < 0.)
                     w[wc] = 0.;
               else if(w[wc] > 1.)
                     w[wc] = 1.;
               wc++;
            }
         }
      }
   }
}
```

Purkinje cells inhibit nuclear cells which in turn inhibit IO cells, producing a stable system: Any activity (disturbance) in the IO higher than F_{io}^{back} will cause a decrease in the PC firing, leading to an increase in nuclear cell activity which inhibits the IO cell. Stability is reached when nuclear activity is such that inhibition has all IO cells firing at F_{io}^{back}. One could think of the nuclear cells providing an *expectation* of the disturbance.

NUC_layer

The PC layer projects topographically onto the 20x1 nuclear layer. Each nuclear cell synapses with all the PCs in its column. These synapses are fixed and inhibitory. Nuclear cells also receive topographical projections with fixed weights from PP. Each nuclear cell receives input from a column of PP cells (coding aim direction) so that without PC intervention (no adaptation) normal throws go in the aim direction. In order to facilitate nuclear cell activity through PC disinhibition, the offset and slope parameters for the nuclear cells are set so that the cells are tonically active at about 10% of their maximum firing rate.

```
public void simRun(){
   int i,j;
   int ix;
   double td;
   // Map PP and PC inputs onto 2x1 array
   nuc_mp = 0.;
   for(i=0;i<10;i++){
      ix = i/5;
      for(j=0;j<10;j++){
            nuc_mp[ix] = nuc_mp[ix]+ 2.*pp_in[i][j];
      }
   }
   for(i=0;i<2;i++){
      for(j=0;j<5;j++){
            nuc_mp[i] = nuc_mp[i] - .2 * pc_in[i][j];
      }
   }
   for(i=0;i<2;i++){
      nuc_out[i] = f_max/(1.+NslExp(slope*(offset-
         nuc_mp[i])));
   }
}
```

IO_layer

As discussed above, each IO cell receives inhibitory projections from the nuclear cells and an excitatory connection from a sensory layer that indicates an error in performance. An interesting aspect is that the IO cells receive inhibition not only from the corresponding nuclear cell (which closes the negative-feedback learning loop), but also from the other (opposing) nuclear cell. The reason for this is that the output of the nuclear cells drive the direction of the eventual throw in a push-pull manner. In such a system, if the slightest disparity exists between LTD and LTP, small random errors will eventually drive the system to saturate with all weights at either their maximum or minimum values. By adding inhibition from the opposing side, however, any coactivation suppresses IO activity (which would increase NUC activity by decreasing PC weights) and steers the system towards reciprocal nuclear cell activation.

```
public void simRun(){
   int i;
   double nuc_act;

   nuc_act = nuc_in[0] + nuc_in[1];
   io_mp = sens_in - .01*nuc_act;
   for(i=0;i<2;i++){
      io_out[i] = f_max/(1.+NslExp(slope*(offset-io_mp[i])));
   }
}
```

SENS_layer

We postulate that a system (also PP) codes the perceived error in the final arm configuration or dart flight direction. This system then projects onto the IO layer where it is combined with inhibition from the nuclear cells to generate the cerebellar training signal. Each of two cells are proportionally receptive to either a leftward or rightward throw error. The module takes as input both the throw direction and the prism angle.

```
public void simRun(){
   double Derror;

   Derror = p_in - t_in;
   if(Derror < 0.){ /* go leftward */
      sens_out[0] = .1-Derror/10.;
      sens_out[1] = 0.1;
   } else { /* go right */
      sens_out[0] = 0.1;
      sens_out[1] = .1+Derror/10.;
   }
}
```

High-level modules

CEREB_module

This module is simply a convenient abstraction of the cell layers comprising the cerebellar part of the model. It does not do any processing, but simply instantiate its child modules and pass on inputs and outputs.

THROW_module

The output of the model is the yaw direction of the throw, derived from the activity of the two nuclear cells. Gilbert and Thach (1977) reported that cerebellar nuclear cells firing is related to arm yaw angle at the end of a trial. Following the hypothesis that cerebellar output influence brainstem motor pattern generators, it is assumed that each synergy cell will activate a combination of spring-like muscles to pull the final arm position to a side in a push-pull configuration. In a simplified model, the throw direction can be computed as the ratio of the activity of one cell to the total activity in both cells as shown below. The formula used gives a range of [-100:100] for the throw direction.

```
public void simRun(){
   throw_out = (.5 - (1.+nuc_in[0])/(2.+nuc_in[1] +
      nuc_in[0]))*100.;
}
```

DART_top

This module acts as a controller, automating the execution of specific experiments by executing a predetermined sequence of trials. A trial consists of setting up the inputs, then letting the simulation run for 6 steps at the end of which the throw direction is computed and the cerebellar weights adapted. An example of an experiment would be to execute a number of warm-up throws, followed by 20 throws while wearing prisms and 20 throws after doffing the prism glasses.

DART_UI_module

Although NSL provides an interface for displaying model variables and setting parameters, this module incorporates Java code for a model-specific user interface. It uses the standard NSL ports and facilities for hierarchical variables to communicate with the model, but is designed to present the experimenter with a more intuitive interface for displaying results and facilitate access to the model parameters.

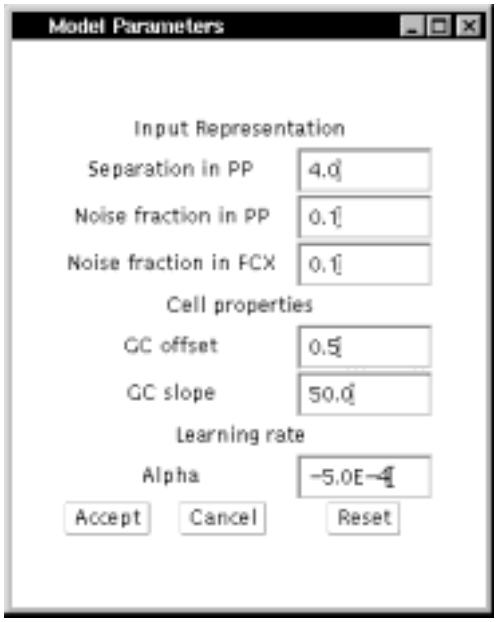

Figure 16.4
The custom user interface window for setting model parameters.

The custom interface window, shown in figure 16.5, pops up alongside the two NSL windows. The center panel indicates where consecutive throws hit the target, color-coded to indicate whether prisms were worn and differentiate between overarm and underarm throwing. From the menu bar users may select between three experiments or choose to set parameter values (shown in figure 16.4).

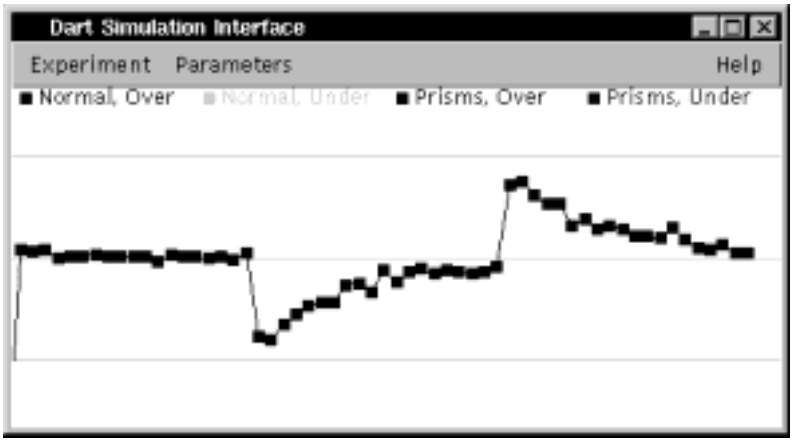

Figure 16.5
Simple adaptation experiment. 30° prism on at trial number 20, prism off at trial number 40

16.4 Simulation and Results[1]

There are three basic experiments: Simple adaptation to wearing prisms and readaptation to overcome the aftereffect; transfer between over- and underarm throwing; and the acquisition of two gaze-throw calibrations. To reproduce the data presented by Martin *et al.* (1995) and Kitazawa *et al.* (1995) it has to be shown that parameters exist to simultaneously satisfy 4 constraints:

1. Rate of adaptation: Approximately 30 throws are required to adapt to the prisms, slightly less to readapt. In both cases errors decrease exponentially, with readaptation occurring at a higher rate.
2. Magnitude of aftereffect: There is some variation, but the first throw after doffing the prisms usually misses by about 80% of the prism angle.

3. Transfer between over- and underarm throwing: Different levels of transfer should be possible.
4. Acquisition of two calibrations: This should take a large number of prism-on/prism-off adaptation trials. The initial error after donning or doffing the prisms should decrease exponentially.

Aftereffect of prism adaptation

Figure 16.5 depicts results of the basic experiment. A subject throws at a target, then dons 30° wedge prism glasses causing him to throw 30° off target. With repeated throws he improves until he once again throws on target. When the prisms are removed the subject misses by almost 30° on the opposite side and has to readjust his aim.

The activity of cells after adaptation to throwing with prisms as displayed by the NSL system is shown in figure 16.6. It can be seen how a depression in Purkinje cell activity (pc_out) leads to an increase in nuclear cell activity (nuc_out) that drives the direction of throwing.

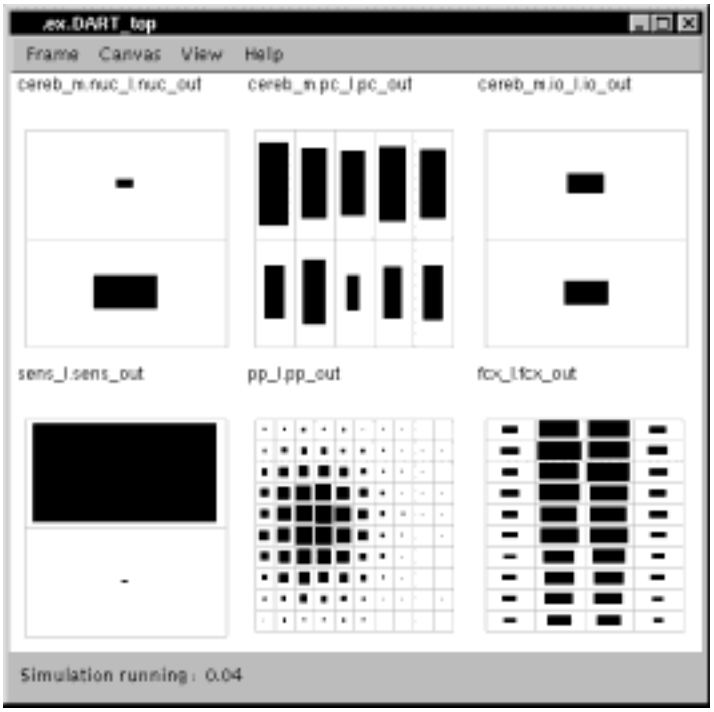

Figure 16.6
NSL output display of model variables after adaptation to prisms is complete. Note the depression in the activity of cells in the Purkinje layer and corresponding higher activity levels in the corresponding nuclear cells.

Transfer between over- and underhand throwing

Martin reported that some patients showed no transfer, i.e., the first underhand throw without prisms after adapting overhand throwing with prisms was on target, while others showed partial transfer. He also noted that for patients that showed partial transfer, the first overhand throw after readjusting underhand throwing was closer to target by an amount roughly equal to the amount adjusted during underarm adaptation. In simulation we can replicated this phenomenon by adjusting the separation between over- and underhand in PP (parameter pp_sep). Due to the Gaussian shape of the activity bump in PP, a separation of 6.0 leads to almost no overlap, while a separation of 1.5 leads to substantial overlap in representation. The results shown in figure 16.7 were obtained with the default setting of 4.0 and produces only limited transfer.

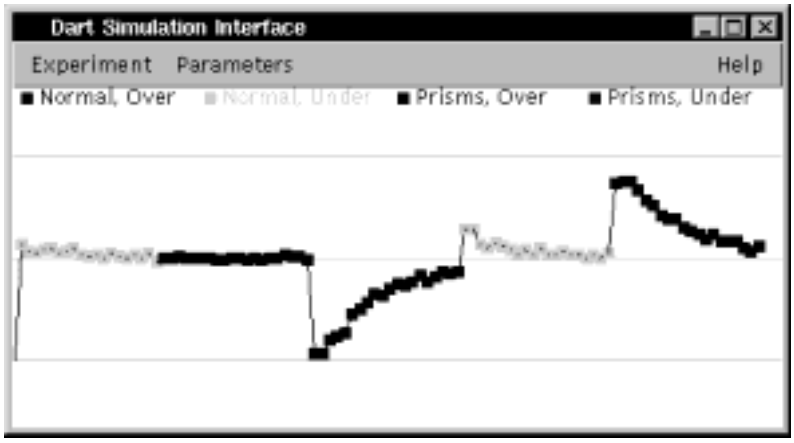

Figure 16.7

Transfer between over- and underhand throwing. At trial 20 30° prisms were donned while throwing overhand. At trial 40 the prisms were taken off but throwing was underhand. From trial 60 throwing is overhand again. In cases where the separation is large (solid line), the adaptation is independent—underhand throwing does not show the prism aftereffect. Where the separation is smaller, overhand adaptation affects underhand throwing and the overhand aftereffect is reduced proportional to the amount of adaptation that was required for underhand throwing.

Relating this result to human studies we would predict that the cortical representations for under- and overhand throwing could differ between subjects in terms of the amount of overlap. One interpretation could be that those who show partial overlap acquired underhand throwing as a variation of the overhand strategy, while those who showed no transfer learned two separate skills.

Relation to other models

The model shares many features with the AST model (Arbib, Schweighofer and Thach, 1994). In both models the cerebellar nuclear cells represent a population code of horizontal direction. However, in the AST model this direction is "added" as a rotation to the gaze angle in premotor cortex, whereas the current model does not need the complicated rotation computation and posits that the cerebellar nuclear neurons contain a direct code of the throw direction.

The AST model further required an artificial error detection system in register with the shoulder position that would activate an array of binary "leftward" and "rightward" cells in the inferior olive depending on where the throw went. While the current model does not yet offer a full explanation of the inferior olive, the error signal is generated in the IO through a combination of excitatory projections from PP and inhibitory projections from cerebellar nuclei to provide realistic IO firing rates and a stable learning system.

The learning circuitry based on inhibitory feedback from the cerebellar nuclear cells to the inferior olive has previously been suggested in models of cerebellar function in classical conditioning by Moore *et al.* [1989] and again by Bartha and Thompson (1995).

Martin et al. (1995) proposed a model where pairs of Purkinje cells, via disinhibition of nuclear cells, control eye, head and shoulder direction. The inputs are the current values of the controlled variables plus again the required "prism detector" input. However, no modeling results were published, so direct comparisons might not be appropriate.

16.5 Summary

The model described in this chapter shows how observed behavior could be generated in a cerebellar-like structure. In this context we offer explanations to the following questions:

What influences partial vs. zero transfer between over- and underhand throwing? The model can replicate the behavior by varying a single parameter that controls representations in (possibly) posterior parietal cortex.

What is the function of the known inhibitory projections from cerebellar nuclei to the inferior olive? The model demonstrates that when combined with a plausible cerebellar learning mechanism that incorporates both LTD and LTP, the loop results in a stable learning system that will adapt to provide the correct output and encourage reciprocal activation.

Notes

1. The Cerebellar model was implemented and tested under NSLJ.