A Connectionist Model of Motivation

Thomas E. Portegys, Lucent Technologies, portegys@lucent.com

Abstract

This paper presents Mona, a connectionist model of motivation implemented by a network of neuron-like components whose interactions drive behavior toward goals which reduce homeostatic needs. The network incorporates a short term memory capability allowing it to model a state-space with relatively few neurons.

Introduction

This paper presents Mona, a connectionist model of motivation. In this context, a motive, instead of being a type of goal state [4], is a function which drives behavior toward goals which reduce homeostatic needs. In living organisms these are basic needs such as thirst, hunger, sex, etc. The earliest neurological mechanisms evolved to ensure survival and reproduction by satisfying these needs. Moreover, given nature's penchant for creating new capabilities by extending and adapting old ones (the reptilian, mammalian, and neocortical layers of the human brain a case in point), it is plausible that these fundamental mechanisms underpin more intelligent behavior.

Mona's neural network incorporates a short term memory capability which allows it to model a state-space with relatively few neurons, thus addressing the problem of managing the intractable size of real-world state-spaces.

Mona uses the same "hardware" as a predecessor, GIL (a Goal-directed Inductive Learner) [6], interacting with its environment as shown in Figure 1. All knowledge of the state of the environment is absorbed through "senses"; there are no special modalities or channels by which instructions or meta-information are given. Responses are expressed to the environment with the goal of eliciting sensory inputs which are internally associated with the reduction of needs.

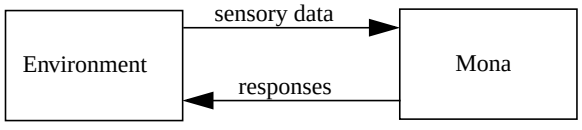

Figure 1 - The Mona/environment interaction

Human and animal intelligence is performed by a network of neurons which operate by mutual excitation and inhibition [2,8]. Mona is an abstraction of a natural neurological system consisting of a network of computational units, each of which is capable of receiving and expressing mutually mediating influences. To elucidate by example, consider a functional view of the nervous system of a simple organism which controls feeding behavior, shown in Figure 2. Feeding consists of the sequence of catching, killing and eating prey.

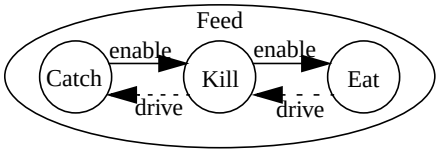

Figure 2 - Feeding control

"Feed", "Catch", "Kill", and "Eat" are neurons which fire when their namesake events occur. The solid arrows are enabling (or disabling) signals directed from one neuron to another; these signals are analogous to the excitatory and inhibitory influences of living neurons. The dotted arrows are drive, or motivating, signals derived from the organism's need for food. The above is read as follows. When the organism becomes hungry, the goal associated with the need of hunger-reduction, "Eat", becomes a source of drive signals propagating to antecedent neurons in a kind of "bucket brigade" [1] from primary to secondary goals, causing them to become motivated, or capable of responding. Once the prey is caught, the killing neuron is enabled, and once that is done, the eating neuron is enabled. The enabling of a neuron means that it is sensitized, or ready to fire. Thus a neuron's state consists of the 3-tuple (drive, enablement, firing). The events with which neurons are associated can be drawn from sensors, responses, or, as in the case of the mediating "Feed" neuron, the states of component neurons.

Neurons maintain a base level of enablement, analogous to long term memory, which is dynamically modified, as part of short term memory, to accomplish a concerted operation. For example, the "Feed" and "Catch" neurons might be enabled by default, while the "Kill" and "Eat" neurons rest in a disabled state awaiting the catching of prey. This would prevent an attempt to kill an object being caught by the organism for the purpose of mating or building a nest.

The ability of neurons to disable other neurons allows further opportunity for context-dependent cooperation. For example, it would be sensible for the "Catch" neuron to disable itself upon firing to prevent the seizing of prey while a catch is being eaten.

Description

The events which neurons represent can be drawn from sensors, responses, or the states of “component” neurons, calling for three types of neurons. Neurons attuned to sensors are “receptors”, those associated with responses are “motors”, and those mediating other neurons are “mediators”. A mediator neuron controls the transmission of drive and enablement through the sequence of its component neurons.

Here is an example task: Mona must get into her home from somewhere out in the world, a locked door barring the way inside, thus necessitating the use of a key to unlock the door. She needs to know several things, such as how to get to the door, how to unlock the door, and how to enter her home through the unlocked door. Mona must produce a sequence of responses to proceed from an initial keyless condition in the world to her home.

Figure 3 depicts the portion of Mona’s neural network which manages the entering of home through an unlocked door. The house-shaped objects are receptor neurons, such as the one marked “Door”; the inverted houses are motor neurons, such as “Move”; and the diamonds are mediator neurons, such as “Enter home”. The numbers in parentheses indicate drive levels, which will be discussed presently; suffice it to say for now that the “Home” receptor has been associated with the reduction of a need, and is thus a goal for Mona. The numbered arrows proceeding from a mediator indicate a sequence of neurons mediated by it, known to the mediator as its “events”. In this case, “Enter home” mediates a sequence of events associated with the receptor “Door”, the motor “Move”, and the receptor “Home”. This mediator thus governs the process of entering home by moving through a door. The type of mediation exerted by “Enter home” is an enabling one, meaning that it allows firing events to propagate enabling influences. Although not depicted in this example, a disabling mediator has dotted arrows instead of solid.

Initially the door is locked, meaning that the “Enter home” mediator is disabled, or not expected to function, and this is represented by the dotted outline of the mediator. In order to enable “Enter home”, another mediator must come into play: “Enable enter home”. This mediator will enable the “Enter home” neuron when the “Unlock door” neuron fires.

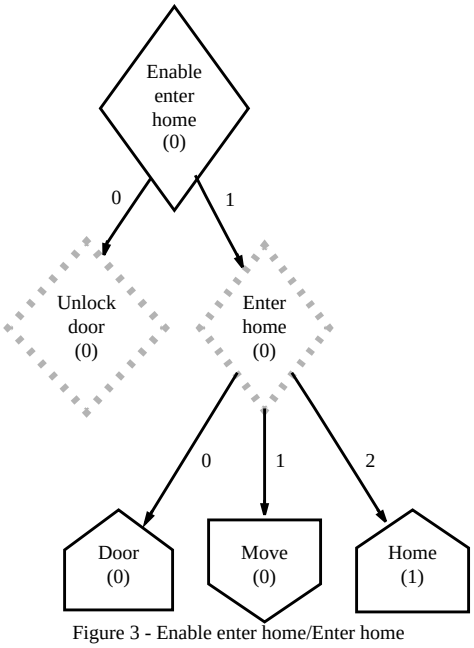

Figure 3 - Enable enter home/Enter home

However, the “Unlock door” neuron is also in a disabled state, requiring “Get key”, shown in Figure 4, to fire as a precondition - the door cannot be unlocked without the key.

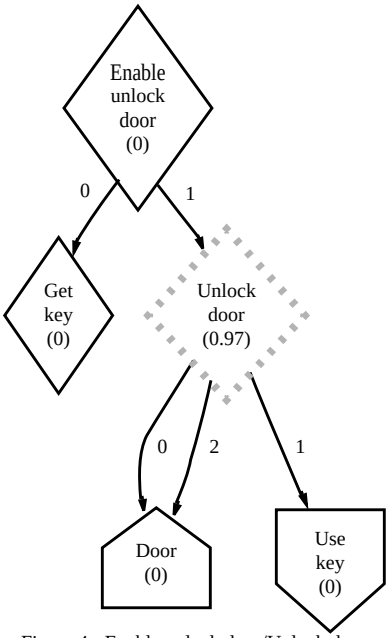

Figure 4 - Enable unlock door/Unlock door

The final two pieces are supplied in Figure 5: how to get a

key (“Get key”), and how to get to the door (“Go to door”).

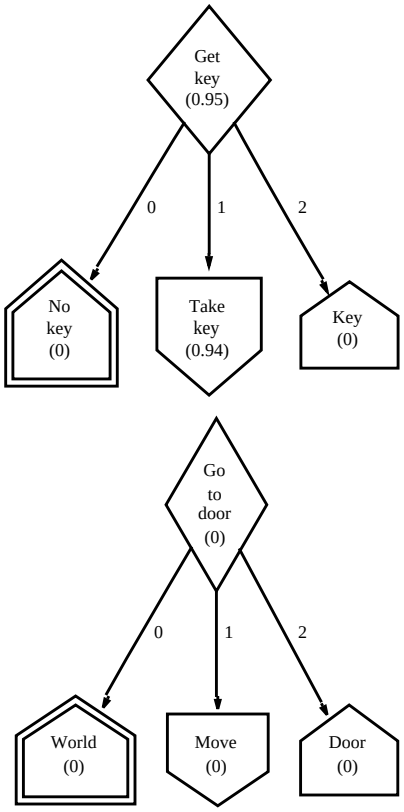

Figure 5 - Get key/Go to door

Since these diagrams show the initial state of network, the “World” and “No key” receptors are firing, denoted by the double outlines on their graphical symbols.

When this example is run on a software implementation, the following trace of firing neurons is obtained:

Receptor firing: No key
Receptor firing: World
Motor firing: Take key

Receptor firing: Key
Mediator firing: Get key
Receptor firing: World
Motor firing: Move

Receptor firing: Key
Receptor firing: Door
Mediator firing: Go to door
Motor firing: Use key

Receptor firing: Key
Receptor firing: Door
Mediator firing: Unlock door
Mediator firing: Enable unlock door
Motor firing: Move

Receptor firing: Key
Receptor firing: Home
Mediator firing: Enter home
Mediator firing: Enable enter home

The following textual notation can also be used to more concisely describe a network.

Receptor neuron:

Receptor{“<name>”: (<sensors>)}

where <sensors> describes the sensory condition under which the receptor fires.

Motor neuron:

Motor{“<name>”: (<responders>)}

where <responders> describes the responder output when the neuron fires.

Mediator neuron:

Mediator{“<name>”: <enabled value>/<enabling value>(<events>)}

where <enabled value> is the mediator’s initial state of enablement, either “enabled” or “disabled”; <enabling value> is the type of enabling influence on its event neurons, either “enabling” or “disabling”; and <events> is a comma-separated sequence of mediated neuron names.

Appendix 1 contains pseudo-code details of the following functions. Neurons use a simple firing threshold function. Receptor and motor neurons fire when their associated sensory/response events occur. A mediator neuron contains an `eventFiring()` function which fires the mediator when each event in its sequence fires within the maximum delay imposed by the mediator’s `maxEventDelay` value. If the mediator is enabled, the `eventFiring()` function also allows enablement/disablement to be propagated from a firing component neuron to the next.

Mona’s *raison d’etre* is need-reduction. For this purpose, some receptors are associated with the reduction of needs and are thereby defined to be goals. For example, a warmth receptor would be associated with a reduction of feeling cold. The `drive()` function allows need to propagate from goal sources to other neurons in the network, attenuating to preferably drive “closer” neurons and to prevent endless propagation. Need causes a mediator neuron to perform a check: if it is enabled, it will pass the need into its expected event neuron in order to motivate it to occur; otherwise, it passes the need to its super-mediating neurons to motivate them to enable it. The mediator-specific `eventDrive()` function provides a way for a mediator to pass need from its final event to its expected event. Upon completion of propagation, the need resident in motor neurons is translated into the response potentials associated with those neurons. The system response is that associated with the

maximum potential value.

The network as a state-space model

The network must embody a model of the environment. However, instead of directly implementing a state-space of possibly intractable proportions, the enabling and disabling operations allow the "topology" of the network to be modified by the act of operating it. The network may be considered to be a hybridization of a logic engine and a state-space search engine having two key properties: (1) more than one state (neuron) can be current (firing) at a particular time, and (2) current states can "prove" (enable) or "disprove" (disable) the reachability of states. These properties allow a network to assume a large number of states relative to the number of neurons which comprise it.

As an illustration, consider the personal financial state-space shown in Figure 6, in which money is earned at a job (pocket), spent at a store (broke), and deposited/withdrawn at a bank (saved). For the sake of simplicity, let there be a single quantum of money in the economy, e.g., if the money is in the bank, more money cannot be earned. The space contains (3 places) X (3 money situations) = (9 states).

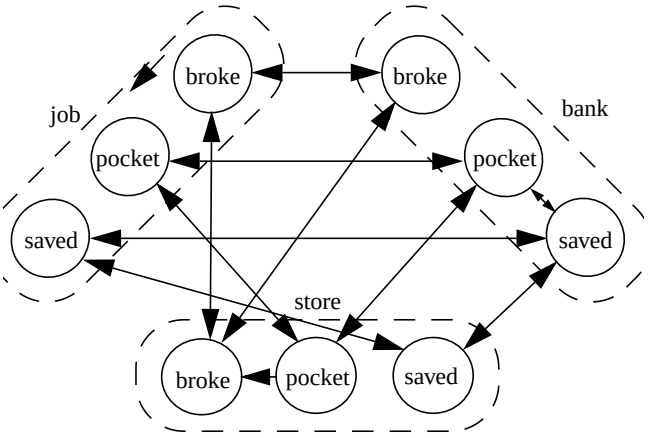

Figure 6 - Financial state-space

Figure 7 shows a network representation of the problem in abbreviated graphical form, for clarity. The '+'/'-' indicate enabling/disabling influences. It can be seen that moving between places (the top portion of the figure) is independent of the transactions which transpire at those places since the enablement states (for earn, spend, deposit, and withdraw) store the transaction possibilities. The addition of places not involved in monetary dealings, a reasonable real-world supposition, alters only the place transition portion of the network, avoiding the combinatorial expansion of the state-space model. For example, if "home", "park", and "museum" are added as places, the state-space expands by 9 states, while the network expands by 3 neurons.

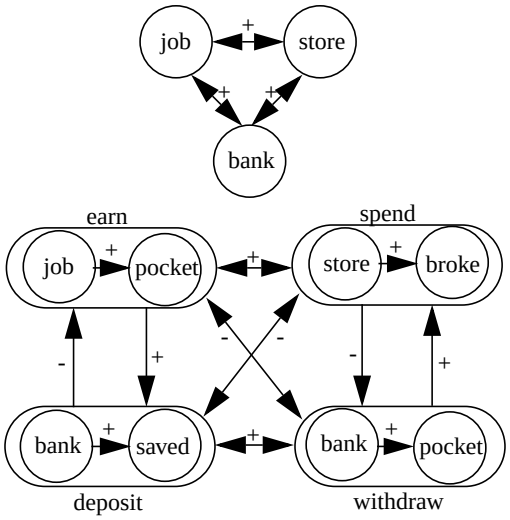

Figure 7 - Financial network

A comparison with other ANNs

In contrast with Hopfield and backpropagating ANNs [3,5], which are primarily stateless pattern classifiers, Mona employs a short term memory capability to navigate the environment toward goals. Short term memory is implemented by the retention of neural firing sequences and by the enabling and disabling operations, which modify the state of the network based on sensory events and responses motivated by needs. The incorporation of memory into ANNs is a topic of current interest [7].

Demonstration

This problem demonstrates how the neural network can be used to simulate a foraging ant. Natural ants are known to follow trails of chemical markers to get about. In this problem, the artificial ant must following a meandering trail of marks from its nest to a piece of food, which it must then carry back to the nest. The problem illustrates the interplay of mediator neurons, using mutual enablement and disablement, to guide the ant to a goal in an unpredictable environment.

A sample trail is shown in Figure 8. The ant starts at its nest and follows the trail marks to the cake. The trail is randomly generated in such a way that it never crosses itself. The ant senses what is at the current location, and has the ability to move forward and backward, grab and drop the food, and orient itself in the direction of the trail. Generally a trail leads straight on, so the most efficient strategy is to plunge ahead and orient upon leaving the trail. An initial positive need is associated with the receptor which detects the presence of food at the nest.

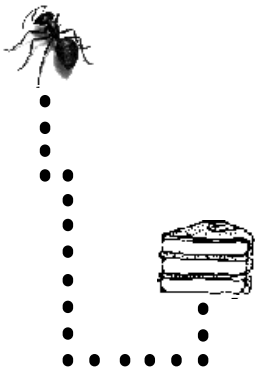

Figure 8 - Meandering ant trail

The following mediators were defined for this problem:

```
# Top-level control: “Forage” is to “Get food” then
# enable “Bring food” to bring it back to the nest.
Mediator{“Forage”: enabled/enabling (“Get
food”,”Bring food”)}
Mediator{“Get food”: enabled/enabling (“Grab
food”,”Orient”,”Mark”)}
Mediator{Grab food”: enabled/enabling
(“Food”,”Grab”,”No food”)}
Mediator{“Bring food”: disabled/enabling
(“Nest”,”Drop”,”Food at nest”)}

# After food obtained, reset “Bring food” to disabled
# state for next forage.
Mediator{“Disable bring food”: enabled/disabling
(“Food at nest”,”Bring food”)}

# “Travel trail” is the normal way to move.
# “To food” and “To nest” associate “Travel trail” with
# getting somewhere.
Mediator{“Travel trail”: enabled/enabling
(“Mark”,”Forward”,”Mark”)}
Mediator{“To food”: enabled/enabling (“Travel
trail”,”Food”)}
Mediator{“To nest”: enabled/enabling (“Travel
trail”,”Nest”)}

# If the ant steps off the trail (“No mark”), these
# control getting it back on:
# 1) Disable normal traveling (“Travel trail”).
# 2) Initiate “Trail search” to back-up and orient.
# 3) Once oriented, enable normal traveling.
Mediator{“Disable travel trail”: enabled/disabling
(“No mark”,”Travel trail”)}
Mediator{“Trail search”: enabled/enabling (“No
mark”,”Backward”,”Mark”,”Orient”,”Mark”)}
Mediator{“Enable travel trail”: enabled/enabling
(“Trail search”,”Travel trail”)}
```

Running the program on the sample results in the following responses to fetch the cake to the nest:

```
Forward X 5
Backward
Orient
Forward X 2
Backward
Orient
Forward X 8
Backward
Orient
Forward X 6
Backward
Orient
Forward X 3
Grab
Orient
Forward X 4
Backward
Orient
Forward X 6
Backward
Orient
Forward X 8
Backward
Orient
Forward X 2
Backward
Orient
Forward X 4
Drop
```

Conclusion

When a cat wants to catch a mouse, why does it wait at one mouse hole and not another? Perhaps because the context surrounding a particular mouse hole - how to get there, a window nearby - mediates the expectation of success. This paper describes a model of this type of motivated behavior: a network wherein needs drive events along enabled paths toward goals. To do this, the network incorporates a short term memory of the firing sequences and the enablement states of neurons.

In order to focus on the existing capabilities of Mona, of which there remains much to explore, a learning capacity was deferred; learning is thus a major area for investigation. It also seems possible that the neural model may be cast into parallel/specialized hardware to improve its performance.

The complete C++ source code for Mona and several problems may be downloaded from http://www.megsinet.net/portegys/.

References

[1] Forsyth, R. and Rada, R. (1986) *Machine Learning: Applications in expert systems and information retrieval*, Ellis Horwood Limited: Chichester, West Sussex.

[2] Holmes, W. and Rall, W. (1992) Electrotonic Models of Neuronal Dendrites and Single Neuron Computation. Thomas McKenna, Joel Davis, and Steven F. Zornetzer (Eds.) *Single Neuron Computation*, Academic Press: San Diego.

[3] Hopfield, J. and Tank, D. (1986) Computing with neural circuits: A model. *Science*, 233, 625-633.

[4] McClelland, D. (1987) *Human Motivation*, Cambridge University Press: Cambridge.

[5] Munakata, T. (1998) *Fundamentals of the new artificial intelligence: beyond traditional paradigms*, Springer-Verlag Inc.: New York.

[6] Portegys, T. (1986) GIL - an experiment in goal-directed inductive learning. Ph.D. dissertation, Northwestern University, 109 pp. (available from UMI at http://www.umi.com/).

[7] Roy, A. (1997) Panel Discussion at ICNN97 on Connectionist Learning, Levine, D. (Ed.) *Neural Networks*, Vol. II, No. 2.

[8] Thompson, R., Berger, T., and Berry S. (1980) Brain Anatomy and Function. Wittrock, M. (Ed.) *The Brain and Psychology*, Academic Press.

Appendix 1 - C++ pseudo-code

```
// Detect firing of event
Mediator::eventFiring(eventNumber)
{
   if (eventNumber != expectedEvent) return;
   if (eventNumber == finalEvent)
   {
      // Mediator firing - notify super-mediators.
      firing = TRUE;
      for (notify = eventNotify.first(); notify != NULL;
        notify = eventNotify.next())
      {
         mediator = notify->mediator;
         event = notify->eventNumber;
         mediator->eventFiring(event);
      }
      expectedEvent = 0;  // Reset event counter.
      return;
   }
   // Expect next event.
   expectedEvent++;
   eventTimer = maxEventDelay;
   // If enabled, propagate enablement.
   if (enabled == TRUE)
   {
      neuron = components[expectedEvent];
      neuron->enabled = eventEnablement;
   }
}

// Neuron drive
Neuron::drive(need) {
{
   if (need <= currentNeed) return;
   currentNeed = need;
   if ((need -= ATTENUATION) <= 0) return;
   // If enabled mediator, drive expected event.
   if (type == MEDIATOR && enabled == TRUE)
   {
      neuron = components[expectedEvent];
      neuron->drive(need);
      return;
   }
   // Drive super-mediators.
   for (notify = eventNotify.first(); notify != NULL;
        notify = eventNotify.next())
   {
      mediator = notify->mediator;
      event = notify->eventNumber;
      mediator->eventDrive(event, need);
   }
}

// Mediator event drive
Mediator::eventDrive(eventNumber, need)
{
   if (eventNumber != finalEvent) return;
   if ((need -= ATTENUATION) <= 0) return;
   if (enabled == FALSE)
   {
      // Drive super-mediators to enable this.
      for (notify = eventNotify.first(); notify != NULL;
        notify = eventNotify.next())
      {
         mediator = notify->mediator;
         event = notify->eventNumber;
         mediator->eventDrive(event, need);
      }
   } else {  // Drive expected event.
      if (expectedEvent != finalEvent)
      {
         neuron = components[expectedEvent];
         neuron->drive(need);
      }
   }
}
```