1 Organisation and interference

We are interested in creating programs which we are not able to write by hand. Many useful tasks fall into this category, such as image recognition, playing perfect Go, proof synthesis, and translation of natural language. In this paper (Intelligent Machinery, 1948) [1], Turing proposes a possible method by which this could be done, with a simple model of reinforcement learning.

Definition 1.1. Consider a class of machines built from some kind of standard components. Machines which are constructed in a systematic way to achieve some purpose are called **organised**. Otherwise, machines which are constructed in largely random way are called **unorganised**.

It is worth emphasising that this refers to the way in which the machine is constructed. It is also of course subjective; a construction could be regarded as organised by one person and unorganised by another.

We will be interested in modifying machines to make them more adept at performing a particular task. One way in which we could do this would be to rebuild or reprogram the machine entirely; we will call this **screwdriver** modification. In the language of Turing machines, this would be analogous to replacing the transition function of a TM by another. Another type is **paper** modification, which is modification of the machine's behaviour merely by communicating information to the machine. This is analogous to changing the code string which is fed to a universal Turing machine. We will mainly be interested in paper modification. In the context of modern neural networks, screwdriver modification is analogous to changing the neural net architecture, while paper modification is tuning the weights.

2 A-type and B-type machines

Informally speaking, a **node** is a memory unit which stores a single bit, and has two input terminals and zero or more output terminals. Each node is connected to a central clock which emits pulses at regular intervals; we write $N_t \in \{0, 1\}$ for the state of a node N at time $t \in \mathbb{N}$. If the input terminals of a node N are connected to the output terminals of nodes A and B, then the update rule is

$$N_{t+1} = 1 - A_t B_t = \overline{A_t B_t}.$$

Definition 2.1. An **A-type machine** consists of a finite set $\mathcal{N}$ of nodes, a subset $\mathcal{I} \subseteq \mathcal{N}$ of input nodes, a nonempty subset $\mathcal{O} \subseteq \mathcal{N}$ of output nodes, a function $s_{\text{init}} : \mathcal{N} \setminus \mathcal{I} \to \{0, 1\}$, and a function $i : \mathcal{N} \setminus \mathcal{I} \to \mathcal{N} \times \mathcal{N}$.

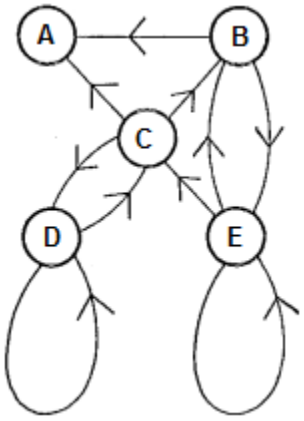
We draw A-type machines as a directed graph of nodes, where each non-input node has exactly two incoming edges, specified by the function i.

Definition 2.2. A **state** of an A-type machine is a function $s : \mathcal{N} \to \{0,1\}$. The time evolution of an A-type machine on input $h : \mathcal{I} \to \{0,1\}$ is a sequence of states $s_0, s_1, s_2, \ldots$ where

$$s_0(x) = \begin{cases} s_{\text{init}}(x) & x \notin \mathcal{I} \\ h(x) & x \in \mathcal{I} \end{cases} \qquad s_{t+1}(x) = \begin{cases} 1 - s_t(i_1(x))s_t(i_2(x)) & x \notin \mathcal{I} \\ h(x) & x \in \mathcal{I} \end{cases}$$

where i_1, i_2 denote projection of i onto the first and second coordinate. As above, we will usually write $x_t := s_t(x)$ to be the state of the node x at time t. The **output** of the machine at time t is $s_t|_{\mathcal{O}} : \mathcal{O} \to \{0,1\}$.

Example 2.3. Consider the following A-type machine:



The update rules are

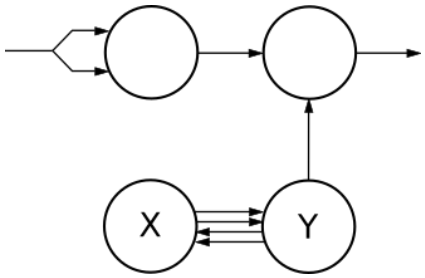
$$\begin{aligned} A_{t+1} &= 1 - B_t C_t \\ B_{t+1} &= 1 - C_t E_t \\ C_{t+1} &= 1 - D_t E_t \\ D_{t+1} &= 1 - C_t D_t \\ E_{t+1} &= 1 - B_t E_t. \end{aligned}$$

A possible run of such a machine (starting from a random initial configuration) is

time	1	2	3	4	5	6
A	1	1	0	0	1	0
B	1	1	1	0	1	0
C	0	1	1	1	1	1
D	0	1	0	1	0	1
E	1	0	1	0	1	0

Behaviour after this point is clearly periodic (as must be true of a deterministic machine with finitely many states).

Definition 2.4. A **B-type connection**[1] is



The nodes marked X and Y are called the **weight nodes**.

A **B-type machine** is any machine which is obtained by replacing every edge in an A-type machine by a B-type connection.

The behaviour of a B-type connection depends on the initial configuration of the two weight nodes.

- If $X_0 \neq Y_0$, then $Y_t = Y_0$ for all t.
- If $X_0 = Y_0$, then $Y_t = \begin{cases} Y_0 & t \text{ even} \\ \overline{Y_0} & t \text{ odd.} \end{cases}$

It follows that the output at time t of a B-type connection whose input node is A is

$$\overline{\overline{A_{t-2}}\, Y_{t-1}} = \begin{cases} 1 & Y_{t-1} = 0 \\ A_{t-2} & Y_{t-1} = 1 \end{cases} = \begin{cases} 1 & X_0 = 1 \text{ and } Y_0 = 0 \\ A_{t-2} & X_0 = 0 \text{ and } Y_0 = 1 \\ \text{alternates} & X_0 = Y_0. \end{cases}$$

Note that there is an additional delay of two moments in passing through a B-type connection. So with appropriate initial conditions of the weight nodes, the B-type connection can be made to act as either a normal A-type connection (ignoring the delay), or a constant signal 1, or to alternate between the two behaviours in consecutive timesteps.

3 Education of machinery

Our goal is to convert an unorganised machine into an organised machine by appropriate training.

[1] In Turing's paper, he defines a B-type connection as one which does not have the additional node in the top left. It turns out that in this case, the resulting class of B-type machines is too computationally weak to be useful. The B-type node (an A-type node with two incoming B-type connections) is essentially an OR gate. It is well known that OR gates are not functionally complete, so B-type machines are not capable of expressing all boolean functions. The fix we describe was proposed by Teuscher; see [2].

Turing's idea is to regularly provide a machine with *reward* and *punishment* signals. The description of a machine contains some uncertain information, and tentative choices are made for the value of nodes. Whenever a reward signal is received, the tentative choices are made permanent, and whenever a punishment signal is received, the tentative choices are randomly reassigned.

This has the advantage in that it allows a teacher to provide feedback without being able to write the program themselves. However there is a significant flaw in that it does not allow any 'fine-tuning'; a reward signal locks too much in place. It would of course be possible that only some of the tentative choices were actually correct, but the reward signal makes all of them permanent.

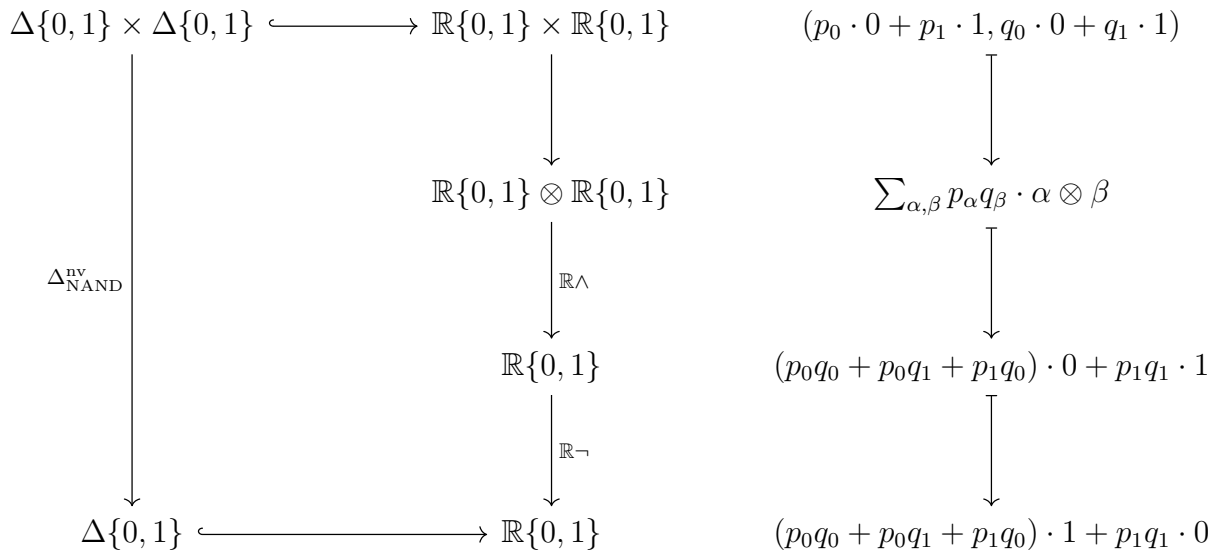
We will instead present a variant of Turing's idea which provides a greater degree of control, by including uncertainty in the nodes. The teacher can then provide graded feedback so that the distributions may converge towards the correct behaviour.

For a finite set X, let ΔX denote the set of probability distributions over X.

Definition 3.1. A state of a **probabilistic A-type machine** is a function $s : \mathcal{N} \to \Delta\{0,1\}$. The time evolution on input $h : \mathcal{I} \to \Delta\{0,1\}$ is a sequence of states $s_0, s_1, s_2, \ldots$ where

$$s_0(x) = \begin{cases} s_{\text{init}}(x) & x \notin \mathcal{I} \\ h(x) & x \in \mathcal{I} \end{cases} \qquad s_{t+1}(x) = \begin{cases} \Delta^{\text{nv}}_{\text{NAND}}(s_t(i_1(x)), s_t(i_2(x))) & x \notin \mathcal{I} \\ h(x) & x \in \mathcal{I} \end{cases}$$

where i_1, i_2 are projection and $\Delta^{\text{nv}}_{\text{NAND}}$ is the map making the following commute.



Note that if there is no uncertainty then this is simply an ordinary A-type machine. We define a probabilistic B-type machine in the obvious way.

For the remainder of this section, let $n = |\mathcal{O}|$ be the number of output bits of a fixed probabilistic B-type machine. Suppose that an external teacher has a way of assigning

a real number to each possible output string; that is, a function

$$c : \{0,1\}^n \to \mathbb{R}.$$

A higher value $c(\mathbf{x})$ corresponds to the teacher being less satisfied with the output $\mathbf{x}$.

Given a probabilistic B-type machine, let $\mathcal{W}$ denote the set of weight nodes. Assume that we run the machine for t timesteps, at which time the teacher looks at the output bits $p = s_t|_{\mathcal{O}} : \mathcal{O} \to \Delta\{0,1\}$. How should we adjust these input weights — more precisely, the distributions $\{s_0(w)\}_{w\in\mathcal{W}}$ — in response to the feedback given by c?

It is convenient to reparametrise the weight distributions by defining[2] $\mathbf{a} = (a_w^0, a_w^1)_{w\in\mathcal{W}}$ where

$$s_0(w) = \frac{e^{a_w^0}}{e^{a_w^0} + e^{a_w^1}} \cdot 0 + \frac{e^{a_w^1}}{e^{a_w^0} + e^{a_w^1}} \cdot 1,$$

as this will ensure that we remain inside the probability simplex after each adjustment.

Definition 3.2. Define the loss function of as the expected value of c over all possible output strings $\mathbf{x}$:

$$L(p) = \sum_{\mathbf{x}\in\{0,1\}^n} p_1(x_1)\dots p_n(x_n)c(\mathbf{x}).$$

Since p is a function of the weight vector $\mathbf{a}$, so is L; we define the update rule for the weights of the B-type Turing machine as

$$\mathbf{a}' = \mathbf{a} - \ell\nabla L(\mathbf{a}),$$

where ℓ is the learning rate. We update the weights using this rule for some number of steps after each reinforcement.

Example 3.3. Say that $n = |\mathcal{O}| = 1$. The weight vector $\mathbf{a} = (a_w^0, a_w^1)_{w\in\mathcal{W}}$ is updated according to the rules

$$(a_w^k)' = a_w^k - \ell \left.\frac{\partial p(0)}{\partial a_w^k}\right|_{\mathbf{a}} c(0) - \ell \left.\frac{\partial p(1)}{\partial a_w^k}\right|_{\mathbf{a}} c(1).$$

4 References

1. A.M. Turing, *Intelligent Machinery*, 1948.
2. C. Teuscher, *Turing's Connectionism*, Springer 2002.
3. J. Clift, D. Murfet, *Derivatives of Turing Machines in Linear Logic*, arXiv:1805.11813

[2]Note that the a_w^k are not uniquely determined.