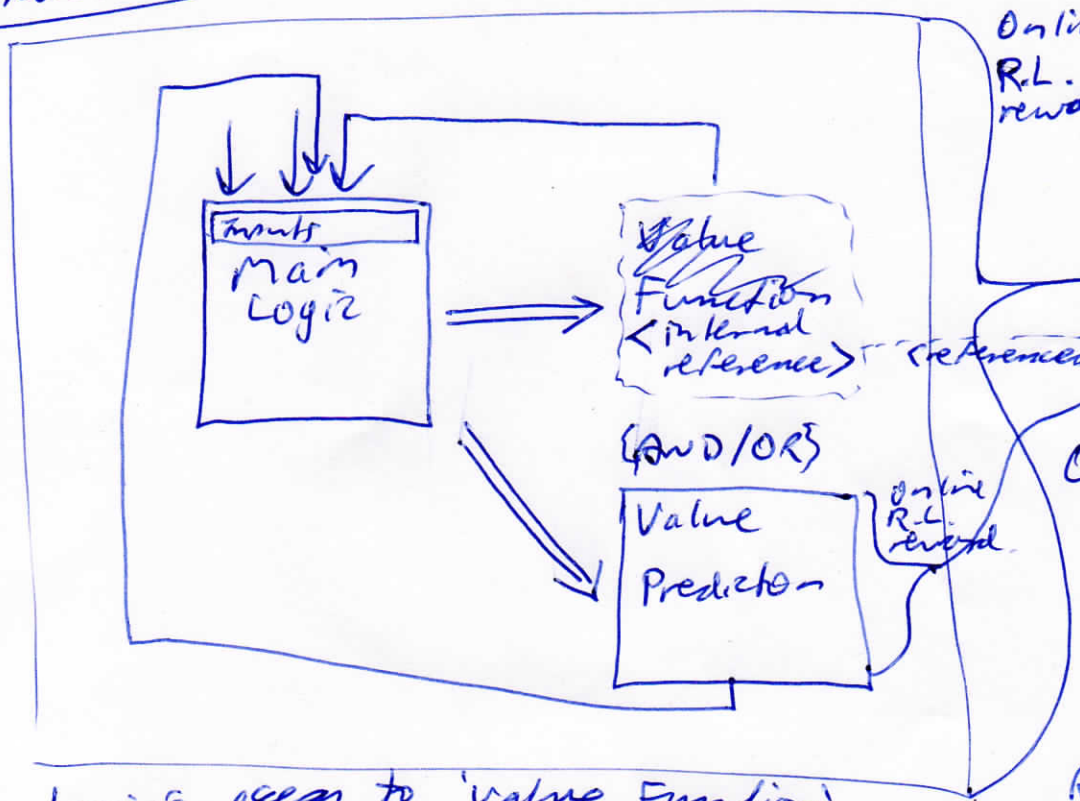
Internalised Reward Function

Value Function:

- N.N — initially trained simply to predict same value as reward from pre-training.
- Hard-coded — optional "evolved" hard-wired value.
  eg: that discourages too much physical or mental effort ← acts to improve efficiency.
  ↑ based on simple hard-wirable measures such as number of main loop iterations.

Usage architectures



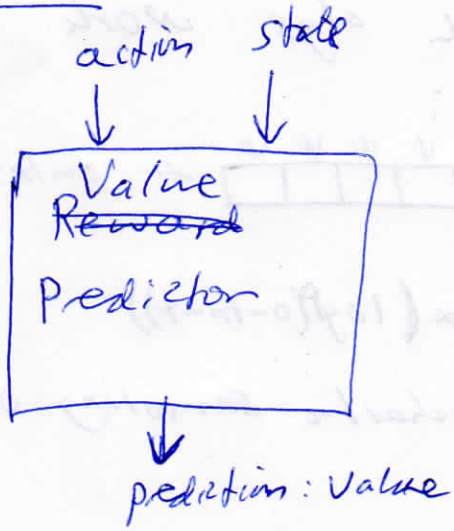
Q: discrete reward only at occasional "achievement" point?
OR continuous for every action?
A: just models whatever original pre-training reward was.

main logic's access to 'value Funtion' would give it a value after actions.
Whereas the 'value predictor' could be used to decide between possible actions before carrying them out. Also 'value predictor' attempts to predict Future reward.
'value Predictor' directly trained by RL from Value Funtion with its own loss Function.

2/ Q: What can provide continual training to Value Function? without that, there is no hope to learn new skills?

Goal: an agent that can intentionally take a course of actions that leads to it learning something from those actions.

Learnable Predictors



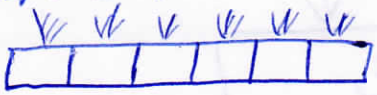
NN. representation

Predicts likelihood of achieving goal ~~given a particular action~~ if taking a particular action from state.

~~AFF~~ works off an AI's prior chosen goal, as ~~represented~~ in whatever way it uses.

4/ Continous vs. Categorized Actions

P.P.O. and other RL algs work sort of categorical actions:



← output layer

action: argmax(log(output))

And RL uses stochastic sampling of that probability distribution.

My AI needs to output a "model" rather than an action selection.

← output layer

action: output.

In order to apply stochastic sampling, treat each output node as an independent probability distribution (node value ⇒ mean?) and sample from each.

Why distribution sampling so important? Because it proves the source of "novel" attempts that may be better than the policy's current preference.