Tyler Streeter
tylerstreeter@gmail.com

This text describes the theoretical components that I consider important for intelligence. It follows the architecture of biological brains as closely as possible; however, it makes simplifications in places that do not affect the overall function. This model is being developed in a bootstrapping fashion. It starts with a set of assumptions and is continually modified to become closer to reality. Reinforcement learning seems to be the best central idea to this model; all other components help improve the process of maximizing rewards (including curiosity rewards).

Theoretical Components of Intelligence - Fundamental Concepts

State Representation

This component processes sensory inputs into a set of features that are more informative than the direct sensory inputs. This is accomplished through methods such as data simplification (transforming a problem from nonlinear to linear, e.g., quantization with RBFs) and data compression (dimensionality reduction and feature extraction, e.g., PCA).

Policy

A policy is simply a mapping from state to action. Given some situation, it chooses an action to perform.

Value Function

A value function is a mapping from state to value. "Value" is usually defined as the expected discounted future sum of rewards.

Reinforcement Learning

This is the process of learning the value function and policy. Both learn incrementally using temporal difference learning.

Planning (i.e. Learning and Using a Predictive Model of the Environment)

A predictive model of the environment enables the agent to speed up reinforcement learning. Instead of interacting directly with the environment (in order to improve the value function and policy), the agent can learn from imagined scenarios. This is much faster and safer than learning in the real world. Such a model can be trained to predict using the actual observations that occur.

A prediction model can be designed to: 1) map states and actions to next states, or 2) map states to next states. (Both methods have been tested in a discrete grid maze task and gave similar results; however, method 2 might only work in

such tasks where the policy converges to a deterministic action selector.) Method 2 would require separate state and action hierarchies (that are joined at the top, via the policy), and predictions would occur only in the state hierarchy. Alternatively, we could just always supply efferent copies of the current action into the state representation; i.e. the action is *part of* the state. (Biological evidence for this: efferent copies of motor signals are represented in the somatosensory cortex.)

Curiosity

Curiosity is an internal drive to learn (i.e. to improve the predictive model). It depends on the presence of a predictive model. Curiosity can be modeled by giving the agent a curiosity reward proportional to 1) the novelty/uncertainty of a state (i.e. the mean squared error of the model predictions), or 2) the reduction in novelty/uncertainty over time (i.e. it is rewarding to learn new things). A curiosity drive constantly exposes the agent to new situations and helps build its predictive model.

According to Oudeyer and Kaplan (see "Intelligent Adaptive Curiosity", or IAC), uncertainty reduction must be measured according to specific situations. In other words, curiosity rewards are proportional to uncertainty reduction over time in a specific state; in other words, curiosity rewards are local.

Uncertainty

Uncertainty is also known as "novelty" and "prediction errors." This is closely tied to predictive models. Uncertainty can be modeled as a measure of prediction errors, i.e. the mean squared error of predictions. It is possible to have a metapredictor that predicts other predictive models' prediction errors. Having an estimate of one's own uncertainty about states is important. This can be used to determine how long to plan; longer plans require more certainty about state transition predictions (there is no use planning if we're not sure the predicted events will occur). Uncertainty can also determine the *novelty* of a state, useful for a curiosity drive.

Theoretical Components of Intelligence - Advanced Concepts

Hierarchical Policies

Using the theory of "options," let's expand the definition of a policy: a policy (or "option") is a mapping from state to action that has an initial state which triggers the policy and a goal state which stops the policy. With this definition we can construct a hierarchy of policies; a policy gets triggered by its initial state and continues until it reaches its goal state. A high level policy proceeds by executing its child policies in the hierarchy. Thus, high level policies are temporally longer than low level ones. At the very bottom of the policy hierarchy is a set of inborn, hard-coded, atomic behaviors. This policy hierarchy helps the agent learn high-level policies faster; learned policies can be reused to achieve other similar goals.

Attention

"Attention" refers to the limited amount of information that can be processed consciously at once. (For example, in the brain most sensory input goes through the thalamus, which could act as a gating system for information processing.) Theoretically, a limited attention channel might correspond to the limited amount of state/policy data that can be used for planning at once - we can only plan at a specific part of the state/policy hierarchy at a time. Attention could be guided by curiosity using a winner-take-all scheme: the area of the state/policy hierarchy that generates the highest curiosity reward (i.e. novelty, or novelty reduction) attracts the most attention. Note that attention can be externally or internally guided; externally guided attention comes from novel sensory inputs, whereas internally guided attention comes from novel situations experienced through planning/imagining. Overall, attention would save computational resources in computational models and metabolic resources in biological brains.

Spatial and Temporal Abstraction

In general, it is more efficient to process only the most salient features of sensory inputs (i.e. the "edges" of spatial objects and temporal events).

Hierarchical Planning and Exploration

With hierarchical policies/options, we can plan at any level of the hierarchy. Planning at higher levels is more efficient as it skips over low-level details. Similarly, exploration at higher levels allow temporally longer exploratory actions.

Chunking

Well-learned motor sequences get "chunked" into distinct modules with well-defined beginning states and end states. These modules become "options."

Methods for Efficient State Representations

A state representation can be made more efficient through several methods:

- Data simplification (transforming from nonlinear to linear with quantization)
- Data compression (dimensionality reduction)
- Hierarchical organization
- Prediction error-based information flow

Data Simplification with RBF Quantization

To make complex problems easier to solve (i.e. by transforming nonlinear problems to linear problems), we can "quantize" continuous values by covering the input spaces with a discrete number of basis functions. This produces a "feature map" which maps data from the input space to a single area in "feature space."

Gaussian radial basis functions (RBFs) let us cast a pattern classification (or function approximation?) problem in a high-dimensional space nonlinearly, making

it more likely to be linearly separable than in a low-dimensional space (Cover's theorem, 1965).

A useful extension to RBF representations (and other quantization methods) is competitive learning/self-organizing maps. Using unsupervised learning rules, units "compete" in a winner-take-all scheme. The "winner" determines the classification of the given input. The winning unit (or neighborhood of units) usually move in the direction of the input data. This approximates "principal curves/surfaces," making it a nonlinear generalization of PCA. K-means clustering is a type of SOM that computes cluster centroids explicitly.

This learning method is local: only one output is active at once. Thus, this is good for classification tasks. This generates a representation that is easier to use for learning than non-local representations, but it does not generalize as well as non-local methods.

Example: A Kohonen self-organizing map (SOM) converts a data point (of arbitrary dimension) into a distinct localized region of activity in a 1D or 2D map, with each unit connected only to a few neighbors. It chooses the centers of units by moving the "winners" closer to the input data. This distributes resources (feature detectors) intelligently across the input space.

Useful Modifications:
1. It is possible to adjust the units' neighborhood widths (e.g., decay over time, or scale them based on unit density).
2. Rather than starting with a full map of random units, the units can be allocated on demand to save resources. This avoids wasting units in low density regions of the input space.

There are various ways to quantize multidimensional data. The following are four different methods:
1. Direct n-dimensional quantization, covering the whole n-dimensional input space with a grid of equally-spaced units
2. Optimized n-dimensional quantization (same as the direct method, except that we use something like a SOM to use resources more efficiently)
3. Direct 1-dimensional quantization, where each dimension is quantized separately
4. Optimized 1-dimensional quantization (same as direct 1-dimensional, but use a SOM)
(The n-dimensional methods require 2^n units, but the 1-dimensional methods only need qn units, where q is the number of units per dimension.)

Data Compresstion with PCA

Using principal components analysis (PCA), we can reduce the dimensionality of a data set without losing much information. This also enables feature extraction: resulting data set represents more informative "features" that have been statistically decorrelated. Note that the result is linearly related to the original data.

This can be performed by neural networks with unsupervised Hebbian learning rules. This is a non-local learning method; many outputs will be active at once, each representing the degree to which the input data varies along the principal components (i.e. which "features" it contains) using the inner product between the input vector and the principal component vectors. Note that non-local learning rules are good at generalization.

Incremental PCA Algorithms (do not require a covariance matrix):
1. Simple Hebb rule w/ normalization: w = w + n(yx), then w = w / ||w||^2; computes the dominant eigenvector
2. Oja's basic adaptive rule (Oja, 1982): using a single output neuron, dW = ny(X - yW)dt; 1st term = pos feedback, 2nd term = neg feedback (forgetting/leakage factor); extracts first principal component
7. Stochastic Gradient Ascent (SGA) (Oja and Karhunen, 1985): slow, possibly unstable
3. Generalized Hebbian Algorithm (GHA) (Sanger, 1989): expansion of Oja's basic rule; components are sequentially computed and subtracted out; sequential; does not use inhibitory connections
4. Backward inhibition (Hrycej, 1989): inhibit inputs w/ outputs to filter out each successive component; sequential
5. Delta-rule self-organization (Hrycej, 1990-1992): dW = nY(X - YW)dt; parallel, less stable, slower convergence
6. Adaptive Principal Components Extraction (APEX) (Kung and Diamantaras, 1990): uses feedback connections
8. Candid Covariance-Free Incremental PCA (CCIPCA) (Weng, Zhang, Hwang, 2003): excellent convergence, fast execution

Simple PCA works best when the problem is linear. However, it might be beneficial to apply PCA on a set of features that are nonlinearly related to the actual input data (e.g., using RBFs), thus generating the principal components in feature space (instead of input space). Kernel PCA allow this by performing regular PCA on the feature vector (assuming an inner product Mercer kernel, e.g., RBF, see Table 6.1 on p. 333 of Haykin, Neural Networks). "Kernel PCA is linear in feature space but nonlinear in the input space" (p. 434-435, Haykin, Neural Networks).

Hierarchical Organization

Instead of having all sensory inputs converge onto a single feature array, we use a hierarchy of feature arrays. The higher levels combine low level elements to represent more abstract concepts. At the very lowest level are the inborn, hard-coded sensors that gather information directly from the environment. In general, each of these arrays must be connected to every array in the level below and the level above. However, this is probably not necessary; for some sensory input arrays (say, vision), the lower parts of the hierarchy will not need to connect to the lowest levels of, say, the somatosensory input array. Only at the higher levels should the various major sensory input hierarchies be combined into a single hierarchy. Still, how do we know which sensory input hierarchies to combine at which levels? We could start with all the higher levels fully connected, then gradually remove connections over time that never get used. Each element in the hierarchy could learn a predictive model of its child elements: an element sends predictions downwards and prediction errors upwards.

By having each module contain its own predictive model, we would automatically enable localized curiosity (which is necessary, according to Oudeyer and Kaplan - see IAC papers).

Information traveling up the hierarchy converges into higher-level invariant representations; info traveling down diverges into lower-level details. The activation of units at higher levels changes slowly; low-level units change rapidly. In other words, high-level units represent the "names" of low-level

sequences; they remain active as long as the sequence lasts. Lower units represent shorter sequences that can be reused by multiple higher units.

Fortunately, having a hierarchy of units that learn sequences automatically enables "chunking" of well-learned sequences (i.e. options). When planning over high level options, it is possible to skip over the low-level details straight to the predicted final state. This can be accomplished by simply not iterating through the child elements sequentially.

A multistage hierarchical structure would help reduce computational complexity by reducing the processing to a number of small suboperations. For example, the following structure combines 4 dimensions:

```
input0 ---> [          ]
input1 ---> [ Abstract ]
input2 ---> [ Features ]
input3 ---> [          ]
```

...and the following hierarchical structure combines 2 dimensions at each node:

```
input0 ---> [ Abstract ]
input1 ---> [ Features ] ---> [                    ]
                              [ Abstract Features ]
input2 ---> [ Abstract ] ---> [                    ]
input3 ---> [ Features ]
```

Using RBFs with 10 RBFs per dimension, the first method requires 10^4 = 10,000 RBFs. The second method requires 3 * 10^2 = 300 RBFs.

How do multiple inputs converge? Two methods:
1. Multidimensional RBFs - probably not biologically realistic (cells are not tuned to values in several dimensions at once)
2. PCA

Prediction Error-Based Information Flow

In a hierarchical structure, we can reduce the amount of information flowing upwards by only encoding prediction errors. This assumes that the elements of the hierarchy develop predictive models of their inputs. Each level receives actual data from below and predicted data from above. Only prediction errors are passed up to higher levels (i.e. if you don't understand what you're seeing, ask someone higher up). Each level recognizes spatial patterns and temporal patterns (sequences). This prediction-based structure can also act as a predictive model for planning.

Note that prediction errors are constantly occurring, even in nearly optimal predictors, assuming the environment is nondeterministic. Another way to think about prediction errors is that they are the "edges" of objects and temporal events; the stuff between the edges is predictable.

(The following ideas on predictive representations in the brain come from Jeff Hawkins's book, On Intelligence.)

Each unit can classify input data (recognition model) and predict future data (generative model).

Lower units send two signals to higher units:

1. A simple activation signal, signifying when the unit is active in a sequence
2. Prediction errors (i.e. "I can't handle this data, so I'll pass it upwards.")

Each unit classifies its lower-level inputs using a discrete set of possibilities. A unit recognizes a sequence of incoming patterns, and it can predict the next input pattern and tell lower units what to expect.

Units can send their output (prediction errors?) back to themselves, enabling "imagining" (planning).

Information flowing downwards (i.e. predictions) can activate many units below it. Each unit becomes active only when the right combination of inputs from below are present. Lateral inhibition enables a winner-take-all scheme during classification; however, this only applies locally within a region, so several units can still be active at once.

Layer 1 of each unit/column receives the "previous state" from the thalamus (p. 144). This "state" includes sensory and motor information (p. 157). Half of the input to layer 1 is from neighboring units (p. 149). The other half comes from higher regions, representing the "name" of a sequence. Thus, a unit receives the name of the current sequence and the position within the sequence, allowing lower units to be reused among many higher-level sequences. Top-down predictions (e.g. a 5th interval in music) and bottom-up data (e.g. current note is D) combine to form a specific prediction: the next note is A.