A simple approach to perceptual and attentional learning using LIDA

Marcel Ferreira Batista

Introduction

The purpose of this work was to consolidate and leverage the knowledge acquired through the performance of the LIDA architecture tutorials and study of the LIDA model.
The tutorials provided by the LIDA website teach the student how to perform basic manipulations of the provided templates and configuration files so the student can get some understanding of how the modules and elements are set up and connected. Even though they are enough for the basic understanding of the architecture, the tutorials do not address important concepts such as generic feature detection, triggering of competition for consciousness, dynamic creation of elements and especially the learning process used by the model.

In the following pages, the development of more generic feature detectors and rudimentary perceptual and attentional learning are described. It is not the ambition of this work to suggest complex algorithms or theories for perceiving or learning. Instead, the goal was to use the framework with the least possible changes to the model so to understand more profoundly how elements work together from perceptual up to attentional level and how to use the conscious broadcasts as the initial step of learning.
Some understanding of the LIDA model theory and framework structure are recommended before the reading of this work.

Generic perception

To apply the concepts and test the implemented features, the basic agent template provided by the LIDA tutorial exercises is used. This agent is a simple ‘button presser’. The environment consists of an image layer where either a red square or a blue circle are painted and two buttons. The agent’s goal is to press one of the buttons if there is a red square or the other if there is a blue circle.

For this task to be accomplished, the first step is to perceive features on the environment. In this case, the agent is seeking for a color feature and a shape feature. The tutorial template uses very simple feature detectors for this task. They are set up in the agent’s configuration file in such a way that for each feature the agent needs to recognize, one feature detector is added with different parameters (one color feature detector for blue, one for red, one shape feature detector for square and one for circle). Once one of the detectors successfully detects the feature it seeks, it excites the related node of the Perceptual Associative Memory and it gets a chance of being added to the agent’s Current Situational Model. This is clearly a very simplistic approach to the problem (due the pedagogical purpose of the tutorials), so one of the changes made in this work was to generalize the color feature detector.

The first problem related to changing this approach is that the nodes of the features to be detected were previously added to the PAM using the agent data configuration file. If we create generic feature detectors, what PAM nodes are they to excite once detection occurs? The simple answer for this would be simply to add a new node for that feature to the PAM if one does not exist. However, this is a form of learning, and the way of it does not comply with the model’s principle that learning starts with a conscious broadcast from Global Workspace. To comply with the model, this newly detected feature must be brought up to conscience before being learnt.

Even though a feature should not be learnt this way, nothing prevents the addition of this feature to the agent’s percept. We can hope that, once the new feature is added to the percept with high excitation, some Attention Codelet will notice it and bring it up to conscience where it can be broadcasted and eventually learnt.

Generic color detection algorithm

```
for each different color of the image
if the PAM has a node for it, excite this node
otherwise, add a new node to the agent’s percept with high activation
```

Generic attention

Once the feature node is added to the PAM with high activation, it has chances of being moved to the Current Situational Model of the agent’s Workspace. This is where Attention Codelets look for nodes that match their subject of attention and leverage them to the Global Workspace in the form of coalitions. In the tutorial template, there were specific pre-built Attention Codelets for red square and blue circle. Just like the case of the non-existent node for the newly perceived features, there can be no pre-existent Attention Codelet for every new perceived node. The easy way of addressing this problem would be the creation of a Basic Attention Codelet that would have the newly perceived feature node as its focus of attention immediately after the new feature is detected. However, we would face the same problems as before: this is a kind of learning that does not comply with LIDA model. We need to hope this node will be brought up to conscience so a new Attention Codelet can be learnt once the conscious broadcast is triggered. Since there is no other way of adding a Coalition to the Workspace, a new kind of Attention Codelet is necessary.

The task of this new codelet would be to find perceived features that are not the subject of any other Attention Codelets and give them a chance to be brought up to conscience. This special kind of codelet, the 'New Feature Attention Codelet', is not bound to any node at all: it just looks for the most active perceived node that is nobody's subject of attention and tries to send it to the Global Workspace in the form of a Coalition. To be able to find out if a node is the subject of attention of any other Attention Codelet, this special Codelet has access to the Attention Module that, in this experiment's implementation, has a record of all existent Attention Codelets.

Perceptual Learning

If everything is working as expected, some features that are unrelated to any PAM node are being broadcasted through the system: it is a chance for learning. According to the LIDA model, perceptual learning is related (amongst other things) to the excitement of the base level activation of existent PAM nodes or adding new nodes to the PAM. For this experiment, only addition and excitation are handled.

In this learning implementation, the PAM will look for the contents of the received broadcast amongst its nodes. If a node is found, it receives some base-level excitation. If not, a new node is added to it. This is a simplistic approach for learning but is enough for the purpose of this work. It should be noted, however, that this can be much more elaborated with some simple change of parameters on the creation of the new PAM node.

Keep in mind that the PAM is a memory, and as such it is expected that its elements fade away with time and lack of stimulus. In the LIDA architecture, every few ticks will make the existent PAM nodes fade away. This is represented by the decay of the base-level activation of the node following some decay strategy. If this activation reaches a minimum threshold, the node will be permanently removed from the memory. However, the default decay rate of this property is significantly low and the effect barely noticeable so it is unlikely that the newly created node will be ever removed from the PAM.

By simply changing these parameters (initial base-level activation, decay rate and removal threshold) on the moment the new node is created, more interesting results can be achieved. Another possible change to be made would be to change the base-level decay strategy of the existent PAM node after it reaches some minimum threshold. This would represent some knowledge that was sedimented to some degree in the agent's memory and would take longer to fade away.

Attentional learning

The approach given to the attentional learning is simple yet meaningful. The Attention Module of the system is one of the Broadcast Listeners of the Global Workspace. Once again, the learning will take place once a chosen coalition is broadcasted to this module. Besides the Current Situational Model, the Workspace has a sub-module called Broadcast Queue. The role of this module is to keep a record of the last *n* Broadcasts received by the Workspace. This first version of the learning consists simply in creating new Attention Codelets as needed. It would be interesting to use a reinforcement approach for the base-level activation of the Attention Codelet that is responsible for creating the broadcasted coalition. However, in order to get that information more of the structure of the model would have to be changed. Another possible solution would be to reinforce all Attention Codelets that pay attention to the nodes of the broadcasted Coalition.

In short, to decide if it is time to create a new Attention Codelet, the Attention Module check is if there are any Attention Codelet already seeking the contents of the broadcasted Coalition. If not, the frequency with which this coalition has being broadcasted is computed through the Broadcast Queue. If this is higher than a pre-defined threshold, a new Attention Codelet will be instantiated and added to the Attention Module (and then executed).

Results

By monitoring the agent's GUI while using these techniques the following characteristics:

1) At the start, no color feature nodes exist in the PAM graph
2) The first feature detections will add nodes that do not exist in the PAM graph to the perceptual buffer of the workspace
3) While the agent is running, new nodes will be added to the PAM graph when new colors are drawn on the image layer
4) After a new node is added, its base level activation will gradually increase but it won't prevent other features of being broadcasted and learnt
5) At the beginning, the special New Feature Attention Codelet will create the coalitions, but once enough broadcasts are made for each newly detected feature, the Generic Attention Codelets created will be responsible for the addition.
6) The Conscious Broadcasts are triggered for either no broadcasts occurring or no coalition arriving events. The first case happens if no Broadcast was triggered for a certain time, and the second when the first condition happens and a certain amount of time goes by without a *new* coalition being added to the Global Workspace. These behaviors should be expected in the start of the experiment because the Attention Modules have a low base-activation level. If an Attention Codelet reinforcement technique was applied on the learning phase of the Attention Module, this base-level activation would gradually increase and the added coalitions would have enough total activation to trigger a regular Broadcast.

Key pieces of code implemented

Many additions were made to the agent configuration files and new classes were created, most of them extending existing implementation provided by the template. Because some of the provided classes had important private attributes (preventing the access to sub-classes) it was made necessary to make complete code copies of some classes to create new classes instead of using heritage.

In the following pages, key points of the altered or created classes are copied while support methods for initialization or simple calculations were not. The complete source code can be found here.

```
public class GenericColorFeatureDetector extends BasicDetectionAlgorithm{

    ...
    public double detect() {
        int[] layer = (int[]) sensoryMemory.getSensoryContent("visual",smParams);

        HashSet<Integer> handledColors = new HashSet<Integer>();

        for(int i=0;i<layer.length;i++){
            int color = layer[i];

            if(!handledColors.contains(color)){
                String label = Integer.toHexString(color).toUpperCase();
                PamNodeImpl node = (PamNodeImpl) pam.getNode(label);

                if(node == null){
                    node = (PamNodeImpl) ElementFactory.getInstance().getNode(
                            "PamNodeImpl",
                            ElementFactory.getInstance().getDefaultDecayType(),
                            ElementFactory.getInstance().getDefaultExciteType(),
                            label,
                            1.0,
                            0.01);

                    pam.addNodeToPercept(node);
                }
                else{
                    pam.receiveExcitation(node, 0.1);
                }

                handledColors.add(color);
            }

        }
       return 0.0;
    }
}
```

```java
public class NewFeatureAttentionCodelet extends AttentionCodeletImpl{

    ...
    @Override
    public NodeStructure retrieveWorkspaceContent(WorkspaceBuffer buffer) {
        NodeStructure model = (NodeStructure) buffer.getBufferContent(null);
        Linkable chosenLinkable = null;

        double winnerActivation = 0.0;

        for (Linkable ln : model.getLinkables()) {
            double linkableActivation = ln.getActivation();

            if(linkableActivation > winnerActivation &&
                    !attentionModule.hasCodeletSeekingFor(ln)){
                chosenLinkable = ln;
                winnerActivation = linkableActivation;
            }
        }

        return linkableToNodeStructure(chosenLinkable);

    }
    ...
    @Override
    public boolean bufferContainsSoughtContent(WorkspaceBuffer buffer) {
        NodeStructure model = (NodeStructure) buffer.getBufferContent(null);
        for (Linkable ln : model.getLinkables()) {
            if(!attentionModule.hasCodeletSeekingFor(ln)){
                return true;
            }
        }
        return false;
    }
}
```

```java
public class GenericAttentionCodelet extends AttentionCodeletImpl {

    @Override
    public void init() {
        super.init();
        HashSet<Node> nodes = (HashSet<Node>) getParam("nodes", null);
        if (nodes != null) {
            for(Node node:nodes){
                soughtContent.addDefaultNode(node);
            }
        }
    }

    @Override
    public boolean bufferContainsSoughtContent(WorkspaceBuffer buffer) {
        NodeStructure model = (NodeStructure) buffer.getBufferContent(null);
        for (Linkable ln : soughtContent.getLinkables()) {
            if (!model.containsLinkable(ln)) {
                return false;
            }
        }
        return true;
    }

    @Override
    public NodeStructure retrieveWorkspaceContent(WorkspaceBuffer buffer) {
        NodeStructure ns = ((NodeStructure) buffer.getBufferContent(null));
        NodeStructure result = new NodeStructureImpl();
        for (Node n : soughtContent.getNodes()) {
            if (ns.containsNode(n)) {
                result.addDefaultNode(ns.getNode(n.getId()));
            }
        }
        for (Link l : soughtContent.getLinks()) {
            if (ns.containsLink(l)) {
                result.addDefaultLink(ns.getLink(l.getExtendedId()));
            }
        }
        return result;
    }
}
```

```
public class AttentionCodeletModuleImpl extends AttentionCodeletModule{
    ...
    @Override
    public void addCodelet(Codelet codelet) {
        this.addedCodelets.add(codelet);
    }

    public boolean hasCodeletSeekingFor(NodeStructure ns){
        for(Codelet codelet: addedCodelets){
            boolean found = true;
            for(Linkable linkable : ns.getLinkables()){
                if(!codelet.getSoughtContent().containsLinkable(linkable)){
                    found = false;
                    break;
                }
            }
            if(found) return true;
        }
        return false;
    }

    public boolean hasCodeletSeekingFor(Linkable linkable){
        NodeStructureImpl ns = new NodeStructureImpl();

        if(linkable instanceof Node) ns.addDefaultNode((Node)linkable);
        else if(linkable instanceof Link) ns.addDefaultLink((Link) linkable);
        else System.out.println("ERROR INVALID LINKABLE " + linkable );

        return hasCodeletSeekingFor(ns);
    }

    @Override
    public void learn(BroadcastContent content) {
        NodeStructure ns = (NodeStructure) content;
        if(this.isValidNewFeature(ns)){
            GenericAttentionCodelet codelet = createNodeAttentionCodelet(ns);
            this.addCodelet(codelet);
            this.taskSpawner.addTask(codelet);
        }
    }
    private GenericAttentionCodelet createNodeAttentionCodelet(NodeStructure ns){

        HashMap<String, Object> params = new HashMap<String, Object>();
        params.put("nodes", new HashSet<Node>(ns.getNodes()));
        params.put("initialActivation", 0.5);
        params.put("refractoryPeriod", 50);

        HashMap<ModuleName, FrameworkModule> modules = new HashMap<ModuleName, FrameworkModule>();
        modules.put(ModuleName.GlobalWorkspace, globalWorkspace);
        modules.put(ModuleName.CurrentSituationalModel, this.currentSituationalModel);

        GenericAttentionCodelet attentionCodelet = (GenericAttentionCodelet)
ElementFactory.getInstance().getFrameworkTask("GenericAttentionCodelet", params, modules);

        return attentionCodelet;
    }

    protected boolean isValidNewFeature(NodeStructure ns){

        float frequency = broadcastQueue.getBroadcastFrequencyForStructure(ns);
        if(!hasCodeletSeekingFor(ns) && frequency >= minimunBroadcastFrequency) return true;
        else return false;
    }
}
```

```
public class ImprovedPerceptualAssociativeMemoryImpl extends PerceptualAssociativeMemoryImpl{
    ...
    @Override
    public void learn(BroadcastContent bc) {

        NodeStructure ns = (NodeStructure) bc;
        Collection<Node> nodes = ns.getNodes();
        for (Node n : nodes) {
            PamNode existentNode = (PamNode) this.getNode(n.getExtendedId());
            if(existentNode != null){
                existentNode.reinforceBaseLevelActivation(nodeBaseLevelReinforcement);
            }
            else{
                PamNodeImpl newNode = (PamNodeImpl) ElementFactory.getInstance().getNode(
                    "PamNodeImpl",
                    ElementFactory.getInstance().getDefaultDecayType(),
                    ElementFactory.getInstance().getDefaultExciteType(),
                    n.getLabel(),
                    newNodeInitialActivation,
                    newNodeRemovalTrheshold);

                newNode.setBaseLevelDecayStrategy(ElementFactory.getInstance().getDecayStrategy(
                        "slowDecay"));
                newNode.setBaseLevelExciteStrategy(ElementFactory.getInstance().getExciteStrategy(
                        "defaultExcite"));

                addDefaultNode(newNode);
            }
        }
    }
}
```

```java
public class ImprovedBroadcastQueueImpl extends FrameworkModuleImpl implements
        WorkspaceBuffer, BroadcastListener {


    protected float getBroadcastFrequencyForLinkable(Linkable ln){

        NodeStructure ns = linkableToNodeStructure(ln);
        return getBroadcastFrequencyForStructure(ns);
    }

    protected float getBroadcastFrequencyForStructure(NodeStructure ns){

        synchronized(broadcastQueue){
            int occurrences = 0;

            for(NodeStructure broadcastedNs : broadcastQueue){

                if(compareNodeStructures(broadcastedNs, ns)) {
                    occurrences++;
                }
            }
            return (float) occurrences/this.broadcastQueueCapacity;
        }
    }

    protected boolean compareNodeStructures(NodeStructure a, NodeStructure b){

        for(Linkable ln : a.getLinkables()){
            if(!b.containsLinkable(ln)) return false;
        }
        return true;
    }
}
```