Nümila model description

Fred Callaway

November 22, 2015

Nümila is a model of language acquisition, comprehension, and production. The core of its knowledge is a set of hierarchical chunks which are composed of base tokens (e.g. words or syllables) and, other chunks. The model discovers these chunks based on transitional probabilities: two words that occur together frequently will form a chunk. These discovered chunks can then form their own transitional probabilities, allowing them to become part of a hierarchical chunk, e.g. `[the [mean cat]]`.

Nümila emphasizes psychologically plausibility by adhering to the strict processing constraints imposed by the Now-or-Never bottleneck (Christiansen and Chater 2015). The working memory of the model can hold only 4 nodes at a time. However, because these nodes can be chunks that span several base tokens, Nümila can still hold an entire utterance in memory at once, provided that it has enough experience to create chunks out of the incoming stream. Aditionally, Nümila relies on a small set of operations that are associational in nature, increasing the plausibility of the model.

We follow ADIOS (Solan et al. 2005) and U-MILA (Kolodny, Lotem, and Edelman 2015) by representing linguistic knowledge with a graph. Tokens and chunks are nodes; transitional probabilities are edges. For simplicity, we restrict chunks to contain exactly two nodes, and we form chunks based entirely on local dependencies (i.e. chunks must be contiguous). However, these are simplifying assumptions rather than theoretical claims. We suspect that relaxing these contraints would improve the models performance. One advantage of this constraint is that it makes the model's chunks equivalent to binary branching trees, allowing the model to connect to traditional linguistic theories. By combining highly structured representations with a psychologically plausible learning model, Nümila aims to unite work in computational linguistics with work in psycholinguistics.

Learning and comprehension

Comprehension and learning occur as part of one process that we call parsing. In this way, the model exemplifies a usage-based approach to language acquisition. The training process is simply parsing a list of utterances in a corpus. Parsing involves passing a short memory window over an utterance. At each step of parsing, the model updates transitional probabilites between the nodes in memory (learning) and attempts to bind two nodes into a chunk (comprehension[1]).

The output of a parse is a list of binary trees (chunks) that span the utterance. An utterance of entirely novel words would result in a list of each word as a singleton tree, whereas a well

[1]Note that we use the term comprehension very loosely, gesturing towards the idea that language comprehension is fundamentally a task of composing the meaning of words (Hagoort 2005). This is an oversimplification, but we are optimistic about the possibility of incorporating more nuanced semantic representations into the model.

known utterance would result in a list containing one tree that spans the whole utterance. For example: `el | modelo | no | aprendió | español` vs. `[[the model] [learned english]]`. In between these two extremes, we would find `[the model] | [kind of] | [learned english]`. In this example, the model picked up on some simple chunks, but was unable to combine them hierarchically.

The following is the main loop in `Parse`. Details for each step are given below.

```
for token in utterance:
    self.graph.decay()
    self.shift(token)
    self.update_weights()
    if len(self.memory) == self.params['MEMORY_SIZE']:
        # Only attempt to chunk after filling memory.
        self.try_to_chunk()  # always decreases number of nodes in memory by 1
```

1. **Decay:** All edge weights in the graph decay. In effect, the distributions defined by edges become closer to uniform.

2. **Shift:** A new token is added to memory. It is assumed that the current number of nodes in memory is less than `MEMORY_SIZE`.

3. **Update weights:** For each pair of adjacent nodes in memory, n_1 n_2, (1) increase the weight of the forward edge from n_1 to n_2, and (2) increase the weight of the backward edge from n_2 to n_1. Given this learning procedure, normalizing a node's edges will result in forward and backward transitional probabilities.

4. **Chunk:** The *chunkability* of each pair of adjacent nodes is the geometric mean of the normalized forward edge weight from n_1 to n_2 and the normalized backward edge weight from n_2 to n_1. Recall that these edges represent forward and backward transitional probabilities, thus this metric is quantifying the degree to which each node predicts the other one. This is very similar to Barlow's principle of suspicious coincidence (Barlow 1990; Kolodny, Lotem, and Edelman 2015). If the pair with the highest chunkability forms a chunk, the two constituent nodes are removed from memory and replaced with the chunk. Othewrise, the oldest node in memory is removed from memory. Thus, chunking always decreases the number of nodes in memory by 1, making room for a new token to be shifted. The model only attempts to chunk when either (1) it has a full memory, or (2) there are no tokens left in the incoming utterance (code for processing the tail not shown).

Production

We simulate production using a bag-of-words task (Chang, Lieven, and Tomasello 2008; McCauley and Christiansen 2014). The model receives an unordered bag of words and must construct a parse out of these word. At each step of the iterative algorithm, the model picks the two "chunkiest" nodes in the bag, i.e. the pair that has the highest chunkability as defined above. These two nodes are removed and replaced with their chunk, reducing the size of the bag by 1. The process continues until there is only one node in the bag, which is a binary tree spanning the full utterance.

```
def speak(self, words, verbose=False):
    nodes = [self[w] for w in words]
```

```
# combine the two chunkiest nodes into a chunk until only one node left
while len(nodes) > 1:
    best_pair = max(itertools.permutations(nodes, 2),
                    key=lambda n1, n2: self.chunkability(n1, n2))
    n1, n2 = best_pair
    nodes.remove(n1)
    nodes.remove(n2)
    # Using `force=True` returns the chunk [n1 n2], regardless of
    # whether it is a node in the graph or not.
    chunk = self.get_chunk(n1, n2, force=True)
    nodes.append(chunk)

return nodes[0]
```

Holographic graph representation

Up to this point, the representations of the model have been described in terms of a directed multigraph, with words and chunks as nodes, and forward and backward transitional probabilities as edges. However, in practice, Nümila does not explicitly represent a graph. Rather, Nümila approximately represents a graph in a distributed way using holographic representations. Holographic representations are

The traditional N x N adjacency matrix is replaced by an N x D matrix where D is the dimensionality of our sparse vectors. Each row represents the outgoing edges of a node as the summation of the id-vector for every node it is connected to. We can represent weighted edges by weighting this sum.

To represent multiple edge types, we use permutations. Each edge type is assigned a random permutation vector, an *edge-vector*. To update a specific edge from node n_1 to n_2, we add the id-vector of n_2 permuted by the corresponding edge-vector to the row of n_1. Thus we can define the row for a node n_0 as

$$\text{row-vector}(n_0) = \sum_{e \in E} \sum_{n \in N} \Big(P_e(\text{id-vector}(n)) \cdot \text{edge-count}(e, n_0, n) \Big)$$

where E is the set of edge types, N is the set of nodes, P_e is a permutation vector for edge e, id-vector(n) is the id-vector of node n, and edge-count(e, n_1, n_2) is an integer weight on the edge of type e connecting n_1 to n_2[2]. For example, in Nümila, edge-count(F, the, dog) would indicate the number of times "dog" followed "the" in the training corpus. Note that we use F and B to refer to forward and backward edges.

For example, here is part 1 of the **update weights** step in graphical terms and vector terms:
(1) increase the weight of the forward edge from n_1 to n_2
(1) add $P_F(\text{id-vector}(n))$ to the row vector of n_1.

We use cosine similarity to approximate normalized edge weights, (approximate and normalized because edges will interfere and compete with each other). For an edge of type e connecting n_1 to n_2, we define edge-weight as

$$\text{edge-weight}(e, n_0, n) = \cos\big(\text{row-vector}(n_1), P_e(\text{id-vector}(n_2))\big)$$

[2] We assume all possible edges exist with default 0 weights.

Intuitively, this value will be higher if the second node's permuted id-vector is a large part of the sum that defines the first node's row-vector. In Nümila, we use this measure when calculating chunkability, which we can now formally define:

$$\text{chunkability}(n_1, n_2) = \sqrt{\cos\left(\text{row-vector}(n_1), P_F(\text{id-vector}(n_2))\right) \cdot \cos\left(\text{row-vector}(n_2), P_B(\text{id-vector}(n_1))\right)}$$

References

Barlow, Horace. 1990. "Conditions for Versatile Learning, Helmholtz's Unconscious Inference, and the Task of Perception." *Vision Research* 30 (11). Elsevier: 1561–71.

Chang, Franklin, Elena Lieven, and Michael Tomasello. 2008. "Automatic Evaluation of Syntactic Learners in Typologically-Different Languages." *Cognitive Systems Research* 9 (3). Elsevier: 198–213.

Christiansen, Morten H, and Nick Chater. 2015. "The Now-or-Never Bottleneck: A Fundamental Constraint on Language." *Behavioral and Brain Sciences*. Cambridge Univ Press, 1–52.

Hagoort, Peter. 2005. "On Broca, Brain, and Binding: A New Framework." *Trends in Cognitive Sciences* 9 (9). Elsevier: 416–23.

Kolodny, Oren, Arnon Lotem, and Shimon Edelman. 2015. "Learning a Generative Probabilistic Grammar of Experience: A Process-Level Model of Language Acquisition." *Cognitive Science* 39 (2). Wiley Online Library: 227–67.

McCauley, Stewart M, and Morten H Christiansen. 2014. "Acquiring Formulaic Language: A Computational Model." *The Mental Lexicon* 9 (3). John Benjamins Publishing Company: 419–36.

Solan, Zach, David Horn, Eytan Ruppin, and Shimon Edelman. 2005. "Unsupervised Learning of Natural Languages." *Proceedings of the National Academy of Sciences of the United States of America* 102 (33). National Acad Sciences: 11629–34.