# Triadic Memory - Semantic Triples and Analogies

Peter Overmann, 01 Aug 2022

Triadic Memory is an associative memory that stores triples of sparse distributed representations (SDRs), our brain's data structure.

It is naturally suited for storing semantic triples of the form {subject, predicate, object}.

Any part of a triple can be queried by giving the two other parts. For example, {subject,predicate,_} will return object.

As an associative memory, Triadic Memory operates simultaneously on parallel information: After storing {A,B,C1} and {A,B,C2}, a query for the third part {A,B,_} will return an SDR which is a superposition (or bundle, or bit-wise OR) of C1 and C2. This superposition can be used as input for further queries.

The computational cost of a query is independent of the number of stored items.

Analogy operators can be expressed as compound queries that effectively traverse the semantic network.

The examples shown in this notebook are adapted from Kanerva's paper *What We Mean When We Say "What's the Dollar of Mexico?"*
https://redwood.berkeley.edu/wp-content/uploads/2020/05/kanerva2010what.pdf

## Triadic Memory Algorithm

```
TriadicMemory[f_Symbol, {n_Integer, p_Integer}] := Module[ {W, tup},

   (* random sparse vector with dimension n and sparse population p *)
   f[] := SparseArray[ RandomSample[ Range[n], p] → Table[1, p], {n}];

   (* initialize memory *)
   W = Table[ 0, n, n, n];

   (* memory addresses activated by input vectors *)
   tup[x__SparseArray] := Tuples[ Flatten[ #["NonzeroPositions"] ] & /@ {x}];

   (* binarize a vector, using sparsity target p *)
   f[0] = SparseArray[{0}, {n}];
   f[ x_] :=
    Module[ {t = Max[1, RankedMax[x, p]]}, SparseArray[Boole[# ≥ t] & /@ x]];

   (* store {x,y,z} *)
   f[x_SparseArray, y_SparseArray, z_SparseArray] :=
    (++W[[##]] & @@@ tup[x, y, z];);

   (* recall x, y, or z *)
   f[Verbatim[_], y_SparseArray, z_SparseArray] :=
    f[ Plus @@ ( W[[All, #1, #2]] & @@@ tup[y, z])];
   f[x_SparseArray, Verbatim[_], z_SparseArray] :=
    f[ Plus @@ ( W[[#1, All, #2]] & @@@ tup[x, z])];
   f[x_SparseArray, y_SparseArray, Verbatim[_]] :=
    f[ Plus @@ ( W[[#1, #2, All]] & @@@ tup[x, y])];

  ];
```

## Encoding Wrapper for Triadic Memory

Same as TriadicMemory, but takes any Mathematica expression as input, encoding different expressions to different random SDRs.

```
TriadicMemoryEncoding[f_Symbol, {n_Integer, p_Integer}] :=
  Module[ {T, encode, decode, pos},

   TriadicMemory[T, {n, p}];

   pos[x_SparseArray] := Sort[Flatten[x["NonzeroPositions"]]];

   encode[x_SparseArray] := x;
   encode[Verbatim[_]] = _;
   encode[Null] = SparseArray[{0}, {n}];
   encode[s_] := encode[s] = Module[ {r = T[]}, decode[pos[r]] = s; r];

   decode[SparseArray[{0}, {n}]] = Null;
   decode[x_SparseArray] := Module[ {r}, r = decode[pos[x]];
     If[Head[r] === decode, x, r]];

   decode[Null] = Null;

   f[] = T[]; (* random generator *)

   f[x_, y_, z_] := decode[T[encode[x], encode[y], encode[z]]];

  ];
```

## Configuration

```
n = 500; p = 5;
TriadicMemoryEncoding[M, {n, p}];
```

## Creating a small database of world facts

Storing semantic triples in the form {subject, predicate, object}

```
M["USA", "currency", "Dollar"];
M["MEX", "currency", "Peso"];

M["USA", "capital", "WDC"]
M["MEX", "capital", "CDMX"];

M["USA", "name", "United States"];
M["MEX", "name", "Mexico"];

M["USA", "continent", "North America"];
M["MEX", "continent", "North America"];
```

## Direct queries

```
M["USA", "currency", _]
```

```
Dollar
```

```
M["USA", _, "Dollar"]
```

```
currency
```

```
M[_, "currency", "Dollar"]
```

```
USA
```

```
M["MEX", _, "Peso"]
```

```
currency
```

```
M["MEX", _, "Dollar"]
```

A compound query:

```
M["MEX", M["USA", _, "Dollar"], _]
```

```
Peso
```

## Storing a simple schema

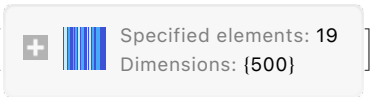
```
M["USA", "predicates", "capital"];
M["USA", "predicates", "currency"];
M["USA", "predicates", "name"];
M["USA", "predicates", "continent"];

M["MEX", "predicates", "capital"];
M["MEX", "predicates", "currency"];
M["MEX", "predicates", "name"];
M["MEX", "predicates", "continent"];


M[ "capital", "predicates", "WDC"];
M[ "capital", "predicates", "CDMX"];
M[ "currency", "predicates", "Dollar"];
M[ "currency", "predicates", "Peso"];
M[ "name", "predicates", "United States"];
M[ "name", "predicates", "Mexico"];
M["continent", "predicates", "North America"];
```

Query for the predicates gives a superposition of possible answers:

```
M["MEX", "predicates", _]
```

```
SparseArray[ 

]
```

## Basic Analogy Operators

### “subject is to object as ... is to ...”

```
SOAnalogy[ sub_, obj_] :=
 { M[ _, M[sub, "predicates", _], obj],  M[ sub, M[_, "predicates", obj], _]}
```

```
SOAnalogy[ "MEX", "Dollar"]
```

```
{USA, Peso}
```

### “object 1  is to object 2 as ... is to ...

```
OOAnalogy[ obj1_, obj2_] :=
 { M[ M[ _, M[ _, "predicates", obj2], obj2], M[_, "predicates", obj1], _],
  M[ M[ _, M[ _, "predicates", obj1], obj1], M[_, "predicates", obj2], _] }
```

```
OOAnalogy["Mexico", "Dollar"]
```

```
{United States, Peso}
```

```
OOAnalogy["Dollar", "Mexico"]
```

```
{Peso, United States}
```

### “what object do two subjects have in common?

```
CommonObject[ sub1_, sub2_] :=
 M[sub1, M[sub2, _, M[sub1, M[sub2, "predicates", _], _] ], _]
```

```
CommonObject[ "USA", "MEX"]
```

```
North America
```

### “what subject do two objects have in common?”)

```
CommonSubject[ obj1_, obj2_] :=
 M[_, M[ M[_, M[_, "predicates", obj1], obj1], _, obj2], obj2]
```

```
CommonSubject[ "North America", "WDC"]
```

```
USA
```