Inference and Execution of Semantic Structures

D. Gueorguiev 12/9/2020

Note on Semantic Signature of a particle

Recall, the semantic signature of compound particle is defined as shown below:
Signature of a thought particle is a matrix $S[i,j]$ where $0 < i < m, 0 < j < n$.
For each particle property the property type, property name and property value of each property are stored in encoded binary format inside the binary signature matrix $S[i,j]$.
Rule for calculation of the semantic signature of a compound thought particle

$V_{comp} = [\, V_{start}\, A_1\, V_2 \, \ldots\, V_{end}\,]$
$sig(V_{start}) = S_{start}[i,j]$
$sig(A_1) = S_1[i,j]$
$sig(V_2) = S_2[i,j]$
$sig([V_{start}\, A_1\, V_2]) = \mathcal{L}A_1(sig(V_{start})) + \mathcal{R}A_1(sig(V_2))$ which is also $m \times n$ matrix

The last line can be written with the following syntax using radicals:

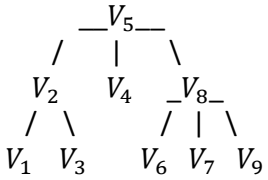
$sig([V_{start}\, A_1\, V_2]) = sig(\{V_{start}\, A_1\}) + sig(\{A_1\, V_2\})$

Obviously, the signature of every compound thought particle is $m \times n$ matrix.
Every connecting particle signature encodes the operation $\mathcal{L}A$ which will be applied to the object particle on the left and the operation $\mathcal{R}A$ which will be applied to the object article on the right. Those operations $\mathcal{L}A$ and $\mathcal{R}A$ will preserve the original information contained in the signature of object particle which is being operated on with an additional information pertaining to the link particle.

An alternative formulation of the semantic signature of V_{comp}

The semantic tree of the compound particle V_{comp} is traversed pre-order. Each particle which is a constituent of V_{comp} is traversed pre-order and its relevant set of properties are added to the signature sorted by type and name. Each particle property occupies a column in the signature matrix. The first particle property serves as a separator from the set of properties corresponding to the previous particle in the pre-order traversal. Recall that the property type, name and value are encoded as binary blobs and are stored as binary arrays within the corresponding signature matrix column.

Let us have the compound particle V_{comp} represented by its elementary particle sequence and semantic tree $stree(V_{comp})$:
$V_{comp} = [V_1\, A_1\, V_2\, A_2\, \ldots\, V_9\, A_9]$
$stree(V_{comp}) =$



The property sets for each V-particle $V_k, k = 1..9$ are given with

$prop\_set(V_k) = \{(t_{k_1}, n_{k_1}, v_{k_1}), (t_{k_2}, n_{k_2}, v_{k_2}), \dots, (t_{k_p}, n_{k_p}, v_{k_p})\}$
and obviously the signature matrix which will be a bitmap will be composed as:

$$S_{comp}[i,j] = \begin{bmatrix} t_{1,1} & t_{1,2} & t_{1,p_1} & t_{2,1} & t_{2,2} & t_{2,p_2} & t_{9,1} & t_{9,2} & t_{9,p_9} \\ n_{1,1} & n_{1,2} \dots n_{1,p_1} & & n_{2,1} & n_{2,2} \dots n_{2,p_2} & \dots & n_{9,1} & n_{9,2} \dots n_{9,p_9} & \\ \underbrace{v_{1,1} \quad v_{1,2} \quad v_{1,p_1}}_{V_1} & & & \underbrace{v_{2,1} \quad v_{2,2} \quad v_{2,p_2}}_{V_2} & & & \underbrace{v_{9,1} \quad v_{9,2} \quad v_{9,p_9}}_{V_9} & & \end{bmatrix}$$

Here $t_{l,p}$, $n_{l,p}$, and $v_{l,p}$ are column bit vectors containing the binary codes which correspond to the property type, name and value of the p-th property of the l-th particle in the pre-order traversal of the semantic tree $stree(V_{comp})$. Notice that the indices i and j in the signature matrix $S_{i,j}$ denote *bitwise access* – i.e. the bit which is on the i-th row and the j-th column in S. The zones

Later we will show that the introduction of *dependent property values* will be desired ~~for paradox detection in thought sequence~~ in certain scenarios. A child particle may have properties with the same type and name as a property in the parent particle V_{par} in the semantic tree of the compound particle V_{comp}.

$V_{comp} = [\dots, V_{child}, A_k, V_{par}, \dots]$

Portion of the semantic tree of V_{comp}:

$$\begin{array}{ccc} & V_{par} & \\ / & & \backslash \\ \dots & & V_{child} \end{array}$$

Let us denote with P_{par} one such property in the parent particle by the triplet $t(V_{par}), n(V_{par}), v(V_{par})$. Then the child particle may have a property with the same type and name as the parent property P_{par} and a value which is dependent of the parent property value $v(V_{par})$. Let us denote by f_v this functional dependence of the child property value. Then the child property can be written as $t(V_{par}), n(V_{par}), f_v\left(v(V_{par})\right)$ or in short-hand notation: $(t, n, v)(V_{par}, f_v)$. For instance, definite terms and indefinite articles can use dependent property values on the parent noun.

Also in general there can be *additive overlap* between the zones corresponding to neighbor particles in the signature matrix. For instance, for V_1 and V_2 the corresponding sector of the signature matrix S_{comp} for the compound particle may look as:

$$\begin{bmatrix} t_{1,1} & t_{1,2} & t_{1,a} & & t_{1,b} & t_{1,p_1} & t_{2,1} & t_{2,2} & t_{2,a-1} & t_{2,a+1} & t_{2,b-1} & t_{2,b+1} & t_{2,p_2} & \\ n_{1,1} & n_{1,2} \dots & n_{1,a} & \dots & n_{1,b} & \dots n_{1,p_1} \dots & n_{2,1} & n_{2,2} \dots & n_{2,a-1} & n_{2,a+1} \dots & n_{2,b-1} & n_{2,b+1} \dots & n_{2,p_2} & \dots \\ v_{1,1} & v_{1,2} & v_{1,a} \oplus v_{2,a} & & v_{1,b} \oplus v_{2,b} & v_{1,p_1} & v_{2,1} & v_{2,2} & v_{2,a-1} & v_{2,a+1} & v_{2,b-1} & v_{2,b+1} & v_{2,p_2} & \end{bmatrix}$$

Here $v_{1,l} \oplus v_{2,l}$ denotes the binary blob which results from summation of the values $v_{1,l}$ and $v_{2,l}$. Here the index l is on the set of all overlapping columns; in this example $l = \{a, b\}$.

Note on vectorization of Semantic Signature

Let us have two elementary particles V_1 and V_2 represented by their corresponding sets of properties:

$prop\_set(V_1) = \{(t_{k_1}, n_{k_1}, v_{k_1}), (t_{k_2}, n_{k_2}, v_{k_2}), \dots, (t_{k_p}, n_{k_p}, v_{k_p})\}$
$prop\_set(V_2) = \{(t_{l_1}, n_{l_1}, v_{l_1}), (t_{l_2}, n_{l_2}, v_{l_2}), \dots, (t_{l_q}, n_{l_q}, v_{l_q})\}$

The property weight of a property models the semantic significance of this property and varies
The property weights for each of the two particles are denoted with:
$prop\_weights(V_1) = \{(t_{k_1}, n_{k_1}, w_{k_1}), (t_{k_2}, n_{k_2}, w_{k_2}), \dots, (t_{k_p}, n_{k_p}, w_{k_p})\}$
$prop\_weights(V_2) = \{(t_{l_1}, n_{l_1}, w_{l_1}), (t_{l_2}, n_{l_2}, w_{l_2}), \dots, (t_{l_q}, n_{l_q}, w_{l_q})\}$

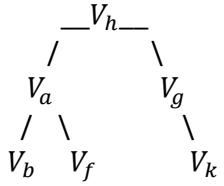
The property weights are stored in the consumed by V P-particle which is a carrier of the corresponding property triplet:

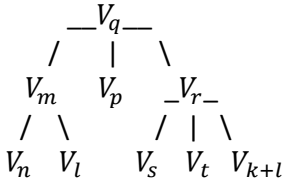
$P_k(V) \longrightarrow (t_k, n_k, v_k), w_k$ such that
$prop\_set(P_k(V)) = \{(t_k, n_k, v_k)\}$
$prop\_weights(P_k(V))$ = $\{w_k\}$
The pairs $(t_1, n_1), (t_2, n_2), \dots, (t_k, n_k)$ form a basis in property space.
Then it becomes obvious how to vectorize a particle signature: it is
$vect(sig(V)) = w_1 v_1 \vec{e}_1 + w_2 v_2 \vec{e}_2 + \cdots + w_k v_k \vec{e}_k$ where the basis vectors $\vec{e}_i, i = 1..k$ are represented by the tuples (t_i, n_i). The basis vectors $\vec{e}_i, i = 1..k$ will define $K(V)$ dimensional vector space also known as the *property dimension* of V. Obviously for composite particles the property dimension will be higher or equal to the property dimension of the elementary particles which compose it: $K(V_{comp}) \geq K(V_i)$.
Two particles V_1 and V_2 have the same semantic meaning if their vectorized semantic signatures are close enough in every property dimension. This is expressed as:
$\forall\ k_i, l_i$ such that $(t_{k_i}, n_{k_i}) = (t_{l_i}, n_{l_i}) \therefore |w_{k_1} v_{k_1} - w_{l_1} v_{l_1}| < \epsilon$ where ϵ is some chosen small constant which is orders of magnitude smaller than the smallest factor wv and the smallest w-factor of the v-quantum $w\Delta v$: $\epsilon \ll min\,(wv) \wedge \epsilon \ll min\,(w)\Delta v$. Since the difference between any two property values is guaranteed to be larger than the v-quantum Δv the condition which remains is $\epsilon \ll min\,(w)\Delta v$ as it implies the other.
If V_1 has a property with type and name (t, n) not available among V_2 property set then V_1 and V_2 do not have the same semantic meaning (abbreviated with $V_1 \not\approx V_2$). Otherwise, we write $V_1 \approx V_2$ stating that V_1 and V_2 are equivalent semantically.

//TO DO: finish this

Note on Semantic distance between thoughts

Let us have the thoughts T_1 and T_2 represented by their particle sequence and their semantic trees:
$T_1 = [V_1\ A_1\ V_2\ A_2\ \dots\ V_k\ A_k]$ and $T_2 = [V_{k+1}\ A_{k+1}\ V_{k+2}\ A_{k+2}\ \dots\ V_{k+l}\ A_{k+l}]$





Let us denote by $T^{(0)}, T^{(1)}, \dots\ T^{(k)}, \dots\ , T^{(n)}$ the tree sequence where each tree $T^{(k)}$ in the sequence is built only from the first k levels of the original tree T. Here $T^{(0)}$ contains only the particle at the root (usually verb) while $T^{(n)}$ consists of the entire tree T.

Two semantic trees T1 and T2 are topologically equivalent if

1) root(T1) = root(T2)
2) for $childlist(root(T1))$ there exist an ordering σ such that $\sigma\left(childlist\left(root(T_1)\right)\right)[i] = childlist(root(T2))[i]$ where 0 < i < |childlist(root(T1))| = $|childlist(root(T2))|$
3) repeat 1) and 2) with $\sigma\left(childlist\left(root(T_1)\right)\right)[i]$ and $childlist(root(T2))[i]$ being the roots of the new pair of trees to compare.

We say that T1 and T2 are $k(\varepsilon)$ - equivalent if and only if all of the following is true:

1) $dist(Vh, Vq)$ is small enough and $cos(vect(Vh), vect(Vq))$ is close enough to 1.0
2)

Comparing thought sequences and paradox analysis

Let us denote by $\mathfrak{S}_A$ a sequence composed of thoughts from a set of contexts $\mathfrak{C}_A$ on the same context path. Let us denote with $T_1(A), T_2(A), \dots, T_K(A)$ the thoughts in $\mathfrak{S}_A$, $K = |\mathfrak{C}_A|$.
Let us denote by $\mathfrak{S}_B$ another sequence composed of thoughts from a set of contexts $\mathfrak{C}_B$ on the same context path. Let us denote with $T_1(B), T_2(B), \dots, T_L(B)$ the thoughts in $\mathfrak{S}_B$.
Let us assume that there are two thoughts $T_k(A)$ and $T_l(B)$ for which the semantic distance is negative.

General Form for the Rules of inference for a sequence of thoughts $\mathfrak{S}_T$

Let us denote by $\mathfrak{S}_T$ a thought sequence composed of thoughts from a set of contexts $\mathfrak{C}$ on the same context path. Let us denote with $T_1, T_2, \dots, T_k$ the thoughts in $\mathfrak{S}_T$. Let us denote by $\mathfrak{T}_V$ a set of V-particles which are entirely contained in the thought sequence $\mathfrak{S}_T$ such that each of the thoughts in $\mathfrak{S}_T$ contains at least one V-particle from $\mathfrak{T}_V$. The set $\mathfrak{T}_V$ will be the *inference trigger* which if present will kick start the synthesis of a new sequence of thoughts $N_1, N_2, \dots, N_l$ which will be the result of the inference. The new ordered sequence of thoughts will be denoted with $\mathfrak{S}_N$. The mapping from $\mathfrak{S}_T$ to $\mathfrak{S}_N$ will represent inference operation which will be triggered by the presence of $\mathfrak{T}_V$.

$$\mathfrak{S}_T(\mathfrak{C}) \xrightarrow{\mathfrak{T}_V} \mathfrak{S}_N(\mathfrak{C})$$

Let us consider the following set of examples:

Example 1a (Simple Inference): I do not know John.
<u>Inference a</u>: I do not know John
I do know John
Example 1b (Simple Inference): I probably do not know John. "Probably" means I am not certain.
<u>Inference a</u>: I am not certain that I do know John. -or-
<u>b</u>: I am not certain that I do not know John.
Example 1c (Simple Inference): He probably does not know John. "Probably" means I am not certain.
<u>Inference</u>: <u>a</u>: He is not certain that he knows John. -or-

b: He is not certain that he does not know John

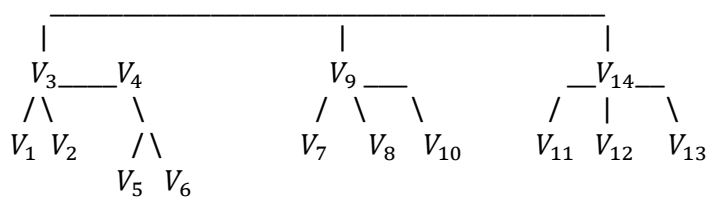
Example 2 (Complex Inference): Dimitar thought John was Peter's son. Peter was scientist and excellent mathematician. Peter's family name was Henderson. John Henderson is from the same city as Dimitar is and apparently has filed a US patent a.b.c.d. I checked and found that US patent a.b.c.d is about a chatbot created and trained from individual personal data scraped from Internet logs. Such chatbot is a spoofing device for somebody's internet persona.
Inference:
Peter's son is the inventor of a device spoofing people's internet persona based on collected from internet personal data.

Example 3 (Inference): Human can breathe using their lungs.
Fish don't have lungs.
Inference: a: Therefore, fish don't breathe! -or-
b: Despite that, fish can breathe!

Example 4 (*Generalization*): *Human can breathe using their lungs.*
Fish don't have lungs.
Generalization: *A can do B using their C. D do not have C. Therefor D cannot do B.*

Learning to Infer

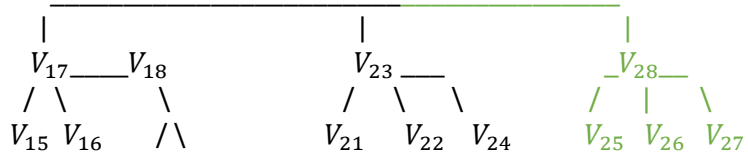
Algorithm for learning to infer can be something like
Learning data:
Humans can breathe using their lungs.
Fish don't have lungs. Therefore, fish don't breathe.
Cars can move using their engines.
Bicycles do not have engines. Therefore bicycles do not move.
People can walk using their legs. Snakes do not have legs.

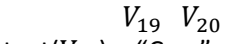


text(V_1) = "Humans"
text(V_2) = "can"
text(V_3) = "breathe"
text(V_4) = "using"
text(V_5) = "lungs"
text(V_6) = "their"

text(V_7) = "Fish"
text(V_8) = "don't"
text(V_9) = "have"
text(V_{10}) = "lungs"

text(V_{11}) = "Therefore"
text(V_{12}) = "fish"
text(V_{13}) = "don't"
text(V_{14}) = "breathe"

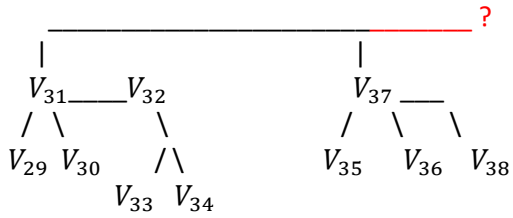




text(V_{15}) = “Cars”
text(V_{16}) = “can”
text(V_{17}) = “move”
text(V_{18}) = “using”
text(V_{19}) = “engines”
text(V_{20}) = “their”

text(V_{21}) = “Bicycles”
text(V_{22}) = “don’t”
text(V_{23}) = “have”
text(V_{24}) = “engines”

text(V_{25}) = “Therefore”
text(V_{26}) = “bicycles”
text(V_{27}) = “don’t”
text(V_{28}) = “move”



text(V_{29}) = “People”
text(V_{30}) = “can”
text(V_{31}) = “walk”
text(V_{32}) = “using”
text(V_{33}) = “legs”
text(V_{34}) = “their”

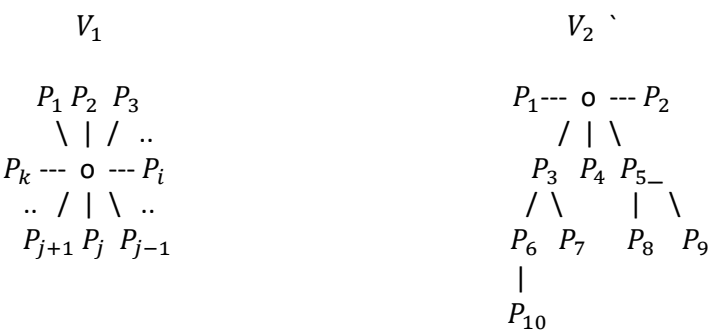
text(V_{35}) = “Snakes”
text(V_{36}) = “don’t”
text(V_{37}) = “have”
text(V_{38}) = “legs”

$T_1 : [V_i\ A_j, i = 1..6, j = 1..5]$
$T_2 : [V_i\ A_j, i = 7..10, j = 6..8]$
$T_3 : [V_i\ A_j, i = 11..14, j = 9..11]$
$T_4 : [V_i\ A_j, i = 15..20, j = 12..16]$
$T_5 : [V_i\ A_j, i = 21..24, j = 17..19]$
$T_6 : [V_i\ A_j, i = 25..28, j = 20..22]$
$T_7 : [V_i\ A_j, i = 29..34, j = 23..27]$
$T_8 : [V_i\ A_j, i = 35..38, j = 28..30]$

Training set: $T_1, T_2 \rightarrow T_3; T_4, T_5 \rightarrow T_6$
Test set: $T_7, T_8 \rightarrow T_9$

How could the inference work? Generally, by setting up a predictor (regressor, neural net, etc) and train it.
Simple way to perform the learning step: *semantic structure property swarm with adaptive plasticity*. The properties which are “seen” more than once in the training examples on similar relative locations are strengthened. Tuning the strengthening algorithm with a meta-algorithm is desirable. How does the meta algorithm do the tuning? One output of the tuning algorithm would be the strengthening or weakening of the retainability rank of each property. Second optimization cohort in the algorithm would be to maintain alternative property representation for each semantic particle and to rank each alternative representation. Each V-particle will be semantically identified by DAG of properties as shown below:

Properties and Dependent Properties:



The set of properties associated with each naked V-particle will be inferred during the continuous semantic parsing which has been discussed in other documents in this collection. Properties are created following the rules for creating properties:

Rules for creating properties:
By splitting existing particles:
Weight, rotation speed, acceleration
New particles:
Semantic inference: Weight comparisons with existing thoughts
//TODO: finish this

We need only one inference algorithm – both for generating new thoughts and for thoughts entering the system.

//TODO: finish this

Types of Inference Processes

We recognize three types of inference processes – Inductive, Deductive and Abductive inference (Peirce, 1878).

Inductive Inference

It is kind of specialization. Replaces entities which are in context with entities which are out of context (Supplement_8-12-20_page4.jpg).

Deductive Inference

It is a kind of generalization. Replaces entities which are out of context with entities in context (Supplement_8-12-20_page4.jpg).

Abductive Inference

Multi-step process for building and refining a hypothesis

Inference can be posed as a discrete optimization problem solved by a reinforcement learning algorithm.

Hypothesis is synthesized and refined in a set of iterations. After it matches the input and output the hypothesis will be used for making an inference, ranked and stored for a future use.

//TODO: finish this

Learning Model for Inference Processes

Hypothesis Synthesis of new thoughts Hypothesis

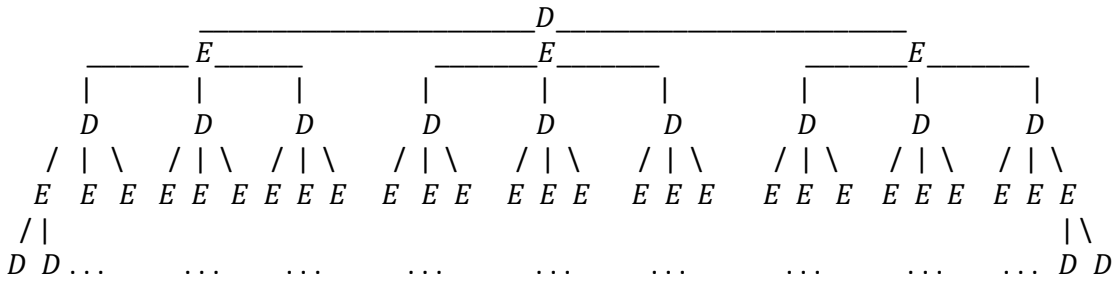
Execution of thoughts

Continuous adaptive learning process for thoughts which are to be executed
Let the graph $G_1(V, A)$ represents a thought which is marked for execution. The set V represents its V-particles and A is the set of its A-particles. Portions of this thought are mapped to various execution structures. Execution structures are constructed of D-particles and various E-particles. D-particles are directors – they redirect the execution path to appropriate subgraphs of E-particles. Therefore the D-particles implement conditional processing of the newly synthesized thought composed originally from naked particles. redirecting execution to a subgraph of E-particles which

//TODO: finish this

Execution Structures and Learning

An *execution structure* is a weighted directed graph $G(E, A)$ composed of the node set E consisting of E-particles and the set of arcs A with weights W. The arc set gives directionality of the execution flow while the weights determine the stopping condition for loops. The E-particles are composite in nature – new E-particle can be obtained from merging k E-particles E_1 E_2 … E_k which are connected in a path P. The rules for merging execution structures determine how new structures relevant for execution will be reconciled with the already existing ones.
An execution structure, as any other structure, is assigned a rank through attachment of ranking particles (R-particle).
An execution structure can operate on a semantic structure (graph) or on another execution structure. In the latter case we are talking about *execution-modifying execution structure*.
Execution particles can be created by executing new or inferred thoughts. New execution structures can be assembled by executing new or inferred thoughts.
Each execution structure has at least one target on which it will operate. The execution structure provides a plan what to do with that target. The target can be another execution structure or a regular semantic structure. Each execution structure can be reduced / merged to a simpler execution structure which has the following form:



The root of the reduced execution tree is a D-particle.

Discussion on Execution Particles

Each E-particle receives a target from D-particle. The target can be either a semantic structure or an execution structure. The E-particle applies its execution sequence to the target and produces an output which it passes down the reduced execution tree to the next D-particle. The output is either an execution structure or a semantic structure.
Each E-particle repels or attracts other E/D-particles based on their execution signature.

The Execution sequence of Execution Particles

The execution sequence of an E-particle can be thought as a pipeline in which some elementary operations are applied selectively to the target.

Mapping of Semantic Structure to an Execution Structure

The Semantic Structure subject to mapping is considered in a set of contexts. Let S denote the semantic structure under consideration. Let $C_1, C_2, \dots, C_n$ be a set of contexts which enclose the semantic structure S. Each execution structure will have affinity to certain semantic signatures. Larger semantic structures can be assembled from lower order execution structures as graph where each node is a lower order execution structure.

//TODO: finish this

Execution Space vs Semantic Space

Each Semantic Structure represents a point in *Semantic Space* and each Execution Structure represents a point in *Execution Space*. As mentioned earlier in Mapping of Semantic Structure to an Execution Structure regions of Semantic Space are mapped to regions in Execution Space.
There is duality between execution space and semantic space. Combining semantic particles and aggregating semantic structures creates new points in semantic space.

Bibliography

Fischer, H. R. (2001). Abductive Reasoning as a Way of Worldmaking. *Foundations of Science, special issue on "The Impact of Radical Constructivism on Science", edited by A. Riegler, 2001, vol. 6, no.4*, 361-383.

Ian J. Goodfellow, J. P.-A.-F. (December 2014). Generative Adversarial Nets. *NIPS'14: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, (pp. 2672–2680).

Kloesel, N. H. (1992). *The Essential Peirce, Selected Philosophical Writings, Vol 1 (1867-1893).* Bloomington, IN, USA: Indiana University Press, 601 North Morton Str, Bloomington, IN 47404-3797 USA.

Peirce, C. S. (1878, August). Deduction, Induction, and Hypothesis. *Popular Science Monthly, Vol 12*, pp. 470-482.

Tesnière, L. (2015). *Elements of Structural Syntax.* Amsterdam / Philadelphia: John Benjamins Publishing Company.

Wang-Zhou Dai, Q. X.-H. (2019). Bridging Machine Learning and Logical Reasoning by Abductive Learning. *33rd Conference on Neural Information Processing Systems (NeurIPS 2019).* Vancouver, Canada.