Hyperdimensional Turing Machine

Misha Klopukh & William Hahn

Department of Mathematical Sciences
Florida Atlantic University

52nd Southeastern International Conference on Combinatorics, Graph Theory and Computing

March 11, 2021

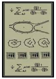


Table of Contents

Table of Contents

FAU DEPARTMENT OF MATHEMATICAL SCIENCES
Charles E. Schmidt College of Science
Florida Atlantic University

What are Hypervectors

A Hypervector space, H, is a set of high dimensional vectors (10,000+ dim) with the following:

1. Similarity $d : H \times H \to \mathbb{R}$
   - $d(x, x) \geq d(y, z)$
   - $x \sim y \iff d(x, y) > t$, where t is some threshold.
   - $x \not\sim y$ with high probability, x, y chosen at random.
2. Bind (*) : $H \times H \to H$
   - $a * a = 1$
   - $a * b \not\sim a$
   - $a * b \not\sim b$
3. Bundle $(+)$: $H \times H \to H$
   - $a + b \sim a$
   - $a + b \sim b$
   - $a * (b + c) = a * b + a * c$
4. Permute $r : H \to H$
   - $r(a) \not\sim a$
   - $r^{-1}(r(a)) = a$

What do hypervectors do?

- Initialize a, b, c, d randomly
- Let $r = a * b + c * d$
- $r \not\sim a$, $r \not\sim b$, $r \not\sim c$, $r \not\sim d$
- $r * a = a * a * b + a * c * d = b + a * c * d$
- $a * c * d \not\sim a, b, c, d$
- $r * a \sim b$, but
- $r * a \not\sim a$, $r * a \not\sim c$, $r * a \not\sim d$
- Similarly, for $r * b, r * c, r * d$

What do hypervectors do?

- r represents a function
  $R: \{a, b, c, d\} \subset H \to \{a, b, c, d\} \subset H$
- $R(x) = \underset{y \in \{a,b,c,d\}}{\arg\max}\ d(r * x, y)$
- $R(x) \begin{cases} a \mapsto b \\ b \mapsto a \\ c \mapsto d \\ d \mapsto c \end{cases}$
- We can encode general "switch" statements in this form

An Example of Hypervectors: Binary Bipolar

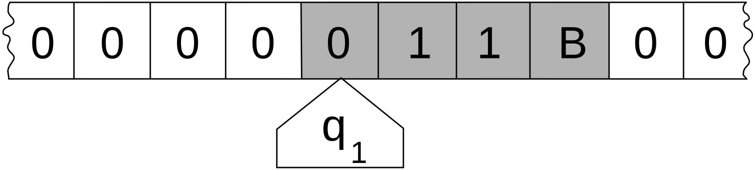
- (10,000-dim+) vectors with elements initialized randomly with each element as -1 or 1
- $d(a, b) = \frac{a \cdot b}{|a||b|}$: cosine similarity
- $a + b$: element-wise addition

  (Can also be select each element from one of the vectors at random)
- $a * b$: element-wise multiplication
- $r(a) = r(a_1, a_2, \ldots, a_n) = (a_n, a_1, \ldots, a_{n-1})$: re-indexing

Table of Contents

Recap of Turing Machines

- Head on an infinite tape
- Finite State space: Q
- Finite Symbol space: Γ
- Head $= \delta : \; Q \times \Gamma \to Q \times \Gamma \times \{\text{Left}, \text{Right}\}$
- Tape $= \Gamma^*$
- Since Q and Γ have finitely many symbols/states, we can encode each of them as a random hypervector.



Turing tape graphic

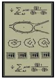
Basic Turing Counter

- As a starter task, we used binary counting
- $Q = \{w, r\}$, $\Gamma = \{0, 1, x\}$
- δ is defined as follows:

Γ / Q	w	r
0	(1, r, Right)	(0, r, Right)
1	(0, w, Left)	(1, r, Right)
x	Impossible	(x, w, Left)

- The tape is initialized as $\ldots 00000000\underset{\wedge}{x}$
- The state is initialized as r
- The number is read from the tape whenever the head reaches the x at the end

Table of Contents

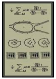


Basic Hypervector Turing Counter

- We create the following random bypolar hypervectors:
  - One for reading each state/symbol: {Read0, Read1, ReadX, ReadR, ReadW}
  - One for setting each state/symbol: {Write0, Write1, WriteX, WriteR, WriteW}
  - One for moving in each direction: {Left, Right}
- We construct our rule hypervector as follows:

$$
\begin{aligned}
T = {} & Read0 * ReadW * (Write1 + WriteR + Right) \\
& + Read0 * ReadR * (Write0 + WriteR + Right) \\
& + Read1 * ReadW * (Write0 + WriteW + Left) \\
& + Read1 * ReadR * (Write1 + WriteR + Right) \\
& + ReadX * ReadR * (WriteX + WriteW + Left)
\end{aligned}
$$

- At each step, we take $T *$ Tape[i] $*$ State and compute the most similar state, symbol, and direction



Less Basic Hypervector Turing Counter

- Having separate symbols for reading and writing states feels inelegant
- We need them in the previous architecture to avoid this: $0 * r * (0 + r + \text{Right}) = 0 * 0 * r + 0 * r * r + 0 * r * \text{Right} = 0 + r + 0 * r * \text{Right}$, which is similar to both 0 and r.
- We can eliminate this need by introducing a new operation: Stack ($\wedge$), and its inverse Unstack ($\vee$)
  - $a \wedge b = a * r(b)$
  - $a \wedge b \not\sim a$, $a \wedge b \not\sim b$
  - $a \vee b = r^{-1}(a * b)$
  - $(a \wedge b) \vee a = b$
  - $(a \wedge (b \wedge c)) \vee (a \wedge b) = c$
  - Stack is not associative, and should be read from right to left

Less Basic Hypervector Turing Counter

- We now only create the following random bypolar hypervectors:
  - One for each state: {R, W}
  - One for each symbol: {0, 1, X}
  - One for moving in each direction: {Left, Right}
- We construct our new rule hypervector as follows:

$$
\begin{aligned}
T = {} & 0 \wedge W \wedge (1 + R + Right) \\
& + 0 \wedge R \wedge (0 + R + Right) \\
& + 1 \wedge W \wedge (0 + W + Left) \\
& + 1 \wedge R \wedge (1 + R + Right) \\
& + X \wedge R \wedge (X + W + Left)
\end{aligned}
$$

- At each step, we take $T \vee (\text{Tape[i]} \wedge \text{State})$ and compute the most similar state, symbol, and direction

A Few Further Improvements:

- If we do not wish to restrict our induced function to 3 subsets of our tape:
- We can create 3 additional random hypervectors: {Symbol, State, Move}
- We can then modify each piece of our rule from the form $0 \wedge W \wedge (1 + R + Right)$ to the form $0 \wedge W \wedge (Symbol * 1 + State * R + Move * Right)$
- Now, to get the new symbol, state, and move, we perform $T \vee (\text{Tape[i]} \wedge \text{CurrentState}) * Symbol$, $T \vee (\text{Tape[i]} \wedge \text{CurrentState}) * State$, and $T \vee (\text{Tape[i]} \wedge \text{CurrentState}) * Move$ respectively, and find the most similar item of the entire known set.

A Few Further Improvements

- For small enough tapes (and high enough dimensions), we can encode the entire tape in a single hypervector.
- We create a new random hypervector called TapeIndex
- We can now define the hypervector

  $$\text{Tape} = \mathit{TapeIndex} * X + \sum_{n=1}^{N} r^n(\text{TapeIndex}) * 0$$
- We initialize the hypervector $\text{CurrentTapeIndex} = \text{TapeIndex}$
- To read the tape, we compute the most similar vector to $\text{Tape} * \text{CurrentTapeIndex}$
- To move the tape left, we set $\text{CurrentTapeIndex} = r(\text{CurrentTapeIndex})$, and to move right we set $\text{CurrentTapeIndex} = r^{-1}(\text{CurrentTapeIndex})$

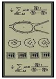
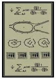
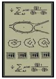


Table of Contents



Hypervectors for Learnable Computing

- Many constructions of hypervector operations, like the ones used, are differentiable
- This allows "learnable" hypervector architectures using methods like SGD
- Hypervectors support various other optimization/search algorithms as well
- These architectures may also funtion as "Neural Turing Machines"



Hypervectors for Robust Computing

- Since hypervectors are built with random noise, they are robust to perturbations
- In preliminary testing, the second architecture can successfully count to 1024 with 10% noise added each step
- This can also be considered a proof of concept for more practical general computing architectures with hypervectors
- Dr. Hahn and I are currently working on a hypervector RAM machine architecture, which should be equally robust
- Fast hardware implementations of hypervectors are being developed

Thank You!

Questions?