UHASSELT
KNOWLEDGE IN ACTION

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie

master in de informatica

Masterthesis

Vector Symbolic Architectures: Foundations and Applications to the Embedding of Words and Databases

Senn Robyns

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :

Prof. dr. Stijn VANSUMMEREN

2023
2024

UHASSELT
KNOWLEDGE IN ACTION

www.uhasselt.be
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie

master in de informatica

Masterthesis

Vector Symbolic Architectures: Foundations and Applications to the Embedding of Words and Databases

Senn Robyns

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :

Prof. dr. Stijn VANSUMMEREN

Acknowledgements

This thesis marks the culmination of my master's journey, and its completion would not have been possible without the support and contributions of several individuals.

First, I would like to express a sincere thank you to my promotor, prof. dr. Stijn Vansummeren. His guidance, expertise, and continuous support throughout this research process have been invaluable. From suggesting this fascinating topic to providing insightful feedback during our weekly meetings, his mentorship was a key part in shaping my research and this thesis.

Second, I would like to thank the Redwood Center for Theoretical Neuroscience and Berkeley Wireless Research Center at UC Berkeley for providing a comprehensive curriculum on VSAs. This program was essential as it provided me with great foundational knowledge about the topic, serving as a strong base upon which I could build my research.

I am also grateful for the professors and assistants of the faculty of Computer Science at UHasselt, with a special thanks to the departments of AI and Data Management. The knowledge and skills I have acquired during my time here provided a strong foundation for this research. I would also like to thank the faculty of Statistics and Data Science for allowing me to take supplementary courses on the theoretical aspects of statistics and data science next to my Computer Science degree.

Lastly, I am incredibly thankful for the support and encouragement I received from my friends and family throughout this intensive journey. They provided me with a constant source of motivation and strength, allowing me to focus on my academic pursuits.

Samenvatting

Deze masterproef verkent *Vector Symbolic Architectures (VSAs)*, een veelbelovend domein binnen *artificiële intelligentie (AI)* dat de kloof tussen de symbolische en connectionistische paradigma's wil overbruggen. Traditioneel domineerden deze twee paradigma's het AI-onderzoek. Symbolische AI, ook bekend als *Good Old-Fashioned AI (GOFAI)*, blinkt uit in kennis representatie en redeneren door symbolen en hun onderlinge relaties te manipuleren. Deze aanpak is echter minder geschikt voor het omgaan met onzekerheid en het leren van ambiguë data. Connectionistische modellen, zoals artificiële neurale netwerken, daarentegen, verwerken informatie op een gedistribueerde manier, waardoor ze beter in staat zijn om te leren van grote datasets en om te gaan met onzekerheid. Echter worstelen deze met het efficiënt representeren en verwerken van complexe, hiërarchische structuren. VSAs bieden een alternatief door mechanismen te introduceren om symbolische informatie te coderen en te manipuleren binnen een gedistribueerde representatie.

Een gedistribueerde representatie slaat informatie op door deze te verspreiden over een groot aantal componenten, in tegenstelling tot lokale representaties waar elk concept een eigen, uniek element heeft.

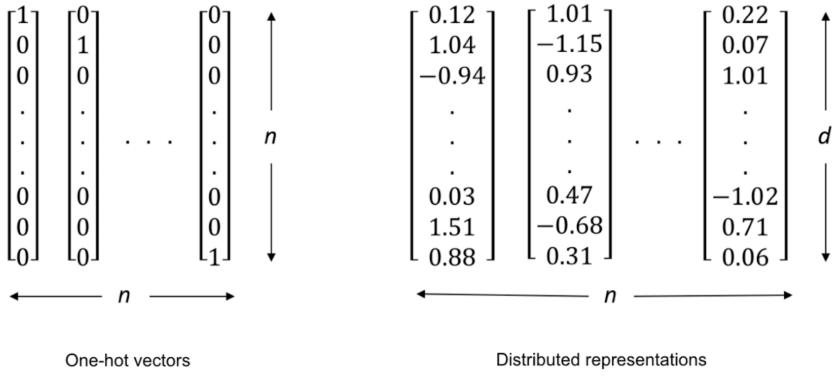


Figure 1: Examples of One-hot encodings vs. Distributed representations over a vocabulary of size n. [1]

We illustreren dit aan de hand van de representaties in figuur 1 waar deze representaties woorden encoderen over een vocabulary van grootte n. Een mogelijke representatie is het gebruik van een one-hot encoding die aan elk woord een unieke vector toekent waarbij de lengte van deze vectoren dus gelijk zijn aan de vocabulary size n. Slechts één positie in de vector is "hot" (1), de rest is nul. Dit is een symbolische, duidelijke codering, maar mist de mogelijkheid om nuances in de relaties tussen woorden (similariteit) weer te geven. Bovendien leidt dit tot zeer hoog-dimensionale, sparse vectoren bij grote vocabularies.

Neurale netwerken streven naar gedistribueerde representaties, waarbij woorden worden gerepresenteerd als dichtere, kortere vectoren met (reële) getallen. Woorden met een vergelijkbare

[1] https://blog.materialis.ai/distributed-representations-of-atoms.html

betekenis (zoals "dokter" en "ziekenhuis") hebben dan 'dichtere' representaties in deze vectorruimte. De nabijheid wordt gemeten met similariteits- of afstandsmaten zoals dot-product, Euclidean distance of Cosine similarity. Het doel is om representaties te creëren waarbij vergelijkbare concepten een hogere similariteit of kleinere afstand hebben.

Echter hebben traditionele connectionistische modellen moeite met het representeren van complexe, hiërarchische structuren. VSAs pakken dit probleem aan door mechanismen te introduceren die symbolische structuren en redeneringen mogelijk maken binnen een connectionistisch framework. Dit maakt VSAs een interessante kandidaat voor *Neuro-Symbolic AI (NeSy)*, dat streeft naar een integratie van de leercapaciteiten van connectionistische modellen met de redeneerkracht van symbolische AI.

Introductie tot Vector Symbolic Architectures

VSAs maken gebruik van hoog-dimensionale vectoren, genaamd hypervectors (HV), om informatie op een gedistribueerde manier te representeren. Deze HVs hebben duizenden componenten (dimensionaliteit) en maken gebruik van de quasi-orthogonaliteits eigenschap die zegt dat willekeurig gekozen vectoren in zulke hoog-dimensionale ruimtes bijna orthogonaal, en dus dissimilar, aan elkaar zijn. Deze willekeurige keuze duid op de sampling van de componenten van een HV, die gebeurd volgens een op voorhand bepaalde distributie.

Ik wil hier even verduidelijken dat de componenten van een HV refereren naar de individuele waardes voor iedere dimensie van een HV. De 4 key componenten van een VSA-architectuur verwijzen naar naar conceptuele definities omtrent de invulling van een VSA-architectuur, dit zijn dus 2 verschillende defenities rondom componenten en de context is dus belangrijk indien we over dit concept rapporteren.

Een typische VSA-architectuur bestaat uit 4 belangrijke componenten. Voor we deze componenten in meer detail bekijken is het belangrijk om hier aan te halen dat er verschillende VSA modellen bestaan. Deze modellen implementeren allemaal de 4 key componenten, hun belangrijkste verschil is de sampling distributie die gebruikt wordt voor de componenten van de HVs. Hierdoor hebben de verschillende modellen ook enkele verschillen op vlak van de invulling en implementatie. We focussen in deze samenvatting op het Binary Splatter Codes (BSC) model dat zijn componenten selecteert van een Bernouilli distributie en dus binaire componenten heeft.

De 4 key componenten van een VSA-architectuur zijn de volgende:

- **Similarity Measures:** VSAs representeren concepten als HVs. Om de relatie tussen deze HVs te bepalen en hun similariteit te kwantificeren, gebruiken VSAs similariteits measures.

  Populaire measures zijn dot product, cosine similarity en Hamming similarity. Waarbij de Hamming similarity de measure is die gebruikt wordt in het BSC model, deze telt het aantal componenten dat overeenkomt in twee binaire HVs en normaliseert deze door dit te delen door het aantal componenten of dimensies. Zoals eerder vermeld hangt de keuze van een specifieke similarity measure af van het gekozen VSA-model.

- **Atomic HVs:** Atomic HVs vormen de basisbouwstenen van representaties in VSAs. Ze representeren de kernconcepten van een probleem, zonder enige vooraf gedefinieerde associaties. Deze atomic HVs worden willekeurig gegenereerd volgens een bepaalde op voorhand gedefinieerde distributie, zodat ze quasi-orthogonaal zijn aan elkaar. De verdeling van de similariteit tussen random HVs voor het BSC model wordt gevisualiseerd in Figuur 2. We observeren dat hoe hoger de dimensionaliteit van de HVs, hoe meer de similariteit geconcentreerd is rond de waarde 0.5. Dit impliceert dat de kans op het genereren van dissimilar HVs sterk toeneemt naarmate de dimensionaliteit toeneemt.

- **Fundamentele operaties:** VSAs gebruiken drie fundamentele operaties om complexe representaties te creëren en te manipuleren:

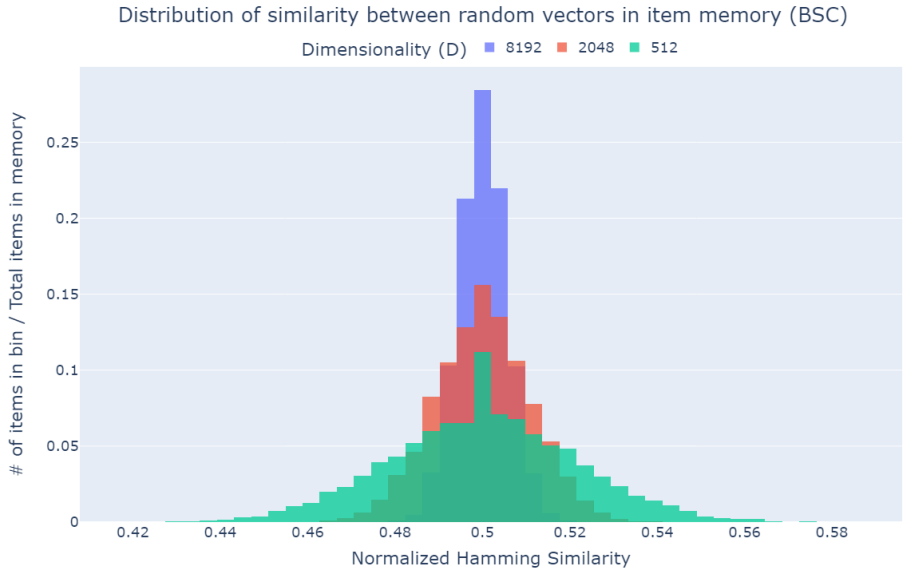


Figure 2: Histogram van normalized hamming similarity tussen random BSC HV en 10 000 andere random gegenereerde HVs.

  - Bundling: Combineert meerdere HVs tot één enkele HV. De resulterende HV behoudt een hoge similariteit met elk van de input HVs, waardoor het geschikt is om sets of collecties te representeren.

    Omdat de bundling operatie typisch geïmplementeerd wordt aan de hand van een sommatie van de componenten stellen we de bundle voor als volgt:

    $$\mathbf{s} = \mathbf{x} + \mathbf{y} + \mathbf{z}$$

    Waar $\mathbf{x}$, $\mathbf{y}$ en $\mathbf{z}$ de input HVs voorstellen en $\mathbf{s}$ de resultaat of bundled HV is. Merk op dat ieder van de input HVs een hoge, en gelijkaardige similariteit heeft aan de resulterende HV op die manier vat de resulterende HV $\mathbf{s}$ de informatie van de input HVs samen.

  - Binding: Creëert een associatie tussen twee HVs, deze associatie van 2 HVs is vergelijkbaar met het creëren van een key-value paar. De resulterende HV is dissimilar aan de originele HVs, maar behoudt wel informatie over de structuur van de associatie. Dit maakt het mogelijk om bijvoorbeeld relaties tussen objecten te representeren. We stellen de binding operatie voor als volgt:

    $$\mathbf{r} = \mathbf{k} \bigcirc \mathbf{v}$$

    De resulterende, gebonden HV $\mathbf{r}$ is dissimilar met zijn input HVs $\mathbf{k}$ en $\mathbf{v}$. Toch behoudt het een ander soort similariteit, genaamd *gestructureerde similariteit*. Dit betekent dat twee gebonden HVs alleen similar zijn als zowel hun key HVs als hun value HVs similar zijn. We illustreren dit met een voorbeeld; als we een andere binding $\mathbf{r}' = \mathbf{k}' \bigcirc \mathbf{v}'$ creëren, dan is $\mathbf{r}$ alleen similar aan $\mathbf{r}'$ als $\mathbf{k}$ similar is aan $\mathbf{k}'$ **en** $\mathbf{v}$ similar is aan $\mathbf{v}'$.

  - Permutatie: Deze operatie werkt op een enkele input HV en herschikt de componenten van deze HV. Dit kan gebruikt worden om ordinale informatie te representeren. Deze operatie wordt niet gebruikt in onze toepassingen, we refereren voor extra info naar de thesistekst.

- **Item memory:** De item memory slaat alle atomic HVs op en koppelt ze aan hun bijbehorende concepten. De item memory is essentieel voor het 'opschonen' van HVs die ruis

bevatten door de combinatie van bundling en binding. Dit 'opschonen' gebeurt door de meest gelijkende atomic HV in de item memory te selecteren.

Door deze bouwstenen te combineren, kunnen VSAs een breed scala aan symbolische informatie representeren en manipuleren, van eenvoudige concepten tot complexe structuren.

Ik verwijs hier ook naar de thesistekst in hoofdstuk 2 voor een meer gedetailleerd overzicht van de operaties alsook een experimenteel onderzoek naar de eigenschappen van deze operaties en hun capaciteit om informatie voor te stellen.

Nu we een begrip hebben van wat een VSA is gaan we de praktische toepasbaarheid van VSAs onderzoeken in twee concrete use-cases. Namelijk het genereren van betekenisvolle word embeddings aan de hand van een verzameling van documenten, alsook het encoderen van tabulaire datasets om deze vervolgens te queryen.

Word Embeddings met VSAs

Voor het genereren van de word embeddings ontwikkelden we een nieuwe VSA-gebaseerde methode. Deze methode is gebaseerd op het principe van word-document co-occurrence, wat inhoudt dat woorden die vaak samen in dezelfde documenten voorkomen, semantisch gerelateerd zijn en dus vergelijkbare vectorrepresentaties zouden moeten hebben.

Het leerproces van de VSA word embeddings is als volgt:

- Initialisatie:
  Elke document in de corpus krijgt een unieke, willekeurig gegenereerde atomic HV. Voor elk woord in de vocabulaire wordt een "lege" HV geïnitialiseerd.
- Coderen van semantische similariteit door bundling:
  Voor elk woord in een document wordt de HV van het document "gebundeld" met de HV van het woord. Dit proces wordt herhaald voor alle documenten in de corpus. Woorden die frequent voorkomen in dezelfde documenten zullen dus vergelijkbare HVs accumuleren, waardoor hun semantisch verwantschap wordt vastgelegd.
- Normalisatie (optioneel):
  Na het doorlopen van de corpus kunnen de HVs van de woorden worden genormaliseerd om de consistentie te waarborgen en te voldoen aan de eigenschappen van het gekozen VSA-model.

We kunnen dit algoritme conceptueel inbeelden als het verplaatsen van woord HVs in een hoogdimensionale ruimte. De document HVs fungeren als ankerpunten in deze ruimte, en de woord HVs worden dichter bij de document HVs geplaatst waarin ze voorkomen. Woorden die vaak in dezelfde documenten voorkomen, zullen dus dichter bij elkaar in de ruimte terechtkomen.

We verwijzen naar de thesistekst in hoofdstuk 3 voor meer details over het algoritme zoals de geïmplementeerde GPU-optimalisaties alsook een analyse omtrent een optimale dimensionaliteit van onze HVs.

Om de effectiviteit van deze VSA-gebaseerde methode te evalueren werden experimenten uitgevoerd met behulp van de WikiText-103 dataset. Deze dataset bestaat uit een verzameling van kwaliteitsvolle en aanbevolen Wikipedia artikels, van deze dataset gebruiken we een willekeurige subset van 10.000 documenten wegens computationele redenen. We pre-processen deze documenten door een binaire co-occurrence matrix op te stellen, deze heeft voor ieder document een vector, waarbij de index van een woord op 1 staat indien dit woord voorkomt in dit document.

Een initiële analyse van de geleerde word embeddings toont aan dat de VSA learning methode in staat is om semantische relaties tussen woorden te coderen. Woorden die semantisch gerelateerd zijn, zoals "film" en "acteur", hebben inderdaad vergelijkbare HVs, zoals we kunnen zien in Figuur 3.

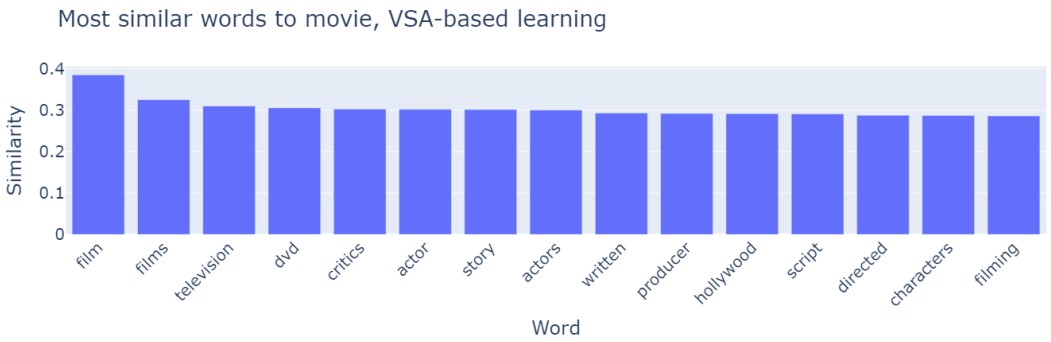


Figure 3: Plot van 15 meest similar woorden in vergelijking met de 'movie' HV uit al onze VSA-based geleerde embeddings.

Voor het formeel evalueren van onze word embeddings maken we gebruik van de WordSim353 dataset. Deze evaluatiedataset bevat human-assigned similarity scores voor woordparen, we vergelijken deze met de similarities van de geleerde VSA-embeddings. Om dit te formaliseren berekenen we de correlatie tussen de human-assigned en de geleerde similarities, een hoge correlatie duidt aan dat de geleerde semantische relatie overeenkomen met die van de evaluatie dataset. Verder vergelijken we de prestatie van onze VSA geleerde embeddings met Latent Semantic Analysis (LSA), een state of the art leermethode voor word embeddings op basis van term-document matrices.

De resultaten van de experimenten, te zien in de eerste rij van tabel 1, toonden aan dat de VSA-embeddings een lagere correlatie behalen met de menselijke beoordelingen in de WordSim353 dataset dan de LSA methode. We concludeerden hier dat state of the art methodes zoals LSA beter generaliseren door de ruis uit dataset te filteren om zo de significante semantische relaties uit de term-document matrix te halen. Echter merken we op dat in deze tabel de score op de validatie dataset van de VSA-embeddings en de originele embeddings identiek zijn. We valideerden dit door te kijken naar de correlatie van de similariteiten tussen de woord paren van de geleerde embeddings met die van de originele term-document matrix. Deze correlatie bedraagt 0.982 wat suggereert dat de VSA-methode wel nauwkeurig de aanwezige semantische relaties in de originele data kan vastleggen maar dus geen ruis wegfiltert.

Word pair	VSA	LSA	Term-document matrix
Pearson correlation (binary term-document matrix)	0.284	0.366	0.284
Pearson correlation (tf-idf term-document matrix)	0.345	0.374	0.389

Table 1: Tabel noteert de Pearson correlatie tussen de geleerde similarities en de human-assigned similarities voor alle woord paren in de WordSim353 dataset. Voor compleetheid noteren we ook de correlatie tussen de similarities voor de woord paren van de vectoren in de originele term-document matrices en die van de human-assigned scores.

Deze laatste observatie leidt tot de hypothese dat we mogelijks de geleerde VSA-embeddings kunnen verbeteren door de semantische relaties die aanwezig zijn in de originele term-document matrix, waar we van leren, te verbeteren.

Om deze hypothese te bevestigen, hebben we een extra experiment uitgevoerd op dezelfde verzameling documenten, maar nu met *term frequency – inverse document frequency (TF-IDF)*-vectorisatie van de gegevens in onze term-document matrix. De waardes binnen de matrix zijn

nu TF-IDF-scores in plaats van simpele binaire co-occurrence waardes. Deze TF-IDF-waarden hebben als doel beter het belang van termen binnen de documenten te vergelijken met die van de term over het hele corpus van documenten, wat leidt tot verbeterde semantische embedding in onze oorspronkelijke term-document matrix.

Zoals we kunnen zien in de tweede rij van tabel 1, valideren we onze hypothese aangezien de score op de evaluatie dataset voor VSA-embeddings sterk stijgt van 0.284 naar 0.345.

We concluderen dus dat de VSA-gebaseerde methode een veelbelovende aanpak is voor het creëren van word embeddings. De experimenten bevestigen dat VSAs in staat zijn om betekenisvolle semantische relaties tussen woorden te coderen. Verder leerden we dat de kwaliteit van de VSA embeddings sterk afhankelijk is van de kwaliteit van de input data. Toekomstig onderzoek zou zich dus kunnen richten op het verder verbeteren van de semantische relatie die aanwezig is in de input term-document matrix.

VSA Databases

In de tweede praktische toepassing van VSAs zullen we onderzoeken hoe we HVs en VSA-operaties kunnen gebruiken om tabulaire gegevens te encoderen, op te slaan en deze vervolgens ook te queryen. Onze benadering heeft dus als doel de principes en eigenschappen van VSAs te benutten om een robuuste en efficiënte methode voor het omgaan met tabulaire gegevens te creëren.

Het is belangrijk op te merken dat we voor dit verkennende experiment aannemen dat de set van attribuutwaarden voor een gegeven attribuut in onze database geen inherente similarity structure heeft. Zo streven we ernaar dat de similarity tussen de HV-codering van twee database records evenredig is aan het aantal identieke attribuutwaarden dat ze delen over alle kolommen. We zullen daarom enkel gebruik maken van categorieke attributen, en dus geen gebruik maken met numerieke of ordinale attributen zoals leeftijd.

Conceptueel idee

Eerst observeren we dat een record conceptueel kan worden gezien als een set of verzameling van key-value pairs, waarbij de sleutel de attribuut- of kolom naam is en de waarde de waarde van dat record voor dat attribuut is. Als we deze conceptuele weergave van een record combineren met de bundle en bind VSA-operaties zoals eerder gedefinieerd, kunnen we de records in tabel 2 als volgt coderen:

record	**attribute 1**	**attribute 2**	**attribute 3**
record 1	value 1	value 2	value 3
record 2	value 4	value 5	value 6

Table 2: Voorbeeld van 2 records, dewelke ieder 3 attributen hebben, uit een tabulaire dataset of database.

Geëncodeerd als volgt:

$$\mathbf{r1} = (\mathbf{a1} \bigcirc \mathbf{v1}) + (\mathbf{a2} \bigcirc \mathbf{v2}) + (\mathbf{a3} \bigcirc \mathbf{v3})$$

$$\mathbf{r2} = (\mathbf{a1} \bigcirc \mathbf{v4}) + (\mathbf{a2} \bigcirc \mathbf{v5}) + (\mathbf{a3} \bigcirc \mathbf{v6})$$

Herinner dat de bundling operatie (+) gebruikt werd om conceptueel sets van HVs weer te geven, en de binding operatie ($\bigcirc$) om associaties te creëren tussen HVs of key-value pairs.

Architectuur van de VSADB

Gebruik makend van dit idee creëren we de *VSA Database (VSADB)* bestaande uit de volgende belangrijke componenten.

Kolomrepresentatie:
Elke kolom in de database wordt gerepresenteerd door een unieke atomic HV en een codebook. De atomic HV dient als identifier voor de kolomnaam, terwijl de codebook een mapping bijhoudt tussen attribuutwaarden en hun corresponderende HVs. Deze codebooks zijn dus dictionaries die gebruik maken van hashing om zo een waarde te kunnen linken aan zijn atomic HV representatie, waarbij we deze in 'constante' tijd op te kunnen vragen dankzij hashing.

Record encodering:
Records of rijen in de database worden gecodeerd als HVs door de volgende stappen te volgen:

1. Initialiseren van attribuutwaarde HVs: Voor elk attribuut in de rij wordt de corresponderende HV opgehaald uit het codebook van de betreffende kolom.
2. Binding: De atomic HV van elke kolom wordt gebonden aan de HV van de attribuutwaarde, waardoor een unieke HV ontstaat voor elk attribute-value pair.
3. Bundling: De resulterende attribute-value HVs worden gebundeld om een enkele HV te vormen die de gehele rij representeert. Dit resulteert in het conceptuele idee dat we eerder besproken hebben rondom het encoderen van records als HVs.
4. Normalisatie: De gebundelde HV wordt genormaliseerd om consistentie te waarborgen en te voldoen aan de eigenschappen van het gekozen VSA-model.

Opslag:
De genormaliseerde record HVs worden vervolgens opgeslagen in de VSADB. Voor dit experiment werd een codebook gebruikt als opslag-structuur, maar andere methoden, zoals gespecialiseerde vector databases, zijn ook mogelijk. De codebook linkt het id van het record in de originele database aan zijn geëncodeerde HV, zodat snelle opvraging aan de hand van deze ids kan gebeuren.

Queryen van de VSADB

De VSADB ondersteunt similariteitsgebaseerde zoekopdrachten door middel van query HVs. Een query HV wordt geconstrueerd op dezelfde manier als een record HV, door attribuutwaarden te binden aan kolom HVs en deze resulterende gebonden HVs te bundelen. Het query proces bestaat dus uit de volgende stappen.

1. **Creëren van de query HV:**
   De gebruiker specificeert de gewenste attribuutwaarden voor de zoekopdracht, en de query HV wordt geconstrueerd op identiek dezelfde manier als een record geëncodeerd wordt, echter worden alleen de attribute-value pairs gebundled die gematched moeten worden volgens de query. Een voorbeeld volgt in de sectie over de experimenten.
2. **Similarity Search:**
   De query HV wordt vergeleken met alle opgeslagen rij HVs met behulp van een similarity measure, zoals cosinus similariteit of Hamming similariteit afhankelijk van het model dat gebruikt wordt.
3. **Ophalen van de meest similar records:**
   De records met de hoogste similarity scores worden opgehaald en aan de gebruiker gepresenteerd.

Merk op dat het queryen van de VSADB aan de hand van een query HV eigenlijk neerkomt op een nearest-neighbor search met de geëncodeerde database records.

Experimenten en Resultaten

Om de prestaties van de VSADB te evalueren, werden experimenten uitgevoerd met een dataset van persoonsgegevens. De dataset bevatte attributen zoals voornaam, achternaam, stad, staat

(provincie) en geslacht. Deze dataset werd uitgebreid met 10 000 willekeurig gegenereerde records om de schaal en complexiteit te verhogen. De originele kleine dataset die nog niet uitgebreid is wordt weergegeven in tabel 3

index	fname	lname	city	state	gender
0	John	Doe	Riverside	NJ	M
1	Jack	McGinnis	Philadelphia	PA	M
2	John	Repici	Riverside	NJ	M
3	Stephen	Tyler	Sioux Falls	SD	M
4	John	Blankman	Sioux Falls	SD	M
5	Joan	Anne	Denver	CO	F
6	Jack	Repici	Riverside	NJ	M
7	Lilly	Repici	Philadelphia	PA	F

Table 3: Kleine real-world tabulaire dataset die gebruikt en ook uitgebreid werd met random records in the experiment. Random gegenereerde records hebben een random fname en lname string, een random gesampelde stad met bijhorende staat en een random gesampelde gender.

De experimenten toonden aan dat de VSADB in staat is om database rijen efficiënt te encoderen als HVs en accurate resultaten terug te geven voor zowel point- als partial match queries. Point-queries specificeren waarden voor alle attributen, terwijl partial match queries waardes voor een subset van attributen specificeren.

We geven een voorbeeld van een partial match query uit onze experimenten. Het doel van deze query is het identificeren van all de leden van de Repici familie die in Riverside wonen.

Ter verduidelijking overlopen we nog even de stappen voor het construeren en uitvoeren van deze query:

1. **Creëren van de Query HV:**
   - Opvragen van de input HVs: Verkrijg de atomaire attribute- of kolom-HVs voor 'lname' en 'city', en de value-HVs voor 'Repici' en 'Riverside', uit hun respectieve kolom codebooks.
   - Binding van de attribute-HVs: Bind elk attribuut zijn atomaire HV met de bijbehorende value-HV.
   - Bundling van de HVs: Bundle de resulterende attribuut-value pair HVs om een enkele query-HV te vormen.
   - De resulterende query-HV wordt dus geconstrueerd als:

$$\mathbf{q} = \mathbf{lname} \bigcirc \mathbf{repici} + \mathbf{city} \bigcirc \mathbf{riverside}$$

2. **Uitvoeren van de NN search:**
   Gebruik de gebundelde query-HV **q** om een NN search uit te voeren op de record-HVs die zijn opgeslagen in het codebook van onze VSADB.

Het resultaat van deze query, oftewel de top 10 meest similar database HVs worden weergeven in tabel 4

Als we kijken naar de resultaten merken we op dat we de 2 relevante records inderdaad identificeren. Maar we merken ook op dat er onder de 10 meest similar records, records zijn die gedeeltelijk overeenkomen met onze query HV. Dit zijn records die ofwel Repici als achternaam hebben of in Riverside wonen. Als we naar de genormaliseerde hamming-similarities van deze volledige en gedeeltelijke matches met onze query HV kijken, merken we dat de similarity voor de volledige matches, zoals verwacht, aanzienlijk hoger is dan de andere. Maar we kunnen ook

fname	lname	city	state	gender	Normalized Hamming Similarity
Jack	Repici	Riverside	NJ	M	0.695
John	Repici	Riverside	NJ	M	0.689
wpmrlq	pnthyi	Riverside	CA	M	0.598
Lilly	Repici	Philadelphia	PA	F	0.597
John	Doe	Riverside	NJ	M	0.592
gnumfw	ilbywx	Orem	UT	M	0.519
xopccc	fxgopl	Colorado Springs	CO	F	0.518
lfkaha	wbutxq	Sioux Falls	SD	F	0.518
nlntid	vuljpm	Reno	NV	F	0.518
zcudtl	mxbtnf	Albuquerque	NM	F	0.517

Table 4: 10 meest similar records uit de VSADB voor de NN search met de query Hv voor de Repici familieleden die in Riverside wonen, we rapporteren ook de normalized hamming similarities met deze query HV voor ieder record HV.

opmerken dat de similarity voor de gedeeltelijke matches ook aanzienlijk hoger is dan die voor records die helemaal niet overeenkomen.

Gebaseerd op deze resultaten kunnen we nu de volgende interessante vraag stellen: Kunnen we een similarity threshold definiëren die ervoor zou zorgen dat we alleen de relevante records ophalen, en dus niet records die slechts gedeeltelijk of zelfs helemaal niet overeenkomen met onze query HV?

Om deze vraag te beantwoorden werd een theoretische analyse uitgevoerd om deze threshold value te bepalen voor het filteren van zoekresultaten. Deze threshold value zorgt ervoor dat alleen rijen met een voldoende hoge similariteit worden geïdentificeerd, waardoor false positives worden geminimaliseerd. Voor een in-depth theoretische analyse van deze threshold value alsook de parameters die de accuraatheid ervan bepalen refereren we naar de thesistekst.

We vermelden hier wel dat de experimenten bevestigden dat de threshold value effectief is in het optimaliseren van de precisie van de zoekresultaten, terwijl de hoge recall behouden blijft. Bijvoorbeeld, voor onze voorbeeld query en zijn parameters, identificeerden we een threshold value van 0.653. Indien we deze toepassen op onze resultaten zien we dat we inderdaad enkel de 2 relevante records zullen bijhouden en dus een perfecte accuraatheid bekomen voor onze query.

Verder is er ook onderzoek gedaan naar een optimale dimensionaliteit. Dit leidde tot de conclusie dat we voor een beperkte dimensionaliteit van 1500, records die tot wel 20 (categorieke) attributen bevatten kunnen encoderen waarbij we deze ook kunnen queryen met een perfecte precisie, recall en dus ook accuraatheid. Merk op dat dit misschien een hoge dimensionaliteit lijkt, echter voor VSA applicaties ligt de dimensionaliteit typisch rond de 10 000. We merken hier ook op dat we met het BSC model werken die binaire componenten heeft, een record wordt dus efficiënt geëncodeerd in slechts 1500 bits.

Merk op dat aangezien onze records die 5 string values bevatten, dus met slechts 1500 bits efficiënt geëncodeerd kunnen worden. Dit betekent dat we de 10 007 records zelf, in principe in slechts 1.9MB kunnen encoderen en opslaan. Er moeten hier echter wel enkele kanttekeningen gemaakt worden rond welke data we moeten bijhouden om deze efficiënte encoderingen te kunnen decoderen. Een mogelijkheid is om de originele records bij te houden met een referentie naar hun geëncodeerde versie aan de hand van hun index, dit is de methode die wij toepasten in de experimenten. Het decoderen kan dan rechtstreeks gebeuren door de indices van de resulterende records van onze similarity based search met onze query HV te gebruiken om de bijhorende originele records op te vragen. Merk op dat bij deze variant de VSADB gebruikt wordt als een extensie van de tabulaire dataset die querying op basis van de encoderingen toelaat. We moeten hier namelijk de volledige originele dataset blijven opslaan om de originele records op te vragen

gegeven de resultaten van de query. Maar we kunnen ook enkel de attribuut of kolom codebooks bijhouden die de mapping van de attribute values naar hun HVs bevat, en dan de resulterende records van een query decoderen aan de hand van de unbinding operatie om zo het opslag-gebruik mogelijks te minimaliseren. Voor een gedetailleerde uitleg van beide mogelijkheden alsook het decoderen van een record aan de hand van de unbinding operatie refereren we opnieuw naar de thesistekst.

Conclusie

We concluderen dat onze VSADB implementatie efficiënte encoderingen van tabulaire records toelaat en daarbij accurate similarity based querying ondersteund aan de hand van een threshold value waarvoor we een theoretische formule afgeleid hebben.

Belangrijk is om te benadrukken dat de huidige implementatie van de VSADB zeker geen vervanging is voor een volledig database systeem. Het bied echter wel een sterke basis voor verder onderzoek waarbij er dus nog vele mogelijkheden zijn tot verbetering en uitbreidingen:

- **Ondersteuning voor geavanceerde queries:** De VSADB ondersteunt momenteel alleen point- en partial match queries. Meer complexe queries, zoals joins of aggregaties, worden nog niet ondersteund.
- **Aanname van dissimilariteit:** De huidige implementatie gaat ervan uit dat er geen inherente similariteit is tussen de waarden binnen een kolom. Dit betekent dat de VSADB vooralsnog alleen geschikt is voor categorieke data, en niet voor numerieke of ordinale data.

Conclusie

We hebben met het onderzoek in de masterthesis de veelbelovende mogelijkheden van VSAs verkend en verduidelijkt. We hebben de basisprincipes van VSAs belicht en hun vermogen om symbolische informatie te representeren en te manipuleren in een gedistribueerde context gedemonstreerd. Door middel van experimenten hebben we de eigenschappen van VSAs onderzocht en twee concrete toepassingen geanalyseerd: het genereren van betekenisvolle word embeddings en het encoderen en queryen van tabulaire data.

We hebben voorstellen gedaan rond algoritmes en architecturen voor het genereren van word embeddings en, het encoderen en queryen van tabulaire data. We hebben hierbij de kwaliteiten en eventuele tekortkomingen onderzocht en geverifieerd aan de hand van onderbouwde, expermimentele analyses.

Ondanks de geïdentificeerde tekortkomingen van deze implementaties, bieden de experimentele en kritische analyses een sterke basis voor eventuele uitbreidingen of een verdere verkenning van de praktische toepassing van Vector Symbolic Architectures.

Contents

Chapter 1

Introduction

Artificial Intelligence (AI) research has historically been characterized by two predominant paradigms, namely the *symbolic paradigm* and the *connectionist paradigm*. [1]

Symbolic AI, also known as *Good Old-Fashioned AI (GOFAI)*, is based on the representation of information through symbols and their interrelations. It aims to enable problem-solving through the manipulation and exploration of these symbols and relationships. This paradigm stems from the ideas behind logic-based reasoning and is typically explored through logic programming [2]. This paradigm thus excels at knowledge exploration using logical inference and rule-based systems.

However, this also reveals its major flaw, namely the inability to handle uncertainty and learn from ambiguous data due to its reliance on explicit symbol/knowledge manipulation. Another drawback, with respect to achieving brain-like computing, is their reliance on highly reliable hardware [3]. Unlike the brain, which can tolerate and recover from minor errors, even small computational errors in symbolic systems can lead to catastrophic failures.

In contrast, the connectionist paradigm, characterized by artificial neural networks, processes information in a distributed manner across interconnected units (neurons), reflecting a more brain-like computation style. While the connectionist paradigm experienced a new burst of interest with the introduction of Deep Learning models and applications like ChatGPT, recent attention has shifted towards exploring alternative approaches. This shift is motivated by the limitations of traditional connectionist models. A common problem with these models is that they often struggle with structured and hierarchical data, leading to challenges in capturing nuanced relationships and patterns within some datasets mainly due to their incapability in handling complex structures and learning elaborate data representations. A more recent problem lies in the fact that these traditional connectionist architectures often pose significant energy demands [4]. This high energy consumption underscores the need for more efficient computational approaches. Moreover, with the rise of stream-based data processing in dynamic and uncertain environments, where AI systems must adapt and perform robustly, energy-efficient solutions become imperative.

Vector Symbolic Architectures (VSAs), also known as *Hyperdimensional Computing (HDC)*, offer a promising alternative to address these challenges. They use distributed representations in high-dimensional vector spaces to encode and manipulate symbolic information in an efficient and robust manner [5]. VSAs thus enable the use of structured, symbolic-like representation of data within connectionist paradigms, aiming to bridge the gap with symbolic approaches.

Thesis Objective

The objective in this thesis is to explore the capabilities and limitations of VSAs in the context of AI in general, and machine learning in particular. By means of two use cases targeting textual data and tabular data respectively, this thesis seeks to experimentally investigate the strengths and limitations of VSAs and their potential as a viable alternative or companion to existing AI solutions in these domains.

Chapter 2

Introduction to Vector Symbolic Architectures (VSAs)

2.1 Motivation: Why VSAs?

As mentioned in the Introduction, connectionist models, primarily artificial neural networks, excel at handling the uncertainty that stems from learning from large amounts of data. This capability can be attributed to the use of *distributed representations* [6].

Distributed representations encompass a holistic approach to encoding information by distributing it across a large number of interconnected units or neurons. This contrasts with local representations, like one-hot encoding, where each concept is represented by a single active unit, and symbolic discrete representations, like records with specific fields, where information is stored in predefined, isolated memory locations using bit patterns.

Example: Representing Words as Vectors

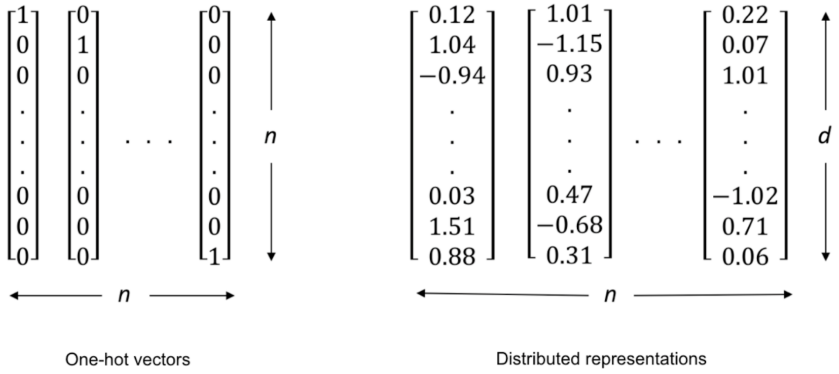


Figure 2.1: Examples of One-hot encodings vs. Distributed representations over a vocabulary of size n. [1]

To illustrate this, let us consider the representation of words in text documents. One-hot encoding offers a simple approach: each word is assigned a unique vector with a length equal to the vocabulary size. Only one position in the vector is "hot" (set to 1), while the rest are zeros. While this method provides a symbolic-like, distinct encoding for each word, it lacks the ability to capture nuanced relationships between words (similarity). Additionally, it leads to very high-dimensional, sparse representations for large vocabularies.

[1] https://blog.materialis.ai/distributed-representations-of-atoms.html

Neural networks aim to compute distributed representations, where words are represented as denser, shorter vectors of (real) numbers. In this scheme, words with similar meanings (like "cat" and "dog") would have 'closer' representations in this fixed-dimension vector space. Where closeness is defined by a vector similarity- or distance measure like the dot-product, euclidean- or cosine distance/similarity. We aim to create distributed representations such that representations of similar concepts have a higher similarity or lower distance.

This distributed manner of representing information (and computing on such representations) has several advantages [5]:

- Robustness to Noise and Errors: Distributing information across many units makes the representation robust to noise and errors. Even if some units are corrupted or malfunctioning, the overall representation can still retain its meaning and be used for inference [7].
- Graceful Degradation: Distributed representations exhibit graceful degradation, meaning that the performance of the system degrades *gradually* as the representation becomes increasingly corrupted [8]. This property is crucial for real-world applications where noise and uncertainty are inevitable.
- Generalization: Distributed representations facilitate generalization, allowing the system to recognize and respond to novel inputs that are similar to previously encountered ones [8, 6]. This ability is essential for learning and adaptation in dynamic environments.
- Representation of Similarity: Distributed representations can capture and represent the similarity between concepts and objects [9]. Similar concepts will have similar representations, allowing the system to leverage this similarity for tasks like classification and analogy-making.

These properties make artificial neural networks well-suited for applications where we need to learn from noisy and incomplete data, generalize to new situations, and perform in the presence of errors.

However, artificial neural networks still have difficulties with representing and learning from large-scale, complex, hierarchical, and compositional structures. These shortcomings of both paradigms raised interest in *Neuro-Symbolic AI (NeSy)*, which aims to obtain a *best-of-both-worlds*: the key goal in NeSy research "is the integration of two fundamental cognitive abilities: the ability to learn from the environment, and the ability to reason from what has been learned" [10].

One particular possibility to integrate these capabilities is to incorporate symbolic structure and symbolic reasoning in connectionist models. VSAs are one concrete proposal in this respect, they introduce mechanisms to encode and manipulate symbolic information within a distributed representation.

2.2 Core principles / Building Blocks of VSAs

When creating distributed representation we aim to distribute the information we want to represent over all components. Individual components of a vector do usually not have a predefined meaning as opposed to symbolic representations, although NN's can assign implicitly learned features to components. This leads to a structural difference between a symbolic and a distributed representation; in the latter the state of an individual component or vector dimension does not convey useful information without knowing the state of the other components.

Distributed representations as a model for representing and computing on information does come with several challenges, as discussed in the *Superposition catastrophe* [11], *Fodor and Pylyshyn criticisms of connectionism* [12] and *Challenges of language modeling by connectionist representations posed by Jackendoff* [13]. Most of the criticism boils down to the fact that conventional connectionist representations struggle to represent hierarchical compositional structures effectively and efficiently. For example representing (simple) fixed length sentences like "Mary likes

John" vs. "John likes Mary" using tuples or sequences, or representing documents by means of *large* sets. The problem becomes even more eminent when we try to represent more complex sentence or document structures by means of recursive structures, trees or graphs. From these examples we conclude that information about the arrangement and/or order of multiple elements within these structures can be lost due to the nature of the classical superposition of vectors used in standard distributed representations [5]. VSAs address these challenges by introducing operations that exploit the probabilistic properties of vectors in *high-dimensional-spaces*. These properties enable VSAs to capture the richness of symbolic representations while maintaining the advantages of distributed representations, paving the way for more efficient and robust information processing.

The fundamental building block of VSAs is the high-dimensional vector, also known as a *hypervector* (HV). These vectors typically have thousands of dimensions, allowing us to depend on the *near-orthogonality property*. This property states that in high-dimensional spaces, randomly chosen vectors are nearly orthogonal, and thus dissimilar to each other. This is a crucial part of VSAs since most of its theory relies on it.

We state that a Vector Symbolic Architecture typically consists of the following core elements:

- A *Similarity Measure*: Because we work with vectors to represent concepts, both simple and complex, we need some similarity measure that allows us to compare them and quantify their relatedness.
- *Hypervectors (HVs)*: We typically start by creating a set of atomic HVs, representing the most basic concepts of our problem. These atomic vectors typically don't inherently contain a form of association, as with symbolic representations, they each represent a unique, dissimilar concept and thus are 'randomly' generated.
- A set of fundamental operations called *bundling*, *binding*, and possibly a *permutation* operation. These operations provide building blocks for creating complex representations from simple ones (atomic HVs) by combining and associating information contained in HVs. They thus allow us to manipulate these representations and reason over them.
- An *item memory* that stores the atomic HVs and associates them to their respective concepts. It is an essential part of the *"clean-up" procedure* which retrieves the closest atomic HV(s), using a similarity measure, from a set of noisy representations generated by applying the fundamental operations. It thus allows for the comparison of more complex, composite HVs to the basic concepts they encompass in our original problem space.

We refer to a *VSA model* as a specific implementation of a Vector Symbolic Architecture. A VSA model thus always contains and builds upon, the 4 core elements mentioned previously. More specifically, it dictates the implementation of the fundamental operations, the similarity measure, and how we create random (atomic) HVs such that they are quasi-orthogonal or dissimilar.

For the remainder of this section we give a general overview of each of these core elements of a VSA. We clarify our general overview by giving the implementation details for one of the most straightforward VSA models called *Binary Spatter Codes* (BSC). Finally, we close this chapter by giving an overview of the implementation details of 2 other VSA models called *Multiply-Add-Permute (MAP)* and *Holographic Reduced Representations (HRR)*.

We will denote HVs using bold lowercase letters, e.g., $\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{examplehv}$.

2.2.1 Similarity Measures

In order to evaluate the properties and capabilities of VSAs we need a function that enables us to measure the similarity between 2 HVs. Because HVs are vectors we can use standard vector similarity measures, of which the most popular is the dot product or, its normalized version, cosine similarity.

Specifically, recall that the dot product is defined as:

$$\delta_{dot}(\mathbf{x}, \mathbf{y}) = \sum_{i}^{\mathbf{D}} \mathbf{x}_i \mathbf{y}_i$$

It's normalized version, the cosine similarity is defined as:

$$\delta_{cos}(\mathbf{x}, \mathbf{y}) = \frac{\delta_{dot}(\mathbf{x}, \mathbf{y})}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2}$$

However, some models like BSC use a specialized similarity measure, allowing for more suitable and efficient measures. Since BSC use binary components we can get a more efficient operation by using the *(normalized) Hamming distance*, or get the *(normalized) Hamming similarity* by taking its additive complement.

The Hamming similarity is defined by the number of components that two input vectors agree on:

$$\delta_{ham}(\mathbf{x}, \mathbf{y}) = \sum_{i}^{\mathbf{D}} 1\{\mathbf{x}_i = \mathbf{y}_i\}$$

We can normalize this similarity measure, restricting the range to [0, 1] as follows:

$$\delta_{ham\_norm}(\mathbf{x}, \mathbf{y}) = \frac{\delta_{ham}(\mathbf{x}, \mathbf{y})}{\mathbf{D}}$$

For the remainder of this thesis we will denote the similarity measure adopted by a VSA model as the bivariate function $\delta()$. While implementation details may differ per model, the core principles discussed in the remainder of the thesis hold true regardless of these variations. The most important being the *concentration of measure phenomenon*, stating that a randomly generated HV will, with high probability, have a similarity to other randomly generated HVs that is close to the *absolute zero*. Where this absolute zero value depends on the similarity measure used by the model. For example, it is 0 for models that use cosine similarity, as can be seen in figure 2.2. The absolute zero value for the BSC model is 0.5 if the normalized hamming similarity is used, as can be seen in figure 2.3. We observe that the shape of the similarity distribution is similar, irrespective of the model used, yet the absolute zero value may differ.

2.2.2 Atomic HVs

Before we begin representing more complex relations and structures, we typically start by generating a set of *atomic HVs*. Their purpose is to represent the core elements of our problem without encapsulating any elaborate connections or structures. Typically these atomic HVs don't contain any form of association between each other. They serve as the building blocks whereupon more complex representations are constructed using the fundamental operations revealed in subsection 2.2.3.

We typically create these random, dissimilar, HVs by drawing their elements from some predefined distribution. For example, the BSC model generates a random HV by selecting for each dimension an element of $\{0, 1\}$, and thus uses a Bernoulli distribution to generate the vector's components.

The key difference with typical connectionist representations is the fixed (high-)dimension of each atomic HV. The random HV generation process, using a carefully designed distribution to pick the HV's components, allows us to create new, dissimilar, vectors on the fly without needing to increase the dimensionality along the way. We can confirm this by taking another look at figure 2.3. From this figure we also observe that that an increase of the dimensionality $\mathbf{D}$, leads to an exponential increase in the amount of quasi-orthogonal vectors.

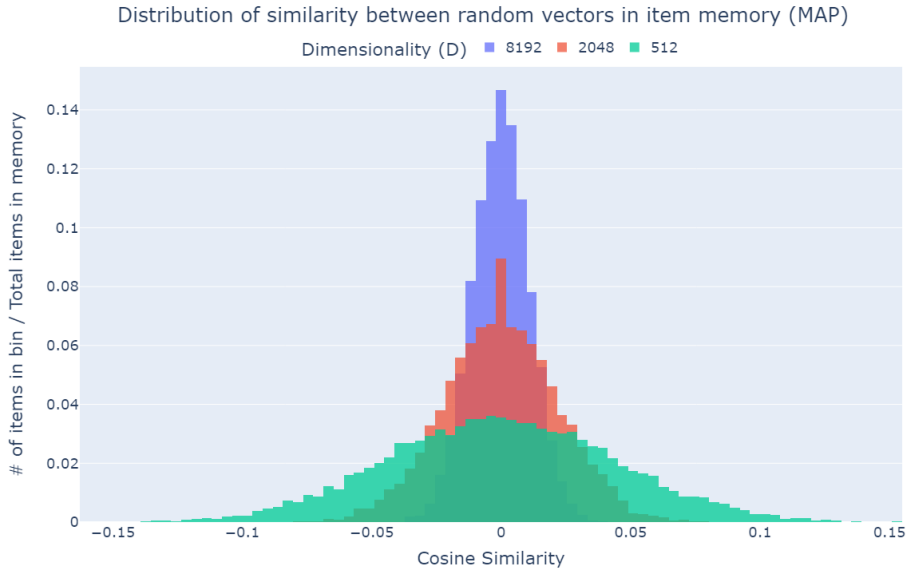


Figure 2.2: Histogram of Cosine similarity between a random MAP HV and all other, randomly generated atomic HVs in item memory (10 000 in total)

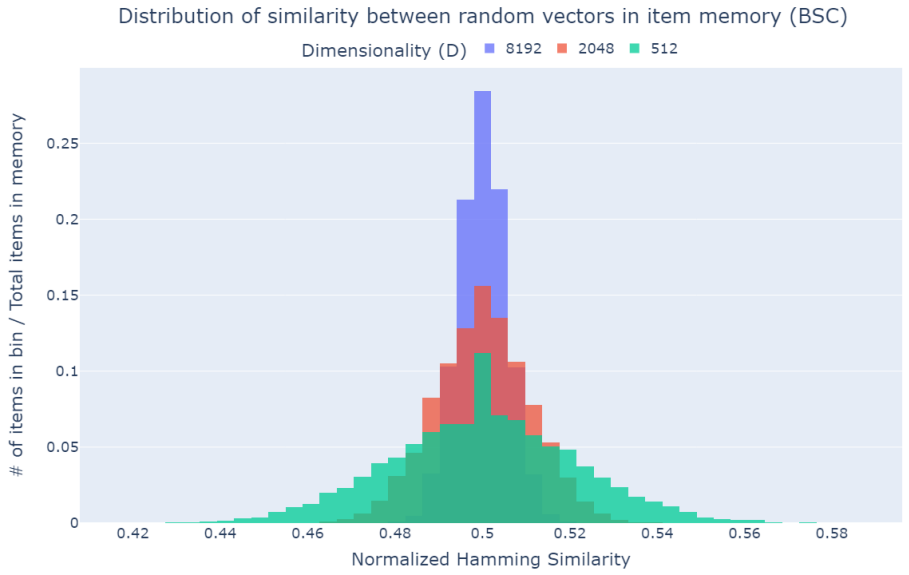


Figure 2.3: Histogram of normalized hamming similarity between a random BSC HV and all other, randomly generated atomic HVs in item memory (10 000 in total)

2.2.3 Fundamental Operations

VSAs provide a set of fundamental operations that allow us to manipulate and reason using the atomic HVs.

Bundling
This operation combines multiple HVs into a single HV, while preserving similarity between the resultant HV and each of its inputs. It is typically implemented using some form of element-wise addition. This is why we will denote it as follows:

$$\mathbf{s} = \mathbf{x} + \mathbf{y} + \mathbf{z}$$

Where $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ are the input HVs and $\mathbf{s}$ is the resultant or bundled HV.

We make a few important notes about this operation:

- After creating the bundled vector it might be necessary to apply a model-specific normalization function $f()$ denoted as:

  $$\mathbf{s} = f(\mathbf{x} + \mathbf{y} + \mathbf{z})$$

  This might be necessary to preserve the *type* of the components in the resultant HV to allow for further computations. BSC for example uses the *thresholded sum* to retain the binary components of the resultant vector as can be seen in Figure 2.4a. A policy for breaking ties is necessary when an even amount of HVs in bundled, it is most common to randomly assign a value from $\{0, 1\}$ to a tied component; see Figure 2.4b.

Dimension	1	2	3	4	...	**D**
x	1	1	0	1		0
y	0	1	0	0		1
z	0	1	1	0		1
$+ (= \mathbf{s})$	1	3	1	1		2
$f(\mathbf{s})$	0	1	0	0		1

(a) Uneven amount of bundled HVs, no tie-breaking policy needed

Dimension	1	2	3	4	...	**D**
x	1	1	0	1		0
y	0	1	0	0		1
$+ (= \mathbf{s})$	1	2	0	1		1
$f(\mathbf{s})$	<u>0</u>	1	0	<u>1</u>		<u>0</u>

(b) Even amount of bundled HVs, random tie-breaking policy used for dimensions 1, 4 and **D**

Figure 2.4: Examples of bundling BSC HVs

- An important property of the bundling operation is that the resultant HV $\mathbf{s}$ is equally similar to its input HVs, yet this similarity decreases with the amount of inputs HVs. We can investigate the interaction between the dimension of the HVs and the size of the bundled HV on the similarity with its input HVs in figure 2.5. Here we observe that the similarity between the input HVs and the resultant bundle HV seems insensitive to the dimensionality of the HVs, yet the standard deviation of this similarity is dependent on the dimensionality. When we combine this with the results from figure 2.3 where we observe that the similarity between random HVs is more concentrated around the absolute zero for larger dimensionalities we can conclude that in higher dimensions, bundled representations maintain a more distinct separation from random vectors, allowing us to more reliably recognize input HVs.

  Using this insight we define the *bundling capacity* as follows: The percentage of input HVs that have a greater similarity with the resultant, bundled HV, than any non-input atomic HV in item memory. Our previous conclusions suggest that a higher dimensionality would allow for a greater bundling capacity. To confirm this hypothesis and validate the effectiveness of this definition, we conducted an experiment.

  In this experiment, we varied both the dimensionality of the atomic HVs and the size of the bundled HV to observe their effect on the bundling capacity. For each dimension we begin by generating an item memory of 10 000 atomic HVs. Within each dimensionality setting, we explore different bundle sizes ($\mathbf{S}$) by randomly selecting $\mathbf{S}$ atomic HVs from item memory as input HVs for the bundle. Lastly, we calculate the bundle capacity by taking the percentage of input HVs that are among the top $\mathbf{S}$ most similar HVs from item memory to the bundled HV. We repeated this experiment, as well as all other experiments in this section, over 10 batches, meaning we repeated the experiment over the different dimensionalities 10 times.

  Our results, depicted in figure 2.6, clearly demonstrate the relationship between dimensionality, bundling size, and bundling capacity stated earlier. We thus confirm that larger dimensionalities lead to greater bundling capacities.

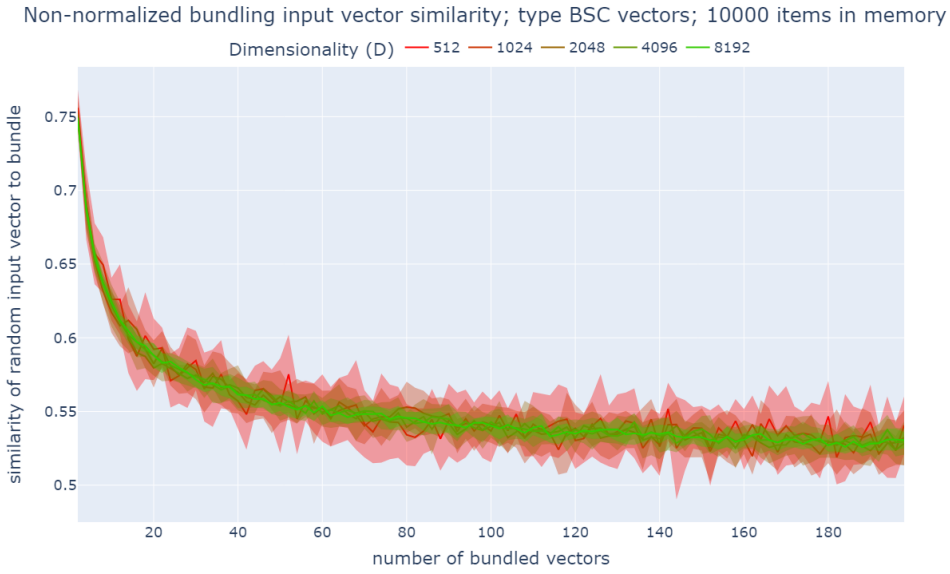


Figure 2.5: Similarity of bundled HV to one of its input HVs for several dimensionalities and bundle sizes. No intermediate normalization, only after all input HVs have been bundled. Experiment repeated over 10 batches, standard deviation reported by bands.

As seen in figure 2.6, the standard deviation of similarity between a bundled HV and its input components decreases with dimensionality.

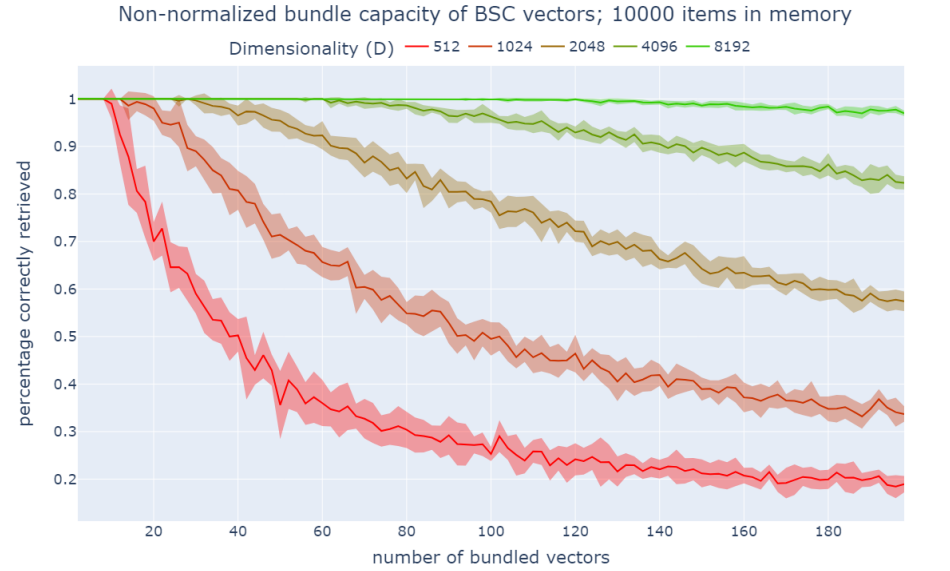


Figure 2.6: Bundle capacity of non-normalized bundle HV for several dimensionalities and bundle sizes. Capacity is referred to as the percentage of input HVs that have a greater similarity to bundle HV than all non-input HVs in item memory. Experiment repeated over 10 batches, standard deviation reported by bands.

We also theoretically confirmed these experimental observations about the bundling operation in section 4.3. We did this by quantifying the expected value and the variance of the similarity between the resulting HV s and any of its input HVs.

- A natural consequence of the resultant HV being equally similar is that the bundling

operation is commutative and associative. Another way to see this is that bundling is a component-wise sum, where summation is a commutative operation. This thus means that the order of the input vectors in the bundling operation does not change the resultant HV;

$$\mathbf{s} = \mathbf{x} + \mathbf{y} + \mathbf{z} = \mathbf{x} + \mathbf{z} + \mathbf{y} = \cdots = \mathbf{z} + \mathbf{y} + \mathbf{x}$$

Yet it is important to notice that when combined with a normalization it is no longer associative;

$$f(f(\mathbf{x} + \mathbf{y}) + \mathbf{z}) \neq f(\mathbf{x} + f(\mathbf{y} + \mathbf{z}))$$

This implies that normalization introduces some kind of 'noise'; i.e. repeatedly normalizing after bundling (we refer to this as *sequentially bundling*) leads to a fade-away effect of similarity between the bundled HV and earlier inputs. The effect of normalization after bundling on the similarity of input HVs to the bundle HV can be seen in figure 2.7. We can clearly see that normalizing has a significant impact on the similarity between input HVs bundled before the normalization compared to those bundled afterwards. When we take a closer look at the data in figure 2.7 we can conclude that the similarity between the sequentially bundled HV and its first input is non-distinguishable from a random HV in item memory after as little as 6 normalizations. As a result normalizing after a bundling operation also greatly impacts the bundling capacity because the similarity between input HVs and the bundled HV after normalization will fade away. From our experiments depicted in figure 2.8 we observe that for a dimensionality of 8192 we can successfully retrieve around 6 input HVs (averaged over 10 experiment batches) from a bundled HV of size 100, we observe the same amount for a bundle size of 200. Moreover, from our experiment data we observe that for sequentially normalized bundles with a bundle size larger than 10, between 5 and 8 input HVs can be correctly retrieved. This confirms our results in figure 2.7 where we concluded that the similarity between the first input HV and the sequentially bundled HV is non-distinguishable from a random HV in item memory after 6 normalizations, with a negligible effect of the dimensionality used.

When and if a normalization is needed should thus be taken into careful consideration when designing a VSA system. As a rule of thumb one should postpone normalizing a bundled HV for as long as possible. Whether this is achievable, is application dependent.

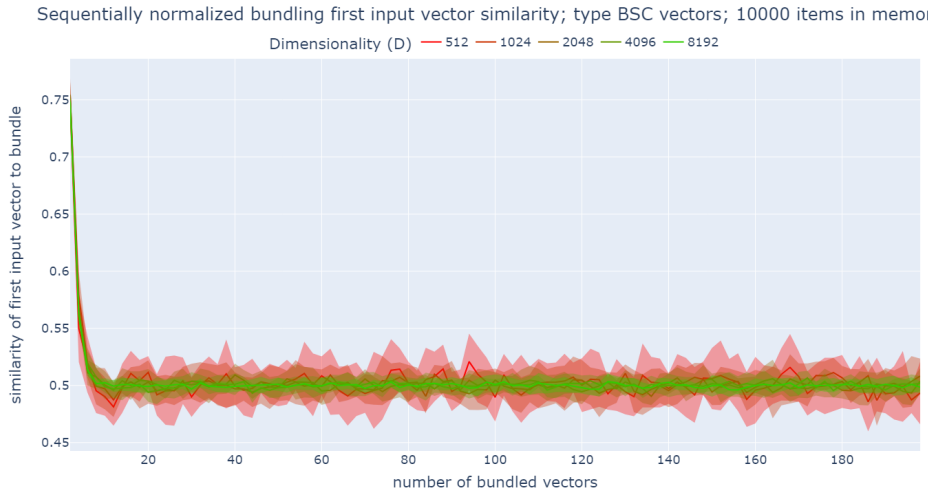


Figure 2.7: Fade-away of similarity of first input HV of bundle over growing bundle size, for several dimensionalities. Normalization applied after each new input HV is added: $f(f(...f(f(\mathbf{x_1} + \mathbf{x_2}) + \mathbf{x_3}) + ...) + \mathbf{x_n})$. We refer to this type of bundling as sequentially (normalized) bundling since we normalize the resultant HV after each bundle with a new input HV. Experiment repeated over 10 batches, standard deviation reported by bands.

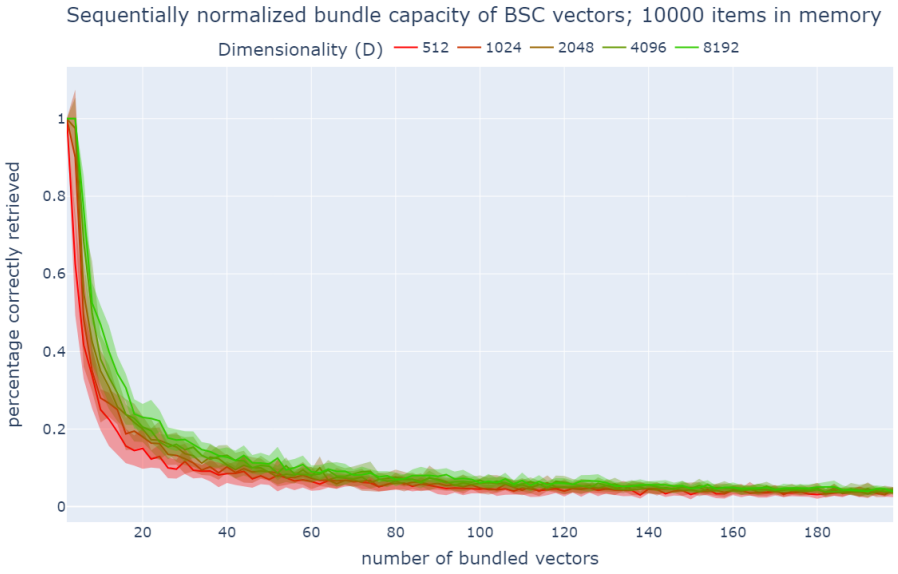


Figure 2.8: Bundle capacity of sequentially normalized bundle HV for several dimensionalities and bundle sizes. Capacity is referred to as the percentage of input HVs that have a greater similarity to bundle HV than all non-input HVs in item memory. Experiment repeated over 10 batches, standard deviation reported by bands.

Bundling thus allows us to represent complex concepts and set-like structures by combining simpler ones. For example, we can represent the concept of "a red apple" by bundling the atomic HVs for "red" and "apple", or on a larger scale, represent documents by bundling the atomic HVs of the words appearing in the document.

Binding

Due to the associative and commutative nature of the bundling operation we lose ordinal, hierarchical, and structural information. Consider for example the following setup where we try to represent a fruit basket:

$$\mathbf{fb} = (\mathbf{red} + \mathbf{apple}) + (\mathbf{yellow} + \mathbf{banana}) = (\mathbf{yellow} + \mathbf{apple}) + (\mathbf{red} + \mathbf{banana})$$

Despite the different orderings, the resulting bundled HV for the fruit basket remains the same. There is thus no method to discern the original ordering or hierarchical structure of the elements within the bundled HV. This means the bundling operation on its own cannot be used to reliably encode tuples, sequences, or other kinds of ordered and hierarchical information.

The binding operation therefore aims to create an association, typically between two HVs, which is why it is often related to creating key-value pairs. Binding is often implemented using multiplicative operations, but the exact implementation is model-dependent and thus might differ. We will denote the binding operation as follows:

$$\mathbf{r} = \mathbf{k} \bigcirc \mathbf{v}$$

The resulting HV $\mathbf{r}$ is dissimilar to its inputs, and binding can thus be thought of as a 'randomizing' operation. Yet it preserves a different kind of similarity called *structured similarity*. We illustrate this by means of an example; if we create another binding $\mathbf{r}' = \mathbf{k}' \bigcirc \mathbf{v}'$, then $\mathbf{r}$ is similar to $\mathbf{r}'$ only if $\mathbf{k}$ is similar to $\mathbf{k}'$ **and** $\mathbf{v}$ is similar to $\mathbf{v}'$. Moreover, this similarity is represented in a multiplicative fashion [5], that is the similarity of $\mathbf{r}$ to $\mathbf{r}'$ is proportional to the product of the similarities between $\mathbf{k}$ and $\mathbf{k}'$, and $\mathbf{v}$ and $\mathbf{v}'$.

For example, if the keys used in the binding operation, $\mathbf{k}$ and $\mathbf{k}'$, for creating $\mathbf{r}$ and $\mathbf{r}'$ are an exact match or they simply are same key HV, then the similarity of $\mathbf{r}$ to $\mathbf{r}'$ is proportional to the similarity between the values, $\mathbf{v}$ and $\mathbf{v}'$ only.

In the BSC model the binding operation is implemented by means of a component-wise XOR operation, as can be seen in figure 2.9a.

Dimension	1	2	3	4	...	**D**
k	1	1	0	1		0
v	0	1	0	0		1
XOR (=**r**)	1	0	0	1		1

(a) Binding of 2 HVs (**k** and **v**) with resulting HV **r**, often used to associate a key-value pair of atomic HVs.

Dimension	1	2	3	4	...	**D**
r	1	0	0	1		1
k	1	1	0	1		0
XOR (=**v**)	0	1	0	0		1

(b) Unbinding of vector **r** with atomic vector **k** (key) to obtain **v** (value).

Figure 2.9: Examples of binding and unbinding BSC HVs using XOR (self-inverse binding operation)

Going back to our fruit basket example we can now associate the fruits with their respective colors and thus retain structural information as follows:

$$\mathbf{fb} = \mathbf{red} \bigcirc \mathbf{apple} + \mathbf{yellow} \bigcirc \mathbf{banana}$$

Unbinding

Creating associations between HVs using the binding operation would not be very useful if we did not have a method to recover or identify which HVs are associated with each other. The unbinding operation is thus designed to recover one of the associated HVs from a bound representation. Its implementation is highly dependent on the implementation of the binding operation as the unbinding operation aims to 'undo' the effect of binding.

Denoting the unbind operation as $\oslash$, we can recover the value of a key-value bound pair as follows:

$$\mathbf{v} = \mathbf{k} \oslash (\mathbf{k} \bigcirc \mathbf{v})$$

In the BSC model, the unbinding operation is the same as the binding operation, making it a self-inverse operation, as can be seen in figure 2.9b. We note here that the unbinding operation, just as binding, typically distributes over bundling (additive operation) as they are multiplicative operations.

However, we run into a problem when we try to recover the yellow fruit from our fruit basket:

$$\begin{aligned}
\mathbf{banana} &\approx \mathbf{yellow} \oslash \mathbf{fb} \\
&= \mathbf{yellow} \oslash (\mathbf{red} \bigcirc \mathbf{apple} + \mathbf{yellow} \bigcirc \mathbf{banana}) \\
&= (\mathbf{yellow} \oslash (\mathbf{red} \bigcirc \mathbf{apple})) + (\mathbf{yellow} \oslash (\mathbf{yellow} \bigcirc \mathbf{banana})) \\
&= \mathit{noise} + \mathbf{banana}
\end{aligned}$$

Thus the result of unbinding a bundled HV of key-value bindings will be *similar* to the original atomic HV but not exactly the same. This phenomenon is called *crosstalk noise* and its impact depends on both the dimension **D** of the HVs and the size of the bundle. This emphasizes the need for a *clean-up or item memory.* This memory keeps the original set of atomic HVs and allows us to compare the resultant, possibly noisy, HV of an unbinding operation to them. If we relate this to our example we would find that **banana** is most similar to $\mathbf{yellow} \oslash \mathbf{fb}$ out of all HVs in item memory, $\{\mathbf{red}, \mathbf{apple}, \mathbf{yellow}, \mathbf{banana}\}$.

This probing of our item memory is called the *clean-up procedure* since it enables us to remove the noise created by combining binding with bundling and find the original bound HV. Thanks to the concentration of measure phenomenon we know that, even with the introduction of some noise, the most similar atomic HV in item memory will likely be the correct match.

How much noise is introduced by this bundling of key-value bound HVs can actually be easily understood. We take a bundle of size $\mathbf{n}$, containing key-value bound atomic HVs and defined as follows:

$$\mathbf{r} = \sum_{i=1}^{\mathbf{n}} (\mathbf{k_i} \bigcirc \mathbf{v_i})$$

As mentioned before, the key atomic HVs ($\mathbf{k_i}$'s) and value atomic HVs ($\mathbf{v_i}$'s) are, by definition, all randomly generated and thus dissimilar. We can now try to recover the value of the i^{th} bound key-value pair as follows:

$$\begin{aligned}
\mathbf{v_i} &\approx \mathbf{k_i} \oslash \mathbf{r} \\
&= \mathbf{k_i} \oslash (\sum_{j=1}^{\mathbf{n}} \mathbf{k_j} \bigcirc \mathbf{v_j}) \quad \text{(distribute)} \\
&= \sum_{j=1}^{\mathbf{n}} \mathbf{k_i} \oslash (\mathbf{k_j} \bigcirc \mathbf{v_j}) \\
&= \mathbf{k_i} \oslash (\mathbf{k_i} \bigcirc \mathbf{v_i}) + \sum_{j=1, j \neq i}^{\mathbf{n}} \mathbf{k_i} \oslash (\mathbf{k_j} \bigcirc \mathbf{v_j}) \\
&= \mathbf{v_i} + \sum^{\mathbf{n-1}} noise
\end{aligned}$$

These calculations show us that the noise, relative to the value HV, encountered while unbinding with the key HV from a bundled HV containing key-value bindings is proportional to its size $\mathbf{n}$, or more precisely $\mathbf{n} - 1$. We confirm this result by looking at figure 2.10, where we also notice the resemblance with figure 2.5. This gives us a useful, and in hindsight intuitive, result stating that the noise introduced during unbinding a bundle of key-value pairs scales with its size $\mathbf{n}$, just as the noise of a bundled HV relative to its inputs, scaled with its size $\mathbf{n}$.

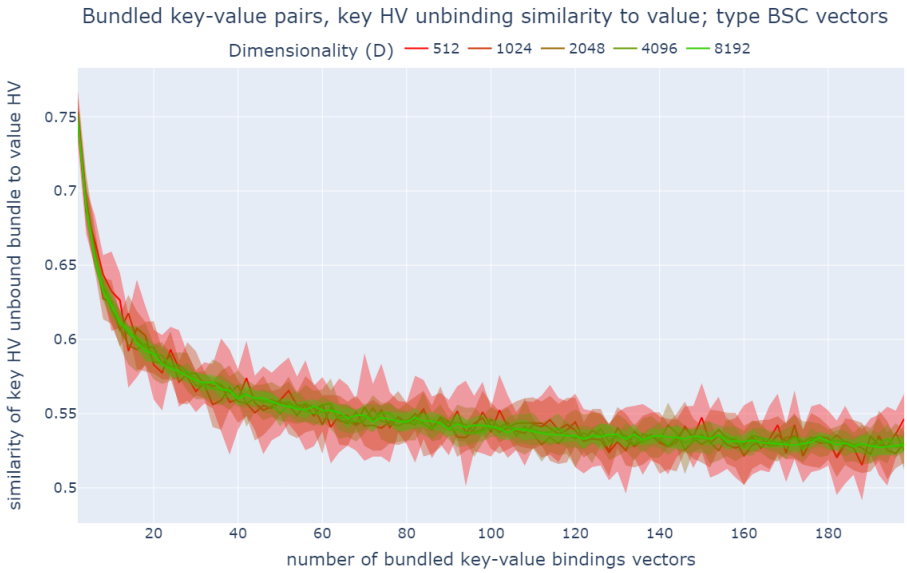


Figure 2.10: Average similarity of key-value pair bundles of several sizes to one of their input HVs, plotted for different dimensionalities. Experiment repeated over 10 batches, standard deviation reported by bands.

Permutation

VSA literature sometimes includes permutation as an additional operation. We mention it for the sake of completeness, although it is not frequently used in practical implementations.

Permutations can be used to represent ordinal information. This operation is implemented by a rearrangement of the HV components. We denote the permutation operation with ρ. The result of a permutation is an HV that is dissimilar to the input HV:

$$\mathbf{x} \not\approx \rho(\mathbf{x})$$

The rearrangement and thus also the permutation can be inverted, denoted by ρ^{-1}. It can also be repeatedly applied, denoted as ρ^i, where we apply the permutation i times. This allows the creation and decoding of sequences as follows:

$$\mathbf{seq} = \rho(\mathbf{x}) + \rho^2(\mathbf{y}) + \rho^3(\mathbf{z})$$

We can now extract the second element by applying the inverse permutation twice:

$$\mathbf{y} \approx \rho^{-2}(\mathbf{seq}) = \rho^{-1}(\mathbf{x}) + \mathbf{y} + \rho^1(\mathbf{z}) = noise + \mathbf{y} + noise$$

Permutation can for example be used to represent the order of words in a sentence as opposed to a bag-based approach only utilizing the bundling operation.

By combining these fundamental operations, VSAs can represent a wide range of symbolic, composite, and hierarchical information.

2.3 A Glimpse into VSA Models

Research into VSAs initially started from the BSC model that we introduced in the previous section. Throughout the years different models have been proposed, each relying on the same assumptions and operations, yet with different implementation details. Each VSA model uses a different representation of HVs, meaning that the distribution from which the components of the HVs are drawn and thus also their *type* type differ. These range from binary components $\sim Bernoulli(0, 1, p = 0.5)$, to bipolar $\sim Bernoulli(-1, 1, p = 0.5)$, and unitary $\sim Normal(0, 1/\mathbf{D})$ components of atomic HVs. As a result, there is also the need to implement model-specific bundling, binding, and unbinding operations which retain their respective properties mentioned in subsection 2.2.3. It is important to note that most models have similar characteristics, they all rely on the concentration of measure phenomenon and retain the properties of the fundamental operations. But there are several nuances that have to be taken into account like bundle capacity, speed of computation (complexity of operations), memory usage and available hardware. Just as with most AI applications there is a trade-off between accuracy (reliable decoding of bundling and binding) and, time- and space-complexity.

We will now briefly discuss two other models called *Multiply-Add-Permute* (MAP) [5] and *Holographic Reduced Representations* (HRR) [5].

Multiply-Add-Permute (MAP)

- HV Representation: MAP uses bipolar HVs (components drawn from -1, 1 with equal probability $\frac{1}{2}$) for its atomic representations.
- Similarity Measure: Cosine similarity
- Bundling: Element-wise addition, potentially with normalization to maintain bipolar components. The normalization is implemented using a binarization to $\{-1, 1\}$ where ties (0's) within the bundled HV are randomly but equally set to either -1 or 1.
- Binding: Element-wise multiplication. Just like BSC this is a simple, efficient and commutative operation since multiplication is commutative.
- Unbinding: Element-wise multiplication between the bound HV and one of its input HVs. Due to the nature of bipolar components, this can be a clean operation (no noise introduction if no bundling afterwards), just as the XOR for BSC. We also note that just as with BSC the binding and unbinding are self-inverse operations.

The MAP model thus shows some resemblance to BSC with respect to its 'binary' representation but varies in its operation implementations and similarity measure (cosine similarity).

Holographic Reduced Representations (HRR)

- HV Representation: HRRs use real-valued components drawn from a normal distribution with mean equal to 0 and variance equal to the inverse of the HV dimensionality $1/\mathbf{D}$.
- Similarity Measure: Cosine similarity
- Bundling: Element-wise addition, potentially with normalization to preserve the approximate unit Euclidean norm of the atomic HVs.
- Binding: Implemented by applying a circular convolution to the input HVs. This is more computationally complex than the component-wise multiplication of MAP or the component-wise XOR of BSC. Unlike in BSC or MAP the binding of two vectors (without bundling thereafter) also introduces some noise.
- Unbinding: Implemented by applying a circular correlation, a very specific operation aiming to invert the effect of a circular convolution.

This model potentially allows for an increased bundling capacity but this comes at a cost of increased computational cost and storage demand due to the increased complexity of both the operations and the components (float or double).

While the implementation details of all these models differ, we can confirm that the concentration of measure phenomenon is present for all of them by looking at figure 2.2, figure 2.3 and figure 2.11.

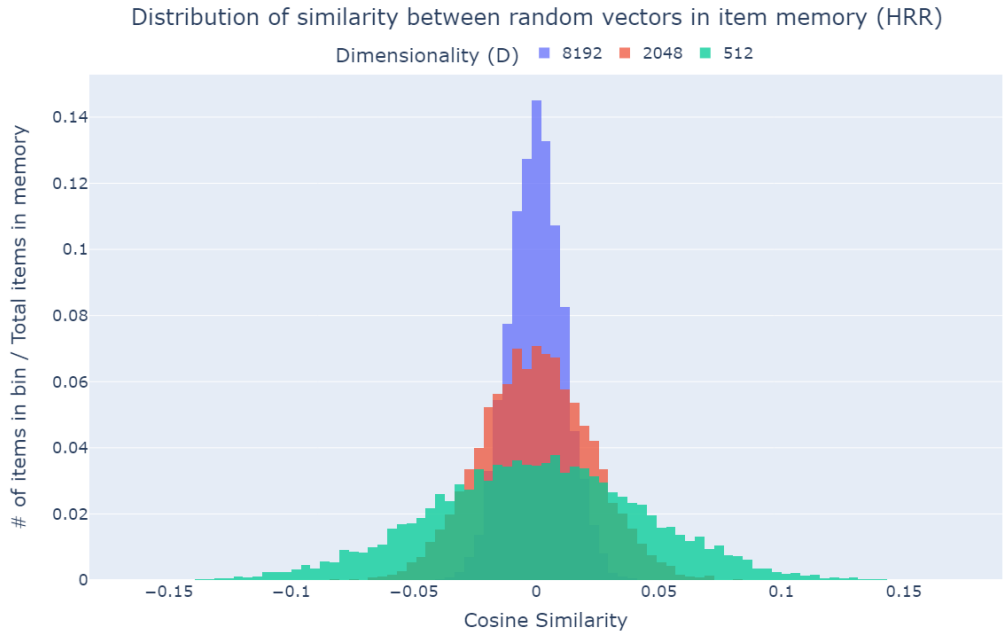


Figure 2.11: Histogram of cosine similarity between a random HRR HV and all other, randomly generated atomic HVs in item memory (10 000 in total)

We conclude that there is no fit-all model, the up- and downsides, and implementation details with their respective properties have to be weighed against the problem-specific characteristics. The interested reader can take a look at [5] for a more in-depth look at the implementation details of several models.

2.4 Conclusion

In this chapter, we have laid the groundwork for understanding Vector Symbolic Architectures. We delved into its core principles: hypervectors, similarity measures, and the fundamental operations, bundling, binding, unbinding, and potentially permutation. The unique properties of high-dimensional spaces and the carefully designed VSA models enable a rich representation scheme that supports complex symbolic structures, compositionality, and robust reasoning capabilities as confirmed by our experiments.

The insights gained in this chapter will guide our explorations into the applicability of VSAs by means of two compelling use cases:

Word Embeddings: We'll discover how VSAs can be used to create meaningful vector representations for words. While many modeling techniques for generating word embeddings exist, we will focus on only one of them called word-document frequency modeling. We will leverage text documents to create HVs carrying semantic information that allow for similarity-based comparisons, which can be used in natural language processing tasks. We will take a closer look at the benefits and the downsides of creating and using VSA-based semantic vectors, by means of comparing them against state-of-the-art word-document frequency modeling techniques by means of established evaluation benchmarks.

VSA Databases: We'll design a VSA-based database architecture, demonstrating how these high-dimensional representations can model entries in tabular databases. This experiment will showcase how VSAs can efficiently encode, store and query data while ensuring resilience to noise and ambiguity. While we don't explore it in this thesis, these representations could be directly leveraged within neural network applications, opening up exciting possibilities for integrating structured data with the power of deep learning.

Chapter 3

Word Embeddings

Word Embeddings lay at the foundation of any *Natural Language Processing (NLP)* application, providing a way to represent the *semantic* meaning of words in a vector format. By capturing complex semantic relationships between words in a language these vector representations enable further processing and computation of text-based data by NNs.

The foundation of word embeddings lies in the *statistical semantics hypothesis* [14], which states that the meaning of linguistic units (sentences or documents) can be inferred from patterns of textual co-occurrence. Stated more simply, if units of text have similar words in them they tend to have similar meanings.

From this, two more specific hypotheses evolved that have driven word embedding development:

- *Bag-of-words hypothesis*: This hypothesis evolved from research surrounding document retrieval. It states that word frequencies within documents directly relate to a document's meaning [15].
- *Distributional hypothesis*: Words appearing in similar contexts tend to have similar meanings [16]. This hypothesis can be seen as a word-level interpretation of the statistical semantics hypothesis.

Historically two primary methods of learning word embedding have emerged from these hypotheses, namely:

- *Count-based methods*: These methods, of which *Latent Semantic Analysis (LSA)* is the prime example, are based on the Bag-of-words hypothesis. They rely on constructing term-document matrices to analyze co-occurrence frequencies. From these matrices, dimensionality reduction techniques, like (truncated) *Singular Value Decomposition (SVD)*, are used to generate 'dense' word representations.
- *Prediction-based methods*: More recent approaches focus on the distributional hypothesis and use context windows around the target word to measure co-occurrence. Two prime examples come from the Word2Vec paper [17]. They utilize shallow NNs in two key architectures:
  - *Continuous Bag-of-Words (CBOW)*, which predicts a target word based on surrounding context words.
  - *Skip-gram*, which predicts surrounding context based on a central word.

  Both of these methods are trained using backpropagation for their respective tasks, after which the learned parameters of the NN are extracted as the word embeddings.

Later models, like GloVe [18] combine global co-occurrence statistics from a corpus with the local context principles. More recent encoder-based transformer models such as BERT [19] use a different approach by generating dynamic context-dependent word embeddings based on input sentences.

In chapter 2 we have seen that VSAs offer an alternative paradigm for representing and manipulating information, making it a viable option to explore for generating vector-based representations of words. In the remainder of this chapter we will explore the applicability of this alternative paradigm to the creation of meaningful word embeddings. We will start off by introducing a novel count-based VSA approach to learning word embeddings. Afterwards, we will introduce a linguistic dataset that is suitable for our count-based learning method. Next we will take a look at our training process in more detail and refine it to optimize the running time and efficiency. Finally we will introduce the WordSim353 [20] validation dataset to evaluate our learned word embeddings and compare it to LSA, a state-of-the art count-based learning method,

3.1 VSA-based Word Embeddings

We begin with a general outline of the learning process using a document-based textual dataset.

Representing Words and Documents
As with any VSA-based application, we need to define a set of core, symbolic elements represented by atomic vectors upon which we can build and generate more intricate composite representations.

We will initialize the document and word HVs as follows:

- **Document HVs:** Each document in our corpus is assigned a unique, randomly initialized atomic HV. The HVs of the documents serve as the core elements of our problem, and can be seen as a way to uniquely identify and represent the documents of our corpus.
- **Word HVs:** These are the HVs we would like to learn, thus each word in our vocabulary will initially be represented by an 'empty' HV. An empty HV, as the name suggests, is a HV that does not contain any information, the exact initialization depends on the VSA model used. Conceptually we can think of empty HVs as hypervectors representing empty sets. More specifically, when an initially empty HV is bundled with a random HV $\mathbf{x}$, the resulting bundled HV is exactly $\mathbf{x}$.

These initial representations lay the foundation for our next step: encoding the semantic relationships, present in our corpus, between the words in our vocabulary.

Encoding Semantic Similarity by Bundling
Our VSA-based learning method focuses on word-document co-occurrence to extract the semantic relationships between words. For each word within a document, we use the bundling operation to add the document's unique HV to the word's HV. Through this process, words that frequently appear in similar documents will accumulate similar learned bundle HVs.

Psuedo-code of the algorithm we just described can be found in Algorithm 1.

We can give an intuitive view of our learning algorithm as follows: We start off by generating random, dissimilar document HVs in our high-dimensional space. The word HVs are initially empty, sitting at the origin. With each bundle of a document HV to a word HV appearing in the respective document, we essentially move the HV for that word closer to the HV of that document in our high dimensional space. We repeat this for each document in our corpus, essentially moving the word embeddings to the 'center' of all the document embeddings they occur in. This repeated bundling of the document HVs thus moves HVs of words frequently occurring together in documents to similar positions in the high-dimensional space.

Algorithm 1 VSA Word Embedding Learning

Require: Text corpus $\mathcal{C}$, vocabulary $\mathcal{V}$, dimensionality of HVs $\mathbf{D}$, VSA model $\mathcal{M}$
  Initialization:
  for each document d $\in \mathcal{C}$ **do**
    Generate a random atomic HV $\mathbf{hv_d}$ of dimension $\mathbf{D}$ according to $\mathcal{M}$
  end for
  for each word w $\in \mathcal{V}$ **do**
    Initialize empty HV $\mathbf{e_w}$ of dimension $\mathbf{D}$ according to $\mathcal{M}$
  end for

  Corpus Traversal:
  for each document d $\in \mathcal{C}$ **do**
    for each word w $\in$ d **do**
      $\mathbf{e_w} \leftarrow \mathbf{e_w} + \mathbf{hv_d}$ ▷ Bundle document HV with word HV
    end for
  end for

  Normalization (Optional):
  for each word w $\in \mathcal{V}$ **do**
    $\mathbf{e_w} \leftarrow f(\mathbf{e_w})$ ▷ Model specific normalization
  end for

3.2 Experiments

The primary goal of our experiment is to evaluate the effectiveness of our VSA-based learning method in capturing meaningful semantic relationships between words that appear in similar documents. We will do so by investigating the following questions:

- **Initial observations:** Is our VSA-based learning method capable of extracting useful information? We will take a look at the learned embeddings and their similarities to each other to validate that our method indeed extracts useful semantic information from the training data.
- **Similarity to Human Judgment:** How closely do the similarity scores derived from our VSA-based embeddings align with human perceptions of word relatedness? For this, we will use the WordSim353 dataset.
- **Comparison to Traditional Methods:** How does the performance of our VSA-based approach compare to established count-based techniques like LSA? Does it offer any advantages in terms of representation quality or computational aspects?

The evaluation method as well as other observations are of course highly sensitive to the training dataset used and thus difficult to evaluate directly, that is why we will also compare our VSA-based results to the evaluation of LSA on the same dataset to get more tangible results.

3.2.1 Dataset

For our VSA-based word embedding method, we require a high-quality document dataset over a large domain of general text. This will ensure that we have a sufficient vocabulary size and a diverse range of contexts in which we can observe word co-occurrences. This is why we selected the WikiText [1] language modeling dataset. The WikiText dataset is a collection of documents extracted from a set of good and featured articles on Wikipedia. These articles meet a core set of editorial standards, are well written and contain factually accurate and verifiable information, while also being broad in coverage [2]. This dataset comes in 2 variants, the WikiText-2 dataset

[1] https://blog.salesforceairesearch.com/the-wikitext-long-term-dependency-language-modeling-dataset
[2] https://en.wikipedia.org/wiki/Wikipedia:Good_articles

	Wikitext-2			Wikitext-103		
	Train	Valid	Test	Train	Valid	Test
Articles	600	60	60	28,475	60	60
Tokens	2,088,628	217,646	245,569	103,227,021	217,646	245,569
Vocab	33,278			267,735		
OoV	2.6%			0.4%		

Table 3.1: Statistics for Wikitext-2 and WikiText-103 dataset

and the WikiText-103 dataset. Their respective sizes and statistics can be found in table 3.1.

For the purpose of this initial investigation into the potential of VSAs for learning document-based word embeddings, we decided to utilize a random sample of 10 000 documents from the training set of documents from the WikiText-103 dataset. This allows us to validate the feasibility of our method, asses its potential, and benchmark it against LSA. The reason we decided to not opt for the smaller WikiText-2 dataset is that it does not allow for a significant dimensionality reduction and thus does not allow us to properly investigate and compare the capabilities of both learning methods. The random subset we use offers a suitable balance between vocabulary size and computational feasibility for our introductory exploration of VSAs in word embedding generation. The decision to use this random subset over the entire dataset is also based on the fact that it allows us to learn our embeddings locally on a GPU (NVIDIA 3070 with 16GB memory).

Preprocessing

The following preprocessing steps were performed on the WikiText-103 dataset to prepare it for our VSA-based learning process:

1. **Tokenization:** Text is split into sentences and then into individual words (tokens).
2. **Title Handling:** Titles are identified, processed, and used to identify separate documents within the dataset. This is needed because the dataset is given in one large compressed file, and not on a per-document basis.
3. **Lowercasing**: All text is converted to lowercase to eliminate the impact of case variations.
4. **Stop Word Removal:** Common English stop words (e.g., "the", "a") and punctuation are removed, as they carry minimal semantic meaning.
5. **Filtering:** We ensure that:
    - Numeric tokens are replaced with a generic "<num>" tag to represent numerical entities collectively.
    - Very short words (less than two characters) are removed.
    - Words appearing in less than 3 documents are removed as they do not provide sufficient semantic information and allow for a reasonable vocabulary size, relative to our document count of 10 000.

From this preprocessed dataset we randomly sampled 10 000 documents, resulting in a vocabulary of 91 613 words. These preprocessing steps aim to reduce noise, standardize the text format, and focus the learning process on semantically relevant words within the corpus. It also delivers separate documents that can be used to form a term-document matrix, simplifying the learning process for both the VSA-based learning method and LSA.

3.2.2 Training

In this section, we will take a closer look at the earlier mentioned learning procedure (Algorithm 1), and optimize it by proposing a GPU-based learning process.

Thereafter, we will take a look at the hyperparameters of our learning method, namely the HV dimensionality and the VSA model used.

Optimizing our learning algorithm
First, we will look at a small adjustment to Algorithm 1, allowing us to utilize vector or tensor operations on a GPU to dramatically increase the running time of our algorithm. The updated version can be found in Algorithm 2. The first difference with Algorithm 1 is in the initialization step, where we now create 3 matrices, a binary term-document matrix, a document matrix containing the atomic document HVs, and the term matrix containing empty HVs used for learning.

The main differences with Algorithm 1 can be found in the actual learning procedure or corpus traversal. It now utilizes the term-document matrix $\mathbf{M_{td}}$ to quickly identify the words that appear in the current document by getting the non-zero indices of the respective document row in the term-document matrix. Notice we store the term document matrix where the rows contain the document vectors and the columns represent the terms in the vocabulary. This allows us to efficiently access the document vector containing the word occurrences for that document during the learning process.

We also easily extract the atomic HV for our current document from the document matrix $\mathbf{M_d}$ using its index. At this point we have the document HV and the indices of the words in our vocabulary that appear in that respective document. All that is left to do now is to update the relevant word embeddings (HVs), indicated by the indices we just extracted, by bundling the current document HV to them.

If we assume that the term matrix $\mathbf{M_t}$ containing the word HVs that we are learning fits in our GPU memory, we can increase the speed of the training process significantly. We do this by bundling the current document HV into the word HVs of all relevant words in our term matrix $\mathbf{M_t}$ using a single GPU-based matrix addition operation.

We achieve this by leveraging broadcasting [3], a mechanism that allows us to perform operations on tensors of different shapes without explicitly reshaping them.

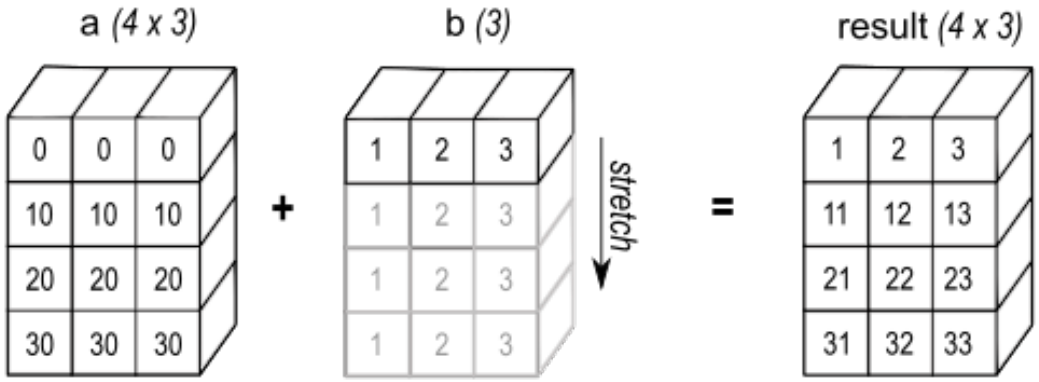


Figure 3.1: Example of a broadcasted vector to matrix addition, extending the addition of a 1x3 vector to a 4x3 matrix. [4]

[3] https://pytorch.org/docs/stable/notes/broadcasting.html
[4] https://numpy.org/doc/1.25/user/basics.broadcasting.html

In our context, broadcasting allows us to bundle the document HV (a vector) to the selected rows of the term matrix in one GPU-based matrix addition operation. Remember that the bundling operation for all VSA models is an additive operation, and thus also for the MAP model we use. A simple visualization of a broadcasted vector to matrix addition can be seen in figure 3.1.

By using this matrix-based update of our word embeddings we eliminate the need for an inner loop over the words and leverage highly parallelized computations using GPUs. This effectively transforms our VSA-based learning process from a multi-minute process to a process that takes only a handful of seconds, running time for the optimized algorithm is reported later in table 3.4.

Algorithm 2 GPU-based VSA Word Embedding Learning

Require: Text corpus $\mathcal{C}$, vocabulary $\mathcal{V}$, dimensionality of HVs $\mathbf{D}$, VSA model $\mathcal{M}$

Initialization:

$|\mathcal{C}| \leftarrow$ number of documents in our corpus

$|\mathcal{V}| \leftarrow$ size of our vocabulary

$\mathbf{M_{td}} \leftarrow$ binary term-document matrix of size $|\mathcal{C}| \times |\mathcal{V}|$

$\mathbf{M_d} \leftarrow$ Document HV matrix of dimension $|\mathcal{C}| \times \mathbf{D}$, rows represent random atomic HVs of dimension $\mathbf{D}$ according to $\mathcal{M}$.

$\mathbf{M_t} \leftarrow$ Word HV matrix of dimension $|\mathcal{V}| \times \mathbf{D}$, rows represent empty HVs of dimension $\mathbf{D}$ according to $\mathcal{M}$. This matrix is stored on the GPU.

Corpus Traversal:

for each document $\mathrm{d} \in 1, ..., |\mathcal{C}|$ **do**

  $\mathbf{I_d} \leftarrow$ non-zero indices in $\mathbf{M_{td}}[\mathrm{d}]$ ▷ Get relevant indices

  $\mathbf{hv_d} \leftarrow \mathbf{M_d}[\mathrm{d}]$ ▷ Extract document HV from document matrix

  $\mathbf{M_t}[\mathbf{I_d}] \leftarrow \mathbf{M_t}[\mathbf{I_d}] + \mathbf{hv_d}$ ▷ Update relevant word embeddings using matrix addition

end for

Normalization (Optional):

for each word $\mathrm{w} \in 1, ..., |\mathcal{V}|$ **do**

  $\mathbf{M_t}[\mathrm{w}] \leftarrow f(\mathbf{M_t}[\mathrm{w}])$ ▷ Model specific normalization

end for

We now have a highly optimized VSA learning algorithm where the computational complexity grows linearly with the number of documents in the corpus, assuming the word HV matrix fits into GPU memory. Even if this is not the case, we could partition our word matrix and thus the vocabulary over several GPUs and let each GPU loop over the entirety of the corpus. After which we can combine the partitions since the learning process of each word embedding does not have cross dependencies on the learning of other word embeddings.

Initializing our hyperparameters

Now that we have an optimized, final version of our learning algorithm we can start initializing its hyperparameters.

First, we notice that at the core of our learning algorithm lies the bundling operation. It is the only VSA operation we use during corpus traversal. Thus there is no need for a normalization of the bundled HV after each step. As we saw in subsection 2.2.3 this non-normalizing way of bundling allows for a drastically improved bundling capacity and thus allows us to capture the semantics of more documents inside a single learned word embedding. If at a later point in time, we would want to create associations between learned embeddings using a binding operation, we could do so by normalizing all HVs when the learning algorithm finishes. If we have these normalized HVs we can safely apply a binding operation to associate word HV pairs.

The next step in implementing Algorithm 2 is to pick a suitable VSA model. If we take the

previous paragraph into consideration, we need a model that allows for a straightforward non-normalized bundling operation that can be applied while we loop over all the documents in the training data. A common first choice is the BSC model due to its simplicity, speed of the binary operations, and memory efficiency of the binary components. However, we run into a problem when we look at its inability to natively support the non-normalized bundling operation. If we want to allow non-normalized bundling for BSC HVs we will need to keep track of the amount of document HVs that will be added to the learned embedding for each word. Otherwise, we can not apply a normalization afterwards because the normalization for the BSC model is a thresholded sum, where the threshold is set to the number of input HVs of the bundle divided by 2. We can solve this by storing the number of documents that each word appears in separately, or we could look at another model for which the normalization step does not need any additional information and thus reduces the overhead. The alternative we chose is the MAP model which we discussed in section 2.3. It resembles BSC in many ways, the main difference is that MAP uses randomly selected components from {-1,1} as opposed to the set {0,1} used in BSC. As stated in section 2.3 the normalization consists of a 'binarization' to {-1,1} for all the components. This avoids the need to store the occurrence counts for each word since we do not need to perform a thresholded sum but a simple binarization to {-1,1} for the respective negative and positive components. Apart from the changes in the implementation details of the similarity measure and the operations, we won't see any changes in performance as MAP offers similar characteristics and capabilities to BSC.

Lastly, we need to select a dimensionality for our document and word HVs. There are no concrete guidelines provided in previous works, yet it is a common practice to pick a dimensionality in the early thousands, where 10 000 is the most commonly selected dimensionality [21]. For our experiment, we could consider the representational capacity of our HVs to determine a suitable dimensionality. Since we solely use a non-normalized bundling operation, we can take a look at the non-normalized bundling capacity depicted in figure 3.2. Here we notice great similarity to figure 2.6, representing the non-normalized bundling capacity of the BSC model.

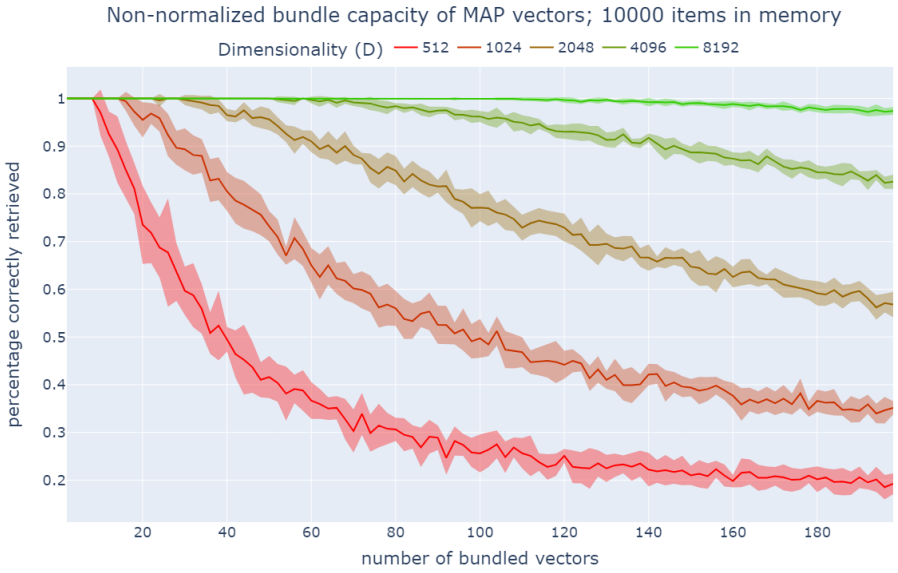


Figure 3.2: Bundle capacity of MAP model, non-normalized, bundle HV for several dimensionalities and bundle sizes. Experiment repeated over 10 batches, standard deviation reported by bands.

Next, we take a look at figure 3.3, representing a histogram created by taking the document count for each word in our vocabulary. This allows us to estimate the representational capacity that our word HVs would need. We decided that a representational capacity of around 100 bundled

HVs would suffice to capture document co-occurrence inside our word embedding HVs.

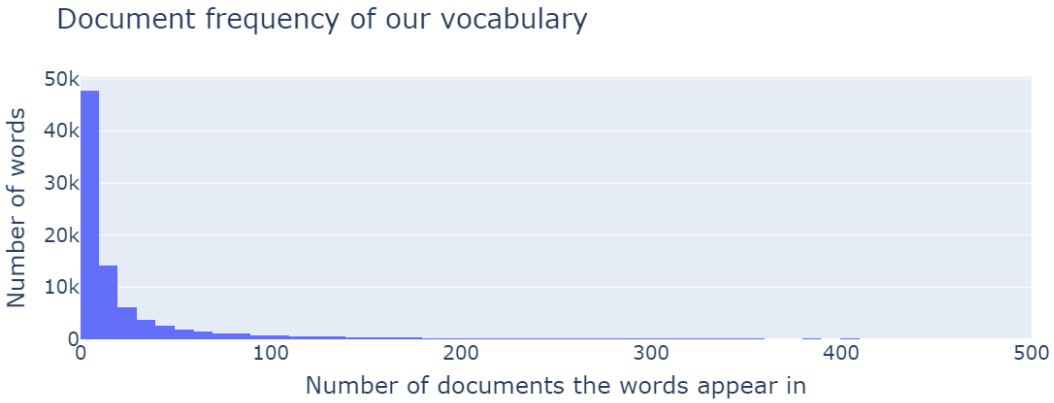


Figure 3.3: Histogram depicting the document frequency for the words in our dataset. This histogram was created by counting the number of documents each word in our vocabulary appears in, and then binning the document frequencies per 10. We thus observe that there are 47 000 words in our vocabulary appearing in at most 9 documents (first bin), while only 126 words of our vocabulary appear in 300 to 309 documents.

If we combine the information found in Figures 3.2 and 3.3, we conclude that a dimensionality of 5000 for our MAP model HVs should suffice to capture the general signal of document co-occurrence for our HV word embeddings,

We also mention that we do apply the optional normalization in Algorithm 2 to maintain the integrity of our HVs, and thus report results based on true MAP HVs, having components from {-1,1} which can thus also be used to apply other VSA operations on.

LSA reference word embeddings

For the training of our reference word embeddings from LSA, we remain brief. We employed a *truncated SVD* algorithm on the same binary term-document matrix we used for our VSA-based algorithm, reducing (or truncating) the dimensionality to 2000.

The truncated SVD algorithm aims to approximate our original term-document matrix $\mathbf{M_{td}}$ by retaining only the r most significant singular values and their corresponding singular vectors [22]. This approximation is achieved by decomposing the original matrix into three separate matrices: a term-concept matrix $\mathbf{U}$, a diagonal matrix containing the top r singular values $\mathbf{\Sigma}$, and a concept-document matrix $\mathbf{V}$. The product of these matrices forms an approximation of the original term-document matrix:

$$\mathbf{M_{td}} \approx \mathbf{U}_r \mathbf{\Sigma}_r \mathbf{V}_r^T$$

Where $\mathbf{U}_r$ is an m by r matrix, $\mathbf{\Sigma}_r$ is an r by r matrix, and $\mathbf{V}$ is an n by r matrix. For our example this would mean that m is equal to the number of documents in our corpus (10 000), n is equal to the vocabulary size (91 613) and r is equal to the reduced dimensionality. From this decomposition we can obtain our dimensionality-reduced word embeddings by the following matrix multiplication: $\mathbf{V}_r \mathbf{\Sigma}_r$. This approximation aims to retain the most salient semantic information captured by the original term-document matrix while reducing the dimensionality.

Next, we briefly discuss our reasoning behind a lower dimensionality of 2000 for our LSA-based embeddings as opposed to our VSA word embeddings. LSA operates through matrix factorization, aiming to closely resemble the original term-document matrix by means of a lower-rank approximation, facilitated by Truncated SVD. It identifies the most important *latent* features in the term-document matrix and approximates the original matrix using these features, hence the name Latent Semantic Analysis. We also note that the matrix factorization results in real-valued vector components, these real-valued components offer more representational capacity as opposed to simple binary {-1,1} components used in MAP. Moreover, the physical storage space

needed for these real-valued components is much greater than that of our binary components, and thus it only seems fair to reduce the dimensionality of the LSA-based embeddings.

3.2.3 Initial Observations

Before we dive into a formal evaluation using WordSim353, we will explore some qualitative insights about the word embeddings learned by our VSA-based method.

First, we take a closer look at the nearest neighbors of a target word and verify that they are semantically related. This helps us get a general sense of whether the embeddings appear to be capturing meaningful semantic relationships.

For example, if we look at the 15 most similar words to 'movie' in figure 3.4, we observe many, if not all, words seem to be meaningfully related. Some of the words that are closely related to 'movie' according to our VSA learned embeddings are: 'film', 'televisions', 'actor', 'producer', and 'script'. These are promising results, and they validate that our learning method is able to extract the semantic relationships for words that co-occur inside documents.

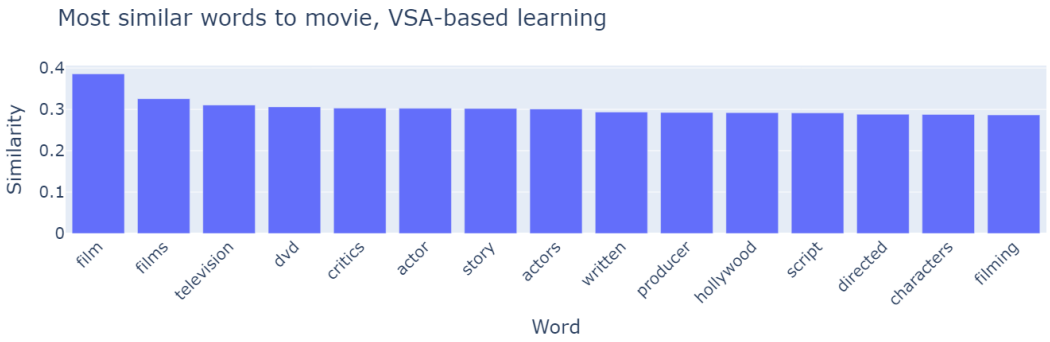


Figure 3.4: Plot of 15 most similar words to 'movie' for the VSA-based learned embeddings.

We can also gain valuable insights by comparing our VSA word embeddings to the ones learned by LSA. We again display the 15 most similar words to 'movie' in figure 3.5, but now for our LSA embeddings.

Figure 3.5: Plot of 15 most similar words to 'movie' for the LSA word embeddings.

Here we observe considerable overlap with the 15 most similar words to 'movie' for our VSA embeddings. We also observe that the ordering is different and also the set of similar words is not identical. Since both methods learn from exactly the same corpus we conclude that while they generate word embeddings with similar semantic meaning, there is a discrepancy between them, and they thus do not extract exactly the same semantic relationships from a corpus of documents.

3.2.4 Evaluation

Now that we have verified the validity of our VSA-based learned word embeddings, we look into how we can assess their quality. We will introduce the WordSim353 dataset, a standard benchmark for evaluating word embeddings, and outline the evaluation methodology.

The WordSim353 Evaluation Dataset
The WordSim353 dataset serves as an established tool for measuring the semantic similarity or relatedness between word pairs. It comprises 353 English noun pairs annotated by human subjects with a score, ranging from 0 to 10, reflecting how similar or related the words are in meaning.

It's important to acknowledge that the quality of any word embedding method is inherently tied to the training data. Ideally, we'd have access to vast amounts of text data from diverse sources, similar to the training of large language models. This would allow the word embeddings to capture a broader range of semantic relationships.

While our training data is of decent quality, it is still 'limited' in size if we compare it to the number of corpora and thus also data that LLMs like ChatGPT are trained on. We may thus not expect state-of-the-art performance on WordSim353. However, we can still gain valuable insights by comparing our model's evaluation score with a baseline from an established method like LSA, trained on the same dataset.

In essence, using LSA as a benchmark helps us understand how well our VSA method captures semantic relationships, even if the absolute score on WordSim353 might not be directly comparable to scores achieved with massive training datasets.

In table 3.2 we report the cosine similarities for 10 word pairs for VSA-based and LSA-learned embeddings together with the human-assigned relatedness scores. To assess our learning methods we also mention the cosine similarities for these word pairs from our original term-document matrix, without any dimensionality reduction or learning applied to them.

Word pair	VSA	LSA	Term-document matrix	Human-assigned
computer - keyboard	0.092	0.194	0.122	7.62
planet - galaxy	0.182	0.447	0.262	8.11
canyon - landscape	0.054	0.170	0.086	7.53
day - summer	0.298	0.485	0.458	3.94
day - dawn	0.144	0.281	0.210	7.53
country - citizen	0.116	0.262	0.193	7.31
planet - people	0.139	0.241	0.190	5.75
environment - ecology	0.096	0.237	0.141	8.81
money - bank	0.167	0.324	0.263	8.5
computer - software	0.297	0.576	0.409	8.5
Pearson correlation	0.284	0.366	0.284	

Table 3.2: First 10 word pairs of the WordSim353 dataset with their respective cosine similarity for the learned VSA embeddings, LSA embeddings, and the embeddings in the original term-document matrix. We also report the human-assigned scores from the WordSim353 validation dataset, ranging from 0 to 10. The final row denotes the Pearson correlation between the learned similarities and the human-assigned similarity scores for all the WordSim353 word pairs for each embedding method, including the non-reduced embeddings of our original term-document matrix.

For all of these word embeddings we also denote the Pearson correlation between the cosine similarities of respective word-pair embeddings and the human-assigned similarity scores. This

allows us to measure the overall performance of the resulting embeddings of a learning method. By using the correlation we essentially look if low (high) cosine similarities for learned embeddings of word pairs correlate with a lower (higher) human assigned similarity scores for the same word pairs, independent of the scale used for each scoring.

3.2.5 Results

If we look at the individual similarity scores of the VSA and LSA embeddings we observe that they are rather different. If we compare them to the original embeddings present in our term-document matrix we observe a striking similarity, although on a different scale, between the VSA embeddings and the original term-document embeddings.

If we then compare the Pearson correlation between the VSA/LSA embeddings and the human-assigned scores we observe that the correlation score of our VSA-based embeddings is lower at 0.284 compared to the LSA correlation of 0.366. We also observe that the correlation for the embeddings in the original term-document matrix is exactly the same as the one for our VSA embeddings, validating the striking similarities we observe for the individual scores.

Reflecting on our observations
We start by commenting on the striking similarity between the semantic information that is captured in the VSA embeddings and the original term-document matrix. This similarity indicates that our VSA learning method succeeds at capturing the semantic information present in the original term-document matrix. To validate this observation we calculate the correlation of the cosine similarities of the word pairs in the WordSim353 dataset between the VSA embeddings and the original term-document matrix. This correlation is 0.982 and thus validates that our VSA embeddings accurately capture the semantic relationship that is present in the original (binary) term-document matrix.

A natural follow-up question is: why do the LSA embeddings have a higher Pearson correlation of 0.366 when the correlation in our original data is only 0.284? One crucial aspect is the dimensionality reduction algorithm employed to calculate the LSA embeddings.

The truncated SVD decomposition, which performs the dimensionality reduction of LSA, has the effect of preserving the most important semantic information in the text while reducing noise and other undesirable artifacts of the original space represented by our term-document matrix. By utilizing 2000 dimensions, our LSA embeddings thus potentially eliminate irrelevant noise captured by the complete term-document matrix, focusing on the most salient latent features. By focusing on these latent features, LSA can mitigate the impact of noise and irrelevant information present in the raw term-document matrix, possibly leading to an enhanced ability to capture meaningful semantic relationships.

This gives us insight into one of the key differences between the VSA-based learning method and the LSA learning method. That is, the VSA-based method essentially moves the HVs of words that co-occur in documents closer to each other during learning since the same atomic document HVs will be bundled to each of them. On the other hand, the LSA method focuses on dimensionality reduction of the term-document matrix, which aims to preserve the relationships between terms and documents by capturing the most significant latent features. This approach does not explicitly manipulate the vectors of individual words based on their co-occurrence patterns within documents but rather seeks to capture the broader semantic structure of the text corpus.

We also note that the effectiveness of dimensionality reduction employed by LSA for improving correlation scores is highly dependent on the evaluation method or dataset used. In some cases, reducing dimensions to lower levels might indeed result in better scores by filtering out noise and focusing on essential semantic features. However, the optimal dimensionality for achieving the highest correlation may vary depending on the specific characteristics of the dataset and the evaluation criteria employed. This idea is validated when we look at figure 3.6, here we observe that lower dimensionalities can effectively filter out the noise. We also see the effect

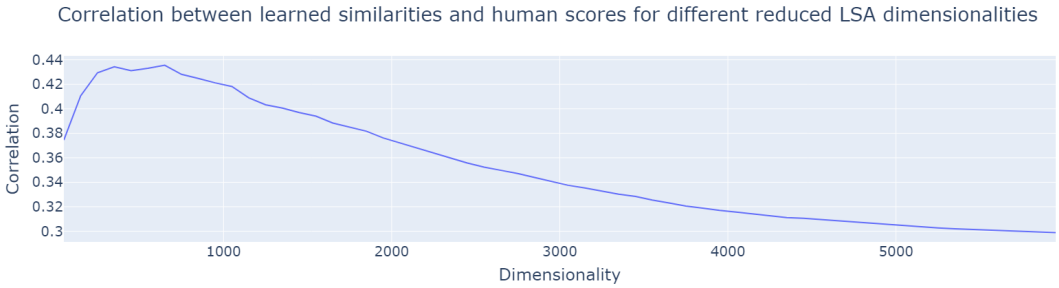


Figure 3.6: Plot of the Pearson correlation between the word pair similarities of our learned LSA embeddings and the human-assigned word pair similarities of our validation dataset, against the dimensionality we reduced our learned embeddings to.

that the reduced or truncated dimensionality **r** has on the quality of our embeddings. First, we observe an optimal dimensionality that seems to lie between 200 and 800, giving us the highest correlation between the similarities of our learned embeddings on the word pairs of the validation dataset and the human-assigned similarity scores for these word pairs. If the dimensionality we reduce to is too small, 200 or less, we discard too much information, leading to a significant drop in the correlation scores. Conversely, if the dimensionality is too high, we start to include more noise and irrelevant details, which also degrades the performance. Lastly, we observe that as the dimensionality increases to 5000 or above, we near the correlation of 0.284 obtained by the embeddings of our original term-document matrix. For these higher dimensionalities, where we do less reduction, we thus obtain the similarity information present in our original term-document matrix.

3.3 Conclusion

From our preliminary experiment we conclude that VSA-based learning methods could offer a viable alternative to LSA.

While LSA focuses on preserving latent semantic information through dimensionality reduction, our VSA-based method emphasizes capturing the exact semantic relationships present in the original term-document matrix by directly encoding contextual (co-occurrence) patterns. We verified that our current learning algorithm effectively preserves this original co-occurrence pattern present in the binary term-document matrix. But we also observed that the co-occurrence relation present in our binary term-document matrix might not be an optimal representation of the semantic relationships between words in our corpus. Because our VSA-based learning algorithm aims to capture this semantic relation present in the original term-document matrix, we know that our learned VSA embeddings can only be as good at capturing a semantic relation as the one that is present in the term-document matrix we learn from.

To confirm this hypothesis we conducted an additional experiment on the same corpus of documents but now with *term frequency – inverse document frequency (TF-IDF)* vectorization of our data for our term-document matrix. The dimensionalities of our term-document matrix remain the same but the values within the matrix now reflect the TF-IDF scores rather than simple binary occurrences. These TF-IDF values aim to better capture the importance of terms within the documents and across the entire corpus, leading to improved semantic embeddings present in our original term-document matrix.

We only give a brief description of the VSA-based learning algorithm that we applied to the TF-IDF matrix. For this experiment we no longer have the binary occurrence data, but instead have TF-IDF scores for a document, where a higher value for a word in our document indicates greater importance of a word within this document relative to its occurrence across the entire corpus. To capture this relation we apply a scaling operation of the values inside the TF-IDF vector

Word pair	VSA	LSA	Term-document matrix	Human-assigned
computer - keyboard	0.03	0.145	0.068	7.62
planet - galaxy	0.023	0.066	0.056	8.11
canyon - landscape	0.001	0.039	0.02	7.53
day - summer	0.084	0.169	0.128	3.94
day - dawn	0.044	0.133	0.071	7.53
country - citizen	0.058	0.139	0.073	7.31
planet - people	0.019	0.023	0.023	5.75
environment - ecology	0.066	0.219	0.110	8.81
money - bank	0.123	0.295	0.185	8.5
computer - software	0.178	0.407	0.288	8.5
Pearson correlation	0.345	0.374	0.389	

Table 3.3: Same information as in table 3.2, now reported for the experiment where we learn our embeddings from a TF-IDF vectorized term-document matrix. The final row again denotes the Pearson correlation between the learned similarities and the human-assigned similarity scores for each embedding method.

for a document, where we scale the TF-IDF importance values to natural numbers between 0 and some predefined maximum value denoted as **m**. Next, we bundle the documents HV to the respective word embeddings as many times as our scaled importance value indicates. For our experiment we set the maximum scaling value **m** to 10.

When a word is indicated to be important for a document by having a high TF-IDF score, the scaled value of this word would be near our maximal scaling value **m**. This would lead to the document HV getting bundled into the word HV many times. This way we ensure that the importance of words within documents is appropriately reflected in the embedding updates.

Conceptually this learning method ensures that words that have high importance, according to the TF-IDF vectorization, in similar documents have similar HV word embeddings because the same document HVs will get bundled with these words many times.

If we then look at the results of our new experiment in table 3.3 we notice the following:

- The Pearson correlation for the LSA embeddings does not differ greatly from our original experiment where we used a simple, binary occurrence, term-document matrix. From this, we conclude that the LSA method is relatively robust to the type of term-document matrix used, whether it is binary or TF-IDF weighted. This consistency suggests that LSA's ability to capture latent semantic structures is not heavily influenced by the weighting of the term-document matrix, as long as the general co-occurrence patterns are preserved.
- The Pearson correlation for the VSA embeddings, however, shows a noticeable improvement, from 0.284 to 0.345 when using the TF-IDF weighted term-document matrix to learn our VSA-based embeddings compared to the binary occurrence matrix. Furthermore, we again notice the striking similarity between the word pair similarity scores of the VSA embeddings and the embeddings in our original (TF-IDF) term-document matrix. We solidify this observation by noting a correlation of 0.953 between the word-pair similarities of our learned VSA embeddings and the original non-reduced embeddings.

These observations confirm our hypothesis that the quality of the semantic relationships captured by VSA embeddings is heavily influenced by the quality of the semantic relationships captured by the input term-document matrix. The TF-IDF weighting scheme better reflects the importance of terms inside documents and our VSA-based learning method still succeeds at capturing most

Input data	Binary term-document matrix		TF-IDF term-document matrix	
Learning method	VSA	LSA	VSA (scaling)	LSA
Mean running time (s)	6.084	6.336	11.257	6.154
Std running time (s)	0.066	0.057	0.298	0.113

Table 3.4: We denote the timings of our learning methods, recorded on a local machine with a Nvidia RTX 3070 GPU having 16GB of VRAM. The mean and standard deviation of timings are reported over 10 repeated experiments. All experiments were conducted over the same training data. The input matrices for each experiment are of size 10 000 (documents) x 91 613 (vocab), the values inside matrices differ between binary occurrence information and tf-idf vector information.

of the information inside the original term-document matrix, thus providing a more accurate representation of semantic relationships according to our validation dataset.

Running times for our learning methods can be found in table 3.4, from this we notice our original GPU-based learning method has similar running times to a GPU-optimized SVD operation implemented in the pytorch library [5]. We do note that the SVD algorithm has been a topic of research for many years and thus is highly optimized.

The algorithm we used for learning from the tf-idf matrix is almost identical to the GPU-based algorithm from our original experiment. The only difference being that we first need to scale the tf-idf vectors of the documents within our range. We note here that we only conducted this additional experiment to verify our hypothesis about the effect of the semantic relation captured in the matrix we learn from on the quality of our VSA embeddings. We did thus not look into possible GPU optimisations for the scaling of the tf-idf vector values. Hence the almost doubling of the running time, since we first need to scale the values within each document tf-idf vector of our term-document matrix.

3.4 Considerations

The conducted experiment is only a validation of applying the VSA representational and computational model to learning patterns present in textual data, more specifically by learning word embeddings. Further investigation is needed to look into its applicability on a larger scale, possibly with different learning algorithms, based on a more intricate application of the VSA operations and exploitation of their properties. Things to take into consideration are:

- **Dimensionality**: Which dimensionality is needed to capture all or most of the semantic information available in a corpus of a certain size? Just as we did for our learning algorithm one should always take into consideration the dimensionality that is needed to capture the information we intend to retain. This is of course dependent on the VSA model used, the VSA operations we apply, and when or if we normalize.

- **VSA model**: Is there a VSA model that gives better performance than MAP? This is highly dependent on the use case and available resources. When picking a suitable model we should evaluate the size (physical storage) encompassed by the HVs, the speed of computation, and the representational capacity of the model, which again depends on the operations and normalization. Models like FHHR might offer a larger bundling capacity, allowing us to capture a larger amount of semantic information for the same dimensionality, but due to the fact that it uses real-valued components, both the amount of memory needed to store the vectors as well as the computational complexity of the bundling operation will increase.

[5] https://pytorch.org/docs/stable/generated/torch.svd_lowrank.html

- **Word Weighing / Scaling**: Another aspect worth investigating is the impact of different word weighing or scaling techniques on the performance of our VSA-based word embeddings. In our additional experiment we already looked at TF-IDF weighting which enhanced the quality of the original embeddings present in the term-document matrix by giving more weight to informative words while downplaying the significance of common or less informative terms. Further research could explore various word-weighing strategies and their effects on the quality of the semantic information contained in the original embeddings. As we did in our experiment, one should then also tailor the learning algorithm to retain this semantic information and investigate how well our VSA-based embeddings succeed at retaining this information.

Chapter 4

VSA Database

In this section we will take a look at the possibilities of encoding data from a table-like structure in high-dimensional spaces using VSAs. We will give an initial implementation of a *VSA Database (VSADB)* and discuss the possibilities of using it for storing and retrieving data.

4.1 Architecture

We will first introduce the architecture of our VSADB. Our approach aims to leverage the principles and properties of VSAs to create a robust and efficient method for handling tabular data. By employing this method we thus aim to create row embeddings of our tabular data with the inherent advantages of noise resistance and similarity-based retrieval.

It is important to note that for the purpose of this exploratory experiment we assume the set of attribute values for a given attribute in our database does not have an inherent similarity structure. As an example, we will consider that 'city' is an attribute of our database. The set of attribute values may contain items like 'London', 'Paris', and 'Brussels'. We will treat these values as distinct and thus unrelated in terms of similarity. Thus, we aim for the similarity of the HV encoding of two database records to be proportional to the number of identical attribute values they share across all columns.

For instance, if two database records share the same values for multiple columns such as 'city', 'state', and 'lastname', the similarity of their corresponding encoded HVs will be higher compared to records that only share a value in one column. This approach ensures that the overall similarity between records is based on the holistic match of their attribute values rather than any assumed inherent similarity between specific attribute values themselves.

We will thus not consider attributes like numerical or ordinal attributes because their values inherently possess a similarity structure. Instead, we will focus on categorical attributes where each possible value is treated as an independent and unrelated entity. This approach simplifies the encoding process and aligns with our goal of exploring the basic capabilities of VSAs for database encoding and retrieval.

4.1.1 Conceptual Idea of Representing Database Records

Before we dive into the details of the architecture of our VSADB, we give an overview of how the VSA operations can be used to encode records.

First, we observe that a record can conceptually be seen as a set or collection of key-value pairs, where the key is the respective attribute or column (name) and the value is that record's value for that attribute. If we combine this conceptual view of a record with the bundle and binding VSA operations as defined in chapter 2, we can encode records as follows:

record	attribute 1	attribute 2	attribute 3
record 1	value 1	value 2	value 3
record 2	value 4	value 5	value 6

Table 4.1: Example of 2 records, each having 3 attributes, in a tabular dataset.

Encoded as:

$$\mathbf{r1} = (\mathbf{a1} \bigcirc \mathbf{v1}) + (\mathbf{a2} \bigcirc \mathbf{v2}) + (\mathbf{a3} \bigcirc \mathbf{v3})$$

$$\mathbf{r2} = (\mathbf{a1} \bigcirc \mathbf{v4}) + (\mathbf{a2} \bigcirc \mathbf{v5}) + (\mathbf{a3} \bigcirc \mathbf{v6})$$

Remember that the bundling operation (+) was used to conceptually represent sets of HVs, and the binding operation ($\bigcirc$) to create associations between HVs or key-value pairs.

4.1.2 Column Representation

At the base of our VSADB lies the concept of a column, which we identify as the combination of a unique column name and its corresponding set of values.

If we take into account our conceptual idea of representing records as HVs from subsection 4.1.1, we conclude that a column must consist of the following key components:

- **Atomic Vector:** Each column is associated with a unique atomic HV, which serves as a fixed identifier for the column name. This atomic HV is crucial for the binding operation, where data values are combined with their respective column identifiers.
- **Codebook:** Each column maintains a codebook, a specialized dictionary-like data structure that maps attribute values to their corresponding HVs. The codebook ensures that each value within the column has a unique and consistent HV representation. As mentioned before we assume the attribute values consist of solely categorical data and thus also may assume these are given in a string format. By using a dictionary or hash-based storage method we thus enable efficient lookups and storage of these values and their corresponding HV representation.

When a new value is added to a column, it is first converted into a HV using the column's codebook. This process involves checking if the value already exists in the codebook. If it does, the existing HV is retrieved. If not, a new HV is generated and added to the codebook. This ensures that each value within a column is encoded by a unique atomic HV.

Overall, the representation of columns in VSADB is designed to be flexible and efficient, allowing for straightforward handling of categorical data. By encapsulating the logic for encoding and managing attribute values, the VSADB architecture maintains a clear separation of concerns. That is each column handles its own attribute values and their respective HV representations and we do not have to worry about this when encoding our rows. We simply add a new value for the attribute, and the column class returns its corresponding HV representation, ensuring the unique representation property and the storage in the background.

4.1.3 Encoding Rows

We now have the generation of the value HVs, that serve as the building blocks for the encoding of our records, in place. All that is left is to utilize these attribute- and value HVs and combine them into records HVs as illustrated by our conceptual idea mentioned before.

The process of encoding a database row thus involves the following steps:

1. **Initialize Row Attribute Value HVs:** For each column, retrieve or generate the HV for the corresponding attribute value. If the value has not been used before in any record for that attribute we generate a new atomic HV representing that value. On the other hand, if the value has been used in another record before, we use the column's codebook

to retrieve the HV for that value. This ensures that records with the same value for that attribute also use the same HV representation for that value.

2. **Binding:** Bind each column's atomic HV with its attribute value HV using the VSA binding operation. This essentially associates the column identifier with the attribute value, creating a unique HV for each attribute-value pair. The binding operation is crucial as it ensures that the attribute value HVs are contextually linked to their respective columns, allowing the database to maintain the structure of the original tabular data. The resulting bound HVs are then ready to be aggregated to form the row HV.

3. **Bundling:** Bundle all the previously created attribute-value pairs to form a single HV representing the entire row. Recall that the bundling operation generates an HV that is similar to all its inputs and thus aggregates their information. The resulting HV is thus a high-dimensional representation of the entire row, capturing the combined information of all attribute-value pairs.

4. **Normalization:** Next, we normalize the bundled HV to ensure consistency. Normalization adjusts the HV so that it conforms to the expected properties of HVs in the VSA model that is used to represent the information. While this step is not necessary for the functioning of the encoding, we employ it to maintain the integrity of the generated encoding with respect to the VSA model used.

   For example, the bundling of MAP HVs will result in components that are in $\mathbb{Z}$, while the magnitude of these positive or negative integers do give us some extra information when using cosine similarity, they do not conform with the MAP model which specifies its atomic HV components to be drawn from $\{-1, 1\}$. As discussed in section 2.3, normalization for the MAP model is implemented by a 'binarization' of the negative or positive integers to $\{-1, 1\}$ respectively. As seen in chapter 2, normalizing thus may introduce some 'noise' because we take away some of the information, for example, the magnitude in the MAP model.

5. **Row Storage:** Store the normalized row HV in the database's internal storage structure, such as the earlier mentioned codebooks, a simple list, or if speed of computation is a necessary requirement, one could use a specialized vector datastore that allows fast retrieval or indexing of stored vectors [1]. This storage structure acts as a mapping between the encoded HVs and their corresponding rows in the original dataset. It allows for quick access to the encoded data based on their original indices or identifiers, ensuring that the original rows can be retrieved after a similarity-based search for target rows using a query vector.

By following these steps, we can encode each row in the database as a single HV, preserving the relationships between attribute values and enabling our goal of similarity-based retrieval of tabular data.

4.1.4 Querying our Database

Now that we have encoded our tabular data as HVs we can perform similarity queries on our VSADB by creating a query HV. We will see that the process of creating a query HV is very similar to our row encoding process, using this query HV we can identify candidate result records based on a similarity search.

We give a more detailed view of the steps we take during the query process:

1. **Create Query HV:** Construct the query HV by binding atomic attribute HVs with the attribute values specified in the query. This process mirrors the row encoding process, ensuring that the query HV is structured in the same way as the stored row HVs.

[1] https://github.com/facebookresearch/faiss

2. **Calculate Similarity:** Compare the query HV with each stored row HV using a similarity measure, such as cosine similarity or Hamming similarity, depending on the VSA model. The similarity scores indicate how closely each row matches the query.

3. **Retrieve Most Similar Rows:** Identify the rows with the highest similarity scores and retrieve them. This step involves ranking the rows based on their similarity scores and selecting the top results.

The interested reader might notice that steps 2 and 3 essentially perform a *nearest-neighbor retrieval (NN-search)* in our high-dimensional space. By following these steps, the VSADB can efficiently retrieve rows that are most similar to the specified query, providing a robust mechanism for data retrieval based on matching attribute values.

4.2 Experiments

Now that we have knowledge of the inner workings of our VSADB, we can conduct an experiment to evaluate the performance of our VSA Database in terms of encoding and retrieval efficiency. The experiment involves using a real-world tabular dataset with categorical attributes, encoding the records or rows of our tabular dataset as HVs, and performing similarity queries to assess the accuracy and robustness of the retrieval process.

4.2.1 Dataset

For our experiment, we utilized a small real-world tabular dataset containing categorical attributes. The dataset consists of the following 5 attributes:

- **fname**: First name of the individual.
- **lname**: Last name of the individual.
- **city**: City of residence.
- **state**: State of residence.
- **gender**: Gender of the individual.

The dataset contains the following records:

index	fname	lname	city	state	gender
0	John	Doe	Riverside	NJ	M
1	Jack	McGinnis	Philadelphia	PA	M
2	John	Repici	Riverside	NJ	M
3	Stephen	Tyler	Sioux Falls	SD	M
4	John	Blankman	Sioux Falls	SD	M
5	Joan	Anne	Denver	CO	F
6	Jack	Repici	Riverside	NJ	M
7	Lilly	Repici	Philadelphia	PA	F

Table 4.2: Small real-world tabular dataset used in the experiment

This dataset served as the basis for our experiments. While it is not large in size, it allows us to execute some interesting queries like finding individuals from the same family by querying on a certain last name, identifying people residing in the same city, or a combination of these.

Additionally, we extended the dataset by generating random rows to increase its size, diversity, and complexity. To extend the dataset we incorporated a supplementary dataset containing information about the top 1000 cities in the United States and their corresponding states. We generated 10 000 additional rows for our dataset as follows:

- Generate random first names (**fname**) and last names (**lname**) by generating random strings of a certain length.
- Randomly select a state from the list of state abbreviations of our supplementary dataset.
- Randomly select a gender as either 'M' or 'F'.
- Assign a city to each record based on the selected state to preserve the city-state relationship using our supplementary dataset.

This combined and extended dataset of 10 008 records each having 5 attributes was used to simulate real-world scenarios and evaluate the performance of our VSADB in terms of encoding efficiency and retrieval accuracy. By using a mix of a small, focused dataset and a larger, more diverse dataset, we were able to comprehensively test the capabilities of our system across different data scales and complexities.

4.2.2 Encoding of Data

The creation of our VSADB based on this data was done according to the steps mentioned in section 4.1, for this experiment we decided to opt for the BSC VSA model with a dimensionality of 1000.

We chose the BSC model because it is conceptually the easiest to understand and also the exemplary VSA model we used throughout the larger part of chapter 2. The reason we do not look for an alternative VSA model as with our word embeddings in chapter 3 is that there is no need for storing inputs of the bundling operation to perform non-normalized bundling. That is all the attribute-value pair HVs needed for creating a record or query HV are available at the moment we have to apply the bundling operation.

For the dimensionality we decided to opt for only 1000 as we will work with relatively small bundle sizes because we have only 5 attributes, a bundle of a record will thus only consist of 5 input HVs. If we look back at figure 2.6, we observe that for bundle sizes that are smaller than 20 a dimensionality of 1000 should suffice to obtain a near-perfect bundling capacity. This dimensionality should thus be sufficient not only for our experiment, but also for the encoding of most database relations as they typically contain less than 20 attributes. Because we are using the BSC model, which consists of binary (bit) components this means we can encode our records using only 1000 bits, or 125 bytes. A more in-depth analysis of the effect of the dimensionality on the performance of our database and query accuracy will follow in section 4.4.

For the storage of our row HVs and thus our encoded database records we used the codebook datastructure. As mentioned before this is a dictionary-like data structure where each key is a unique identifier for a row (e.g., a primary key), and the corresponding value is the encoded HV of the row. We pick the primary key or identifier for each row to be the index or location of this row in the original dataset, this facilitates easy access to the stored encoded data based on their original indices. For this exploratory experiment where we aim to verify the inner workings of our setup we are not interested in the speed of retrieval but in the quality of the retrieved results and thus decided not to opt for more advanced vector store methods.

For the sake of completeness, we do mention that the encoding of the 10 008 records each having 5 attributes took an average of 6.388 seconds over 10 repeated encodings of the entire database with a standard deviation of 0.621 seconds.

4.2.3 Analysis

Now that we have the data in place and encoded, we can perform a series of analyses to evaluate the performance and robustness of our VSADB.

We start our analysis by looking at the result of some queries, performed as discussed in subsection 4.1.4.

Query 1: The Repici family
A first query could be to retrieve all the records that concern the members of the Repici family as seen in our original dataset in table 4.2. We perform this query by first creating the correct query HV, which we can then use to perform a NN-search.

The steps involved in constructing and executing this query are as follows:

- **Constructing the Query HV:** To construct the query HV, we need to bind the 'lname' atomic HV with the HV corresponding to the value 'Repici' from this column. The 'lname' atomic HV can be retrieved from the column object representing last names, and the HV for 'Repici' can be retrieved from the codebook maintained by this column. The resulting Query HV is thus constructed as follows: $\mathbf{q} = \mathbf{lname} \bigcirc \mathbf{repici}$.

  As seen in chapter 2, $\bigcirc$ denotes the binding operation, **lname** is the unique HV key representation of the 'lname' column and **repici** is the atomic HV representing the value 'Repici' for the 'lname' column. The resulting query HV thus uniquely represents the combination of the column ('lname') and its value ('Repici').

- **Performing the NN-Search:** Once we have the query HV, we use it to perform a NN search on the row HVs stored in the codebook of our VSADB. The NN search identifies the rows with HVs most similar to the query HV, effectively retrieving the records of interest.

fname	lname	city	state	gender	Normalized Hamming Similarity
John	Repici	Riverside	NJ	M	0.684
Jack	Repici	Riverside	NJ	M	0.675
Lilly	Repici	Philadelphia	PA	F	0.667
dmdroj	rufnyo	Gary	IN	F	0.552
tzvcpa	lzrbtw	Lawrence	KS	F	0.546
pmvnwd	eirkgp	Missoula	MT	F	0.546
sxbzsi	mbnyye	Little Rock	AR	F	0.545
jgelra	ytyflj	Washington	DC	F	0.545
dvwxoa	rncqos	Midland	TX	M	0.544
jnjlfa	znyxwz	Little Rock	AR	F	0.543

Table 4.3: 10 most similar records retrieved for the query concerning the Repici family with the normalized hamming similarities with respect to the query HV

If we now look at the 10 most similar resulting HVs in table 4.3 we indeed observe that we correctly retrieved all HVs concerning the Repici family from the 10 008 records in our VSADB.

However, we also see irrelevant results further down the list as only the top 3 are the relevant members of the Repici family in our dataset. This indicates that retrieving the most similar results to our query HV is indeed a valuable strategy, but there still is a need for manual verification of the top-k retrieved results. We also observe that the normalized Hamming similarity for the relevant results (≈ 0.67) is significantly higher than those of the most similar irrelevant results (≈ 0.545). Thus to avoid retrieving irrelevant results, we could look into the existence of a similarity threshold. A more in-depth analysis into the use of such a similarity threshold to avoid false positives will be discussed later.

Query 2: The Repici family, living in Riverside
We can extend the previous query to as many attributes as we would like. We could for example add an additional attribute-value pair to the query HV concerning the city attribute. For this example we would like to identify all the members of the Repici family that live in the Riverside city.

The steps for constructing and executing this query are as follows:

- **Constructing the Query HV:**
  - Retrieve the HVs: Retrieve the atomic column HVs for 'lname' and 'city', and the value HVs for 'Repici' and 'Riverside', from their respective columns.
  - Bind the Attribute HVs: Bind each attribute's atomic HV with its corresponding value HV.
  - Bundle the HVs: Bundle the resulting attribute-value HVs to form a single query HV.
  - The resulting query HV is thus constructed as:

$$\mathbf{q} = \mathbf{lname} \bigcirc \mathbf{repici} + \mathbf{city} \bigcirc \mathbf{riverside}$$

- **Perform the NN Search:** Use the bundled query HV **q** to perform a NN search on the record HVs stored in the codebook of our VSADB.

We can again look at the 10 most similar HV to our query HV in table 4.4. Here we observe that we again successfully identified the 2 records that answer our query. But we also notice that among the 10 most similar records there are records that partially match our query. These are records that either have Repici as their last name or live in Riverside. If we look at the normalized hamming similarities of these full and partial match results with our query HV, we notice that the similarity for the full matches, as expected, is significantly higher than the others. But we also notice that the similarity for the partial matches is also significantly higher than those that do not match at all, this conforms with our holistic similarity view, where the similarity of record HVs (or the query HVs) is proportional to the number of matching attribute values.

fname	lname	city	state	gender	Normalized Hamming Similarity
Jack	Repici	Riverside	NJ	M	0.695
John	Repici	Riverside	NJ	M	0.689
wpmrlq	pnthyi	Riverside	CA	M	0.598
Lilly	Repici	Philadelphia	PA	F	0.597
John	Doe	Riverside	NJ	M	0.592
gnumfw	ilbywx	Orem	UT	M	0.519
xopccc	fxgopl	Colorado Springs	CO	F	0.518
lfkaha	wbutxq	Sioux Falls	SD	F	0.518
nlntid	vuljpm	Reno	NV	F	0.518
zcudtl	mxbtnf	Albuquerque	NM	F	0.517

Table 4.4: 10 most similar records retrieved for the query concerning the Repici family living in Riverside with their respective normalized hamming similarities to the query HV

From these observations, we conclude that we can correctly retrieve the results for partial match queries where we specify values for a subset of the attributes and subsequently also for point queries if we were to specify a value for each attribute of the relation.

Yet an interesting question arises when we look at the results of Query 2: Can we define a similarity threshold that would ensure that we only retrieve the relevant records, and thus not records that only partially match our query HV?

To answer this question we look for a similarity threshold that ensures only the relevant records are retrieved. For the calculation of such a threshold value we conducted a theoretical analysis that can be found in section 4.3. This mathematical derivation gives us many useful insights, not only for the definition of such a similarity threshold, but it also verifies some of the experimental results from chapter 2.

We mention that reading the mathematical section is not mandatory to understand the continuation of our analysis of this experiment as well as the results and conclusions. The non-mathematically inclined reader may thus continue to section 4.4, where we give a brief description of the result of the theoretical analysis, the interested reader may verify the mathematical derivations in section 4.3.

4.3 Mathematical Analysis

In the following we denote:

- D as the dimensionality of the HVs;
- N as the number of rows in the database, i.e. the number of HVs in item memory;
- All VSA vectors as random binary HVs of dimensionality D (BSC)

4.3.1 Atomic BSC HV's Similarity

We will first take a look at the Hamming Similarity between two atomic BSC HVs x and y of dimensionality D.

As in chapter 2, we define the Hamming similarity by the number of components that two input vectors agree on:

$$\delta_{ham}(\mathbf{x}, \mathbf{y}) = \sum_{i}^{\mathbf{D}} 1\{\mathbf{x}_i = \mathbf{y}_i\}$$

We can normalize this similarity measure, restricting the range to $[0, 1]$ if we divide it by the dimensionality D:

$$\delta_{ham\_norm}(\mathbf{x}, \mathbf{y}) = \frac{\delta_{ham}(\mathbf{x}, \mathbf{y})}{\mathbf{D}}$$

As we did in chapter 2 we will use the normalized Hamming Similarity since it is more intuitive, and not proportional to the dimensionality used.

If we assume that x and y are BSC atomic HVs, we know the value for each dimension follows a $Bernoulli(p = 0.5)$ distribution. From this, it is straightforward to see that the Hamming Similarity between them is also a random variable that follows a binomial distribution.

$$\delta_{ham\_norm} \sim \text{Bin}(D, 0.5) * \frac{1}{D}$$

The expected value of this random variable is hence:

$$\mathbb{E}[\delta_{ham\_norm}(x, y)] = \mathbb{E}[\text{Bin}(D, 0.5) * \frac{1}{D}] = \frac{1}{D} * D * 0.5 = 0.5$$

Note that the expected value of the Hamming similarity is independent from D.

The variance is calculated as follows:

$$\text{Var}[\delta_{ham\_norm}(x, y)] = \text{Var}[\text{Bin}(D, 0.5) * \frac{1}{D}] = \frac{1}{D^2} * D * 0.5^2 = \frac{0.5^2}{D}$$

Observe that the variance is inversely proportional to D.

4.3.2 Bundle Similarity

We will now take a look at the similarity between a bundled HV v and its input HVs $x_1, x_2, ..., x_k$. Recall that bundling for BSC is implemented using a majority vote (thresholded sum) over all the input HVs for each dimension to decide if the dimension of the resulting HV should be set

to 0 or 1. For these calculations we assume that k is uneven, if not we can add a random vector to the bundle to make it uneven. We will denote the bundle as $v = x_1 + x_2 + ... + x_k$.

For notational simplicity we will now look at the similarity between v and x_1, but these results can be generalized to any input HV $x_j : j \in [1, k]$. In the following we will denote $x_{j,i}$ as the i^{th} dimension of the j^{th} input HV, and v_i as the i^{th} dimension of our bundled or resulting HV.

Assume $x_{1,i} = 1$, then $v_i = 1$ if and only if at least half of the other $k-1$ components are 1. We can now calculate the probability of $x_{1,i} = v_i$ as follows:

$$\begin{aligned} p(x_{1,i} = v_i) &= p(X \geqslant \frac{k-1}{2}) \text{ where } X \sim \text{Bin}(k-1, 0.5) \\ &= \sum_{i=\frac{k-1}{2}}^{k-1} \binom{k-1}{i} 0.5^i 0.5^{k-1-i} \\ &= 0.5^{k-1} \sum_{i=\frac{k-1}{2}}^{k-1} \binom{k-1}{i} \end{aligned}$$

We can now simplify the sum using the binomial theorem:

$$\begin{aligned} p(x_{1,i} = v_i) &= 0.5^{k-1} \left[\sum_{i=0}^{k-1} \binom{k-1}{i} - \sum_{i=0}^{\frac{k-1}{2}-1} \binom{k-1}{i} \right] \\ &\quad \text{using the binomial theorem: } \sum_{i=0}^{k} \binom{k}{i} = 2^k \\ &= 0.5^{k-1} \left[2^{k-1} - \left(\sum_{i=0}^{\frac{k-1}{2}} \binom{k-1}{i} - \binom{k-1}{\frac{k-1}{2}} \right) \right] \\ &= 1 - 0.5^{k-1} \sum_{i=0}^{\frac{k-1}{2}} \binom{k-1}{i} + 0.5^{k-1} \binom{k-1}{\frac{k-1}{2}} \\ &= 1 - 0.5^{k-1} \left[\sum_{i=0}^{\frac{k-1}{2}} \left(0.5 \binom{k-1}{i} + 0.5 \binom{k-1}{k-1-i} \right) \right] + 0.5^{k-1} \binom{k-1}{\frac{k-1}{2}} \\ &= 1 - 0.5^{k} \left[\sum_{i=0}^{\frac{k-1}{2}} \binom{k-1}{i} + \sum_{i=0}^{\frac{k-1}{2}} \binom{k-1}{k-1-i} \right] + 0.5^{k-1} \binom{k-1}{\frac{k-1}{2}} \\ &= 1 - 0.5^{k} \left[\sum_{i=0}^{k-1} \binom{k-1}{i} + \binom{k-1}{\frac{k-1}{2}} \right] + 0.5^{k-1} \binom{k-1}{\frac{k-1}{2}} \\ &= 1 - 0.5^{k} 2^{k-1} - 0.5^{k} \binom{k-1}{\frac{k-1}{2}} + 0.5^{k-1} \binom{k-1}{\frac{k-1}{2}} \\ &= 1 - 0.5^{k} 2^{k-1} + 0.5^{k-1} \left[\binom{k-1}{\frac{k-1}{2}} - 0.5 \binom{k-1}{\frac{k-1}{2}} \right] \\ &= 1 - 0.5^{k} 2^{k-1} + 0.5^{k} \binom{k-1}{\frac{k-1}{2}} \\ &= \frac{1}{2} + 0.5^{k} \binom{k-1}{\frac{k-1}{2}} \end{aligned}$$

Note the previous calculation is the same for every dimension i. This probability gives us the building block of the normalized Hamming similarity between v and x_1 or actually any of its input HVs x_j, which we can now calculate as follows:

$$\delta_{ham\_norm}(v, x_1) = \sum_{i=1}^{D} p(x_{1,i} = v_i) * \frac{1}{D}$$

From this we see that the normalized Hamming similarity between v and x_1 follows a binomial distribution with D trials and probability $p(x_{1,i} = v_i)$.

$$\delta_{ham\_norm}(v, x_1) \sim \text{Bin}(D, p(x_{1,i} = v_i)) * \frac{1}{D} \tag{4.1}$$

The expected value of this binomial distribution is:

$$\mathbb{E}[\delta_{ham\_norm}(v, x_1)] = \mathbb{E}[\text{Bin}(D, p(x_{1,i} = v_i)) * \frac{1}{D}] = \frac{1}{D} * D * p(x_{1,i} = v_i) = p(x_{1,i} = v_i) \tag{4.2}$$

With the variance calculated as follows:

$$\begin{aligned}
\text{Var}[\delta_{ham_norm}(v, x_1)] &= \text{Var}[\text{Bin}(D, p(x_{1,i} = v_i)) * \frac{1}{D}] \\
&= \frac{1}{D^2} * D * p(x_{1,i} = v_i) * (1 - p(x_{1,i} = v_i)) \\
&= \frac{p(x_{1,i} = v_i) * (1 - p(x_{1,i} = v_i))}{D}
\end{aligned}$$

From this we conclude that the expected value of the normalized Hamming similarity between v and x_1 is equal to the probability of $x_{1,i} = v_i$ and the variance is inversely proportional to D.

4.3.3 Similarity threshold

We can now calculate a threshold for the normalized Hamming similarity between v and x_1. Where we are $\alpha * 100\%$ confident that the normalized Hamming similarity between v and x_1 is greater than the thershold t. In these derivations we will use the normal approximation of the binomial distribution, which is valid for large $D > 30$.

$$\begin{aligned}
\delta_{ham_norm}(v, x_1) &\sim \text{Bin}(D, p(x_{1,i} = v_i)) * \frac{1}{D} \\
&\approx \mathcal{N}(D * p(x_{1,i} = v_i), D * p(x_{1,i} = v_i) * (1 - p(x_{1,i} = v_i))) * \frac{1}{D} \\
&= \mathcal{N}(p(x_{1,i} = v_i), p(x_{1,i} = v_i) * (1 - p(x_{1,i} = v_i)) * \frac{1}{D})
\end{aligned}$$

We can now calculate the threshold t as follows:

$$\begin{aligned}
&\mathbb{P}[\delta_{ham_norm}(v, x_1) \geqslant t] = \alpha \\
\iff &\mathbb{P}(\frac{\delta_{ham_norm}(v, x_1) - p(x_{1,i} = v_i)}{\sqrt{p(x_{1,i} = v_i) * (1 - p(x_{1,i} = v_i)) * \frac{1}{D}}} \geqslant \frac{t - p(x_{1,i} = v_i)}{\sqrt{p(x_{1,i} = v_i) * (1 - p(x_{1,i} = v_i)) * \frac{1}{D}}}) = \alpha \\
\iff &\frac{t - p(x_{1,i} = v_i)}{\sqrt{p(x_{1,i} = v_i) * (1 - p(x_{1,i} = v_i)) * \frac{1}{D}}} = z(1 - \alpha) \\
&\text{where } z(1 - \alpha) \text{ is the } (1 - \alpha) \text{ quantile of the standard normal distribution} \\
\iff &t = p(x_{1,i} = v_i) + z(1 - \alpha) * \sqrt{p(x_{1,i} = v_i) * (1 - p(x_{1,i} = v_i)) * \frac{1}{D}}
\end{aligned}$$

4.4 Experiment Review

From the theoretical analysis, we identified that the normalized Hamming similarity between a bundled HV and its input HVs follows a binomial distribution. This distribution can be approximated by a normal distribution for large dimensionalities $D > 30$, we may assume this is true as dimensionalities used by VSA models typically are in the thousands. Using this information we derived a threshold value that ensures with high confidence α that the normalized similarity between a bundled HV and its input HV is above this threshold. The objective is defined as follows:

$$\mathbb{P}[\delta_{ham\_norm}(v, x_j) >= t] = \alpha$$

If we now calculate this for our threshold value t we get:

$$t = p(x_{j,i} = v_i) + z(1-\alpha) * \sqrt{p(x_{j,i} = v_i) * (1 - p(x_{j,i} = v_i)) * \frac{1}{D}}$$

Where $p(x_{j,i} = v_i)$ is the probability that an individual dimension of the bundled HV matches that of an input HV, and $z(1-\alpha)$ is the quantile of the standard normal distribution for the desired confidence level α.

We can now apply this threshold value to our earlier examples to verify that they do indeed allow us to correctly identify the records of interest and only these. If we calculate the threshold value for a confidence level $\alpha = 0.99$, $D = 1000$, and bundle size of 5 we get $t = 0.653$.

If we look at our results in table 4.3 and table 4.4 we see that if we only keep the records that have a normalized Hamming similarity above our threshold value $t = 0.653$ that we correctly retrieve the records that answer our query. Moreover, we do not keep any partial matches of false positives, we only keep the relevant records that fully match with our query HV.

Recall vs. Precision
The interested reader might notice that the level of confidence α we set for our threshold value is equal to the recall rate that we would achieve when filtering our query results using the threshold value calculated based on this α.

We clarify this by looking at the formula we based the calculation of t on:

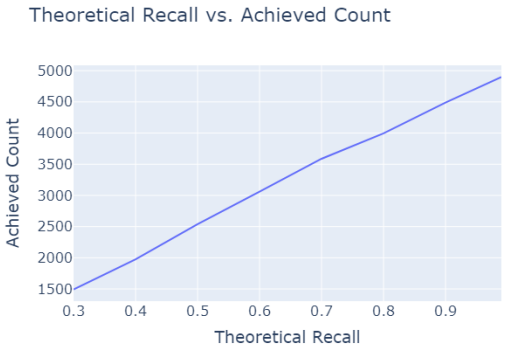
$$\mathbb{P}[\delta_{ham_norm}(v, x_j) >= t] = \alpha$$

This value stands for the probability that the normalized Hamming similarity between the bundled HV and its input HVs meets or exceeds the threshold t. Hence, α directly translates to the recall rate, which is the proportion of relevant records that we retain when filtering our results based on their similarity with the query HV using this threshold. In our example we set $\alpha = 0.99$, we thus calculated a threshold value t such that there is a 99% probability that the similarity between the query HV and any of the relevant HVs is above this value t.

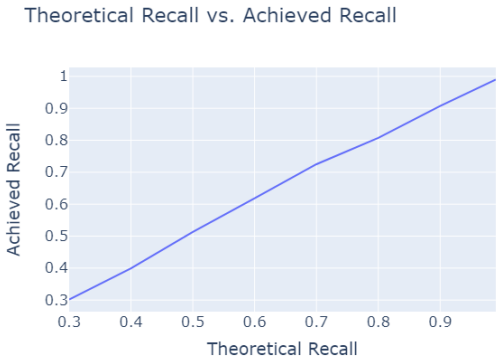
We verify this by retrieving the relevant records that have gender equal to male (or female) from our dataset for different threshold values. Since the gender attribute is assigned with a probability of 0.5 to our randomly generated values we have a larger scale of data to verify the recall.

We can see that the results in figure 4.1 verify our hypothesis that the significance level α we use for calculating our threshold value t is equal to the recall of the resulting records we obtained by querying with a query HV and filtering the results based on the threshold t.

One might wonder if using such a high recall as in our example $\alpha = 0.99$ might lead to a low precision. In other words, we might retain irrelevant HVs that thus do not or only partially match with our query HV (false positives). Yet in our experiments concerning Query 1, Query 2, and in our recall experiment querying on gender, we do not observe any false positives or irrelevant records being retained after filtering the results using the threshold value.



(a) We plot the theoretical recall or significance level α that is set when calculating the threshold value t against the number of male records (true positives) obtained after querying for male records and filtering on similarities above the threshold value t. We note that the total number of records with gender equal to male for our dataset is equal to 4897. We also note that we did not obtain any false positives in these experiments, that is we did not retain a record having the female gender for any of the queries.



(b) We plot the theoretical recall or significance level α that is set when calculating the threshold value t against the recall of the obtained records after querying for male records and filtering on similarities above the threshold value t. The recall was calculated by dividing the number of true male records retrieved for a certain threshold as seen in the figure on the left by the total number of male records in the dataset (4897).

Figure 4.1: Experiment on the equivalence of recall based on threshold value calculated using a certain significance level α

We identify why this is the case by looking at the following result obtained from our theoretical analysis in section 4.3: the variance of the similarity between the resulting, bundled, HV and any of its input HVs is proportional to the number of input HVs (see definition $p(x_{j,i} = v_i)$) of the bundle and inversely proportional the dimensionality.

Thus a higher bundle size and/or a lower dimensionality of the HVs makes it more difficult to distinguish between the input HVs of the bundle and any other non-input (atomic) HV. This thus means that there is a higher probability of incorrectly identifying answers to our query when setting a high recall if we have a larger bundle size or a lower dimensionality.

But this is not a problem for our database experiment, thanks to the fact that our records only consist of 5 attributes, we thus have an extremely small bundle size of only 5 key-value pairs. For our dimensionality of 1000 this bundle size is small enough to obtain a high precision next to the high recall value that we specify.

To verify this we run an experiment, where we generate $n = 100$ bundles of size $k = 5$, the same as the records in our database, and use these bundled HVs to query a database of 10 000 random HVs. This database or set of 10 000 random HVs (item memory) of course contains the inputs used for the bundle or query HVs otherwise we would not get valid results. We calculate the threshold value t for filtering the results of our query, based on a recall or significance level of $\alpha = 0.95$, and look at the precision, recall, and accuracy of the retrieved results based on this threshold t. We give a brief overview of the key metrics used for this experiment:

- **True Positive (TP):** The number of relevant HVs correctly identified by having a normalized Hamming similarity with the query HV that is equal to or higher than our similarity threshold t.
- **False Positive (FP):** The number of irrelevant HVs incorrectly identified as relevant by the query HV.
- **True Negative (TN):** The number of irrelevant HVs correctly identified as irrelevant by having a normalized Hamming similarity with the query HV that is smaller than our similarity threshold t.

- **False Negative (FN):** The number of relevant HVs not identified as a positive by the query HV.

These metrics allow us to calculate the following performance measures:

- **Recall:** $\frac{TP}{TP+FN}$ In our context, it represents The proportion of relevant HVs that were correctly identified by the query HV and our threshold value.
- **Precision:** $\frac{TP}{TP+FP}$ This metric measures the proportion of retrieved instances that are relevant. In our context, it represents the proportion of retrieved records that are actually correct matches to the query HV.
- **Accuracy:** $\frac{TP+TN}{TP+TN+FP+FN} \sim \frac{TP+TN}{\textbf{size of item memory}}$ This metric measures the overall correctness of the system, it represents the proportion of all correctly retrieved instances (both positive and negative), with respect to the query HV and the threshold used, out of all instances in item memory.

We run this experiment for dimensionalities ranging from 50 to 3000, results can be seen in figure 4.2.

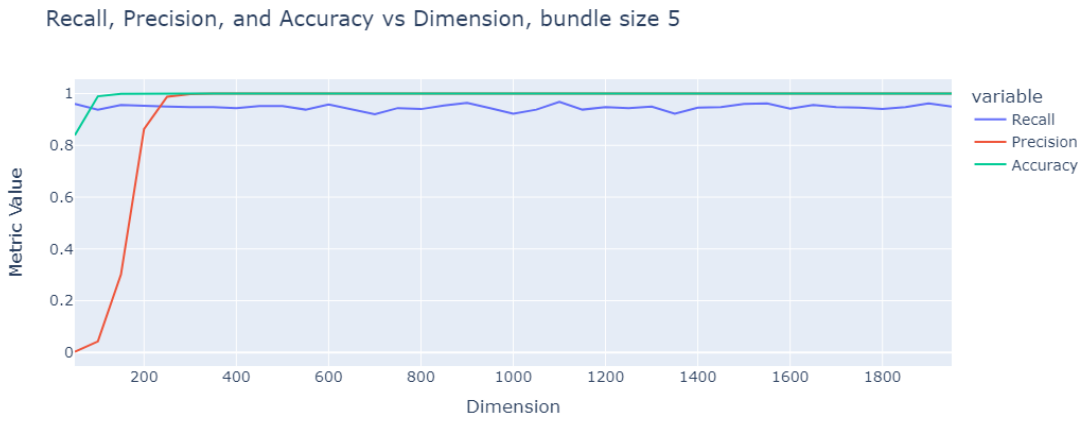


Figure 4.2: Experiment identifying the effect of the dimensionality on our ability to accurately retrieve results using a threshold value t calculated based on a recall or level of significance of $\alpha = 0.95$, the dimensionality D and the size of our bundles $k = 5$, which remains constant. The bundle size is set to 5 to simulate the effect on records in our database with 5 attributes.

First, we observe that we achieve a recall of 0.95, irrespective of the dimensionality. This is no surprise since we used a level of significance $\alpha = 0.95$ to calculate our threshold value t. More importantly, we observe we also obtain perfect precision and thus also accuracy for dimensionalities of as little as 300. This proves that we can use our threshold method to correctly identify the relevant records, and only these, by querying our database that encodes records of 5 attributes using 1000 dimensional BSC HVs.

For the sake of completeness, we also ran the experiment for a near-perfect recall of 0.995, as can be seen in figure 4.3.

From this experiment, we conclude that we could actually use a dimensionality of as little as 350 to encode our records of 5 attributes and obtain near-perfect recall (0.995), precision, and accuracy for our queries.

To conclude our experiments we also ran the last experiment for a larger bundle size of 20, to see what dimensionality would suffice in case we would encode records of a relation that contains up to 20 attributes.

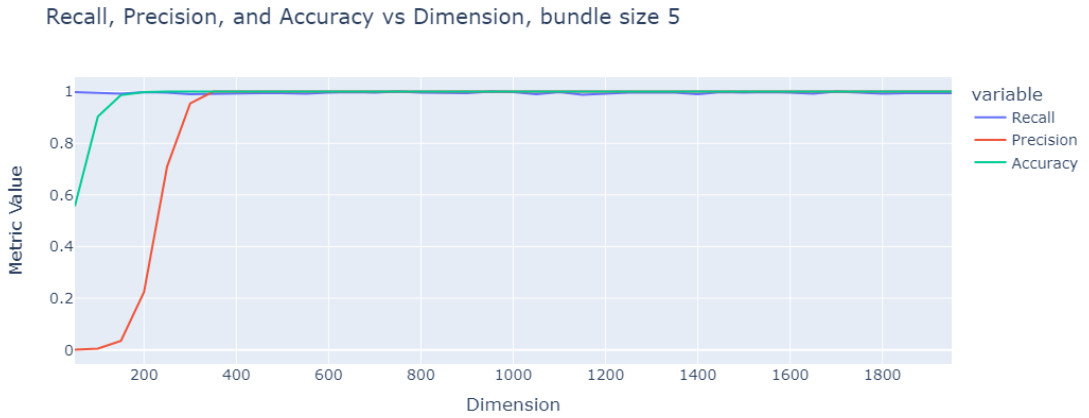


Figure 4.3: Experiment identifying the effect of the dimensionality on our ability to accurately retrieve results using a threshold value t calculated based on a recall or level of significance of $\alpha = 0.995$, the dimensionality D and the size of our bundles $k = 5$, which remains constant.

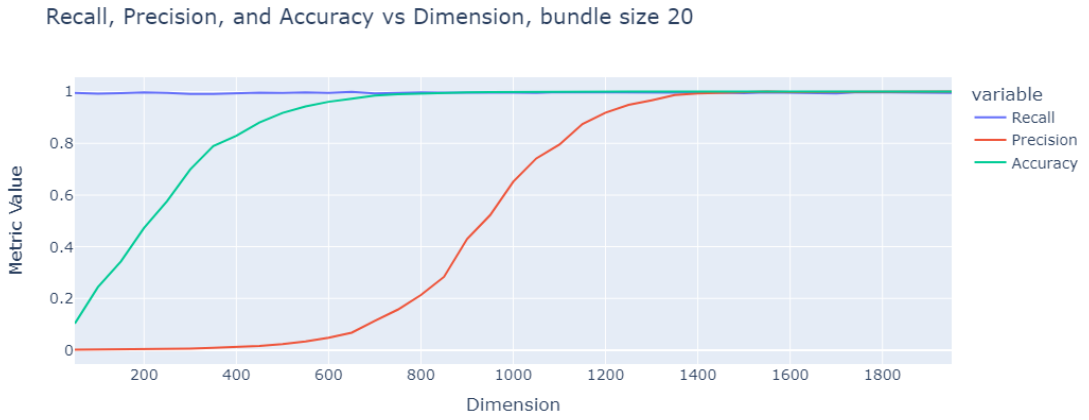


Figure 4.4: Experiment identifying the effect of the dimensionality on our ability to accurately retrieve results using a threshold value t calculated based on a recall or level of significance of $\alpha = 0.995$, the dimensionality D and the size of our bundles $k = 20$, which remains constant.

If we observe these results in figure 4.4, we see that a higher bundle size indeed has a negative impact on our accuracy when retrieving results based on our threshold. We observe that we still attain a recall of 0.995 since we calculated the threshold value t using a recall or level of significance of $\alpha = 0.995$. But we also observe that the precision is considerably lower for lower dimensionalities, indicating that we retain false positive results when filtering on the threshold value t. This not only gives us insight into what dimensionality we might need to use to encode database records of 20 attribute values but it also verifies our theoretical observation. That is, increasing the bundle size and/or lowering the dimensionality negatively impacts the precision and accuracy because it becomes more challenging to distinguish between the HVs corresponding to the input attributes and those of unrelated HVs in the database. Specifically, we see that for a bundle size of 20, a higher dimensionality of at least 1400 is required to maintain perfect precision and accuracy.

4.5 Conclusion

In this chapter, we introduced the architecture of a VSA Database for encoding, storing, and retrieving tabular data by means of point or partial match queries. We conducted experiments to validate the potential of our method using this VSADB architecture for efficient and robust encoding and retrieval of tabular, categorical data.

We also did a theoretical analysis to identify a threshold value, which allowed us to retrieve only the relevant records for a given query HV.

We experimentally identified the dimensionality that can be used for a fixed recall and bundle size to obtain a near-perfect accuracy, having a minimal amount of false positives and negatives. For a bundle size of 5 and a near-perfect recall of 0.995 we identified that a dimensionality of 350 should suffice to not only achieve a near-perfect recall of 0.995 but also a near-perfect precision and accuracy. Increasing the number of attributes or bundle size to 20, leads to the need for 1400 dimensions to represent our HVs and obtain near-perfect retrieval.

From this, we may conclude that for typical databases or tabular datasets, ranging from 5 to 20 attributes, a dimensionality of 1500 should suffice to encode and reliably retrieve all and only the relevant results from this encoded dataset. As we based the experiments and our theoretical analysis on the BSC model, which has binary components, we may assume we can reliably encode and decode a dataset consisting of 10 000 records, each having 5 to 20 attributes, in as little as 1.9MB. It is important to note that the reliability of the encoding and decoding only depends on the number of attributes and the dimensionality used to calculate the threshold value for a certain pre-defined recall, and thus not on the number of encoded records in our VSADB.

An important side note for this conclusion is that this size in MegaBytes only considers the space needed for the encoded record HVs. To encode these records we also used the column representations, containing the HVs for the values in that respective column. If we assume in the worst case that the set of values for each column or attribute does not contain any duplicates, this would mean that each column maintains a codebook of 10 000 atomic value HVs of size 1.9MB. In the previous assumption, we naively disregard the space needed to maintain the hash-based dictionary or index structure, as each value HV has to be related to its actual value using this dictionary structure.

Another point of consideration is that after we perform the similarity search using our constructed query HV on the VSADB of encoded records of size 1.9MB, we still have to decode the relevant HVs we identified. Thus once we have retrieved these relevant HVs or more specifically their indices, we still rely on the original dataset, to identify the original records that correspond to the retrieved HVs of the encoded dataset. The decoding of the retrieved records could also be done using the unbinding operation on the retrieved record, using the column or attribute codebooks to identify the correct value. For example, if we retrieve the following encoded record from our VSADB using a query HV for the 'Repici' family in Query 1:

$$\mathbf{r} = (\mathbf{fname} \bigcirc \mathbf{jack}) + (\mathbf{lname} \bigcirc \mathbf{repici}) + (\mathbf{city} \bigcirc \mathbf{riverside}) + (\mathbf{state} \bigcirc \mathbf{nj}) + (\mathbf{gender} \bigcirc \mathbf{m})$$

If we would now like to identify the value of the 'fname' attribute for this retrieved record we could easily do this by unbinding using the atomic HV for the 'fname' column:

$$\begin{aligned}\mathbf{fname} \oslash \mathbf{r} &= \mathbf{fname} \oslash (\mathbf{fname} \bigcirc \mathbf{jack}) + \mathbf{fname} \oslash (\mathbf{lname} \bigcirc \mathbf{repici}) \\ &+ \mathbf{fname} \oslash (\mathbf{city} \bigcirc \mathbf{riverside}) + \mathbf{fname} \oslash (\mathbf{state} \bigcirc \mathbf{nj}) + \mathbf{fname} \oslash (\mathbf{gender} \bigcirc \mathbf{m}) \\ &= \mathbf{jack} + \mathit{noise} + \mathit{noise} + \mathit{noise} + \mathit{noise}\end{aligned}$$

We then compare this resulting unbound HV, to the value HVs in the 'fname' codebook and pick the most similar item. Note that the noise introduced here is not significant enough to lead to a false positive value. We can understand why this is the case by making the following simple analogy, in the previous equation we notice that the unbinding of the 'fname' atomic HV to an

encoded record leads to a HV that is similar to the original name value HV ('jack'), but with the introduction of 4 noise terms. The probability that we obtain the correct value from the 'fname' codebook is thus directly related to the bundling capacity, depicted in figure 2.6. Again this is correct because the resulting unbound HV is equivalent to the original HV for the 'jack' value bundled with 4 noise terms which can be seen as 4 atomic, random HVs that introduce noise. In figure 2.6 we observe that for a bundle size of 5 to 20, we obtain perfect bundling capacity for dimensionalities of 1024 upwards. For our specified dimensionality of 1500, there should thus be no problem in retrieving the correct value HV from a column's codebook given the noisy estimate of the value HV achieved by unbinding using the column atomic HV. The interested reader might notice that we already performed an experimental analysis identifying the noise introduced for a bundle of key-value bound HVs in the unbinding section of chapter 2, a review of this section might help with understanding the more practical explanation of this section.

To verify our idea we execute the earlier mentioned example for our experiment dataset, we thus perform a nearest-neighbour search on the 'fname' column codebook using the resultant HV from unbinding the encoded VSADB HV of the 'Jack Repici' record ($\mathbf{r}$) with the atomic HV of the 'fname' column, results can be found in table 4.5. Here we noted the top 10 most similar values ot the earlier mentioned unbound HV to identify the 'jack' value, we clearly see that by simply picking the most similar HV we correctly identify the value for 'fname' for the record. Similar results would be obtained to identify the values for the other columns.

Value	Normalized Hamming Similarity
Jack	0.695
akubhv	0.557
pxbfax	0.556
fpiwfa	0.554
kbcros	0.554
bqcfzr	0.552
fefeph	0.552
aeaszn	0.551
hiljkm	0.551
fsiwfa	0.550

Table 4.5: Identifying the correct fname value for our 'Jack Repici' record, table reports the fname column values and their corresponding normalized Hamming similarities to the unbound record using 'fname' column atomic HV.

If we now take all these conclusions into account we could opt for a VSADB where we do not keep the original DB records and only keep the encoded records and the column codebooks that were used to create these encoded records. Using these we would then have to decode a record by unbinding it with each atomic column HV separately and retrieving the most similar value HV for that column. This means we save space because we do not keep the original database, but it also increases the overhead as each of the relevant records retrieved from a query HV needs to be decoded.

We can conclude that we proposed an architecture to effectively encode and decode a tabular dataset, where each record has 5 to 20 attributes, using only 1500 dimensions of a BSC model HV. The end user should evaluate whether they use this encoded VSADB for indexing results of a nearest-neighbour search, identifying the relevant records using a query HV, and loading these from the original database, as we did for our experiments. Alternatively, we identified in this conclusion that the user could only store the encoded records and column codebooks. This allows for a saving in space but introduces the need to decode the obtained relevant records using the value codebooks for each column.

Either way, we have validated that using a dimensionality of 1500 and a threshold value based

on this dimensionality, the number of attributes, and a (high) pre-defined recall value to query the VSADB, we can obtain perfect recall, precision, and accuracy for point or partial match queries.

Future research
In this chapter we verified the validity of the current proposed architecture and implementation of the VSADB. Yet there is still room for improvement to extend this implementation.

Future research could focus on extending this approach to handle numerical and ordinal attributes while maintaining the robustness we achieved for our current implementation.

More specifically we could look into the integration of values that have an inherent similarity structure, where the HV embedding of these values inside a column respect this similarity structure. A straightforward example is the embedding of a numerical attribute such as age, where the HV for a lower age like 15 has a higher similarity with the HV for age 12 than it has with the HV for age 43. More specifically, we could look into the design of VSA embeddings that have a linear similarity structure, that decreases proportionally to the amount of discrete steps we move away from a target value.

Another valid point of research could be the support of more advanced or intricate querying operations like joins, aggregations, and selections based on inequalities rather than exact matches for attribute values. This last query type of course goes hand in hand with the suggestion to look into the support of ordinal data.

Chapter 5

Conclusion

For this thesis I researched the concept of Vector Symbolic Architectures (VSAs), a field that promises to bridge the gap between traditional symbolic and connectionist approaches to artificial intelligence.

The research began with an extensive literature review, as VSAs are a relatively new concept for which I had little to no prior knowledge. This began with going through an extensive curriculum consisting of 12 modules on VSAs provided by the Redwood Center for Theoretical Neuroscience and Berkeley Wireless Research Center at UC Berkeley during fall 2021 [1]. This provided me with insights into the basic operations and also provided me with research that was already done on some potential use cases. It thus laid the perfect foundation for conducting experiments to gain a deeper understanding of the properties and capabilities of VSAs for storing and retrieving information. This curriculum also sparked my interest as it became clear that VSAs lie at the center of my interests, combining ideas from AI and data representation, through a lens of statistics.

As I gained a better understanding of the inner workings of VSAs, I documented this knowledge in a concise introductory format for individuals with no prior knowledge of VSAs, as seen in chapter 2. This included an overview of the core principles of VSAs, followed by a comprehensive review of the fundamental operations and their respective properties, as well as a series of experiments to explore their capabilities for storing and retrieving information. These experiments helped to uncover the statistical properties of these operations.

Building on this foundation, the thesis delved into the practical applicability of VSAs by investigating two real-world use cases: the creation of meaningful word embeddings and the encoding or vectorization of tabular datasets.

Word Embeddings
For the first use case, detailed in chapter 3, I developed a new approach for creating meaningful word representations using VSAs. This method, based on word-document co-occurrence, was tested on a corpus of Wikipedia articles and compared to the established Latent Semantic Analysis (LSA) method.

The results showed that while VSA-based word embeddings were not able to surpass the performance of LSA on the WordSim353 validation dataset, they preserved the original semantic information of the term-document matrix more accurately. This led to the conclusion that the quality of the VSA word embeddings is highly dependent on the quality of the input term-document matrix, and that there is significant potential for future research into various data preprocessing methods and more advanced VSA learning strategies.

[1] https://www.hd-computing.com/course-computing-with-high-dimensional-vectors

VSA Databases
The second use case, explored in chapter 4, focused on creating a VSA Database (VSADB) for encoding, storing, and retrieving tabular data. The proposed VSADB architecture successfully demonstrated the ability to encode rows as HVs and perform efficient similarity-based retrieval using query HVs.

To avoid retrieving false positives I conducted a theoretical analysis to identify a threshold value for filtering query results, ensuring that only relevant records were retrieved. Here, my knowledge of statistics really proved to be useful.

It is important to state that there are still many caveats in the current, basic, implementation. For example, there is currently no direct support for more advanced queries that require joins or aggregations as we only support point or partial match queries, based only on the equalities of attribute values. More importantly, we made a significant assumption about our data that there is no inherent similarity structure between the values in a single column or for values across columns. We thus only support categorical data, for which all unique values are dissimilar, and not numerical or ordinal data.

General Reflection on Research
Based on the research in this thesis, I have successfully explored the capabilities and possibilities offered by VSAs.

First, I thoroughly examined and clarified the fundamental characteristics and properties of VSAs through various experiments. Second, I proposed algorithms and architectures for 2 practical use cases, generating word embeddings and, encoding and querying tabular data. I examined and verified their qualities and potential shortcomings through well-founded experimental analyses. Despite the identified shortcomings, the experimental and critical analyses provide a strong foundation for potential extensions of these implementations.

Personal Reflection
This research journey has also been a significant personal growth experience. I feel like the challenges I encountered in navigating a relatively new field, understanding complex concepts, and setting up experiments to validate hypotheses have strengthened my analytical thinking and research capabilities.

I was initially drawn to VSAs by my promotor due to their potential to enhance AI systems by combining the strengths of symbolic and connectionist approaches. However, as I delved deeper into the subject, I immediately noticed that the properties of VSAs are deeply rooted in statistical principles. That is, the most critical part of any VSA application, the creation of atomic HVs relies on specific probability distributions. Throughout the initial experiments, we uncovered that the operations and their capabilities are intricately linked to these distributions, and validated this idea for a specific model, the BSC model, in the mathematical analysis in section 4.3.

I can thus conclude that the subject of this thesis offered a perfect culmination of my masters degree, as it provided me with the chance to utilize the knowledge I gained through the supplementary courses I took on statistics. This knowledge could then be applied to two use cases that were rooted in the two key areas of my masters degree, AI and Data Management.

Lastly, I cannot conclude the personal reflection without mentioning the guidance that was provided to me by my promotor prof. dr. Stijn Vansummeren. He suggested this fascinating topic, made time for weekly meetings to go over my progress, and guided me in many parts of this journey. He provided me with constructive and concise feedback on my experiments and also on the written text of this thesis.

In conclusion, I am fulfilled and proud of the research I delivered and believe that this thesis opens up a world of opportunities for further exploration of the potential of Vector Symbolic Architectures.

Bibliography

[1] Philip Boucher. *Artificial Intelligence: How does it work, why does it matter, and what can we do about it?: Study*, pp. 1–17.

[2] Norvig Peter. *Artificial Intelligence: A modern approach, global edition.* Pearson Education Limited, 2021.

[3] Nicholas Wang et al. "Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline". In: Jan. 2004, pp. 61–. DOI: 10.1109/DSN.2004.1311877.

[4] Alexandra Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. "Estimating the Carbon Footprint of BLOOM, a 176B Parameter Language Model". In: abs/2211.02001 (Nov. 2022). DOI: 10.48550/arXiv.2211.02001.

[5] "A Survey on Hyperdimensional Computing aka Vector Symbolic Architectures, Part I: Models and Data Transformations". In: 55 (), pp. 1–40. DOI: 10.1145/3538531. URL: https://dl.acm.org/doi/10.1145/3538531.

[6] David E. Rumelhart, James L. McClelland, and PDP Research Group. "Parallel Distributed Processing, Volume 1: Explorations in the Microstructure of Cognition: Foundations." In: MIT Press, Mar. 1986, pp. 77–109. DOI: https://doi.org/10.7551/mitpress/5236.001.0001.

[7] Tony Plate. "Distributed Representations and Nested Compositional Structure". In: (1994).

[8] David E. Rumelhart, James L. McClelland, and PDP Research Group. "Parallel Distributed Processing, Volume 1: Explorations in the Microstructure of Cognition: Foundations." In: MIT Press, Mar. 1986, pp. 3–44. DOI: https://doi.org/10.7551/mitpress/5236.001.0001.

[9] P. D. Turney and P. Pantel. "From Frequency to Meaning: Vector Space Models of Semantics". In: *Journal of Artificial Intelligence Research* 37 (Feb. 2010), pp. 141–188. ISSN: 1076-9757. DOI: 10.1613/jair.2934. URL: http://dx.doi.org/10.1613/jair.2934.

[10] Artur d'Avila Garcez et al. "Neural-Symbolic Computing: An Effective Methodology for Principled Integration of Machine Learning and Reasoning". In: (2019). arXiv: 1905.06088 [cs.AI].

[11] Hiroshi Fujii et al. "Dynamical Cell Assembly Hypothesis — Theoretical Possibility of Spatio-temporal Coding in the Cortex". In: *Neural Networks* 9.8 (1996). Four Major Hypotheses in Neuroscience, pp. 1303–1350. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(96)00054-8. URL: https://www.sciencedirect.com/science/article/pii/S0893608096000548.

[12] Jerry A. Fodor and Zenon W. Pylyshyn. "Connectionism and cognitive architecture: A critical analysis". In: *Cognition* 28.1 (1988), pp. 3–71. ISSN: 0010-0277. DOI: https://doi.org/10.1016/0010-0277(88)90031-5. URL: https://www.sciencedirect.com/science/article/pii/0010027788900315.

[13] Ray Jackendoff. *Foundations of Language: Brain, Meaning, Grammar, Evolution.* Oxford University Press UK, 2002.

[14] Claude E. Shannon, Warren Weaver, and A. Chapanis. "The Mathematical Theory of Communication". In: 26.3 (Sept. 1951). DOI: 10.1086/398349.

[15] G. Salton, A. Wong, and C. S. Yang. "A vector space model for automatic indexing". In: *Commun. ACM* 18.11 (Nov. 1975), pp. 613–620. ISSN: 0001-0782. DOI: 10.1145/361219.361220. URL: https://doi.org/10.1145/361219.361220.

[16] Zellig S. Harris. "Distributional Structure". In: 10.2 (Jan. 1954). DOI: 10.1080/00437956.1954.11659520.

[17] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[18] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.

[19] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[20] Eneko Agirre et al. "A Study on Similarity and Relatedness Using Distributional and WordNet-based Approaches". In: *Proceedings of the 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL-2009)*. Boulder, Colorado, 2009, pp. 19–27. URL: http://alfonseca.org/pubs/2009-naacl-long.pdf.

[21] Pentti Kanerva. "Computing with High-Dimensional Vectors". In: 36.3 (June 2019). DOI: 10.1109/MDAT.2018.2890221.

[22] *Latent semantic analysis*. URL: https://en.wikipedia.org/w/index.php?title=Latent_semantic_analysis&oldid=1218342466.