# Introduction to Large Language Models (LLMs)

**Prof. Tanmoy Chakraborty, Prof. Soumen Chakraborti**
**Department of Computer Science & Engineering**
**Indian Institute of Technology, Delhi**

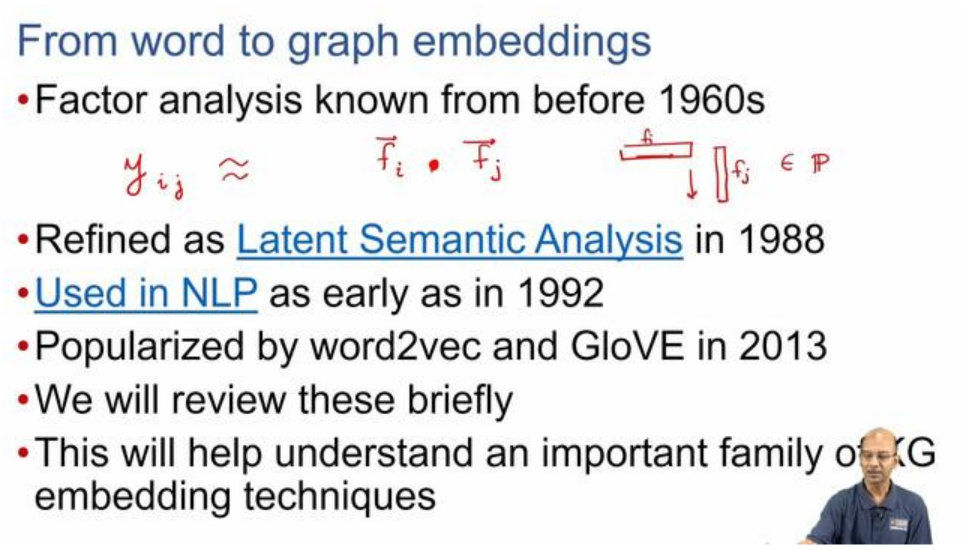
**Lecture 33**

## Knowledge and Retrieval: Multiplicative models

Welcome back. We recently reviewed translation and rotation-based models for knowledge graph representation. Recall that in translation models the subject is a point in space and the relation is a displacement that displaces the subject toward the object vector. In case of rotation, the subject may be simplified as a complex number with an angle and then the relation is a rotation to the object position. Today, we will cover a different family of knowledge graph embeddings.

They involve multiplicative or factor models. Factor analysis has been known from before 1960s. The basic idea is if you want to predict some property of two items i and j, let's call that property yij, I would like to model that as some featurization, let's call that f of i, and a featurization of j, this is a vector, that is a vector, and now we have an inner product between them. So, think of f of i as a row vector like this, and f of j as a column vector like this. If we multiply these together, we get a scalar number, and that's the property I'm trying to predict.

$$y_{ij} \approx \vec{f_i} \cdot \vec{f_j}$$



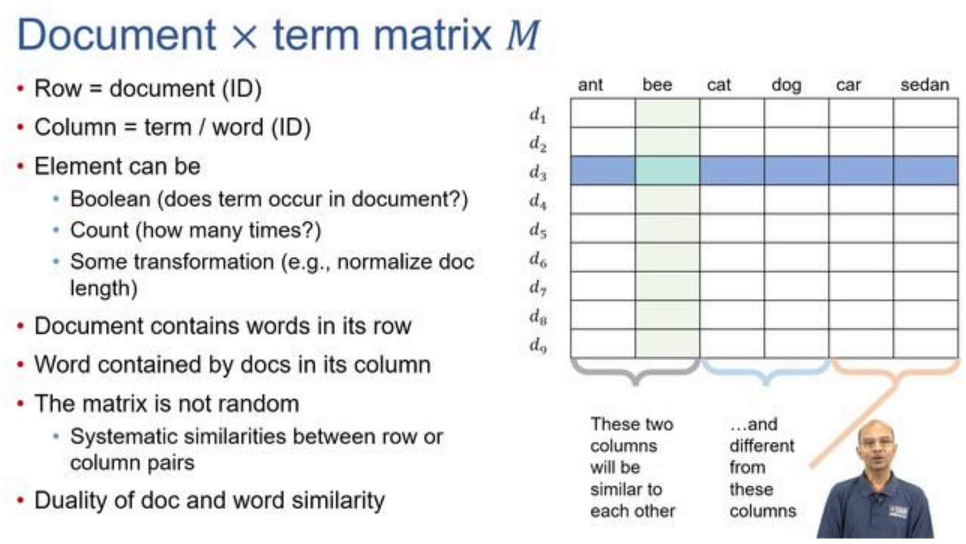
## From word to graph embeddings

- Factor analysis known from before 1960s

$y_{ij} \approx \vec{f}_i \bullet \vec{f}_j$ $\quad f_i \quad f_j \in \mathbb{P}$

- Refined as Latent Semantic Analysis in 1988
- Used in NLP as early as in 1992
- Popularized by word2vec and GloVE in 2013
- We will review these briefly
- This will help understand an important family of KG embedding techniques

This technique was refined as latent semantic analysis in 1988 and has been used in natural language processing as early as from 1992. This was resurfaced as word embeddings Word2vec and GloVe in 2013 and this has been covered earlier in our course. We'll still review this very briefly because it will help us understand this important family of knowledge graph embedding techniques. Consider a corpus represented as a document by term matrix. Documents go down rows and words are shown across the columns.



## Document × term matrix *M*

- Row = document (ID)
- Column = term / word (ID)
- Element can be
  - Boolean (does term occur in document?)
  - Count (how many times?)
  - Some transformation (e.g., normalize doc length)
- Document contains words in its row
- Word contained by docs in its column
- The matrix is not random
  - Systematic similarities between row or column pairs
- Duality of doc and word similarity

| | ant | bee | cat | dog | car | sedan |
|---|---|---|---|---|---|---|
| $d_1$ | | | | | | |
| $d_2$ | | | | | | |
| $d_3$ | | | | | | |
| $d_4$ | | | | | | |
| $d_5$ | | | | | | |
| $d_6$ | | | | | | |
| $d_7$ | | | | | | |
| $d_8$ | | | | | | |
| $d_9$ | | | | | | |

These two columns will be similar to each other
…and different from these columns

For example, ant might appear in document D1, in which case I might put a one over here. Ant may not appear in document two, in which case I may put a zero in there. That would be the Boolean representation of this corpus. Or I might say that ant appears five times in document D1, but only twice in document D4. That keeps more information in this matrix.

We could do more fancy things such as transform these numbers to say, normalize document length. So a document is represented by a row. This row tells you what words are contained in the document, but a column also represents a word. And if you read down the column, you get to know which documents contain that word. This matrix is very far from random.

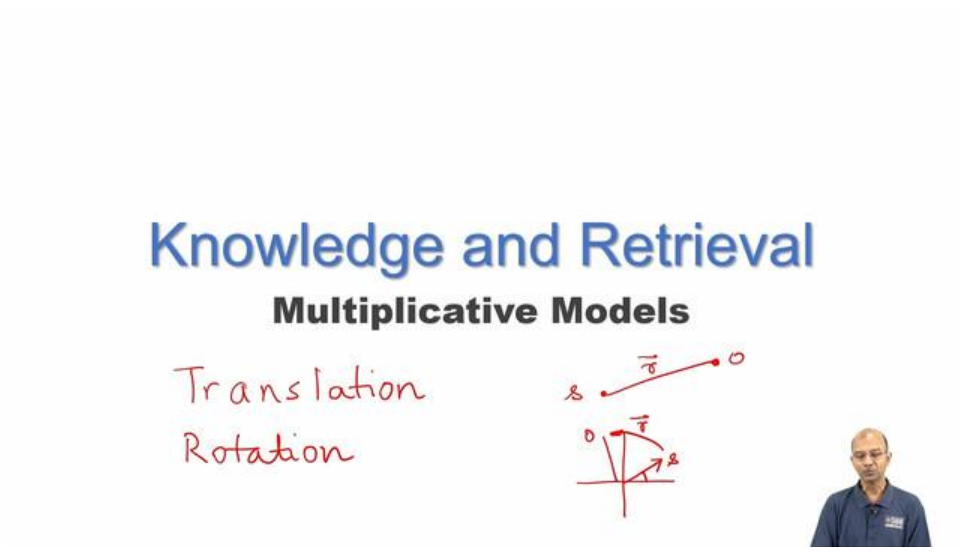
There has to be systematic similarities between row or column pairs. For example, we expect some reasonable similarity between cat and dog as columns, or car and sedan as columns. But we do not anticipate much similarity between the columns corresponding to, say, ant on one hand and car on the other hand. So there is this duality between documents and words in terms of similarity. When two documents are similar, they tend to contain or not contain similar words.

Conversely, two words are regarded as similar if similar documents contain them. This intuition was used in these factorization models. Any matrix M, such as the one we saw in the previous slide, can be factored into three parts. This is called a singular value decomposition, or SVD. Mij can thus be written as a sum over k, uik times sigma kk times vjk.

Knowledge and Retrieval

Multiplicative Models

So let me draw what I mean here. I goes across the rows of the first matrix U. k goes across the columns of the first matrix. The second matrix, sigma, is actually diagonal. Only the diagonals have non-zero values.



Knowledge and Retrieval

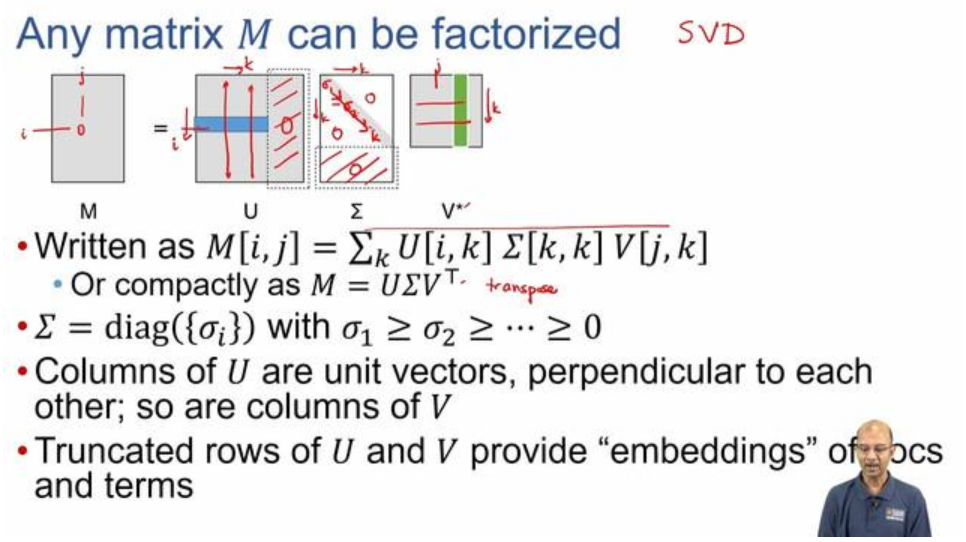
Multiplicative Models

The off diagonals are all zeros. And k simultaneously goes down rows and columns because it's diagonal. So I'm actually running down the diagonal here. And finally, if I'm trying to model the element at i and j, then I use the ith row here and the jth column in the last matrix, but I still run down rows indexed as k. That is the meaning of this three-way multiplication.

We can write it more compactly as m is equal to u as a matrix times sigma as a matrix times v transpose. So this symbol in both cases means transpose. I mentioned that sigma is a diagonal matrix. We can rearrange the rows and columns suitably so that the first element sigma one is greater than equal to the second element and so on. And all of them are non-negative.

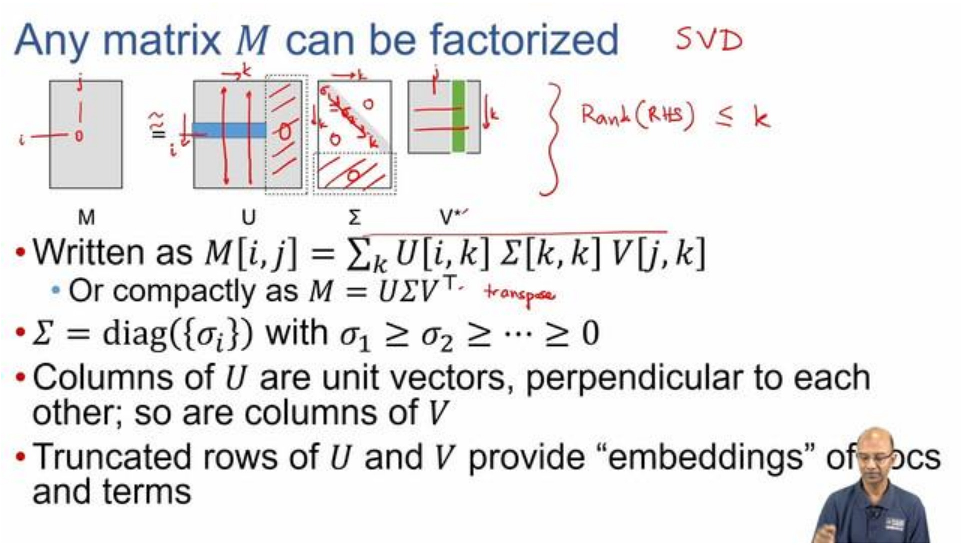
It's known that the SVD factorization can ensure that the columns of U are unit vectors. So every column has L2 norm equal to one. and they're perpendicular to each other. So if you pick two such vectors, their dot product is zero. And so are the columns of V or the rows of V transpose.

Now, you can see here that some part of the matrix is encased in dotted rectangles. This means that even if they were not zero to start with, I can replace them with zero chunks. And that will give me an approximation to the original matrix. If I start zeroing out more and more, this equality will be replaced by an inequality. Now observe that if I have only k rows and k columns and k diagonals, then the rank of the right hand side is at most k.



## Any matrix $M$ can be factorized SVD

- Written as $M[i,j] = \sum_k U[i,k]\,\Sigma[k,k]\,V[j,k]$
  - Or compactly as $M = U\Sigma V^{\top}$ transpose
- $\Sigma = \text{diag}(\{\sigma_i\})$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$
- Columns of $U$ are unit vectors, perpendicular to each other; so are columns of $V$
- Truncated rows of $U$ and $V$ provide "embeddings" of docs and terms

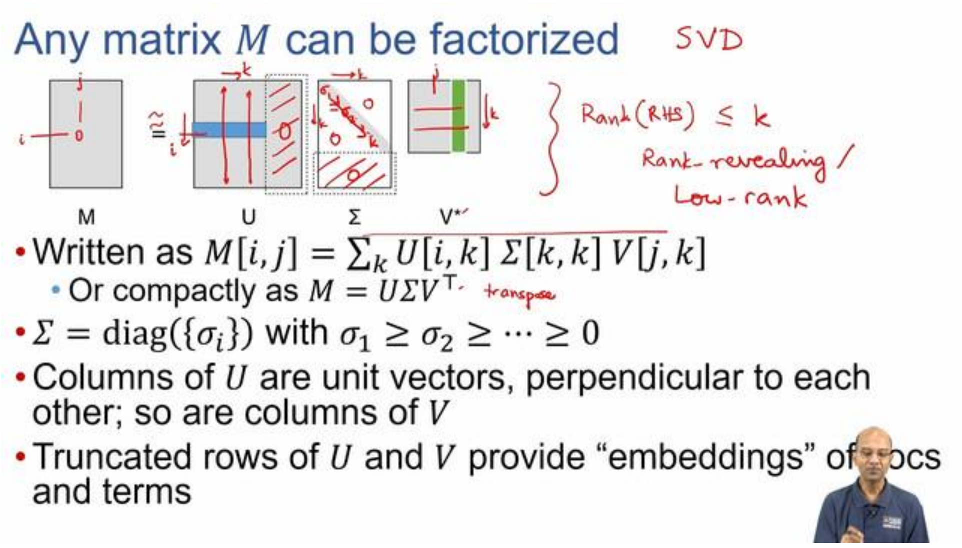


## Any matrix $M$ can be factorized

SVD

- Written as $M[i,j] = \sum_k U[i,k]\,\Sigma[k,k]\,V[j,k]$
  - Or compactly as $M = U\Sigma V^{\top}$ transpose
- $\Sigma = \text{diag}(\{\sigma_i\})$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$
- Columns of $U$ are unit vectors, perpendicular to each other; so are columns of $V$
- Truncated rows of $U$ and $V$ provide "embeddings" of docs and terms

So that is why SVD is also called a rank revealing or a low rank factorization of the original matrix. Even if the original matrix had full rank by first factorizing it through SVD and then selectively dropping these low signal parts of the matrix, the largest sigmas are taken care of. After some point, I start truncating it. That gives me a good approximation to the original matrix. When I do that, these truncated rows of U give me approximate representations of documents.



## Any matrix $M$ can be factorized

SVD

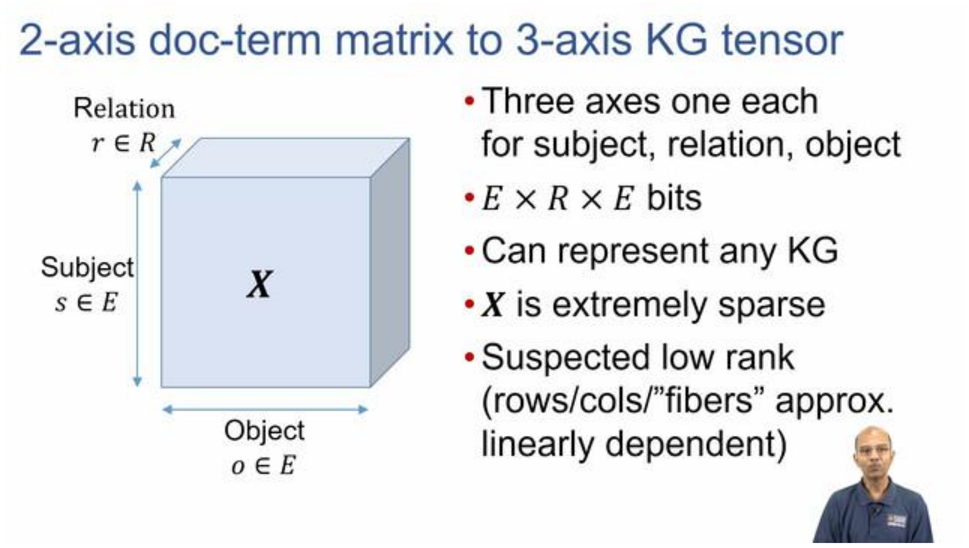
- Written as $M[i,j] = \sum_k U[i,k]\,\Sigma[k,k]\,V[j,k]$
  - Or compactly as $M = U\Sigma V^{\top}$ transpose
- $\Sigma = \text{diag}(\{\sigma_i\})$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$
- Columns of $U$ are unit vectors, perpendicular to each other; so are columns of $V$
- Truncated rows of $U$ and $V$ provide "embeddings" of docs and terms

And the truncated columns of V transpose gives me approximate representation of words. In fact, it has been shown that Word2Vec finds word vectors which are approximately these columns. So hereafter, we will keep this matrix factorization in mind. What do we say is the similarity between two word vectors? Suppose I want to find in this original matrix before factorization the similarity between the word cat and the word dog. To do that, I may take the original corpus matrix and transpose it.

So this is M transpose. Documents now go across the columns and words go across the rows and my focus row is the row of cat. I take another copy of the original matrix M and I focus on the column for dog and then I multiply those together. which means this first value for d1 times the first value for d1 plus etc. to the end.

What does this give me? So that is the quantity m transpose m at ij where i is for cat and j is for dog. And it's easy to see that the quantity M transpose M ij is the number of documents that contain both words i and j  in the form where the matrices are Boolean. Similar extended interpretations are possible for other elements that we can fill into these matrices. Now, if we go back to the earlier slide, if I instead take the column of cat and the column of dog and I find their inner product, that is an approximation to taking the column of cat and the column of dog over here and finding their inner product. In fact, it's known that SVD has noise reduction capabilities and vi star, which is this row vector,  dot vj star, which is this column, gives a smooth similarity score between these words i and j.

We now extend from this act of two dimensional factorization to three dimensions because that is how our knowledge graph is going to be encoded. Remember that the knowledge graph consists of triples, subject, relation and object. We plot the subject along the first axis over here we plot the object on the second axis over there and the third axis which goes into the plane of the paper is the relation. So now there are three axis instead of two. And this matrix is also a Boolean matrix.



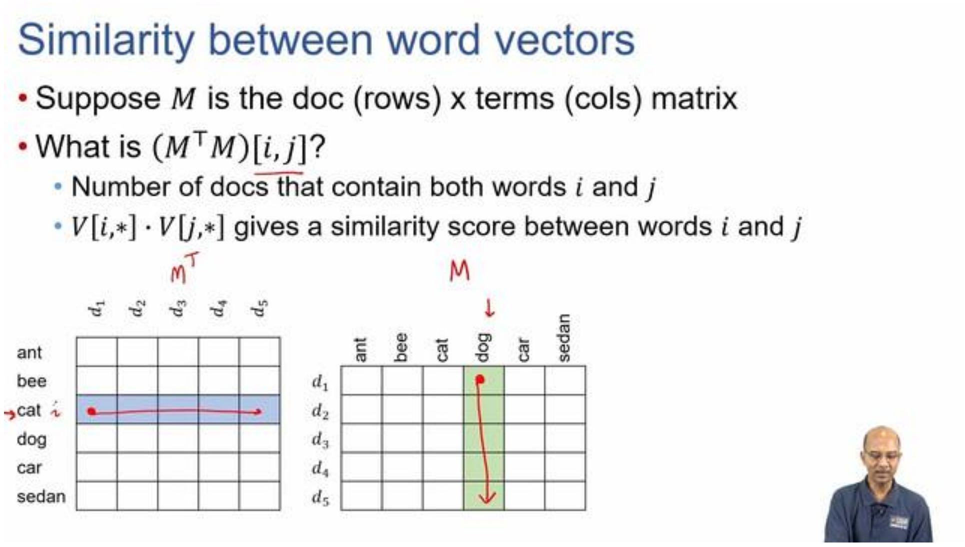
## 2-axis doc-term matrix to 3-axis KG tensor

- Three axes one each for subject, relation, object
- $E \times R \times E$ bits
- Can represent any KG
- $\boldsymbol{X}$ is extremely sparse
- Suspected low rank (rows/cols/"fibers" approx. linearly dependent)

We set the element of X S O R equal to one if SRO is in the knowledge graph and 0 otherwise. So you can see that this E times R times E bits can represent any knowledge graph in the world. Any bit pattern can fit inside this 3-axis kg tensor. X is extremely sparse. Of all possible facts which can hold in the universe, very few actually do.

People are only born in one place, a country typically has only one capital, a company only has one CEO. We also suspect that this tensor X has low rank just like our document by word matrix has low rank and for much the same reasons that there are very similar entities and there are very similar relations. For example, born in and citizen of are very strongly correlated relations. Only very few people statistically change citizenship. If you are a CEO of a company, you're most definitely an employee of the company.

So that's an implication relation. For all these reasons, we expect this tensor X to have low rank, meaning that there will be a lot of linearly dependent rows or columns or fibers. In case of matrices, various chunks and slices of the matrix can be called as rows and columns. Now there are two kinds of things in a 3D tensor. There are the edges and then there are the planes which cut through the tensor and we expect many of them to be linearly dependent on each other. So therefore, let's try to factor this three-axis kg tensor in the same style of singular value decomposition. Again, we will have three tensors being multiplied together to form that original three-axis tensor. However, we are going to simplify two of those tensors to actually be matrices. and in fact, the same matrix.



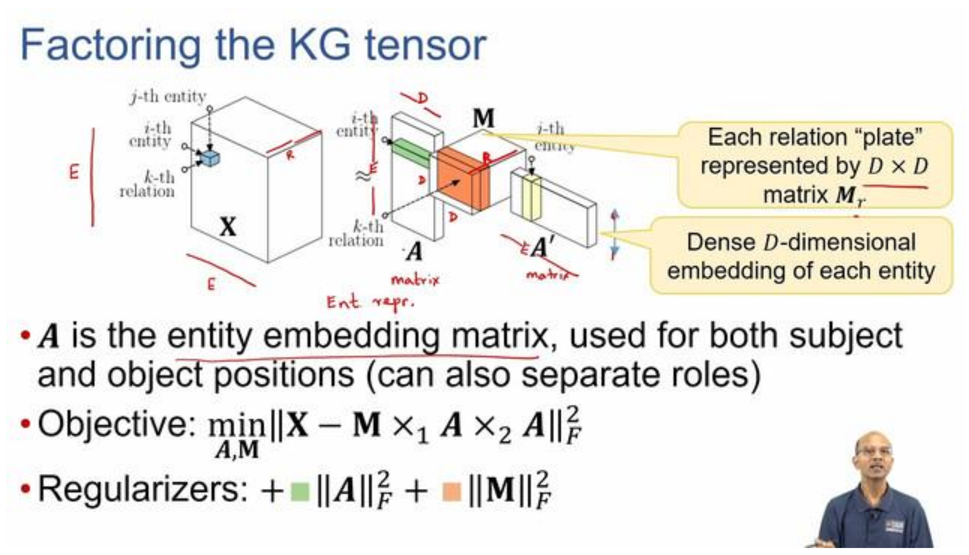
## Similarity between word vectors

- Suppose $M$ is the doc (rows) x terms (cols) matrix
- What is $(M^\top M)[i,j]$?
  - Number of docs that contain both words $i$ and $j$
  - $V[i,*] \cdot V[j,*]$ gives a similarity score between words $i$ and $j$

$M^T$
$d_1$ $d_2$ $d_3$ $d_4$ $d_5$
ant
bee
cat $i$
dog
car
sedan
M
ant bee cat dog car sedan
$d_1$ $d_2$ $d_3$ $d_4$ $d_5$

In particular, this matrix will give us the entity representations or the entity embedding matrix, which has only two axes. So let's look carefully at the sizes and shapes of these tensors. As in the previous slide, the ith entity or the subject, this can take values in the whole entity space. the jth entity or the object can take any of E values again and the third axis has any of R relations. So it's an E by E by R tensor.

On the other hand, the entity embedding matrix still has a height of E because I have to represent all the E entities and the same over here. But its height which is the same as the width of A over here is some arbitrary dimensions D which will be the width of our entity embeddings. Now if there are R relations the depth of the M tensor in between is still going to be R but on the other two sides I'm just going to have D by D. So in other words each relationship R is represented by a D times D matrix which we might call the Rth slice of the matrix R written as M subscript R. Generally speaking we expect A and A prime to be dense D dimensional embeddings of each entity just like we had dense D dimensional embedding for each word in case of singular value decomposition or word2vec.


j-th entity
i-th entity
k-th relation
X
E
R
E
i-th entity
k-th relation
M
D
D
D
R
E
A
matrix
Ent. repr.
j-th entity
A′
E
matrix
Each relation "plate" represented by $D \times D$ matrix $\boldsymbol{M}_r$
Dense $D$-dimensional embedding of each entity

## Factoring the KG tensor

- $\boldsymbol{A}$ is the entity embedding matrix, used for both subject and object positions (can also separate roles)
- Objective: $\min_{\boldsymbol{A},\mathbf{M}} \|\mathbf{X} - \mathbf{M} \times_1 \boldsymbol{A} \times_2 \boldsymbol{A}\|_F^2$
- Regularizers: + 🟩 $\|\boldsymbol{A}\|_F^2$ + 🟧 $\|\mathbf{M}\|_F^2$

Now our objective in this factorization is that X be approximately equal to this three tensor product on the right hand side. In other words, I want to find the entity embeddings A and the relation tensor M optimally so that the difference between the original matrix X and the approximation computed by this three-way tensor product is minimized. Now what is this difference? This is the tensor difference which is element by element. So it will be X of SOR minus the counterpart in this product tensor. And then we take a square error over the two sides and then we add up over SOR.

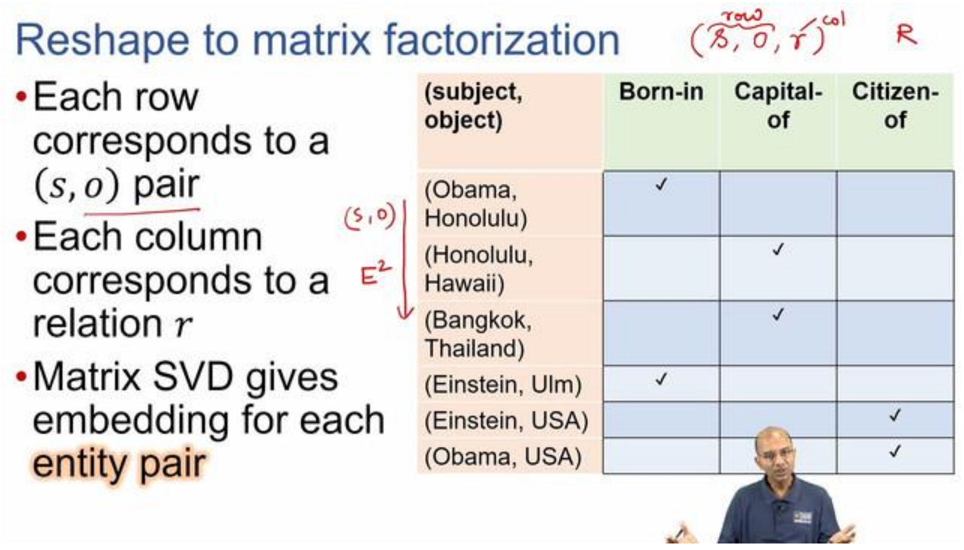
That's sometimes called the Frobenius norm. and we would like to minimize that with regard to the choice of A and M. To keep the model well conditioned, sometimes regularizers are added, meaning that to that loss objective in the previous line, we add the Frobenius norm of A itself and the Frobenius norm of M itself, so that the elements in A and M remain under check and their magnitude cannot grow arbitrarily large. This still is a relatively simple optimization, although it is neither exact nor as well behaved as singular value decomposition. In fact, the common way to optimize is to do it in alternating sub-steps.

We first fix M for the moment and we improve A via gradient descent, which is easy to compute. And then in the next alternating sub-step, we fix A for the time being and we improve M using gradient descent. Unlike singular value decomposition, a unique optimal solution is not guaranteed. We can only get to a local optimum. And the process is also slower than matrix factorization, which is highly optimized.

To address these difficulties, two main approaches are known. The first is to somehow convert this three-axis tensor factorization into a two-axis matrix factorization and then apply singular value decomposition. The other approach is to take our main troublemaker, the tensor M in between, and restrict it to occupy only simpler forms. And then the optimization can again become easier. The first technique.

Suppose we want to reshape tensor factorization into an act of matrix factorization. Remember that we start with SOR we group S and O together into a pair and make that the row. So Obama and Honolulu is a pair. Honolulu and Hawaii is a pair. Obama and Nigeria is a pair.

So every possible entity pair appears here in principle. And every column is a relation. So born in is a column, capital of is a column, citizen of is a column and so on for all the R relations. So you could see that the number of rows in principle is E squared and the number of columns is of course R. What is in the matrix? Just bits, boolean.



## Reshape to matrix factorization

$(\overset{row}{\overbrace{s, o}}, \overset{col}{r})$ $R$

- Each row corresponds to a $(s, o)$ pair
- Each column corresponds to a relation $r$
- Matrix SVD gives embedding for each entity pair

$(s,o)$ $E^2$

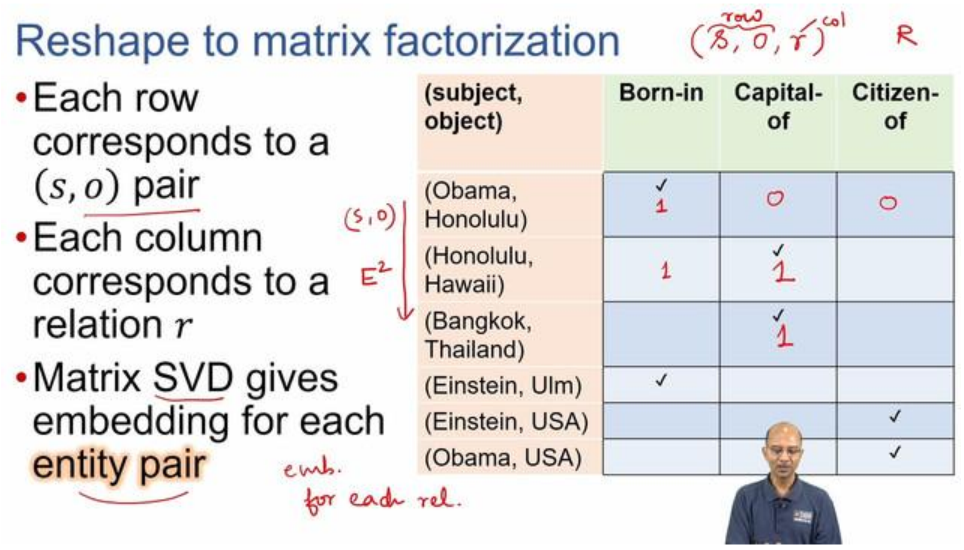
| (subject, object) | Born-in | Capital-of | Citizen-of |
|---|---|---|---|
| (Obama, Honolulu) | ✓ | | |
| (Honolulu, Hawaii) | | ✓ | |
| (Bangkok, Thailand) | | ✓ | |
| (Einstein, Ulm) | ✓ | | |
| (Einstein, USA) | | | ✓ |
| (Obama, USA) | | | ✓ |

Obama was born in Honolulu, therefore born in column has a one or a tick in that row. But Obama is not the capital of Honolulu. Therefore, this is 0. Obama is not a citizen of Honolulu. Citizenship is only defined on nations.

Therefore, this is also 0. Honolulu is capital of Hawaii. But Honolulu was not born in Hawaii in an unusual sense. Bangkok is the capital of Thailand. So that's how these bits are filled out.

Once again, you see that this matrix is likely to be low rank. because similar relations like born-in and citizen-of are going to have similar columns. And this is exploited by taking this matrix now and subjecting it to singular value decomposition. This gives us two sets of embeddings. One is an embedding for each entity pair and the other is an embedding for each relation as before.

This method has some disadvantages in the sense that we only get an embedding for each entity pair, but not individual entities. So if tomorrow after training this system, I wanted to predict the property of an unknown new entity coming in, I cannot form an individual vector for it and then compare its vector against the vectors of known entities. I really need all the entity pairs to be available ahead of time before the factorization is attempted. The second technique you might recall is to simplify the relation tensor M. Remember our diagram where A was here, A transpose was there and this was the troublemaking three axis tensor M.



## Reshape to matrix factorization

$(\overset{\text{row}}{\overbrace{s, o}}, r)^{\text{col}}$ R

- Each row corresponds to a $(s, o)$ pair
- Each column corresponds to a relation $r$
- Matrix SVD gives embedding for each entity pair

$(s,o)$ $E^2$

| (subject, object) | Born-in | Capital-of | Citizen-of |
|---|---|---|---|
| (Obama, Honolulu) | ✓ 1 | 0 | 0 |
| (Honolulu, Hawaii) | 1 | ✓ 1 | |
| (Bangkok, Thailand) | | ✓ 1 | |
| (Einstein, Ulm) | ✓ | | |
| (Einstein, USA) | | | ✓ |
| (Obama, USA) | | | ✓ |

emb. for each rel.

Now our approximation can also be written as X of SRO or OR depending on the convention of the axis ordering was approximately the embedding for the subject times the embedding of the object but mediated by the r-th relations plate M R. So see what's happening if I multiply these three tensors together. Every plate in M corresponds to one relation and the score for the truth or falsity of that fact involving R is given by this three-way vector matrix vector product. Observe that the overall vector matrix vector product is actually just a real number. A shorthand for this, if the matrix M was made diagonal, suppose all the off diagonals were made zero, then it's easy to see that our approximation is further simplified to X SRO is approximately equal to, as I run over these dimensions, I am basically rolling down this row, rolling across this diagonal and rolling down this column.

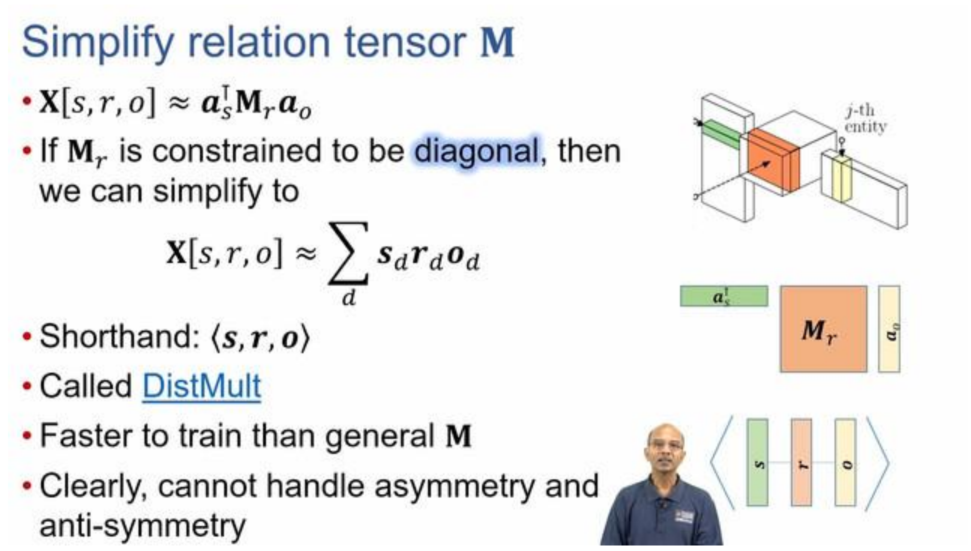
So it's sum over D, S, D, R, D, O, D, where those are their embeddings. So we'll shorthand this to be inner product of S, R, O, a three-way inner product or dot product. This is called distmult for multiplicative model over distributed representations. This is of course much faster to train than general M's because if the general M had D squared dimensions or capital D squared dimensions, only D of them are non-zero. Everything else is zero.

So the complexity has decreased quite substantially and the system is easier to train. However, observe that SRO is the same as ORS. It's just a multiplication which is symmetric. You can reorder these three terms any which way you like. So it clearly cannot handle asymmetric relations and antisymmetric relations.

What's an asymmetric relation? If a person follows another person on social media, if A follows B, it does not necessarily follow that B follows A. B might follow A or B might not follow A. So following is not asymmetric relation. And there are anti-symmetric relations where if subject relation object holds, then object relation subject most definitely does not hold. For example, S is the father of O, when certainly O cannot be the father of S.

So there's asymmetry and anti-symmetry and DISTMULT cannot handle either of them. So that's a limitation. This can, however, be fixed. But before we get into that, DISTMULT has actually been surprisingly effective on the standard benchmarks that we mentioned in a previous part of this lecture. But has DISTMULT been sufficiently stress tested? A paper in later days claimed that that was not the case.

They claimed that the good performance of DistMult on datasets that are themselves asymmetric or non-symmetric, such as FB15K or WN18, is surprising. Now, then they observed that these datasets, when they evaluate the query SR question mark, they rarely also evaluate in the test set the query ?, r and s for a non-symmetric predicate. In fact, in FB15K dataset, these kinds of challenge or problematic queries make up only 4% of the test set. And that's why the failings of DISCMULT never really float up to aggregate numbers. How do we fix it? There's actually a very simple way to fix it, and it's called simple.

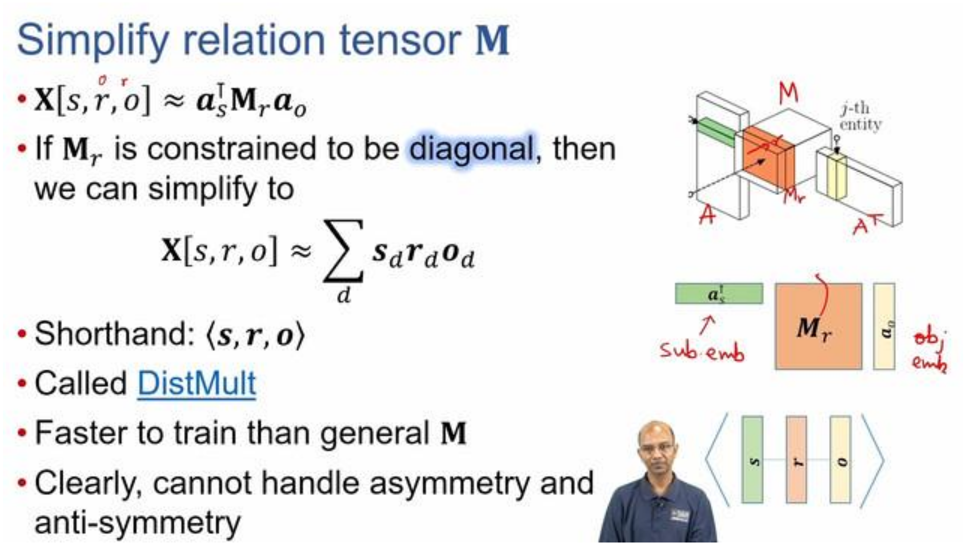


## Simplify relation tensor $\mathbf{M}$

- $\mathbf{X}[s,r,o] \approx \boldsymbol{a}_s^\top \mathbf{M}_r \boldsymbol{a}_o$
- If $\mathbf{M}_r$ is constrained to be diagonal, then we can simplify to

$$\mathbf{X}[s,r,o] \approx \sum_d \boldsymbol{s}_d \boldsymbol{r}_d \boldsymbol{o}_d$$

- Shorthand: $\langle \boldsymbol{s}, \boldsymbol{r}, \boldsymbol{o} \rangle$
- Called DistMult
- Faster to train than general $\mathbf{M}$
- Clearly, cannot handle asymmetry and anti-symmetry



## Simplify relation tensor $\mathbf{M}$

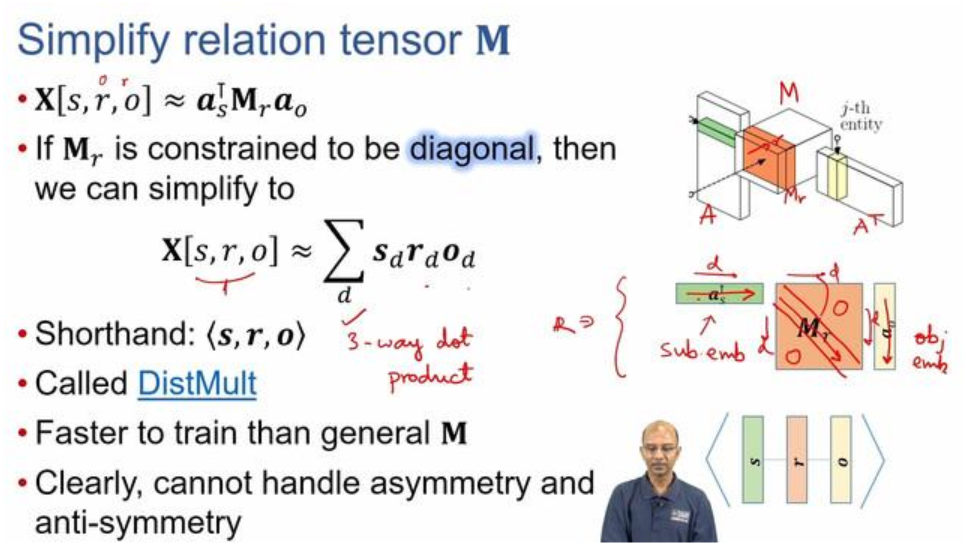
- $\mathbf{X}[\overset{o}{s}, \overset{r}{r}, o] \approx \boldsymbol{a}_s^\top \mathbf{M}_r \boldsymbol{a}_o$
- If $\mathbf{M}_r$ is constrained to be diagonal, then we can simplify to

$$\mathbf{X}[s, r, o] \approx \sum_d \boldsymbol{s}_d \boldsymbol{r}_d \boldsymbol{o}_d$$

- Shorthand: $\langle \boldsymbol{s}, \boldsymbol{r}, \boldsymbol{o} \rangle$
- Called DistMult
- Faster to train than general $\mathbf{M}$
- Clearly, cannot handle asymmetry and anti-symmetry

To address asymmetry or anti-symmetry, each entity now, instead of getting just one vector, gets two associated vectors, sub-e and obj, so subject of e and object of e, depending on what role the entity e plays in a fact. If it appears in the subject position, we'd have to use sub-e. If it appears in the object position, then we have to use obj-e. For each relation R in the database, we automatically introduce the inverse relation R minus unless it's explicitly provided. For example, if we have city is capital of nation is already part of the kg, then we automatically introduce the inverse relation nation has capital city.

So if this is R, that is R inverse. We don't need to worry too much about whether R inverse has already been explicitly provided or not. If accidentally the KG already has an R prime, which is secretly the same as R inverse, all that will happen is that their embeddings will evolve toward the same embedding or representation. So we can always feel free to even redundantly introduce the inverse relation into the database. Finally, the score of a fact.



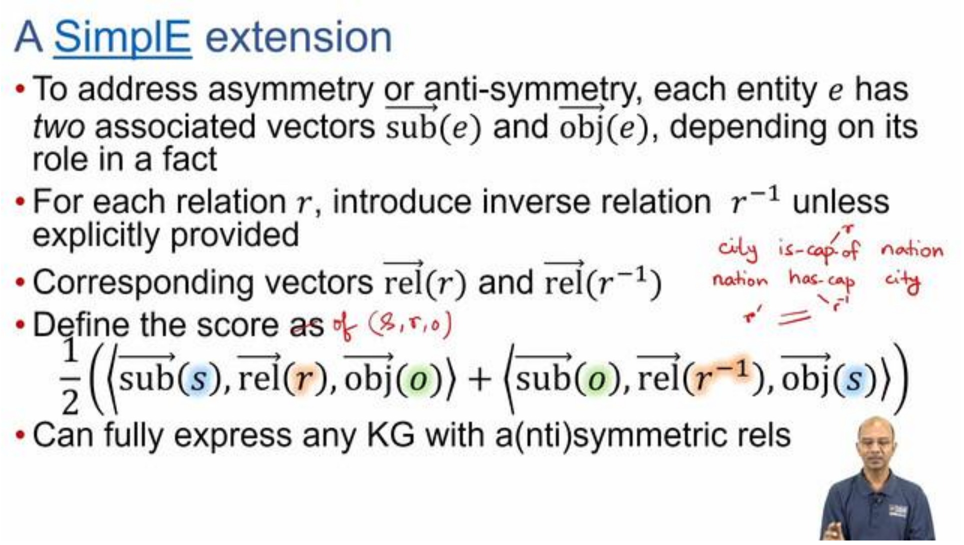
## Simplify relation tensor $\mathbf{M}$

- $\mathbf{X}[s,\overset{o}{r},\overset{r}{o}] \approx \boldsymbol{a}_s^\top \mathbf{M}_r \boldsymbol{a}_o$
- If $\mathbf{M}_r$ is constrained to be diagonal, then we can simplify to

$$\underbrace{\mathbf{X}[s,r,o]} \approx \sum_d \boldsymbol{s}_d \boldsymbol{r}_d \boldsymbol{o}_d$$

- Shorthand: $\langle \boldsymbol{s}, \boldsymbol{r}, \boldsymbol{o} \rangle$ ✓ 3-way dot product
- Called DistMult
- Faster to train than general $\mathbf{M}$
- Clearly, cannot handle asymmetry and anti-symmetry

is now expressed as the average of two scores. The forward score, where we take the subject vector of S, the vector for the relation, and the object vector of O, and we multiply all of them together using the formula in the last slide, and then we average that with the converse. We take the inverse relation, we flip the subject and the object, and also score that. And we take the average of these two vectors. It is possible to convince ourselves that this formulation can fully express any knowledge graph which may have symmetric relations, asymmetric relations, and anti-symmetric relations in the same knowledge graph.



## A SimplE extension

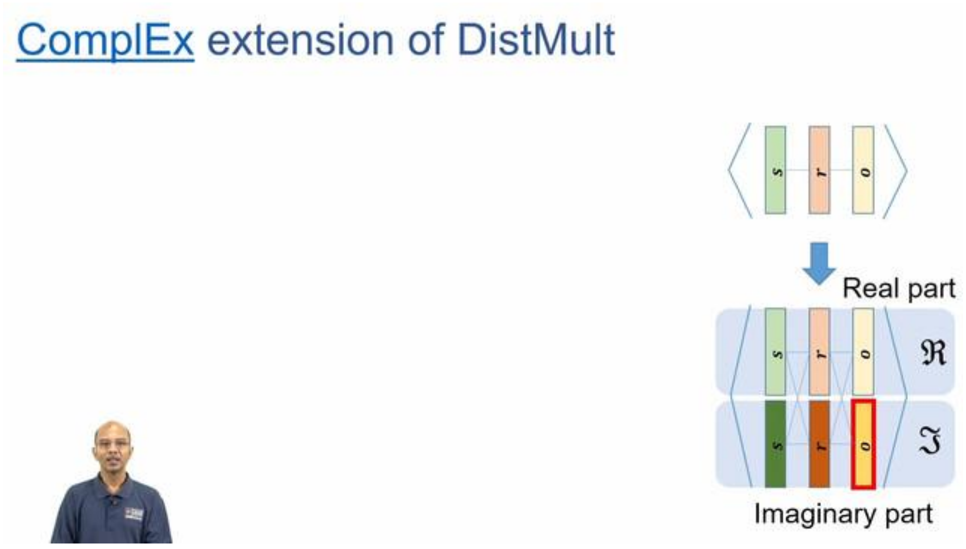
- To address asymmetry or anti-symmetry, each entity $e$ has *two* associated vectors $\overrightarrow{\mathrm{sub}}(e)$ and $\overrightarrow{\mathrm{obj}}(e)$, depending on its role in a fact
- For each relation $r$, introduce inverse relation $r^{-1}$ unless explicitly provided
- Corresponding vectors $\overrightarrow{\mathrm{rel}}(r)$ and $\overrightarrow{\mathrm{rel}}(r^{-1})$
- Define the score as of $(s,r,o)$

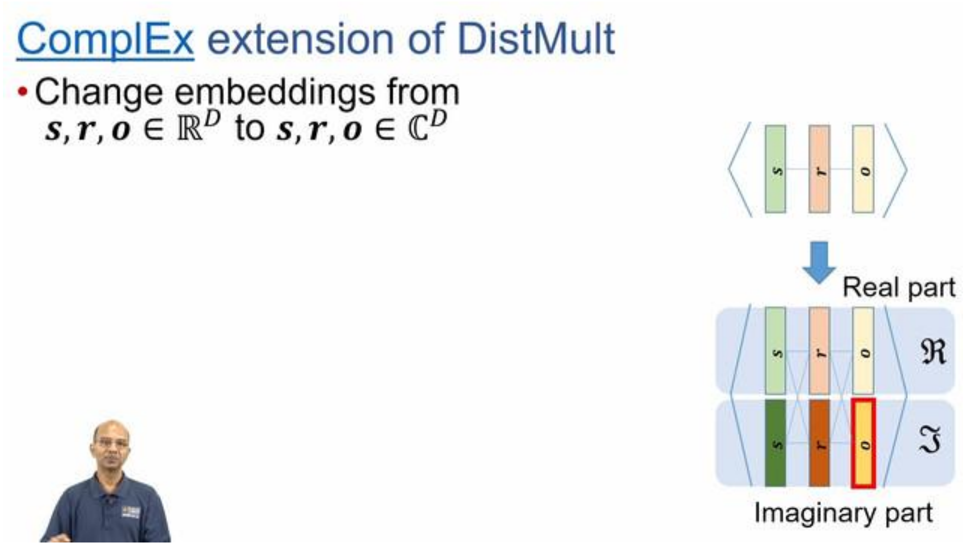
$$\frac{1}{2}\Big(\big\langle \overrightarrow{\mathrm{sub}}(s), \overrightarrow{\mathrm{rel}}(r), \overrightarrow{\mathrm{obj}}(o) \big\rangle + \big\langle \overrightarrow{\mathrm{sub}}(o), \overrightarrow{\mathrm{rel}}(r^{-1}), \overrightarrow{\mathrm{obj}}(s) \big\rangle\Big)$$

- Can fully express any KG with a(nti)symmetric rels

city $\overset{r}{\text{is-cap-of}}$ nation
nation has-cap city
$r' = r^{-1}$

By choosing suitable forms and formats of subject and relation and object vectors, I can always represent this kg provided D is large enough. Now, there is another way to look at this kind of extensions of distmult and that's called complex. Complex, as the name might suggest, uses complex numbers just like rotateE did in a previous part of this lecture. In distmult, our subject entity was turned into a real vector. and so was the relation on the object entity and remember this inner product really meant I took the first element, I multiplied it by the first element, I multiplied by this first element, then I added on the product of the second row and so on.


## ComplEx extension of DistMult

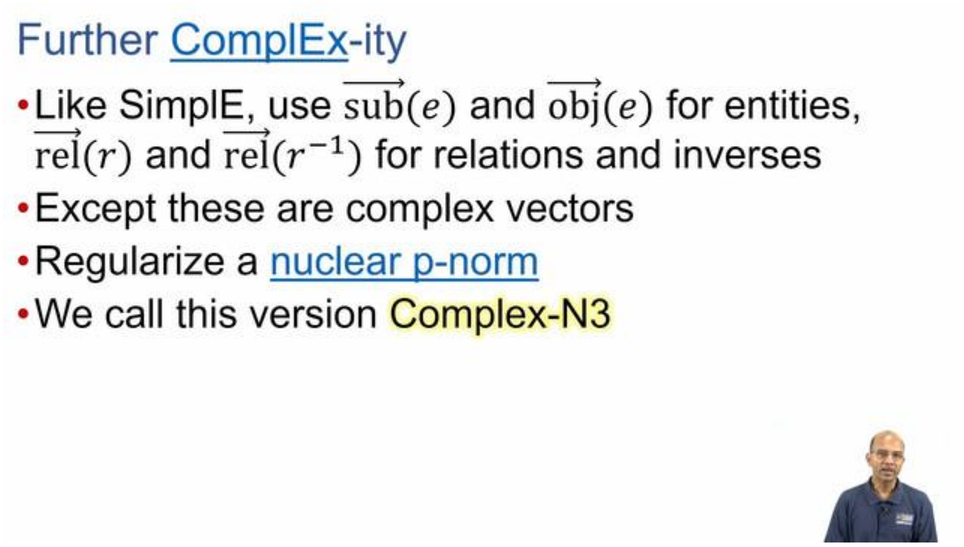

## ComplEx extension of DistMult

- Change embeddings from $\boldsymbol{s}, \boldsymbol{r}, \boldsymbol{o} \in \mathbb{R}^D$ to $\boldsymbol{s}, \boldsymbol{r}, \boldsymbol{o} \in \mathbb{C}^D$

That is now going to change because subject, relation and object are now going to become complex numbers where I've shown their real part in the upper rectangle and their imaginary parts in the lower rectangle. However, if you think of the formula through which complex multiplication happens, their real and imaginary parts are going to be mixed up in all kinds of ways to arrive at the final three-way inner product of complex vectors. Now, the three-way inner product is in general a complex number. So to turn that into something like a real score, I need to take the real part. And that's the symbol for taking the real part of the inner product of subject relation and object.

Also observe that the object is not taken as is, but it's taken with a star. Here, O star is the complex conjugate of O. which means that if o was a plus j times b, where j is square root of minus 1, then the conjugate is a minus square root of minus 1 times b. If a relation is symmetric, we can always model this as before, doing nothing more than dissmult by choosing the imaginary part of r to be 0. and that then reduces to dismult.

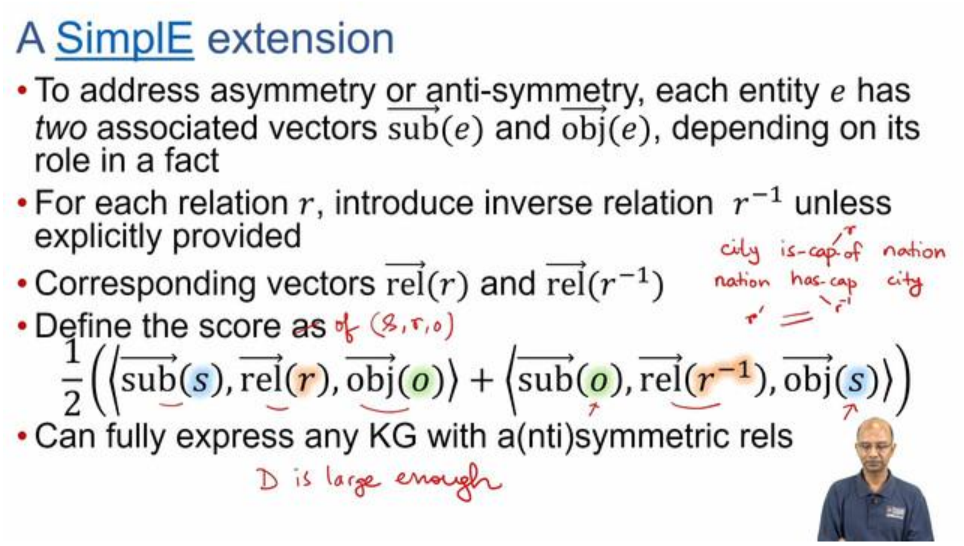
On the other hand, if a relation is anti-symmetric, it can be shown that we want to set the real part of R to zero. This ensures that SRO star real part is minus real part of ORS star if you flip the arguments, the score negates, which means if the original fact has a high score, the swapped fact will have a low score and vice versa. And that will take care of anti-symmetry for us. We can combine the ideas in simple E and complex. Like simple E, we can use two representation vectors for each entity, subject and object of E, and we can also add R inverse for every relation R given to us, except that these sets of vectors, subject E, object E, and the relation vector for R and the relation vector for R inverse will all become complex vectors instead of real vectors.



## Further ComplEx-ity

- Like SimplE, use $\overrightarrow{\mathrm{sub}}(e)$ and $\overrightarrow{\mathrm{obj}}(e)$ for entities, $\overrightarrow{\mathrm{rel}}(r)$ and $\overrightarrow{\mathrm{rel}}(r^{-1})$ for relations and inverses
- Except these are complex vectors
- Regularize a nuclear p-norm
- We call this version Complex-N3

Furthermore, although I don't have time to get into details, the nuclear p-norm of these vectors will be regularized. If that p-norm increases, that will add to our loss, unlike the standard L2 or Frobenius loss in earlier parts of this discussion. This version of complex is called complex N3 and is among the most competitive knowledge graph completion algorithms. We already discussed some of the popular knowledge graph completion benchmarks previously. They are WN18 or WordNet 18 relations, FB15K from Freebase with 15,000 entities and YAGO310 from the YAGO Knowledge Graph.



## A SimplE extension

- To address asymmetry or anti-symmetry, each entity $e$ has *two* associated vectors $\overrightarrow{\mathrm{sub}}(e)$ and $\overrightarrow{\mathrm{obj}}(e)$, depending on its role in a fact
- For each relation $r$, introduce inverse relation $r^{-1}$ unless explicitly provided
- Corresponding vectors $\overrightarrow{\mathrm{rel}}(r)$ and $\overrightarrow{\mathrm{rel}}(r^{-1})$
- Define the score ~~as~~ of $(s, r, o)$
$$\frac{1}{2}\left(\left\langle \overrightarrow{\mathrm{sub}}(s), \overrightarrow{\mathrm{rel}}(r), \overrightarrow{\mathrm{obj}}(o)\right\rangle + \left\langle \overrightarrow{\mathrm{sub}}(o), \overrightarrow{\mathrm{rel}}(r^{-1}), \overrightarrow{\mathrm{obj}}(s)\right\rangle\right)$$
- Can fully express any KG with a(nti)symmetric rels

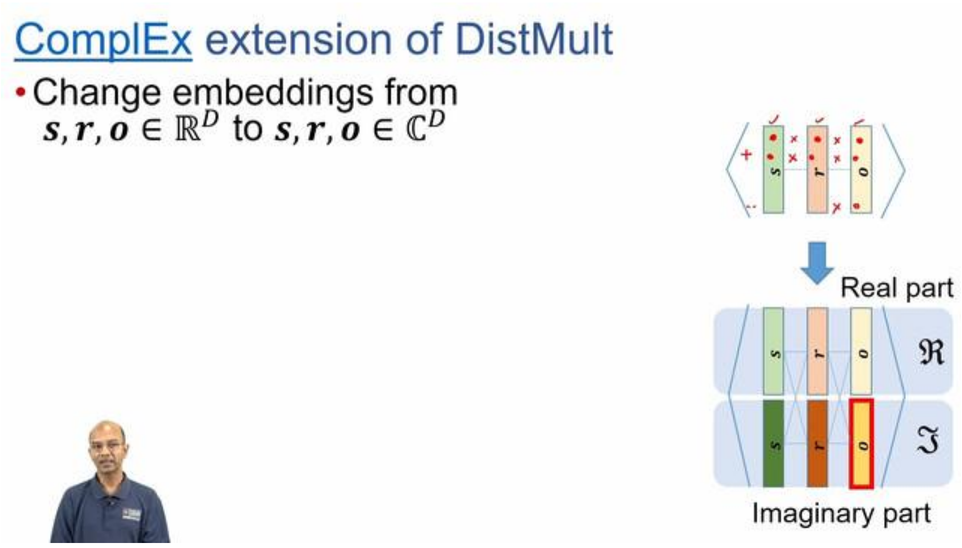
D is large enough

city $\overset{r}{\text{is-cap-of}}$ nation

nation $\underset{r^{-1}}{\text{has-cap}}$ city

$r' = r^{-1}$

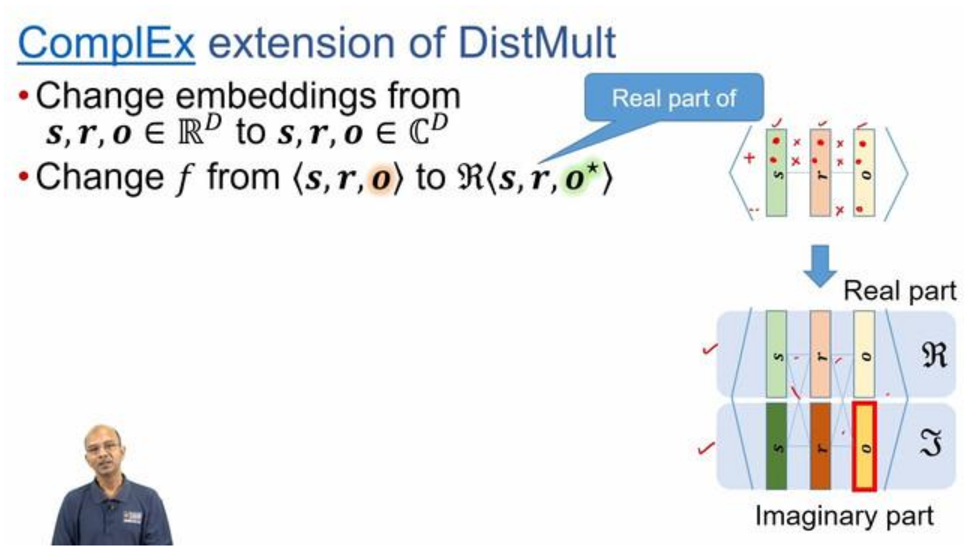
However, WN18 was hardened to WN18RR and FB15K was hardened to FB15K237 after certain weaknesses were found in the original datasets. Remember that WN18 was extracted from the English WordNet, which is a lexical network. It has 18 relations, like antonymy, synonymy, polysemy, et cetera, and about 41,000 entities. What was found is that many test triples SRO had corresponding inverse triples in the train fold, enabling many algorithms to overfeed or cheat. Removing these conflicting or leaking triples gave 93,000 triples over 40,943 entities and 11 relation types.


ComplEx extension of DistMult

- Change embeddings from $\boldsymbol{s},\boldsymbol{r},\boldsymbol{o} \in \mathbb{R}^D$ to $\boldsymbol{s},\boldsymbol{r},\boldsymbol{o} \in \mathbb{C}^D$

Real part
$\mathfrak{R}$
$\mathfrak{I}$
Imaginary part

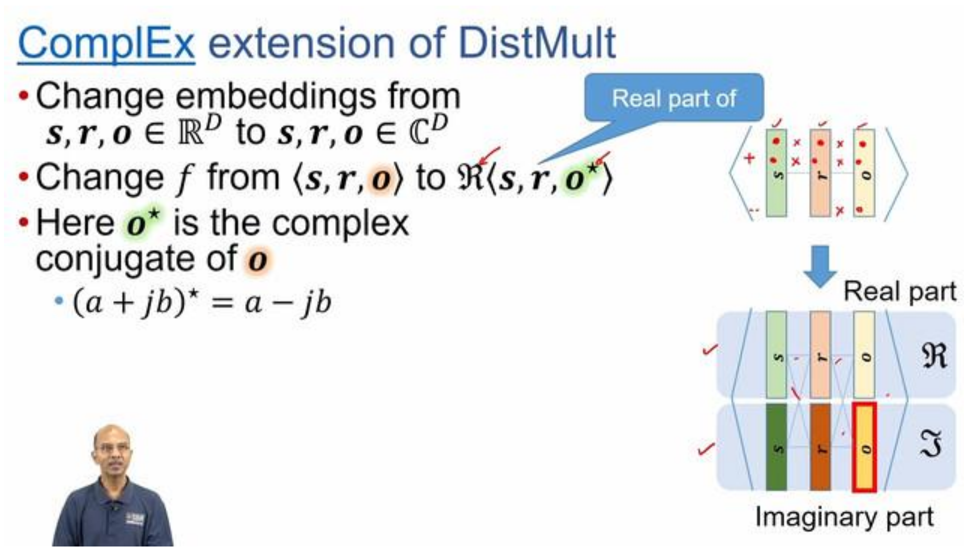
So this WN18RR is a harder benchmark with less leakage. The evolution of FB15K to FB15K237 followed a similar story, starting from a larger number of triples entities and relations. After the cleanup, people ended up with about 310,000 triples over about 14,500 entities and only 237 relations instead of 1,345 relations. These tables are a bit busy but we can only look at the general trends. Complex V2 uses large or full negative sampling and it's almost as good as complex N3 which basically leads the chart.



## ComplEx extension of DistMult

- Change embeddings from $\boldsymbol{s},\boldsymbol{r},\boldsymbol{o} \in \mathbb{R}^D$ to $\boldsymbol{s},\boldsymbol{r},\boldsymbol{o} \in \mathbb{C}^D$
- Change $f$ from $\langle \boldsymbol{s},\boldsymbol{r},\boldsymbol{o} \rangle$ to $\Re\langle \boldsymbol{s},\boldsymbol{r},\boldsymbol{o}^\star \rangle$

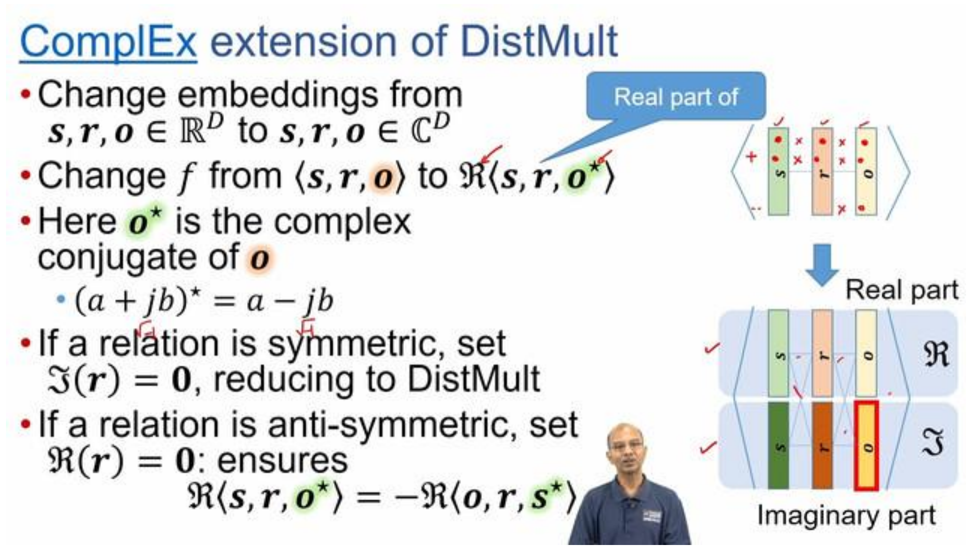
And both of these are better than earlier methods like SimpleE and Complex alone. This is true across all these datasets. So here's the summary so far. We defined what a knowledge graph is in terms of entities, relations, and fact triples. We noted that knowledge graphs are very incomplete and we need to either complete them or infer missing facts.



## ComplEx extension of DistMult

- Change embeddings from $\boldsymbol{s},\boldsymbol{r},\boldsymbol{o} \in \mathbb{R}^D$ to $\boldsymbol{s},\boldsymbol{r},\boldsymbol{o} \in \mathbb{C}^D$
- Change $f$ from $\langle \boldsymbol{s},\boldsymbol{r},\boldsymbol{o} \rangle$ to $\Re\langle \boldsymbol{s},\boldsymbol{r},\boldsymbol{o}^\star \rangle$
- Here $\boldsymbol{o}^\star$ is the complex conjugate of $\boldsymbol{o}$
  - $(a + jb)^\star = a - jb$

To do that we need a triple scoring function f which given a subject relation and object gives us a real confidence score in that fact. We can turn that into a probability if we wish and use that probability in a loss function. Then we explored two families of knowledge base encoding or representation schemes. The first used either translation or rotation. And the second, which we discussed just now is to use matrix and tensor factorization models.



## ComplEx extension of DistMult

- Change embeddings from $\boldsymbol{s}, \boldsymbol{r}, \boldsymbol{o} \in \mathbb{R}^D$ to $\boldsymbol{s}, \boldsymbol{r}, \boldsymbol{o} \in \mathbb{C}^D$
- Change $f$ from $\langle \boldsymbol{s}, \boldsymbol{r}, \boldsymbol{o} \rangle$ to $\Re\langle \boldsymbol{s}, \boldsymbol{r}, \boldsymbol{o}^\star \rangle$
- Here $\boldsymbol{o}^\star$ is the complex conjugate of $\boldsymbol{o}$
  - $(a + jb)^\star = a - jb$
- If a relation is symmetric, set $\Im(\boldsymbol{r}) = \mathbf{0}$, reducing to DistMult
- If a relation is anti-symmetric, set $\Re(\boldsymbol{r}) = \mathbf{0}$: ensures

$$\Re\langle \boldsymbol{s}, \boldsymbol{r}, \boldsymbol{o}^\star \rangle = -\Re\langle \boldsymbol{o}, \boldsymbol{r}, \boldsymbol{s}^\star \rangle$$