TransE

A Keras implementation

Francesco Casciola

Introduction

The request of the project was to produce an implementation for an algorithm (TransE, described in the paper *Translating Embeddings for Modeling Multi-relational Data*[1]) to learn embeddings of entities and relationships within the Knowledge Base FB15K and test the model on the Knowledge Base Completion task

[1] Antoine Bordes et al. "Translating Embeddings for Modeling Multi-relational Data". In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges et al. Curran Associates, Inc., 2013, pp. 2787–2795. URL: http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data.pdf.

The Knowledge Base (KB) FB15K

FB15K is a small subset of the Freebase KB consisting in $592,213$ triplets with $14,951$ entities and $1,345$ relationships. The triplets are composed by two entities (head and tail) and the relationship which connects them:

$$\Big((H)ead, (R)elationship, (T)ail\Big)$$

In particular, Freebase also comprises triplets that express the relationship which connects back the tail to the head, but these triplets are not included in FB15K.

Knowledge Base Completion Task

The Knowledge Base completion is the task which automatically infers missing facts, exploiting the information already present in the Knowledge Base. This means that, once the TransE model is trained over a set of triplets, it will have to be tested in the following tasks:

- ⋆ Predicting the head of a test triplet knowing the relationship and the tail,
- ⋆ Predicting the tail of a test triplet knowing the relationship and the head,
- ⋆ Predicting the relationship of a test triplet knowing the two entities.

Embeddings

The concept behind TransE is that, given the head h, the tail t and the relationship ℓ of a triplet, the embeddings shoud be such that

$$h + \ell \simeq t \ . \tag{1}$$

Which means that the relationship ℓ should operate the translation from the point represented by h to the one represented by t.

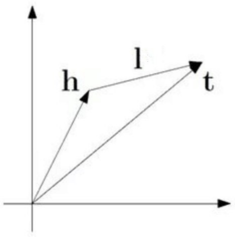


Figure: Simplified, $2-$dimensional embeddings' representation where $h + \ell = t$.

The Algorithm - 1

While a triplet from the training set (said positive triplet) (h, ℓ, t) must be such that (1) holds, if the positive triplet is modified by changing either the head or the tail, producing a *corrupted* (a.k.a negative) triplet not present in the training set, (1) shouldn't hold anymore.

So, said (h', ℓ, t') the corrupted triplet and given a distance measure $d(\cdot)$, the aim of the algorithm is to minimise $d(h + \ell - t)$ while making sure that $d(h' + \ell - t')$ doesn't become too small.

The Algorithm - 2

Algorithm 1 Learning TransE

input Training set $S = \{(h, \ell, t)\}$, entities and rel. sets E and L, margin γ, embeddings dim. k.

initialize $\boldsymbol{\ell} \leftarrow \text{uniform}(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each $\ell \in L$

$\quad\quad \boldsymbol{\ell} \leftarrow \boldsymbol{\ell} / \|\boldsymbol{\ell}\|$ for each $\ell \in L$

$\quad\quad \mathbf{e} \leftarrow \text{uniform}(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each entity $e \in E$

loop

$\quad \mathbf{e} \leftarrow \mathbf{e} / \|\mathbf{e}\|$ for each entity $e \in E$

$\quad S_{batch} \leftarrow \text{sample}(S, b)$ // sample a minibatch of size b

$\quad T_{batch} \leftarrow \emptyset$ // initialize the set of pairs of triplets

$\quad$ **for** $(h, \ell, t) \in S_{batch}$ **do**

$\quad\quad (h', \ell, t') \leftarrow \text{sample}(S'_{(h,\ell,t)})$ // sample a corrupted triplet

$\quad\quad T_{batch} \leftarrow T_{batch} \cup \{((h, \ell, t), (h', \ell, t'))\}$

$\quad$ **end for**

$\quad$ Update embeddings w.r.t. $\displaystyle\sum_{((h,\ell,t),(h',\ell,t')) \in T_{batch}} \nabla \left[\gamma + d(\boldsymbol{h} + \boldsymbol{\ell}, \boldsymbol{t}) - d(\boldsymbol{h'} + \boldsymbol{\ell}, \boldsymbol{t'})\right]_+$

end loop

The Algorithm - 3

The previous slide shows the original algorithm from the paper. The bold variables represent the embeddings. Embeddings are updated via SGD on a batch of size b using the loss function:

$$\mathscr{E} = \left[\gamma + d(\mathbf{h} + \ell - \mathbf{t}) - d(\mathbf{h}' + \ell - \mathbf{t}')\right]_+$$

Where the operator $[\,\cdot\,]_+$ stands for $\max\{\cdot, 0\}$. This implies that only the embeddings belonging to triplets such that $\mathscr{E} > 0$ are updated.

The Algorithm - 4

$$\mathscr{E} = \left[\gamma + d(\mathbf{h} + \ell - \mathbf{t}) - d(\mathbf{h}' + \ell - \mathbf{t}')\right]_+$$

The variable γ is an hyperparameter called ‘margin’ and defines a *threshold* for the difference between $d(\mathbf{h}' + \ell - \mathbf{t}')$ and $d(\mathbf{h} + \ell - \mathbf{t})$, after which the embeddings won't be updated.

In the algorithm, there is no mention about the way in which the sampling of corrupted triplets is done. The paper [1] proposes to randomly modify, for each triplet, either the head or the tail.

Setup

As a programming language, **Python** was chosen and the following packages were employed:

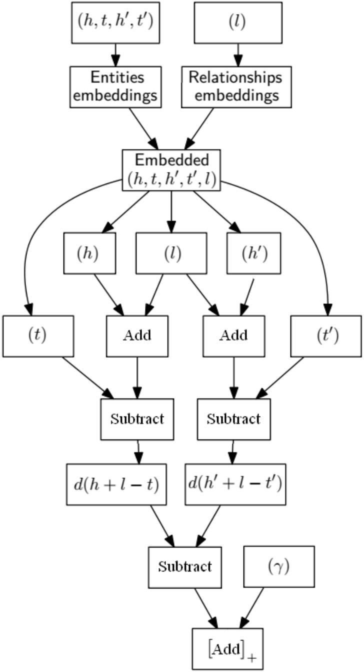
⋆ Pandas (for managing the Dataset)	version 1.0.3
⋆ Numpy (math and linear algebra functions)	version 1.17.3
⋆ Keras (defining and training the model)	version 2.2.4
with Tensorflow (GPU) backend	version 1.4.0
⋆ Scikit-klearn (to help with the corrupted triplets sampling)	version 0.21.3

Why Keras?

A first attempt of implementing the algorithm without using neural networks was made, but the training of the model would have required too much time using that code, making impossible the evaluation of the performances of the generated embeddings.

Keras is a Python library employed for building neural networks (so it also contains implementations of gradient descent algorithms) and offers *embedding layers*, whith customisable constraints and initialisation rules.

The Neural Network



Each couple of positive-negative triplets is fed to the neural network shown on the left.

Training this model in Keras requires having all the positive and negative triplets already generated, so the algorithm was modified moving the sampling of corrupted triplets outside the loop, but making sure of obtaining an equivalent result.

Sampling

The loop in the original algorithm includes two samplings:

- ⋆ Sampling b positive triplets from the training set.
- ⋆ Sampling a negative triplet for each sampled positive one.

In order to move these two samplings outside the loop, the following algorithm was employed:

Algorithm 1 Generate Neural Network Inputs

1. **Let** T be the training set, provided with a column containing unique indices for the entries.
2. **Let** N and P be copies of T.
3. **Split** N in two parts N_h and N_t.
4. Randomly **generate** corrupted heads (tails) for the triplets in N_h (N_t) to substitute the original ones.
5. $N = N_h \cup N_t$.
6. **return** N **left join** P on the index column.

Training - 1

During the project the Training, Validation and Test sets provided by the authors of the paper [1] were emplyed. The training set contains $483,142$ triplets, the validation set $50,000$ and the test set $59,071$.

After the introduction of **Algorithm 1**, said

- T the training set,
- b the batch size,
- N the number of epochs,

the overall training algorithm becomes:

Training - 2

Algorithm 2 Training

Initialise the embeddings.
for $e = 0$ to N **do**
  Generate neural network inputs I.
  for $j = 1$ to $\lceil size(T)/b \rceil$ **do**
    Feed the rows $[(j-1) \cdot b, j \cdot b)$ of I to the neural network.
    Sum together all the outputs to compute the loss function.
    Use gradient descent to update the embeddings.
  end for
end for

Validation

During the training phase, every t epochs, the validation set can be fed to the network instead of the training set.

The output of the network is the value of the loss function

$$\left[\gamma + d(\mathbf{h} + \ell - \mathbf{t}) - d(\mathbf{h}' + \ell - \mathbf{t}')\right]_+ \quad ,$$

and can be used as a measure of the performance of the embeddings over the validation set.

This validation score can be used to implement early stopping (if in the last n times in which the validation score was computed the latter doesn't improve, then the training can be stopped before reaching N epochs).

Test Method - 1

The knowldege base completion task for a generic test triplet ν, consists in removing one of its elements and use the remaining two to predict the removed one.

To evaluate how good the model is in the task, all the entities (or relationships, in case the missing element is a relationship) known to the model are used to replace the missing element, generating a new set of triplets (exactly one element of the set will be equal to ν). For each element in it, the distance measure $d(h + \ell - t)$ is computed.

Test Method - 2

Finally, the set of triplets is sorted by ascending distance measure and the first element of the set will be prediction of the model.

Since the algorithm returns the first element of the set, the model is considered good if the test triplet is consistently part of the elements at the beginning of the set. Therefore, for the evaluation of the model we'll use two metrics, derived from the *rank*, which is the position, in the sorted set, where the test triplet was found.

For each element of the test set the rank is computed using the following algorithm:

Test Method - 3

Algorithm 3 Rank computation for triplets in Test set

Let P be an empty array.
Select the element x to remove from all the triplets.
for each triplet ν in Test set **do**
    if $x ==$HEAD or $x ==$TAIL **then**
        Generate multiple triplets by replacing x with all the *entities* for which an embedding is available.
    else
        Generate multiple triplets by replacing x with all the *relationships* for which an embedding is available
    end if
    for each of the generated triplets **do**
        Compute $d(h + \ell - t)$.
    end for
    Sort the generated triplets according to $d(h + \ell - t)$.
    Insert in P the position of ν in the sorted set of triplets.
end for
return P

Metrics

The output of the previous algorithm, is a set containing the rank in which the model placed each test triplet. This set makes possible to compute the following metrics

- Mean Rank: The mean value between the elements in the set,
- Hit@10: The percentage of elements in the test set with rank in the interval $[0, 10)$.

To be able to compare the performances of this model with the ones of the paper [1], the metrics are computed for the prediction of missing head and missing tail and then the final scores are the average between the ones achieved in the two tasks.

Results - 1

According to the paper [1], the set of hyperparameters providing the best results in the task of predicting an entity is $\{k = 50,$ learning rate $= 0.01, \gamma = 1, d = L1\}$. Said results are:

DATASET	FB15K			
METRIC	MEAN RANK		HITS@10 (%)	
Eval. setting	*Raw*	*Filt.*	*Raw*	*Filt.*
TransE	**243**	**125**	**34.9**	**47.1**

The *Filt.* (filtered) results are obtained by removing, when computing the rank, all the generated triplets which happen to be true triplets from the dataset (excluding the one being tested).

Moreover, in [1] is not possible to find any results for the task of predicting a relationship.

Results - 2

The paper [1] doesn't declare the batch size b, so it was arbitrarily set to 300 (other hyperparameters are the same as the paper) obtaining the following scores:

Mean Rank	Hit@10 (%)
240	36.66

These results show that the proposed implementation should be equivalent to the original one. The embeddings produced during the experiment were also tested in the prediction of the relationship, obtaining Mean Rank $= 531$ and hit@10 $= 17.67\%$.

EXTRA

Testing a small modification to the algorithm.

Extra - The Idea

The process of learning the embeddings depends on:

- The minimization of the distance measure related to elements belonging to positive triplets,
- While ensuring a higher distance measure related to elements belonging to negative triplets.

When randomly generating corrupted triplets, it could happen that some generated triplets belong to the training set (*fake corrupted triplets*). How would the results change if the fake corrupted triplets were removed?

Extra - The Implementation

The algorithm to generate the neural network's inputs would become:

Algorithm 4 Generate Neural Network Inputs

Let T be the training set, provided with a column containing unique indices for the entries.
Let N and P be copies of T.
Split N in two parts N_h and N_t.
Randomly **generate** corrupted heads (tails) for the triplets in N_h (N_t) to substitute the original ones.
$N = N_h \cup N_t$.
for each entry x in N **do**
    if $x \in T$ **then**
        $N = N \setminus \{x\}$
    end if
end for
return N **left join** P on the index column.

Extra - Results - 1

The following table shows the performances on the test set with $b = 300$ and checking the presence of fake corrupted triplets using the training set:

Model's parameters				Entity prediction		Relationship Prediction	
Distance Measure	k	Optimizer	Learning rate	Mean Rank	Hit@10 (%)	Mean Rank	Hit@10 (%)
L1	20	Adam	0.001	234	32.21	83	61.45
		SGD	0.001	230	34.47	43	71.38
			0.01	233	33.38	162	46.7
	50	Adam	0.001	245	34.68	254	41.25
		SGD	**0.001**	**211**	**42.35**	78	68.15
			0.01	232	38.45	433	19.64
L2	20	Adam	0.001	327	27.23	35	84.35
		SGD	0.001	310	30.43	35	85.53
			0.01	311	28.9	38	82.49
	50	Adam	0.001	331	27.56	34	85.88
		SGD	0.001	310	30.45	35	85.58
			0.01	305	29.7	37	83.2

Extra - Results - 2

In the previous table, the highlighted results and hyperparameters are the ones obtaining the best score in predicting an entity. The hyperparameters providing the second best result are ones provided by the paper [1], but in this case the metrics have a slightly higher score:

Algorithm	Mean Rank	Hit@10 (%)
Original	240	36.66
Modified	232	38.45

At the same time, though, these hyperparameters provide the worst score out of the ones in the table when predicting a relationship.

Extra - Results - 3

The best result obtained with this modified version of TransE is also quite good in predicting relationships, with Mean Rank $= 78$ and Hit@10 $= 68.15\%$.

With a better look at the table it's possible to see that $L2-$norm, despite not providing as good results as $L1-$norm in predicting entities, is very suitable for relationship prediction.

Thank You.