Hybrid Embeddings with Attention Mechanism and Autoencoded Meta-Embedding

Rameez Mohammed QURESHI * Fatima HABIB † Asmaa DEMNY ‡
Tuan-Anh VO §

Abstract

In the article from Dieudonat *et al* (2020) [5], we see an attempt to enhance the performance of different type of embeddings by constructing a simple concatenative model between them. With the projection of advancing the results from it by producing a more robust concatenation architecture, we try to integrate this time two state-of-the-art approaches to the experiment. Firstly, we add to the encoder self-attention layers inspired directly from Transformer-based architecture, however with a specific design in a manner we call *"parallel attention mechanism"*. Secondly, we implemented the whole concatenation architecture with an approach that combines the strength of both meta-embedding and auto-encoder model, namely auto-encoded meta-embedding. We demonstrate the effectiveness of the approach over the same tasks and data set from [5]. With our combination of different methods, we achieve encouraging results that show substantial improvement from the concatenate model of [5].

Keywords: *attention mechanism, multi-head attention, Transformer, auto-encoded meta-embedding.*

1 Introduction

Dieudonat *et al* (2020) proposed a set of tasks, namely entity typing and relation prediction. To evaluate the performance of embedding models, it is designed in a way that knowledge and textual information can be used on the same level. The authors used two pretrained embedding models viz. ELMo as contextual embedding and ComplEx (PyTorch BigGraph) as Knowledge graph embedding to construct their tasks, and they used 2 variants of Freebase dataset, the FB15K and FB NewYorkTimes. Their experimental model simply concatenated the two embeddings, with the resulting embedding is almost twice as large as each of the separate embeddings. It contains the information of both the KG node representation and the textual description from the ELMo and BigGraph do separately, without removing or adding anything. Therefore, the concatenative model was expected to achieve performance equivalent to the best performing embedding on each task. However, for relation prediction, the concatenated embedding used in [5] does not fulfill this expectation.

*IDMC, Université de Lorraine Nancy, France. Email: rameez.mrq@gmail.com
†IDMC, Université de Lorraine Nancy, France. Email: fatimahabibaa95@gmail.com
‡IDMC, Université de Lorraine Nancy, France. Email: asmaademny@gmail.com
§IDMC, Université de Lorraine Nancy, France. Email: tuan-anh.vo8@etu.univ-lorraine.fr

Inheriting this previous experiment as a baseline, we recognize that one of the most crucial aspects of it is the concatenation model, which could be significantly improved with more involved model. This should solve the above mentioned issue with relation prediction. We therefore decide to keep the same experiment setting, i.e the data, the separated embeddings and the tasks, and focus on integrating the two following novel architectures to advance the concatenation process.

First is Attention Mechanism, a powerful technique that change the way we work with deep learning algorithms, and has achieved excellent performance on difficult learning tasks. It is originally used to improve neural machine translation (NMT) by selectively focusing on parts of the source sentence during translation [1]. On the other hand, auto encoded meta embedding, introduced first in [3] is an attempt to model the meta-embedding learning problem, a proven technique of combining word embeddings from different sources, as an auto encoding problem. This joint structure create a meta-embedding space that could accurately reconstruct all source embeddings simultaneously and be enforced to capture complementary information in different sources. These two techniques with their strength in capturing information effectively, therefore could hypothetically advance our initial simple concatenation model.

We structure this report in the following scheme. First we introduce the background knowledge on the attention mechanism and its specific branch of self-attention from Transformer that we use for our experiment. In the background we also briefly introduce the basic idea of meta-embedding and auto-encoder. Then we proceed to the Methodology section, by presenting our two main novel properties adding to the concatenation architecture: the idea of "parallel attention", and the auto-encoded meta embedding. In the experiment and result, we report in details the whole working process as well as analysing our results. Finally, we lay out some future projection for this project and conclude the report.

2 Background

2.1 Attention Mechanism

2.1.1 Seq2seq architecture

An important technique prior to attention is the seq2seq model. It's invented by researchers in the Google Brain team back in 2014, mainly to solve complex problems in the field of Language Modeling [11]. Since then, it has proven its immense power in numerous systems that we face on a daily basis, such as Machine translation, Question Answering, creating Chatbots, Text Summarization, etc.

At the core, seq2seq architecture aims to transform an input sequence (source) to an output sequence (target). Both sequences can be of arbitrary lengths and different from each other. Utilising the architecture composed of encoder and decoder (both are recurrent neural networks), the distinguished feature of it is the capacity of the encoder to compress all the information from the input sequence into a context vector of fixed length. This vector then serves as an initialiser of the decoder to emit the transformed output, expecting to yield some good results.

This fixed length context vector feature, however, proves to be inadequate when it comes to tasks with a long sequence, like remembering long sentences.Bahdanau *et al.* (2015) [1] came up with

a simple but elegant idea where they suggested that not only can all the input words be taken into account in the context vector, but relative importance should also be given to each one of them. This is the precursor of what we now know as attention mechanism.

2.1.2 Attention mechanism

Attention mechanism therefore was invented for this very reason of helping memorize long input sentences in neural machine translation (NMT). According to Vaswani *et al.* 2017, [12] attention mechanisms have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences. In all but a few cases [10], however, such attention mechanisms are used in conjunction with a recurrent network.

In the context of neural networks, it can be described as a vector of importance weights. According to [1], in the task of predicting a word for example, it allows the model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly. Applying for the vector, this (soft-)search technique is equivalent to how strongly the vector is correlated with, or "attend' to other elements surrounding the target word. Then we take the sum of these elements' value, weighted by the attention vector to yield the approximation of the target.

The key strength of attention mechanism is the capability of it to consume three pieces of information: the hidden states in both encoder and decoder, and most importantly, alignment between source and target. Being learned and control by the context vector, this alignment help solve the problem of forgetting when accessing the long input sequence

The alignment model assigns a score to the pair of input at one position and output at the coordinate position, based on how well they match. The set of scores are weights defining how much of each source hidden state should be considered for each output. In [1], the alignment score is parametrized by a feed-forward network with a single hidden layer and this network is jointly trained with other parts of the model. The alignment score function is therefore in the following form, given that tanh is used as the non-linear activation function:

$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$

The alignment score function varies according to the matrix of alignment scores therefore could show the correlation between source and target words.

With the help of the attention, the dependencies between source and target sequences are not restricted by the in-between distance anymore. Given the big improvement by attention in machine translation, it soon got extended into the computer vision field [13] and people started exploring various other forms of attention mechanisms ([8], [4], [12]

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

FIGURE 1: Family of Attention Mechanism techniques. [1]

2.1.3 Transformer

A part of our experiment is to apply novel properties from an important variant architecture of attention mechanism, which is the Transformer. It came from one of the most influential papers in NLP in recent years, “Attention is All you Need” [12]. According to the paper, “The Transformer is the first transduction model: "(“transduction” means the conversion of input sequences into output sequences.) relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution.” The self-attention then, "is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence.”

[1]Image source: https://lilianweng.github.io/

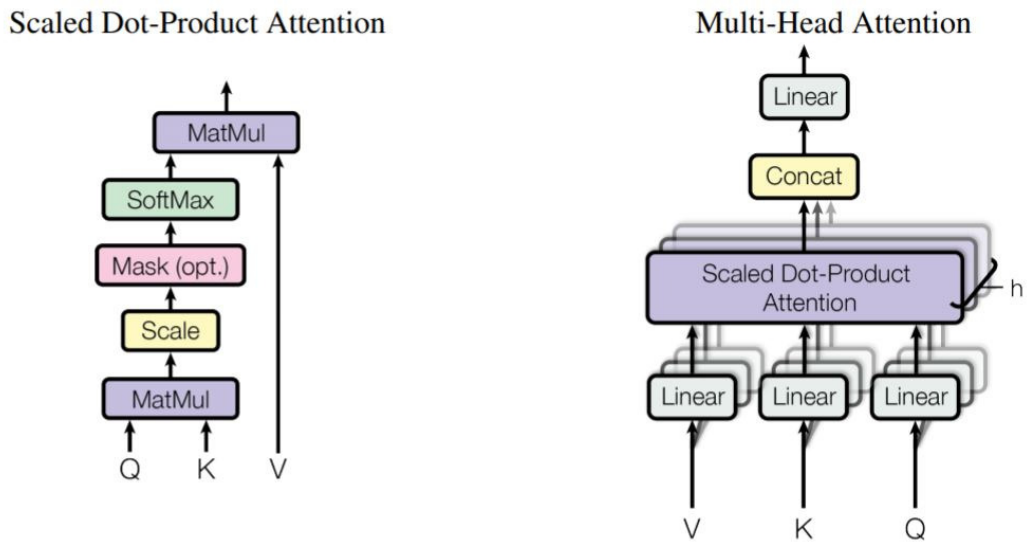

FIGURE 2: Scaled Dot-Product Attention(left) and Multi-Head Attention (right) with several attention layers running in parallel.

They also referenced another concept called multi-headed Attention. The author also call it "scale dot-product attention", in which self-attention is computed not once but multiple times in the Transformer's architecture, in parallel and independently. The independent attention outputs are simply concatenated and linearly transformed into the expected dimensions.They then redefined the self-attention by providing a very generic and broad definition of Attention based on keys, queries, and values.

2.1.4 Key, query and value

The use of concepts of key, query and value, though in a novel sense from the paper that we will discover later on, obviously got its inspiration from the retrieval systems. Intuitively, in a search engine like Youtube for example, it will map the words or phrases in searching as a query, against a set of keys (video titles, description, etc.) that associated with candidate videos in the database, then presenting the best matched videos, most of the time in some sort of ranking (values). The attention operation turns out can be thought of as a retrieval process as well, so the key/value/query concepts also apply here.

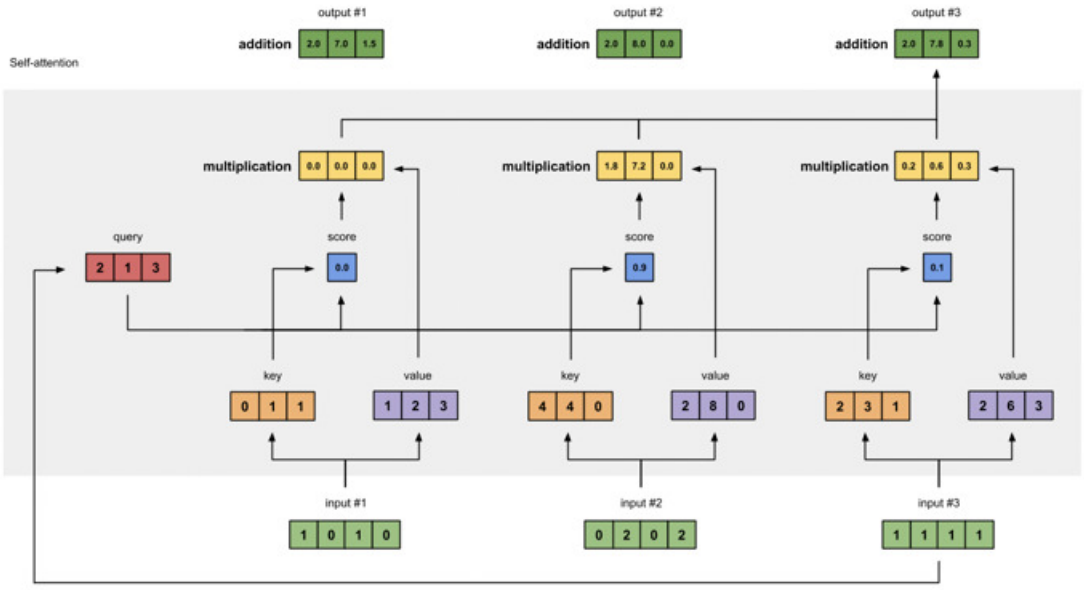

FIGURE 3: Self attention mechanism with key, query and value

Here we have the diagram of what happens under the hood of a multi-head self attention layer. Every input must have three representations. These representations are called key (orange), query (red), and value (purple). The query is from the decoder hidden state and the key and value are from the encoder hidden states. In order to obtain these representations, every input (green) is multiplied with a set of weights for keys, a set of weights for queries, and a set of weights for values.The context vector is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key. The score is the compatibility between the query and key, which can be a dot product between the query and key. The scores then go through the softmax function to yield a set of weights whose sum equals 1. Each weight multiplies its corresponding values to yield the context vector which utilizes all the input hidden states. The values that come as input to the attention layers are calculated from the outputs of the preceding layers of the network.

If we manually set the weight of the last input to 1 and all its precedence to 0s, we reduce the attention mechanism to the original seq2seq context vector mechanism. That is, there is no attention to the earlier input encoder states. The difference here is that the queries, keys, and values are transformations of the corresponding input state vectors. The others remain the same.

2.2 Meta- Embedding

The goal of the meta-embedding learning is to learn a single (possibly lower-dimensional) common embedding space by combining multiple, pre-trained source word embeddings without retraining the sources or requiring access to the linguistic resources such as corpora or dictionaries used to train the source embedding[9].

First meta-embedding learning method(1TON) was represented in Yin & Schütze [14]that projects a meta-embedding of a word into the source embeddings using separate projection matrices, the meta-embedding in their experiments outperform Glove and Word2Vec on analogy tasks. The most straightforward meta-embeddings approaches are **Concatenation** (CONC) and **averaging** (AV).

In O'Neill & Bollegala [9] work, they proposed a new method of meta-embedding, a direct averaging of embeddings. This method also outperform the source embeddings sets in similarity task and word analogy task. One advantage of averaging meta-embedding is the reduction of the dimensionality where it does not increase beyond the maximum dimension present within the source embeddings those resulting embeddings are easier to process and store.

The concatenation was performed in Dieudonat et al. [5], where their model was simply a concatenation of the embedding generated with the two pre-trained models we use for KB and contextual data (BigGraph and ELMo respectively). The resulting embedding is almost twice as large as each of the separate embeddings (512 cells from ELMo and 400 from BigGraph so 912 cells once concatenated). The concatenation of these two embeddings did not achieve significant improvement in results. The most recent meta-embedding approach is the Autoencoded meta-embeddings method which was used in Bollegala & Bao [3] ,see section3.2.

2.3 Autoencoder

An autoencoder is a neural network that is trained to attempt to copy its input to its output[**?**] commonly used in feture selection and extraction,these neural networks are designed to encode the input into a compressed and meaningful representation, and then decode it back such that the reconstructed input is similar as possible to the original one[2],they are used for the tasks of representation learning.
Autoencoder is a nonlinear generalization of PCA, it uses an adaptive, multi-layer "encoder" network in order to reduce the dimensionality of the data into a low-dimensional code , also a "decoder" network to decode the encoded data. The auto encoders can be trained end-to-end or gradually layer by layer.

2.3.1 Undercomplete Autoencoders

We want the auto-encoder to keep the important attributes in h(output of the decoder) from the input data not just copy the input which is an useless process, this may happens when the encoder and decoder are allowed too much capacity [6],therefore the dimension of h will be less than the dimension of the input x. This reduction in the dimension force the auto-encoder to capture the most notable attributes of the input data, these auto-encoders called **under-complete**.
An under-complete autoencoder has explicit regularization term,the model is trained according to the reconstruction loss. The only way to make sure that the model does not recognize the input data is to restricted the number of the nodes in the hidden layers.

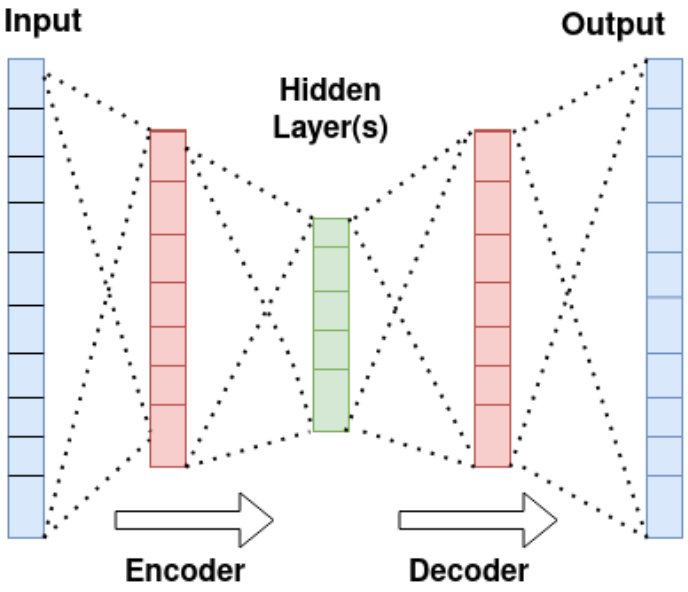

FIGURE 4: The architecture of the Undercomplete Auto-encoders

2.3.2 Sparse Autoencoders

An important feature in autoencoders is the bias-variance trade off, the sparsity on the hidden activation is one way to deal with this tradeoff [2]. No reduction of number of nodes is applied in the hidden layers , instead we lead the network to learn an encoding and decoding which only relies on activating a small number of neurons in the hidden layers. Unlike the undercomplete autoencoder, a sparse autoencdoer will be forced to activate some nodes in the hidden layers. There are two ways to enforce the sparsity regularization :(1)L1 Regularization, (2) KL-Divergence (quantifies how much one probability distribution differs from another probability distribution.[7]) . In 5 we can see how not all the nodes are activated in the hidden layers.

2.3.3 Denoising Autoencoders

The denoising auoencoder is a extension version of the autoencoder, it creates a corrupted copy of the input, a copy disrupted by some noise in order to prevent the network from learning identity function.

3 Methodology

3.1 Parallel Attention

3.1.1 Description

Here we introduce the main idea behind our implementation of attention. In the encoder, we add two different multihead attention layers. These two attention layers function as a kind of “filter” that would hypothetically help us to ‘attend’ to more relevant information from the input vectors, in our case are two source embeddings ELMo and BigGraph. The “parallel mechanism” work in a sense

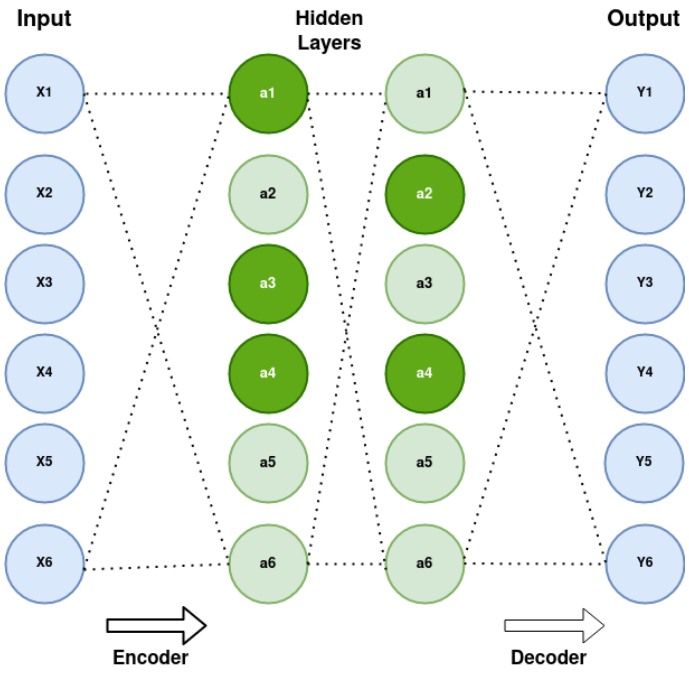

FIGURE 5: The architectureof the Sparse Autoencoders

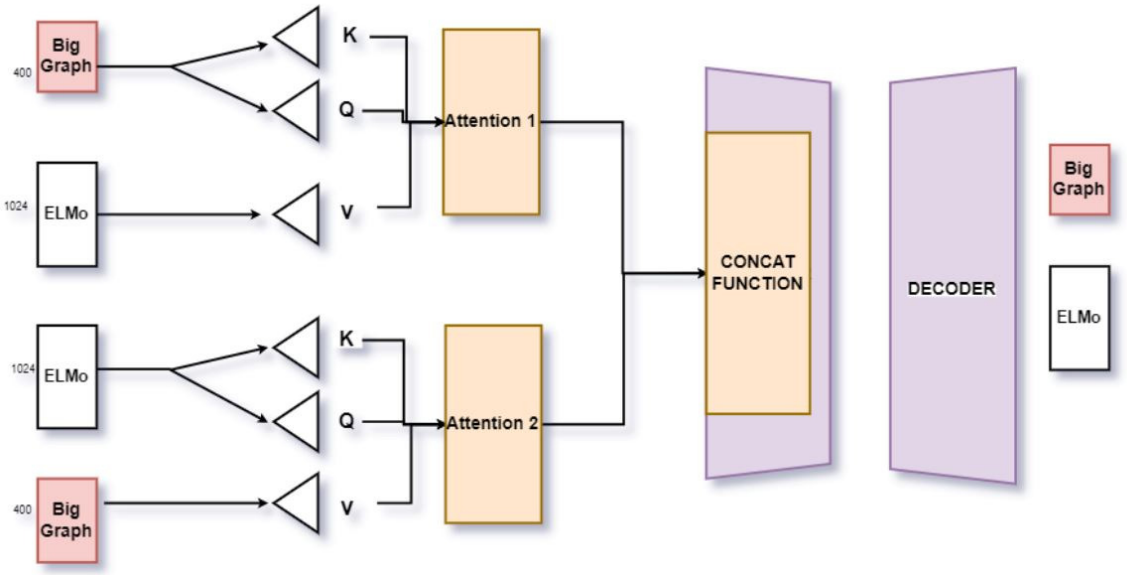

FIGURE 6: The mechanism of Parallel Attention experiment

that we arrange different orders, or more precisely, exact opposite orders, when assigning the key, query and value of each attention layer to the source embeddings.

So for the first layer, the score, which is the dot product between the query and the key, comes from Big Graph Embedding. It then goes through the softmax function to yield an attention weight, which then be multiplied by the corresponding value come as input from ELMo embedding, to yield a context vector. The process is the same for the second attention layer, except the score now

comes from ELMo, and the value from BigGraph. The two different context vectors of the same size then serve as the input of a simple concatenation function that we have seen in the previous experiment, which is then embedded in the encoder, before further implemented in our different auto encoded-meta embedding models.

The capability of the Transformer's multi-head attention mechanism to have key, query, and value come from different source inputs is a major inspiration for our experiment . By changing the order we can yield different attention weights which hypothetically could attend the relevant information from both textual and knowledge graph resources, via our source embeddings.

3.2 Auto-Encoded Meta-embedding

Representing the meaning of individual words is the first step that could help to find the meaning of complex and larger lexical units like phrases, sentences, paragraphs. As we discussed in the previous sections, Meta-embedding is the process of generating a single (meta) word embedding from a given set of pre-trained input (source) word embeddings.[14] When meta-embedding are used it's important to consider that :

Firstly, we do not re-train any of the source word embedding learning algorithms during meta embedding. Pre-trained source word embeddings are only used.[14] This makes the meta embedding learning algorithms independent of the source word embedding learning algorithms.

Secondly, we do not assume the availability of the text corpora that were used to train the source word embeddings. In fact, no text corpus is ever used. Sometimes all what we can get with source word embeddings is the pre-trained vectors, but not the language resources.

To solve the problem of learning a single metaembedding from two given source embeddings we use the autoencoding methods. In the figures, the two sources are denoted by S1 and S2 and the dimensionality of two source embeddings S1 and S2 are given respectively d1 and d2.

The first one is **Decoupled Autoencoded Meta-embedding (DAEME)** in which meta-embedding is represented as the concatenation of two encoded source embedding E1(s1(w)) and E2(s2(w)) for each word w V for each word. To train meta-embeddings using Autoeconders we cannot use ElMo because it requires context with the tokens, this is why we use as input token-embedding matrix. The

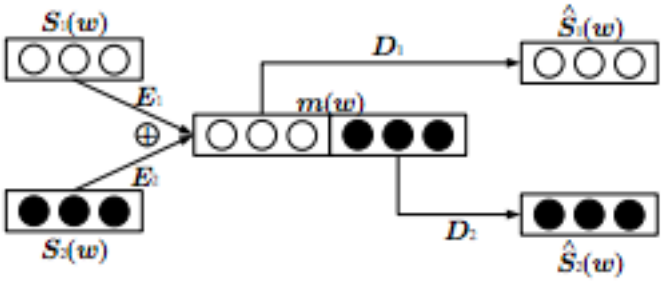

FIGURE 7: Decoupled Autoencoded Meta-embedding [3]

concatenation is the result of the creation's meta-embeddings from multiple source embeddings, we

can see its structure in Figure 7: it doesn't have linear neural networks, the transformation of each encoder is independent from the source embedding.[3]

The second one is **Concateneted Autoencoded Meta-embedding (CAEME)**, It is also represented by the concatenation of two source embedding but instead of treating the meta-embedding as two individual components[3], CAEME reconstructs the source embeddings from the same meta-embedding. The dimensionality M is the same as DAEME. The coefficients 1 and 2, as DAEME,

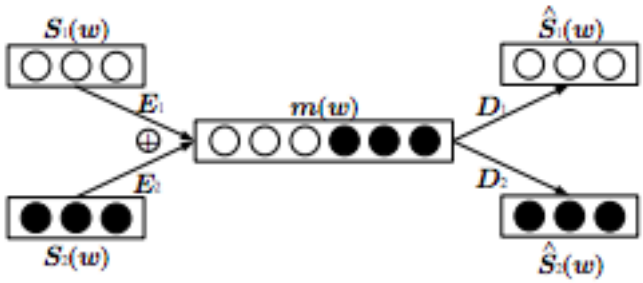

FIGURE 8: Concateneted Autoencoded Meta-embedding [3]

are used to give different emphasis to the reconstruction of the two sources, the difference between DAEME is that CAEME imposes a tighter integration between the two sources in their metaembedding.

The third one is **Averaged Autoencoded Meta-embedding (AAEME)** where the meta-embedding is computed by averaging the two encoded sourced instead by concatenation. This autoencoded Meta-embedding works as CAEME but it does not increase the dimensionality [3], the operation of transforming each source embedding independently using two encoders permit that they could be averaged in the same vector space. AAEME computes the meta-embedding of a word w from its

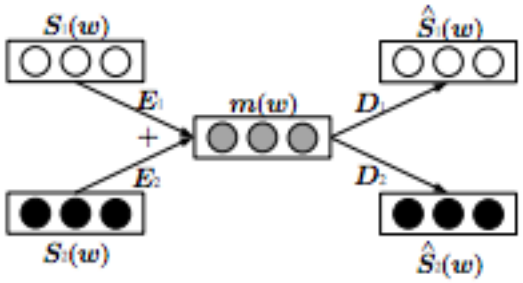

FIGURE 9: Averaged Autoencoded Meta-embedding [3]

two source embeddings s1(w) and s2(w) as the '2-normalised 1 sum of two encoded versions of the source embeddings E1(s1(w)) and E2(s2(w)).

AAEME is different from the other autoencoded meta-embeddings because the outputs of the two encoders should be equal in order to facilitate summation.

4 Experiments

In the following section, the experimental setup of the proposed task has been explained. We start with explaining the choice of input embeddings, the tasks performed for evaluating the models and properties of the dataset, continuing with the details of the different combination we employed during the process, and finally, conclude with the results.

4.1 Chosen Embeddings

For the experiments, we chose ELMo (contextual word embedding) and ComplEX (knowledge graph embedding). ELMo is based on bi-directionally trained language model, which represents each word based on its context and results in a 1024 dimensional vector representation (in our case). Having trained bidirectionally, the word representations depends on both the previous as well next neighbouring words. Having less complicated architecture as compared to BERT, and modelling more nuanced information than basic word2vec models, ELMo allows to not compromise with the information modelled while working on comparatively less demanding computations.

On the other hand, selection of ComplEX is based on its better performance on predicting the relations in FB15k dataset [see Section 4.3] as compared to other alternatives (TransE and DistMult). To obtain the ComplEX embedding, PyTorch-BiGraph (trained on FB15k) has been implemented, the output of which is a 400-dimensional vector.

4.2 Tasks Performed

As the nature of the hypothesis and base input embeddings (namely, ELMo and Knowledge Graph embeddings) is similar to what has been used in [5], it is wise to follow the similar set of tasks for the sake of evaluation and comparison. Therefore, the tasks used are Entity Typing and Relation Prediction. Observing the nature of both the input embeddings, it is expected that KB embeddings should perform better than contextual word embeddings. This is due to the fact that KB embeddings represent entities learned from triplets, whereas, word embeddings utilizes unstructured raw text for the same. We describe the task briefly below, before proceeding to the properties of the dataset.

4.2.1 Entity Typing

The task of entity typing aims at identifying the semantic types of an entity in a given text when provided with labelled data with entities and relations between them. This information is useful for many interesting NLP tasks, including question answering, co-reference resolution, semantic search, etc. This task is chosen in order to emphasize the context-fullness of the provided embedding, or how well the embedding carries the relations and contextual information.

4.2.2 Relation Prediction

As the name suggests, this task is aimed at identifying the named relation between two named entities. This task is performed in the information extraction domain very frequently. However,

#	Count
Train samples	11607
Test samples	1268
Sentences	13,874
Entities	798
Relations	16
Samples	29,492

TABLE 1: Composition of dataset used for experiments

due to its property to evaluate the relatedness between embeddings of given entities, this task looks promising to evaluate our hypothesis.

4.3 Dataset

Given the contrasting nature of chosen input embeddings (ELMo and Graph Embeddings), it isn't easy to find a dataset which satisfies the kind of input both the embedding model requires. To explain this further, please note that, to calculate contextual embeddings (such as ELMo), the required inputs are the position of the target word as well its context in the text. Whereas entity-relation triplets are essential to calculate graph embedding. Therefore, the dataset required to perform the proposed task with the chosen embeddings should fulfil two following conditions, namely, entity-relation triplets as well as the surface realization should be present for all possible entities. Hence, we chose Freebase-NewYorkTimes ("FB-NYT") dataset for all the experiments as it contains the surface realization of triplets found in Freebase. We followed the process used in [5] to pre-process the data, more information about which can be found in the respective work. We mention the important numbers and properties of the dataset used in table 1.

4.4 Evaluation

As discussed above, the tasks to be performed are task typing & relation prediction, and both the tasks are types of multi-class classification. However, the only difference between the tasks is: there is only one relation to predicting in relation prediction against predicting more than one types per sample in entity typing. As the motivation behind the experiments is to observe the relative performance of the proposed models, and not improving their performances individually, we chose Logistic Regression for the mentioned tasks. The reason behind this choice is the simplicity and expressiveness of logistic regression as compared to the more complex neural network-based alternatives. It also allows us to deal with the large dimensionality of the input embeddings quite effortlessly. Although, we encourage in using more complex classifiers to verify the results of this work if the higher computation costs can be managed. Regarding the evaluation of the results, the following metrics have been used:

4.4.1 Precision@n

In cases where the sample belongs to more than one classes, precision@n is deployed to measure the performance of the model up to a certain cut-off predictions(i.e. n). It is expressed mathematically as followed:

$$\text{Precision@n} = \frac{\text{no. of correct predictions up to n}}{\text{n}} \tag{1}$$

4.4.2 Mean Average Precision@k

As the name suggests, the Mean Average Precision for a given dataset is the mean of the Average Precision up to k (AP@k) for each sample. Mathematically, it is expressed as follows:

$$\text{AP@k} = \frac{1}{m}\sum_{i=1}^{k} \mathbf{P}(label_k), \text{if the } k^{th} \text{ item is relevant} \tag{2}$$

where, k is the cut-off position, and m is the available number of labels for the given sample.

4.4.3 Mean Reciprocal Rank

Reciprocal rank is defined as $\frac{1}{\text{rank}}$, where rank is the position of the highest ranked correct answer. In case of no correct answer, the reciprocal rank is 0. Hence the Mean Reciprocal Rank (MRR) is defined as:

$$\text{MRR} = \frac{1}{\mathbf{N}}\sum_{i=1}^{\mathbf{N}} \text{RR}(output_i) \tag{3}$$

where N is the total number of samples in the dataset.

4.5 Combinations of Embedding

In addition to the concatenation, we have devised three different approaches to merge the input embedding. First one of them is the PCA reduction of the concatenated embedding to several dimensionalities, in order to set up a baseline and measure the effect of dimensionality reduction on the concatenated version. Secondly, we employed already proposed methods like Auto Encoded Meta Embed-dings(AEMEs) (see Section 3.3) with contextual and graph embeddings. This will further explore and compare the performance of a model based on neural networks (autoencoder) over the classical dimensionality reduction methods like PCA. For this, we followed the hyperparameters and loss functions as specified in the paper [3], respectively. We proceed with including multi-head attention mechanism in the encoder stage of the autoencoder, as described in the Section 3. As the attention mechanisms are widely appreciated to attend the necessary chunk of information from the given inputs while encoding, this method verifies the importance of attention mechanisms in the domain of meta embedding. To the best of our knowledge, no such approach has been discussed before. We discuss the results of these three approaches in the section to follow.

Dimension	MRR	MAP@1
400	0.708	0.481
800	0.716	0.493
1200	0.710	0.476
2848 (w/o PCA)	0.738	0.533

TABLE 2: Results for the relation prediction task with PCA applied to the concatenation model.

5 Results

As discussed in [5], the concatenated model tend to perform poorly in the case of the relation prediction task. Therefore, one of the main motivation of this project is to improve the performance in relation prediction. Hence, let's start with observing the effect of PCA on concatenated embeddings for the relation prediction task. Results in table 2 show that, reducing the dimensions of the concatenated model does not have a significant impact on the results. Also, the performance is inferior as compared to the original embedding (with 2848 dimensions). This sets up the baseline for our future experiments and motivates us to implement more advanced techniques of merging.

Moving forward, we tested PCA and autoencoder mechanism for entity typing as well, results of which can be found in table 3. Please notice that the performance obtained by applying for entity typing is substandard. Alarmingly AEMEs failed miserably in predicting the entity types. Although, when attention is included, performance of CAEME increased *significantly* (0.137 to 0.775 MAP@10). The performance of CAEME is still substandard, but this hints towards the impact attention can have in improving the quality of embedding.

Table 4 shows the results obtained for the relation prediction tasks. As the results obtained with AEMEs (w/o attention) was not promising, and due to limited computation resources, we decided to run the experiments with AEMEs with the attention only. As discussed before, it can be observed in table 4 that PCA does not offer improvement in case of relation prediction. However, things become interesting while observing the results for AEMEs with an attention mechanism. Observe that all the variants of AEMEs performed better than the best performing model in previous works. This is when all the models have lesser dimensions. This indicates that attention indeed adds up significantly to the information encoded by the embeddings. However, decreasing the dimension further may lead to lesser accuracy (see AAEME with 200 dimension performs poorer than 400 dimension variant). This suggests that there should be some optimum dimension size of such meta embedding for better performance. Also, among the three variants of AEMEs, interestingly DAEME performed best even after having the most simple architecture. The difference is not significant for arriving at some conclusions about the optimum architecture design. Nevertheless, we achieved promising results in relation prediction task while keeping the dimensions of the hybrid embeddings nearly one-fifth of the embeddings proposed previously.

Model (w/ dimensionality)	**MAP@10**	**Precision@10 (mean ± sd)**
Base Models		
Contextual Embeddings (1024)	0.631	0.449 ± 0.271
KG Embeddings (400)	0.825	0.528 ± 0.269
Concatenation (1424)	**0.828**	0.527 ± 0.268
PCA Reduction		
Contextual Embeddings w/ PCA (200)	0.190	0.208 ± 0.249
KG Embeddings w/ PCA (200)	0.673	0.476 ± 0.261
Concatenation w/ PCA (200)	0.297	0.306 ± 0.277
Auto Encoded Meta Embeddings (AEMEs)		
DAEME	0.101	0.200 ± 0.282
CAEME	0.137	0.246 ± 0.299
AAEME	0.125	0.198 ± 0.212
AEMEs w/ Multi Head Attention		
DAEME w/ attn. (400)	0.602	0.461 ± 0.283
CAEME w/ attn. (400)	**0.775**	0.514 ± 0.271
AAEME w/ attn. (400)	0.277	0.283 ± 0.271

TABLE 3: Results for the Entity Typing Task.

Model (w/ dimensionality)	**MRR**	**MAP@1**
Base Models		
Contextual Embeddings (2048)	0.554	0.554
KG Embeddings (800)	**0.817**	**0.663**
Concatenation (2848)	0.738	0.533
PCA Reduction		
Contextual Embeddings w/ PCA (400)	0.656	0.433
KG Embeddings w/ PCA (400)	0.750	0.557
Concatenation w/ PCA (400)	0.708	0.481
AEMEs w/ Multi Head Attention		
DAEME w/ attn. (400)	**0.829**	**0.687**
CAEME w/ attn. (400)	**0.816**	**0.666**
AAEME w/ attn. (400)	**0.820**	**0.671**
AAEME w/ attn. (200)	0.809	0.653

TABLE 4: Results for the Relation Prediction Task.

6 Future Works

Although the inclusion of attention mechanisms improved the model's performance in the relation prediction task, we still have a long way to go when it comes to entity typing. Given the interesting

gap between the results from the entity typing task compared to relation prediction, the future work could focus specifically on the entity typing to understand substandard results.

Methods to intervene directly to the concatenation process could also be considered, including using random initialisation, transfer learning, and training while testing to repetitively return the model to induce bias in the training process. Utilising a pretrained model is also a possible choice.

Finally, it is interesting to observe the performance of the model, while one implements different methods like additive attention. In our approach, the merging of the attended embeddings are still dependent on the statistical methods (concatenation and averaging), therefore integrating another attention layer for merging could be envisioned.

7 Conclusion

In conclusion, in this work, our main motive was to merge the two different embeddings in order to improve performance in entity typing and, more importantly, relation prediction. Building upon the already available concatenation model as a baseline, we started with introducing PCA and observed that PCA did not offer significant performance changes.

In the later stages of the project, we implemented the autoencoder mechanism with multi-head self-attention in order to merge the embeddings. Although the results obtained in entity typing were below par, auto encoded models with attention showed promising performance in predicting relations. This work further motivates us to continue exploring the application of attention in calculating more robust meta-embeddings. Additionally, we hope our work can serve as the basis for future experiments, and with stronger reasoning and more extensive try-and-error process, truly efficient concatenated meta embeddings could be totally possible.

References

[1] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[2] Bank, D., Koenigstein, N., & Giryes, R. (2020). Autoencoders.

[3] Bollegala, D. & Bao, C. (2018). Learning Word Meta-Embeddings by Autoencoding. In *Proceedings of the 27th International Conference on Computational Linguistics* (pp. 1650–1661). Santa Fe, New Mexico, USA: Association for Computational Linguistics https://www.aclweb.org/anthology/C18-1140.

[4] Britz, K. & Varzinczak, I. J. (2017). Context-based defeasible subsumption for dSROIQ. In *COMMONSENSE*.

[5] Dieudonat, L., Han, K., Leavitt, P., & Marquer, E. (2020). Exploring the Combination of Contextual Word Embeddings and Knowledge Graph Embeddings.

[6] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[7] Kullback, S. & Leibler, R. A. (1951). On Information and Sufficiency. *Ann. Math. Statist.*, *22*(1), 79–86, https://doi.org/10.1214/aoms/1177729694 https://doi.org/10.1214/aoms/1177729694.

[8] Luong, M.-T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

[9] O'Neill, J. & Bollegala, D. (2020). Meta-Embedding as Auxiliary Task Regularization. In *ECAI*, volume 325 (pp. 2124–2131).: IOS Press.

[10] Parikh, A. P., Täckström, O., Das, D., & Uszkoreit, J. (2016). A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*.

[11] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*, 3104–3112.

[12] Vaswani, A., et al. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

[13] Xu, K., et al. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning* (pp. 2048–2057).

[14] Yin, W. & Schütze, H. (2016). Learning Word Meta-Embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 1351–1360). Berlin, Germany: Association for Computational Linguistics https://www.aclweb.org/anthology/P16-1128.