Multi-Modal Information Extraction in a Question-Answer Framework

Vamsi Chitters
Stanford University
vamsikc@stanford.edu

Fabian Frank
Stanford University
fabfrank@stanford.edu

Lars Jebe
Stanford University
larsjebe@stanford.edu

Abstract

Multi-modal information extraction is a challenging task and a topic of ongoing research in the areas of Natural Language Processing and Computer Vision. This paper presents an approach for extracting product specifications for a given attribute query, based on descriptions and images from the MuMIE Dataset provided by Diffbot. Building upon methods from existing literature, we propose an LSTM and CNN-based model for this task. Furthermore, we include a detailed account of the incremental changes that were made to the model throughout the development process. The paper focuses on the textual mode, since the descriptions contain the most relevant information, but also includes preliminary results for the image mode. The final model which relies on a pointer network and domain-aware heuristics achieves a test accuracy of 72.7 %.

1. Introduction

Classical machine learning typically uses one homogenous mode of data to classify examples or to make predictions. In contrast, humans rely on various modes (e.g., visual, auditory, textual) to understand context in order to make informed decisions. This is why even today, human performance is still superior to machine learning algorithms for tasks such as natural language processing or image classification. In this paper, an approach for multi-modal information extraction is presented. This method aspires to train models to think more like a human does and to make use of different modes concurrently to extract domain knowledge.

The objective of this project is to leverage multi-modal source information, in the form of text, images, and attribute-value pairs, to answer attribute queries in an efficient manner. The presented model is trained and tested on the MuMIE dataset using supervised learning.

2. Related Work

The task of multi-modal information extraction is a topic of ongoing research [4]. There are several approaches to combining information from multiple modes, such as co-learning, translation [3], or fusion (which our project makes use of). In addition to multi-modal problems, a lot of research has been carried out on single modes and many groups have tackled certain dimensions of this project, such as Q&A on text [7] or images [20].

Currently, the SQuAD dataset [13] is one of the most commonly used datasets for natural language question answering. LSTM is often utilized to tackle this problem and achieves state-of-the art results [19, 18]. Hence, LSTM is also used as a core component of our model. Moreover, the architecture we use treats text and attributes separately, similar to the attention mechanism proposed in [16]. Finally, we use answer pointers to handle multi-word values. While the pointer network in [17] is more complex and designed for inputs of different lengths, we simplify it to fit our model using a constant text input length.

As for the image mode, literature review includes both non-deep and deep learning models (which explore aspects like attention) [14]. The most straightforward way involves combining CNNs and LSTMs to map images and questions to a common feature space [1]. Other approaches rely on, for example, memory-augmented and modular architectures that interface with structured knowledge bases. One dataset of particular interest is the Visual7W genome dataset, a frequently used multimodal dataset for Q&A based on images [21]. Zhu et al. exploit the semantic link between textual descriptions and image regions using object-level grounding and spatial attention based LSTM models. Many of the core aspects are complementary to the work done in this paper, although not in the same detail.

Since no literature for multi-modal info extraction on product descriptions has been published yet, we have a distinct opportunity to be one of the first teams to explore this problem, while leveraging past research on similar problems.

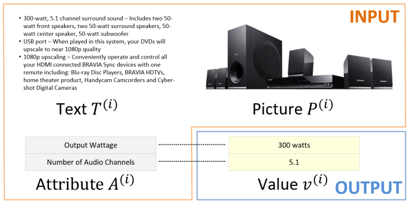
3. Dataset

The MuMIE Dataset is provided by the startup *Diffbot*. This is a new dataset that is not publicly available, but we have been given access to it. The dataset specs can be seen below. Some challenges include: length of description varies from 20-2783 words, non-sensical attribute value pairs: 'm': 'xs, s, l' and duplicate values: 'bluetooth: 4, 4.0, v4. Refer to `data_parser.py` for preprocessing techniques (regex clauses, data sanitization, stopword-removal, image normalization)

# of attribute-value pairs	8,306,549
# of unique attributes	2,185
# of unique values	14,787
# of products	2,517,709
# of images	4,891,284
dataset size	206 GB

3.1. Input-output Behavior

Training Set: Items I such that for each $i \in I$ there is a textual description $T^{(i)}$, set of pictures $P^{(i)}$, and set of attribute-value pairs $< A^{(i)}, v^{(i)} >$.
Task: answer a question regarding the value $v^{(i)}$ of a test attribute $A^{(i)}$, based on $T^{(i)}$ and $P^{(i)}$:


- 300-watt, 5.1 channel surround sound – Includes two 50-watt front speakers, two 50-watt surround speakers, 50-watt center speaker, 50-watt subwoofer
- USB port – When played in this system, your DVDs will upscale to near 1080p quality
- 1080p upscaling – Conveniently operate and control all your HDMI connected BRAVIA Sync devices with one remote including: Blu-ray Disc Players, BRAVIA HDTVs, home theater product, Handycam Camcorders and Cyber-shot Digital Cameras

Output Wattage	300 watts
Number of Audio Channels	5.1

Figure 1: Input-output-behavior for one product example

3.2. Evaluation metric

The performance of the model is evaluated using the accuracy evaluation metric: $\text{Acc}@k = \frac{1}{N}\sum_{i=1}^{N} 1[v^{(i)} \in \hat{v}_k^{(i)}]$. We focus on evaluating the algorithm for $k = 1$. This means that the correct answer has to be predicted on the first try. Only an exact match (note that values might consist of several words/numbers) is considered a correct prediction.

4. Method

4.1. Core Components

This paper now presents a deep-dive on the three core components associated with the task in a principled manner.

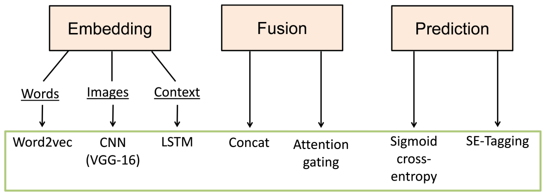


Figure 2: Overview of the core components

Embedding

Words: For multi-modal information extraction, it is imperative to leverage word representations to capture meaningful syntactic and semantic regularities. We make use of *Word2vec*, a group of models that are used to produce distributed word embeddings. One aspect we explored in detail was whether it was more effective to directly use Google's *Word2vec* to form our embedding layer or to build the word representation using only words in our vocab. Clearly this is a trade-off: by using Google's pre-trained Word2vec, there are more words (even web crawled) and general context. However, there is a higher chance of skew and incorrect value predictions for a given product example because many words may correlate with a context that is irrelevant to our problem. In the end, we chose to use Word2vec for the description/title while resorting to a custom embedding for attributes (smaller cardinality).

Images: A pre-trained VGG-16 model on ImageNet is used in order to extract semantic meaning from the images [5]. We elected to not use a more intricate model like Inception V3 in favor of model simplicity. There are two approaches we explored in regards to this: (1) caption the images and append those words to the product text description to add any additional information that was lacking before (2) convert the image into feature representation and merge that with the word vector prior to the prediction [2]. The advantage of (2) is that the vector representation of the image may capture image features that cannot be observed by looking at the image but at the same time, compounding the word vector with a large vector may detract prediction quality.

Context: It is essential to understand the context a word appears in (meaning) and utilize long-term dependencies (which options such as RNNs have difficulty with). This is why an LSTM with a sequential structure that uses the information from previous words when looking at the next word is used [9].

Fusion:

Concatenation vs. Attention Gating: The advantage of using concatenation is that it is simple and efficient, but on the other hand, attention gating which involves using a sigmoid activation and taking the element wise product as the primary gating function can sometimes lead to more accurate predictions because it steers the model to focus on important parts of the text.

Predictions:

Sigmoid Cross Entropy: Cross-entropy loss [6] seemed most apt for this task since it measures the performance of a model whose output is a probability between 0 and 1 and increases as the predicted probability diverges from the actual label. Some alternatives we considered included contrastive loss, categorical hinge loss and negative sampling. Although computing softmax is expensive, we optimized for accuracy in this case over a less-intensive sampling based approach which relies on a different loss to approximate the softmax.

'SE-Tagging': Since our evaluation metric requires the predicted value to be an exact (multi-word) match in order to be considered correct, it is paramount to address this with a sound strategy. Therefore, we later present 'SE-tagging', an approach that is similar to another commonly used approach in state of the art research (e.g., BIO-Tagging) [12]. It is a probabilistic approach that prioritizes accuracy over efficiency while determining the most likely start and end index span for a prediction value [5.4].

5. Model Evolution (Experiments)

Throughout the project, the model was developed step-by-step from the baseline to the final model. Each change to the model was made in order to overcome a specific shortcoming. The following paragraphs describe the evolution of our model and the building blocks that it consists of. Each change is motivated based on the results of prior conducted experiments.

5.1. Baseline

The baseline model (Test Accuracy: 0.5 %) works as follows:

1. Train a Word2vec [11] word embedding (skip-gram architecture, code template from [15])
2. Predict the value $v^{(i)}$ as most similar word to the attribute $A^{(i)}$ (cosine similarity), so that $v^{(i)} \in T^{(i)}$. The cosine similarity is calculated by first normalizing the embedding matrix, and then choosing as a prediction the word vector from the description $T^{(i)}$ that has the largest dot product with the embedded $A^{(i)}$.

Weaknesses: (1) model cannot understand context and (2) it is not able to predict multi-word values.

5.2. Adding Context

LSTM can be used to equip the model with the ability to understand semantic context. Our first approach concatenates $T^{(i)}$ and $A^{(i)}$ and incorporates a LSTM layer with 150 memory units after the embedding layer. Finally, a softmax output layer over the core vocabulary is used to predict a single-word value from the text description. In order to avoid a large output, we pivoted to predicting the position of the value in our text using softmax. In addition, we utilized a pre-trained word embedding from Google. This (combined with threshold-based extension as explained in Section 5.4) resulted in a test accuracy of 12.4 %.

5.3. T-A-Separation

The next shortcoming we tackled was the loss of information due to simply concatenating $T^{(i)}$ and $A^{(i)}$. To overcome this issue, we decided to process $T^{(i)}$ and $A^{(i)}$ independently and then concatenate them just before the dense output layer ("T-A-Separation", see top part of figure 3).

5.4. Predicting Multi-Word-Values: Two approaches

Threshold-based extension. Given a softmax output over all possible positions in the text description, the goal is to find an way to predict multi-word values. One approach that worked well was to look at the most likely position for a value and then potentially include surrounding words from the description, based on the softmax probability. To formalize this approach, let n denote the length of the despcription $T^{(i)}$, $\mathbf{p} \in \mathbb{R}^n$ the softmax output over all positions, and $V_{\text{th}} \in (0,1)$ a threshold value, which is a hyperparameter that is used to decide whether to include a neighboring word in the value. Let $\hat{v}$ be the index of the largest softmax output element: $\hat{v} = \arg\max_{i\in\{0,\ldots n\}} p_i$. Then, neighboring values $v_{\text{right}} = \hat{v}+1$ and $v_{\text{left}} = \hat{v}-1$ are included if

$$p_{i+1} > V_{\text{th}} p_i \quad \text{respectively} \quad p_{i-1} > V_{\text{th}} p_i \tag{1}$$

If v_{right} and v_{left} are indeed included, the process is repeated for their neighbors. This method enables the model to predict values of any length from 1 to n. Different test runs have shown that $V_{\text{th}} = 0.45$ is a good choice.

Pointer Network. An alternate approach was instead to use a Pointer Network similar to [17], where one model output points to the start of a value v_{start} and one output points to the end v_{end} ('SE-Tagging). To train this model, our ground truth values ought to be transformed into a one-hot representation, where the one appears at the position of the value in the text description. Ultimately, two independent outputs for v_{start} and v_{end} are generated. This model improved the accuracy to 16.3 %. Figure 3 shows the Pointer Network model with T-A-Separation and a custom attribute embedding. At the time, the model had proven to be the most accurate on the dev set.

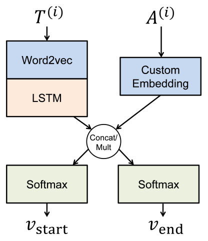


Figure 3: Pointer Network network with two independent pointers for start and end position

5.5. Enhanced Pointer Network

One flaw of the proposed tagging method is that the predicted start position is mutually independent from the end position. A very expressive model that is fed with a lot of data can probably figure out the dependence between v_{start} and v_{end} itself. However, instead of having the model figure it out, we directly imposed dependence on the start and end values using our domain knowledge via the following two constraints: (1) $v_{\text{start}} < v_{\text{end}}$, because values don't have negative length, (2) $v_{\text{end}} - v_{\text{start}} \leq L_{\text{v}}$, where L_{v} is the maximum value length, since all values contain fewer words than the longest value.

These constrains are added by creating and indexing a list of all possible tuples $(v_{\text{start}}, v_{\text{end}})$ that meet the requirements given the maximum length of the description.

5.6. Recommendation Vectors

Since the dataset contains far fewer unique attributes and unique values, it is possible to use the training data to create a set of potential values for each attribute, denoted by $S(A)$. In addition, it is possible to count the number of times each value appears for each given attribute. This map from attributes to a set of values with corresponding counts can be normalized and treated as a prior distribution of the values. The prior distribution $p(A)$ over all possible values for each attribute is used to create a recommendation vector $R^{(i)}$ that serves as an additional input to the model by following these steps:

1. For a given description $T^{(i)}$ and attribute $A^{(i)}$, get all values from the set of possible values that are in the product description: $V_{\text{possible}} = \{v_j^{(i)} \in S(A^{(i)}) : v_j^{(i)} \in T^{(i)}\}$
2. For each possible value $v_j^{(i)}$ in V_{possible}, get the index k_j that has been assigned to the $(v_{\text{start}}, v_{\text{end}})$ tuple (see Section 5.5)
3. For all j, set the k_j-th element of the recommendation vector $R^{(i)}$ to the value of the prior distribution.
4. Normalize $R^{(i)}$ using L2-Norm.

There are some test cases in which V_{possible} is empty. In that case, we cannot make a recommendation and the recommendation vector is all zeros. We see later that this does not impact the performance of the model (on the contrary!), because the model is trained to learn when to rely on recommendations and when to rely on its understanding of the context (see 5.7).

5.7. Concatenating Tensors and Learning Weights.

In contrast to the other inputs, the recommendation vectors do not need to be embedded in any way, as they are already shaped similarly to the desired output. They must, however, be fused with the LSTM output from the text descriptions and with the embedded attributes. Since the recommendations are essentially heuristic features, the natural solution is to concatenate everything and use a hidden layer with softmax activation to learn how to weight recommendations and the LSTM output. The output should be a probability distribution over possible start-end-tuples. The size of the hidden layer n is thus equal to the number of tuples:

$$n = L_{\text{t}} L_{\text{v}} - \sum_{i=1}^{L_{\text{v}}-1} i \tag{2}$$

where L_{t} denotes the maximum length of the text description and L_{v} denotes the maximum value length. In the case $L_{\text{t}} = 150$ and $L_{\text{v}} = 10$ the hidden layer size is 1445.

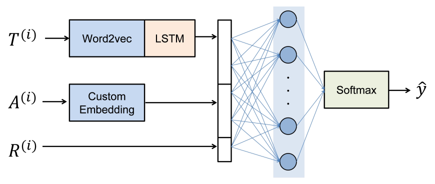
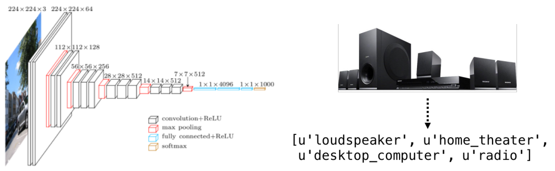


Figure 4: Final text model using recommendations and an additional hidden layer

Figure 4 shows the architecture of the final text model. For this, the recommendations were constructed from the prior distribution of values. This model architecture yielded a significant performance boost, as our model now achieves 72.7 % accuracy when trained on a sufficient amount of data.

5.8. Text + Image Model

We explored two approaches for the task of using images to answer attribute queries. Both approaches made use of a pre-trained VGG-16 model on the ImageNet data corpus [5]. The VGG network architecture was chosen in order to leverage its simplistic design, involving 3x3 convolution layers, followed by max pooling to reduce volume and fully connected layers in the end.



```
[u'loudspeaker', u'home_theater',
 u'desktop_computer', u'radio']
```

Figure 5: VGG-16 [8] and Image Caption Approach

Image captioning: Although this approach isn't widely popular across conducted literature review for this task, it seemed like a reasonable starting point. We captioned the images with respect to individual class labels (specific to ImageNet), ranked and selected the top 10 values (Figure 5). The resulting words are then concatenated with the words from the product description prior to the prediction of the final value. It is evident that the class level labels are too broad for any meaningful boost in overall prediction accuracy, as discussed in the Results section.

Image feature extraction: In the second approach, rather than captioning the images, we extracted features from the images instead. We excluded the 3 fully-connected layers at the top of the VGG-16 network in order to obtain the feature weights. The size of the extracted vector was originally 1×25088. One of the difficulties we faced involved reducing the vector size dimension to a more manageable size so that it didn't overshadow the product description weights. To combat this, we ended up normalizing the two disjoint vectors (text and image) prior to concatenation rather than using a more complicated dimensionality reduction technique.

Despite experimenting with a few different approaches, the results were not very impressive. Hence, we don't go into detail about this in the subsequent Results and Error Analysis sections. This presents itself as an area of future research as a means to improving overall multi-modal accuracy.

6. Performance Improvements

To improve performance, several experiments were conducted with different hyperparameter-, model- and algorithm choices. The test setup default parameters for these experiments are as follows (unless otherwise specified): Optimizer: Adam, Learning Rate: 0.001, Loss: Categorical Crossentropy, Dropout: 0, Dataset: Toy dataset.

6.1. Hyperparameter Tuning

One algorithm hyperparameter that we experimented with was the learning rate of Adam [10]. The top right plot in figure 6 shows our model performance for different learning rates. A learning rate of 0.001 proved to work best. With a lower training rate, the algorithm doesn't converge quickly enough. With a higher learning rate the model overfits, slightly reducing the test accuracy.

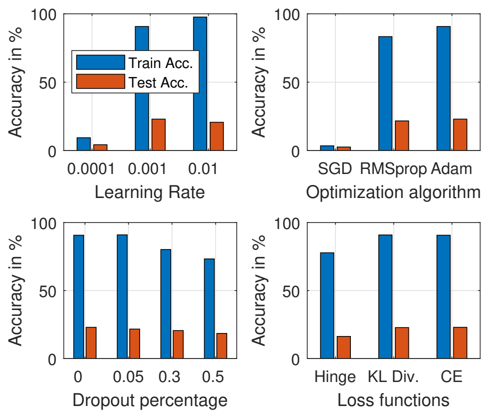


Figure 6: Hyperparameters Analysis

Besides the learning rate, we experimented with the number of memory units in the LSTM cell. The number of units functions as a knob for the amount of context the model should maintain prior to making a prediction. We found that generally, more LSTM units led to a better test accuracy. However, at some point the accuracy improvement with more LSTM memory units was not large enough anymore to justify the increasing computational cost. Eventually, we settled for 80 LSTM Memory Units, because the model still trained relatively fast and adding more units did not improve the performance significantly.

6.2. Modeling and Algorithm Choices

To improve the model even further, experiments for different degrees of regularization, optimization algorithms and loss functions were conducted.

Dropout: induces neural networks to randomly drop units during training to prevent their co-adaptation. While, as expected, more dropout (regularization) decreases the training accuracy and thereby decreases overfitting, we did not find that it improves the performance on the dev set (see bottom left plot of figure 6). This, combined with the fact that we have much more data than is included in the toy dataset, meaning overfitting is not as much of a problem when the model is trained, led to the decision not to include dropout.

Optimizers: While Adam has shown to be highly performing and efficient [10], it was initially not clear that this is the best algorithm choice. So, besides Adam, RMSprop and SGD were explored. Figure 6 shows the results for different optimization algorithms. While SGD did not manage to get close to the global optimum, RMSprop and Adam both performed much better. We eventually ended up utilizing the Adam for optimization in our model as it produces slightly better results than RMSProp does. In accordance with this, empirical results in a preliminary literature survey also demonstrate that Adam is well suited for training networks with many parameters as well as for for tackling problems with large amount of data.

Loss functions: The model output is a softmax distribution over a large number of categories (exact number depends on L_{t}, and L_{v}, see section 5.7). Not every loss function is appropriate for this setting. Three loss functions that work well for multi-class classification are Categorical Crossentropy, KL-Divergence and Categorical Hinge Loss. The effect of each of these three loss functions on the model performance is shown in the bottom right plot of Figure 6. In the comparison, Categorical Hinge Loss did not perform as well as the other two loss functions. While Categorical Crossentropy slightly outperformed KL-Divergence, it is interesting to notice that the model classified significantly more multi-word values correct with KL-Divergence compared to Categorical Crossentropy, but fewer single-word values.

7. Results

Please refer to Table 1 for a detailed breakdown of the accuracy results of different experiments on the train and test set, with respect to the total accuracy for k = 1.

Exp	Specifications	Train	Test
(1)	Word2vec cosine similarity	1.5 %	0.5 %
(2)	T-A-Concatenation	56.5 %	12.4 %
(3)	Two independent pointers	86.5 %	16.3 %
(4)	Additional hidden layer	90.7%	23.0 %
(5)	With image captioning	88.6 %	22.4 %
(6)	Trained on 342,099 examples	77.7 %	**72.7 %**

Table 1: Experiment Results for (1) Baseline (2) LSTM-Single Pointer-Net (3) LSTM-Pointer Network (4) Recommendation-Ptr-Net (5) Rec-Ptr-Net + Images (6) Recommendation-Ptr-Net (first four experiments on toy dataset rely on train dataset size: 2150 & test dataset size: 544)

The table only shows the most relevant experiments that were conducted, displayed in the order of experiment evaluation. The first 12 % of accuracy (after baseline) came from using LSTM and a single pointer based on the largest softmax output value. Another 4 % was gained by introducing two independent pointers that point to the start and the end of the value in the description. Imposing a dependence to these two pointers as well as adding the recommendation feature yielded 23 % in total. Unfortunately, our approach to include information from images was not very successful. The biggest improvement for the test accuracy was training the model on more data, which led to 72.7 % test accuracy, the highest accuracy we were able to achieve so far.

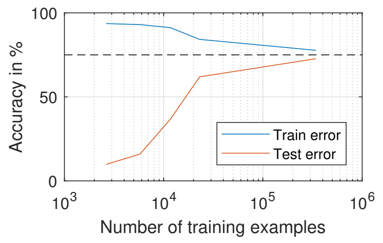


Figure 7: Learning Curve: Variance decreases when the model is trained on more training examples

As shown in figure 7, the model overfits less and generalizes much better to the test set when trained on more data. Initially, the variance is high when there is not enough training data and the gap between train and test error is very large but with more training data, this gap becomes significantly smaller.

8. Error Analysis

For the predictions that are incorrect, there are three main types of error: a single word close miss, where the predicted word captures the intention but not the right answer, a multi-word close miss where some of the predicted words are indeed included in the correct answer and a complete miss, where the prediction is incorrect. Table 2 shows an example for each error category.

Err	Attribute	Prediction	Actual Value
(1)	batteries included	included	yes
(2)	measurements	height 1.17 in	heel height 1.17 in
(3)	finish	price	semi-gloss

Table 2: Error Analysis for (1) Single Word (close miss) (2) Multi Word (close miss) (3) Complete Miss

To investigate how often close miss errors appear compared to complete misses, we calculated the “soft” accuracy for each prediction, as a word-based “intersection over union”. The ‘Multi-Word (close miss)’ example from Table 2 would yield a soft accuracy of 0.75, since the intersection of prediction and ground truth contains three words, and the union contains four words. Figure 8 shows the percentage of test examples that were classified into one of four categories using soft accuracies between 0 and 1.

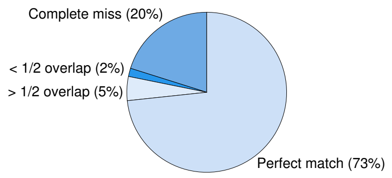


Figure 8: Percent of classified test examples that are completely correct, incorrect or partially correct.

Surprisingly, the partially correct predictions are relatively rare. Only 7 % of all predictions fall into this category, while 93 % of predictions are either an exact match or a complete miss.

9. Conclusion and Future Work

The goal of this project was to predict product specifications based on different modes (textual, visual etc.) using the MuMIE dataset. Data quality played an important role, since the dataset is web-scraped and had to be processed with various data parsing methods first. The high variety (0 to 14 words) of value lengths was a particular challenge, which the pointer network addressed effectively using mutually dependent SE-Pointers. This eventually was the breakthrough that led to an accuracy of 72.7 %. Contrary to our initial hypothesis, fusing text with images did not improve the accuracy in a meaningful way with the approaches we tried.

Further experimentation with more complex models like Microsoft’s R-NET is needed to gain more intuition for the task. Similarly, BIO-Tagging is worth exploring as an alternative to the developed SE-Tagging particularly for multi-word values. Also, re-training the last few hidden layers of the ImageNet to our own images may be fruitful. Lastly, adding domain knowledge (e.g., constraints such as [value for ‘dimensions’ must include numbers]) may improve the accuracy further.

In conclusion, there evidently is tremendous scope for research in this area as there are still many open-ended subproblems associated with the task of multi-modal information retrieval.

Contributions

During almost all of our coding and writeup sessions we sat together in one conference room to ensure everyone is contributing an equal amount to the project. We often split the work into several sub-problems that were tackled individually (divide and conquer) and later merged in our private Bitbucket repository.

References

[1] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang. Bottom-up and top-down attention for image captioning and vqa. *arXiv preprint arXiv:1707.07998*, 2017.

[2] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2425–2433, 2015.

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

[4] T. Baltrušaitis, C. Ahuja, and L.-P. Morency. Multimodal machine learning: A survey and taxonomy. *arXiv preprint arXiv:1705.09406*, 2017.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[6] L.-Y. Deng. The cross-entropy method: a unified approach to combinatorial optimization, monte-carlo simulation, and machine learning, 2006.

[7] D. Elworthy. Question answering using a large nlp system. In *TREC*, 2000.

[8] D. Frossard. Vgg in tensorflow. *University of Toronto*, 2016.

[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] A. Lazaridou, N. T. Pham, and M. Baroni. Combining language and vision with a multimodal skip-gram model. *arXiv preprint arXiv:1501.02598*, 2015.

[12] L. Màrquez, P. Comas, J. Giménez, and N. Catala. Semantic role labeling as sequential tagging. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 193–196. Association for Computational Linguistics, 2005.

[13] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[15] B. Steiner. Tensorflow word2vec tutorial. `https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/word2vec/word2vec_basic.py`, 2015.

[16] M. Tan, C. d. Santos, B. Xiang, and B. Zhou. Lstm-based deep learning models for non-factoid answer selection. *arXiv preprint arXiv:1511.04108*, 2015.

[17] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[18] S. Wang and J. Jiang. Learning natural language inference with lstm. *arXiv preprint arXiv:1512.08849*, 2015.

[19] S. Wang and J. Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*, 2016.

[20] Z. Yang, X. He, J. Gao, L. Deng, and A. Smola. Stacked attention networks for image question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 21–29, 2016.

[21] Y. Zhu, O. Groth, M. Bernstein, and L. Fei-Fei. Visual7w: Grounded question answering in images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4995–5004, 2016.