OPEN INDIVIDUAL ASSESSMENT

Author:
Y3507677

Module:
Natural Language Processing
(NLPR)

May 3, 2019

Abstract

CONTENTS

I. Introduction

The field of question answering (QA) is paramount in human computer interaction in the modern age. Understanding and modelling this mechanism is a large field in and of itself, and can lead us into a future unshackled from conventional input mechanisms, better integrating AI within our lives. Recent work in the field has postured that the vast majority of problems within the field of natural language processing (NLPR) can be expressed within the framework of question answering. Given a large enough data set within which to form 'facts' about the world, it follows logically that asking the right questions of that data set, and reasoning about the facts contained within it, can divulge greater insight and meaning.

At its fundamental level, the key to question answering is extracting meaning through understanding of natural language. Once the semantics of the speech are stripped and minimised, TODO: BLAH BLAH

A. Utilising the Dynamic Memory Network

The process and mechanism by which questions are answered is performed by a system primarily of 4 parts; a built up fact database, describing the fundamental knowledge required to answer a question. Following from this, a similar representation of the facts contained within the question is required. A subsequent system designed to perform reasoning upon these data stores is required, eliminating and inferring results by drawing upon the stored knowledge - driving it's 'attention' towards the result. This is extremely similar to mechanisms by which our brains perform this process, at a layman level. The 'fact' store aligns well with the brains memory. The final element composing the system will be needed to produce the answer from the reasoning representation, in a sense this can be thought of as speech in the human analogy. We will see shortly how it is possible to implement these components using a multitude of techniques, and compare the performance of them in search of the final solution architecture.

II. Literature Review

Within the context of this project, it is important to qualify the performance of the solution using a dataset, and similarly to train said solution on a dataset. For the purposes of QA, the bAbI dataset from Facebook is employed. The bAbI dataset is an industry standard dataset comprising 20 tasks that test both understanding and reasoning of an NLPR solution. Each task "tests a unique aspect of text and reasoning, and hence tests different capabilities of learning models" [1]. An example of a bAbI question and answer pair is given below, along with the associated world knowledge:

World Knowledge: John travelled to the hallway. **Question:** Where is John? **Answer:** hallway

The bAbI dataset is industry standard within QA research circles, and therefore allows for a quantitative comparison across competing QA architectures. This will be the starting point for identification of the ideal final system architecture.

III. Problem Modelling

A Dynamic Memory Network (DMN) expresses the design and subsystems necessary for a coherent question answering system [2, p.1]. It is an umbrella term encompassing a multitude of independent modules, including:

1) The frontside, input module. Responsible for putting the input data into a dimension and representation that subsequent modules are able to process. As is common for most network inputs, the data (in this case, a question) is encoded, in the case of a DMN this encoding produces "facts" (a set of distributed vectors) [3, p.1]. The input module will deal with text inputs from a range

of sources, be they tweets or sentences parsed from speech input - it operates independently of origin. The encoded data from this stage will be responsible for providing answers once processed via the *Episodic Memory* module.

2) The query module functions similarly to the input module in its task of encoding, with the key proviso that it directly influences the initial state of the subsequent Episodic Memory module. The query module, as its name implies, parses the question into the distributed vector representation required for storage into Episodic Memory *cells* [4, p.3].
3) The Episodic memory module, given the vector representations from both the input and query modules, is responsible for "determining which parts of the input to focus on, using its attention mechanism". [4, p.3]. This mechanism is implemented by means of a Gated Recurrent Unit (GRU); a network that iteratively updates output state dependent upon the hidden state h_t at each time step t [4, p.3]. This is covered in more detail later, as GRUs form the backbone of many of these modules.
4) The answer module is the simplest of all thus far. Its job is purely to amalgamate the answer representation from the Episodic memory module.

Each of these modules will be explained in more detail further on. For now it is important purely to understand their function in the context of the larger question-answering problem, figure 1 is provided to ease this.

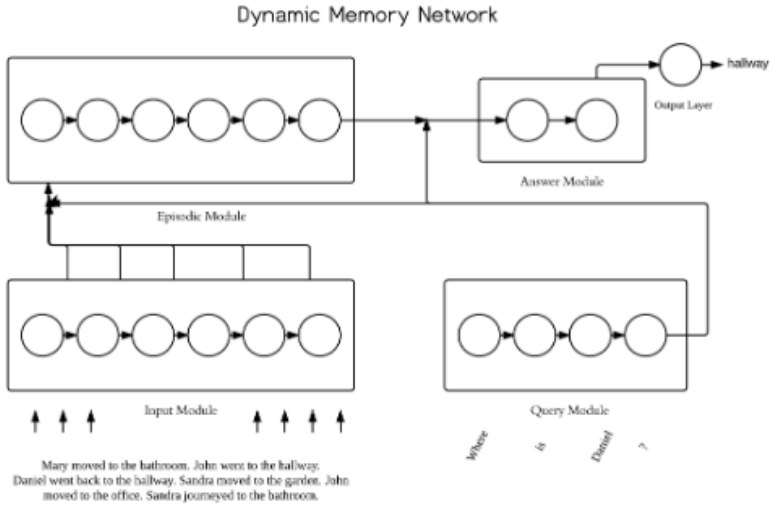

Fig. 1: Dynamic Memory Network composition from modules [4, p.2]

A. Gated Recurrent Unit

Introduced in 2014 by Cho et al [5], a Gated Recurrent Unit (GRU) functions similarly to a Long short-term memory (LSTM) but no forget gate is present. GRUs are an ideal component choice for a DMN and the bAbI dataset due to the short length of the facts that are provided to the network; much longer fact lengths are known to be best suited to LSTMs but this is strongly dependant on the dataset that is being used.

Kumar et al. [2] implemented their network using a single GRUs and while they did experience high levels of accuracy, the ability to perform multiple passes on the data allows for a context of tense to be gathered.

A GRU contains gates inside that control the flow of information throughout them but unlike LSTMs they do not contain additional memory cells. An update for a gate is computed as follows for time interval i:

$$u_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1} + b^{(z)}\right) \tag{1}$$

Resetting a gate effectively sets it to as if it is reading the first character of a sequence once again. To reset gate $\mathrm{r_t}$:

$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1} + b^{(r)}\right) \tag{2}$$

The activation function $\tilde{\mathrm{h}}_\mathrm{t}$ can be calculated as follows. Where $\circ$ is the Hadamard product (element-wise multiplication):

$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1} + b^{(h)}\right) \tag{3}$$

An update $\mathrm{h_t}$ calculates how much the gate updates its activation or contents is given by:

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t \tag{4}$$

Where

$$W^{(z)}, W^{(r)}, W \in \mathbb{R}^{n_H \times n_I} \text{ and } U^{(z)}, U^{(r)}, U \in \mathbb{R}^{n_H \times n_H} \tag{5}$$

are the n dimensions and hyper-parameters (n_H being hidden size and n_I being input size) [6].

B. Input Module

The input module features two components: a sentence reader and an input fusion layer. The sentence reader encodes words into embedded sequences of words and the input fusion layer creates interactions between layers using a bi-directional GRU; allowing for an understanding of both past and future tense. For sentence length $\mathrm{M_i}$ the encoding is $[\mathrm{w_1^i}, ..., \mathrm{w_{M_i}^i}]$ and is the input to the GRU. This DMN features a bi-directional GRU where information is shared between the multiple layers and is defined as follows:

$$\overrightarrow{f_i} = GRU_{fwd}(f_i, \overrightarrow{f_{i-1}}) \tag{6}$$

Where $\mathrm{f_i}$ is input fact at time i and $\overrightarrow{f_i}$ is the hidden state of the forward GRU. For the backward GRU, $\overleftarrow{f_i}$:

$$\overleftarrow{f_i} = GRU_{bwd}(f_i, \overleftarrow{f_{i+1}}) \tag{7}$$

This allows for both future and past facts to impact $\overleftrightarrow{f_i}$:

$$\overleftrightarrow{f_i} = \overleftarrow{f_i} + \overrightarrow{f_i} \tag{8}$$

1) Weights: Xiong et al [3] used Xavier initialisation for weights with random uniform initialisation with a range of $[-\sqrt{3}, \sqrt{3}]]$. This initialisation strategy was not used for their Visual Question Answering (VQA) system as this used uniform distribution with $[-0.08, 0.08]$. An improvement on the system will be made through the use of a truncated normal distribution with a standard deviation of 0.05.

2) Positional Encoder: This uses an implementation of the research that Sukhbaatar et al. [7] performed. This encoding system encodes the position of a word within a sentence and this results in the ordering of the word affecting $\mathrm{m_i}$. This takes the form of:

$$m_i = \sum_j l_j \cdot Ax_{ij} \tag{9}$$

Where $\mathrm{m_i}$ is the final encoded sentence, A is the layer's embedding matrix and $\mathrm{x_{ij}}$ is the jth word of sentence i. Where $\mathrm{l_j}$ is:

$$l_{kj} = (1 - \frac{j}{J}) - (\frac{k}{d})(1 - \frac{2j}{J}) \tag{10}$$

Where J is the number of words in a sentence and d is the number of dimensions. This is important as several bAbI tasks such as the positional reasoning tests require an understanding of the position of a word in a sentence.

C. Episodic Module

Accepts the outputs of the input module and focuses on which areas that may contain the answer to the provided question. While iterating over these inputs it produces vector representations of memories which are formed of the previous memories and the current context, the question. Episodes are formed of both a recurrent neural network and an attention gate $\mathrm{g_i^t}$ that is associated with each fact $\overleftrightarrow{f_i}$. The attention gate is defined as:

$$g_i^t = \frac{exp(Z_i^t)}{\sum_{k=1}^{M_i} exp(Z_k^t)} \tag{11}$$

This gives us the attention for gate $\mathrm{gt_i}$. This allows us to correlate similarities between facts for an episode are to previous facts and the question and adjust them with each iteration over the information.

1) Batch Normalisation: In order reduce internal covariate shift, "the change in distribution of network activations due to the change in network parameters during training" [8, p2], batch processing is performed. This is defined as:

$$y^{(k)} = \gamma^{(k)} \widehat{x}^{(k)} + \beta^{(k)} \tag{12}$$

Where both γ and β are learned during the training process, and $\mathrm{\widehat{x}^{(k)}}$ is the activation. Batch normalisation is performed as the distribution of the entire input is not known for each batch. As a result of this addition, it is possible to increase the learning rate slightly and adds the benefit of allowing for dropout regularisation changes to be made to the network.

D. Answer Module

bAbI features many questions that have one worded outputs and as such, simple softmax activation is used to predict the answer for a model.

IV. Implementation

The network has been implemented as its own Python class to aid in an efficient structure of the configuration; `improved.py`. Configuration flags are present in `main.py` and any adjustments to the network must be configured here before running. The code has been developed using Python 3.6.7, a virtual environment and pip3. If the reader using virtualenv as opposed to globally installed packages then the following command line argument - executed from the directory that the code is in - can be used to install the requirement packages: `pip install -r requirements.txt`; else, the required packages must be installed that are listed within the `requirements.txt file`.

Following any flag changes made, the code can be run with `python main.py` and the application will run.

The network described in section III has been implemented using TensorFlow 1.13.1 and the network architecture is described in figure 2. This figure describes the interaction between the various modules and highlights the addition of the batch normalisation process that takes place.

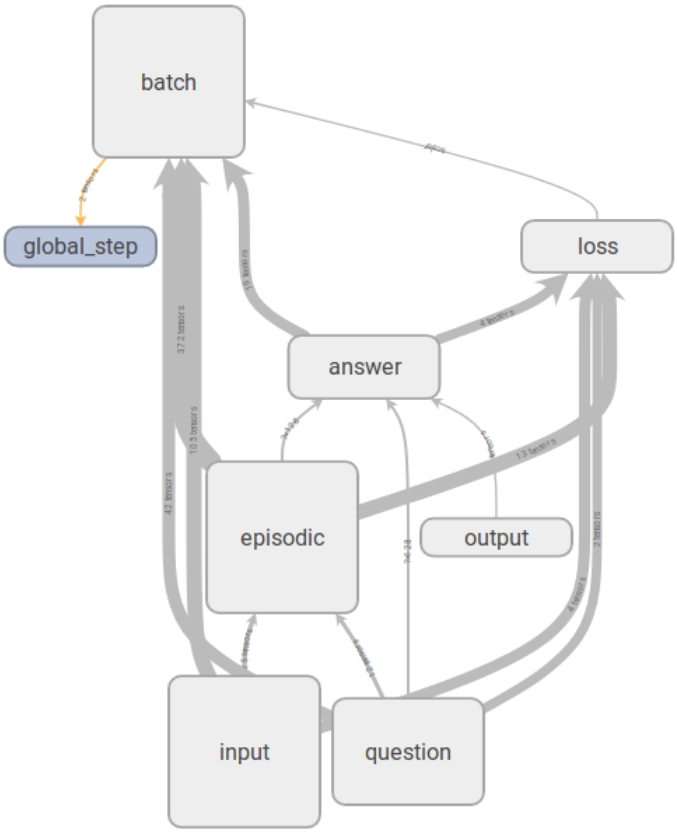

Fig. 2: TensorBoard network graph

1) Data Structures: TensorFlow uses tensors to represent data between edges in the network. A tensor's rank is identified by its dimensionality; for example, a rank 2 tensor is a matrix. All weight and bias variables are initialised as 32-bit single-precision floating-point tensors. Other variables, such as that of the sentence length, are initialised as int32 tensors.

A. Episodic Module

Episodes for the DMN are implemented as their own Python class. An episode accepts inputs from both the input and question modules and provides an output to the answer module. An episode is converted from

B. Answer Module

The answer module initialises a weight using truncated normal distribution with a standard deviation of 0.1 and a bias of 0.1. Dropout is set to 0.1 and is multipled with the weight of the module.

While the reference specification called for Xavier weights to have been used, truncated normal distribution has been favoured for avoiding dead neurons in the rectifier that is present in the attention units [9]. This may happen if the weight is not initialised with enough of a positive bias. The reference specification called for weight initialisation of $[-\sqrt{3}, \sqrt{3}]]$ and this may result in many dead neurons.

C. Loss and Optimisation

The learning rate is annealed over time up until a maximum of 0.01 is reached. This is done through the use of a `global_step` that is added to the Adam optimiser and is increased with every iteration in the network. This learning rate was selected following batch normalisation being implemented and it aiding in reducing the internal covariate shift in the network.

The Adam optimiser uses beta1 of 0.9, beta2 of 0.999 and epsilon of 1e-08 as per TensorFlow's recommended documentation for training. No criteria for stopping is implemented other than a maximum number of epochs for the network to take.

D. Pre-processing

The implemented DMN uses Global Vectors for Word Representation (GloVe) for training the network. GloVe maps words distances to similar words as vectors and allows the network to gain an understanding of when a word has been substituted with one of a similar meaning. The GloVe file is provided and during initialisation of the application, the file is parsed and for every word that is present in the bAbI task file the corresponding embeddings from the GloVe file are added to a set that is used when training the network. The addition of this understanding aids in bAbI facts such as: "John went to the hallway" and "Sandra journeyed to the kitchen".

V. Evaluation and Results

REFERENCES

[1] 2019. [Online]. Available: https://research.fb.com/downloads/babi/

[2] A. Kumar, P. Ondruska, M. Iyyer, J. Bradbury, V. Zhong, R. Paulus, and R. Socher, *Ask Me Anything:Dynamic Memory Networks for Natural Language Processing*, 2015. [Online]. Available: https://arxiv.org/pdf/1506.07285.pdf

[3] C. Xiong, S. Merity, and R. Socher, *Dynamic Memory Networks for Visual and Textual Question Answering*. MetaMind, 2016. [Online]. Available: https://arxiv.org/pdf/1603.01417.pdf

[4] A. Jiang and C. Asawa, *Dynamic Inference: Using Dynamic Memory Networks for Question Answering*. [Online]. Available: https://cs224d.stanford.edu/reports/allan.pdf

[5] K. Cho, B. V. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoderâĂŞdecoder for statistical machine translation," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[6] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *CoRR*, vol. abs/1409.1259, 2014. [Online]. Available: http://arxiv.org/abs/1409.1259

[7] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, *End-To-End Memory Networks*, 2015. [Online]. Available: https://arxiv.org/pdf/1503.08895.pdf

[8] S. Ioffe and C. Szegedy, *Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift*, 2015. [Online]. Available: https://arxiv.org/pdf/1502.03167.pdf

[9] 2019. [Online]. Available: https://medium.com/tinymind/a-practical-guide-to-relu-b83ca804f1f7

End of Examination