DeepQA

A PyTorch RNN Implementation for Solving QA bAbi Tasks

Pascal Wieler, Todor Kostov, David Schneider
Karlsruhe Institute of Technology
{uolje, uidhz, david.schneider2}@student.kit.edu

Abstract

In 2015, Weston et. al released the bAbi dataset, a collection of toy tasks in the field of sentence analysis and question answering (Weston et al., 2015). In this paper we present multiple RNN based approaches implemented in PyTorch for solving specific bAbi tasks. We provide a short introduction into the structure of the tasks on which we focus, explain the approaches we use to solve them and compare our results to existing solutions like the ones presented in (Weston et al., 2015) itself.

Keywords: bAbi Dataset, Artificial Intelligence, Recurrent Neural Networks

1. Introduction

In the light of improving artificial intelligence by using artificial neural networks, systems for speech and language understanding advance fast and can solve tasks of language analysis in a quality that has not been achieved in the past. However, the advancement, focused on certain cababilites of such systems, do not give a clear insight into what this means for the greater goal, of building an overall intelligent dialogue agent for communication with humans. It is furthermore difficult to evaluate varying approaches and compare them among each other in a more general sense. For these reasons it is desirable to have a universal standard to measure progress in this field. The bAbi Dataset is such a standard, providing different tasks which have to be completed by a language analysis system.
In this paper we present the different approaches that we used and implemented in order to solve a specific set of tasks from the bAbi dataset. Furthermore we provide an evaluation of the performance of our architectures on these tasks.

2. The QA bAbi Dataset

To provide a testing framework for text understanding and reasoning, Weston et al (Weston et al., 2015) provides a set of toy tasks in the QA bAbi dataset: A collection of training and test data for 20 different challenges which focus on more or less independent capabilities which the authors indentify as important for a cognitive text reasoning system.
The authors try to implement an approach which is similar to traditional software testing in computer science. The different tasks follow the idea of test cases, which can be solved each on their own or be combined to build more advanced tests.

2.1. Structure of a bAbi Task

Each bAbi task consists of a train and a test data set. In each task data set there are multiple stories and 1000 questions that need to be answered taking the information of these stories into account.
Each story consists of multiple enumerated facts which are given as (short) sentences. Each fact is enumerated chronologically by its appperance in the story (basically this is a line enumeration restarting at one every time a new story begins). At certain points within the stories, questions are asked which have to be answered taking the facts up to this point into account. There might be more facts to this story after the question which are not included at this point but have to be reconsidered for upcoming questions.
Both, the train and the test data also include the answers to the questions in the same line. For the bAbi tasks in general, answers provided to the questions consist either of a single word or of a list of words. For the tasks we focus on, answers always consist of a single word.

```
1 John travelled to the hallway.
2 Mary journeyed to the bathroom.
3 Where is John?       hallway   1

4 Daniel went back to the bathroom.
5 John moved to the bedroom.
6 Where is Mary?       bathroom  2

7 John went to the hallway.
8 Sandra journeyed to the kitchen.
9 Where is Sandra?     kitchen   8

10 Sandra travelled to the hallway.
11 John went to the garden.
12 Where is Sandra?    hallway   10

13 Sandra went back to the bathroom.
14 Sandra moved to the kitchen.
15 Where is Sandra?    kitchen   14
```

Figure 1: Example of a story with multiple questions (Weston et al., 2015).

2.2. Focused Tasks in this Paper

In this work we focus on solving a few selected tasks of the bAbi Dataset which we introduce and explain more thoroughly in this section. The tasks we focus on are:

- Task 1: Single Supporting Facts
- Task 2: Two Supporting Facts
- Task 3: Three supporting Facts
- Task 6: Yes/No Questions

2.3. Task 1: Single Supporting Facts

Task 1: Single Supporting Fact
Mary went to the bathroom.
John moved to the hallway.
Mary travelled to the office.
Where is Mary? A:office

Figure 2: Example for task one (Weston et al., 2015).

The single supporting fact task consists of questions that can be answered by taking only one of the given facts into account without combining it with other information. An example can be seen in figure 2 where only the last fact is necessary to answer the question referring to Mary's location. The question could still be answered correctly, if all other facts exept the last one would be discarded.

2.4. Task 2: Two Supporting Facts

Task 2: Two Supporting Facts
John is in the playground.
John picked up the football.
Bob went to the kitchen.
Where is the football? A:playground

Figure 3: Example for task two (Weston et al., 2015).

In contrast to the single supporting fact task, the two supporting fact task offers an harder problem, since it introduces the combination of multiple facts in the story. Questions as seen in figure 3 cannot be answered by evaluating either one of the two necessary facts on its own (line one and line two) but only by combining the information of both sentences.

2.5. Task 3: Three Supporting Facts

Task 3: Three Supporting Facts
John picked up the apple.
John went to the office.
John went to the kitchen.
John dropped the apple.
Where was the apple before the kitchen? A:office

Figure 4: Example for task three (Weston et al., 2015).

The three supporting facts task may seem like a simple extension of the two supporting facts case. However, due to the structure of the questions given, a learning system has to acount for some kind of temporal or at least sequential reasoning in order to answer questions about the state/location of objects at certain time points. In a longer story, this requires maintaining information about facts that might have been provided quite early. This stands in contrast to task one and two, where it is often enough to maintain the most recent information that has been gathered about a person or respectively an object.
Figure 4 gives an example for a question of this kind. Note, that there might be a lot of additional facts between the question and the information it refers to. The apple in the question might be picked up by another person and travel to several different rooms meanwhile; the system should still be able to provide the correct answer (at least as long as it does not become ambiguous, for example when the apple crosses the kitchen twice).

2.6. Task 6: Yes/No Questions

Task 6: Yes/No Questions
John moved to the playground.
Daniel went to the bathroom.
John went back to the hallway.
Is John in the playground? A:no
Is Daniel in the bathroom? A:yes

Figure 5: Example for task six (Weston et al., 2015).

At a first glance, task six appears to be an easy one for two reasons:
Firstly, the questions in figure 5 are of a simple kind, similar to the ones seen in task one. Secondly, there are only two possible answers; "yes" or "no". Therefore a system should provide at least a 50% score, even if it is providing those two answers randomly.
However, looking more closely, this is not the case. If a system is applied which is expected to solve several tasks at once, the possible answer the system might give could be any word, even if only "yes" or "no" could possibly be correct. After training on multiple task data, it therefore has to be able to differentiate among different types of questions.
It is also an increase in difficulty, that the answering word in question is no longer part of the facts in the story. A learning system might have learned by itself to detect keywords in the story when given a question and apply mechanisms to propose those keywords as answers. However, the transformation into a boolean requires real interpetation of the given information.

3. Background

3.1. LSTMs

In contrast to vanilla RNNs cells, which always use their output of a previous time step as input for the next time step, LSTMs have additional control about the information which is kept in the recurrent loop. An LSTM cell does not only have an output value h_t but also a cell state c_t which can be manipulated by the cell depending on the behavior

the cell has learned and the input data it is given.
The principle of an LSTM is shown in figure 6. The current sequence input concatenated with the output of the previous time step is used to influence and read the cell state by applying different gates. From left to right we can see the following gates in figure 6, depicted by yellow rectangles containing a σ:

- **Forget gate**:
  Scales how much of the previous cell state to keep by multiplicating it with a number between 0 and 1.
- **Input gate**:
  Scales how much of the information of the tanh-layer is added to the cell state by multiplicating the output of that layer with a number between 0 and 1.
- **Output gate**:
  Scales the output (hidden state) which is calculated by applying a tanh activation function on the current cell state and multiplicating it with a value between 0 and 1.

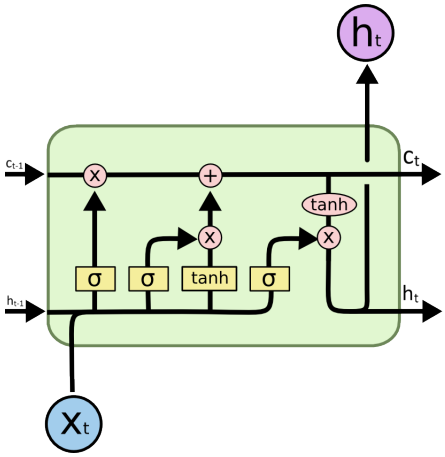

Figure 6: Long Short Term Memory Unit (Christopher Olah, 2015b).

Note, that each of the gates has learnable parameters which enables the cell to develop memorization strategies. By applying this approach, LSTMs can effectively be used on longer sequences than normal vanilla RNNs.

3.2. GRUs

GRUs follow a similar idea as LSTMs but are simpler in design. Instead of keeping a separate cell state, they merge it with the hidden state and only need two gates instead of three:

- **Update gate** z_t:
  Scales the influence of previous unchanged hidden state h_{t-1} and hidden state proposal $\tilde{h}_t$ on the output hidden state h_t.
- **Reset gate**:
  Scales the influence of the previous hidden state h_{t-1} on the new hidden state proposal $\tilde{h}_t$.

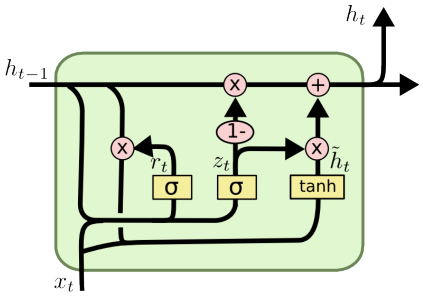

Figure 7: Gated Recurrent Unit (Christopher Olah, 2015a).

4. Architecture

4.1. Preprocessing

Since the bAbi tasks are given as text files with a structure as seen in figure 1, we need to do some preprocessing in order to prepare the data to be used as input for our networks. In order to provide a word representation for the network, we simply build up a vocabulary of indices with all words seen in the training data plus special placeholders for unknown words and the null index (which is reserved to be used for batchwise packing and unpacking features in PyTorch). The vocabulary is basically an extension of a map, mapping from words to indices and back. Nex, we use this vocabulary to translate every word of the dataset to its specific index representation. Furthermore, we construct triples of data, so that every triple consists of story-, question- and answer- word index vector. Those triples are used in the models for training and test data.

4.2. Basic Structure

In order to solve the bAbI-Task,s multiple models have been implemented. We distinguish our implemented models in the more basic models, that are the FB-LSTM-Baseline, the Story-Question-Model and the Attention-Based-Model, and a more advanced model, where we have implemented the Relation-Network-Approach presented in (Santoro et al., 2017). Although all implemented models have different architectures, the basic structure is the same; the models are split into three parts: Word-Embedding, Feature Extraction and Feed-Forward-Classification. The basic model structure can be seen in 8.

Figure 8: Basic structure of our models.

For the Word Embedding part, we decided to use the embedding layer provided by the PyTorch-framework. During testing the network, we experienced, that on the bAbi dataset we could not increase performance by actually using a pre-trained embedding. This is probably due to the small vocabulary of only about 20-40 different words (depending on the task). We therefore decided to use an untrained embedding and train it as part of our

models. For adapting our models to other tasks with bigger vocabularies, we would then use a pretrained embedding and freeze the embedding's parameters during training.

For the feature extraction part, we decided to use a recurrent architecture, since the goal is to model long-term dependencies between information from different facts of the story. Because of the problems of vanilla RNN, like the vanishing or exploding gradient with processing long sequences, we use more advanced recurrent units like LSTMs (Long-Short-Term-Memory-Units) or GRUs (Gated Recurrent Unit) that are presented in 3.

The different architectures of our implemented models are presented in the following sections. The evaluation of the models is presented in chapter 5.

4.3. FB-LSTM-Baseline

The FB-LSTM-Baseline is the first model, that we have implemented to have a proper baseline for the problem. The architecture is presented in (Weston et al., 2015) as a weakly supervised LSTM-Baseline for solving the bAbI-Tasks. The performance of this model on the bAbI-Dataset is worse than the performance of the more advanced models like the also in the paper presented Dynamic Memory Network. However, it is a simple approach to solve the problem and to see if the data can be learned. In the following, the approach is described in more detail:
As input for the FB-LSTM-Baseline-Architecture the whole story sequence and the whole question sequence is concatenated to one long input vector. This input vector contains the indices of all the words that occur in the story and the question in the correct order. The input sequence is then fed into the word embedding that delivers a sequence of word-vector representations. The word embedding is a linear mapping from word indices to word vectors. So, for every single word index we get a different word vector representation.
In the next step, the word-vector-sequence is processed by the LSTM. By using bi-directional LSTMs (or GRUs) the order of question and story in the concatenated input sequence is irrelevant here. After processing the whole input-sequence, we extract the last hidden state of the LSTM. This hidden state is called "story-question-code", because it should contain all the information that is needed to answer the question. This information should be existent, because it is the hidden state after reading in the whole story and the whole question. The hope is here, that by training the network with the supervised training data, the LSTM can find out, which information of the input stream is relevant to answer the question, and which is not. Since the story consists out of multiple facts (sentences), but mostly only few of them are relevant to answer the question, it is very important that the LSTM (or GRU) is able to identify the relevance of the input.
In the last step, the generated story-question-code is used in a fully-connected feedforward network to predict the output labels. Since we want to answer the given question and every word out of the vocabulary could be a possible answer, the size of the output layer is identical to the size of the vocabulary. Furthermore, we use a softmax-activation to get probabilities on the answer words.
The network is trained by using the cross-entropy-loss:

$$L_{CE}(o, l) = -\sum_i l_i * log(o_i))$$

The cross-entropy-loss describes the distance between the ground-truth probability vector l and the predicted probability vector o. By using gradient descent and backpropagation this loss is minimized. The ground-truth probability vector l is the one-hot-vector of the correct answer-word that we get from the given training data. The predicted probability vector o is the output of the network. The network is trained with the Adam optimizer with learning-rate 0.001. The visualization of the FB-LSTM-Baseline architecture can be seen in figure 9.

4.4. Story-Question-Model

In this section, the architecture of our second model "Story-Question-Model" is presented. In order to improve the performance of our first model we try out different architectures to see where we can improve our baseline. Since in the FB-LSTM-Baseline the input data is the concatenation of story and question, the first change in the new architecture was the idea to separate story and question.
By using two separate RNNs (RNN either with LSTM or GRU units), namely "Story-RNN" and "Query-RNN", to process the story and question input streams respectively, the idea is that the Story-RNN can focus more on getting a proper understanding of the story and the Query-RNN on getting proper understanding of the query. Again we use the hidden states of the RNNs after reading in the whole sequences and call it "story-" and "question-code" respectively. Since Story-RNN and Query-RNN are separated, it is important to use the same word embedding for both.
To get a fusion of story and question, we then concatenate the story- and the question-code that are extracted from the RNNs. This fusioned code is then again used in the feedforward classification network to predict the output-answers. This is identical to the classification part of the FB-LSTM-Baseline. The network is also trained by Cross-Entropy-Loss on the output-probabilities. Since GRU-Networks are faster to train and achieve still comparable results, we use GRUs instead of LSTMs for the RNNs here. The visualization of the Story-Question-Model-architecture can be seen in figure 10.
With the Story-Question-Model we are able to improve the performance slightly in comparison with the FB-LSTM-Baseline, however it was not a significant improvement. The reason for that might be, that the first time story and question get together is in the concatenation part before the classification. Consequently, the Story-RNN does not know which information is important to answer the question, because it simply does not know the question. So, by processing the story words it cannot focus on the question and cannot ignore unimportant information. Because of the mentioned problem we developed one further architecture that tries to solve this problem.

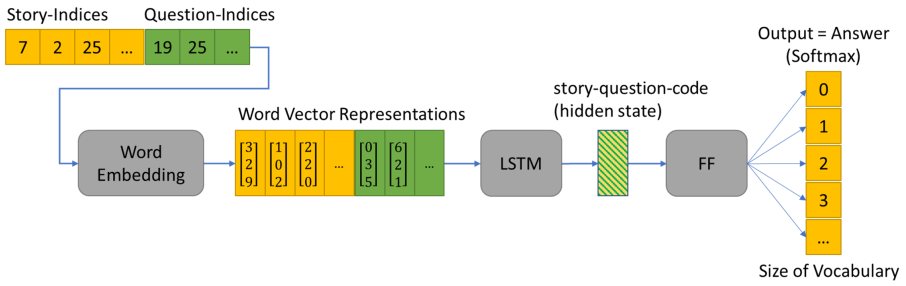

Figure 9: FB-LSTM-Baseline.

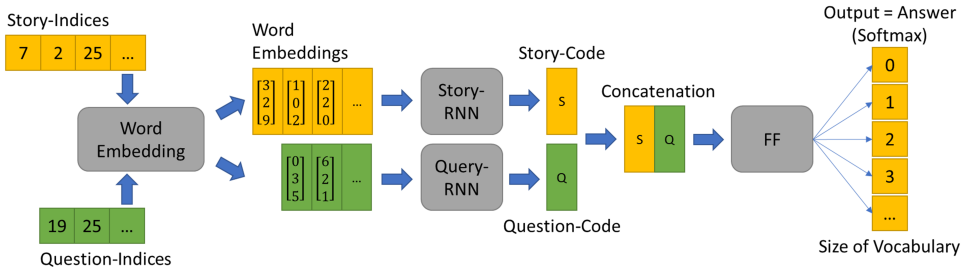

Figure 10: Story-Question-Model.

4.5. Attention-Based-Model

In this section, the architecture of our third model "Attention-Based-Model" is presented. The main idea here is that we want to enable the Story-RNN to focus on the question again. We do this by creating the question code first, and then adding this question code to all the story word embeddings. Then we process the fusioned sequence of story word embeddings and question code by the Story-RNN. The resulting hidden state, called the "Fusion-Code", is then used in the feedforward-classification, similar to the previous models. The entire process can be seen in the visualization of the architecture in figure 11.

Although the approach sounds quite simple, it is yet effective. With this architecture we could achieve the best overall results in the bAbI-tasks (for more details see chapter 5.). The success of this model in comparison to the previous models might come from the kind of "attention-mechanism", that is applied in the model. By adding the question-code to every single Story-Word-Embedding, the information about the question is existent in every single Story-RNN time step. Consequently, the Story-RNN can focus more on the question and is able to evaluate the incoming information based on the relevance for answering the question. We thought about this process as a kind of attention mechanism, as it is also known from attention-based machine translation. This is also the reason for the naming of this architecture.

The attention-based model is our main model, because it achieved the best results on the given bAbI-tasks. To avoid overfitting to the training data, we also added Dropout-Layer with a high dropout-probability (for specs, see table 8. in the attachments). Since we also wanted to evaluate the difference between using GRU or LSTM for the recurrent basis, we implemented this model separately with GRU and LSTM-Cells. The results of this evaluation can be seen in chapter 5..

4.6. Relation-Network-Model

With the Attention-Based-Model, we already have a strong model for solving the given bAbI-Tasks. However, to have a comparison we also implemented one further model. Here we implemented the Relation-Network architecture for natural language data, presented in (Santoro et al., 2017). We decided to implement this model, because it is not completely different to our existing architectures. Furthermore, according to the paper, they could succeed on almost all 20 bAbI-tasks and could achieve more than 83% accuracy on all tasks. Unfortunately, we could not reproduce these numbers with our implementation.
Relation-Networks are specifically designed to do relational reasoning. The authors describe the property of this network as "[...] the capacity to compute relations is baked into the RN architecture without needing to be learned, just as the capacity to reason about spatial, translation invariant properties is built-in to CNNs [...]" (Santoro et al., 2017). Since the bAbI-tasks also need a kind of reasoning about the facts of a story, the relation-network should be a suitable architecture for this problem. In the following, a brief overview of the relation-network-architecture, in the way we implemented it, is presented.

4.6.1. Preprocessing

In contrast to our other models, here the story is separated into its single facts. Then every single fact-sequence of word-embeddings is read-into the Fact-RNN, to generate a "fact-code". The fact-codes get tagged by the number of the

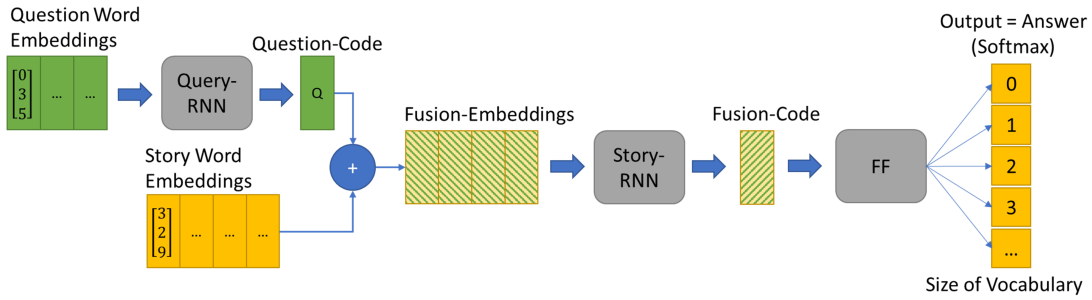

Figure 11: Attention-Based-Model.

order they occur in the story. This assures, that the correct order of facts can be considered in the further processing. The next step, is to generate "fact-combinations", where all pairs of facts are concatenated with the question-code. To get a better understanding, this process is visualized in figure 12.

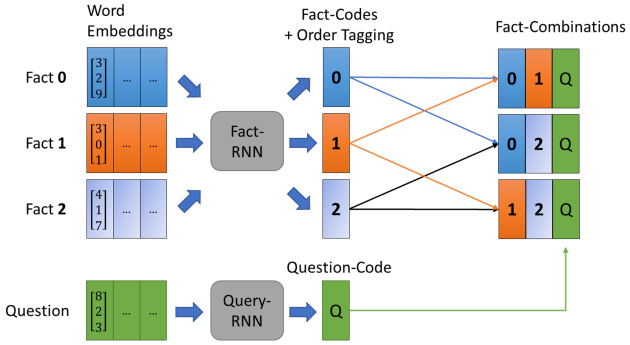

Figure 12: Relation-Network-Model: Preprocessing.

4.6.2. Answer-Finding

The fact-combinations that are created in the preprocessing part of the relation-network, are then processed by a feed-forward network, called "Inference Network", that should find out, if the facts and the question of one fact combination are related to each other. For every fact combination, the inference network creates one "Relation"-code, that should contain that information. All the relation-codes are then summed up, to get invariance of the relation order. The resulting fusion-code is then processed in a second classification-network to predict on the output-answers. This last step is identical to the classification-part of our previous models. The answer-finding process is visualized in figure 13.

4.6.3. Relation-Network Discussion

The relation-network should be particularly suitable for the bAbI-task "two-supporting-facts", because in this case exactly one fact combination has the correct information inside (the two supporting facts and the question). However, through combining all relations, also all other tasks should be solvable.

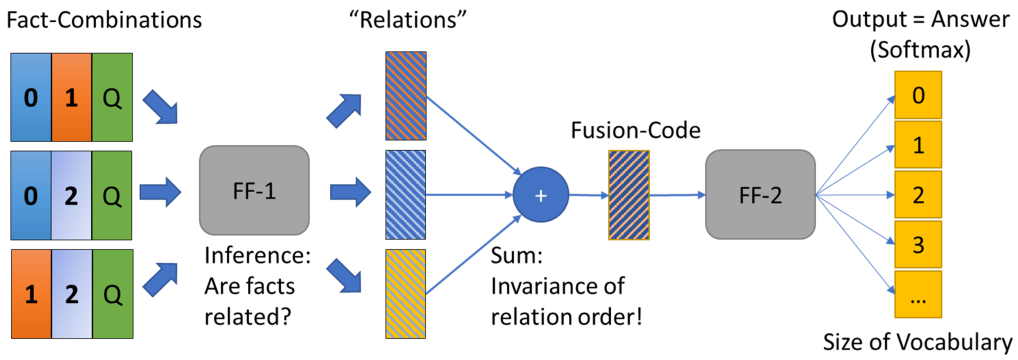

Figure 13: Relation-Network-Model: Answer-Finding.

5. Evaluation

5.1. Evaluation Metric

We use the accuracy of the model as the metric to be optimized. We define accuracy as the amount of correctly predicted responses divided by the number of all predictions

$$acc = number_{correct}/number_{predictions}$$

5.2. Evaluation Protocol

To evaluate the performance of our model we designed a simple repeatable experimental setting, which is described in the following protocol.

First we split the train dataset in two parts, training and evaluation following the 80/20 rule. We use the 80% of the training data to fit the model while we record the loss and the accuracy, both against the training data and the evaluation data. This is done for 50 iterations. Then we use the accuracy over the iterations against the 20% validation data to implement an early stop criteria, so that our model does not overfit. Shortly, we select the best number of epochs to train the actual model using the train-validation split. Then we use the provided number of epochs to train with all the training data and evaluate against the training dataset.

Having the number of epochs for training, we merge the two splits of the train dataset together and use it to train the model. We record the performance and the metrics of the model at each epoch of the training. During training we collect loss data, accuracy data, best 3 predictions, best 5 predictions, where the best n predictions are the best n values of the softmax classifier at the end of the model. The questions, the stories and the answers are saved as well for the later analysis. Finally we report the accuracy of the last epoch against the test dataset. This procedure is done fully automatically in order to reduce our chance of tweaking the model to get higher results even unintentionally.

5.3. Overview Of The Models

In this section we present the performance of the models compared to each other. Since at the beginning of our work we managed to replicate the results of the FB-LSTM-Baseline (FBB), but discarded the model as not sufficiently accurate, we are copying the results from (Weston et al., 2015). Our story-question model (SQM) was slightly better, but still insufficient compared to our attention based model (ABM). We did not manage to replicate the results with the relation network model (RNM) from (Santoro et al., 2017), mainly because it was too slow to train with our implementation. For that reason we focused our efforts to train and evaluate the ABM in two different variations - GRU and LSTM based models. Additionally we also introduce the so called Multinet. The Multinet the standard GRU-ABM, pretrained on all 4 train datasets for 20 epochs and then tuned on the individual datasets using the standard train-evaluation-test protocol.

5.4. Overview Of The Results

The following table presents a brief overview of the accuracy of the models, which we evaluated in detail, compared to the FBB. One can see that the Multinet outperforms the other models in all tasks but in the second. We did multiple runs over the course of development and observed this drop in performance in the results sporadically. Since we trained with the same parameters, data and settings, the only explanation is that it is due to the randomness when shuffling the batches and the random initialization. In most cases the result was better than the other networks.

Task	FBB	GRU-ABM	LSTM-ABM	Multinet
1	50%	89.3%	70.6%	94.9%
2	20%	36.6%	31.8%	33.7%
3	20%	26%	30.3%	45.1%
6	48%	63.3%	48.9%	81.2%
Average	35.4%	53.8%	45.4%	63.7%

Table 1: Summary Of The Results

Table 1 depicts that the proposed 3 models clearly outperform the FBB.

5.5. Convergence And Generalisation

The accuracy of the test and evaluation data-sets is the key indicator if the network converges and if it tends to overfit. The following plots show the accuracy curves of the Multinet on the different tasks. We chose to present Multinet as our best performing model. Figure 14 shows that on one supporting fact the Multinet achieves near perfect result in 7 iterations of finetuning with 20 interations of pretraining. One can clearly see how the two curves converge

together towards the 90%. The dropout method in combination with the early stopping procedure are preventing the net from overfitting without causing underfitting. However this is not the case with tasks 2 and 3. Those tasks require relational reasoning, which means that the network needs to calculate which facts are related and which not in order to produce a good prediction. In Figure 15 and Figure 16 we see that the network overfits, thus being able to learn the training dataset, but not being able to generalize. We did experiment with larger dropout, but those experiments resulted in underfitting. Our interpretation of this behavior is that the network does not have the architecture complexity to extract the relevant relationships between the facts or that the dropout method is causing the network to be unable to relate between different facts since different connections are allays missing. An alternative approach to the problem would be to have a different kind of regularization to push the convergence in the right direction and eventually not to break the inter-factual reasoning of the network. Such different regularization approach is the L2 regularization, which can be shown is equivalent to dropout, but not so destructive. In the next section we show, that the accuracy measured by taking the best 3 predictions is a lot better, thus implying that the model can find the relationships between the facts, but the confidence in the correct answer is lower. We did not test that hypothesis by implying different regularization strategy.

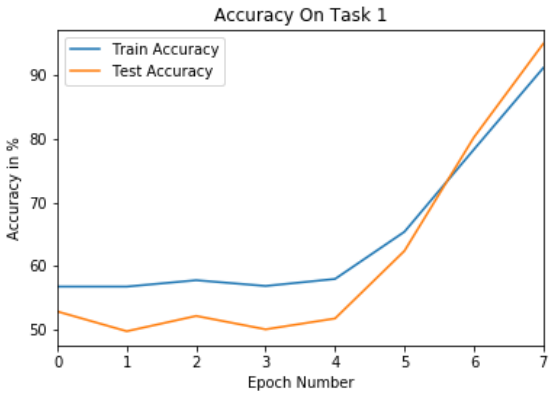

Figure 14: Accuracy Of Multinet

Best Predictions Investigating the best 3 answers, shows that the accuracy of the different models can be increased by carefully finetuning the network. In this section we present the best 3 accuracy of our models. We say that a prediction is best 3 accurate if one of the 3 best predictions equal to the ground thru.

False Prediction Analysis Since we are working with text, the next reasonable question is how exactly did our models fail. In this section we are presenting part of the mistakes that Multinet did on the third task, since it was the most complicated one. We note the ground truth as GT and the predicted answer as PT.
The rest of the mistakes on the third problem are also exclusively locality related.Table 3 indicates that the network has mainly problems with answering questions about a place or

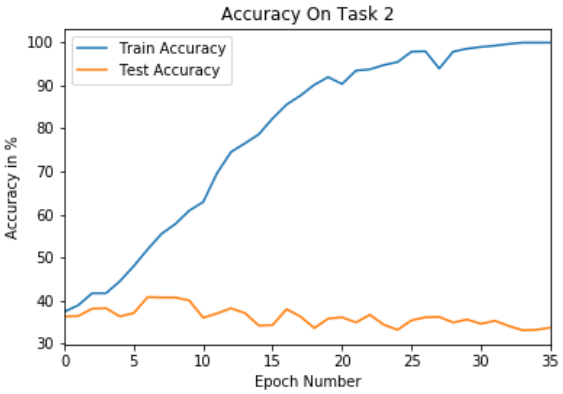

Figure 15: Accuracy Of Multinet

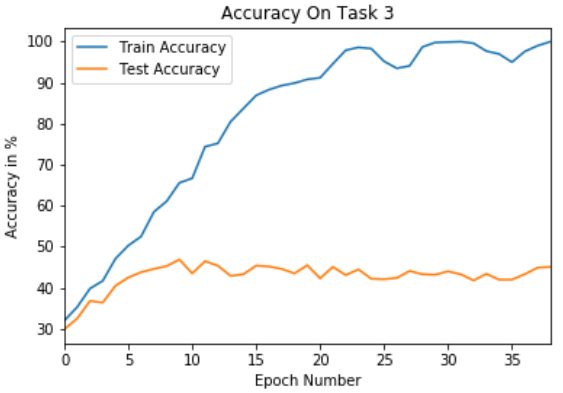

Figure 16: Accuracy Of Multinet

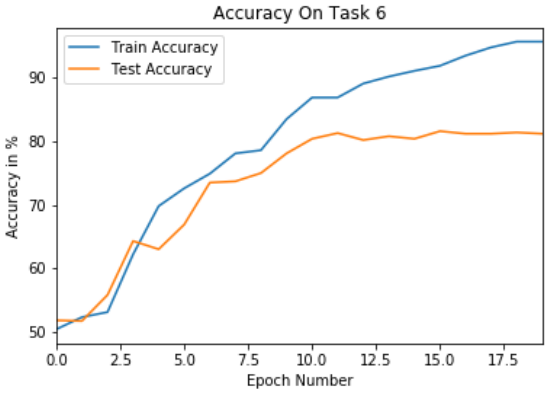

Figure 17: Accuracy Of Multinet

a locality. It also shows that the network understands the concept of a locus and predicts an answer that is a locus. Evaluating the stories in more detail suggests that the problem is most likely related with the temporal ordering of the locations in the story. Further evaluation is needed in order to validate this hypothesis.

Task	GRU-ABM	LSTM-ABM	Multinet
1	98.9%	95.7%	99.5%
2	78.3%	73.2%	75.9%
3	67.1%	71.7%	82.8%
6	100%	100%	100%
Mean	86%	85.2%	89.6%

Table 2: Best 3 Predictions Accuracy

GT	PA
bathroom	hallway
hallway	bedroom
garden	bathroom
hallway	bathroom
office	bathroom
bedroom	office
garden	kitchen
office	kitchen

Table 3: False Predictions

5.6. Discussion

The presented plots clearly show that the proposed models evaluate better on the tasks 1,2,3 and 6 of the QA bAbI dataset, compared to the original Facebook benchmark LSTM model. However they are still not as good as the memory network from (Weston et al., 2015) and suffer from overfitting on tasks 2 and 3. During the evaluation we showed that the models actually are quite close to predicting the correct answer, but need to be fine-tuned in order to boost the confidence of the correct predictions. A new scheme for regularization should be employed for the tasks 2 and 3, with L1 regularization being the preferred candidate, because it would allow for more complex models, while removing the unnecessary connections. Additional to L1 one should consider L2 regularization as a substitute for the dropout. Using the Multinet we showed that using data with different structure of the facts and their relations greatly improves the results. This suggest that using larger datasets, or datasets with higher variance of the relationship structure would greatly benefit the model.

6. Conclusion

Question answering is a fundamental part of the human speech apparatus and as such is a hot topic in the AI community. The rapid development of the field is partially due to the available datasets such as QA bAbi, which provide the researches and practitioners with the fundamentals to develop and evaluate models for the task at hand. Notable approaches in the field are simple RNN-based methods and the dynamic memory networks. For this lab we decided to develop our own model, even if it is not so good as the dynamic memory networks, which seem to be the best performers in the field (Weston et al., 2015).

We presented and evaluated 3 different models for the QA bAbi Dataset. The 3 models are identical in structure, but either rely on different building blocks - LSTM/GRU or use different training strategies. The 3 models perform well on the first and sixth tasks, but suffer underperformance on the second and the third. During the evaluation we found out that the models overfitted the later two tasks, and need different regularization strategies. This gives a direction for future work. The first issue to solve, would be to fix the overfitting. Preventing the network from overfitting would allow the use of more complex models, which are better for task of temporal relational reasoning. The fine-tuning is the next important step, which should guarantee better convergence toward the correct answers, as we showed with Table 2.

New training strategies such as semisupervised learning might prove helpful for increasing the performance of the models, as well for building new and larger datasets.

7. Bibliographical References

Christopher Olah. (2015a). Gru cell. [Online; accessed February, 2018].

Christopher Olah. (2015b). Lstm cell. [Online; accessed February, 2018].

Santoro, A., Raposo, D., Barrett, D. G. T., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. P. (2017). A simple neural network module for relational reasoning. *CoRR*, abs/1706.01427.

Weston, J., Bordes, A., Chopra, S., and Mikolov, T. (2015). Towards ai-complete question answering: A set of prerequisite toy tasks. *CoRR*, abs/1502.05698.

8. Appendix

Hyperparameter	
GRU/LSTM-Networks	1 Layer
Story-hidden-size	100
Query-hidden-size	40
Embedding-size	40
Learning-rate	0.001
Batch-Size	256
Feed Forward Network	1-3 Layer
Dropout in GRU	70-90%

Table 4: Hyperparameter of the Attention based model.