# 1. Project design

The goal of my project was to build a question answering network that performed well on Facebook's bAbI tasks.

# 2. Tools

1. `keras` for deep learning models.
2. `spacy` for text tokenization.
3. `pandas` for saving and aggregating model performance.

# 3. Data

The primary bAbI dataset contains 1000 training and 1000 testing examples for each of 20 different types of question answering task.

# 4. Algorithms

## 4.1. Baseline model

My baseline model encoded questions with two bidirectional GRU layers. The final states of these layers was fed to a companion pair of decoder GRU layers which took as input the question as well as the output of the decoder at the previous time step (or the <START> token on the initial time step). During training, but not testing, the decoder was always given the correct previous word (teacher forcing).

## 4.2. Dynamic memory network

For my second model, I implemented a Dynamic Memory Network, as described in this paper. In this model, the encoder RNN emits a separate fact vector at the end of each sentence. Another RNN network takes these fact vectors as inputs and outputs an episode vector that consolidates the facts. This episode generating RNN makes multiple passes over the input, attending to different fact vectors on each pass, and yet another RNN iteratively consolidates the emitted episode vectors into a single memory vector. At each pass, the current contents of the memory vector guides what the episode generating network attends to. Initially, the memory contains only the question, but new information is added after each pass until the memory contains all information needed to answer the question.

My network differed from the original in several ways:

- The original network decided on its own how many passes to make over the input sentences, while my network made a fixed number of passes. I used 3 passes as a default as I never found using more than 3 passes to improve the results. Generally, unneeded passes made the network harder to train and slightly reduced accuracy, possibly because it allowed the network more opportunity to attend to useless information.
- I found that the RNN responsible for consolidating the episode vectors into a single memory vector didn't actually improve results. It's sufficient to use the previous episode to guide the attention of the next iteration and then throw it away. In this version, the episode vector of the final iteration is passed directly to the decoding RNN, taking the place of the memory vector.

- In the original network, the input RNN processed the input in a continuous sequence, emitting a fact vector after each sentence. In my network, each sentence is its own input and I encode each sentence separately, using the same encoder and weights. I tried keeping the hidden state between encodings to more closely match the original implementation, but I found the network easier to train if I encoded each sentence with an empty initial hidden state.
- In the original network, the gate weights were applied to the episode generating RNN's hidden state as it passed between time steps using the equation: $w_t * h_t + (1 - w_t) * h_{t-1}$. The advantage of this approach is that if a time step has a 0 weight, it effectively didn't happen and the previous hidden state moves forward unmodified. In my network, I applied the gate weights to the fact vectors before passing them into the episode generating RNN. This seemed to work reasonably well, but it does mean that that the input to the episode generating RNN was separated by potentially dozens of time steps with 0 or near 0 input, which isn't ideal as even with zeroed input, bias terms could potentially result in a change in state. I did try implementing the original version with `Keras`'s functional interface by applying the episode generating RNN one time step at a time, weighting the hidden state using a lambda layer, and then feeding the result to the episode generating RNN at the next time step, but this made the network too slow. The right way to do this is probably to implement a custom GRU layer that takes the gate weights as an additional input, but I didn't have time to write one during the course of this project (though it's high on my list of things to do post graduation).

### 4.3. Gate supervision

The bAbI dataset contains a list of the sentences in each story that are required to solve each problem. I utilized these by creating a separate output in the training model for the gate weights during the last iteration of the memory module, explicitly penalizing the network if it didn't figure out what to pay attention to by the final iteration.

## Results

Success on a bAbI task requires getting 95% or greater accuracy on the test set for that task. My baseline network passed a single task while my DMN implementation passed 16. Below are my accuracy numbers for the 3-iteration version of the model I presenting in class, as well the accuracy scores for the DNM model trained without gate supervision.

| Task | Baseline | DMN | DMN w/o gate supervision |
|---|---|---|---|
| 1. Single supporting fact | 48.4% | 100% | 98.6% |
| 2. Two supporting facts | 19.2% | 30% | 34.6% |
| 3. Three supporting facts | 17% | 30.4% | 43.8% |
| 4. Two argument relations | 74.6% | 100% | 96.4% |
| 5. Three argument relations | 81.6% | 98.2% | 75.2% |
| 6. Yes/no questions | 46.8% | 99.8% | 67% |
| 7. Counting | 79% | 98.8% | 80% |
| 8. Lists/sets | 74% | 99.6% | 74.8% |
| 9. Simple negation | 59.8% | 99.8% | 64% |
| 10. Indefinite knowledge | 46.4% | 98.4% | 71% |
| 11. Basic conference | 74% | 100% | 71% |
| 12. Conjunction | 78% | 100% | 82.2% |
| 13. Compound coreference | 94% | 99.8% | 93.8% |
| 14. Time reasoning | 35.8% | 99.2% | 43.8% |
| 15. Basic deduction | 56.6% | 100% | 54.4% |
| 16. Basic induction | 48.8% | 98.6% | 48.8% |
| 17. Positional reasoning | 61.2% | 60% | 62.6% |
| 18. Size reasoning | 93.4% | 99% | 95.8% |
| 19. Path finding | 8% | 22.4% | 8.4% |
| 20. Agent's motivations | 97.6% | 100% | 97.8% |

## What didn't work

- I didn't pass task 2 or 3, which the Sanford model I based mine was able to pass. I did once get over 90% accuracy on task 2 using a 2-iteration version of the model, but I couldn't repeat it so didn't feel comfortable putting it the results table.
- I didn't get to my stretch goals.