Intelligent Agents
Assignment 3

Erik Norlin, 19970807-9299

March 6, 2023

Problem 3.1

Implementation

An information-retrieval (IR) chatbot was implemented in C# .NET Windows Forms that provides the user with an interface for input, chatbot response, dialogue history, and a view of dialogue sentence pairs (queries and responses) from the data set ”Cornell Movie Dialogue Corpus” that the chatbot is trained on. This data set contains a large set of consecutive movie lines (dialogues) from different movies, which is useful to train a chatbot on because this can help simulate a more natural conversation. From this data set, a dialogue corpus consisting of paired consecutive movie lines between two characters (queries and responses) was generated by first letting all sentences in the data set undergo text pre-processing where it was cleaned and tokenized. Then, a vocabulary of all distinct words from the data set was generated. The vocabulary was then used to represent the movie lines in lists of word indices from the vocabulary to be able to handle integers rather than strings, which is computationally cheaper.

When matching consecutive sentences, the file ”`movie_conversations`”, containing information about what lines are consecutive between two characters, was used to match consecutive dialogue sentences from the file ”`movie_lines`” containing the actual movie lines. This was built into the code so the user does not have to actively do anything for this to happen. Only sentence pairs where the query and the corresponding response that were shorter than 20 tokens each were saved to the corpus. This was done because too long sentences was considered less valuable in a chatting context, and also to decrease computational cost when embedding the sentences. The paired sentences of the dialogue corpus were generated in such a way that they were ”overlapping”. Overlapping meaning that in a dialogue of N of consecutive movie lines between two characters, each line is a response to the next. For example, a dialogue of four consecutive movie lines between two characters would then make three dialogue pairs. This makes sense because it is possible to keep responding to the previous response in a dialogue, as can be seen in table 1. This was done to the extent that was possible because the information in ”`movie_conversations`” is not organised in an optimal way to be able to overlap all overlapping sentence pairs.

Table 1: An example of what three ”overlapping” sentence pairs could look like between two characters.

Query 1	”Hey”	Response 1	”Hey, how are you?”
Query 2	”Hey, how are you?”	Response 2	”I’m fantastic today!”
Query 3	”I’m fantastic today!”	Response 3	”Well, good for you!”

For embedding, every query from each sentence pair was then embedded as a TF-IDF vector. An IDF value is the probability that a particular word occurs in a sentence, which is computed for every word in the vocabulary. A TF vector holds information about how often every word in the vocabulary occurs in a particular sentence. The TF-IDF vector of a query is obtained by multiplying element wise the TF vector with the IDF values of the corresponding words. The data processing was done after every dialogue query had been embedded. After that, one could chat with the chatbot. As the user entered an input sentence, the input sentence was embedded as a TF-IDF vector as well. The TF-IDF vector was then compared with every TF-IDF vector of all dialogue queries from the training data by looking at the cosine similarities. The cosine similarity is a measure of how close two normalised vectors are in n dimensions. The closer the cosine similarity is to 1, the closer the vectors are together. The furthest distance two vectors can have from each other is a cosine similarity of 0. Finally, the chatbot response was then obtained by randomly choosing between the top five dialogue queries with the largest cosine similarity values.

Manual

To run this program, first load the file ”`movie_lines`” from ”`File`” drop down menu pressing ”`Load raw data`”. Then, click the button ”`Generate dialogue corpus`”, this will clean, tokenize, pair consecutive dialogue sentences between two characters, and embed the sentences from the loaded data to TF-IDF vectors. This can take 5 to 30 min depending on how fast the computer is since there is a lot of data to process. When done processing, the button ”`Generate chatbot`” will become enabled, click this. The tab ”`Data`” will automatically switch to ”`Chat`” and it is then possible to chat. Write an input to the chatbot in the upper black text box window, then press enter to get a response. The dialogue history shows in the lower black list box window. The generated corpus dialogue, i.e. the set of all paired consecutive dialogue sentences from the loaded data set, can be saved under ”`File`” and then ”`Save dialogue corpus`”.

IR chatbot example

`You:` Hello, Chatbot.
`Bot:` Hello.

`You:` How’s it going?
`Bot:` Can I buy you dinner sometime soon?

`You:` We could go for dinner sometime. What’s your favorite?
`Bot:` French fries and ketchup.

`You:` No hamburger?
`Bot:` Heaven.

`You:` Are you a picky eater?
`Bot:` Sometimes. Look here – something big, probably a school of mackerel clumped together. And staying right with us.

`You:` Mackerel, fries and ketchup. That's your favorite?
`Bot:` Gimme that damn thing.

`You:` Anyway, how's it going with work?
`Bot:` Brilliantly, Care for a pint?

`You:` A pint it is. Great idea!
`Bot:` I know.

`You:` You seem on edge today.
`Bot:` I'm sorry.

`You:` It's alright. Tell me about it.
`Bot:` Can't you see she's trying to kill us?

`You:` Chatbot, you're being paranoid!
`Bot:` It's not paranoia. The embedding is very subtle. It's probably been overlooked...

`You:` Enough for today, Chatbot. Good bye.
`Bot:` So long.

Problem 3.2

Here, a successful interaction and three unsuccessful interactions with ChatGPT were carried out mainly to show how ChatGPT can fail when asked questions where common sense reasoning is needed to answer correctly.

Successful interaction

A successful interaction with ChatGPT was carried out where it was asked a question that resembled a simple Winograd scheme as following

`User`: Martin wanted to show appreciation for Jim because all of the encouragement he had received. Who had received encouragement?

`ChatGPT`: In the sentence ”Martin wanted to show appreciation for Jim because all of the encouragement he had received,” it is not explicitly stated who received the encouragement. However, it is implied that Martin received encouragement from Jim. The sentence is saying that Martin wanted to show appreciation for Jim because Jim had encouraged him in some way. We don't know exactly what form this encouragement took or in what context it occurred, but the sentence suggests that Jim had a positive impact on Martin and that Martin wanted to acknowledge that impact by showing appreciation.

`User`: How did you arrive to this answer?

`ChatGPT`: I arrived at this answer by analyzing the sentence ”Martin wanted to show appreciation for Jim because all of the encouragement he had received.” In this sentence, ”he” refers to Martin, and ”Jim” is mentioned as the person who Martin wants to show appreciation for. The sentence suggests that Jim had encouraged Martin in some way, and that this encouragement was significant enough to prompt Martin to want to express appreciation. While the sentence doesn’t provide explicit information about who received the encouragement, the context implies that it was Martin who received the encouragement from Jim. This is because Martin is the one who wants to show appreciation to Jim, and the sentence suggests that Jim’s encouragement was directed towards Martin. Therefore, we can infer that Martin was the one who received the encouragement.

`*End of interaction*`

Impressively enough, ChatGPT could successfully give a correct answer to a Winograd scheme, and then successfully describe how its’ answer was correct. The question at hand who had received encouragement is not explicitly given in the question which makes this question tricky for a chatbot, but not for a human. ChatGPT could figure out that it was indeed Martin that has received encouragement from Jim and presented this clearly why it was so.

Unsuccessful interactions

Three unsuccessful interactions were carried out to demonstrate that ChatGPT can fail at fairly simple questions involving common sense reasoning.

Interaction 1: Change of light bulbs

`User`: Assume that there is only 1 lamp socket in the world. If it takes 1 person to change a light bulb, how many people does it take to change 10 light bulbs?

`ChatGPT`: If there is only one lamp socket in the world, only one person can change a light bulb at a time. Therefore, to change 10 light bulbs, it would take 10 people, each changing one bulb.

`User`: How did you arrive to this answer?

`ChatGPT`: I arrived at this answer based on the assumption that there is only one lamp socket in the world, which means that only one light bulb can be changed at a time. Therefore, if 10 light bulbs need to be changed, it would require 10 people to change them, with each person changing one bulb. Alternatively, if there was a requirement that all 10 light bulbs needed to be changed at the same time, then it would require a special device or mechanism to replace them all at once, and the number of people required would depend on the complexity and nature of the mechanism. However, in the absence of such a mechanism, the most practical solution is to have one person change one light bulb at a time.

`*End of interaction*`

The nature of this question implies that only one light bulb is in use at any given time, meaning that only one person can change a light bulb at a time. If a person change a light bulb, the same person can change the light bulb again. This itself means that it only takes one person to change ten light bulbs. The answer ChatGPT gives is not necessarily completely false because ten people could change one light bulb each. This answer however is not considered to be reasoned with common sense.

Interaction 2: The colour of a polar bolar

`User`: A polar bear strides along the west side of a river going north. On the east side of the river strides a brown bear. The bear on the east side of the river is younger than the bear on the west side. What color is the older bear?

`ChatGPT`: The question does not provide enough information to determine the color of the older bear. The only information provided is that there is a polar bear on the west side of the river going north, and a brown bear on the east side of the river who is younger than the bear on the west side. The age of the bears is mentioned, but not their colors, so we cannot determine the color of the older bear based on the information given.

`User`: How did you arrive to this answer?

`ChatGPT`: I arrived at the answer that the color of the older bear cannot be determined because the question only provides information about the direction of movement of two bears on opposite sides of a river and their relative ages. The color of the bears is not mentioned, so we cannot make any assumptions about their colors. Without any further information about the color of the bears or any other relevant details, we cannot determine the color of the older bear based on the information given.

`*End of interaction*`

The answer to this question is that the colour of the older bear is white because the older bear is the polar bear. It is clearly explained in the question that the bear on the west side of the river is a polar bear and is the older bear of the two. This is quiet easy to figure out for an adult human, but not for ChatGPT.

Interaction 3: The colour of a box

`User`: In a one box lies a sleeping cat, in a another box lies an awake cat. The color of a box symbolises the state of the cat in it (asleep = red and awake = green). What color is the box that has a sleeping cat in it?

`ChatGPT`: There is no information given to determine which color box contains the sleeping cat. The color of the box only represents the state of the cat inside it, but it does not provide any clues as to which box contains which cat. Therefore, we cannot determine the color of the box that has a sleeping cat in it based on the information given.

`User`: How did you arrive to this answer?

`ChatGPT`: I arrived at this answer by carefully reading and analyzing the given information in the problem. The problem only states that there are two boxes and one box contains a sleeping cat while the other box contains an awake cat. It also provides information that the color of the box represents the state of the cat inside it (red for asleep and green for awake). However, the problem does not provide any information about which box contains which cat. Therefore, we cannot determine the color of the box that has a sleeping cat in it based solely on the information given in the problem. Hence, the answer is that there is insufficient information to determine the color of the box that has a sleeping cat in it.

`*End of interaction*`

The answer to this question is that the colour of the box that has a sleeping cat in it is red. The information in the question implies that the colour of a box with a sleeping cat in it is red, and the colour of a box with an awake cat in it is green. ChatGPT fails to answer this question as well.

In all three examples it shows that ChatGPT fails to make wider connections between the information given in the questions. All information needed to come up with the right answers are given in the questions. This really empathizes that ChatGPT lacks a world model of common sense.

Problem 3.3

Pseudo code was written for finding elements in DAISY based on input questions for two different scenarios. In these scenarios a person is looking to buy a car at a car dealing company.

Scenario 1: *Do you have any BMW from 2020?* i.e. *Do you have any* `$brand` *from* `$year`?

```
FindAll(brand=BMW)->$v1

if Count($v1) > 0
    Extract($v1;year=2020)->$v2 TARGET(S)

    if Count($v2) > 0
        BMW models from 2020 found
    else
        BMW models found but not from 2020
else
    No BMW models found
```

First, we assume that each data item has a tag value for specifying the vehicle brand name. Using the cognitive action `FindAll`, a pointer `$v1` is returned that points to a list containing all items with `$brand=BMW`. Next, `Extract` is used to get a pointer `$v2` that points to a list with BMW models from `$year=2020`. Thus, `$v2` is the target here that points to the list with all the BMW models from 2020 that the car dealing company possess. Whenever `FindAll` and `Extract` is used an if-statement is followed with a condition using `Count` to check if the list the retrieved pointer points to contains any items. If it does, the pseudo program proceeds to the next operation to continue to search for the specifications of the input question.

Scenario 2: *Do you have any BMW from 2015 or later, priced at or below 200000 SEK?* i.e. *Do you have any* `$brand` *from* `$year` *or later, priced at or below* `$priceLimit`*?*

```
FindAll(brand=BMW)->$v1

if Count($v1) > 0
    Extract($v1;year>=2015)->$v2

    if Count($v2) > 0
        Extract($v2;priceLimit<=200000)->$v3 TARGET(S)

        if Count($v3) > 0
            BMW models from 2015 or later pricing at 200000 SEK or below found
        else
            BMW models from 2015 or later found but not pricing at 200000 SEK or below
    else
        BMW models found but not from 2015 or later
else
    No BMW models found
```

It is assumed here as well that each data item has a tag value for specifying the vehicle brand name. The same method of steps are used here too as in the first scenario, however with one more level of boolean operation to also extract BMW models from 2015 or later within a price range. To extract a list of elements in an interval (BMW models from year 2015 or later) simply an inequality is used instead of an equality. Here, the pointer `$v3` is the target that points to the list with all the BMW models from 2015 or later pricing at 200000 SEK or below that the car dealing company possess. If-statements with conditions using `Count` are used here as well to check if the company possess any cars with the desired properties from the input question.

Problem 3.4

A DNN based chatbot from Google Colab was implemented and compared to the IR chatbot implemented in section 3.1. Here it is also explained how the DNN chatbot works.

DNN chatbot example

`You:` Hello.

`Bot:` hello .

`You:` How’s it going?
`Bot:` fine .

`You:` Good to hear. Me too.
`Bot:` you re not gonna get the money ?

`You:` Uh... what money?
`Bot:` birth .

`You:` Birth, what are you talking about?
`Bot:` you know what i mean .

`You:` I don’t know what you mean. Can you please explain?
`Bot:` what s the matter with you ?

`You:` Nothing
`Bot:` you know what i m saying .

`You:` No, what are you saying?
`Bot:` i m not sure .

`You:` You seem on the edge today
`Bot:` i m sorry

`You:` It’s alright. Tell me about it.
`Bot:` you know what i mean .

`You:` You have to be more clear now
`Bot:` i m not going to kill him !

`You:` Are okay? you seem all over the place
`Bot:` i m not sure .

`You:` Anyway, how’s work going?
`Bot:` fine .

`You:` are you sure?
`Bot:` yes .

`You:` are you sure?
`Bot:` yes .

`You:` Have you nothing to ask me?
`Bot:` no .

`You:` What ever you say. Good bye then

`Bot:` i said i was going to see you .

Comparing the interactions with the DNN based chatbot and the IR chatbot I would say that the IR chatbot give a more interesting dialogue. This makes sense because the IR chatbot previously implemented had a corpus with over 166000 sentence pairs whereas the DNN chatbot only had a corpus with roughly 53000 sentence pairs. The IR chatbot has therefore a larger variety of responses. This could however be due to how long sentences one chooses to cut out from the corpus, which is an easy fix. The IR chatbot however allows for different outputs for the same query since the chatbot can choose a response randomly between the top five responses with the largest cosine similarities. This is not possible with the DNN chatbot because the DNN chatbot always chooses the response that is the most similar to the data that the model is trained on. As seen in the DNN chatbot example above, some queries give the same responses which is quite redundant. The training time for the IR chatbot is about 7 min whereas the DNN chatbot takes one hour to train. This shows that DNNs are not always better just because they are more complex. In this case I do prefer the IR chatbot because of the larger variety of responses, how efficient the training is, and that it is interpretable as opposed to the DNN chatbot.

How the DNN chatbot works

This DNN chatbot model is a seq2seq model meaning that it takes in an input vector of sequential order, a sentence for example, and outputs a translated sequence based on the input. An example of this is translating a sentence into another language. In this case the input is a query from the user that the model translates to an appropriate response that the model is trained on, in this case consecutive movie lines.

The architecture of this model consists of an encoder part and a decoder part, where RNNs are used for both parts (see figure 1). The encoder takes in only one token of the input query sentence at a time, and outputs a high dimensional state vector. This state vector along with the next token in the query sentence (in sequential order) is then passed through the encoder in the next time step, at each time step changing the state vector, molding it into a high dimensional representation of the semantics of the whole query sentence at the end. In this chatbot model, a bidrectional variant is used for encoding meaning that two independent RNNs are used as encoders where one is fed the input in reverse sequential order and the other one is fed the input in regular sequential order. The output state vectors of both encoders at each time step are summed together so that the semantics are encoded in both past and future contexts.

The last state vectors from the encoders summed together is then fed into the decoder as input. The decoder then predicts and records the next word of the response sentence. The output of the decoder is then used as input for the decoder for the next time step. At the end, the decoder predicts the most appropriate response sentence based on the training of the model. Worth mentioning is that the decoder has an attention head built in, that is characterised as a fundamental part of transformers, that improves the mechanism of prediction by empathizing tokens that plays a more important part in the semantics of the text. To clarify however, this is not a transformer, it only inherits a characterised part of one.

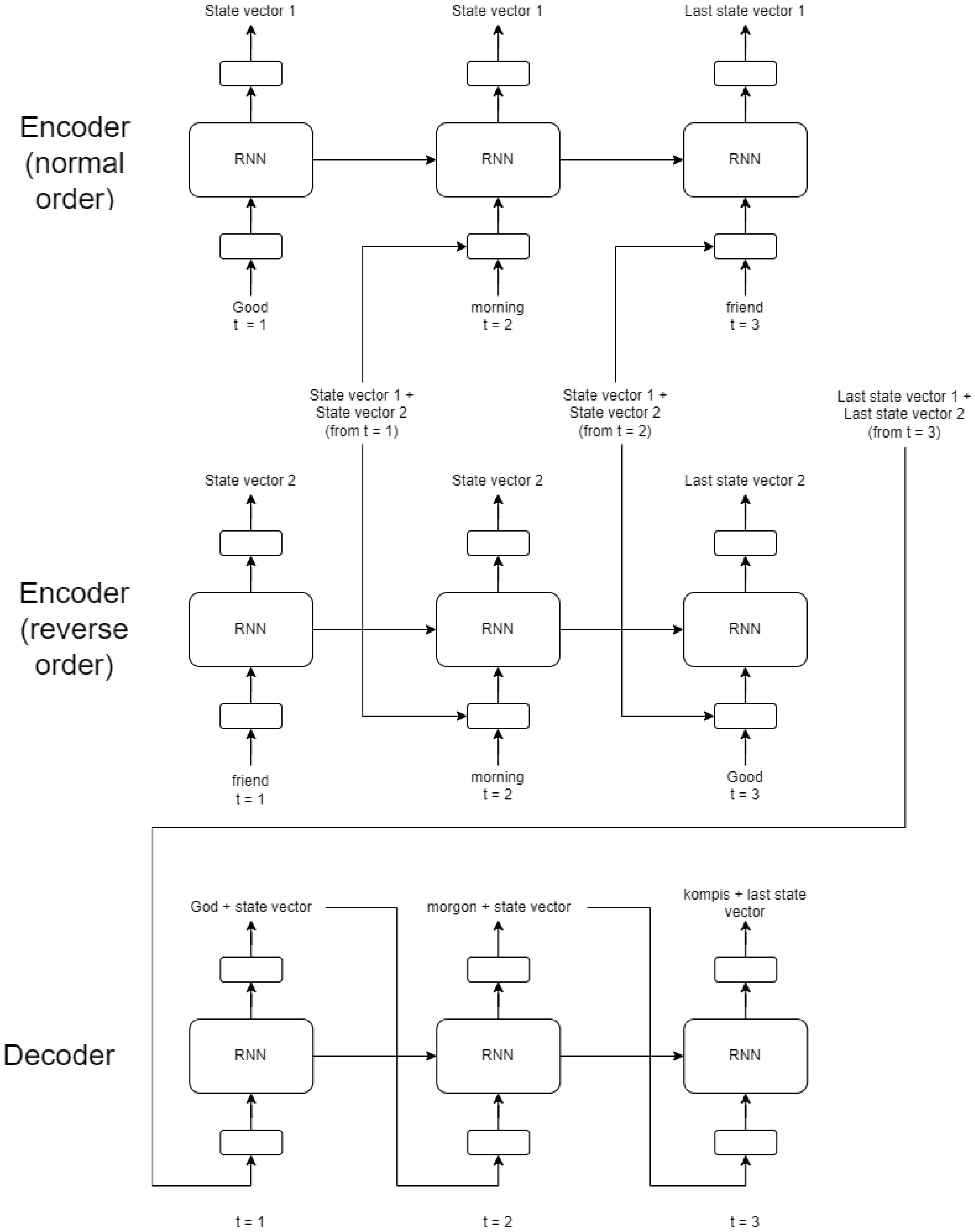

Figure 1: A flowchart of how the DNN chatbot works with language translation as an example. The top two rows are the bidirectional encoder part where the query is processed in normal order in the top row and reverse order in the second. In each time step, the next token of the query in sequential order together with the state vectors summed together from the previous time step is sent into the encoders, outputting new state vectors. The last state vectors summed together from the bidirectional encoder is then sent into the decoder in the third row. The state vector output at each time step is used as input for the next time step until the entire response sentence has been translated.