In-Class Assignment 1: Pretrained and Deterministic Chat Bots

DATA 5420

Name:Dallin Moore

In this introductary assignment we will examine the pros and cons of different phases of chatbots, including determininstic, learning-based, and generative (which you will interact with separately).

We will begin by examining some prebuilt deterministic chat-bots available through `nltk`, then I will provide a brief demonstration in building a chatbot off of conversational pairs, and you will create your own. Finally, I will demonstrate a very basic learning-based chatbot trained with a shallow neural network. You will finally choose an LLM of your choice and hold a brief conversation, then reflect on the pros and cons of each method of chatbot creation.

Import Dependencies

```
import nltk
from nltk.chat.util import Chat, reflections
```

```
reflections
```

```
    {'i am': 'you are',
     'i was': 'you were',
     'i': 'you',
     "i'm": 'you are',
     "i'd": 'you would',
     "i've": 'you have',
     "i'll": 'you will',
     'my': 'your',
     'you are': 'I am',
     'you were': 'I was',
     "you've": 'I have',
     "you'll": 'I will',
     'your': 'my',
     'yours': 'mine',
     'you': 'me',
     'me': 'you'}
```

Try out having a brief conversation with one these chat bots from nltk, what are their strengths and weaknesses?

```
# nltk.chat.eliza_chat()
nltk.chat.iesha_chat()
# nltk.chat.rude.rude_chat()
#
nltk.chat.zen_chat()
```

```
Iesha the TeenBoT
---------
Talk to the program by typing in plain English, using normal upper-
and lower-case letters and punctuation.  Enter "quit" when done.
========================================================================
hi!! i'm iesha! who r u??!
>dallin. what r u up to?
booooring! how old r u?
>16
do u watch anime? i like sailor moon! ^_^
>no. gross.
do u like anime?
>no
ur funny! kekeke
>stop
i wish i was a kitty!! kekekeke ^_^
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-3-7bbd9e0ca31f> in <cell line: 2>()
      1 # nltk.chat.eliza_chat()
----> 2 nltk.chat.iesha_chat()
      3 # nltk.chat.rude.rude_chat()
      4 # nltk.chat.zen_chat()

 3 frames
/usr/local/lib/python3.10/dist-packages/ipykernel/kernelbase.py in _input_request(self,
prompt, ident, parent, password)
    893             except KeyboardInterrupt:
    894                 # re-raise KeyboardInterrupt, to truncate traceback
--> 895                 raise KeyboardInterrupt("Interrupted by user") from None
    896             except Exception as e:
    897                 self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user
```

SEARCH STACK OVERFLOW

Strenths:

- asking questions
- mimicking a specific style (therapist/teen/etc.)

Weaknesses:

- answering questions
- using correct grammer ('how are me?')

Now I'm going to demonstrate how I can make an extremely basic chat-bot by creating a list of statements and reponses for a determinstic chat-bot.

These statements will use some basic *regular expressions*, we will use throughout the semester to perform text cleaning.

Let's try some basic statements and responses for small-talk.

```
pairs = [
    [
        r"(hi|hello|howdy|howzit?)",
        ["%1"]
    ],
    [
        r"my name is (.*)",
        ["Hello %1, how are you doing today?",]
    ],
    [
        r"good|bad|awful",
        ["That's good to hear!, What can I do for you?"]
    ],
    [
        r"Not much. I just wanted to (.*)",
        ["Great! I can %1."]
    ],
    [
        r"quit",
        ["See ya later alligator"]
    ],
]
```

```
def chat():
    print("Howdy! I'm a chatbot here to chat!")
    chat = Chat(pairs, reflections)
    chat.converse()
#initiate the conversation
if __name__ == "__main__":
    chat()
```

```
Howdy! I'm a chatbot here to chat!
>howzit?
howzit
>my name is dallin
Hello dallin, how are you doing today?
>awful
That's good to hear!, What can I do for you?
>Not much. I just wanted to chat with you
That's good to hear!, What can I do for you?
>Not much. I just wanted to chat with you
That's good to hear!, What can I do for you?
>quit
That's good to hear!, What can I do for you?
```

Now you try!

Try creating your own pairs of statements and responses then rerun the chat function to see if you can hold a longer conversation then my very basic one.

```
pairs = [
    [
        r"(display|charging|battery|screen|camera)",
        ["Oh no! I hate when the %1 doesn't work on my phone! Have you tried restarting your phone?"]
    ],
    [
        r"no",
        ["Why don't you try that first!",]
    ],
    [
        r"yes",
        ["That's a tough one! I don't think I'll be able to help you today."]
    ],
    [
        r"it works|it's fixed",
        ["Great! Looks like I solved your issue!"]
    ],
    [
        r"quit",
        ["Thanks for letting me help you!"]
    ],
]
```

```
def chat():
    print("I'm here to help with you broken phone. What part of your phone is broken?")
    chat = Chat(pairs, reflections)
    chat.converse()
#initiate the conversation
if __name__ == "__main__":
    chat()
```

```
I'm here to help with you broken phone. What part of your phone is broken?
>battery
Oh no! I hate when the battery doesn't work on my phone! Have you tried restarting your phone?
>no
Why don't you try that first!
>it's fixed
Great! Looks like I solved your issue!
>quit
Thanks for letting me help you!
```

Then try swapping with a partner (who doesn't know the statements you programmed) and see how far your chat-bot can go before breaking down!

How many turns could your chatbot keep up the conversation for?

Further than I thought it would! the chatbot didn't prove to be very useful, but it does somewhat pointed questions that make it easy to follow and give the answers it wants.

Training a learning-based chatbot with a shallow neural network

Here is a starting point of a few conversational turns, but obviously this is quite limited. Challenge yourself to expand on this list and experiment with how adding data and changing hyperparameters like epochs impacts accuracy.

Don't worry about understanding neural nets for now, the main things to play around with here would be the `conversations` and the number of `epochs`.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import random
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.utils import to_categorical

# sample conversational pairs
conversations = [
    ["hi", "hello"],
    ["what is your name?", "my name is dallin"],
    ["how are you?", "i'm doing well"],
    ["good morning", "good morning to you too"],
    ["how old are you?", "i'm just a program, so I don't have an age"],
    ["who created you?","I was developed by dallin"],
    ["what's your favorite color?","my favorite color is green"]
]

# extract vocaba nd create training data
vocab = set()
for conv in conversations:
  for sentence in conv:
    vocab |= set(sentence.split())

vocab = list(vocab)

# create empty list for inputs (x) and outputs (y)
X = []
y = []
for conv in conversations:
  x = [0] * len(vocab)
  for word in conv[0].split():
    x[vocab.index(word)] = 1
  X.append(x)
  y.append(conv[1])

# adjust y-values to be indices in the response list
y_indices = list(range(len(y)))

# set up training and test set
X_train, X_test, y_train_indices, y_test_indices = train_test_split(X, y_indices, test_size=0.2, random_state=42)

# build shallow neural net model
model = Sequential()
model.add(Dense(8, input_dim=len(vocab), activation='relu'))
model.add(Dense(len(vocab), activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(np.array(X_train), np.array(y_train_indices), epochs=200, verbose=1)

loss, acc = model.evaluate(np.array(X_test), np.array(y_test_indices), verbose=0)
print("Test Accuracy: %.2f%%" % (acc*100))
```

```
1/1 [==============================] - 0s 14ms/step - loss: 2.8407 - accuracy: 0.6000
Epoch 94/200
1/1 [==============================] - 0s 13ms/step - loss: 2.8277 - accuracy: 0.6000
Epoch 95/200
1/1 [==============================] - 0s 19ms/step - loss: 2.8146 - accuracy: 0.6000
Epoch 96/200
1/1 [==============================] - 0s 22ms/step - loss: 2.8014 - accuracy: 0.6000
Epoch 97/200
1/1 [==============================] - 0s 25ms/step - loss: 2.7880 - accuracy: 0.6000
Epoch 98/200
1/1 [==============================] - 0s 22ms/step - loss: 2.7746 - accuracy: 0.6000
Epoch 99/200
1/1 [==============================] - 0s 12ms/step - loss: 2.7611 - accuracy: 0.6000
Epoch 100/200
1/1 [==============================] - 0s 17ms/step - loss: 2.7475 - accuracy: 0.6000
Epoch 101/200
1/1 [==============================] - 0s 14ms/step - loss: 2.7338 - accuracy: 0.6000
Epoch 102/200
1/1 [==============================] - 0s 13ms/step - loss: 2.7201 - accuracy: 0.6000
Epoch 103/200
1/1 [==============================] - 0s 22ms/step - loss: 2.7062 - accuracy: 0.6000
Epoch 104/200
1/1 [==============================] - 0s 22ms/step - loss: 2.6923 - accuracy: 0.6000
Epoch 105/200
1/1 [==============================] - 0s 15ms/step - loss: 2.6783 - accuracy: 0.6000
Epoch 106/200
1/1 [==============================] - 0s 17ms/step - loss: 2.6643 - accuracy: 0.6000
Epoch 107/200
1/1 [==============================] - 0s 27ms/step - loss: 2.6502 - accuracy: 0.6000
Epoch 108/200
1/1 [==============================] - 0s 22ms/step - loss: 2.6360 - accuracy: 0.6000
Epoch 109/200
1/1 [==============================] - 0s 19ms/step - loss: 2.6221 - accuracy: 0.6000
Epoch 110/200
1/1 [==============================] - 0s 16ms/step - loss: 2.6079 - accuracy: 0.6000
Epoch 111/200
1/1 [==============================] - 0s 13ms/step - loss: 2.5937 - accuracy: 0.6000
Epoch 112/200
1/1 [==============================] - 0s 10ms/step - loss: 2.5794 - accuracy: 0.6000
Epoch 113/200
1/1 [==============================] - 0s 16ms/step - loss: 2.5650 - accuracy: 0.6000
Epoch 114/200
1/1 [==============================] - 0s 18ms/step - loss: 2.5505 - accuracy: 0.6000
Epoch 115/200
1/1 [==============================] - 0s 15ms/step - loss: 2.5361 - accuracy: 0.6000
Epoch 116/200
1/1 [==============================] - 0s 15ms/step - loss: 2.5216 - accuracy: 0.6000
Epoch 117/200
1/1 [==============================] - 0s 30ms/step - loss: 2.5071 - accuracy: 0.6000
Epoch 118/200
1/1 [==============================] - 0s 18ms/step - loss: 2.4924 - accuracy: 0.6000
```

```
# save model and vocab used to train model
np.save('vocab.npy', vocab)
model.save('chatbot_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `m
  saving_api.save_model(
# load model
from keras.models import load_model
model = load_model('chatbot_model.h5')

# load vocab
vocab = np.load('vocab.npy')
vocab = vocab.tolist()

# process user input text
def process_user_input(text):
  bag = [0] * len(vocab)
  for word in text.split():
    if word in vocab:
      bag[vocab.index(word)] = 1
  return np.array(bag)

# decode the response back to word from the predicted index number
def decode_response(predicted_index):
    return y[predicted_index]

# chatbot function that takes user input and return the highest probability response
def get_bot_response(user_input):
    input_vector = process_user_input(user_input)
    prediction = model.predict(input_vector.reshape(1, len(input_vector)))[0]
    predicted_index = np.argmax(prediction)
    return decode_response(predicted_index)

# Interact with bot
print("Chatbot: Hi there! Ask me anything.")
while True:
  user_input = input("You: ")
  if user_input == "quit":
    break

  bot_response = get_bot_response(user_input)
  print("Chatbot:", bot_response)
```

```
Chatbot: Hi there! Ask me anything.
You: how are you?
1/1 [==============================] - 0s 95ms/step
Chatbot: i'm just a program, so I don't have an age
You: how are you doing today?
1/1 [==============================] - 0s 18ms/step
Chatbot: i'm just a program, so I don't have an age
You: what is you name
1/1 [==============================] - 0s 19ms/step
Chatbot: good morning to you too
You: good morning
1/1 [==============================] - 0s 17ms/step
Chatbot: good morning to you too
You: who created you
1/1 [==============================] - 0s 17ms/step
Chatbot: good morning to you too
You: what is your favorite color?
1/1 [==============================] - 0s 19ms/step
```

How did your learning-based chatbot compare to your deterministic chatbot? What are the pros and cons of each approach?

```
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-3-86d90643af57> in <cell line: 30>()
```

It was actually worse. Getting the chatbot enough data is difficult. I was not able to come up with enough data that the chatbot could be accurate.

```
     32   if user_input == "quit":
```

Now navigate to the LLM of your choice, and hold a brief conversation. How does it compare, and what are some factors you think may contribute to this difference in performance?

```
/usr/local/lib/python3.10/dist-packages/ipykernel/kernelbase.py in _input_request(self,
prompt, ident, parent, password)
```

The way that the LLM is vastly different from the chatbot. With the chatbot every scenario must be thought out and accounted for, but with the LLM I can ask it tough questions that it may have never heard before and it will still have a useful answer. The large dataset definitely helps; if our chatbots had bigger dataset of Qs and As they would also perform better, but what sets the LLM apart is being able to construct language that it thinks I want to hear.

```
KeyboardInterrupt: Interrupted by user
```