Andromeda :: Poetry Generator
Team Members: Keisha Arnold, Jacob Karcz, Carrie Treeful
CS 467 Capstone
Demonstrate Project- Instructions

Submission Contents

- Zip file of functional code for Keras and TensorFlow model (see content descriptions below)
  - This includes source code for our Keras and TensorFlow models as well as Windows executable files to run the programs
  - Instructions (PDF)

Code contents

*The two directories in bold contain the source code for the different neural network models:

- Keras Bidirectional LSTM model
- Gated Recurrent Language model by Miyamoto and Cho

**See instructions to run the executables in the next section below.

```
demo_char_lstm.exe  (customizable text generating LSTM to train and generate text samples)
generate_char_lstm.exe  (generates a 14 line sonnet from our trained baseline model)
generate_yoon_kim.exe  (generate a poem fragment from the Yoon Kim model)
README.txt
/data
    model_char_lstm.h5
    model_char_lstm.json
    sonnets.txt
    train.txt
    test.txt
    valid.txt
    epoch024_6.1626.model.data-00000-of-00001
    epoch024_6.1626.model.index
    epoch024_6.1626.model.meta
/source_code
    /website
        Dockerfile
        /app
            main.py  (Flask app)
            /static
                fonts.css
                default.css
            /fonts
            /images
        /templates
            index.html
            about.html
            useful_links.html
        /results
            line_number.txt  (saves the spot in the char_lstm_poems.txt file)
            char_lstm_poems.txt  (text file of generated poems)
/demo_char_lstm  (customizable text generating LSTM to train and generate text samples)
    demo_char_lstm.py
    README.txt
    /data
        sonnets.txt
/generate_char_lstm  (generates a 14 line sonnet from our trained baseline model)
    generate_char_lstm.py
    convert.py
    README.txt
 /generate_yoon_kim  (generate a poem fragment from the Yoon Kim model)
   generate_yoon_kim.py
    README.txt
/char_lstm_model (Keras Bidirectional LSTM Model)
    utils.py  (reads and processes the data from the text files)
    test.py  (tests a trained model from file)
    train_BI_3_512_ND_adam.py  (trains a 3-layer bidirectional model with 512 nodes)
    sample.py  (generates text from a trained model file)
    README.txt
    /models
    /results
    /checkpoints
    /logs
    /data
        train_sonnets.txt  (124 sonnets used for training)
        test_sonnets.txt  (15 sonnets used for validation during training)
        val_sonnets.txt  (15 sonnets used for testing)
        sonnets.txt  (text file containing all 154 sonnets)
/gated-rlm  (Gated Recurrent Language Model)
    README.txt
    base.py  (basic helper functions & Keras' Progbar object to visualize training)
    data_preprocess.py  (from M & C source code, some functions were modified)
    layers.py
    gated_model_w_layers.py  (an initial, messy implementation of the model)
    gated_rlm.py   (a cleaner implementation of the model)
```

```
train.py  (training script, though most of the training logic still resides in gated_rlm.py)
/data
    /MnC_dicts
        char_dict.pkl  (character vocab for the model from M & C script)
        word_dict.pkl  (word vocab for the model from M & C script)
    /Yoon_dicts
        char_vocab.pkl  (character vocab for the model from Yoon Kim’s script)
        Word_vocab.pkl  (word vocab for the model from Yoon Kim’s script)
    /Basic_dicts
        char_dict.txt  (simple char vocab without indices, one char per line)
        word_dict.txt  (word vocab with occurrences, for GloVe)
    /shakespeare_corpus
        /raw
                citation-n-legal_stuff.txt
                shakespeare_complete.txt
                Shakespeare_sonnets.txt
        /tokenized
                shakespeare_tokenized.txt  (complete works of shakespeare tokenized)
    /sonnets
        /raw  (our initial text files by Carrie)
                sonnets.txt
                train.txt
                test.txt
                valid.txt
        /tokenized  (same files as /raw, but tokenized)
                sonnets_tokenized.txt
                train_tokenized.txt
                test_tokenized.txt
                valid_tokenized.txt
    GloVe_vectors_trimmed.200d.npz  (trimmed word emb trained on entire corpus)
/tools
    build_dictionary_char.py  (build a char dictionary in cPickle, by Miyamoto et al)
    build_dictionary_word.py  (build a word dictionary in cPickle, by Miyamoto et al)
    tokenize_file.py  (preprocess files fed to the model, dictionary scripts, and GloVe)
```

Instructions to run the executables

These specific instructions to run the .exe files are also in the README.txt located in the same directory as the executables. Since our deep learning models require many different libraries and dependencies, we have included three Windows executable files of our source code that can be run on Windows without having to download anything.

The folder contains three Windows executable files: demo_char_lstm.exe and generate_char_lstm.exe, and generate_yoon_kim.exe. To run the programs, open Command Prompt in Windows and navigate to the folder that contains the executable files. The files rely on information stored in the data directory so they shouldn’t be moved from this location. Below are instructions for running each program.

* Note: Each time a program runs it displays a short message about tensorflow CPU support. This is normal and can be ignored.

Program 1: demo_char_lstm.exe
A customizable text generating LSTM to train and generate text samples. Because our models are far too large to run on a regular CPU, we created this program to demonstrate how the training and tuning process works. The program can be used with the default input text file of Shakespeare’s sonnets or with any text file specified in the input_text argument.

How to run:
Run with defaults in the Windows Command Prompt window type: demo_char_lstm
Run with optional arguments: demo_char_lstm —-epochs=5 —-file=model.h5 —-layers=2

Optional arguments:
--epochs: Number of epochs, type=int, default=1
--layers: Number of hidden layers, type=int, default=1
--nodes: Number of nodes per hidden layer, type=int, default=128
--batch_size: Number of streams processed at once, type=int, default=128
--seq_len: Length of each data stream, type=int, default=50
--optimizer: Optimizer: adam, sgd, rmsprop, type=str, default=adam
--model_type: Type of model: lstm, gru, bidirectional, type=str, default=lstm
--dropout: Dropout value between 0 and 1, type=float, default=0.0
--input_text: Path to input text, type=str, default=data/sonnets.txt
--sample_len: Length of generated text in chars, type=int, default=500
--diversity: Diversity of output between 0 and 1.5, type=float, default=0.5
--load: Path to load model from file, type=str, default=None
--save: Path to save model to file, type=str, default=None
--checkpoints: Interval for saving checkpoint files. They can be used to load the model from a specific epoch. type=int, default=1
--logs: Save log file for use with TensorBoard (0 is false, 1 is true), type=int, default=1)
--reduce_lr: Reduce learning rate on plateau (0 is false, 1 is true), type=int, default=0)

Suggested configurations:
The default arguments create a small model designed to be run on any CPU. As a result, the text output will not be very interesting. To create better results, test out different configurations using the optional arguments. For example:

- Increase the number of epochs
- Increase the number of hidden layers to 2 or 3
- Increase the number of nodes to 256 or 512
- Decrease batch_size to 64 or 32.
- Increase or decrease seq_len
- Save the model and reload it using —-save and —-load. This is helpful for saving and resuming training progress, especially when using slower CPUs.

The source code can be found in source_code/demo_char_lstm/demo_char_lstm.py

Program 2: generate_char_lstm.exe
This program generates a sonnet from our trained baseline model. The baseline model is a character level Bidirectional LSTM. Because of it's large size, it can take up to a few minutes to generate the poem depending on the speed of the CPU.

How to run:
In the windows Command Prompt window type: generate_char_lstm
The source code can be found in source_code/generate_char_lstm/generate_char_lstm.py
The source code for our baseline model can be found in source_code/char_lstm_model

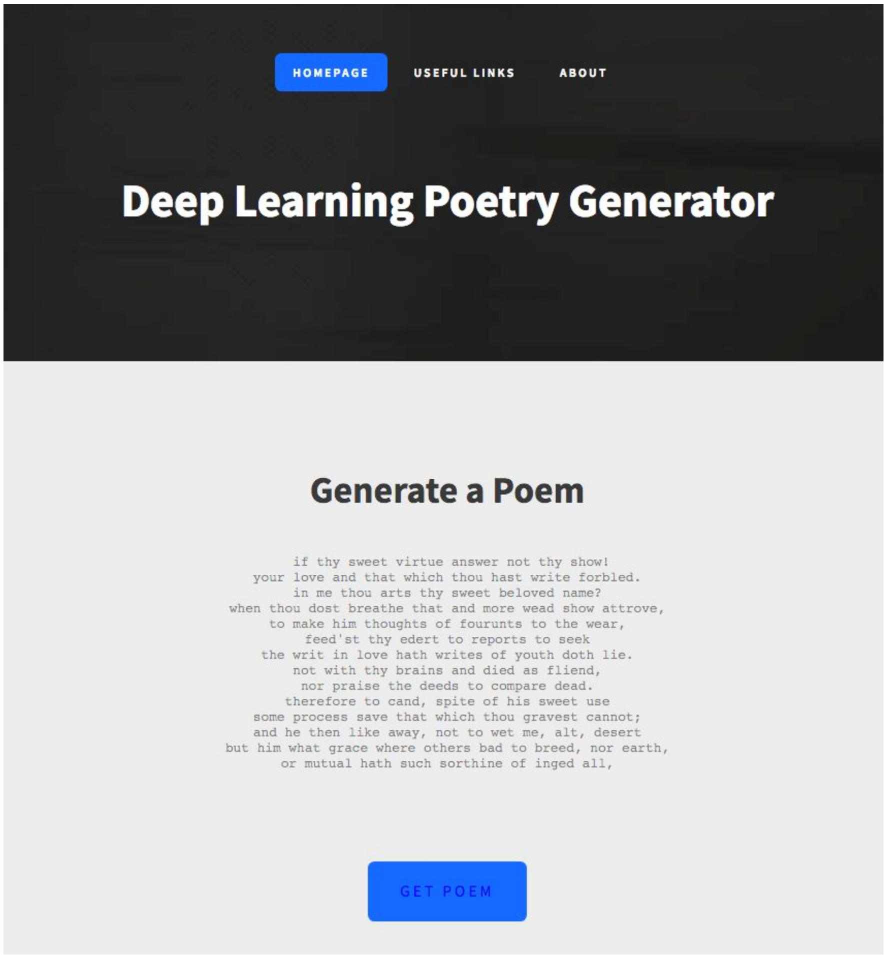
Program 3: generate_yoon_kim.exe
This program generates a sonnet from the Yoon Kim character aware word-level CNN-LSTM model that we trained but did not write. It is included only as an example of what we hoped to achieve with the output of our gated-rlm TensorFlow model and to compare output with our char lstm model. Because of it's large size, it can take up to a few minutes to generate the poem depending on the speed of the CPU.

How to run:
In the windows Command Prompt window type: generate_yoon_kim

The source code we used for the Yoon Kim character aware word-level CNN-LSTM model can be found here: https://github.com/mkroutikov/tf-lstm-char-cnn
The source code for this executable is in source_code/generate_yoon_kim
The source code for our TensorFlow Gated Recurrent Language Model can be found in source_code/gated-rlm

Website

URL: http://ec2-18-217-70-169.us-east-2.compute.amazonaws.com/



Deep Learning Poetry Generator

Generate a Poem

```
         if thy sweet virtue answer not thy show!
     your love and that which thou hast write forbled.
         in me thou arts thy sweet beloved name?
when thou dost breathe that and more wead show attrove,
     to make him thoughts of fourunts to the wear,
          feed'st thy edert to reports to seek
    the writ in love hath writes of youth doth lie.
         not with thy brains and died as fliend,
          nor praise the deeds to compare dead.
        therefore to cand, spite of his sweet use
     some process save that which thou gravest cannot;
     and he then like away, not to wet me, alt, desert
but him what grace where others bad to breed, nor earth,
       or mutual hath such sorthine of inged all,
```

We wanted to create a user interface if the user was interested in getting generated poems from our trained model. We made a simple web application using Flask framework and Jinja2 templating and deployed it on an EC2 instance with Docker and nginx.

The Homepage includes a button the user can click to get a poem from our neural network model. A poem from a machine! What would Shakespeare think?!

The Useful Links page includes links to sources we found helpful during our journey in creating the Deep Learning Poem Generator.

The About page includes some background on the model demonstrated on our web application, as well as the progress we've been making to improve on that model with a TensorFlow implementation of a Gated Recurrent Language Model based on a paper by Miyamoto and Cho.