LSTM Model for playing Hangman

Aditya Raut

September 11th, 2024

1 Task description

The provided file `'words_250000_train.txt'` contains a total of 227,300 words made of small alphabet letters. The Jupyter notebook `Hangman_run_experiments.ipynb'` contains all the necessary functions to play Hangman games after loading a pre-trained model. The task is to maximize the win rate playing the game with 6 lives, i.e., you can make 6 incorrect letter guesses.

2 Proposed strategy

I trained a bidirectional LSTM neural network created using PyTorch locally on a custom dataset created from the given word dictionary. The best model as per my judgment was used for predictions.

- **Dataset creation -**
  - For every `word` in given dictionary, I created 15 data points of 'guessed made in the past'.
  - Every generated set of guesses has the following realistic game conditions -
    * at least 1 letter remaining to guess correctly in the `word`
    * at most 5 incorrectly guessed letters (at most 5 lives lost)
  - Encode the 'masked `word`' of length n containing `_` symbols as a PyTorch tensor of size $n \times 27$, where each of the n rows has 1 at `ord(letter)-97` or at index `26` for `_` and 0 everywhere else.
  - The ideal output of the model should be equal probability for all remaining letters in the `word`.
  - Due to creating 15 different guesses per word, we have over 3.4 million data points.
- **Model architecture -**
  - I used a bidirectional LSTM that first takes input of 'masked `word`' encoded as above.
    * LSTM has `num_layers = 3, hidden_size = 256, dropout = 0.2`
  - This LSTM gives output of size `512`, which is `2 * hidden_size`.
  - We add a dropout layer of 0.4 before connecting this fully to a linear layer.
  - We then concatenate a 1-0 indicator tensor of size 26 indicating all guesses made.
  - This tensor of size `538` is passed to a linear layer of dimensions `(538,128)`.
  - Another dropout layer with 0.4 value between fully connected layers.
  - Finally a linear layer of size `(128,26)`, which gives probabilities for all letters.
- **Loss criterion -** `CrossEntropyLoss`, since this uses a `softmax` function to compare with labels.
- **Training parameters -**
  - Random 9:1 ratio data split for training and validation, `batch_size = 500`.
  - ADAM Optimizer with initial `learning_rate=1e-3`, `weight_decay=1e-5`
  - StepLR scheduler that cuts learning rate by factor of $\frac{1}{2}$ every 10 epochs.
  - Trained for a total of 250 epochs.
- **Final model and guess selection -**
  - Model at epoch 241 had the least validation loss, and was selected for final submissions.
  - The highest probability un-guessed letter from the model is returned by `guess(self, word)`.

3 Important files

1. `Hangman_models_training.ipynb` - Dataset creation and model training
2. `Hangman_run_experiments.ipynb` - to load pre-trained models and run on random or chosen words
3. `Hangman_Strategy_Aditya_Raut.pdf` - This PDF, description of strategy

Very large sized outputs in the training Jupyter notebook are cleared for better readability.