Advanced Methods in NLP Lecturer: Jonathan Berant

Home Assignment 2: Language Models

Due Date: *April 25, 2017*

In this home assignment we will implement an n-gram and a neural language model.

Please copy the data and supporting code from `~dormuhlg/advanced_nlp/assignment2/code`. Note, you will need to re-use some of the code implemented for assignment1 - `gradcheck.py, sgd.py, sigmoid.py` and `softmax.py`.

Your code should run properly on Python 2.7 on linux (it doesnt have to run on Python 3 and above). It is required that it should run on nova.cs.tau.ac.il using /usr/bin/python. Test it before submitting it!

To submit your solution, create a directory at `~dormuhlg/advanced_nlp/assignment2/submissions/<id1>_<id2>` (where id1 refers to the ID of the first student) and put all relevant files in this directory. The submission directory should include the code necessary for running the tests provided out-of-the-box, as well as a written solution, and a text file including an e-mail of one of the students. Don't copy the dataset to your directory! Instead, Add a soft link to the data from your submission directory:

```
cd <your submission directory>
ln -s ~/advanced_nlp/assignment2/code/data/./
```

1 N-gram language model

(a) Implement a trigram language model with linear interpolation. Fill your implementation in `ngram_model.py`. Your implementation should include the training algorithm and model evaluation function. Use `data/lm/ptb-train.txt` to train the n-gram models. Use `data/lm/ptb-dev.txt` to evaluate your perplexity.

(b) Implement grid search to tune the linear interpolation coefficients (λ_i). Find the perplexity for every setting of the coefficients and the setting that maximizes perplexity on the dev set. Add the results of this search to your written solution. Make sure to evaluate with $\lambda_1 = 0$ and $\lambda_1 = \lambda_2 = 0, \lambda_3 = 1$, which will effectively be training bigram and unigram language models.

2 Neural language model

(a) Derive the gradient with regard to the inputs of a softmax function when cross entropy loss is used for evaluation, i.e., find the gradients with respect to the softmax input vector $\boldsymbol{\theta}$, when the prediction is made by $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{\theta})$. Cross entropy and softmax

are defined as:

$$\text{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_i y_i \cdot \log(\hat{y}_i)$$
$$, \text{softmax}(\boldsymbol{\theta})_i = \frac{\exp(\theta_i)}{\sum_j \exp(\theta_j)}.$$

The gold vector $\boldsymbol{y}$ is a one-hot vector, and the predicted vector $\hat{\boldsymbol{y}}$ is a probability distribution over the output space.

(b) Derive the gradients with respect to the inputs $\boldsymbol{x}$ in a one-hidden-layer neural network (i.e., find $\frac{\partial J}{\partial \boldsymbol{x}}$, where J is the cross entropy loss $\text{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}})$). The neural network employs a sigmoid activation function for the hidden layer, and a softmax for the output layer. Assume a one-hot label vector $\boldsymbol{y}$ is used. The network is defined as:

$$\boldsymbol{h} = \sigma(\boldsymbol{x}\boldsymbol{W_1} + \boldsymbol{b_1}),$$
$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{h}\boldsymbol{W_2} + \boldsymbol{b_2}).$$

The dimensions of the vectors and matrices are $\boldsymbol{x} \in \mathbb{R}^{1 \times D_x}, \boldsymbol{h} \in \mathbb{R}^{1 \times D_h}, \hat{\boldsymbol{y}} \in \mathbb{R}^{1 \times D_y}$. The dimensions of the parameters are $\boldsymbol{W_1} \in \mathbb{R}^{D_x \times D_h}, \boldsymbol{W_2} \in \mathbb{R}^{D_h \times D_y}, \boldsymbol{b_1} \in \mathbb{R}^{1 \times D_h}, \boldsymbol{b_2} \in \mathbb{R}^{1 \times D_y}$.

(c) Implement the forward and backward passes for a neural network with one sigmoid hidden layer. Fill in your implementation in `neural.py`. Sanity check your implementation with `python neural.py`.

(d) Use the neural network to implement a bigram language model in `neural-lm.py`. Use glove word embeddings to represent the vocabulary (`~/advanced_nlp/assignment2/data/lm/vocab.embeddings.glove.txt`). Implement the `lm_wrapper` function, that is used by `sgd` to sample the gradient, and the `eval_neural_lm` function that is used for model evaluation. Add the results to your written solution. Include in your submission the trained parameters.