עיבוד שפה טבעית | תרגיל 3

מגישים:
תומר קטיעי
אייבי כץ
אחמד חלאילה
July 22, 2024

חלק 1

חלק זה הוא בעברית, סליחה אבל הסתבכנו עם התרגום שלו בליך, שאר החלקים יהיו באנגלית.

סעיף a

ראשית נעביר את ה-Cross Entropy Loss לצורה שיותר נוח לגזור

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

$$= -\sum_i y_i \cdot \log\left(\exp(\theta_i)\right) + \log\left(\sum_j \exp(\theta_j)\right)\sum_i y_i = -\sum_i y_i\theta_i + \log\left(\sum_j \exp(\theta_j)\right)$$

כעת נגזור לפי θ_k לכל k:

$$\frac{\partial CE(y,\hat{y})}{\partial \theta_k} = -y_k + \frac{\exp(\theta_k)}{\sum_j \exp(\theta_j)} = -y_k + softmax(\theta)_k$$

כלומר הגרדיינט הוא

$$\frac{\partial CE(y,\hat{y})}{\partial \theta} = -y + softmax(\theta) = \hat{y} - y$$

סעיף b

ניעזר בכלל השרשרת:

$$\frac{\partial J}{\partial x} = \frac{\partial CE(\hat{y}-y)}{\partial x} = \frac{\partial CE(\hat{y}-y)}{\partial (hW_2+b_2)} \cdot \frac{\partial (hW_2+b_2)}{\partial h} \cdot \frac{\partial h}{\partial (xW_1+b_1)} \cdot \frac{\partial (xW_1+b_1)}{\partial x}$$

$$= (\hat{y}-y)\cdot W_2^T \cdot h(1-h) \cdot W_1^T$$

סעיף c

הקוד מומש ב-q1c_neural.py.

סעיף d

לאחר מימוש הקוד ב-q1d_neural_lm.py הרצנו SGD ל-40000 איטרציות וקיבלנו dev-perplexity = 113.313.

section 2

part a

The advantage of a character-based model is that it has far fewer parameters to train compared to a word-based model like skipgram. This is because there are significantly fewer characters (e.g., if working with ASCII characters) than words. Another advantage is the ability to read and generate words that are not in the predefined vocabulary, such as the ability to read misspellings and work with new names. On the other hand, an advantage of a word-based model is that there is no risk of misspellings or invented words when generating text (since the valid words are predefined). Another advantage is the ability to base the model on the semantic proximity of words, similar to how humans think about language: humans do not rely on the relationship between letters when they speak and read, but on the relationship between the words themselves.

part b

We have completed the code in the attached .ipynb file.

section 3

part a

$$2^{-\frac{1}{M}\sum_{i=1}^{M}\log_2 p(s_i|s_1,\ldots,s_{i-1})} = 2^{-\frac{1}{M}\log_2\left(\prod_{i=1}^{M} p(s_i|s_1,\ldots,s_{i-1})\right)} = \left(2^{\log_2\left(\prod_{i=1}^{M} p(s_i|s_1,\ldots,s_{i-1})\right)}\right)^{-\frac{1}{M}}$$

$$= \left(\prod_{i=1}^{M} p(s_i \mid s_1,\ldots,s_{i-1})\right)^{-\frac{1}{M}} = \left(e^{\ln\left(\prod_{i=1}^{M} p(s_i|s_1,\ldots,s_{i-1})\right)}\right)^{-\frac{1}{M}} = e^{-\frac{1}{M}\ln\left(\prod_{i=1}^{M} p(s_i|s_1,\ldots,s_{i-1})\right)} = e^{-\frac{1}{M}\sum_{i=1}^{M}\ln p(s_i|s_1,\ldots,s_{i-1})}$$

part b

For the bi-gram LM model we got 111.72 perplexity for the Shakespeare data, and 82.90 for the Wikipedia data.
As for the model from the previous section, we got 7.15 perplexity for the Shakespeare data, and 18.35 for the Wikipedia data.

part c

In the results we noticed various differences.
Firstly the difference between Shakespeare data and Wikipedia data perplexity in the Bi-Gram model is due to the model being trained on

more 'regular' text (in a brief look it seems like some newspaper text) which is closer to what Wikipedia contains, in contrast to Shakespeare texts carrying a unique style of writing that doesn't use every day words like Wikipedia.
We can also notice a symmetric phenomenon in the character-based model from the previous section: it is trained on Shakespeare texts and thus performs better on them in comparison to Wikipedia.
Another thing we can notice is the difference in the magnitude of the perplexities. This is due to the second model being character-based, offering less options for the next token and resulting in a smaller perplexity, as opposed to the first being word-based. There are way more words than characters and so the perplexity is way higher.

section 5

part 1

The authors of the article made the T5 (short for Text-to-Text Transfer Transformer) model publicly available. The dataset used to benchmark sentiment analysis is SST-2. In order to evaluate success in this task we measure accuracy (percentage of correct predictions) of the trained model over a test set.

part 2

We used this model: "michelecafagna26/t5-base-finetuned-sst2-sentiment" from hugging-face.
Then we tested the 4 given sentences and got these results:
1. "This movie is awesome" - positive
2. "I didn't like the movie so much" - negative
3. "I'm not sure what I think about this movie." - negative
4. "Did you like the movie?" - positive

part 3

loading the SST2 dataset and evaluating the accuracy of the T5 model yielded a 94.95% accuracy. This indeed matches the results shown on Papers with Code, which are around 95%.

part 4

The data in the SST2 dataset is quite balanced. We measured (in code) that about 51% of the validation set is labeled positive. That is very important when evaluating the model's performace, because otherwise a model biased positively/negatively might mistakenly look better (in terms of accuracy on the unbalanced dataset) than a well-balanced model although the real-life distribution of positive-negative reviews is balanced. The bottom line is that the dataset distribution should represent the real-life distribution and apparently SST2 achieves that.

part 5

Simply testing the accuracy can generally work, but there are other metrics that a human can think of that help evaluating a model's quality. One non-trivial capability that one would like to test for is a model's ability to detect **sarcasm** and label a review negative although it might contain many 'positive' words. Another example is to check for a model's response to references to other movies that are considered good/bad, for instance: "This movie reminds me of Titanic" should be classified as a positive review.

part 6

First and foremost, we'd advise them to use a sentiment analysis model for telling whether each summary is positive/negative. We can think of 3 main approaches that can come in handy:

1. Taking an existing pre-trained model and fine-tuning it specifically for family-doctor summaries sentiment analysis.
2. Taking a pre-trained model on sentiment analysis which was trained on some general sentiment analysis task (for example movies) and then fine-tuning it for family-doctor summaries sentiment analysis.
3. Training a model from zero on sentiment analysis.

Each task requires a different amount of labeled family-doctor summaries data to train on, and they should take into account how many labels they can afford to create (manually).

Approach 1 seems the best as transfer learning is very recommended for sentiment analysis and usually drug companies have enough man-power for labeling summaries. Approach 2 might require less labels but can possibly be biased towards keywords that are significant in movie reviews but not in doctor visit summaries. Approach 3 seems the worst because it might require expert knowledge on sentiment-analysis and way more data than a transfer learning method.