David Jose Fernandez

# NLP HW #3

1a) $P(w_i | w_{i-1}) = \dfrac{C(w_i, w_{i-1})}{C(w_{i-1})}$

$P(\text{store} | \text{computer}) = \dfrac{4}{10} = \dfrac{2}{5}$

$P(\text{monitor} | \text{computer}) = \dfrac{4}{10} = \dfrac{2}{5}$

Both words have the same bigram probability so they are both equally as likely to happen.

1b) Kneser-Ney smoothing $(d = .5)$

$$P(w_i | w_{i-1}) = \frac{\max(C(w_{i-1}, w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1}) \frac{|v : C(v, w_i) > 0|}{\sum_{w_i} |v : C(v, w') > 0|}$$

$$P(\text{store} | \text{computer}) = \frac{\max(3.5, 0)}{10} + .15 \left( \frac{3}{2+2+2+3} \right) = .4$$

$$\lambda(\text{computer}) = \frac{.5}{10} * 3 = .15$$

$$P(\text{monitor} | \text{computer}) = \frac{\max(3.5, 0)}{10} + .15 \left( \frac{2}{2+2+2+3} \right) = .3833$$

In this case, $P(\text{store} | \text{computer}) > P(\text{monitor} | \text{computer})$ so "store" would be chosen to complete the sentence. This makes sense since Kneser-Ney smoothing favors words that have higher likelihoods of being combined w/ various words.

1c) Kneser-Ney smoothing ($d=.1$)

$$P(\text{store}|\text{computer}) = \frac{\max(3.9,0)}{10} + .03\left(\frac{3}{9}\right) = .4$$

$$\lambda(\text{computer}) = \frac{.1}{10} * 3 = .03$$

$$P(\text{monitor}|\text{computer}) = \frac{\max(3.9,0)}{10} + .03\left(\frac{2}{9}\right) = .3966$$

"store" is still favored over "monitor" but the difference between probabilities is now smaller.

2a) $tf = \log_{10}(\text{count}(t,d)+1)$

$idf = \log_{10}\left(\frac{N}{df}\right)$

*calculations done in excel sheet.

$tf\text{-}idf = tf * idf$

| | Doc 1 | Doc 2 | Doc 3 |
|---|---|---|---|
| car | .3211 | .1551 | .3101 |
| auto | 0 | .4610 | .4447 |
| insurance | .3148 | .6686 | 0 |
| best | .1140 | 0 | .1216 |
| | $\vec{d_1}$ | $\vec{d_2}$ | $\vec{d_3}$ |

2b)

$\text{cosine}(\vec{d_1}, \vec{d_2}) = 0.6786$

$\text{cosine}(\vec{d_1}, \vec{d_3}) = 0.4401$

$\text{cosine}(\vec{d_2}, \vec{d_3}) = 0.5509$

3a) Simplify $CE(y,\hat{y})$ for skipgram model

$$CE(y,\hat{y}) = -\sum_i y_i \log(\hat{y}_i)$$

$$P(O=w_o \mid I=w_I) = \frac{\exp(u_{w_o}^T \cdot v_{w_I})}{\sum_w \exp(u_w^T \cdot v_{w_I})}$$

$$\boxed{CE(y,\hat{y}) = -\log(P(O=w_o \mid I=w_I))}$$

3b) Want to find $\frac{\partial}{\partial v_{w_I}}\left[-\log(P(O=w_o \mid I=w_I))\right]$

First expand $P(O=w_o \mid I=w_I)$:

$$\frac{\partial}{\partial v_{w_I}}\left[-\log\left(\frac{\exp(u_{w_o}^T \cdot v_{w_I})}{\sum_{w \in vocab} \exp(u_w^T \cdot v_{w_I})}\right)\right] \quad ①$$

$$\frac{\partial}{\partial v_{w_I}}\left[\log\left(\sum_w \exp(u_w^T \cdot v_{w_I})\right) - u_{w_o}^T v_{w_I}\right] \quad ②$$

Now take the derivative w/ respect to $v_{w_I}$ of equation 2:

$$\frac{\partial}{\partial v_{w_I}}[②] = \frac{\sum u_w^T \exp(u_w^T v_{w_I})}{\sum \exp(u_w^T v_{w_I})} - u_{w_o}^T$$

$$= \sum_{w \in vocab} u_w^T * \frac{\exp(u_w^T v_{w_I})}{\sum \exp(u_w^T v_{w_I})} - u_{w_o}^T$$

$$= \sum_w u_w^T \hat{y}_w - u_{w_o}^T$$

$$= U\hat{y} - u_{w_o}^T$$

$$= \boxed{U\hat{y} - Uy}$$

3c) Find derivative w/ respect to $w_o$ of expression in part a for $w_o = O$ + $w_o \neq O$.

From previous parts, expanding expression of a results in:

$$\log\left(\sum_w \exp(u_w^T v_{w_I})\right) - u_{w_o}^T v_{w_I} \quad ①$$

Compute $\frac{\partial}{\partial w_o}[①]$ for $w_o = O$:

$$\frac{\partial}{\partial w_o}[①] = \frac{v_{w_I} * \exp(u_{w_o}^T v_{w_I})}{\sum_w \exp(u_w^T v_{w_I})} - v_{w_I}$$

$$= v_{w_I} \hat{y}_{w_o} - v_{w_I}$$

$$= \boxed{v_{w_I}(\hat{y}_{w_o} - 1)}$$

Compute $\frac{\partial}{\partial w_o}[①]$ for $w_o \neq O$:

$$\frac{\partial}{\partial w_o}[①] = \frac{v_{w_I} * \exp(u_{w_o}^T v_{w_I})}{\sum_w \exp(u_w^T v_{w_I})}$$

$$= \boxed{v_{w_I} \hat{y}_{w_o}}$$

3d) the idea behind negative sampling is to use a loss function which is less computationally expensive than the softmax function which has a summation over the entire vocabulary in the denominator. Negative sampling works similar to binary classification in which the negative samples are used to reduce the probability of positive samples. the value of $k$ in negative sampling is used to determine how many negative samples to use. the loss function for negative sampling is provided below:

$$\log \sigma ({v'}_{w_O}^{T} v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[ \log \sigma (-{v'}_{w_i}^{T} v_{w_I}) \right]$$

Credit: Mikolov et. al. 2013

Question 4 Written Responses

a. Done. Submitted to gradescope.
b. The reason behind the consistent behavior of the first letter in the char ngram model can be attributed to the fact that the first char in the training corpus is F and since we read the entire corpus as one line, F is the only character associated with the starting context (~*n). The same thing happens for the word level ngram model, since the first word in the training corpus is “In” the model only associates that one word with the starting context (~*n) and thus “In” always has the highest probability when starting a random text.
c. Word vs Char on shakespeare_sonnets (both with N=2, K=.25, no interpolation)
    i. Word Model perplexity for shakespeare_sonnexts: 60550.778575062905
    ii. Char Model perplexity for shakespeare_sonnexts: 7.9664707724881545
    iii. From the perplexity values provided above it is evident that when evaluating the performance of the model intrinsically char level models perform better. This being said, when looking at the text generated by each model the extrinsic performance of the word model was better since it produced somewhat coherent text, in the case of the char model the text generated seemed like pure gibberish.
d. Best Values of perplexity
    i. Char Ngram (training: Shakespeare_inputs, dev: Shakespear_sonnest): 5.443296 using N=4, K=.05, and not interpolation
    ii. Word Ngram (training: Shakespeare_inputs, dev: Shakespear_sonnest): 3854.7768 using n=3, K=1.25, and lambda values that are determined by the frequency of the context.
    iii. Word Ngram (training: train_e, dev: val_e): 1496.6144 using N=2, k=.03125, and lambda values of 1 /( N + 1) for all lambdas.
e. Comparing Char-RNN model and Char Ngram Model
    i. Best Char Ngram perplexity on shakespeare_sonnets: 5.443296 using N=4, K=.05, and not interpolation
    ii. Best RNN Ngram perplexity on shakespeare_sonnets: 6.90747 using hidden_size=3, epochs=2000, lr=.005, n_layers=3
    iii. Sample Generated by Char-RNN model:
    The like the bate will have thew
    As a come her have the love the procesters in the saint be my streech.
    HARTONE:
    A say, my shall but a bent of the part the comes,
    What con the prisens our shall thy horders the wand of the done.
    QUEE HERNY:
    O courter
    iv. In terms of intrinsic performance the char ngram model had a better complexity but without a doubt when you compare the results from both models the text generated by the Char-RNN has recognizable words and seems to have some structure to it, unlike the results of the char ngram model.