1: (a) Raw Bigram probability:

$$P(\text{store} \mid \text{computer}) = \frac{\text{count}(\text{computer, store})}{\Sigma\ \text{count}(\text{computer})}$$

$$= \frac{4}{10} = 0.4$$

$$P(\text{monitor} \mid \text{computer}) = \frac{\text{count}(\text{computer, monitor})}{\Sigma\ \text{count}(\text{computer})}$$

$$= \frac{4}{10} = 0.4$$

Both the words are equally likely to occur.

(b): Kneser - Ney Smoothening

$d = 0.5$

$$P(w_i, w_{i-1}) = \frac{\max(C(w_{i-1}, w_i) - d, 0)}{C(w_{i-1})} +$$

$$\lambda(w_{i-1}) \frac{|v: C(v, w_i) > 0|}{\Sigma\ |v: C(v, w') > 0|}$$

$$\lambda(w_{i-1}) = \frac{d}{\sum c(w_{i-1})} * |w: C(w_{i-1} w) > 0|$$

P(store , computer) ?

discounted bigram probability { store | computer }

$$= \frac{C(\text{computer store}) - d}{C(\text{computer})}$$

$$= \frac{4 - 0.5}{10} = \frac{3.5}{10} = 0.35 \quad \text{— (1)}$$

$$\lambda(w_{i-1}) = \lambda(\text{computer})$$

$$= \frac{0.5}{10} * 3$$

$$= \frac{1}{10} \quad \text{— (2)}$$

$$\frac{|v: C(v, w_i) > 0|}{\sum |v: C(v, w) > 0|} = \frac{\#\text{ of diff string type preceding final word}}{\text{length of Bigram table for monitor}}$$

computer monitor = 4

monitor monitor = 1

$$= \frac{3}{9}$$

computer store = 4 — (3)

keyboard store = 1

monitor store = 1

combining equation ①, ②, ③

$$P(\text{store}, \text{computer}) = 0.35 + \frac{1}{10} \times \frac{1}{2}$$

$$= 0.35 + 0.05$$

$$= 0.4$$

PC monitor, computer)

discounted Bigram Prob (monitor, computer)

$$= \frac{4 - 0.5}{10} = \frac{3.5}{10} = 0.35 \quad -①$$

$$\lambda(w_{i-1}) = \lambda(\text{computer})$$

$$= \frac{0.5}{10} \times 3 \quad -②$$

$$\frac{|v: C(v, w_i) > 0|}{\sum |v: C(v, w) > 0|} = \frac{\#\text{ of diff string type preceding final word}}{\text{length of Bigram table for store}}$$

$$= \frac{2}{9} \quad -③$$

combining ①, ②, ③

$$P(\text{monitor}, \text{computer}) = 0.35 + \frac{0.5}{10} \times \frac{3}{3} \times \frac{2}{3}$$

$$= 0.3833$$

we can see that:

$$P(\text{store}, \text{computer}) > P(\text{monitor}, \text{computer})$$

Word computer store is more likely than computer monitor.

The result is changed because we now also considered absolute discounting and we redistribute our resulting probability using $P_{\text{continuation}}$.

(E): $d = 0.1$

$P(\text{store}, \text{computer}) = ?$

$$P(w_i, w_{i-1}) = \frac{\max(C(w_{i-1}, w_i) - d, 0)}{C(w_{i-1})} +$$

$$\lambda(w_{i-1}) \frac{|v: C(v, w_i) > 0|}{\sum |v: C(v, w') > 0|}$$

$$\lambda(w_{i-1}) = \frac{d}{\sum C(w_{i-1})} \times |w: C(w_{i-1} w) > 0|$$

$P(\text{store, computer}) = \frac{4-0.1}{10} +$

$\frac{0.1}{10} * 3 \quad \times \quad \frac{3}{9}$

$= \frac{3.9}{10} + \frac{0.1}{10}$

$= 0.4$

$P(\text{monitor, computer}) = \frac{4-0.1}{10} +$

$\frac{0.1}{10} \times 3 \quad \times \quad \frac{2}{9}$

$= 0.39 + 0.03 + 2/9$

$= 0.3966$

$P(\text{Store, computer}) > P(\text{monitor, computer})$

"Computer store" is more likely than "computer monitor" because we build the smoothening on the basis that words that appear more content in past are likely to appear in new contexts as well, and store has appear in more content in the past.

(2):

term- Doc.	Doc1	Doc2	Doc3
car	27	4	24
insurance	3	18	0
auto	0	33	29
best	14	0	17

Term frequency = $\log_{10}(\text{count}(t,d) + 1)$

Inverse Doc. freq = $\log_{10}(N / df_i)$

car	12
insurance	6
auto	10
best	16

Doc 1

	TI	IDf	TI × IDf
car	1.447	0.222	0.321
insurance	0.602	0.523	0.315
Auto	0	0.301	0
Best	1.176	0.097	0.114

Doc 2

	TI	IDF	TI × IDF
Car	0.699	0.222	0.155
insurance	1.278	0.523	0.668
Auto	1.532	0.301	0.461
Best	0	0.097	0

Doc 3

	TI	IDF	TI × IDF
Car	1.398	0.222	0.310
insurance	0	0.523	0
Auto	1.477	0.301	0.445
Best	1.255	0.097	0.122

(b): Cosine Similarities

$$\cos(a,b) = \frac{\sum (a_i * b_i)}{\sqrt{(\sum (a_i^2)} * \sqrt{\sum (b_i^2)}}$$

$$\cos(doc1, doc2) =$$

$$\frac{0.321 * 0.155 + 0.315 \times 0.668 + 0 + 0}{0.464 * 0.827}$$

$$\frac{0.260}{0.464 * 0.827} = 0.677$$

$$\cos(doc2, doc3) =$$

$$\frac{0.155 \times 0.310 + 0.461 \times 0.445}{0.464 * 0.827}$$

$$= 0.551$$

$$\cos(doc1, doc3) =$$

$$= \frac{0.321 * 0.310 + 0.114 * 0.122}{0.464 * 0.827}$$

$$= 0.440$$

③ $P(O|I)$ - Given an inside word w_I, what is the probability that and outside word w_O lies in the context window of w_I.

$$CE(p,q) = -\sum_i p_i \log(q_i)$$

where $p_i \Rightarrow$ True Probability distribution with vector [1 being correct word in the context & rest of the words as 0]

q_i = our prediction

Lets suppose

$$p = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

then

$$\begin{aligned} CE(p,q) &= -\sum p \log q \\ &= -\log q_o - \sum_{w \neq o} 0 \cdot \log q_w \\ &= -\log q_o \\ &= -\log P(O = w_o | I = w_I) \end{aligned}$$

↳ Distribution we want to learn

(b):

$$CE(p,q) = -\log p(O = w_o \mid I = w_I)$$

$$= -\log \frac{\exp(v_{w_o}^T \cdot v_{w_I})}{\sum \exp(u_w^T \cdot v_{w_I})}$$

$$\log \frac{a}{b} = \log a - \log b$$

$$= -\log \exp(v_{w_o}^T \cdot v_{w_I}) + \log \sum \exp(u_w^T \cdot v_{w_I})$$

$$= -v_{w_o}^T \cdot v_{w_I} + \log \sum \exp(u_w^T \cdot v_{w_I})$$

$$\frac{\partial (E(p,q)}{\partial v_{w_I}} = -u_{w_o}^T + \frac{\sum u_w^T \cdot \exp(u_w^T \cdot v_{w_I})}{\sum \exp(u_w^T \cdot v_{w_I})}$$

$$= -u + \frac{\sum u_w \exp(u_w^T \cdot v_{w_I})}{\sum \exp(u_w^T \cdot v_{w_I})}$$

$$= -u_o + \sum_{i=1}^{V} p(O = w_o \mid I = w_i) \cdot u_i$$

$\hat{y}$ = predicted distribution

$$\frac{\partial CE(p,q)}{\partial v_{w_I}} = -u_o + \sum_{i=1}^{V} \hat{y}_i \cdot u_i$$

©: $CE(p,q) = -\log P(O = w_o \mid I = w_i)$

$$\frac{\partial}{\partial v_w} CE(p,q) = \frac{\partial}{\partial v_w}\left[-\log \frac{\exp(v_w^T \cdot v_{w_I})}{\sum \exp(v_w^T \cdot v_{w_I})}\right]$$

$$= \frac{\partial}{\partial v_w}\left[-\log \exp(v_w^T \cdot v_{w_I}) + \log \sum \exp(v_w^T \cdot v_{w_I})\right]$$

$$= \frac{\partial}{\partial v_w}\left[-v_w^T \cdot v_{w_I} + \log \sum \exp(v_w^T \cdot v_{w_I})\right]$$

$$= \underbrace{\frac{\partial}{\partial v_w}\left[-v_w^T \cdot v_{w_I}\right]}_{\text{I}} + \underbrace{\frac{\partial}{\partial v_w}\left[\log \sum \exp(v_w^T \cdot v_{w_I})\right]}_{\text{II}}$$

I: $$\frac{\partial}{\partial v_w}\left[-v_w^T \cdot v_{w_I}\right] = \begin{cases} -v_{w_I}, & \text{if } w = O \\ 0, & \text{otherwise} \end{cases}$$

II: $$\frac{\partial}{\partial v_w}\left[\log \sum \exp(v_w^T \cdot v_{w_I})\right]$$

$$= \frac{\frac{\partial}{\partial v_w}(v_w^T \cdot v_{wI}) * \sum \exp(v_w^T \cdot v_{wI})}{\sum \exp(v_w^T \cdot v_{wI})}$$

$$= \hat{y} \cdot v_{wI} \quad \left[\because \hat{y} = \frac{\sum \exp(v_w^T \cdot v_{wI})}{\sum \exp v_w^T \cdot v_{wI}} \right.$$

Combining I and II

$$\frac{\partial}{\partial v_w} = \begin{cases} \hat{y}_0 \cdot v_{wI} - v_{wI}, & w = 0 \\ \hat{y}_w \cdot v_{wI}, & \text{otherwise} \end{cases}$$

(d): We use negative sampling as an alternative to Naive Softmax classifier. The goal of negative sampling is:

(a): Maximize the similarity of our target word, context word pair drawn from positive sample

(b) Minimize the similarity of target word, context word pair drawn from negative example.

why K

if we draw K negative samples from the vocabulary, the advantage is that whereas in Naive Softmax weight update happen for the whole words and that can be computationally expensive, but in negative sampling it is O(K) where K is typically 10 - 20.

The loss function for negative sampling is given as:

$$L(v_{w_I}, v_{w_O}, K) =$$

$$-\log \sigma(u_{w_O}^T \cdot v_{w_I}) - \sum_{R=1}^{K} \log \sigma(-u_R^T v_{w_I})$$

Q4)

Part-1: Generations

Q: what do you think? Is it as good as 1000 monkeys working at 100 typewriters?

⇒ I think the model predicts words that doesn't exist in the vocabulary, given that it is dependent on next most likely character.

Q 4.b: They all start with f in char-level Ngram and for all word level N gram it starts with In. Did you get such results? Explain what is going on?

Ans. Yes, I got similar results.

⇒ we can think of this in terms of n.

if $n=0$.

Then the letters are context independent and anything can be generated for starting char. The same is true for word-level N gram model. The first word being In, $n=0$, will always give us In as first word.

if $n = 1$.

2> Then f is always chosen as the first char because it is the opening letter of the input file, similarly In is chosen always as In is the first word in the word level input file.

if $n = 2$

2> the character depends on the f, meaning only char that have high probability of following a f at the beginning of the sentence can appear. for word level N-gram, the word with high probability following "In" word can appear.

And finally with $n > 3$ we always obtain a word as the opening in both char-level and word level.

Q4.c :

Shakespeare_sonnet Article

Perplexity (word)	Perplexity (char)	Order (n)
4658.7	23.45	0
12061	14.26	1
4338	10.28	2
5130	7.961	3

We can observe from the above table that perplexity improves with higher order (n = 1, 2, 3) as than we have more context to look up, which improves the probability of the predicted word. This is consistent with our intuition as well.

However, for word level Ngram we can observe that perplexity performance reduces from n = 2 to n = 3, and that is perhaps because for seeing 3 previous words, it is less predictable to guess the frequency distribution in test file than the train file. Also in word level, the next word prediction is more affected by immediate previous word.

PART 2: PERPLEXITY, SMOOTHING AND INTERPOLATION.

Q4-d. In your report experiment with few different lambdas and values of k and discuss their effects. for both char level and word level Ngram.

Ans: Considering higher order n gives good result (from 4-c), we fix $n=2$ and experiment with k, λ_s.

CHAR-LEVEL (Shakespeare_sonnet - txt)

→

λ_1	λ_2	λ_3	k	Perplexity
0.9	0.05	0.05	0	7
			1	7.9
			2	8
0.1	0.2	0.7	0	14
			1	53
			2	234
0.7	0.2	0.1	0	8
			1	13
			2	33
0.5	0.4	0.1	0	8.9
			1	15
			2	46
0.3	0.4	0.3	0	10
			1	21
			2	78

WORD - LEVEL

(val_e.txt)

λ_1	λ_2	λ_3	k	Perplexity
0.1	0.2	0.7	0.1	3201
			1	5744
			2	12855
0.7	0.2	0.1	0.1	1750
			1	2304
			2	2100
0.5	0.4	0.1	0.1	1651
			1	2472
			2	2707
0.1	0.1	0.8	0.1	3850
			1	6559
			2	7516

Based on the above 2 tables, I observed that for char level n-gram the perplexity performance improves if we give more weight towards Lambda-1, and that is reasonable as in char level n gram the higher the context length we have the better the prediction in words.

In Word-level n-gram the observations were opposite. Higher weights towards Lambda - n gives better performance in perplexity because in word level n-gram the next word prediction is dependent on the immediate previous word.

for K, I observed Lower value of K favours better perplexity.
finally the best hyperparameters that I was able to get were.

Char-level (Shakespear_sonnet.txt)		Word level (val-e.txt)	
K =	0.001	K =	0.001
n =	4	n =	2
λ_1 =	0.9	λ_1 =	0.1
λ_2 =	0.05	λ_2 =	0.2
λ_3 =	0.02	λ_3 =	0.7
λ_4 =	0.02	perplexity =	975
λ_5 =	0.01		
perplexity =	5.2		

Q4.e: In the results of RNN and char level n-gram model we see that Perplexity Char level n-gram is 5.2 (with hyperparameter tuning) whereas in rnn the perplexity comes around 2.2 which is better than char level n gram, because in rnn the model accounts dynamically previous n-char, so we no need to manually look for best set of hyperparameters in rnn, and that is why Deep learning models are efficient.