1 SVD and PCA [35 Points]

Problem A [3 points]:

1 SVD and PCA [35 Points]

Problem A [3 points]: Let X be a $N \times N$ matrix. For the singular value decomposition (SVD) $X = U\Sigma V^T$, show that the columns of U are the principal components of X. What relationship exists between the singular values of X and the eigenvalues of XX^T?

From the slides, we know PCA gives the following

$XX^T = U\Lambda U^T \leftarrow$ PCA

From SVD decomposition, we have:

$\Sigma^T = \Sigma \Leftarrow$ diag matrix

$$X = U\Sigma V^T$$

$$X \cdot X^T = U\Sigma V^T X^T = U\Sigma V^T (U\Sigma V^T)^T$$

$$= U\Sigma V^T V \Sigma U^T$$

Considering V is an orthogonal matrix, $V^T V = 1$

$$\therefore X \cdot X^T = U\Sigma^2 U^T .$$

From the previous equation, we have the matrix U solves PCA with $\Lambda = \Sigma^2$.

Thus, each column of U is an eigenvector of XX^T, and we know XX^T is a principal component of X.

$\therefore$ The columns of U in the SVD equation are the principal components of X.

The relationship between the singular values of X and the eigenvalues of XX^T is that the latter are squared of the former

Problem B [4 points]:

The mathematical justification is that they are squared of the singular values of X and thus, non negative. A more intuitive explanation is that eigenvalues represent the total amount of variance that can be explained by a given principal component. In theory they can be positive or negative, but as in practice they represent the variance they can not be negative, as variance is always positive.

Problem C [5 points]:

Problem C [5 points]: In calculating the Frobenius and trace matrix norms, we claimed that the trace is invariant under cyclic permutations (i.e., $\mathrm{Tr}(ABC) = \mathrm{Tr}(BCA) = \mathrm{Tr}(CAB)$). Prove that this holds for any number of square matrices.

Hint: First prove that the identity holds for two matrices and then generalize. Recall that $\mathrm{Tr}(AB) = \sum_{i=1}^{N}(AB)_{ii}$. Can you find a way to expand $(AB)_{ii}$ in terms of another sum?

From the hint:

$$\mathrm{Tr}(AB) = \sum_{i=1}^{N}(AB)_{ii} = \sum_{i=1}^{N}\sum_{j=1}^{N} A_{ij} \cdot B_{ji} \quad \ldots (1)$$

$$\mathrm{tr}(BA) = \sum_{i=1}^{N}(BA)_{ii} = \sum_{i=1}^{N}\sum_{j=1}^{N} B_{ij} \cdot A_{ji} = \sum_{i=1}^{N}\sum_{j=1}^{N} A_{ji} B_{ij}$$

$$\mathrm{tr}(BA) = \sum_{j=1}^{N}\sum_{i=1}^{N} A_{ji} \cdot B_{ij} \quad \ldots (2)$$

From (1) and (2), we have that $\mathrm{tr}(AB) = \mathrm{tr}(BA)$ because both of them give the same sum.

For the general case, we can show that this relationship holds by grouping matrices in order to have the product of just 2 matrices.

$$\therefore \mathrm{tr}(ABC) = \sum_{i=1}^{N}\sum_{j=1}^{N} A_{ij} \cdot (BC)_{ji}$$

$$\mathrm{tr}(BCA) = \sum_{j=1}^{N}\sum_{i=1}^{N} (BC)_{ji} \cdot A_{ij}$$

$\therefore \mathrm{tr}(ABC) = \mathrm{tr}(BCA)$ and similar for more matrices.

Problem D [3 points]:

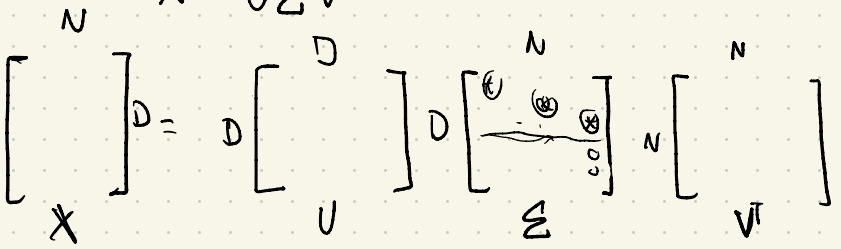
For k singular values, we need to store k columns of both U and V (which are initially of length N), so we will end with N*k*2 values (from U and V) plus k values from the matrix Σ (which has k term on the diagonal). Thus, we will end with 2*N*k + k = k*(1+2*N) values stored in the truncated SVD. Storing the truncated SVD will be more efficient than storing the whole SVD matrix when k<N^2/(2*N+1)

Dimensions & Orthogonality

Problem E [3 points]:

Problem E [3 points]: Assume that $D > N$ and that X has rank N. Show that $U\Sigma = U'\Sigma'$, where Σ' is the $N \times N$ matrix consisting of the first N rows of Σ, and U' is the $D \times N$ matrix consisting of the first N columns of U. The representation $U'\Sigma'V^T$ is called the "thin" SVD of X.

$X = U \Sigma V^T$



$N \times D$

$U' = D \times N$

$\Sigma' = N \times N$

Assume $D > N$. X has rank N

We have to show that $U\Sigma = U'\Sigma'$

$$(U\Sigma)_{ij} = \sum_{k=1}^{D} U_{ik}\,\Sigma_{kj} = \sum_{k=1}^{N} U_{ik}\cdot\Sigma_{kj} + \underbrace{\sum_{k=N+1}^{D} U_{ik}\cdot\Sigma_{kj}}_{0}$$

Since Σ only has non-zero entries in the diagonal elements until the position Σ_{nn}, the summation highlighted in blue will be zero, because for $k > N$ the element U_{ik} will be multiplied by terms Σ_{kj}, which are zero.

Thus $(U\Sigma)_{ij} = \sum_{k=1}^{N} U_{ik}\cdot\Sigma_{kj} = (U'\Sigma')_{ij}$ □

for every $i \in [1, D]$, $j \in [1, N]$

Problem F [3 points]:

We know that U' is a DxN matrix so U'^T is a NxD matrix. Thus $U'*U'^T$ is a DxD matrix, meanwhile U'^T*U' is a NxN matrix. Hence $U'*U'^T$ and U'^T*U' are never the same matrices because they are of different dimensions and as a consequence the definition of orthogonality can not hold.

Problem G [4 points]:

Problem G [4 points]: Even though U' is not orthogonal, it still has similar properties. Show that $U'^T U' = I_{N\times N}$. Is it also true that $U'U'^T = I_{D\times D}$? Why or why not? Note that the columns of U' are still orthonormal. Also note that orthonormality implies linear independence.

Since U' has orthonormal columns, we know that for $U'^T U$ we will have 1 on diagonal elements ($\langle U, U\rangle = 1 \ \forall \in \mathbb{R}^N$) and 0 in the out of diagonal elements ($\langle U_1, U_2\rangle = 0 \ \forall \ U_1, U_2 \in \mathbb{R}^n$). Thus, the result is

$I_{N\times N}$ □

for the case of $U'U'^T$, we know that U' is a $D\times N$ matrix, and that $D > N$. So, considering U' has more rows than colums, they are not all linearly independent and hence, they can not be all orthonormal. and thus, the dot product between two different rows can be different of zero

$\therefore U'.U'^T \neq I_{DD}$

Pseudoinverses

Problem H [4 points]:

Considering the properties of diagonal matrices and inverses, we know that the inverse of a diagonal matrix is a diagonal matrix with each entry equal to the reciprocal of the corresponding entry in the original matrix. Taking into account that Σ is invertible, and considering what was mentioned in lecture 10 regarding pseudoiverses, we know that Σ^+ with entries $\sigma^+ = 1/\sigma$ if $\sigma>0$ and 0 otherwise is equivalent to Σ^{-1}, which as mentioned above, has entries equal to the reciprocal of the corresponding entry in the original matrix.

$$A^+ = V\Sigma^+U^T$$

Pseudoinverse

$$\Sigma^+ = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \sigma_D \end{bmatrix} \qquad \sigma^+ = \begin{cases} 1/\sigma & \text{if } \sigma > 0 \\ 0 & \text{otherwise} \end{cases}$$

Machine Learning & Data Mining Diego Causarano

Set 5 February 2023

Problem I [4 points]:

Problem I [4 points]: Another expression for the pseudoinverse is the least squares solution $X^{+'} = (X^T X)^{-1} X^T$. Show that (again assuming Σ invertible) this is equivalent to $V\Sigma^{-1}U^T$.

$$X^{+'} = (X^T X)^{-1} X^T$$

$$X^{+'} = X^{-1}(X^T)^{-1} X^T$$

$$X^{+'} = (U\Sigma V^T)^{-1} (V\Sigma U^T)^{-1} (V\Sigma U^T)$$

$$X^{+'} = (V^T)^{-1} \Sigma^{-1} U^{-1} U^{-T} \Sigma^{-1} V^{-1} V\Sigma U^T$$

$$X^{+'} = (V^T)^{-1} \Sigma^{-1} U^{-1}$$

$$X^{+'} = V\Sigma^{-1} U^T \quad \square$$

$U = V^T$, $U^T = U^{-1}$ } Orthogonal

$V \cdot V^T = I$

$V^T V = I$

Problem J [2 points]:

Problem J [2 points]: One of the two expressions in problems H and I for calculating the pseudoinverse is highly prone to numerical errors. Which one is it, and why? Justify your answer using condition numbers.

$X^+ = V\Sigma^+ U^T$ ← part H

$X^+ = (X^TX)^{-1}X^T$ ← part I

We have that $X^TX = (U\Sigma V^T)^T(U\Sigma V^T)$

$= V\Sigma U^T U\Sigma V^T$ $\quad U^TU = I$ (-ort

$= V\Sigma^2 V^T$

∴ X^TX has singular values equal to the entries of Σ^2

Using condition number to analyze the stability

For (X^TX) (case part I) → $\frac{\sigma_{max}^2}{\sigma_{min}^2} = k_1$

For Σ (case part II) → $\frac{\sigma_{max}}{\sigma_{min}} = k_2$

According to condition number theory, a matrix is well conditioned while k is not significantly larger than 1, - at the contrary the matrix is said to be ill-conditioned.

∴ considering $k_1 \geq k_2$, computing the inverse of (X^TX) is more prone to error than calculating the inverse of Σ and thus, part I is more prone to error.

2 Matrix Factorization [30 Points]

Problem A [5 points]:

2 Matrix Factorization [30 Points]

In the setting of collaborative filtering, we derive the coefficients of the matrices $U \in \mathbb{R}^{M \times K}$ and $V \in \mathbb{R}^{N \times K}$ by minimizing the regularized square error:

$$\arg\min_{U,V} \frac{\lambda}{2}\left(\|U\|_F^2 + \|V\|_F^2\right) + \frac{1}{2}\sum_{i,j}\left(y_{ij} - u_i^T v_j\right)^2$$

where u_i^T and v_j^T are the i^{th} and j^{th} rows of U and V, respectively, and $\|\cdot\|_F$ represents the Frobenius norm. Then $Y \in \mathbb{R}^{M \times N} \approx UV^T$, and the ij-th element of Y is $y_{ij} \approx u_i^T v_j$.

Problem A [5 points]: Derive the gradients of the above regularized squared error with respect to u_i and v_j, denoted ∂_{u_i} and ∂_{v_j} respectively. We can use these to compute U and V by stochastic gradient descent using the usual update rule:

$$u_i = u_i - \eta\partial_{u_i}$$
$$v_j = v_j - \eta\partial_{v_j}$$

where η is the learning rate.

we have

$\|U\|_F^2 = \sum_i u_i^T u_i$ ← Frobenious Norm.

$\partial u_i = \frac{\lambda}{2}(2u_i) + \frac{1}{2}\sum_j (-v_j)(2(y_{ij} - u_i^T v_j))$

$\partial u_i = \lambda u_i - \sum_j v_j (y_{ij} - u_i^T v_j)^T$

and similarly for ∂v_j

$\partial v_j = \lambda v_j - \sum_i u_i (y_{ij} - u_i^T v_j)^T$

Problem B [5 points]:

Problem B [5 points]: Another method to minimize the regularized squared error is alternating least squares (ALS). ALS solves the problem by first fixing U and solving for the optimal V, then fixing this new V and solving for the optimal U. This process is repeated until convergence.

Derive closed form expressions for the optimal u_i and v_j. That is, give an expression for the u_i that minimizes the above regularized square error given fixed V, and an expression for the v_j that minimizes it given fixed U.

1) Expression for u_i given fixed V

For minimization, we must make the last derivative equal to zero.

$$f_{u_i} = \lambda u_i - \sum_j v_j (y_{ij} - u_i^T v_j)^T = 0$$

$$\lambda u_i = \sum_j v_j y_{ij}^T - \sum_j v_j v_j^T u_i = \sum_j v_j y_{ij}^T - \left(\sum_j v_j v_j^T\right) u_i$$

$$\left(\lambda I + \sum_j v_j v_j^T\right) u_i = \sum_j v_j y_{ij}^T$$

$$u_i = \left(\lambda I + \sum_j v_j v_j^T\right)^{-1} \left(\sum_j v_j y_{ij}^T\right)$$

2) Expression for v_j given fixed U

$$f_{v_j} = \lambda v_j - \sum_i u_i (y_{ij} - u_i^T v_j)^T = 0$$

$$\lambda v_j = \sum_i u_i y_{ij}^T - \sum_i u_i u_i^T v_j$$

$$\left(\lambda I + \sum_i u_i u_i^T\right) v_j = \sum_i u_i y_{ij}^T$$

$$v_j = \left(\lambda I + \sum_i u_i u_i^T\right)^{-1} \left(\sum_i u_i y_{ij}^T\right)$$

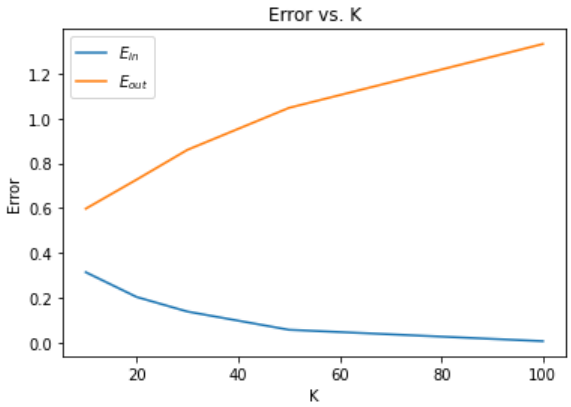
Machine Learning & Data Mining Diego Causarano

Set 5 February 2023

Problem C [10 points]:

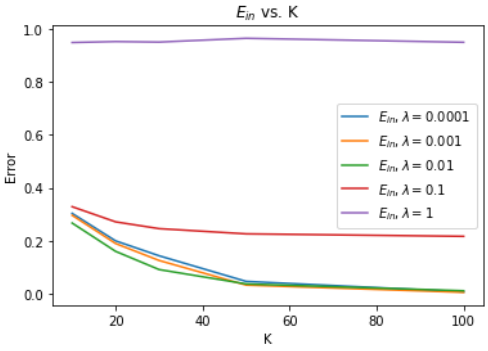
https://colab.research.google.com/drive/18PvFSwLD0ICcgxmkL3zq6wDTw3gaZzeR?usp=sharing

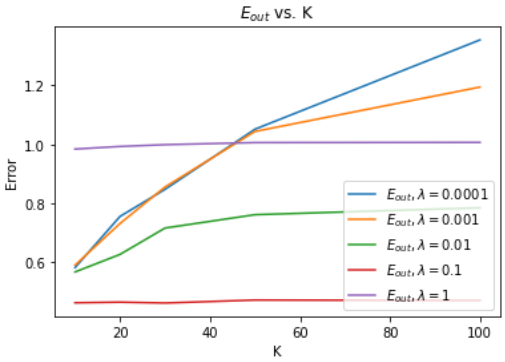
Problem D [5 points]:



As we can see in the plot, as K goes up E_{in} decreases and E_{out} increases. This can be explained considering that K determines the dimensionality of matrices U and V. When K is lower, the input data is smaller and we are summarizing the original data, making the model not to learn the singularities of the training data and with this, prevent overfitting. On the contrary, when k increases and dimensionality of matrices U and V goes up, the model tend to capture this singularities from the training data when the test error goes up, which means that the model is overfitting and it decreases its capacity of generalize for new data.

Problem E [5 points]:





The pattern is similar to part C. In general, as K goes up E_{in} tends to decrease meanwhile E_{out} tends to increase. For increasing regularization, such as for λ = 1 or 0.1, both curves E_{in} and E_{out} are flatten, but for smaller λ 's both of them follow the pattern from previous part that where λ =0, and when K increases the models start overfitting.

3 Word2Vec Principles [35 Points]

Problem A [5 points]:

Problem #3

Problem A [5 points]: If we wanted to train this model with naive gradient descent, we'd need to compute all the gradients $\nabla \log p(w_O|w_I)$ for each w_O, w_I pair. How does computing these gradients scale with W, the number of words in the vocabulary, and D, the dimension of the embedding space? To be specific, what is the time complexity of calculating $\nabla \log p(w_O|w_I)$ for a single w_O, w_I pair?

We have $\log p(w_O|w_I) = \log\left(\frac{\exp(v_{w_O}^T v_{w_I})}{\sum_{w=1}^{W} \exp(v_w^T v_{w_I})}\right)$

$$\frac{\partial}{\partial v_{w_O}}\left(\log(w_O|w_I)\right) = \frac{\partial}{\partial v_{w_O}}\left(\log(\exp(v_{w_O}^T v_{w_I})) - \log\left(\sum_{w=1}^{W} \exp(v_w^T v_{w_I})\right)\right)$$

$\log(\exp(v_{w_O}^T v_{w_I})) = v_{w_O}^T v_{w_I}$

$$= v_{w_I} - \frac{\partial}{\partial v_{w_O}}\left(\log\left[\sum_{w=1}^{W} \exp(v_w^T v_{w_I})\right]\right)$$

$$\frac{\partial}{\partial v_{w_O}}(\log(w_O|w_I)) = v_{w_I} - \frac{v_{w_I}}{\sum_{w=1}^{W} \exp(v_w^T v_{w_I})} \cdot \left(\exp(v_{w_O}^T \cdot v_{w_I})\right)$$

//

Similarly for ∂v_{w_I} we have:

$$\frac{\partial}{\partial v_{w_I}}(\log(w_O|w_I)) = v_{w_O} - \frac{1}{\sum_{w=1}^{W} \exp(v_w^T v_{w_I})} \cdot \sum_{w=1}^{W} (v_w^T v_{w_I}) \cdot v_w$$

//

The time complexity for computing gradients of w_I and w_O is dominated by the numerator term in ∂v_{w_I}. We can see that, the dot product of v_w and v_{w_I} (D), and then multiplies it by v_w (D). considering this operation is made for each w in $\{1, \ldots, W\}$ (W), the time complexity for calculating a gradient for each pair is $O(W*D)$.

Problem B [10 points]:

Problem B [10 points]: When the number of words in the vocabulary W is large, computing the regular softmax can be computationally expensive (note the normalization constant on the bottom of Eq. 2). For reference, the standard fastText pre-trained word vectors encode approximately $W \approx 218000$ words in $D = 100$ latent dimensions. One trick to get around this is to instead represent the words in a binary tree format and compute the hierarchical softmax.

When the words have all the same frequency, then any balanced binary tree will minimize the average representation length and maximize computational efficiency of the hierarchical softmax. But in practice, words occur with very different frequencies — words like "a", "the", and "in" will occur many more times than words like "representation" or "normalization".

Table 1: Words and frequencies for Problem B

Word	Occurrences
do	18
you	4
know	7
the	20
way	9
of	4
devil	5
queen	6

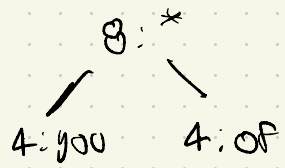
The original paper (Mikolov et al. 2013) uses a Huffman tree instead of a balanced binary tree to leverage this fact. For the 8 words and their frequencies listed in the table below, build a Huffman tree using the algorithm found here. Then, build a balanced binary tree of depth 3 to store these words. Make sure that each word is stored as a *leaf node* in the trees.

The representation length of a word is then the length of the path (the number of edges) from the root to the leaf node corresponding to the word. For each tree you constructed, compute the expected representation length (averaged over the actual frequencies of the words).

1)

frecuency	word
4	you
4	of
5	devil
6	queen
7	know
9	way
18	do
20	the

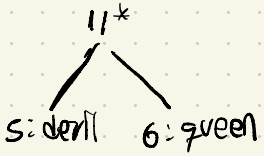
1)



2)

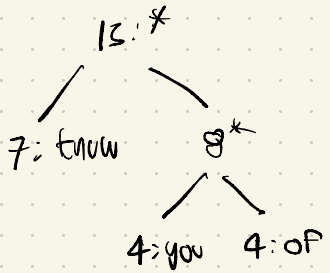
2)

5	devil
6	green
7	know
8	*
9	way
10	do
20	the



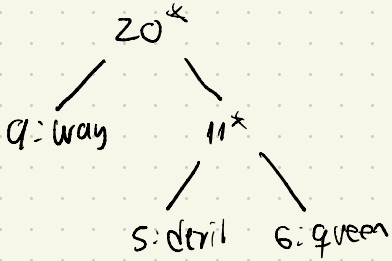
3)

7	know
8	*
9	way
11	*
10	do
20	the



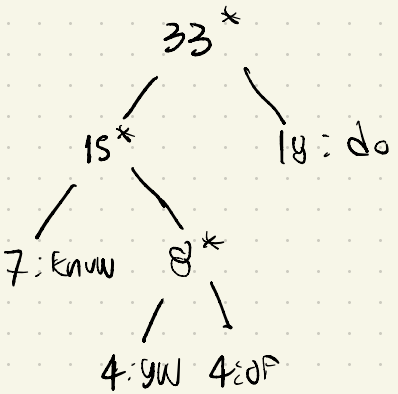
4)

9	way
11	*
15	*
10	do
20	the



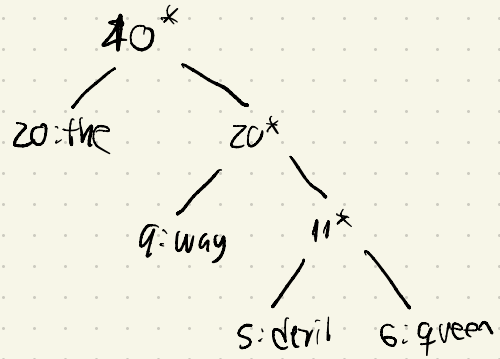
5)

15	*
10	do
20	the
20	*



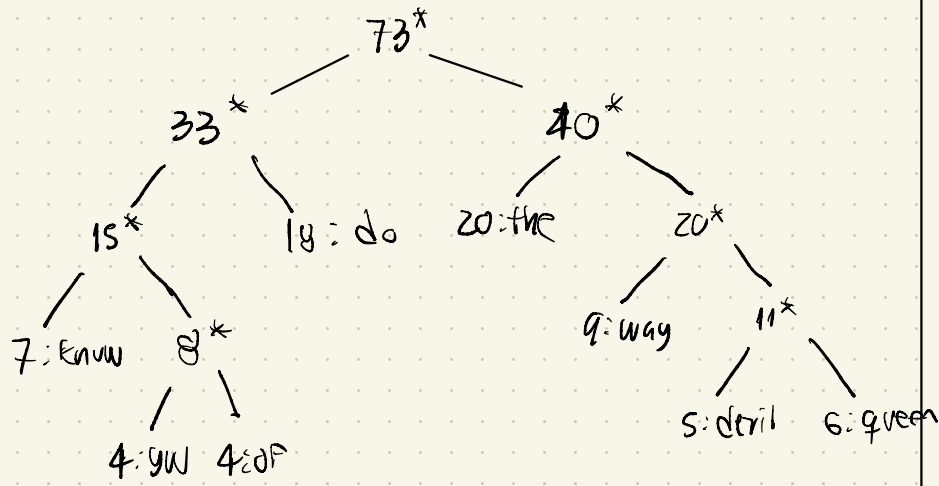
7)

20	the
20	*
33	*

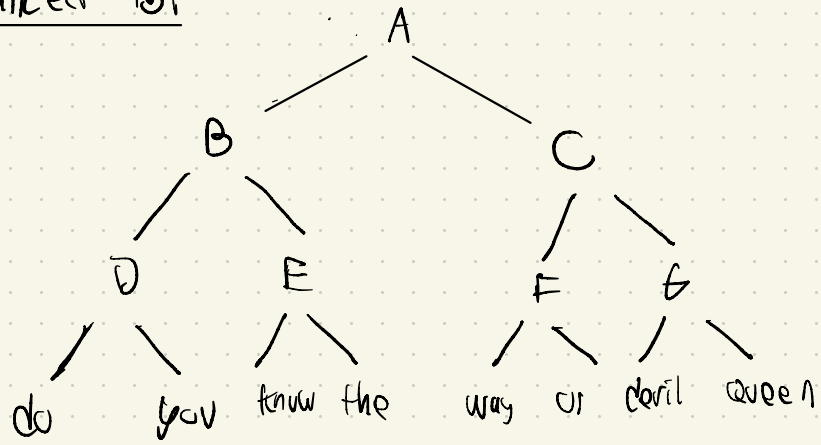


9)

33	*
40	*



Balanced BT



The expected representation lenght is 2.74 for the Huffman tree and 3 for balanced tree

Problem C [3 points]:

As we increase D, the value of the training objective will also increase, considering that $p(w_O|w_I)$ involves multiplying v^T and v, where $v \in R^D$. Thus, as D increases, it becomes more computationally expensive to make these calculations. Another aspect to considering when increasing D, is that the model will be more likely to overfit the training data, decreasing its capacity to generalize for new data.

Problem D [10 points]:

https://colab.research.google.com/drive/1gRLo3mP0kC-8vVBBYBBEweJSSWbspfbI?usp=sharing

Problem E [2 points]:

The dimension of the weigh matrix in the hidden layer is 308x10, where 308 is the number of unique words in the text file and 10 is the number of neurons in the hidden layer

Problem F [2 points]:

The dimension of the weight matrix of the output layer is 10x308

Problem G [1 points]:

```
Pair(finger, top), Similarity: 0.9980181
Pair(top, finger), Similarity: 0.9980181
Pair(foot, cold), Similarity: 0.9855466
Pair(cold, foot), Similarity: 0.9855466
Pair(shoe, off), Similarity: 0.98533446
Pair(off, shoe), Similarity: 0.98533446
Pair(eggs, ham), Similarity: 0.9851607
Pair(ham, eggs), Similarity: 0.9851607
Pair(green, ham), Similarity: 0.9828462
Pair(every, day), Similarity: 0.98120433
Pair(day, every), Similarity: 0.98120433
Pair(today, gone), Similarity: 0.9808445
Pair(gone, today), Similarity: 0.9808445
Pair(eight, nine), Similarity: 0.979169
Pair(nine, eight), Similarity: 0.979169
Pair(four, six), Similarity: 0.9791032
Pair(six, four), Similarity: 0.9791032
Pair(drink, thing), Similarity: 0.97856116
Pair(thing, drink), Similarity: 0.97856116
Pair(long, way), Similarity: 0.97736496
Pair(way, long), Similarity: 0.97736496
Pair(wire, goodbye), Similarity: 0.9767371
Pair(goodbye, wire), Similarity: 0.9767371
Pair(likes, drink), Similarity: 0.9729196
Pair(sad, glad), Similarity: 0.97162807
Pair(glad, sad), Similarity: 0.97162807
Pair(wink, pink), Similarity: 0.97104865
Pair(pink, wink), Similarity: 0.97104865
Pair(hot, sun), Similarity: 0.9701199
Pair(sun, hot), Similarity: 0.9701199
```

Problem H [2 points]:

The first thing that I noticed is that there appear many words that rhyme between them, such as sad and glad, or wink and pink. This makes sense considering the way the poem was written, where words that rhyme with others tends to be close to each other in the text. Another pattern is that many pairs appears twice in both ways ((a,b) and (b,a)), and usually they are immediately below the other. Another pattern is the appearance of words that have the same context, such as (hot, sun) or (eggs, ham).