Machine Learning & Data Mining
Set 5
20 HOURS LATE

Caltech CS/CNS/EE 155
Manuel Rodriguez
February 2023

Policies

- Due 9 PM PST, February 22nd on Gradescope.
- You are free to collaborate on all of the problems, subject to the collaboration policy stated in the syllabus.
- In this course, we will be using Google Colab for code submissions. You will need a Google account.

Submission Instructions

- Submit your report as a single .pdf file to Gradescope (entry code K3RPGE), under "Set 5 Report".
- In the report, **include any images generated by your code** along with your answers to the questions.
- Submit your code by **sharing a link in your report** to your Google Colab notebook for each problem (see naming instructions below). Make sure to set sharing permissions to at least "Anyone with the link can view". **Links that can not be run by TAs will not be counted as turned in.** Check your links in an incognito window before submitting to be sure.
- For instructions specifically pertaining to the Gradescope submission process, see https://www.gradescope.com/get_started#student-submission.

Google Colab Instructions

For each notebook, you need to save a copy to your drive.

1. Open the github preview of the notebook, and click the icon to open the colab preview.
2. On the colab preview, go to File → Save a copy in Drive.
3. Edit your file name to "lastname_firstname_originaltitle", e.g."yue_yisong_3_notebook_part1.ipynb"

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?

Solution A:
There is an equivalence between SVD and PCA. This is noted by the following calculation:

$$XX^T = (U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma V^T V\Sigma U^T = U\Sigma^2 U^T$$

In PCA, the columns of the othogonal matrix are the principal components of X, so in this case that is U. The Eigenvalues of XX^T are the diagonal entries of Σ^2. Since the singular values of X are the diagonal entries of Σ, this means that the eigenvalues of XX^T are the squares of the singular values of X.

Problem B [4 points]: Provide both an intuitive explanation and a mathematical justification for why the eigenvalues of the PCA of X (or rather XX^T) are non-negative. Such matrices are called positive semi-definite and possess many other useful properties.

Solution B: *The eigenvalues of the PCA of X are non-negative because they are the square of the singular values of X. n^2 has a range only on the non-negatives, and thus the eigenvalues of the PCA of X must be non-negative.*

Problem C [5 points]: In calculating the Frobenius and trace matrix norms, we claimed that the trace is invariant under cyclic permutations (i.e., Tr(ABC) = Tr(BCA) = 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 Tr$(AB) = \sum_{i=1}^{N}(AB)_{ii}$. Can you find a way to expand $(AB)_{ii}$ in terms of another sum?

Solution C: *For one $N \times N$ matrix, this trivially holds.*
For two $N \times N$ matrices A and B,

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

by symmetry of i and j,

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

So we have shown that for two $N \times N$ matrices, their traces are invariant under cyclic permutation. We will now show using inducting that this is true. Assume that for $\leq m$ square matrices, we know that their traces are invariant under cyclic permutation. Let $M^1, M^2, M^3, \ldots, M^{m+1}$ be any $m+1$ square matrices. Notice that we can take the product of exactly m of them and define $M^{i:i+1}$, as the matrix product of adjacent matrices M^i and M^{i+1} where if $i = m+1$, we multiply M^{m+1} by M^1. Taking the remaining matrices and $M^{i:i+1}$, by our induction hypothesis, we know that the trace of these m matrices is invariant under cyclic permutation. I.e., we know that $Tr(M^1 M^2 \cdots M^{i:i+1} M^{i+2} \cdots M^{m+1}) = Tr(M^2 \cdots M^{i:i+1} M^{i+2} \cdots M^{m+1} M^1) = ... = Tr(M^{m+1} M^1 \cdots M^{i:i+1} M^{i+2} \cdots M^m)$, and since $M^{i:i+1} = M^i M^{i+1}$, we have shown that the trace is invariant under cyclic permutation for $m+1$ matrices.
Therefore by induction, for $m \geq 1$ square matrices, the trace is invariant under cyclic permutation.

Problem D [3 points]: Outside of learning, the SVD is commonly used for data compression. Instead of storing a full $N \times N$ matrix X with SVD $X = U\Sigma V^T$, we store a truncated SVD consisting of the k largest singular values of Σ and the corresponding columns of U and V. One can prove that the SVD is the best rank-k approximation of X, though we will not do so here. Thus, this approximation can often re-create the matrix well even for low k. Compared to the N^2 values needed to store X, how many values do we need to store a truncated SVD with k singular values? For what values of k is storing the truncated SVD more efficient than storing the whole matrix?

Hint: For the diagonal matrix Σ, do we have to store every entry?

Solution D: *Since we are truncating Σ, the only values that matter in U and V are the first k columns. In other words, we would need to store $k \times N$ values in U and then store k values in Σ. Note that if we had U be a $N \times k$ matrix and Σ a $k \times k$ matrix, no information is lost. In terms of the product of Σ with V, the last two rows of V^T always are multiplied by zeros and thus do not do anything. This means we can reduce V^T to be a $k \times N$ matrix $\implies$ V is a $N \times k$ matrix. In total, we would need to store $k \cdot 2N + 1$ values. This would be more efficient to store when $k \cdot 2N + 1 < N^2$.*

Dimensions & Orthogonality

In class, we claimed that a matrix X of size $D \times N$ can be decomposed into $U\Sigma V^T$, where U and V are orthogonal and Σ is a diagonal matrix. This is a slight simplification of the truth. In fact, the singular value decomposition gives an orthogonal matrix U of size $D \times D$, an orthogonal matrix V of size $N \times N$, and a rectangular diagonal matrix Σ of size $D \times N$, where Σ only has non-zero values on entries $(\Sigma)_{ii}$, $i \in \{1, \ldots, K\}$, where K is the rank of the matrix X.

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.

Solution E:

$$U = \begin{bmatrix} u_{11} & \cdots & u_{1D} \\ \vdots & \ddots & \vdots \\ u_{D1} & \cdots & u_{DD} \end{bmatrix}, \textit{since X has rank N}, \Sigma = \begin{bmatrix} \sigma_{11} & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_{22} & 0 & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & 0 & \cdots & \vdots \\ 0 & \cdots & 0 & \sigma_{NN} & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix}$$

Notably,

$$U\Sigma = \begin{bmatrix} \sigma_{11}\mathbf{u_1} & \cdots & \sigma_{NN}\mathbf{u_N} & 0 & \cdots & 0 \end{bmatrix}$$

where $\mathbf{u_i}$ is the i^{th} column of U. The columns after the N^{th} column of U and Σ do not convey any information (since those $\mathbf{u_i}$ would be multiplied by a scalar 0), and thus we can reduce U and Σ to be their first N columns. These new matrices U' and Σ' would have dimension $D \times N$ and $N \times N$, respectively, as specified in the problem statement.

Problem F [3 points]: Show that since U' is not square, it cannot be orthogonal according to the definition given in class. Recall that a matrix A is orthogonal if $AA^T = A^TA = I$.

Solution F: *Assume U' is orthogonal. Then the dimensionality of $U'U'^T$ is $D \times D$ and U'^TU' has dimensionality $N \times N$. Therefore, $U'U'^T \neq U'^TU'$. This is a contradiction, and thus U' cannot be orthogonal.*

Problem G [4 points]: Even though U' is not orthogonal, it still has similar properties. Show that $U'^TU' = I_{N \times N}$. 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.

Solution G: *Since U is orthogonal, we know that for any column u_i in U, $u_i^Tu_i = 1$ and $u_i^Tu_j = 0$ when $i \neq j$. Orthogonality also implies that this holds for the rows of U. Since the columns in U' are taken from the columns in U, they still hold these properties. Therefore, $U'^TU' = I_{N \times N}$. However, when doing $U'U'^T$, you are not getting the dot product of the same rows as U, you are only getting the dot product of the first N components since U' is truncated. Therefore, it is not necessarily the case that $U'U'^T = I_{D \times D}$.*

Pseudoinverses

Let X be a matrix of size $D \times N$, where $D > N$, with "thin" SVD $X = U\Sigma V^T$. Assume that X has rank N.

Problem H [4 points]: Assuming that Σ is invertible, show that the pseudoinverse $X^+ = V\Sigma^+ U^T$ as given in class is equivalent to $V\Sigma^{-1}U^T$. Refer to lecture 10 (slide 53) for the definition of pseudoinverse.

Solution H: *From lecture, we defined the Σ^+ to be of the same size as Σ except that for every diagonal entry σ_{ii} in Σ, the corresponding entry in Σ^+ is $\sigma_{ii}^+ = 1/\sigma_{ii}$. Note that using "thin" SVD, Σ is a square matrix and thus $\Sigma^+\Sigma = \Sigma\Sigma^+ = I \implies \Sigma^+ = \Sigma^{-1}$. Therefore $X^+ = V\Sigma^+U^T = V\Sigma^{-1}U^T$.*

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

Solution I: *Note that $X^TX = (U\Sigma V^T)^T(U\Sigma V^T) = V\Sigma^2V^T$*
Want to show: $(X^TX)^{-1}X^T = V\Sigma^{-1}U^T \iff X^T = (X^TX)\cdot V\Sigma^{-1}U^T = V\Sigma^2V^T \cdot V\Sigma^{-1}U^T = V\Sigma U^T$.
Since we know that X^T does indeed equal to $V\Sigma U^T$, we equivalently know that $(X^TX)^{-1}X^T = V\Sigma^{-1}U^T$.

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.

Hint: Note that the transpose of a matrix is easy to compute. Compare the condition numbers of Σ and X^TX. The condition number of a matrix A is given by $\kappa(A) = \frac{\sigma_{max}(A)}{\sigma_{min}(A)}$, where $\sigma_{max}(A)$ and $\sigma_{min}(A)$ are the maximum and minimum singular values of A, respectively.

Solution J: *We want to find the singular values of Σ and X^TX to find their condition numbers.*

$$Singular(\Sigma) = \sqrt{Eigenvalues(\Sigma^T\Sigma)} = \sqrt{Eigenvalues(\Sigma^2)} = Singular(X)$$

$$Singular(X) = \sqrt{Eigenvalues(X^TX)}$$

$$Singular(X^TX) = \sqrt{Eigenvalues((X^TX)^2)} = Eigenvalues(X^TX)$$

since the Eigenvalues of a matrix A^2 equal the square of the eigenvalues of A.

Therefore, $Singular(X^TX) = (Singular(\Sigma))^2$ which implies that $\kappa(X^TX) = (\kappa(\Sigma))^2$. Since the condition number of X^TX is greater than zero, the $(X^TX)^{-1}X^T$ method is harder to compute and more prone to numerical errors.

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.

Solution A:
$\|U\|_F^2 = \sum_{m,k}(U_{mk})^2$ and $\|V\|_F^2 = \sum_{n,k}(V_{nk})^2$. Since the only parts of $\|U\|_F^2$ that are affected by u_i are those that are components in u_i, we can say that:

$$\implies \partial_{u_i} = \frac{\lambda}{2} \cdot 2 \cdot u_i - \sum_j v_j (y_{ij} - u_i^T v_j)^T = \lambda u_i - \sum_j v_j (y_{ij} - u_i^T v_j)^T$$

By symmetry of U and V and u_i and v_j,

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

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.

Solution B: *Want to find where $\partial_{u_i} = 0$ and $\partial_{v_j} = 0$ because this is the minimizing value of u_i and v_j.*

$$\begin{aligned}
\partial_{u_i} = 0 &= \lambda u_i - \sum_j v_j (y_{ij} - u_i^T v_j)^T \\
&= \lambda u_i - \sum_j v_j y_{ij}^T + \sum_j v_j v_j^T u_i \\
&= (\lambda I_K + \sum_j v_j v_j^T) u_i - \sum_j y_{ij} v_j \\
&\implies u_i = (\lambda I_K + \sum_j v_j v_j^T)^{-1} \sum_j y_{ij} v_j
\end{aligned}$$

and by symmetry with u_i,

$$\implies v_j = (\lambda I_K + \sum_i u_i u_i^T)^{-1} \sum_i y_{ij} u_i$$

Problem C [10 points]: Download the provided MovieLens dataset (train.txt and test.txt). The format of the data is (*user, movie, rating*), where each triple encodes the rating that a particular user gave to a particular movie. Make sure you check if the user and movie ids are 0 or 1-indexed, as you should with any real-world dataset.

Implement matrix factorization with stochastic gradient descent for the MovieLens dataset, using your answer from part A. Assume your input data is in the form of three vectors: a vector of is, js, and y_{ij}s. Set $\lambda = 0$ (in other words, do not regularize), and structure your code so that you can vary the number of latent factors (k). You may use the Python code template in 2_notebook.ipynb; to complete this problem, your task is to fill in the four functions in 2_notebook.ipynb marked with TODOs.

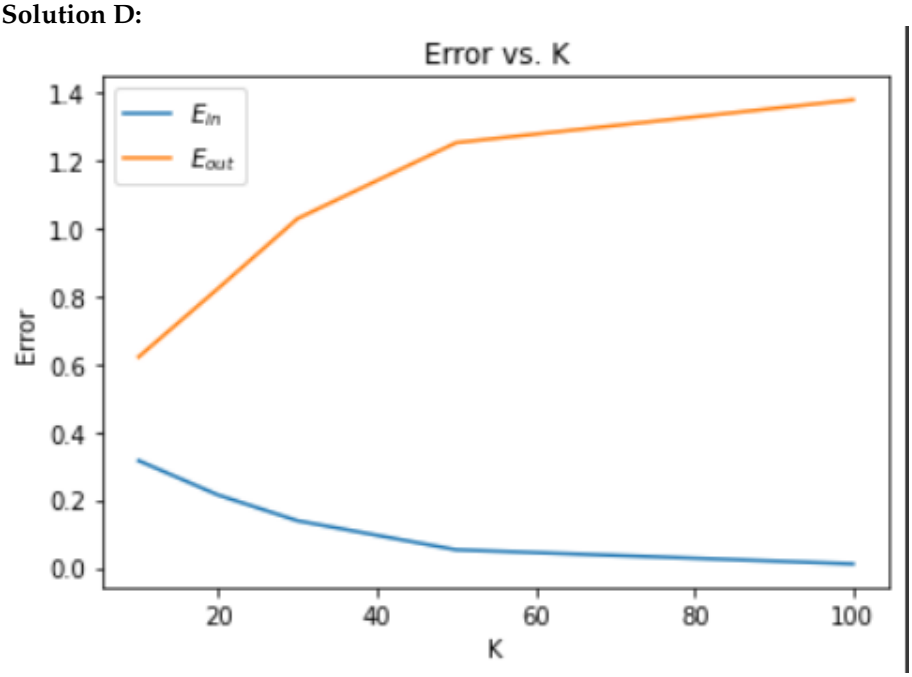
In your implementation, you should:

- Initialize the entries of U and V to be small random numbers; set them to uniform random variables in the interval $[-0.5, 0.5]$.
- Use a learning rate of 0.03.
- Randomly shuffle the training data indices before each SGD epoch.
- Set the maximum number of epochs to 300, and terminate the SGD process early via the following early stopping condition:
  - Keep track of the loss reduction on the training set from epoch to epoch, and stop when the relative loss reduction compared to the first epoch is less than $\epsilon = 0.0001$. That is, if $\Delta_{0,1}$ denotes the loss reduction from the initial model to end of the first epoch, and $\Delta_{i,i-1}$ is defined analogously, then stop after epoch t if $\Delta_{t-1,t}/\Delta_{0,1} \leq \epsilon$.

Solution C:
https://colab.research.google.com/drive/1xq6ay4UFAbaRP5m3ioCEhvQHG7-bMEpr

Problem D [5 points]: Use your code from the previous problem to train your model using $k = 10, 20, 30, 50, 100$, and plot your E_{in}, E_{out} against k. Note that E_{in} and E_{out} are calculated via the squared loss, i.e. via $\frac{1}{2}\sum_{i,j}\left(y_{ij} - u_i^T v_j\right)^2$. What trends do you notice in the plot? Can you explain them?

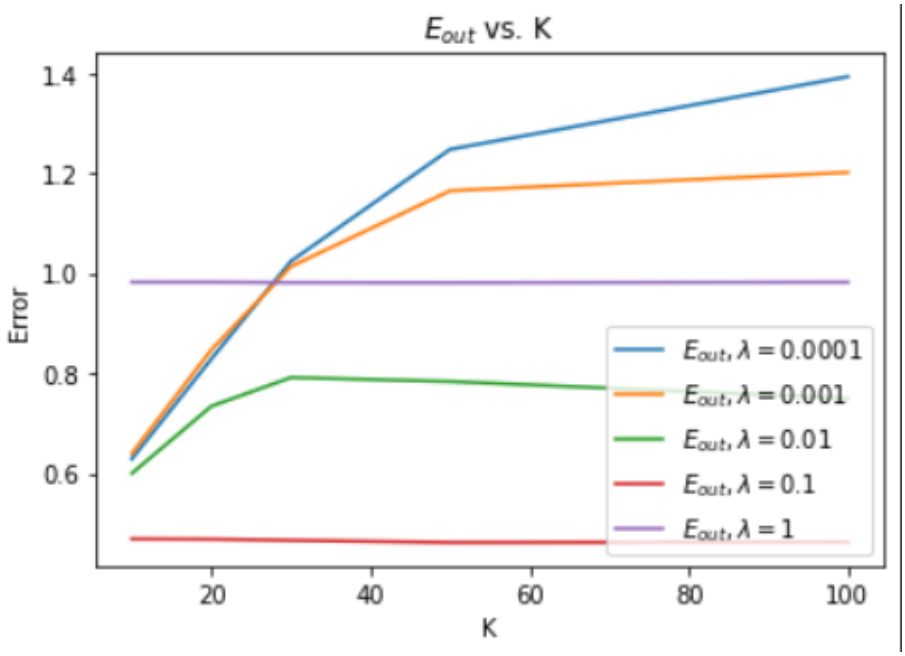
Solution D:



I notice that as the number of latent factors increases, the model better fits the training the data but the out-of-sample error increases. This means that there is overfitting and a very small k is enough to minimize the E_{out}.

Problem E [5 points]: Now, repeat problem D, but this time with the regularization term. Use the following regularization values: $\lambda \in \{1e-4, 1e-3, 0.01, 0.1, 1\}$. For each regularization value, use the same range of values for k as you did in the previous part. What trends do you notice in the graph? Can you explain them in the context of your plots for the previous part? You should use your code you wrote for part C in 2_notebook.ipynb.

Solution E:



We notice that for very small values of λ like $\lambda <= 0.001$, we notice very poor error and it is very similar to our unregularized test error. However, with $0.01 \leq \lambda \leq 0.1$, we note that the test error is significantly improved over our unregularized model. This makes sense because we noted in the last section that our model was prone to overfitting and thus regularization could help improve our test accuracy. However, we note that once $\lambda = 1$, the model is likely underfitting.

3 Word2Vec Principles [35 Points]

The Skip–gram model is part of a family of techniques that try to understand language by looking at what words tend to appear near what other words. The idea is that semantically similar words occur in similar contexts. This is called "distributional semantics", or "you shall know a word by the company it keeps".

The Skip–gram model does this by defining a conditional probability distribution $p(w_O|w_I)$ that gives the probability that, given that we are looking at some word w_I in a line of text, we will see the word w_O nearby. To encode p, the Skip-gram model represents each word in our vocabulary as two vectors in $\mathbb{R}^D$: one vector for when the word is playing the role of w_I ("input"), and one for when it is playing the role of w_O ("output"). (The reason for the 2 vectors is to help training — in the end, mostly we'll only care about the w_I vectors.) Given these vector representations, p is then computed via the familiar softmax function:

$$p(w_O|w_I) = \frac{\exp\left({v'}_{w_O}^T v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'}_{w}^T v_{w_I}\right)} \tag{2}$$

where v_w and v'_w are the "input" and "output" vector representations of word a $w \in \{1, ..., W\}$. (We assume all words are encoded as positive integers.)

Given a sequence of training words $w_1, w_2, \ldots, w_T$, the training objective of the Skip-gram model is to maximize the average log probability

$$\frac{1}{T}\sum_{t=1}^{T} \sum_{-s\leq j\leq s, j\neq 0} \log p(w_{t+j}|w_t) \tag{1}$$

where s is the size of the "training context" or "window" around each word. Larger s results in more training examples and higher accuracy, at the expense of training time.

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?

Solution A: *For each w_O, w_I pair, we need to loop all v_{w_t} in the denominator $\implies O(W)$. Then we must loop through all possible pairs of w_i and w_j of which there are W^2 pairs. This implies that the time complexity of calculating all the gradients $\nabla \log p(w_O|w_I)$ is $O(W^3)$.*

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. **??**). 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.

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

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”.

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.

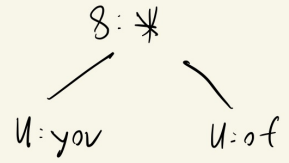
The representation length of a word is then the length of the path (the number of edges) from the root to the leaf node corresponding to the word. For each tree you constructed, compute the expected representation length (averaged over the actual frequencies of the words).

Solution B:

Problem 3.B. Manuel Rodriguez

Constructing Huffman Tree

- two lowest freq. are 4:you and 4:of

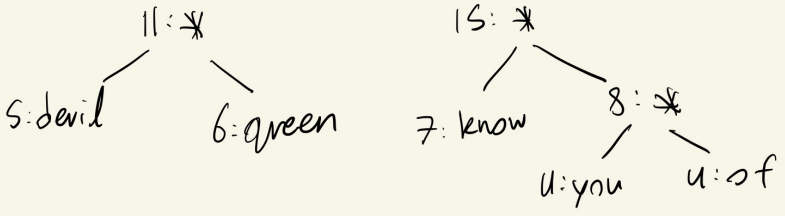


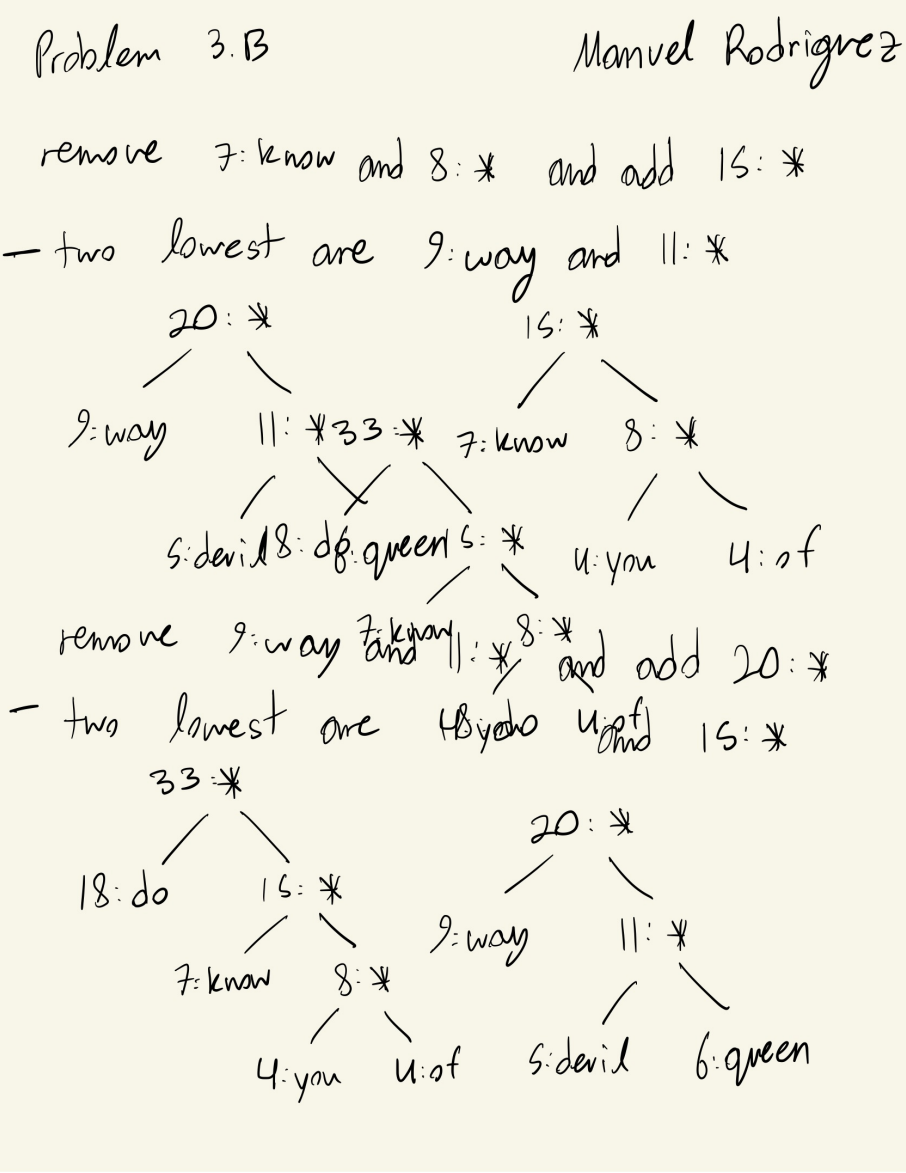
remove 4:you and 4:of and add 8:*

- two lowest are 5:devil and 6:queen

remove 5:devil and 6:queen and add 11:*

- two lowest are 7:know and 8:*

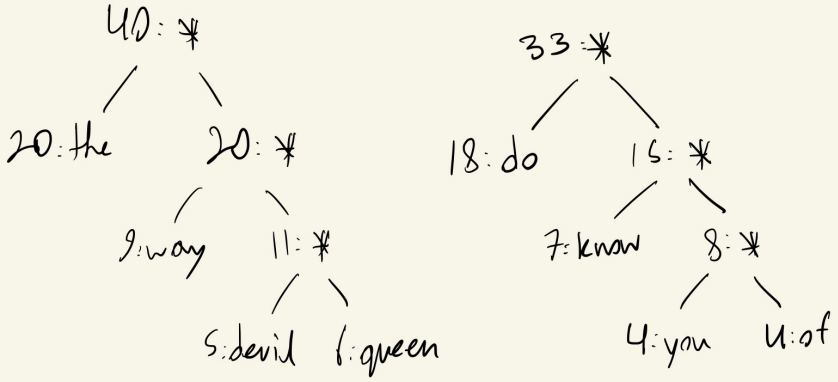



Problem 3.B
Manuel Rodriguez
remove 7: know and 8: * and add 15: *
- two lowest are 9: way and 11: *
20: *
15: *
9: way
11: *
33: *
7: know
8: *
5: devil
18: do
6: queen
5: *
4: you
4: of
remove 9: way and 7: know and add 20: *
- two lowest are 18: do and 4: of
15: *
33: *
20: *
18: do
15: *
9: way
11: *
7: know
8: *
4: you
4: of
5: devil
6: queen

Problem 3.B — Manuel Rodriguez

remove 18:do and 15:* and add 33:*

- two lowest are 20:the and 20:*



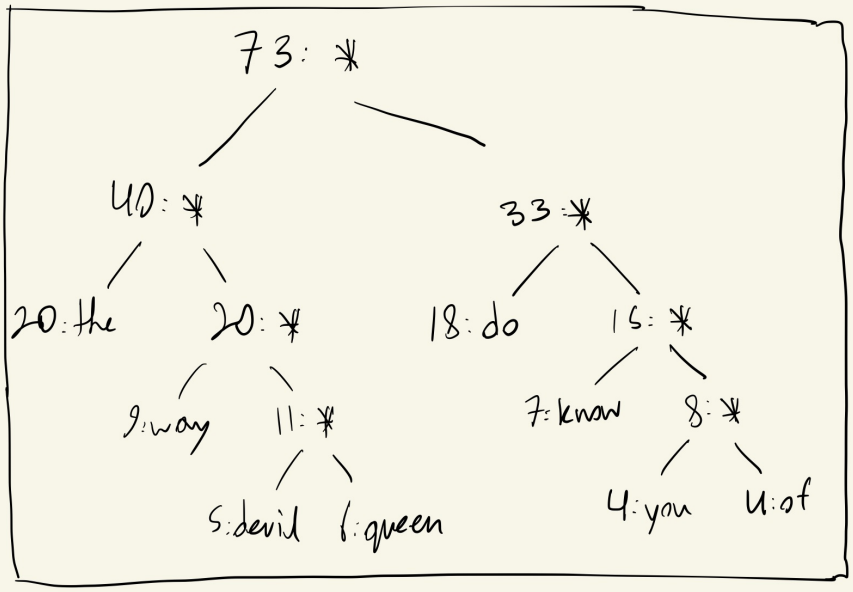
remove 20:the and 20:* and add 40:*

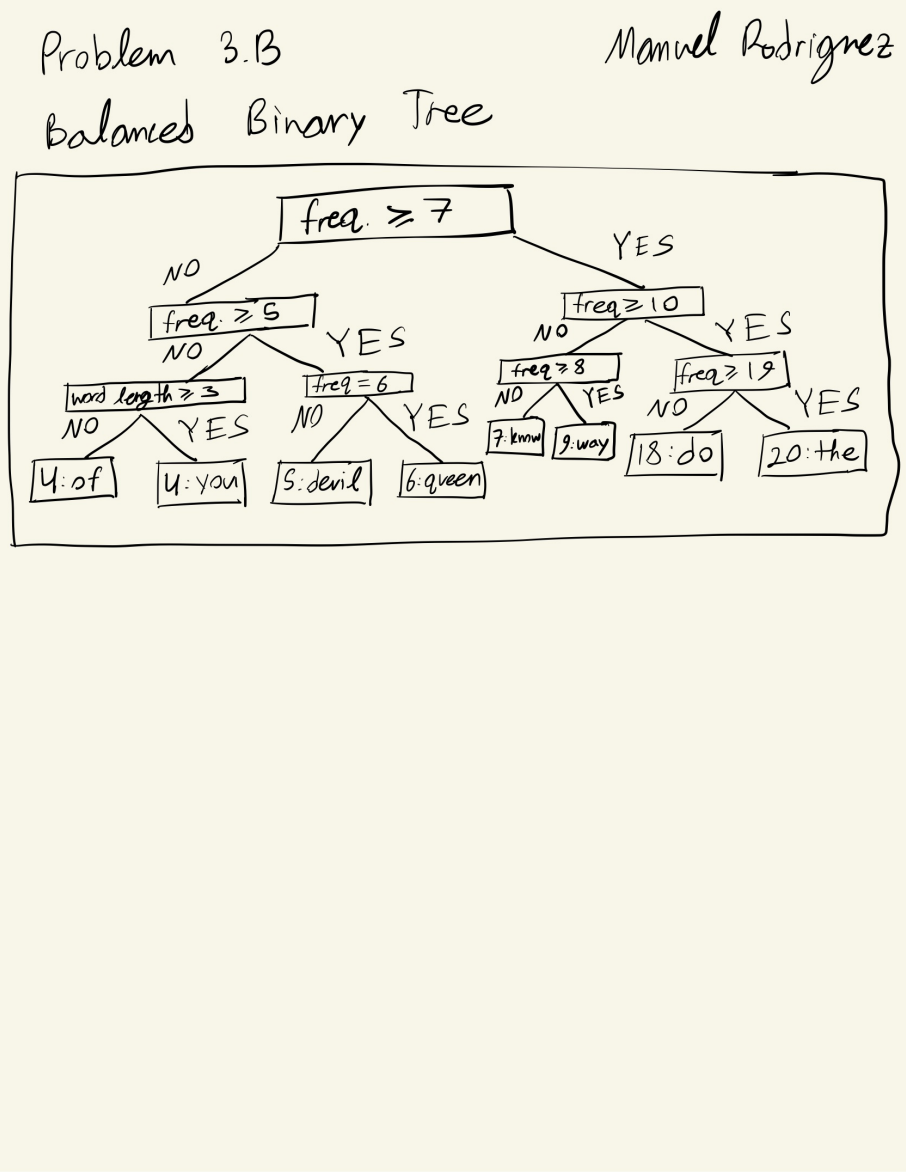
- only ones left are 40:* and 33:* to complete tree

Problem 3.B

Manuel Rodriguez

Completed Huffman Tree:





Problem C [3 points]: In principle, one could use any D for the dimension of the embedding space. What do you expect to happen to the value of the training objective as D increases? Why do you think one might not want to use very large D?

Solution C: *The purpose of having an embedding space is to capture more succinctly the information of the original space while reducing the effects of noise and overfitting to high dimensional data. If we increase D to be very large, this becomes the same as just not doing any embedding at all which would increase computation time*

and also likely cause overfitting of the training data.

Implementing Word2Vec

Word2Vec is an efficient implementation of the Skip–gram model using neural network–inspired training techniques. We'll now implement Word2Vec on text datasets using Keras. This blog post provides an overview of the particular Word2Vec implementation we'll use.

At a high level, we'll do the following:

(i) Load in a list L of the words in a text file

(ii) Given a window size s, generate up to $2s$ training points for word L_i. The diagram below shows an example of training point generation for $s = 2$:

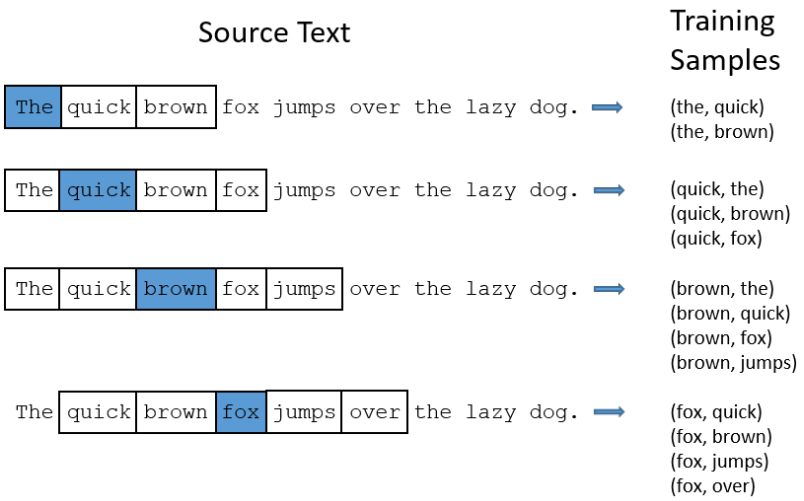


Figure 1: Generating Word2Vec Training Points

(iii) Fit a neural network consisting of a single hidden layer of 10 units on our training data. The hidden layer should have no activation function, the output layer should have a softmax activation, and the loss function should be the cross entropy function.

Notice that this is exactly equivalent to the Skip–gram formulation given above where the embedding dimension is 10: the columns (or rows, depending on your convention) of the input–to–hidden weight matrix in our network are the w_I vectors, and those of the hidden–to–output weight matrix are the w_O vectors.

(iv) Discard our output layer and use the matrix of weights between our input layer and hidden layer as the matrix of feature representations of our words.

(v) Compute the cosine similarity between each pair of distinct words and determine the top 30 pairs of most-similar words.

Implementation

See 3_notebook.ipynb, which implements most of the above.

Problem D [10 points]: Fill out the TODOs in the skeleton code; specifically, add code where indicated to train a neural network as described in (iii) above and extract the weight matrix of its input–to–hidden weight matrix. Also, fill out the generate_traindata() function, which generates our data and label matrices.

Solution D:
https://colab.research.google.com/drive/12sU1gFDjMZ_P72KzXAiGEFb_Qu0EyKGE?usp=sharing

Running the code

Run your model on dr_seuss.txt and answer the following questions:

Problem E [2 points]: What is the dimension of the weight matrix of your hidden layer?

Solution E: *Since we take in 308 words and have 10 units in our hidden layer, the dimension of the weight matrix of our hidden layer is* 308×10.

Problem F [2 points]: What is the dimension of the weight matrix of your output layer?

Solution F: *Similarly, since there are 10 units in our hidden layer and 308 output units, the dimension of the weight matrix of our output layer is* 10×308.

Problem G [1 points]: List the top 30 pairs of most similar words that your model generates.

Solution G: *From the output of my code: Pair(dont, pet), Similarity: 0.9551969295493797*
Pair(pet, dont), Similarity: 0.9551969295493797
Pair(they, name), Similarity: 0.9502663073608049
Pair(name, they), Similarity: 0.9502663073608049
Pair(star, think), Similarity: 0.9445964097336859
Pair(think, star), Similarity: 0.9445964097336859

Pair(quiet, or), Similarity: 0.941129125126621
Pair(or, quiet), Similarity: 0.941129125126621
Pair(sits, was), Similarity: 0.9256490892060885
Pair(was, sits), Similarity: 0.9256490892060885
Pair(did, pop), Similarity: 0.9180139785711674
Pair(pop, did), Similarity: 0.9180139785711674
Pair(cow, upon), Similarity: 0.916990391906577
Pair(upon, cow), Similarity: 0.916990391906577
Pair(of, in), Similarity: 0.9151490827981822
Pair(in, of), Similarity: 0.9151490827981822
Pair(have, day), Similarity: 0.9110737870160983
Pair(day, have), Similarity: 0.9110737870160983
Pair(i, stick), Similarity: 0.9088037098102797
Pair(stick, i), Similarity: 0.9088037098102797
Pair(go, pet), Similarity: 0.9038116241031461
Pair(mother, was), Similarity: 0.9034304275848821
Pair(fingers, how), Similarity: 0.9016397626926735
Pair(how, fingers), Similarity: 0.9016397626926735
Pair(he, thing), Similarity: 0.8998058346274731
Pair(thing, he), Similarity: 0.8998058346274731
Pair(mr, big), Similarity: 0.8885309277384515
Pair(big, mr), Similarity: 0.8885309277384515
Pair(bike, fear), Similarity: 0.8864382291502074
Pair(fear, bike), Similarity: 0.8864382291502074

Problem H [2 points]: What patterns do you notice across the resulting pairs of words?

Solution H: *I notice that words that large words tend to be correlated with small words, words that rhyme tend to be correlated with each other, and that verbs tend to be correlated with nouns (as opposed to with other verbs or adverbs). These trends follow from the fact that the dataset is from Dr. Seuss, a writer known to write simple sentences for children and rhyme in nearly every two adjacent lines.*