CSCI567: Machine Learning	USC, Spring 2026
Homework 4	
Instructor: Vatsal Sharan	*Due: April 16 by 11:59 am PST*

A reminder on collaboration policy and academic integrity: Our goal is to maintain an optimal learning environment. You can discuss the homework problems at a high level with other groups, but you should not look at any other group's solutions. Trying to find solutions online or from any other sources for any homework or project is prohibited, will result in zero grade and will be reported. To prevent any future plagiarism, uploading any material from the course (your solutions, quizzes etc.) on the internet is prohibited, and any violations will also be reported. Please be considerate, and help us help everyone get the best out of this course.

Please remember the Student Conduct Code (Section 11.00 of the USC Student Guidebook). General principles of academic honesty include the concept of respect for the intellectual property of others, the expectation that individual work will be submitted unless otherwise allowed by an instructor, and the obligations both to protect one's own academic work from misuse by others as well as to avoid using another's work as one's own. All students are expected to understand and abide by these principles. Students will be referred to the Office of Student Judicial Affairs and Community Standards for further review, should there be any suspicion of academic dishonesty.

Notes on notation:

- Unless stated otherwise, scalars are denoted by small letter in normal font, vectors are denoted by small letters in bold font and matrices are denoted by capital letters in bold font.
- $\|.\|$ means L2-norm unless specified otherwise, *i.e.*, $\|.\| = \|.\|_2$.

Instructions

We recommend that you use LaTeX to write up your homework solution. However, you can also scan handwritten notes. The homework will need to be submitted on Gradescope.

Theory-based Questions

Problem 1: Decision Trees (12pts)

Consider a binary dataset with 400 examples, where half of them belong to class A and the rest belong to class B. Next, consider two decision stumps (i.e. trees with depth 1) $\mathcal{T}_1$ and $\mathcal{T}_2$, each with two children. For $\mathcal{T}_1$, the left child has 150 examples in class A and 50 examples in class B. For $\mathcal{T}_2$, the left child has 0 examples in class A and 100 examples in class B. (You can infer the number of examples in the right child using the total number of examples.)

1.1 (6 pts) In class, we discussed entropy and Gini impurity as measures of uncertainty at a leaf. Another possible metric is the classification error at the leaf, assuming that the prediction at the leaf is the majority class among all examples that belong to that leaf. For each leaf of $\mathcal{T}_1$ and $\mathcal{T}_2$, compute the entropy (base e), Gini impurity and classification error. You can either exactly express the final numbers in terms of fractions and logarithms, or round them to two decimal places.

1.2 (6 pts) Compare the quality of $\mathcal{T}_1$ and $\mathcal{T}_2$ (that is, the two different splits of the root) based on conditional entropy (base e), weighted Gini impurity, and total classification error. Intuitively, which of $\mathcal{T}_1$ or $\mathcal{T}_2$ appears to be a better split to you (there may not necessarily be one correct answer to this)? Based on your conditional entropy, Gini impurity, and classification error calculations, which of the metrics appear to be more suitable choices to decide which variable to split on?

Problem 2: Attention (12pts)

(This problem is partly adapted from an assignment (`https://web.stanford.edu/class/cs224n/assignments_w26/a3.pdf`) at CS224N at Stanford, and we thank the staff of CS224N for letting us adapt it. There are also other good problems in that assignment if you are interested in understanding transformers further.)

For a query $\boldsymbol{q}$, keys $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_n$, and values $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$, the single-headed attention mechanism outputs a weighted combination of the values, i.e., $\boldsymbol{o} = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i$, where the weights $\alpha_1, \ldots, \alpha_n$ are given by

$$\alpha_i = \frac{\exp\left(\boldsymbol{q}^T \boldsymbol{k}_i\right)}{\sum_{j=1}^{n} \exp\left(\boldsymbol{q}^T \boldsymbol{k}_j\right)}, \quad \forall\, i = 1, \ldots, n.$$

This allows the attention mechanism to selectively aggregate information from different values depending on the query. In this problem, you shall explore some fundamental properties of the single-headed attention mechanism.

2.1 (3 pts) **Copying.** The attention mechanism can copy a specific value (say $\boldsymbol{v}_j$) to the output $\boldsymbol{o}$, i.e., $\boldsymbol{o} \approx \boldsymbol{v}_j$. Note that exact copying ($\boldsymbol{o} = \boldsymbol{v}_j$) is not possible with the softmax function since $0 < \alpha_i < 1$ for all i, but approximate copying, i.e. $\boldsymbol{o} \approx \boldsymbol{v}_j$, is achievable. Derive a relationship between the query and keys such that this is possible.

Hint: Try to relate α_j with α_i for $i \neq j$.

2.2 (5 pts) **Averaging.** The attention mechanism can aggregate information equally from two different values $\boldsymbol{v}_a, \boldsymbol{v}_b$, $a \neq b$, so that $\boldsymbol{o} \approx \frac{1}{2}(\boldsymbol{v}_a + \boldsymbol{v}_b)$. Assume that the keys are orthogonal, i.e., $\boldsymbol{k}_i^\intercal \boldsymbol{k}_j = 0$ for all $i \neq j$, and have unit norm ($\|\boldsymbol{k}_i\|_2 = 1$ for all i). Derive an expression for the query $\boldsymbol{q}$ (in terms of the keys) such that this is possible.

Hint: The desired $\boldsymbol{q}$ can be expressed as a linear combination of the keys. You might also want to introduce a large positive constant in the expression for $\boldsymbol{q}$.

Self-Attention. In natural language, a word often derives its meaning or reference from other words in the same sentence. Consider the sentence *"Mary packed her bags before the trip."* To correctly interpret *her*, a model must link it back to *Mary* — the word *her* must *attend* to *Mary*. Beyond such coreference resolution, the meaning of a word can shift entirely based on context — *bank* means something different when surrounded by *river* and *fishing* versus *loan* and *interest*. In all these cases, understanding a token requires aggregating information from other tokens in the sequence.

Self-attention achieves this by applying the attention mechanism above within a sequence: each token acts as a query and attends over all other tokens, using learned matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ to determine which tokens to focus on. Formally, let $d > n$ and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ be an input sequence of token embeddings, where $\boldsymbol{x}_i \in \mathbb{R}^d$. For each token i, the self-attention layer maintains query $\boldsymbol{q}_i = \boldsymbol{Q}\boldsymbol{x}_i$, key $\boldsymbol{k}_i = \boldsymbol{K}\boldsymbol{x}_i$, and value $\boldsymbol{v}_i = \boldsymbol{V}\boldsymbol{x}_i$, with attention weights

$$\alpha_{i,j} = \frac{\exp\left(\boldsymbol{q}_i^\intercal \boldsymbol{k}_j\right)}{\sum_{j'=1}^{n} \exp\left(\boldsymbol{q}_i^\intercal \boldsymbol{k}_{j'}\right)},$$

and output $\boldsymbol{o}_i = \sum_{j=1}^{n} \alpha_{i,j} \boldsymbol{v}_j$. As a simple first instance of token referencing, we say the self-attention layer allows each token to reference only its immediately preceding token if $\boldsymbol{o}_i \approx \boldsymbol{v}_{i-1}$ for all $i \geq 2$, i.e., the output of token i is primarily determined by the value of token $i - 1$.

2.3 (4 pts) Assume $\boldsymbol{x}_i = \boldsymbol{e}_i$ for all i, where $\boldsymbol{e}_i$ is the i-th standard basis vector. Show that there exist matrices $\boldsymbol{Q}$ and $\boldsymbol{K}$ such that $\boldsymbol{o}_i \approx \boldsymbol{v}_{i-1}$ for all $i \geq 2$, and prove your construction is correct. We recommend spending some time trying to figure out the construction on your own first. Then, if you want help you can look at the hint in the footnote here and prove that the construction there is correct.[1]

[1] Construction hint: Let c be a large positive constant. Set $\boldsymbol{K} = \boldsymbol{I}$ and $\boldsymbol{Q}$ such that the i-th row of $\boldsymbol{Q}$ is $ce_{i+1}^\intercal$ for $i = 1, \ldots, d-1$ and the p-th row of $\boldsymbol{Q}$ is the zero vector.

Programming-based Questions

As in previous homeworks, you need to have your coding environment setup for this part. We use python3 (version $\geq$ 3.7) in our programming-based questions. There are multiple ways you can install python3, for example:

- You can use conda to configure a python3 environment for all programming assignments.
- Alternatively, you can also use virtualenv to configure a python3 environment for all programming assignments

After you have a python3 environment, you will need to install the following python packages:

- numpy
- matplotlib (for plotting figures)

Note: You are **not allowed** to use other packages such as *tensorflow, pytorch, keras, scikit-learn, scipy*, etc. for 3.1-3.2. If you have other package requests, please ask first before using them. You are **allowed** to use any packages for 3.3-3.5.

Download the files for the programming part from `https://vatsalsharan.github.io/spring26/hw4.zip`.

Problem 3: Exploring Decision Trees and Random Forests (12pts)

In this question, we will observe the effect of different hyperparameters in training decision trees and random forests, and also visualize what features the random forest model is using to make predictions. We will do this on a Colab notebook `HW4-Exploring-Random-Forests.ipynb`.

Instructions to run the notebook: Upload this file to your USC Google Drive. Then, add Google Colab to your Google App by New $\rightarrow$ More $\rightarrow$ Connect more apps $\rightarrow$ Type in Google Colab and install it, as in Fig. 1. After that, you can run the notebook with your browser.

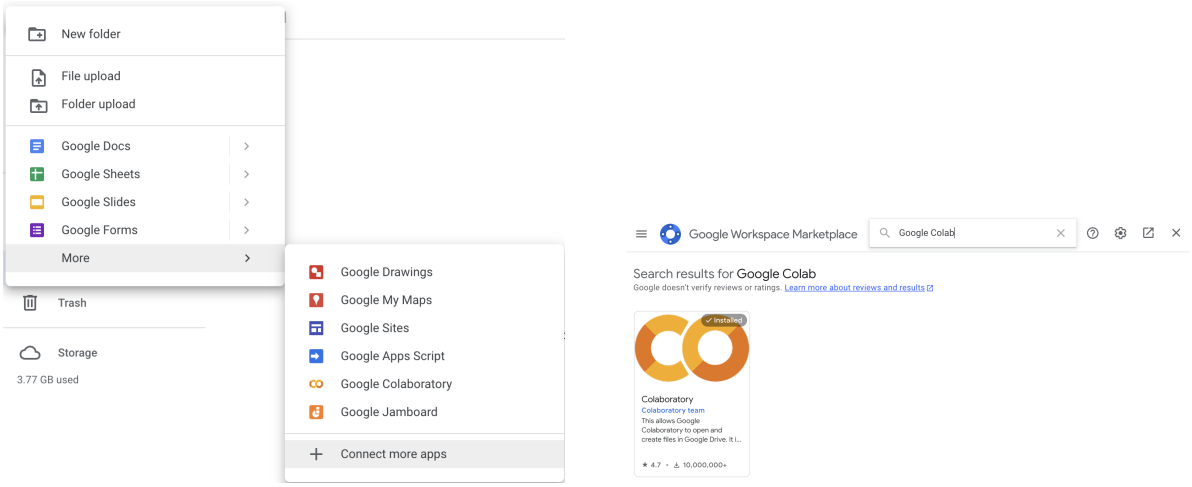

Figure 1: Screenshots showing how to install Colab.

We use a variant of the MNIST dataset for this problem. As mentioned in HW 3, the MNIST dataset contains images of handwritten digits (0, 1, 2, ..., 9) and is generally used for the (10) digit classification problem. Here, we work with a binary classification task of predicting whether the digit is less than 5 or not. Fig. 2 shows some sample images from the dataset with original and binary labels, respectively.

You should go through the code we provide and understand what's happening, but for the purpose of answering the questions you will mainly have to run the code and understand the results. All the models (decision trees, random forests, etc.) are imported from the `sklearn` library. You should feel free to explore the role of other hyperparameters and other methods (such as bagging, and boosting), and go through the documentation to better understand these things, but for this question, we will focus on decision trees and random forests.

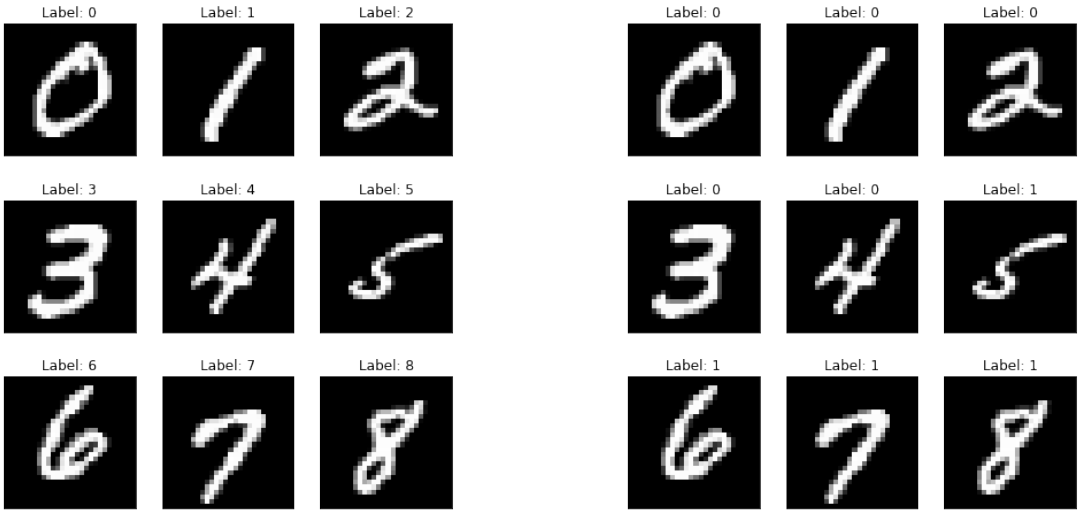

Figure 2: Some sample images from the MNIST dataset with original (left) and binary (right) labels, respectively.

We will look at the effect of 5 parameters:

- `max_depth`: The maximum depth of the decision tree.
- `n_estimators`: The number of decision trees in the forest.
- `min_samples_leaf`: The minimum number of samples required to be a leaf node.
- `max_samples`: The number of samples to draw from the training set to train each decision tree in the forest.
- `max_features`: The number of features to consider when looking for the best split for any node.

To observe the effect of a parameter, we look at train and test accuracy for different values of that parameter while keeping the rest of the parameters fixed.

3.1 (2 pts) The first set of plots shows the train and test accuracy for different values of `max_depth` using a decision tree and a random forest, respectively. How do the accuracies and the generalization gap vary with `max_depth` in these cases? For a particular value of `max_depth`, how does the generalization gap for the decision tree compare with that of the random forest? Explain your observations.

3.2 (2 pts) The next plot shows the train and test accuracy for different values of `n_estimators` for a random forest. Comment on your observations regarding the accuracies and the generalization gap. What value(s) (range or approximate values are enough) would you prefer to use for this parameter? Give reasons why. Hint: Are there any drawbacks to using very high values of `n_estimators`?

For the next three parts of this question, we will also look at how the size of the training set influences the accuracy trends for a given parameter. In each case, for the first plot, the training set consists of 4000 samples whereas for the second plot, it contains 1000 samples.

3.3 (2 pts) The next set of plots shows the train and test accuracy for different values of `min_samples_leaf` for a random forest. Taking into account the behaviors for different training set sizes, explain your observations for very low and very high values of `min_samples_leaf`. What do you conclude from this trend? What could be the reasons for such a behavior?

3.4 (2 pts) The next set of plots shows the train and test accuracy for different values of `max_samples` for a random forest. What do you observe for very low and very high values of `max_samples`? Would you prefer to use

low, intermediate, or high values for this parameter in both cases? Give reasons why. Hint: How does the size of the training set influence the choice of this parameter?

3.5 (2 pts) The next set of plots shows the train and test accuracy for different values of `max_features` for a random forest. Comment on your observations regarding the accuracies and the generalization gap for the two training set sizes. What is the best range of values for this parameter in both cases? Is it similar/different? Explain your observations.

3.6 (2 pts) In class, we discussed how ensembles are usually not as interpretable as a single decision tree. While this is true, there are still ways to explore which features are used the most by our ensemble. We will explore one such technique in this part.

We visualize the *feature importances* of a random forest model trained for the binary classification task on the MNIST dataset. Intuitively, features with higher importance are the pixels which are used more often in the decision trees in the forest and which lead to better splits, i.e. which contribute more to improving the performance of the model. For more details, you can see Section 18.6.1 of the PML book. The last plot shows the importance of different pixels/portions of the image for a trained random forest model to make its predictions. What portions of the image does the model seem to be focusing on? In other words, can you think of reasons why the pixels with higher importance are indeed important for the prediction task (classifying whether the digit is smaller than 5 or not)? As is usually true for such open-ended questions, there can be multiple correct answers here and we're looking more for your reasoning than a specific answer.

Problem 4: PCA for Learning Word Embeddings(30pts+5pts Bonus)

This question is about *word embeddings*. We saw word embeddings in class in lecture 8. As we discussed then, word embedding is simply a vector space representation of words which captures some of the semantic and syntactic structures in the language—for example, words similar in meaning have representations which are close to each other in the vector space. Word embeddings have taken natural language processing (NLP) by storm in the past decade or so, and have become the backbone for numerous NLP tasks such as question answering and machine translation. There are neural approaches to learning word embeddings, but in this question, we will study a simple PCA-based scheme which does a surprisingly good job at learning word embeddings.

We have created a word co-occurrence matrix $\mathbf{M}$ of the 10000 most frequent words from a Wikipedia corpus with 1.5 billion words. The co-occurrences were obtained by using a sliding window of length 5 across the Wikipedia corpus. Entry M_{ij} of the matrix denotes the number of times words i and j occur in the corpus within the same sliding window. The file `co_occur.csv` available at this link[2] contains the symmetric co-occurrence matrix. `dictionary.txt` contains the dictionary for interpreting this matrix, the ith row of the dictionary is the word corresponding to the ith row or column of $\mathbf{M}$. The dictionary is sorted according to the word frequencies. Therefore the first word in the dictionary—"the" is the most common word in the corpus and the first row or column of $\mathbf{M}$ contains the co-occurrence counts of "the" with every other word in the dictionary.

We provide some starter code in the file `hw4-pca.py` with some useful functions (e.g. to read files, generate plots, etc.) which can be used directly, and instructions on how to complete the functions required for this problem.

4.1 (6 pts) First, read the co-occurrence matrix and the list of all words from the given files. Let the matrix $\mathbf{M}$ be the $n \times d$ ($n = d = 10000$) matrix of word co-occurrences. As we discussed in class, a suitable normalization or scaling is often very helpful to get PCA to work well. In light of the power law distribution of word occurrences, in this case, we will work with the normalized matrix $\tilde{\mathbf{M}}$ such that each entry $\tilde{M}_{ij} = \log(1 + M_{ij})$. We regard the i-th row of $\mathbf{M}$ as the datapoint for the i-th word.

We will use PCA to find the first 100 principal components of the data. Let $\tilde{\mathbf{M}}_c$ be the centered version of $\tilde{\mathbf{M}}$. Use the `PCA` function from the `sklearn` library (refer `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html`) to get $\mathbf{V}$, *i.e.* the set of first 100 principal components or eigenvectors of $\tilde{\mathbf{M}}_c$. Note that you can directly use the `fit` method with $\tilde{\mathbf{M}}_c$ as the input. Also, get the eigenvalues of the covariance matrix (check the documentation of the function) and plot all the 100 eigenvalues. Do the eigenvalues seem to decay? What percent of the variance in the data is explained by the first 100 eigenvalues we calculated (note that there are 10,000 eigenvalues in total)?

4.2 (6 pts) In this question, we find embeddings for all 10,000 words in the dictionary using the principal components $\mathbf{V}$. Then, we will use word embeddings to find word(s) which are 'similar' to a given word.

To obtain the embeddings of the words, we will project the datapoint corresponding to each word (its co-occurrences with every other word) onto the 100-dimensional space spanned by the first 100 principal components, similar to the general approach we laid down in class. Here are the two steps you should follow. Recall that we regard the i^{th} row of $\tilde{\mathbf{M}}_c$ as the datapoint for the i^{th} word. Given the 100 PCs ($\mathbf{V}$), we now project each datapoint (row of $\tilde{\mathbf{M}}_c$) onto these PCs. Denote this $n \times k$ (10000×100) matrix as $\mathbf{P}$ (you should write $\mathbf{P}$ as a matrix operation using $\tilde{\mathbf{M}}_c$ and $\mathbf{V}$). Next, to ensure that each PC gets equal importance, we normalize the vector of the projections of all the words onto the j^{th} PC (*i.e.* the j^{th} column of $\mathbf{P}$) to have unit norm, for all $j = \{1, ..., 100\}$. Denote this $n \times k$ (10000×100) matrix as $\mathbf{E}$. Finally, normalize the rows of $\mathbf{E}$ such that each row has unit ℓ_2 norm, to get a new matrix $\tilde{\mathbf{E}}$.

We regard the ith row of $\tilde{\mathbf{E}}$ as the embedding of the ith word. Next, we will define a similarity metric for the word embeddings. We will use the cosine-similarity as the similarity metric. As all the vectors have unit ℓ_2 norm, the cosine similarity between two words i and j with embeddings $\mathbf{w}_i$ and $\mathbf{w}_j$ is equal to the inner product $\langle \mathbf{w}_i, \mathbf{w}_j \rangle$. Now that we have a similarity metric defined, we can have some fun with these embeddings by querying for the closest word to any word we like! Try finding the closest words to some common words, such as "learning", "university", "california",

[2] `https://drive.google.com/file/d/1XvQ6s0d8QSiSG8bncqcpTQtR8XODb9-G/view?usp=sharing`

and comment on your observations.

4.3 (6 pts) We'll now interpret the principal components/eigenvectors (columns of $\mathbf{V}$). For any i, denote $\mathbf{v}_i$ as the eigenvector corresponding to the ith largest eigenvalue. Note that the entries of this vector correspond to the 10000 words in our dictionary, we'll call these our 10000 variables. By sorting the entries of $\mathbf{v}_i$ by absolute value, and observing what the top 10 variables and their (signed) entries are, we can infer what information the ith eigenvector roughly captures. Can you find 5 interesting eigenvectors, and point out what semantic or syntactic structures they capture? Can you do this for all 100 eigenvectors? Hint: What do you observe about PCs or eigenvectors with small eigenvalues?

4.4 (12 pts) In this question, we will explore a curious property of the word embeddings—that certain directions in the embedded space correspond to specific syntactic or semantic concepts. Let $\mathbf{w}_1$ be the word embedding for "woman" and $\mathbf{w}_2$ be the word embedding for "man". Let $\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_2$, and $\hat{\mathbf{w}} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$.

4.4.1 (6 pts) Project the embeddings of the following words onto $\hat{\mathbf{w}}$: *boy, girl, brother, sister, king, queen, he, she, john, mary, wall, tree*. Present a plot of projections of the embeddings of these words marked on a line. For example, if the projection of the embedding for "girl" onto $\hat{\mathbf{w}}$ is 0.1, then you should label 0.1 on the line with "girl". What do you observe?

4.4.2 (6 pts) Present a similar plot of the projections of the embeddings of the following words onto $\hat{\mathbf{w}}$: *math, history, nurse, doctor, pilot, teacher, engineer, science, arts, literature, bob, alice.* What do you observe? Why do you think this is the case? Do you see a potential problem with this? Remember that word embeddings are extensively used across NLP. Suppose LinkedIn used such word embeddings to find suitable candidates for a job or to find candidates who best match a search term or job description. What might be the result of this?

If you want to learn more about this, you might find it interesting to read the original paper[3] which pointed out this issue in word embeddings.

4.5 (Bonus) (5 pts) In this question, we will explore the property that directions in the embedded space correspond to semantic or syntactic concepts in more depth.

Because word embeddings capture semantic and syntactic concepts, they can be used to solve word analogy tasks. For example, consider an analogy question— *"man is to woman as king is to ___"*, where the goal is to fill in the blank space. This can be solved by finding the word whose embedding is closest to $\mathbf{w}_{\text{woman}} - \mathbf{w}_{\text{man}} + \mathbf{w}_{\text{king}}$ in cosine similarity. You can do this by a nearest neighbor search across the entire dictionary—excluding the three words *man, woman, king* which already appear in the analogy as they cannot be valid answers to the question. Here $\mathbf{w}_i$ represents the word embedding for the word i. Refer to Fig. 3 for why this makes sense because directions in the embedded space correspond to semantic/syntactic concepts.

We have provided a dataset, `analogy_task.txt`, which tests the ability of word embeddings to answer analogy questions, such as those represented in Fig. 3. Using the cosine similarity metric, find and report the accuracy of the word embeddings you have constructed on the word analogy task. Look at the incorrect/correct answers of the approach and comment on the results. For example, what types of analogy questions seem to be harder to answer correctly for this approach?

Deliverables: Discussions for all the parts. Plots for parts **4.1**, **4.4.1**, **4.4.2**. Code for all the parts as a separate Python file or iPython notebook.

[3] `https://proceedings.neurips.cc/paper/2016/file/a486cd07e4ac3d270571622f4f316ec5-Paper.pdf`

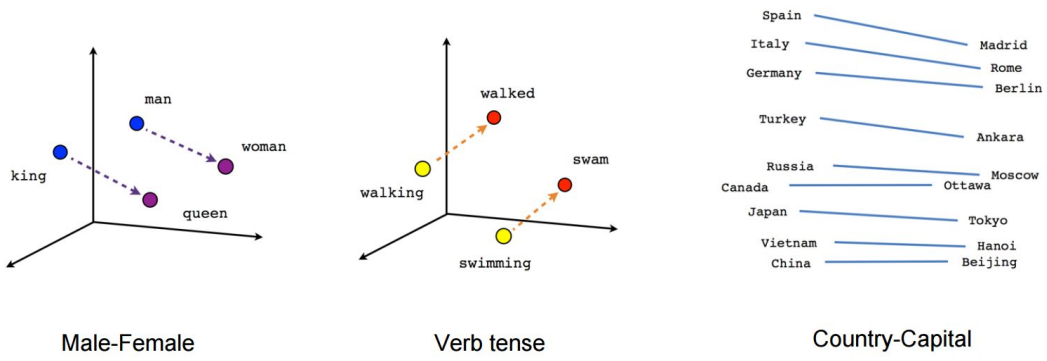

Figure 3: Figure denoting how word embeddings might encode gender, tense, or country-capital relationships. Image source.

Problem 5: Bonus Question on Transformers (10pts)

The Transformer architecture is important for ML because it revolutionized NLP by introducing the self-attention mechanism, enabling effective modeling of long-range dependencies in sequential data. This led to state-of-the-art performance in various NLP tasks such as machine translation, text generation, and sentiment analysis. In this problem, you will run a tiny decoder-only Transformer architecture and gain insight into its basic structure. We will do this using a Colab notebook, `transformer.ipynb`. To run the notebook, upload the `transformer.ipynb` to your USC Google Drive and follow the instructions in Problem 3 Part II (Explore on Colab) of HW3. The notebook is based on PyTorch, a popular library for neural networks. Similar to TensorFlow, PyTorch enables us to build and train neural networks with just a few simple lines of code.

The main purpose of this assignment is to familiarize you with the PyTorch framework and the Transformer architecture. To do this, you will train a small decoder-only Transformer model with the tiny Shakespeare dataset and observe the effects of some hyperparameters.

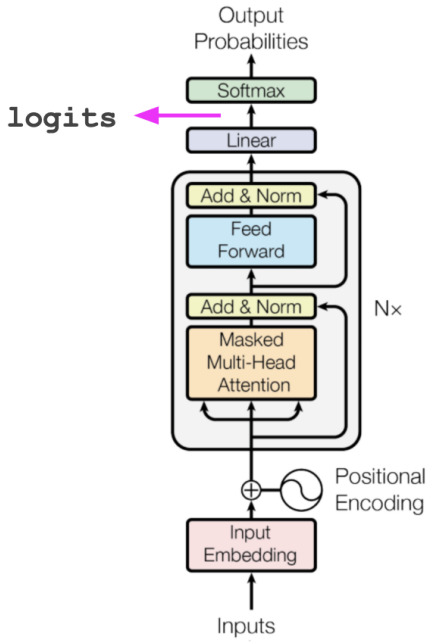

Figure 4: Decoder-only transformer architecture

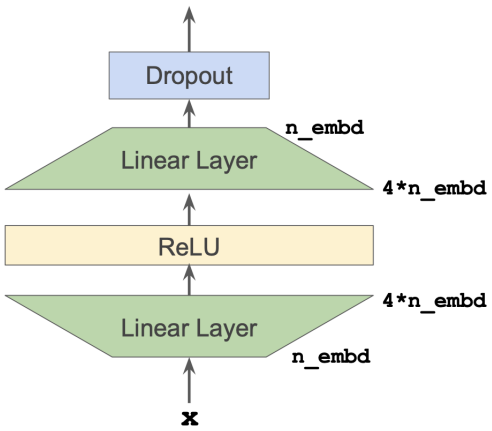

Figure 5: Feed Forward Module of Transformer

Transformer Structure First, you are expected to understand the code of the transformer which composed of multiple components. This section is not graded.

(a) Attention Layer: We divide theattention layer into two main components: Head, and MultiHeadAttention. Inside Head module, the attention mechanism is implemented. We provide you the implemented version, however you are highly encouraged to understand the code. MultiHeadAttention module is responsible for combining the results of Heads. It first gets all results from Heads, then concatenates them. Finally, this is followed by a linear layer and dropout regularization.

(b) Feed Forward (FF) Module: FF module in our network contains 2 linear layerswith the ReLU activation function applied in between. After the second linear layer, dropout regularization is applied. FF module first increases the embedding dimension 4 times via the first linear layer. Then decreases it to its original embedding dimension via the second linear layer. You can see the diagram in Figure 5.

(c) Transformer Block: A block contains 4 components (See also Figure 4, grey shaded block): Self-Attention module, Feed Forward module and 2 Layer-Normalization modules.

(d) Model: BigramLanguageModel module brings everything together and creates the whole transformer architecture (Figure 4). We also calculate the loss inside this function. We also provide the implementation of the `generate` function.

5.1 Training the Model (10pts) Now, we will train our model. In this section, you are not expected to implement anything.

(i). Before training the model, make it generate some sequence. Then, you will train the model with tiny Shakespeare dataset. We provided you the initial hyper-parameters. With Colab, the training session should take around 10 minutes. After training the model, make it generate some sequence again. Compare it with the first generation. Also, compare the generation with training data. Do they look similar? Explain the reason why the generation is not like a Wikipedia article, for example. (3pts)

(ii). Provide the loss curve for train and validation during training. (1pt)

(iii). For different values of block-size (sequence-length/context-length) [2, 16, 64], train the model from scratch. Provide the train-validation loss curves and model's generations for different sequence-lengths. Discuss how the sequence-length changes the generation of the model. (3pts)

(iv). For different number of heads [2, 4, 8], train the model from scratch. Provide the train-validation loss curves and model's generations. Discuss how the number of heads changes the train-validation loss curves. (3pts)

Deliverables for Problem 5: In general you should include all explanations we asked for in the question. For 5.1 (i), discussion on generations and screenshots of the generations. Plots for 5.1 (ii), (iii) (iv). Discussion and analysis for 5.1 (iii) and (iv). Screenshots of the generations for 5.1 (iii).