Consider the following word-context matrix with the three words "pear","banana" and "bus" and the three context words "juice", "the" and "drive".

	juice	the	drive	
pear	10	20	0	30
banana	8	20	0	28
bus	1	20	10	31
	19	60	10	

(a) Compute the MLEs using frequencies for the probabilities $P(w)$, $P(c)$ and $P(w,c)$ for each word w and each context word c.

total words = 30 + 28 + 31 = 89

P(w)
P(pear) = 30/89
P(banana) = 28/89
P(bus) = 31/89

P(c)
P(juice) = 19/89
P(the) = 60/89
P(drive) = 10/89

P(w,c)
P(pear, juice) = 10/89
P(pear, the) = 20/89
P(pear, drive) = 0
P(banana, juice) = 8/89
P(banana, the) = 20/89
P(banana, drive) = 0
P(bus, juice) = 1/89
P(bus, the) = 20/89
P(bus, drive) = 10/89

Based on these: i) write the PMI formula, and ii) compute the PMI values for the cells in the matrix

$$Pmi(w,c) = \log\left(\frac{P(w,c)}{P(w)\,P(c)}\right)$$

es. $Pmi(pear, juice) = \log\left(\frac{10/89}{(30/89)\,(19/89)}\right) = 0.446$

	juice	the	drive
Pear	0.446	-0.011	$-\infty$
banana	0.291	0.058	$-\infty$
bus	-1.890	-0.044	1.055

	NNP	MD	VB	JJ	NN	RB	DT
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

Figure 1: Matrix A.

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Figure 2: Matrix B.

W.r.t. Hidden Markov models and POS tagging, and given the two matrices A and B (see Figure 1, 2).

(a)
- identify the set Q of N states: $Q = \{\dots\}$
- identify the set O of observations: $O = \{\dots\}$

a) $Q = \{NNP, MD, VB, JJ, NN, RB, DT\}$

$O = \{janet, will, back, the, bill\}$

b) Each cell a_{ij} in A represents the transition probability from i to j.
$a_{ij} = P(s_t = j \mid s_{t-1} = i)$

c) The initial probabilities can be found in row <s> in matrix A, representing the probabilities of transitioning from start state to each Pos tag

d) Each cell $b_i(o_j)$ in B represents the emission probability of observing o_j given state i
$b_i(o_j) = P(o_t = o_j \mid s_t = i)$

You have a word2vec model trained with skip-gram over a corpus that has vocabulary of 5,000,000 word tokens. The word embedding dimension is set up to be 100.

- How many parameters you have to learn in this model?
- Describe in details the loss used to train word2vec.
- Why in the word2vec model you have to compare the center word with all context words, if in the loss you are using just the index of the correct context word?

Params = $V \times D \times 2 = 5.000.000 \times 100 \times 2 = 1.000.000.000$

Loss is negative sampling, maximize the probability of the context word given the center word

$L = -\log(\sigma(v_c \cdot v_w)) - \sum_{i=1}^{k} \log(\sigma(-v_n \cdot v_w))$

Comparing center word with all context (positive and negative) ensures that the embedding vectors learn to represent meaningful relationship between words.

Positive samples ensures that similar context have similar vectors

Negative samples the model learn correct and incorrect pairs

Using indices significantly reduces complexity, instead of computing gradients for the entire vocabulary, the model only updates a small number of negative samples

You have computed the term-to-document matrix $\mathbf{X} \in \mathbb{R}^{|D| \times |V|}$ using TF-IDF, where D is the number of documents and $|V|$ is the vocabulary size. You want to perform latent semantics analysis (LSA) searching for k latent topics.

- Explain the technique that we reviewed in the course to do LSA using linear algebra.
- Describe how you can select the most dominant topic and, once you have selected that topic; describe how do you find the terms that contribute the most to that topic and the terms that contribute negatively to it.
- How can you encode a new document in the new space with k latent topics?
- In which part of the algorithm, LSA makes sure sure to learn different latent topics?

1) Using SVD we decompose X in 3 matrices $X = USV$

U → D × k colums are left singular vectors
S → k × k diagonal matrix with singular values
V → V × k colums are right singular vectors

k is the number of latent topics

2) most dominant topic has the largest singular value in S

terms that contribute the most: look at V the terms with highest abs values is the most significant contributors to the topic.

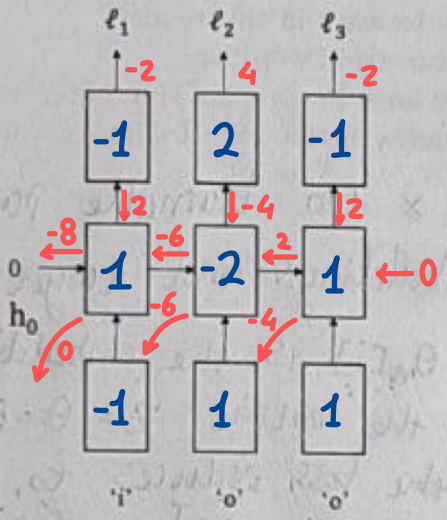
Positive values → contribute positively
Negative values → contribute negatively

3) encode a new document: represent the document as a term vector and project into latent space using V (multiply d by V)

4) LSA ensures different latent topics during SVD. During decomposition it captures the underlying structure in X. Each vector in U and V represents a different latent topic. While S indicate the importance of each latent topic.

We are given a **Elmann RNN** trained at the character level. The input squence of character is shown in Fig. 3. The vocabulary is `['!','o','i']` and the associated char embeddings are `[-2,1,-1]`. The details of the RNN are:

- hidden layer $W_h = -1$, hidden layer input $W_x = -1$, hidden layer classification $W_y = -1$. The activation functions are all identity functions. There are NO biases. The initial hidden state is $h_0 = 0$.
- RNN is trained so that the scalar value at each branch of the RNN after the classification layer is the loss at that level raised to the square. The final loss is thus $\ell = \sum_{i=1}^{3} \ell_i^2$.



$\nabla(W_x \cdot x + W_h \cdot h_{t-1})$

$L = 1 + 4 + 1 = 6$

$\frac{\partial L}{\partial x} = 2X_t$

Figure 3: Elman network considered.

(a) Compute the numerical value of $\frac{\partial \ell}{\partial \text{'o'}}$, that is the partial derivative of the loss wrt to the **char embedding 'o'**, showing all the derivation with a computational graph by breaking it down into forward pass and backward pass. I strongly suggest you rebuild the graph on paper.

Assume that you have a recurrent model (RNN) and you want to give this model the ability to being able to forget what it gets from its previous state. Let assume that you are at state t and you have a "straight through" connection entering the current node, c_{t-1}. The connection is "straight through" so the next state is defined simply as:

$$c_t \doteq c_{t-1}. \tag{1}$$

- Which model we reviewed in class employs a forgetting gating factor?
- You want to give the model the ability to learn to forget part of c_{t-1}. The forget factor should depend only on the the new input x_t at time t. Describe how you can change Eq. (1) so to incorporate the forgetting factor, remembering that c_t is a vector.

The model with a forgetting gating factor is LSTM

To forget we have to add a forget gate $f_t = \sigma(W_f x_t + b_f)$. New state $c_t = f_t \overset{\text{element-wise mult}}{\odot} c_{t-1}$

Consider a transformers network decoder with the learnable cross-attention mechanism. It receives as input word embeddings $\mathbf{X} \in \mathbb{R}^{D \times N}$ where D is the word embedding dimensions and N is the number of tokens and it has to attend to $\mathbf{Y} \in \mathbb{R}^{D \times N}$ embeddings from the encoder. Write the equation of the learnable cross-attention mechanism with a single head with liner algebra, in function of the given input $\mathbf{X}$ and $\mathbf{Y}$, writing down all the parameters that you need to learn and explain what is their role. Pay attention to the fact that matrices size should match.

- In the cross-attention is it necessary that $\mathbf{Y}$ matrix has to have the same number of embeddings N as the ones in the decoder or can have different size?
- You are training a transformer encoder-decoder network with language modeling and you are using plain cross-attention, after one iteration you see that suddenly the loss goes to zero. What is the culprit of the problem? What do you have to change in your model?
- Assume that you have two embedding vectors $\mathbf{x}$ and $\mathbf{y}$ that do not have the same dimensionality. How can you change the self-attention in this case to still make it to work. Note that the two embedding vectors have still different dimensions even after the $\mathbf{W}$ matrices.

$$\text{CrossAttention} = \text{softmax}\left(\frac{Q K^T}{\sqrt{d_k}}\right) V = \text{softmax}\left(\frac{(W_Q X)(W_K Y)^T}{\sqrt{d_k}}\right)(W_V Y)$$

1) No the Y matrix not need to have same number of embeddings. Only dimensions should match

2) An issue could be that weights become too small or large due to scaling of $\sqrt{d_k}$. We have to scale correctly and use layer normalisation

3) You need to project on common dimensional space before attention. Using $x' = W_x x$ and $y' = W_y y$ trasformed vectors. In addition we can add a linear tranformation followed by projection layer

$$\rightarrow \begin{array}{l} x \in \mathbb{R}^{D_x \times N} \\ y \in \mathbb{R}^{D_y \times N} \end{array} \longrightarrow \begin{array}{lll} Q = W_Q x & \text{where} & W_Q \in \mathbb{R}^{d \times D_x} \\ K = W_K y & \text{where} & W_K \in \mathbb{R}^{d \times D_y} \\ V = W_V y & \text{where} & W_V \in \mathbb{R}^{d \times D_y} \end{array}$$

A transformer is used also in the Contrastive Language-Image Pre-training (CLIP) model. Let's assume that you have an image **x** of a scene and need to use CLIP to make a fine-grained classification of an object `<object>` that is in the image. To perform the fine-grained classification, you have available a hierarchy of classes to test where the root is `<object>`, the next level we have `<animals>`, `<plants>`, `<fungi>` and so on and so forth. The categories at the same level of the hierarchy are mutually exclusive. The hierarchy is very deep, every node branches in three with a total depth of 16. You want to perform the fine-grained classification at the deepest level.

- State if you can use CLIP to solve the fine-grained classification.
- If you answered yes then briefly describe which method you can use to get a fine-grained prediction at the deepest level possible, describing an algorithm that can work in practice.
- If you replied no, then say why CLIP cannot do it.

a) Yes Clip is useful, it can comprehend textual description of categories

b) 1. Define the hierarchy of classes in textual form
2. generate embeddings for image x using Clip
3. compute the cosine similarity between image embedding and each class description
4. starting from root "object" compare the similarities and select the child with highest similarity
5. repeat the process until leaf node.