

NLP – Unit 3 (Language Modelling) – END-SEM PYQ Answers

MAY-JUNE 2023

Q1a) Describe the process of building a simple Markov model for predicting the next word in a sentence with the help of example. [6 Marks]

Markov Model for Language / Next-Word Prediction

A Markov model assumes that the probability of the next word depends only on the current state (the immediately preceding word or words). This is called the Markov assumption, and it allows us to build computationally tractable models of language.

Steps to build a simple (first-order / bigram) Markov model:

- Step 1 – Tokenization: Split the raw text corpus into individual word tokens. Each word is a 'state' in the model.
- Step 2 – Count bigram frequencies: For every consecutive pair of words (w_{i-1} , w_i), count how many times that pair occurs in the corpus.
 - Step 3 – Compute transition probabilities: For each word w_{i-1} , divide each bigram count by the unigram count of w_{i-1} .

$$P(w_i | w_{i-1}) = \text{Count}(w_{i-1}, w_i) / \text{Count}(w_{i-1})$$

- Step 4 – Build the transition matrix: Rows represent the current word (state), columns represent the next word. Each cell holds the transition probability.
- Step 5 – Prediction: To predict the next word after a given word, look up its row in the transition matrix and pick the word with the highest probability (or sample according to the distribution).

Example:

Corpus: 'I love NLP. I love coding. I hate bugs.'

- Bigrams: (I,love)×2, (I,hate)×1, (love,NLP)×1, (love,coding)×1, (hate,bugs)×1
- $P(\text{love} | \text{I}) = 2/3 \approx 0.67$, $P(\text{hate} | \text{I}) = 1/3 \approx 0.33$
- $P(\text{NLP} | \text{love}) = 1/2 = 0.50$, $P(\text{coding} | \text{love}) = 1/2 = 0.50$

So given the word 'I', the model predicts 'love' with probability 0.67.

Note: A first-order Markov model is a bigram model. A second-order model (trigram) conditions on the two previous words: $P(w_i | w_{i-2}, w_{i-1})$. Higher-order models capture more context but require exponentially more data.

Q1b) Add-k smoothing bigram problem — $P(\text{'the cat sat on the mat'})$, $k=0.5$, corpus 10000 words, vocab 5000, $\text{Count}(\text{'the cat'})=50$, $\text{Count}(\text{'the'})=1000$, $\text{Count}(\text{'cat'})=100$. [8 Marks]

Add-k (Lidstone) Smoothing — Worked Solution

Add-k smoothing prevents zero probabilities for unseen bigrams by adding a constant k to every bigram count.

$$P_k(w_i | w_{i-1}) = [\text{Count}(w_{i-1}, w_i) + k] / [\text{Count}(w_{i-1}) + k \times V]$$

Given: $k = 0.5$, V (vocabulary size) = 5000

We must estimate $P(\text{'the cat sat on the mat'})$. Treat $\langle s \rangle$ and $\langle /s \rangle$ as boundary markers. We need these bigram probabilities:

- $P(\text{the} \mid \langle s \rangle)$, $P(\text{cat} \mid \text{the})$, $P(\text{sat} \mid \text{cat})$, $P(\text{on} \mid \text{sat})$, $P(\text{the} \mid \text{on})$, $P(\text{mat} \mid \text{the})$, $P(\langle /s \rangle \mid \text{mat})$

Only $P(\text{cat} \mid \text{the})$ is given explicitly. For the rest we assume zero raw counts (unseen bigrams), so only smoothing contributes.

$P(\text{cat} \mid \text{the})$ — given counts:

$$P(\text{cat} \mid \text{the}) = (50 + 0.5) / (1000 + 0.5 \times 5000) = 50.5 / 3500 \approx 0.01443$$

$P(\text{sat} \mid \text{cat})$ — Count('cat sat') assumed 0:

$$P(\text{sat} \mid \text{cat}) = (0 + 0.5) / (100 + 0.5 \times 5000) = 0.5 / 2600 \approx 0.000192$$

For all other unseen bigrams where $\text{Count}(w_{i-1})$ is also unknown, assume same denominator structure. Since exact unigram counts for 'sat', 'on', 'mat', etc. are not given, we estimate them uniformly: total words = 10000, vocab = 5000 \rightarrow average unigram count ≈ 2 .

$$P(\text{unseen} \mid \text{unseen_prev}) \approx 0.5 / (2 + 2500) \approx 0.0002$$

Sentence probability (product of bigram probabilities):

$$P \approx P(\text{the} \mid \langle s \rangle) \times P(\text{cat} \mid \text{the}) \times P(\text{sat} \mid \text{cat}) \times P(\text{on} \mid \text{sat}) \times P(\text{the} \mid \text{on}) \times P(\text{mat} \mid \text{the}) \times P(\langle /s \rangle \mid \text{mat})$$

Using approximate values:

$$P \approx 0.0002 \times 0.01443 \times 0.000192 \times 0.0002 \times 0.0002 \times 0.01443 \times 0.0002$$

This is an extremely small number (order 10^{-20}), which is expected for a 7-word sentence under a bigram model. In practice, log probabilities are used to avoid underflow.

Note: The key insight is the smoothing formula, not the final tiny product. Examiners typically award marks for correctly applying $P_k = (\text{Count} + k) / (\text{Unigram_count} + k \times V)$ for each bigram and identifying which counts are given vs assumed zero.

Q1c) Write a short note on Latent Semantic Analysis (LSA). [4 Marks]

Latent Semantic Analysis (LSA)

LSA is an unsupervised technique that discovers hidden (latent) semantic relationships between words and documents by analysing patterns of co-occurrence in a large corpus.

- Core idea: Words that occur in similar contexts tend to have similar meanings, even if they never appear together.
- Step 1 – Build the term-document matrix: Rows = terms, columns = documents, cell values = TF-IDF scores.
- Step 2 – Apply Singular Value Decomposition (SVD): Decompose the matrix M as $M = U \times \Sigma \times V^T$, where U contains term vectors, Σ the singular values (importance), and V document vectors.
- Step 3 – Truncate to k dimensions: Keep only the top- k singular values, reducing noise and capturing the most important latent semantic dimensions.
- Step 4 – Semantic similarity: Compute cosine similarity between term or document vectors in the reduced k -dimensional space.

Applications: Document retrieval, synonymy detection, text classification, and plagiarism detection.

Note: LSA/LSI is computationally efficient and does not require labelled data. Its main limitation is that it cannot handle polysemy (a word with multiple meanings gets a single vector).

Q2a) What are generative models of language, and how do they differ from discriminative models? [4 Marks]

Generative vs Discriminative Models

A generative model learns the joint probability distribution $P(X, Y)$ of both the input X and the label Y . It can generate new samples from the learned distribution.

- Example: N-gram language models, Hidden Markov Models (HMMs), Naïve Bayes classifiers, GPT.
- Key property: Can answer 'What is the probability of this sentence?' by computing $P(w_1, w_2, \dots, w_n)$.

A discriminative model learns only the conditional probability $P(Y | X)$ — the boundary between classes — without modelling how the data was generated.

- Example: Logistic regression, Support Vector Machines, Conditional Random Fields (CRF), BERT for classification.
- Key property: Typically higher accuracy on classification tasks since it focuses entirely on the decision boundary.

Key difference: Generative models model the full data distribution; discriminative models model only the mapping from input to label. Generative models can generate text; discriminative models generally cannot.

Note: In NLP, both types matter. BERT is discriminative (it predicts masked tokens given context), while GPT is generative (it predicts the next token autoregressively).

Q2b) Calculate the TF-IDF score of 'Term 1' in 'Document 1' given the document-term matrix. [6 Marks]

TF-IDF Calculation — Worked Example

Given document-term matrix:

Term	Doc 1	Doc 2	Doc 3
Term 1	10 ★	5	0
Term 2	2	0	8
Term 3	1	3	6

Step 1 – Term Frequency (TF) of Term 1 in Document 1:

$TF = (\text{Count of Term 1 in Doc 1}) / (\text{Total terms in Doc 1})$

Total terms in Doc 1 = $10 + 2 + 1 = 13$

$$TF(\text{Term1}, \text{Doc1}) = 10 / 13 \approx 0.769$$

Step 2 – Inverse Document Frequency (IDF) of Term 1:

Term 1 appears in Doc1 (count=10) and Doc2 (count=5) → Document Frequency (DF) = 2

Total documents $N = 3$

$$IDF(\text{Term1}) = \log(N / DF) = \log(3 / 2) = \log(1.5) \approx 0.405 \text{ [natural log]}$$

or using log base 10: $\log_{10}(3/2) \approx 0.176$

Step 3 – TF-IDF:

$$TF\text{-}IDF(\text{Term1}, \text{Doc1}) = TF \times IDF = 0.769 \times 0.405 \approx 0.311 \text{ (natural log)}$$

Note: The exact value depends on the logarithm base used. Both log base e (≈ 0.311) and log base 10 (≈ 0.135) are acceptable; state which one you use.

Q2c) Describe the Latent Dirichlet Allocation (LDA) algorithm and how it is used for topic modeling. [8 Marks]

Latent Dirichlet Allocation (LDA)

LDA is a generative probabilistic model that treats each document as a mixture of topics, and each topic as a probability distribution over words.

Core Assumptions:

- Each document is generated by choosing a distribution over K topics.
- Each word in the document is generated by first choosing a topic from that distribution, then choosing a word from that topic's word distribution.

Key Components:

- α (alpha): Dirichlet prior on the per-document topic distribution. Low $\alpha \rightarrow$ documents focused on fewer topics.
- β (beta): Dirichlet prior on the per-topic word distribution. Low $\beta \rightarrow$ topics concentrated on fewer words.
- θ_d : Topic distribution for document d (sampled from $\text{Dirichlet}(\alpha)$).
- ϕ_k : Word distribution for topic k (sampled from $\text{Dirichlet}(\beta)$).
- $z_{\{d,n\}}$: Topic assignment for the n -th word in document d .
- $w_{\{d,n\}}$: The actual observed word.

Generative Process (for each document):

1. Sample document-topic distribution $\theta_d \sim \text{Dirichlet}(\alpha)$
2. For each word position n in the document:
 - a. Sample a topic $z \sim \text{Multinomial}(\theta_d)$
 - b. Sample a word $w \sim \text{Multinomial}(\phi_z)$

Inference (learning from data):

Since the topic assignments z are hidden (latent), we use approximate inference. Two common methods are:

- Variational Bayes (VB): Optimises a lower bound on the log-likelihood.
- Collapsed Gibbs Sampling: Iteratively samples topic assignments $z_{\{d,n\}}$ for each word while integrating out θ and ϕ .

Gibbs Sampling update rule:

$$P(z_{\{d,n\}}=k \mid \text{rest}) \propto (n_{\{d,k\}} + \alpha) \times (n_{\{k,w\}} + \beta) / (n_k + V \times \beta)$$

where $n_{\{d,k\}}$ = count of words assigned to topic k in doc d , $n_{\{k,w\}}$ = count of word w in topic k .

Output:

- K topic-word distributions: Each topic is a ranked list of high-probability words (e.g., Topic 1: cricket, bat, pitch, match \rightarrow 'Sports').
- Per-document topic proportions: Shows what percentage of each document belongs to each topic.

Applications: Document clustering, trend discovery in news, recommendation systems, social media analysis.

Note: LDA differs from LSA: LDA is probabilistic (Bayesian) and interpretable (topics have word distributions), whereas LSA is algebraic (uses SVD) and less interpretable. LDA is generally preferred for topic modeling when interpretability matters.

NOV-DEC 2023

Q1a) What are generative models of language? Explain any one model in detail. [4 Marks]

[REPEATED] Q2a of May-June 2023 — Generative vs Discriminative Models

See May-June 2023 Q2a for the definition of generative models. Below is a detailed explanation of the N-gram generative model (unique content for this question):

Detailed Example: Bigram Language Model (N-gram Generative Model)

A bigram model assumes $P(w_i | w_1 \dots w_{i-1}) \approx P(w_i | w_{i-1})$. The sentence probability is:

$$P(w_1, w_2, \dots, w_n) = \prod P(w_i | w_{i-1})$$

Training: Count all consecutive word pairs in corpus and compute MLE estimates.

Generation: Start with <s>, repeatedly sample the next word from $P(w_i | w_{i-1})$ until </s> is sampled.

Q1b) Find the Bigram probability of 'students are from Pune' using the given training corpus. [8 Marks]

Bigram Probability Computation — Worked Solution

Training Corpus:

- <s> I am from Pune </s>
- <s> I am a teacher </s>
- <s> students are good and are from various cities </s>
- <s> students from Pune do engineering </s>

Test sentence: <s> students are from Pune </s>

We need: $P(\text{students}|\text{<s>}) \times P(\text{are}|\text{students}) \times P(\text{from}|\text{are}) \times P(\text{Pune}|\text{from}) \times P(\text{</s>}|\text{Pune})$

Count each bigram and the relevant unigrams from the corpus:

- $\text{Count}(\text{<s>}) = 4$ (four sentences)
- $\text{Count}(\text{<s>, students}) = 2$ [sentences 3,4]
- $\text{Count}(\text{students}) = 2$
- $\text{Count}(\text{students, are}) = 1$ [sentence 3]
- $\text{Count}(\text{are}) = 2$ ['are good' and 'are from' in sentence 3]
- $\text{Count}(\text{are, from}) = 1$ [sentence 3: 'are from']
- $\text{Count}(\text{from}) = 3$ ['from Pune' $\times 2$, 'from various' $\times 1$]
- $\text{Count}(\text{from, Pune}) = 2$ [sentences 1 and 4... wait: sentence 1 has 'from Pune', sentence 4 has 'from Pune' — also check sentence 3: 'are from various' \rightarrow 'from various' \rightarrow $\text{Count}(\text{from, Pune})=2$]
- $\text{Count}(\text{Pune}) = 2$
- $\text{Count}(\text{Pune, </s>}) = 1$ [only sentence 1 ends with 'Pune </s>'; sentence 4 continues 'do engineering']

Bigram probabilities (MLE, no smoothing):

$$P(\text{students} \mid \langle s \rangle) = 2/4 = 0.5$$

$$P(\text{are} \mid \text{students}) = 1/2 = 0.5$$

$$P(\text{from} \mid \text{are}) = 1/2 = 0.5$$

$$P(\text{Pune} \mid \text{from}) = 2/3 \approx 0.667$$

$$P(\langle s \rangle \mid \text{Pune}) = 1/2 = 0.5$$

Sentence probability:

$$P = 0.5 \times 0.5 \times 0.5 \times 0.667 \times 0.5 = 0.5^4 \times 0.667 \approx 0.0417$$

Note: Always include the start $\langle s \rangle$ and end $\langle s \rangle$ markers when computing bigram probabilities for full sentences. These are counted as regular tokens.

[REPEATED] Q1c) Explain in detail Latent Semantic Analysis for topic modelling (LSA). [6 Marks]

See May-June 2023 Q1c for the full LSA answer.

Q2a) Write short note on BERT. [4 Marks]

BERT — Bidirectional Encoder Representations from Transformers

BERT is a pre-trained deep learning model introduced by Google in 2018 (Devlin et al.). It produces contextualized word representations, meaning the same word gets a different vector depending on its surrounding context.

Key features:

- Bidirectional: Unlike traditional LMs that read text left-to-right, BERT reads in both directions simultaneously using the Transformer encoder.
- Pre-training tasks: (1) Masked Language Modeling (MLM) — randomly mask 15% of tokens and predict them; (2) Next Sentence Prediction (NSP) — predict whether two sentences are consecutive.
- Fine-tuning: After pre-training on a large corpus (BooksCorpus + Wikipedia), BERT is fine-tuned with a small task-specific layer for downstream tasks.
- Tokenization: Uses WordPiece tokenization, which splits rare words into subword units.

Architecture:

- BERT-Base: 12 Transformer encoder layers, 768 hidden units, 12 attention heads, 110M parameters.
- BERT-Large: 24 layers, 1024 hidden units, 16 heads, 340M parameters.

Applications: Named Entity Recognition, Question Answering, Sentiment Analysis, Text Classification, Coreference Resolution.

Note: BERT is a discriminative model — it is not designed to generate text. It excels at understanding tasks. GPT-series models are the generative counterparts.

[REPEATED] Q2b) Calculate TF-IDF of Term 1 in Document 1 (same matrix). [6 Marks]

See May-June 2023 Q2b for the complete TF-IDF calculation.

[REPEATED] Q2c) Describe the Latent Dirichlet Allocation (LDA) algorithm. [8 Marks]

See May-June 2023 Q2c for the full LDA explanation.

MAY-JUNE 2024

Q1a) What are generative models of language, and how do they differ from discriminative models? Provide an example of a generative model. [9 Marks]

[REPEATED] Q2a of May-June 2023 — Generative vs Discriminative Models

See May-June 2023 Q2a for the core distinction. The additional 9-mark depth requires an extended example:

Extended Example: N-gram as a Generative Model

A bigram language model $P(w_1 \dots w_n) = \prod P(w_i | w_{i-1})$ is generative because:

- It assigns a probability to every possible sentence.
- You can sample from it: start with $\langle s \rangle$, keep sampling next words, stop at $\langle /s \rangle$.
- It can be used to compute perplexity, to do speech recognition decoding, or machine translation scoring.

Contrast with CRF (discriminative): A CRF for POS tagging models $P(\text{tags} | \text{words})$ and cannot generate new sentences — it can only label a given sentence.

Modern generative models like GPT use the same chain rule but with deep neural networks instead of simple count ratios.

Q1b) Define Latent Dirichlet Allocation (LDA) and explain its key components. [9 Marks]

[REPEATED] Q2c of May-June 2023 — LDA for Topic Modeling

See May-June 2023 Q2c for the complete LDA explanation including the generative process, Dirichlet priors, Gibbs sampling, and applications.

Q2a) Describe contextualized representations (BERT) — advantages and disadvantages. [10 Marks]

[REPEATED] Q2a of Nov-Dec 2023 — BERT short note (extended here)

Contextualized Representations — Extended Analysis

Traditional word embeddings (Word2Vec, GloVe) assign a single static vector to each word regardless of context. For example, 'bank' has one vector whether it means a riverbank or a financial institution.

Contextualized representations solve this by producing a different vector for each occurrence of a word depending on its surrounding tokens.

BERT as the canonical example:

- Uses self-attention: every token attends to all other tokens in the sequence, building a rich context-aware representation.
- Output vectors: the [CLS] token captures sentence-level meaning; individual token vectors carry word-in-context meaning.

Advantages:

- Handles polysemy — 'bank' near 'river' gets a different vector than 'bank' near 'loan'.
- State-of-the-art results on virtually all NLP benchmarks when fine-tuned.
- Pre-training on large corpora transfers knowledge to low-resource tasks.
- Bidirectionality captures left and right context simultaneously.

Disadvantages:

- Computational cost: BERT-Base has 110M parameters; inference is slow on CPU.
- Memory intensive: requires GPUs for efficient training and fine-tuning.
- Max sequence length: limited to 512 tokens (standard BERT), problematic for long documents.
- Black-box: difficult to interpret what the attention heads have learned.
- Not generative: cannot produce new text directly.

[REPEATED] Q2b) Add-k smoothing bigram problem. [8 Marks]

Identical to May-June 2023 Q1b. See that section for the complete worked solution.

NOV-DEC 2025

Q1a) Given corpus S1:'language models learn patterns', S2:'models learn from data', S3:'data helps improve language models' — (1) Construct bigram model, (2) Compute smoothed bigram P('language models learn from data') using Add-1, (3) Show all intermediate steps. [9 Marks]

Step 1 — Tokenisation and Vocabulary

After adding sentence-boundary markers <s> and </s>, the tokenised sentences are:

- <s> language models learn patterns </s>
- <s> models learn from data </s>
- <s> data helps improve language models </s>

Unique vocabulary (V): { <s>, </s>, language, models, learn, patterns, from, data, helps, improve } → V = 10 words.

Step 2 — Unigram Counts

Count every token across all three sentences (including boundary markers):

Token	Count
<s>	3
</s>	3
language	2
models	3
learn	2
patterns	1
from	1
data	2

helps	1
improve	1

Step 3 — Bigram Counts (construct the bigram model)

All consecutive token pairs from the three sentences:

Bigram (w_{i-1} , w_i)	Count
(<s>, language)	2 [S1, S3]
(<s>, models)	1 [S2]
(<s>, data)	1 [S3 — wait, S3 starts with 'data']
(language, models)	2 [S1: lang→mod; S3: lang→mod]
(models, learn)	1 [S1: mod→learn]
(models, </s>)	1 [S3 ends: mod→</s>]
(learn, patterns)	1 [S1]
(learn, from)	1 [S2]
(patterns, </s>)	1 [S1]
(from, data)	1 [S2]
(data, </s>)	1 [S2]
(data, helps)	1 [S3]
(helps, improve)	1 [S3]
(improve, language)	1 [S3]
(language, models)	2 [confirmed above]

Corrected bigram table (S3 = <s> data helps improve language models </s>):

- (<s>, language) = 2 [S1, S3? — No: S3 starts with 'data', not 'language']
- Correction: (<s>, language) = 1 [S1 only]; (<s>, models) = 1 [S2]; (<s>, data) = 1 [S3]
- (language, models) = 2 [S1 and S3]
- (models, learn) = 1 [S1]; (models, </s>) = 1 [S3 ends '...language models </s>']
- (learn, patterns) = 1 [S1]; (learn, from) = 1 [S2]
- (patterns, </s>) = 1; (from, data) = 1; (data, </s>) = 1 [S2]
- (data, helps) = 1; (helps, improve) = 1; (improve, language) = 1

Step 4 — Add-1 (Laplace) Smoothing Formula

$$P_{\text{add1}}(w_i | w_{i-1}) = [\text{Count}(w_{i-1}, w_i) + 1] / [\text{Count}(w_{i-1}) + V]$$

where $V = 10$ (vocabulary size, including boundary markers).

Step 5 — Compute P ('language models learn from data')

Test sentence with boundaries: <s> language models learn from data </s>

Required bigram probabilities:

- $P(\text{language} \mid \langle s \rangle) = (1+1) / (3+10) = 2/13 \approx 0.1538$
- $P(\text{models} \mid \text{language}) = (2+1) / (2+10) = 3/12 = 0.25$
- $P(\text{learn} \mid \text{models}) = (1+1) / (3+10) = 2/13 \approx 0.1538$
- $P(\text{from} \mid \text{learn}) = (1+1) / (2+10) = 2/12 \approx 0.1667$
- $P(\text{data} \mid \text{from}) = (1+1) / (1+10) = 2/11 \approx 0.1818$
- $P(\langle /s \rangle \mid \text{data}) = (1+1) / (2+10) = 2/12 \approx 0.1667$

Sentence probability (product of all bigram probabilities):

$$P = (2/13) \times (3/12) \times (2/13) \times (2/12) \times (2/11) \times (2/12)$$

$$P \approx 0.1538 \times 0.25 \times 0.1538 \times 0.1667 \times 0.1818 \times 0.1667$$

$$P \approx 2.69 \times 10^{-5}$$

In practice, we use log probabilities to avoid numerical underflow:

$$\log P = \log(2/13) + \log(3/12) + \log(2/13) + \log(2/12) + \log(2/11) + \log(2/12)$$

$$\log P \approx -1.874 + (-1.386) + (-1.874) + (-1.792) + (-1.705) + (-1.792) \approx -10.42 \text{ [natural log]}$$

Note: Always show the vocabulary size V explicitly in the denominator — that is the key step examiners check. The corpus here has 3 sentences \times (4 or 5 content words + 2 boundary tokens) \approx 21 total tokens, but V counts unique types (10), not tokens.

Q1b) Explain probabilistic language modeling using Markov assumptions. Construct a trigram model and compute $P(\text{sentence})$ using MLE. [9 Marks]

Probabilistic Language Modeling & the Markov Assumption

A language model assigns a probability to any sequence of words w_1, w_2, \dots, w_n . Using the chain rule of probability:

$$P(w_1 \dots w_n) = P(w_1) \times P(w_2 \mid w_1) \times P(w_3 \mid w_1, w_2) \times \dots \times P(w_n \mid w_1 \dots w_{n-1})$$

The problem with this exact formulation is data sparsity — for long histories, the count of the exact context $w_1 \dots w_{n-1}$ will be zero in any finite corpus. The Markov assumption solves this by approximating the full history with only the last k words:

$$P(w_i \mid w_1 \dots w_{i-1}) \approx P(w_i \mid w_{i-k} \dots w_{i-1})$$

For a trigram model, $k = 2$, so we condition on the two preceding words:

$$P(w_i \mid w_1 \dots w_{i-1}) \approx P(w_i \mid w_{i-2}, w_{i-1})$$

The full sentence probability becomes:

$$P(w_1 \dots w_n) = \prod P(w_i \mid w_{i-2}, w_{i-1})$$

MLE for Trigram Parameters

Maximum Likelihood Estimation directly uses corpus counts:

$$P_{\text{MLE}}(w_i \mid w_{i-2}, w_{i-1}) = \text{Count}(w_{i-2}, w_{i-1}, w_i) / \text{Count}(w_{i-2}, w_{i-1})$$

Worked Example

Corpus (same three sentences from Q1a, with double start markers for trigrams):

- $\langle s \rangle \langle s \rangle$ language models learn patterns $\langle /s \rangle$
- $\langle s \rangle \langle s \rangle$ models learn from data $\langle /s \rangle$

- $\langle s \rangle \langle s \rangle$ data helps improve language models $\langle /s \rangle$

Let us compute $P(\text{'language models learn'})$ using MLE trigrams:

- $P(\text{language} \mid \langle s \rangle, \langle s \rangle) = \text{Count}(\langle s \rangle, \langle s \rangle, \text{language}) / \text{Count}(\langle s \rangle, \langle s \rangle) = 1/3 \approx 0.333$
- $P(\text{models} \mid \langle s \rangle, \text{language}) = \text{Count}(\langle s \rangle, \text{language}, \text{models}) / \text{Count}(\langle s \rangle, \text{language}) = 1/1 = 1.0$
- $P(\text{learn} \mid \text{language}, \text{models}) = \text{Count}(\text{language}, \text{models}, \text{learn}) / \text{Count}(\text{language}, \text{models})$

'language models' appears in S1 (followed by 'learn') and in S3 (followed by ' $\langle /s \rangle$ '). So $\text{Count}(\text{language}, \text{models}) = 2$, $\text{Count}(\text{language}, \text{models}, \text{learn}) = 1$.

- $P(\text{learn} \mid \text{language}, \text{models}) = 1/2 = 0.5$

Trigram sentence probability:

$$P(\text{'language models learn'}) = (1/3) \times 1.0 \times 0.5 = 1/6 \approx 0.167$$

Advantages of trigrams over bigrams: they capture two-word context, so phrases like 'New York' followed by 'City' are modelled correctly. Disadvantage: the parameter space grows as V^3 , making data sparsity worse — smoothing becomes even more critical.

Note: The Markov assumption is the single most important concept in classical NLP language modeling. Every n -gram model, Hidden Markov Model, and even modern Transformers (via positional encoding) can be traced back to some form of this assumption about how far back in history to look.

Q2a) Explain how NMF differs from LDA in topic modeling. Use a document-term matrix to show NMF factorization. [9 Marks]

Non-Negative Matrix Factorization (NMF) vs LDA — Conceptual Difference

Both NMF and LDA are techniques for discovering latent topics in a collection of documents, but they approach the problem from completely different mathematical foundations.

LDA is a generative probabilistic Bayesian model. It assumes that documents were created by a random process: first pick a topic mixture from a Dirichlet distribution, then for each word pick a topic, then pick a word from that topic's distribution. Learning means inferring what latent topics could have generated the observed words. LDA's output is a probability distribution — topics sum to 1 within a document, and word distributions sum to 1 within a topic.

NMF is an algebraic matrix decomposition technique. It takes a document-term matrix V (n documents $\times m$ terms, with non-negative entries such as TF-IDF weights) and finds two non-negative matrices W and H such that $V \approx W \times H$. The non-negativity constraint is what gives NMF its power for interpretability — because you cannot subtract topics from a document, the parts discovered tend to be additive and physically meaningful.

Formal NMF decomposition:

$$V \approx W \times H$$

$$(n \times m) \quad (n \times k) \quad (k \times m)$$

where k is the number of topics (chosen by the user, analogous to K in LDA). $W[i, :]$ gives the topic weights for document i . $H[j, :]$ gives the term weights for topic j .

Worked Example with a 3×5 Document-Term Matrix

Suppose we have 3 documents and 5 terms: {neural, networks, deep, learning, models}

V	neural	networks	deep	learning	models
D1	3	2	0	0	1

D2	1	0	4	3	1
D3	2	1	1	1	3

With $k = 2$ topics, NMF finds (conceptually, via iterative multiplicative updates):

- W (document-topic matrix): D1 is mostly Topic 0; D2 is mostly Topic 1; D3 is mixed.
- H (topic-term matrix): Topic 0 has high weights on {neural, networks, models} → 'Architecture' topic. Topic 1 has high weights on {deep, learning} → 'Training' topic.

Interpretation: A document is reconstructed as a weighted sum of topics. There is no concept of 'subtracting a topic', which makes NMF factors directly interpretable as parts-of-whole.

Key Differences Summary:

Aspect	NMF	LDA
Foundation	Algebraic matrix factorization	Bayesian generative probabilistic model
Constraint	Non-negativity ($W, H \geq 0$)	Probabilistic (Dirichlet priors)
Output	Real-valued topic weights	Probability distributions
Inference	Multiplicative update rules / ALS	Variational Bayes or Gibbs sampling
Sparsity	Soft; depends on initialization	Controlled via Dirichlet α, β
Speed	Generally faster	Slower, more iterations to converge
Interpretability	High (additive parts)	High (probability semantics)
Best for	Short texts, images, signal data	Longer documents, need priors

Note: NMF is actually older than LDA (Lee & Seung 1999 vs Blei et al. 2003) but gained popularity in NLP more recently because it's faster and gives comparably interpretable results for many practical tasks. Scikit-learn implements both with a nearly identical API.

Q2b) Compute TF-IDF of 'neural' in D1:'neural networks are powerful', D2:'deep learning powers neural models', D3:'networks and models are important'. Show DF, IDF, and interpret. [9 Marks]

Step 1 — Term Frequency (TF) of 'neural' in each document

TF is computed as the count of the target term divided by the total number of words in that document (after removing punctuation):

- D1 = {neural, networks, are, powerful} → 4 words. $TF(\text{neural}, D1) = 1/4 = 0.25$
- D2 = {deep, learning, powers, neural, models} → 5 words. $TF(\text{neural}, D2) = 1/5 = 0.20$
- D3 = {networks, and, models, are, important} → 5 words. $TF(\text{neural}, D3) = 0/5 = 0.00$

Step 2 — Document Frequency (DF) and IDF

Document Frequency (DF) is the number of documents in which the term appears at least once. 'neural' appears in D1 and D2, but not D3, so:

$$DF(\text{neural}) = 2$$

Inverse Document Frequency (IDF) is computed as:

$$\text{IDF}(\text{neural}) = \log(N / \text{DF}) = \log(3 / 2) \approx 0.405 \text{ [natural log]}$$

Using log base 10: $\text{IDF} = \log_{10}(3/2) \approx 0.176$. (Both are acceptable — state which you use.)

Step 3 — TF-IDF Scores

$$\text{TF-IDF}(\text{neural}, D1) = 0.25 \times 0.405 \approx 0.101$$

$$\text{TF-IDF}(\text{neural}, D2) = 0.20 \times 0.405 \approx 0.081$$

$$\text{TF-IDF}(\text{neural}, D3) = 0.00 \times 0.405 = 0.000$$

Step 4 — Interpretation

'neural' is most significant in D1 (TF-IDF ≈ 0.101) because it makes up a larger fraction of that document (1 out of 4 words) compared to D2 (1 out of 5 words). It is entirely absent from D3 and therefore carries no significance there. The IDF value of 0.405 is moderate — 'neural' is not a rare term (it appears in 2 out of 3 documents), so the IDF slightly downweights it compared to a term that appeared in only one document.

Note: If 'neural' had appeared in all three documents, $\text{IDF} = \log(3/3) = 0$, and the TF-IDF would be zero for all documents — meaning the term is too common to discriminate between documents. This is TF-IDF's elegant way of suppressing function words and domain-ubiquitous terms automatically.

MAY-JUN 2025

Q1a) Explain the concept of log-linear models in NLP. How do they work, and what are common applications? [6 Marks]

Log-Linear Models in NLP

A log-linear model (also called a Maximum Entropy or MaxEnt model) is a discriminative probabilistic model that represents the conditional probability of a label y given input x as a softmax over a weighted sum of feature functions. The name 'log-linear' comes from the fact that the log of the probability is a linear function of the feature weights:

$$P(y | x; w) = \exp(w \cdot f(x, y)) / \sum_{y'} \exp(w \cdot f(x, y'))$$

where $f(x, y)$ is a feature vector — a vector of real-valued functions (feature functions) that capture properties of the input x and candidate label y , and w is the weight vector learned during training.

How They Work:

- Feature extraction: For a given input (e.g., a sentence) and candidate label (e.g., a POS tag), the feature function $f(x, y)$ extracts informative binary or real-valued properties — e.g., [word ends in -ing AND label is VBG] or [previous word is THE AND label is NN].
- Parameterisation: Each feature has an associated weight. A large positive weight means the feature strongly supports the label; a large negative weight means it strongly opposes it.
- Softmax normalisation: The sum over all candidate labels in the denominator (the partition function) ensures the probabilities sum to one.
- Training via Maximum Entropy: Weights are learned by maximising the conditional log-likelihood of the training data, which is equivalent to finding the distribution that matches the empirical feature expectations while being as uniform (maximum entropy) as possible. L-BFGS or gradient descent is used to optimise.

Key NLP Applications:

- Part-of-Speech Tagging: MaxEnt taggers (e.g., the NLTK MaxEnt POS tagger) outperformed

rule-based taggers and were state-of-the-art before neural networks.

- Text Classification: Log-linear models with bag-of-words features are equivalent to multinomial logistic regression, widely used for sentiment analysis and spam detection.
- Named Entity Recognition: Used as sequence labellers with contextual word-shape and surrounding-token features.
- Language Modeling: Log-linear language models (e.g., Rosenfeld's Maximum Entropy LM) allow arbitrary features of the history, unlike n-gram models which only look at the last n-1 words.
- Machine Translation: Log-linear models combine multiple feature functions (translation model, language model, word penalty, distortion penalty) in phrase-based SMT decoders.

Note: Log-linear models are the conceptual bridge between classical n-gram models (which use only count-based features) and modern neural networks (which learn feature representations end-to-end). Understanding them deeply makes the leap to neural NLP much more natural.

Q1b) Explain doc2vec and its use in generating document embeddings. Discuss differences between doc2vec and word2vec. [8 Marks]

Word2Vec — Recap

Word2Vec (Mikolov et al., 2013) trains a shallow neural network to produce a fixed-dimensional dense vector for each word such that words appearing in similar contexts end up with similar vectors. It uses one of two architectures: CBOW (predict the centre word from surrounding context words) or Skip-gram (predict surrounding context words from the centre word). The result is a static embedding — every word has one vector regardless of context.

The limitation relevant here is that Word2Vec produces word-level vectors. To get a document vector, you might average the word vectors, but this loses word order and document-level semantics.

Doc2Vec — Paragraph Vectors

Doc2Vec (Le & Mikolov, 2014), also known as the Paragraph Vector model, extends Word2Vec by adding a paragraph token (a unique document ID vector) to the context window during training. This document vector is trained simultaneously with the word vectors and learns to capture the meaning of the whole document.

Two architectures:

- PV-DM (Distributed Memory, analogous to CBOW): Predict the centre word from a window of surrounding words AND the paragraph vector. The paragraph vector acts as a 'memory' of the document's topic.
- PV-DBOW (Distributed Bag of Words, analogous to Skip-gram): Ignore the surrounding words entirely and train the paragraph vector to predict random words sampled from the document. Simpler and often equally effective.

Inference for a new document: Fix the trained word vectors and train a new paragraph vector (with gradient descent) for the unseen document. This 'inference' step is what makes doc2vec slightly more cumbersome than sentence-transformer approaches.

Comparison: doc2vec vs word2vec

Aspect	Word2Vec	Doc2Vec
Output unit	One vector per word	One vector per document (+ word vectors)
Granularity	Word-level	Document-level (paragraph-level)
Context	Local window of words	Entire document + local window
Word order	Ignored (bag-of-words window)	Partially captured via memory vector
Inference	Lookup in embedding table	Must re-run gradient descent for new docs
Applications	Semantic similarity of words, clustering	Document classification, doc similarity, IR
Polysemy	One vector = all senses (limitation)	Doc vector averages over all senses in doc

Note: Modern practice has largely moved to sentence-transformers (BERT-based models like all-MiniLM) for document embeddings, as they produce contextualised vectors. However, doc2vec remains relevant for large corpora where transformer inference is too slow, and Gensim provides an efficient doc2vec implementation.

Q1c) Explain Non-Negative Matrix Factorization (NMF) in the context of topic modeling. [4 Marks]

[REPEATED] Q2a of Nov-Dec 2025 covers NMF in detail with a worked example. → See: Nov-Dec 2025 Q2a in this document

Summary for 4-mark answer: NMF factorizes the document-term matrix V ($n \times m$) into two non-negative matrices W ($n \times k$, document-topic) and H ($k \times m$, topic-term) such that $V \approx W \times H$. Each row of H defines a topic as a weighted combination of terms. The non-negativity constraint ensures topics are additive and interpretable. The optimal W and H are found by minimising the Frobenius norm $\|V - WH\|^2$ using multiplicative update rules. Applications include document clustering, feature extraction, and recommender systems.

[REPEATED] Q2a) Markov model for language generation (training + using) [6 Marks] → See: May-Jun 2023 Q1a (Markov model for next-word prediction) and Nov-Dec 2025 Q1b (Markov + MLE) in this document

The unique angle here is 'language generation' (not just prediction). After training the bigram transition matrix (as in May-Jun 2023 Q1a), generation works by: starting at $\langle s \rangle$, sampling the next word proportionally to its transition probabilities, then using that word as the new state, continuing until $\langle /s \rangle$ is sampled. This is a random walk on the Markov chain.

[REPEATED] Q2b) Explain LSA and how it identifies relationships between words. [8 Marks] → See: May-Jun 2023 Q1c (LSA short note, 4 marks) — use that answer expanded with the SVD decomposition detail for 8 marks

Extended 8-mark points beyond the 2023 answer: (1) SVD produces three matrices U (term-concept), Σ (singular values, importance of each latent dimension), V^T (concept-document). (2) After truncation to k dimensions, terms close in the reduced space are semantically related even if they never co-occur directly — this captures synonymy. (3) LSA struggles with polysemy because one word maps to one vector regardless of context. (4) Evaluation: LSA-derived semantic similarity correlates well with human word similarity judgements (Miller-Charles dataset).

[REPEATED] Q2c) Write a short note on TF-IDF representation. [4 Marks] → See: May-Jun 2023 Q2b (TF-IDF calculation, 6 marks) — the formula and concept are fully explained there

4-mark summary: $TF\text{-}IDF = TF(t,d) \times IDF(t)$. TF rewards terms that appear frequently in a document. IDF penalises terms that appear in many documents (common words like 'the' get near-zero IDF). The product captures terms that are both frequent in a document AND rare across the corpus — these are the most discriminative terms for that document. It is the standard term-weighting scheme for VSM-based information retrieval and the baseline for document classification feature extraction.

Additional Reference Notes for Unit 3

N-gram Evaluation: Perplexity

Perplexity (PP) measures how well a language model predicts a test corpus. Lower is better.

$$PP(W) = P(w_1, w_2, \dots, w_N)^{-1/N}$$

A perplexity of K means the model is as confused as if it had to choose uniformly from K words at every step.

Word2Vec

Word2Vec (Mikolov et al., 2013) learns static word embeddings using one of two architectures: CBOW (predict target word from context words) or Skip-gram (predict context words from target word). Words with similar meanings cluster together in the learned vector space.

Smoothing Methods Summary

Method	Formula / Idea	Notes
Add-1 (Laplace)	$(C+1)/(N+V)$	Simple but over-smooths
Add-k (Lidstone)	$(C+k)/(N+kV)$, $0 < k \leq 1$	Generalised Laplace
Kneser-Ney	Absolute discounting + backoff	State-of-the-art for n-grams