

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

May–June 2023

Q1(a) Markov Model for Predicting Next Word

[6 Marks]

- **Definition:** A Markov model predicts the probability of the next word based only on the preceding word(s) — the Markov Assumption. It is the foundation of N-gram language models.
- **Markov Assumption (1st order / Bigram):** $P(w_n | w_1, w_2, \dots, w_{n-1}) \approx P(w_n | w_{n-1})$
- **Higher-order Markov:** Trigram uses 2 previous words: $P(w_n | w_{n-2}, w_{n-1})$. More context but exponentially more parameters needed.
- **Steps to Build a Bigram Markov Language Model:**
 - Step 1 – Collect corpus and tokenize into individual word tokens
 - Step 2 – Add <s> (start) and </s> (end) sentence boundary tokens to each sentence
 - Step 3 – Count all bigram occurrences: $C(w_{i-1}, w_i)$ for every consecutive word pair
 - Step 4 – Count all unigram occurrences: $C(w_{i-1})$ for every word
 - Step 5 – Compute MLE transition probability: $P(w_i | w_{i-1}) = C(w_{i-1}, w_i) / C(w_{i-1})$
 - Step 6 – Represent as a Transition Matrix (rows = current state, cols = next state) or State Diagram
- **Worked Example:** Corpus = 'the cat sat on the mat'
 - Bigrams: (the, cat), (cat, sat), (sat, on), (on, the), (the, mat)
 - $C(\text{the}) = 2$, $C(\text{cat}) = 1$, $C(\text{sat}) = 1$, $C(\text{on}) = 1$
 - $P(\text{cat} | \text{the}) = 1/2 = 0.50$ $P(\text{mat} | \text{the}) = 1/2 = 0.50$
 - $P(\text{sat} | \text{cat}) = 1/1 = 1.00$ $P(\text{on} | \text{sat}) = 1/1 = 1.00$ $P(\text{the} | \text{on}) = 1/1 = 1.00$
- **Transition Probability Table:**

Current Word	Next Word	$C(w_1, w_2)$	$C(w_1)$	$P(w_2 w_1)$
the	cat	1	2	0.50
the	mat	1	2	0.50
cat	sat	1	1	1.00
sat	on	1	1	1.00
on	the	1	1	1.00

- **Language Generation Using Markov Model:**
 - Start at <s> token; look up all next-word probabilities; sample next word; repeat until </s>
 - Deterministic: always choose highest-probability next word (greedy decoding)

- Stochastic: sample proportional to probabilities — produces different sentences each run
- **Limitations:**
 - Memory-less: ignores all history beyond the last word — misses long-range dependencies (e.g., subject-verb agreement across many words)
 - Data sparsity: many bigrams never seen in training → zero probability → whole sentence probability = 0 (requires smoothing)
 - Higher-order models improve context but worsen sparsity exponentially: V^2 bigrams, V^3 trigrams
 - Cannot model semantics: treats words as discrete tokens with no meaning relationship

A Markov model is memory-less — only the last word matters for prediction, not the full sentence history. Real language has long-distance dependencies (e.g., ‘The keys to the cabinet are...’) that simple Markov models cannot capture. Neural LMs (LSTM, Transformer) solve this by operating in continuous vector space with explicit memory mechanisms.

Q1(b) Add-k Smoothing: Bigram Probability Calculation

[8 Marks]

- **Motivation:** Without smoothing, any bigram not seen during training gets $P = 0$, making the entire sentence probability zero. Smoothing adds a small constant to all bigram counts so no probability is exactly zero.
- **Given:** Corpus = 10,000 words | Vocabulary $V = 5,000$ | $k = 0.5$
 - $C(\text{the, cat}) = 50$ | $C(\text{the}) = 1000$ | $C(\text{cat}) = 100$
- **Add-k Smoothing Formula:**

$$P_k(w^I | w^{I-1}) = [C(w^{I-1}, w^I) + k] / [C(w^{I-1}) + k \times V]$$

Numerator: bigram count + k

Denominator: unigram count + $k \times |\text{vocabulary}|$

Effect: adds k pseudo-counts to every bigram, including unseen ones

- **Step-by-step for sentence ‘the cat sat on the mat’:**

$$\begin{aligned} P(\text{cat} | \text{the}) &= (50 + 0.5) / (1000 + 0.5 \times 5000) = 50.5 / 3500 = 0.01443 \\ P(\text{sat} | \text{cat}) &= (0 + 0.5) / (100 + 0.5 \times 5000) = 0.5 / 2600 = 0.000192 \\ P(\text{on} | \text{sat}) &= (0 + 0.5) / (C(\text{sat}) + 2500) \approx 0.5 / 2501 = 0.000200 \\ P(\text{the} | \text{on}) &= (0 + 0.5) / (C(\text{on}) + 2500) \approx 0.5 / 2501 = 0.000200 \\ P(\text{mat} | \text{the}) &= (0 + 0.5) / (1000 + 2500) = 0.5 / 3500 = 0.000143 \end{aligned}$$

- **Final Sentence Probability:**

$$\begin{aligned} P &= P(\text{cat}|\text{the}) \times P(\text{sat}|\text{cat}) \times P(\text{on}|\text{sat}) \times P(\text{the}|\text{on}) \times P(\text{mat}|\text{the}) \\ &= 0.01443 \times 0.000192 \times 0.000200 \times 0.000200 \times 0.000143 \\ &\text{(very small positive number — no longer zero!)} \end{aligned}$$

k value	Name	Effect	When to Use
$k = 1$	Laplace / Add-1	Strongest smoothing; equal pseudo-count to all unseen bigrams	Simple baseline
$k = 0.5$	Jeffreys-Perks	Moderate; gentler redistribution of probability mass	Common alternative
$k \rightarrow 0$	Unsmoothed MLE	No smoothing; zero problem for unseen bigrams	Never for real LMs
$k \rightarrow \infty$	Uniform LM	All bigrams equally likely; ignores corpus statistics	Never practical

- **Limitation of Add-k Smoothing:** Redistributes too much probability mass away from seen events when vocabulary V is large. Better alternative: Kneser-Ney smoothing, which allocates probability based on how contextually diverse (versatile) a word is.

Add-k smoothing is the simplest exam-relevant smoothing method. $k = 1$ is Laplace smoothing. The denominator $k \times V$ ensures all conditional probabilities still sum to 1 over the vocabulary. For the exam: always write out the formula, substitute values, and compute step-by-step.

Q1(c) Latent Semantic Analysis (LSA)

[4 Marks]

- **Definition:** LSA is an unsupervised dimensionality reduction technique that discovers latent semantic structure in text by applying Singular Value Decomposition (SVD) to a term-document matrix.
- **Core Hypothesis:** Words that appear in similar documents (distributional hypothesis) tend to have similar meanings. LSA maps words and documents into a shared low-dimensional 'semantic' space.
- **LSA Steps:**
 - 1. Build Term-Document Matrix X (m terms \times n documents): each cell = TF-IDF(term, doc)
 - 2. Apply full SVD: $X = U \times \Sigma \times V^T$
 - 3. Truncate to top k singular values: $X_k = U_k \times \Sigma_k \times V_k^T$ (typical $k = 100\text{--}500$)
 - 4. Use X_k for: document similarity, query expansion, topic discovery

Matrix	Dimensions	Content
U	$m \times k$	Term-to-latent-topic mapping; each row is a term's 'topic fingerprint'
Σ	$k \times k$	Diagonal; singular values in decreasing order; magnitude = topic importance
V^T	$k \times n$	Document-to-latent-topic mapping; each column is a document's 'topic fingerprint'

- **What LSA Captures:**
 - Synonymy: 'car' and 'automobile' appear in similar documents \rightarrow nearby in LSA space \rightarrow query for one retrieves both

- Polysemy (partial): ambiguous words get averaged vectors across all their contexts
- Cross-document similarity: documents about the same topic cluster together even without shared vocabulary

LSA is algebraic and deterministic (same result every run). Adding a new document requires recomputing the full SVD. Unlike LDA, it gives no probabilistic interpretation. LSA vs LDA: LSA is faster but less interpretable; LDA is slower but gives explicit probability distributions over topics.

Q2(a) Generative vs Discriminative Models

[4 Marks]

- **Generative Models:** Model the joint probability $P(X, Y)$. Learn how data in each class is generated — can create new samples.
- **Discriminative Models:** Model only $P(Y | X)$. Learn decision boundaries, not data distributions. Cannot generate new data.
- Generative language models learn the statistical patterns of text from large corpora, then use that learned distribution to produce new sequences of words token by token. At their core, they answer a single question repeatedly: *given everything written so far, what comes next?*

Feature	Generative Models	Discriminative Models
Models	$P(X, Y)$ joint distribution	$P(Y X)$ conditional only
Approach	How each class generates data	Decision boundary between classes
Examples (NLP)	Naïve Bayes, HMM, LDA, N-gram LM	Logistic Regression, SVM, CRF, BERT
Can generate new data?	Yes — sample from $P(X, Y)$	No
Data requirements	Works with less labeled data	Typically needs more labeled data
Handles missing features	Yes (marginalize out)	Not directly
Typically used for	Text generation, language modeling, topic modeling	Classification, NER, POS tagging
Accuracy (abundant data)	Often lower	Often higher

- **Generative NLP Example — N-gram LM:**
 - Learns: $P(w_1, w_2, \dots, w_n) = P(w_1) \times P(w_2|w_1) \times \dots \times P(w_n|w_{n-1})$
 - Can generate: sample words sequentially from learned bigram/trigram table
 - Uses Bayes' rule: Naïve Bayes computes $P(\text{class}|\text{doc}) = P(\text{doc}|\text{class}) \times P(\text{class}) / P(\text{doc})$
- **Discriminative NLP Example — Logistic Regression for Sentiment:**
 - Directly models $P(\text{positive} | \text{tweet_features})$ without modeling what tweets look like
 - Features: word counts, n-grams, sentiment scores; weights learned via gradient descent

The generative vs discriminative distinction is fundamental in ML and NLP. LDA (topic model) is generative. BERT (encoder for classification) is discriminative. GANs use both: the Generator is generative, the Discriminator is discriminative. For the exam: always give examples from both categories.

Q2(b) TF-IDF Calculation**[6 Marks]**

- Given Document-Term Matrix:

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

- Find:** TF-IDF of Term 1 in Document 1

- Step 1 – TF (Term Frequency):**

$$TF(t, d) = \text{count}(t \text{ in } d) / \text{total_words_in_}d$$

$$\text{Total terms in Doc1} = 10 + 2 + 1 = 13$$

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

- Step 2 – IDF (Inverse Document Frequency):**

$$IDF(t) = \log(N / df(t)) \quad [\text{natural log or log base 2 — specify in exam}]$$

$$N = 3 \text{ (total documents)}$$

$$df(\text{Term1}) = 2 \text{ (Term1 appears in Doc1 and Doc2; NOT in Doc3)}$$

$$IDF(\text{Term1}) = \log(3/2) = \log(1.5) = 0.405$$

- Step 3 – TF-IDF Score:**

$$TF-IDF(\text{Term1}, \text{Doc1}) = TF \times IDF = 0.769 \times 0.405 = 0.3114$$

- Insight:**

- Term1 in only 1 doc: $IDF = \log(3/1) = 1.099 \rightarrow TF-IDF = 0.769 \times 1.099 = 0.845$ (highly distinctive)
- Term1 in all 3 docs: $IDF = \log(3/3) = 0 \rightarrow TF-IDF = 0$ (treated as stopword — no discriminating power)

IDF variants: (1) Standard: $\log(N/df)$. (2) Smooth IDF: $\log(N/(1+df)) + 1$ — avoids $\log(0)$. (3) Probabilistic (BM25 style): $\log((N-df)/df)$. Always state which variant you use. For the exam, standard $\log(N/df)$ is the default.

Q2(c) Latent Dirichlet Allocation (LDA) for Topic Modeling**[8 Marks]**

- Definition:** LDA is a probabilistic generative model that assumes each document is a mixture of K latent topics, and each topic is a probability distribution over the vocabulary.
- Core Assumptions:**

- Each document covers K topics in varying proportions (mixed membership)
- Each topic is characterized by a probability distribution over all vocabulary words
- Observed words were ‘generated’ by: pick a topic from the document’s mixture, then pick a word from that topic’s distribution

- **Key Parameters:**

Parameter	Symbol	Meaning	Practical Effect
Number of topics	K	User-specified number of latent themes	Too few: broad topics; too many: redundant/fragmented
Doc-topic prior	α	Dirichlet prior over topic proportions per doc	Small α (<1): doc focuses on few topics
Topic-word prior	β	Dirichlet prior over word distribution per topic	Small β (<1): topic has few dominant words
Topic-word dist	ϕ_k	$P(\text{word} \mid \text{topic } k)$ for each topic k	Learned: defines topic’s vocabulary
Doc-topic dist	θ_d	$P(\text{topic} \mid \text{document } d)$	Learned: defines document’s topic mix

- **LDA Generative Process:**

- For each document d:
 - 1. Draw topic mixture: $\theta_d \sim \text{Dirichlet}(\alpha)$ [what fraction of doc is each topic]
 - 2. For each word position i:
 - a. Draw topic: $z_i \sim \text{Multinomial}(\theta_d)$ [which topic generates this word]
 - b. Draw word: $w_i \sim \text{Multinomial}(\phi_{z_{\text{sub}:i}})$ [which word from that topic]

- **Inference (Learning from Data):**

- Observed: w (all words). Hidden: z (topic assignments), θ (doc-topic), ϕ (topic-word)
- Gibbs Sampling: iteratively resample each word’s topic assignment \rightarrow converges to posterior
- Variational Bayes: approximate posterior with a simpler distribution; faster but approximate

- **Example Output (K = 3 topics):**

Topic ID	Top Words (highest ϕ_k)	Human Interpretation
Topic 1	election, vote, government, minister, party	Politics
Topic 2	goal, match, player, team, trophy, league	Sports
Topic 3	stock, market, revenue, profit, shares, investor	Finance

- **Evaluation Metrics:**

- Perplexity: $\exp(-1/N \times \log\text{-likelihood})$ — lower = better model fit

- Topic Coherence (NPMI): measures whether top-K words frequently co-occur in an external corpus — higher = more interpretable topics

LDA vs LSA: LDA provides probabilistic, interpretable topics. LSA gives algebraic, continuous representations with possible negative values. LDA handles polysemy better because each word occurrence gets its own topic assignment (same word can belong to different topics in different contexts). K must be tuned using perplexity on held-out data or coherence scores.

November–December 2023

Q1(a) Generative Models of Language

[4 Marks]

REPEATED — Refer to: May–June 2023 → Q2(a) [Full comparison table: generative vs discriminative]

- **One Generative Model in Detail — N-gram Language Model:**
 - Models: $P(\text{sentence}) = \prod P(w_i | w_{i-1})$ for all consecutive word pairs
 - Training: count all bigrams in corpus, divide by unigram counts to get probabilities
 - Generates new text: start with <s>, sample next word from $P(w | <s>)$, repeat, stop at </s>
 - Evaluation: Perplexity = $\exp(-1/N \times \sum \log P(w_i | w_{i-1}))$ — lower is better

Q1(b) Bigram Probability from Given Corpus

[8 Marks]

- **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>
- **Step 1 – Count all relevant bigrams and unigrams:**

Bigram	$C(w_1, w_2)$	Unigram $C(w_i)$	Source
<s> → students	2	4 (4 sentence starts)	S3, S4
students → are	1	2 (students in S3, S4)	S3
are → from	1	2 (are appears in S3 twice)	S3
from → Pune	2	4 (from in S1, S3, S4 + once more)	S1, S4
Pune → </s>	1	2 (Pune in S1, S4)	S4

- **Step 2 – Compute Bigram Probabilities:**

$$P(\text{students} | <s>) = C(<s>, \text{students}) / C(<s>) = 2/4 = 0.500$$

$$P(\text{are} | \text{students}) = C(\text{students}, \text{are}) / C(\text{students}) = 1/2 = 0.500$$

$$\begin{aligned}
 P(\text{from} \mid \text{are}) &= C(\text{are, from}) / C(\text{are}) = 1/2 = 0.500 \\
 P(\text{Pune} \mid \text{from}) &= C(\text{from, Pune}) / C(\text{from}) = 2/4 = 0.500 \\
 P(</s> \mid \text{Pune}) &= C(\text{Pune, </s>}) / C(\text{Pune}) = 1/2 = 0.500
 \end{aligned}$$

• **Step 3 – Final Sentence Probability:**

$$P = 0.5 \times 0.5 \times 0.5 \times 0.5 \times 0.5 = 0.5^5 = 0.03125$$

Always count unigrams carefully — include ALL occurrences across ALL sentences, not just those relevant to the test sentence. $C(w_i)$ is the total count of w_i in the entire training corpus.

Q1(c) LSA for Topic Modelling

[6 Marks]

REPEATED — Refer to: May–June 2023 → Q1(c) [LSA definition, steps, SVD breakdown]

Aspect	LSA	LDA
Method	SVD algebraic decomposition	Bayesian generative probabilistic model
Output	Real-valued vectors (can be negative)	Probability distributions (sum to 1)
Interpretability	Difficult — negative values have no clear meaning	Easy — each topic is a probability over words
New documents	Must recompute SVD (expensive)	Run inference with learned model
Speed	Fast for small corpora	Slower (Gibbs sampling / variational EM)
Polysemy handling	Partially (averages context)	Better — each word occurrence gets its own topic

Q2(a) Short Note on BERT

[4 Marks]

- **Full Name:** Bidirectional Encoder Representations from Transformers (Google AI, 2018, Devlin et al.)
- **Core Innovation:** Contextual embeddings — same word gets a different vector depending on surrounding context. 'bank' in 'river bank' vs 'bank account' get completely different BERT representations.
- **Architecture:**
 - Transformer Encoder only (no decoder), 12 layers (Base) / 24 layers (Large)
 - Input: [CLS] token + sentence tokens + [SEP] token
 - Hidden size: 768 (Base), 1024 (Large); Multi-head self-attention: 12 heads (Base), 16 (Large)
 - Parameters: 110M (Base), 340M (Large)

- **Pre-training Tasks:**

- MLM (Masked Language Modeling): randomly mask 15% of tokens; predict masked tokens using full bidirectional context
- NSP (Next Sentence Prediction): given sentence pairs (A, B), predict if B is the actual next sentence (50%) or random (50%)

Feature	BERT	word2vec / GloVe
Context-aware?	Yes — same word, different vectors	No — one static vector per word always
Training objective	Masked LM + Next Sentence Prediction	Predict neighboring words (CBOW/Skip-gram)
Architecture	12-layer Transformer Encoder	2-layer shallow neural network
Input scope	Entire sentence (512 tokens max)	Local window (5–10 words)
Handles polysemy	Yes — fully context-dependent	Partially — single averaged vector
Transfer learning	Yes — fine-tune for any task	Usually frozen (fixed embeddings)

BERT variants: RoBERTa (removed NSP, larger batches — often outperforms BERT), DistilBERT (40% smaller, 60% faster), ALBERT (parameter sharing), BioBERT (biomedical literature), mBERT (104 languages simultaneously).

Q2(b) TF-IDF Calculation (Same Matrix)

[6 Marks]

REPEATED — Refer to: May–June 2023 → Q2(b) [Identical document-term matrix — same formula and calculation]

Q2(c) LDA for Topic Modeling

[8 Marks]

REPEATED — Refer to: May–June 2023 → Q2(c) [LDA full explanation with generative process, parameters, example]

May–June 2024

Q1(a) Generative vs Discriminative Models with Example

[9 Marks]

REPEATED — Refer to: May–June 2023 → Q2(a) [Full comparison table]

- **Extended — Bayesian Framework for Generative Models:**

- Model $P(X|Y)$ — likelihood: how does class Y generate features X?
- Model $P(Y)$ — prior: how often does class Y occur in the dataset?
- Apply Bayes: $P(Y|X) = P(X|Y) \times P(Y) / P(X)$ — posterior from likelihood \times prior
- Classification: pick $Y^* = \operatorname{argmax}_Y P(X|Y) \times P(Y)$

Key practical takeaway: use discriminative models (logistic regression, SVM, BERT) when you have sufficient labeled data and care only about the decision boundary. Use generative models (HMM, LDA, N-gram LM) when you need to generate new data, work with limited labeled data, or model the full joint distribution.

Q1(b) LDA: Key Components and Topic Modeling**[9 Marks]****REPEATED — Refer to: May–June 2023 → Q2(c) [LDA generative process, parameters, inference, example]**

- **Extended — Gibbs Sampling Steps:**
 - 1. Randomly initialize: assign each word in each document to a random topic z_{ij}
 - 2. For each word w_{ij} in document d , remove it from its current topic assignment
 - 3. Compute $P(z=k \mid \text{everything else}) \propto (\text{topic } k \text{ words in doc } d + \alpha) \times (\text{word } w \text{ in topic } k + \beta) / (\text{total words in topic } k + V\beta)$
 - 4. Sample new topic from this distribution; reassign w to sampled topic
 - 5. Repeat thousands of iterations until assignments stabilize (convergence)

Q2(a) Contextualized Representations (BERT): Advantages and Disadvantages**[10 Marks]****REPEATED — Refer to: Nov–Dec 2023 → Q2(a) [BERT architecture + pre-training tasks + comparison table]**

- **Advantages of Contextualized Representations:**
 - Resolves polysemy: ‘lead’ as metal vs ‘lead’ as to guide → different vectors based on surrounding context
 - Rich linguistic knowledge: captures syntax (subject–verb relations), semantics, coreference
 - Transfer learning: pre-train once on massive unlabeled text; fine-tune for many tasks with few labeled examples
 - State-of-art results: BERT achieved SOTA on 11 NLP benchmarks simultaneously at release
 - Works for diverse tasks: QA, NER, sentiment, NLI, summarization — same base model, different heads
- **Disadvantages:**
 - Computational cost: BERT-Base needs ~4 GB GPU for fine-tuning; inference slow without GPU acceleration
 - Large memory: 110M parameters (Base) to 340M (Large); impractical for edge devices
 - Black box: difficult to interpret what each attention head or layer has learned
 - Maximum input length: 512 tokens — long documents must be truncated or chunked
 - Encodes biases: pre-trained on internet text which contains gender, racial, and cultural biases

DistilBERT retains 97% of BERT performance with 40% fewer parameters and runs 60% faster — suitable for mobile/edge deployment. Longformer and BigBird extend BERT’s context window beyond 512 tokens using sparse attention for long documents.

Q2(b) Add-k Smoothing**[8 Marks]****REPEATED — Refer to: May–June 2023 → Q1(b) [Same corpus values and calculation]**

May–June 2025

Q1(a) Log-Linear Models in NLP

[6 Marks]

- **Definition:** Log-linear models compute class probability as a weighted sum of features passed through a softmax function. The log of the probability is linear in the feature weights.

- **Core Formula:**

$$P(y | x) = \exp(w \cdot f(x, y)) / Z(x)$$

where:

y = output class (e.g., POS tag, entity label, sentiment)

x = input (word, sentence, context features)

w = learned weight vector

f(x, y) = feature function: extracts relevant features for class y given input x

Z(x) = normalization: $\sum_{y'} \exp(w \cdot f(x, y'))$ [ensures probabilities sum to 1]

Log-linear property: $\log P(y|x) = w \cdot f(x,y) - \log Z(x)$ [linear in w]

- **Feature Types f(x, y) in NLP:**

Feature Category	Examples	Used For
Lexical	Current word, previous word, next word	POS tagging, NER
Morphological	Suffix (-ing, -ed, -tion), prefix (un-, re-)	OOV handling, morphological tagging
Syntactic	POS tag of neighboring words, parse depth	NER, parsing
Binary indicator	Is word capitalized? Is it in a city list?	NER (detects proper nouns)
Word embeddings	Pre-trained word2vec/GloVe vector	Rich semantic features
Contextual	BERT embedding of current token	State-of-art feature for any task

- **Training:** Maximize log-likelihood: sum of $\log P(y_i | x_i)$ over training examples. L1 or L2 regularization prevents overfitting.

Model	Relation to Log-Linear
Logistic Regression (binary)	Simplest log-linear: 2 classes ($y = 0$ or 1)
Softmax Regression (multinomial)	Log-linear with $K > 2$ classes
MaxEnt Classifier	Log-linear with arbitrary hand-crafted NLP features
CRF (Conditional Random Field)	Log-linear extended to sequences; global normalization over all tag sequences

Log-linear models are discriminative — they model $P(y|x)$ directly, not $P(x,y)$. CRFs extend log-linear models to sequences: the normalization Z is computed over all possible tag sequences, enabling globally consistent predictions (e.g., I-PER cannot follow B-LOC).

Q1(b) doc2vec vs word2vec**[8 Marks]**

- **word2vec:** A shallow neural network that learns dense, fixed-size vector representations for words by training to predict context words (Skip-gram) or predict center word from context (CBOW).

- **word2vec Famous Property — Word Analogies:**

$\text{vec}(\text{'king'}) - \text{vec}(\text{'man'}) + \text{vec}(\text{'woman'}) \approx \text{vec}(\text{'queen'})$
 $\text{vec}(\text{'Paris'}) - \text{vec}(\text{'France'}) + \text{vec}(\text{'Germany'}) \approx \text{vec}(\text{'Berlin'})$

The vector space encodes semantic/syntactic relationships as geometric offsets.

- **doc2vec:** Extends word2vec by adding a paragraph/document ID vector trained alongside word vectors, enabling fixed-size representations for entire documents of any length.

Feature	word2vec	doc2vec
Unit of embedding	Single word	Entire document/paragraph
Output	Word embedding (e.g., 300-dim)	Document embedding (same dims)
Architectures	CBOW, Skip-gram	PV-DM, PV-DBOW
Context window	Local window (5–10 words)	Entire document content
Variable-length input?	Per-word (not applicable)	Yes — any length → fixed-size vector
Captures document semantics?	No (only per-word)	Yes — primary purpose
OOV (new words)	Not handled	Not handled (use FastText instead)
Primary use case	Semantic similarity, feature input for ML	Document classification, retrieval, clustering
Inference for new doc	Average word vectors (approximation)	Requires gradient descent to infer doc vector

- **GloVe vs word2vec:**
 - GloVe: learns from global co-occurrence matrix statistics (word–word co-occurrence across entire corpus)
 - word2vec: learns from local context windows; online training from text stream
 - Both produce static embeddings; GloVe often slightly better for semantic tasks, word2vec for syntactic analogies

doc2vec inference on new documents requires an additional forward pass with the paragraph vector as a trainable parameter while keeping word vectors fixed — typically 10–50 steps of gradient descent.

Q1(c) Non-Negative Matrix Factorization (NMF) for Topic Modeling [4 Marks]

- **Concept:** NMF factorizes a document-term matrix V into two non-negative matrices W (document-topic) and H (topic-term) such that $V \approx W \times H$, with all values ≥ 0 .

$$V \approx W \times H$$

V : ($m \times n$) — m documents, n vocabulary terms; values are TF-IDF weights

W : ($m \times k$) — document-topic matrix; $W[d,k]$ = how much doc d belongs to topic k

H : ($k \times n$) — topic-term matrix; $H[k,t]$ = how much term t belongs to topic k

k : user-specified number of topics

Aspect	NMF	LDA	LSA
Approach	Algebraic factorization	Bayesian probabilistic model	SVD decomposition
Non-negative?	Yes (strict)	Yes (probabilities)	No (negatives allowed)
Interpretability	High	High	Medium
Deterministic?	Yes (given initialization)	No (stochastic sampling)	Yes
Best for	Short texts, sparse docs	Long documents, uncertainty	Semantic similarity search

NMF is particularly effective for short texts (tweets, product reviews) where LDA struggles due to few observed words per document. The non-negative constraint gives parts-based representation — no cancellation, every term contributes positively to every topic it belongs to.

Q2(a) Markov Model for Language Generation [6 Marks]

REPEATED — Refer to: May–June 2023 → Q1(a) [Markov model definition, steps, worked example, transition table]

- **Extended — Language Generation Process:**
 - Step 1: Initialize current_state = <s>
 - Step 2: Look up transition probabilities $P(\cdot \mid \text{current_state})$
 - Step 3: Sample next word $w \sim \text{Categorical}(P(\cdot \mid \text{current_state}))$
 - Step 4: Append w ; update current_state = w
 - Step 5: Repeat until </s> is sampled or max length is reached
- **Higher-Order Markov Models:**

Model	History	Formula	Parameters
Unigram	None	$P(w_n)$	V
Bigram	1 previous word	$P(w_n \mid w_{n-1})$	V^2
Trigram	2 previous words	$P(w_n \mid w_{n-2}, w_{n-1})$	V^3
N-gram	$n-1$ previous words	$P(w_n \mid w_{n-n+1} \dots w_{n-1})$	V^n

Q2(b) Latent Semantic Analysis (LSA)**[8 Marks]****REPEATED — Refer to: May–June 2023 → Q1(c) [LSA definition, steps, SVD matrices]**

- **Extended — LSA for Information Retrieval:**
 - Offline: build term-document matrix X , compute truncated SVD \rightarrow store U_k, Σ_k, V_k^T
 - Query q arrives: represent as pseudo-term vector \vec{q} (TF weights for query terms)
 - Project into LSA space: $q_{lsa} = \vec{q}^T \times U_k \times \Sigma_k^{-1}$
 - Find similar documents: $\text{cosine_similarity}(q_{lsa}, V_k[:, d])$ for each document d
- **Limitations of LSA:**
 - No probabilistic interpretation: cannot say ‘document d belongs to topic k with probability p ’
 - Negative values in U and V — hard to interpret as ‘concept weights’
 - SVD is expensive for large matrices; adding new vocabulary requires full recomputation
 - Bag-of-words: ignores word order within documents

Q2(c) TF-IDF Short Note**[4 Marks]**

- **TF-IDF:** Term Frequency–Inverse Document Frequency — a numerical statistic reflecting how important a word is to a document in a corpus.

$TF(t, d) = \text{count}(t \text{ in } d) / \text{total_words_in_}d$
 $IDF(t) = \log(N / df(t))$
 $TF-IDF(t, d) = TF(t, d) \times IDF(t)$

- **IDF Variants:**
 - Standard: $IDF = \log(N/df)$ — can be 0 if term appears in all docs
 - Smooth IDF: $IDF = \log(N/(1+df)) + 1$ — always positive; most common in practice
 - Probabilistic (BM25): $IDF = \log((N - df + 0.5) / (df + 0.5))$ — better theoretical grounding

November–December 2025**Q1(a) Bigram Model with Add-1 Smoothing****[9 Marks]**

- **Given Corpus:**
 - S1: ‘language models learn patterns’
 - S2: ‘models learn from data’
 - S3: ‘data helps improve language models’
- **Target Sentence:** ‘language models learn from data’
- **Step 1 – Vocabulary ($V = 8$ unique words):**
 - {language, models, learn, patterns, from, data, helps, improve}

- **Step 2 – Relevant Bigram Counts:**

Bigram	$C(w_1, w_2)$	Sentence
language → models	2	S1 and S3
models → learn	1	S1 only
learn → from	0	NEVER seen in training data
from → data	1	S2 only

- **Step 3 – Add-1 (Laplace) Smoothed Probabilities:**

Formula: $P_{\text{smooth}}(w_{\text{sub:2}} | w_{\text{sub:1}}) = [C(w_{\text{sub:1}}, w_{\text{sub:2}}) + 1] / [C(w_{\text{sub:1}}) + V]$

$P(\text{models} | \text{language}) = (2+1) / (2+8) = 3/10 = 0.300$

$P(\text{learn} | \text{models}) = (1+1) / (3+8) = 2/11 = 0.182$

$P(\text{from} | \text{learn}) = (0+1) / (2+8) = 1/10 = 0.100 \leftarrow \text{was ZERO without smoothing!}$

$P(\text{data} | \text{from}) = (1+1) / (1+8) = 2/9 = 0.222$

- **Step 4 – Final Sentence Probability:**

$P(\text{sentence}) = 0.300 \times 0.182 \times 0.100 \times 0.222 = 0.001212 \ (\approx 1.21 \times 10^{-3})$

Without smoothing: $P(\text{from} | \text{learn}) = 0/2 = 0 \rightarrow \text{entire sentence probability} = 0$. With Add-1: $P(\text{from} | \text{learn}) = 1/10 = 0.100 \rightarrow \text{non-zero}$. The denominator $C(w_i) + V$ ensures all $P(w_i | w_{i-1})$ sum to 1 over the vocabulary.

Q1(b) Probabilistic Language Modeling: Trigram MLE

[9 Marks]

- **Probabilistic Language Modeling:** The task of assigning probability to sequences of words: $P(w_1, w_2, \dots, w_n)$.

- **Chain Rule (Exact but Intractable):**

$P(w_1, w_2, \dots, w_n) = P(w_1) \times P(w_2 | w_1) \times P(w_3 | w_1, w_2) \times \dots$

Problem: $P(w_n | w_1, \dots, w_{n-1})$ requires counting every possible sentence prefix
→ intractable for long sentences

- **Trigram MLE Formula and Example:**

$P_{\text{MLE}}(w^I | w_{-2}^I, w_{-1}^I) = C(w_{-2}^I, w_{-1}^I, w^I) / C(w_{-2}^I, w_{-1}^I)$

Corpus: 'I like to play football. I like to play cricket.'

$C(\text{like}, \text{to}, \text{play}) = 2$

$C(\text{like}, \text{to}) = 2$

$P(\text{play} | \text{like}, \text{to}) = 2/2 = 1.00$

$P(\text{football} | \text{like}, \text{to}) = 1/2 = 0.50$

- **Perplexity:**

$$\text{Perplexity}(\text{LM}, \text{test}) = \exp(-1/N \times \sum^I \log P(w^I | w_{-I}^I))$$

Lower perplexity = better LM

Unigram LM: perplexity $\approx V$ (30,000–50,000)

Bigram LM: perplexity ≈ 200 –1,000

Trigram LM: perplexity ≈ 50 –200 (with smoothing)

LSTM/Transformer: perplexity ≈ 20 –60

Perplexity intuition: ‘How confused is the model?’ A perplexity of 100 means the model is on average as uncertain as if it were choosing uniformly from 100 equally likely next words. Trigram models need smoothing to achieve competitive perplexity — Kneser-Ney trigrams achieve ~70–100 on PTB.

Q2(a) NMF vs LDA for Topic Modeling

[9 Marks]

REPEATED — Refer to: May–June 2025 → Q1(c) [NMF] + May–June 2023 → Q2(c) [LDA]

Feature	NMF	LDA
Type	Algebraic matrix factorization	Probabilistic graphical model
Optimization	Minimize $\ V - WH\ ^2$ (reconstruction error)	Maximize log-likelihood $P(\text{documents} \text{topics})$
Non-negative?	Yes (strict constraint)	Yes (probabilities sum to 1)
Deterministic?	Yes (reproducible given same init)	No (stochastic Gibbs sampling)
Topic-word format	$H[k, :] = \text{weights (any positive scale)}$	$\phi_k = \text{probabilities (sum to 1 over vocab)}$
Doc-topic format	$W[d, :] = \text{weights (unnormalized)}$	$\theta_d = \text{probabilities (sum to 1 over K topics)}$
Speed	Faster (≈ 200 iterations)	Slower (1000s of Gibbs iterations)
Best for	Short texts, images, signal processing	Long documents, uncertainty estimation

Q2(b) TF-IDF Calculation on 3 Documents

[9 Marks]

- **Documents:**
 - D_1 : ‘neural networks are powerful’ (4 words)
 - D_2 : ‘deep learning powers neural models’ (5 words)
 - D_3 : ‘networks and models are important’ (5 words)
- **Find:** TF-IDF of ‘neural’ in D_1, D_2, D_3

- **Step 1 – Term Frequencies:**

$$\text{TF}(\text{neural}, D_1) = 1/4 = 0.250$$

$$\text{TF}(\text{neural}, D_2) = 1/5 = 0.200$$

$$\text{TF}(\text{neural}, D_3) = 0/5 = 0.000$$

- **Step 2 – IDF:**

$df(\text{neural}) = 2$ [appears in D_1 and D_2 , not D_3]
 $IDF(\text{neural}) = \log(3/2) = \log(1.5) = 0.405$

- **Step 3 – TF-IDF Scores:**

$TF\text{-}IDF(\text{neural}, D_1) = 0.250 \times 0.405 = 0.1013$
 $TF\text{-}IDF(\text{neural}, D_2) = 0.200 \times 0.405 = 0.0810$
 $TF\text{-}IDF(\text{neural}, D_3) = 0.000 \times 0.405 = 0.0000$

- **Interpretation:**

- D_1 scores higher than D_2 for 'neural' because $1/4 > 1/5$ (same IDF, higher TF)
- If 'neural' appeared in all 3 docs: $IDF = \log(3/3) = 0 \rightarrow TF\text{-}IDF = 0$ for all (treated as stopword)
- If 'neural' appeared in only 1 doc: $IDF = \log(3/1) = 1.099 \rightarrow TF\text{-}IDF(D_1) = 0.250 \times 1.099 = 0.275$

Topic Frequency Analysis — Unit 3

Rank	Topic	Sessions	Priority
1	LDA (Latent Dirichlet Allocation)	MJ23, ND23, MJ24, ND25	VERY HIGH — 4×
2	Bigram/N-gram Probability Calculations	MJ23, ND23, MJ25, ND25	VERY HIGH — 4×
3	TF-IDF Calculation	MJ23, ND23, ND25	HIGH — 3×
4	LSA (Latent Semantic Analysis)	MJ23, ND23, MJ25	HIGH — 3×
5	Generative vs Discriminative Models	MJ23, ND23, MJ24	HIGH — 3×
6	BERT / Contextualized Representations	ND23, MJ24	MEDIUM — 2×
7	Markov Models	MJ23, MJ25	MEDIUM — 2×
8	NMF	MJ25, ND25	MEDIUM — 2×
9	doc2vec vs word2vec	MJ25	LOW — 1×
10	Log-linear Models	MJ25	LOW — 1×

EXAM STRATEGY — Unit 3: (1) LDA: know all 5 parameters ($\alpha, \beta, K, \phi, \theta_d$) + the generative process steps + Gibbs sampling concept + evaluation metrics. (2) Bigram calculations: memorize both Add-1 and Add-k formulas; practice counting bigrams and unigrams carefully. (3) TF-IDF: know Standard and Smooth IDF variants; practice full calculations with tables. (4) BERT: know MLM + NSP tasks, bidirectionality advantage, comparison table with word2vec.