টেক্সট রিপ্রেজেন্টেশন / ফিচার এক্সট্রাকশন — ডিপ এবং সহজ ব্যাখ্যা

মেশিনকে ভাষা শেখাতে হলে তাকে সংখ্যা শিখাতে হয়।
কারণ কম্পিউটার ভাষা বোঝে না — সে বোঝে সংখ্যা, ম্যাট্রিক্স আর ভেক্টর।
এই টেক্সট → ভেক্টর রূপান্তর প্রক্রিয়াকেই বলা হয় **Text Representation / Text Vectorization**।

এটা ঠিক যেমন —
প্রতিটি মানুষের মুখের আলাদা ফিচার থাকে (চোখ, নাক, রং, উচ্চতা),
মেশিন লার্নিং মানুষের মুখকে ভেক্টর বানায় ফিচারের ওপর ভিত্তি করে।

একইভাবে টেক্সটকেও ফিচারে ভাঙা হয়।

Core Terminology (খুব গুরুত্বপূর্ণ)

ধরো তোমার কাছে ৩টা রিভিউ আছে:

1) The food was good
2) The service was not good
3) The food was very bad

Corpus

সব তিনটি বাক্যের সমস্ত শব্দ মিলিয়ে = Corpus
→ ["The", "food", "was", "good", "service", "not", "very", "bad"]

Vocabulary

ইউনিক শব্দগুলো
→ 8টি unique word

Document

প্রতিটি বাক্য একটি ডকুমেন্ট

Word

ডকুমেন্টের প্রতিটি শব্দ

One Hot Encoding (OHE) — বাচ্চাদের লেভেল রিপ্রেজেন্টেশন

ধরো Vocabulary:
→ ["food", "good", "bad", "not", "very"]

তাহলে:

"good" → [0,1,0,0,0]
"bad" → [0,0,1,0,0]

✪ সুবিধা:

- খুব সহজ
- স্পষ্ট বোঝা যায়

✪ বড় সমস্যা:

- ভেক্টর খুব বড় হয়ে যায় → Memory Problem
  (Example: 1 লাখ শব্দ → ভেক্টর সাইজ 1 লাখ)
- শব্দের অর্থ ⌀
- OOV সমস্যা

বাস্তব উদাহরণ:

“cat” vs “dog”

OHE বলবে → সম্পূর্ণ ভিন্ন
কিন্তু আসলে → দুটোই প্রাণী

OHE এটা বুঝতে পারে না।

Bag of Words (BoW) — শব্দ গোনা

“good good service bad”

→ vocabulary size 5 হলে

good = 2
service = 1
bad = 1

→ vector = [0,2,1,1,0]

সুবিধা:

- frequency তথ্য দেয় (একদম বেসিক ML models এ useful)

⚠ অসুবিধা:

- শব্দের ক্রম হারিয়ে যায়
  “not good” vs “good not” → SAME
- সেন্টিমেন্ট লস হয়
- context নেই

দুটি বাক্য তুলনা করো:

1. "Movie was good"
2. "Movie was not good"

Bag of Words → দুটোতেই "good" আছে
→ sentiment difference বুঝে না

N-Grams — Context-aware BoW

এখন ব্যাগে শুধু শব্দ নয়, শব্দের জোড়া নিব

"not good" = bi-gram
"very good movie" = tri-gram

কী লাভ?

→ negation detect করতে পারে
→ phrase meaning বোঝে

উদাহরণ:

"not good"
"very good"

BoW দুটোতেই "good" same
কিন্তু bi-gram:

→ not good = negative
→ very good = positive

TF-IDF — স্মার্ট ফিচার ওয়েটিং

মূল ধারণা:
সব শব্দ সমান গুরুত্বপূর্ণ নয়!

“The”, “is”, “a” → everywhere
→ কম গুরুত্ব

“amazing”, “terrible”, “fraud” → rare but important
→ বেশি গুরুত্ব

TF → কতবার এসেছে

IDF → কতটা বিরল

Formula idea (simple):

TF = word frequency / total words
IDF = log(total docs / docs containing word)

অর্থাৎ:

common word → low weight
rare & meaningful → high weight

উদাহরণ:

Two reviews:

A: “The food was amazing”
B: “The service was bad”

TF-IDF দেবে:

amazing = high
bad = high
the/was = low

Custom Features — Human Intelligence + Machine Learning

Domain knowledge example:

Sentiment analysis এ:
✔ positive word count
✔ negative word count
✔ exclamation (!) count
✔ review length

Spam detection:
✔ কতগুলো লিঙ্ক আছে
✔ uppercase ratio
✔ email domain

Example:

"This is GREAT!!!"

Features:

- uppercase words = GREAT
- exclamation = 3
- sentiment score high

বাস্তব প্রয়োগ উদাহরণ **(Super Helpful)**

Movie review dataset

Review	Model Input
"Movie was not good"	BoW → good=1, not=1
	n-gram → ["not good"]
	TF-IDF → good low weight, not good phrase high weight

Limitations of Traditional Methods

সবগুলোর সমস্যা:

❌ semantic meaning নেই
❌ synonyms বুঝতে পারে না

❌ context shallow
❌ word order partial

Example:

"beautiful" ≈ "gorgeous"
"angry" ≠ "hungry"

কিন্তু traditional vector → same or random

এ সমস্যা সমাধান করে →

Word Embeddings (Word2Vec, GloVe, FastText)

যেখানে:
cat → similar to dog
king - man + woman = queen

Word Embeddings — টেক্সট রিপ্রেজেন্টেশনের বুদ্ধিমান যুগ

কেন **Word Embedding** দরকার?

আগের মডেলগুলোতে সমস্যা ছিল:

- "good" vs "nice" → একই অর্থ, কিন্তু ভেক্টর আলাদা
- স্পার্স ভেক্টর (mostly zeros)
- word-order তথ্য কম
- context ধরা কঠিন

Word embedding সমাধান দেয়:

✔ dense (compact) vectors
✔ semantic similarity
✔ context awareness
✔ efficient memory
✔ learn relationships

উদাহরণ:

king - man + woman = queen

এটা TF-IDF বা BoW করতে পারে না কখনও!

১. Word2Vec

Word2Vec হলো সবচেয়ে জনপ্রিয় ও প্রাথমিক embedding technique।

এটি দুটি আর্কিটেকচারে কাজ করে:

1️⃣ CBOW (Continuous Bag of Words)

Context → target word predict করে

Example sentence:

The food was very good

Context শব্দ:

[The, food, was, very] → good প্রেডিক্ট করবে

2️⃣ Skip-Gram

Target word → context predict করে

good → very, was, food

Key Idea:

উপস্থিতিতে (co-occurrence) ভিত্তি করে meaning শেখানো।

বাস্তব উদাহরণ:

Word2Vec শেখার পরে:

dog ≈ cat
king ≠ banana

Vector উদাহরণ (সহজ):

(dimension কমিয়ে ধরছি)

king = [0.8, 0.65]
queen = [0.79, 0.67]
man = [0.50, 0.30]
woman = [0.51, 0.32]

দেখো!
king এবং queen → কাছাকাছি
king এবং banana → দূরে

২. GloVe (Global Vectors)

Word2Vec local context দেখে,
GloVe global + local দুটোই ব্যবহার করে।

মূল ধারণা:

- শব্দ pair এর co-occurrence probability
- পুরো corpus এ global statistics

উদাহরণ:

"ice" vs "steam"

metric দেখবে:

- ice – "cold" → strong
- ice – "hot" → weak
- steam – "cold" → weak
- steam – "hot" → strong

ফলে embeddings meaning শেখে।

৩. FastText

Word2Vec শুধু শব্দে ফোকাস করে।
FastText → subword পর্যন্ত ভাঙে।

Example:

playing → play + ing
unhappy → un + happy

এটার সুবিধা:

✔ rare word handle
✔ better morphology
✔ OOV কম

উদাহরণ:

"Bangladesh" → না থাকলেও
"Bangla" + "desh" subword থেকে meaning বের করতে পারে।

৪. Contextual Embeddings — BERT, GPT, ELMo

Word2Vec / GloVe static embedding
→ "bank" সবসময় same meaning

কিন্তু BERT/ELMo/GPT contextual:

Sentence 1:

I went to the bank to withdraw money.

Sentence 2:

River bank is beautiful.

এখন:

$bank_1$ → finance vector
$bank_2$ → river vector

অর্থাৎ:
✔ same শব্দ
✔ ভিন্ন অর্থ
✔ ভিন্ন ভেক্টর

👉 এটাকেই contextual embedding বলে।

ফলে:

- polysemy detect পারে
- context dependency
- deep semantic representation

Embedding Space Visualization (intuition)

৩-D space ভাবো।

dogs, cats, tigers — কাছাকাছি
rose, tulip — কাছাকাছি
visa, passport — কাছাকাছি

embedding space cluster করে।

Real Example — Similarity

Word2vec similarity score:

cosine_similarity("good", "nice") ≈ 0.85
cosine_similarity("good", "bad") ≈ 0.2

cosine_similarity("good", "banana") ≈ 0.01

অর্থাৎ embedding → meaning বোঝে!

Practical Use Cases

Sentiment Analysis

word2vec + LSTM →
bert → state-of-the-art

Machine Translation

embedding → words relationship

ChatGPT / GPT

transformers + contextual embedding

Search Engine

query expansion:
doctor → physician → clinic

Big Picture

embedding = **text meaning → math meaning**

Traditional → surface level
Embedding → deep level

Transformer Architecture — NLP-এর Game Changer

Transformer 2017 সালে Google-এর paper “Attention is All You Need” দিয়ে এসেছে।

কেন **Transformer?**

- আগের RNN/LSTM sequential processing করত → slow, long sentences এ gradient vanish সমস্যা।
- Transformer **parallel processing** করতে পারে → দ্রুত।
- **Attention mechanism** দিয়ে context বোঝে → শব্দের গুরুত্বপূর্ণ অংশ ধরা সহজ।

মূল কম্পোনেন্ট

Transformer মূলত ২ ভাগে ভাগ:

1. **Encoder** → input sentence process করে context vector তৈরি করে
2. **Decoder** → output sentence generate করে (translation, summarization)

যদি শুধু embeddings/understanding দরকার → শুধু Encoder যথেষ্ট।

Encoder Structure

প্রতিটি encoder block এর ভিতরে থাকে:

1. **Multi-Head Self-Attention**
    - প্রতিটি শব্দ নিজের sentence এর অন্য সব শব্দের সাথে সম্পর্ক দেখবে।
    - উদাহরণ:
      Sentence: "The cat sat on the mat"
        - Attention: “sat” শব্দের context → "cat" (subject), "mat" (location)
2. **Feed Forward Neural Network (FFN)**
    - Linear layers + ReLU → feature transformation
3. **Add & Norm**
    - Residual connection + Layer normalization → training stability
4. **Positional Encoding**
    - Transformer sequential না হলেও শব্দের position দরকার
    - sine/cosine function দিয়ে position info যোগ করা হয়

Encoder Block Diagram (সাধারণভাবে)

Input Embedding + Positional Encoding
        |
 Multi-Head Self-Attention
        |
 Add & Norm
        |
 Feed Forward Network
        |
 Add & Norm
        |
Output → next encoder / decoder

Decoder Structure

Decoder encoder এর output ব্যবহার করে sentence generate করে।
Extra কম্পোনেন্ট:

- **Masked Self-Attention** → future words hide
- **Encoder-Decoder Attention** → input sentence থেকে info নেয়

Attention Mechanism

Scaled Dot-Product Attention

Attention(Q, K, V) = softmax(QK^T / sqrt(d_k)) * V

- Q = Query
- K = Key
- V = Value

💡 Intuition:

- “Query” = কোন শব্দের context আমরা খুঁজছি
- “Key” = অন্য শব্দের relevance
- “Value” = info যা pick করব

Multi-Head Attention

- একাধিক attention একসাথে ব্যবহার করা হয়
- বিভিন্ন "subspace" থেকে context শেখে

- ফলে model better generalization পায়

BERT (Bidirectional Encoder Representations from Transformers)

BERT হলো **Google 2018** এর innovation।

মূল বৈশিষ্ট্য**:**

1. **Bidirectional**
    - সব words এর context দুই দিকে (left+right) দেখে
    - আগের models (ELMo, GPT) শুধু left-to-right context
2. **Pre-trained**
    - Massive corpus (Wikipedia, BookCorpus) → general knowledge capture
    - Fine-tune করে downstream tasks (QA, Sentiment, NER)
3. **Architecture**
    - শুধুই Encoder blocks (stacked 12/24 layers depending on model)
    - Masked Language Model (MLM) → 일부 words mask করে predict করা
    - Next Sentence Prediction (NSP) → sentence relationship শেখা

BERT Input Representation

BERT input = WordPiece tokens + special tokens

- `[CLS]` → classification tasks start
- `[SEP]` → sentence separator
- Example:

Sentence: "The movie was great"
Tokens: [CLS] The movie was great [SEP]

- Each token → embedding (token + position + segment)
- Then pass through encoder layers

Fine-Tuning Example

Task: Sentiment Classification

1. Input: `[CLS] The movie was great [SEP]`
2. BERT output: `[CLS]` vector (768-dim)
3. Pass through dense layer → Softmax → Positive/Negative

Key Advantages of BERT

- Context-aware
- Bidirectional → polysemy detect
- Transfer learning ready → small dataset-ൽ high accuracy
- Standard for NLP tasks (QA, NER, Summarization, Sentiment)

Intuition with Example

Sentence:

"The bank will not approve my loan."

- Word “bank” → financial context

BERT vector = bank(finance)

Sentence:

"The river bank is full of water."

- Word “bank” → river context

BERT vector = bank(river)

Same word, different vectors → context-aware

Summary Table

Feature	Transformer	BERT
Architecture	Encoder+Decoder	Encoder only
Attention	Self + Multi-Head	Multi-Head (Bidirectional)
Context	Limited (decoder attention)	Full bidirectional
Pretraining	N/A	MLM + NSP
Use Case	Seq2Seq (Translation)	Downstream NLP tasks
Input	Embedding + Positional Encoding	WordPiece + CLS/SEP + Positional

BERT Internals: Attention, QKV, Multi-Head

BERT = **stacked Transformer Encoders**
প্রতিটি encoder block এর প্রধান উপাদান হলো **Multi-Head Self-Attention** + Feed-Forward Network।

আমরা এখানে focus করব **Attention math + QKV + Multi-Head**।

🔹 Input Embedding

BERT input =

[CLS] The movie was great [SEP]

Step 1: Convert each token → vector (Token embedding)
Step 2: Add **Positional embedding** → position info যোগ করা

ধরো input sequence length = 5
vector size = 768

X = [x1, x2, x3, x4, x5] # 5 x 768

---

২ Query, Key, Value (QKV)

Attention মূল ধাপ = QKV calculation

- প্রতিটি input token x → তিনটি vector তৈরি হয়:

Q = x * W_Q
K = x * W_K
V = x * W_V

- W_Q, W_K, W_V = trainable weight matrices (768x64, ধরলে head size 64)

Intuition:

Component	Role
Q (Query)	কোন token-এর জন্য attention খুঁজছি
K (Key)	অন্য token কত relevant
V (Value)	যা আমরা pick করব / context info

---

৩ Scaled Dot-Product Attention

Step-by-step formula:

Attention(Q, K, V) = softmax((Q . K^T) / sqrt(d_k)) * V

- Q.K^T → similarity matrix (how much each token attends other tokens)
- sqrt(d_k) → scaling factor (prevent large dot products)

- softmax → probability distribution
- multiply V → final context vector

উদাহরণ **(small numbers)**

Sentence: "The movie was great" → 4 tokens

Suppose Q,K,V = 2-dim simplification

Q1 = [1,0], K1=[1,0], V1=[2,1]
Q2 = [0,1], K2=[0,1], V2=[1,3]
...

Compute Q1.K^T → similarity with all tokens
Apply softmax → weights
Multiply weights with V → final vector for token1

✅ ফলে প্রতিটি token নিজ sentence context capture করে

8️⃣ Multi-Head Attention

BERT uses **multiple attention heads** → different "subspaces" থেকে attention শেখে

- Suppose 12 heads, each head size = 64
- Step:

1. Split input embeddings (768-dim) → 12 heads x 64-dim each
2. Compute **Attention** independently for each head (different W_Q, W_K, W_V)
3. Concatenate all head outputs → 768-dim vector
4. Linear projection → feed to next layer

Intuition:

- Head1 → focus on subject
- Head2 → focus on verb
- Head3 → focus on object

💡 এক sentence → একাধিক "perspective" থেকে context বুঝে

9️⃣ Add & Norm

- Attention output + original input → residual connection
- Layer normalization → stability

Z = LayerNorm(X + Attention(X))

8️⃣ Feed-Forward Network (FFN)

- 2-layer fully connected network
- Activation: GELU
- Applied **token-wise**

FFN(Z) = max(0, Z*W1 + b1) * W2 + b2

- Residual + LayerNorm again

9️⃣ Stacking Encoders

BERT Base: 12 encoder layers
BERT Large: 24 encoder layers

- Output final layer → contextual embedding for each token
- `[CLS]` vector → sentence-level representation

Summary: Flow of BERT Encoder

Input tokens → Embedding + Positional Encoding
↓
Multi-Head Self-Attention (Q,K,V → Scaled Dot-Product → Context)
↓
Add & Norm
↓
Feed-Forward Network + Add & Norm
↓
Output → Next encoder layer or final embeddings

Intuition Example

Sentence:

"The bank will not approve my loan."

- Query = "bank"
- Keys = ["The", "bank", "will", "not", "approve", "my", "loan"]
- Attention → detect "loan", "approve" are highly related
- Multi-Head → simultaneously detects negation "not" and context "financial"

Result:

- “bank” embedding → financial meaning
- “loan” embedding → related meaning
- `[CLS]` vector → entire sentence meaning → sentiment/QA tasks