Summarization

Text summarization is a technique in natural language processing (NLP) used to condense large amounts of text into shorter, meaningful summaries. There are two primary types of summarization techniques: **Extractive Summarization** and **Abstractive Summarization**.

Extractive Summarization

Extractive summarization involves selecting and combining the most important sentences or phrases directly from the original text without altering their wording. It ranks sentences based on relevance using algorithms and outputs them as the summary.

Abstractive Summarization

Abstractive summarization generates new sentences by understanding the meaning of the original text and rewriting it in a concise manner. This approach requires advanced language generation techniques, such as neural networks, to produce summaries that may include words or phrases not present in the original text.

Input Text

Let's consider the following input text:

"Apple has announced that it will be releasing a new iPhone in the coming months. The new phone, called the iPhone 12, will feature a completely redesigned exterior and a host of new features, including a more powerful processor and improved camera. The iPhone 12 will be available in a variety of colors and storage capacities, and will be available for pre-order later this month."

Extractive Summarization Example
Step 1: Tokenize Sentences
Split the input text into individual sentences:
1. "Apple has announced that it will be releasing a new iPhone in the coming months."
2. "The new phone, called the iPhone 12, will feature a completely redesigned exterior and a host of new features, including a more powerful processor and improved camera."
3. "The iPhone 12 will be available in a variety of colors and storage capacities, and will be available for pre-order later this month."
Step 2: Rank Sentences
Using algorithms like TextRank or TF-IDF, assign scores to sentences based on their importance:
• Sentence 1: Score = 0.8
• Sentence 2: Score = 0.9
• Sentence 3: Score = 0.7
Step 3: Select Top Sentences
Choose sentences with the highest scores:
• Selected Sentences: Sentence 2 ("The new phone, called the iPhone 12, will feature a completely redesigned exterior and a host of new features, including a more powerful processor and improved camera.")
Output Extractive Summary:
"The new phone, called the iPhone 12, will feature a completely redesigned exterior and a host of new features, including a more powerful processor and improved camera."

Abstractive Summarization Example

Step 1: Understand Context
A neural network model (e.g., Transformer-based models like T5 or GPT) processes the input text to understand its meaning.
Step 2: Generate New Sentences
The model rewrites the content by synthesizing information:
• Input to Model: Full text.
• Output from Model: "Apple is releasing a new iPhone called iPhone 12 with improved features like better design, enhanced camera, and faster processor."
Output Abstractive Summary:
"Apple is releasing a new iPhone called iPhone 12 with improved features like better design, enhanced camera, and faster processor."

Comparison Between Extractive and Abstractive Summarization

Feature	**Extractive Summarization**	**Abstractive Summarization**
Method	Selects key sentences from original text	Generates new sentences based on understanding of content
Output	Direct quotes from original text	Paraphrased or reworded summary
Complexity	Computationally simpler	Computationally complex
Example Output	"The new phone... improved camera."	"Apple is releasing... enhanced camera."

https://www.perplexity.ai/search/abstractive-summarization-extr-3Rf1IaS2S62KH_esjYP4fw

Text Summarization

Extractive Summarization Techniques with Examples

Let's work with this 5-sentence document:

1. "**Machine** learning enables computers to learn from data."
2. "Neural networks are a popular **machine** learning approach."
3. "Data preprocessing improves model performance significantly."
4. "**Machine** learning algorithms require substantial computing resources."
5. "Transfer learning reuses knowledge from one task for another task."

TF-IDF Calculation

Term	Sentence	TF	Document Frequency	IDF	TF-IDF
machine	1	1/6 = 0.167	3/5 = 0.6	log(5/3) = 0.51	0.167 × 0.51 = 0.085
learning	1	1/6 = 0.167	4/5 = 0.8	log(5/4) = 0.22	0.167 × 0.22 = 0.037
enables	1	1/6 = 0.167	1/5 = 0.2	log(5/1) = 1.61	0.167 × 1.61 = 0.269
computers	1	1/6 = 0.167	1/5 = 0.2	log(5/1) = 1.61	0.167 × 1.61 = 0.269
learn	1	1/6 = 0.167	1/5 = 0.2	log(5/1) = 1.61	0.167 × 1.61 = 0.269
data	1	1/6 = 0.167	2/5 = 0.4	log(5/2) = 0.92	0.167 × 0.92 = 0.154
neural	2	1/6 = 0.167	1/5 = 0.2	log(5/1) = 1.61	0.167 × 1.61 = 0.269
networks	2	1/6 = 0.167	1/5 = 0.2	log(5/1) = 1.61	0.167 × 1.61 = 0.269
popular	2	1/6 = 0.167	1/5 = 0.2	log(5/1) = 1.61	0.167 × 1.61 = 0.269
machine	2	1/6 = 0.167	3/5 = 0.6	log(5/3) = 0.51	0.167 × 0.51 = 0.085
learning	2	1/6 = 0.167	4/5 = 0.8	log(5/4) = 0.22	0.167 × 0.22 = 0.037
approach	2	1/6 = 0.167	1/5 = 0.2	log(5/1) = 1.61	0.167 × 1.61 = 0.269

(Similar calculations would continue for all terms in all sentences)

Sentence Scores (Sum of TF-IDF values)

Sentence	Sum of TF-IDF	Rank
1	1.083	3
2	1.198	2
3	1.452	1
4	0.876	4
5	0.764	5

For a 2-sentence summary, we would select sentences 3 and 2:

- "Data preprocessing improves model performance significantly."
- "Neural networks are a popular machine learning approach."

TextRank Algorithm

Step 1: Calculate Sentence Similarity Matrix

Using cosine similarity between sentence vectors (simplified):

	S1	**S2**	**S3**	**S4**	**S5**
S1	1.00	0.35	0.12	0.42	0.18
S2	0.35	1.00	0.25	0.30	0.22
S3	0.12	0.25	1.00	0.15	0.28
S4	0.42	0.30	0.15	1.00	0.20
S5	0.18	0.22	0.28	0.20	1.00

Step 2: TextRank Score Calculation

Initial score for each sentence = 1.0 Damping factor (d) = 0.85

Iteration 1: For S1:

Score(S1) = (1-0.85) + 0.85 × [0.35×1.0/1.15 + 0.12×1.0/1.80 + 0.42×1.0/1.07 + 0.18×1.0/0.88]

Score(S1) = 0.15 + 0.85 × [0.304 + 0.067 + 0.393 + 0.205]

Score(S1) = 0.15 + 0.85 × 0.969 Score(S1) = 0.15 + 0.824 Score(S1) = 0.974

(Similar calculations for all sentences through multiple iterations until convergence)

Final TextRank Scores (after convergence):

Sentence	Final Score	Rank
S1	1.124	2
S2	0.986	3
S3	0.763	5
S4	**1.245**	**1**
S5	0.882	4

For a 2-sentence summary, we would select sentences 4 and 1:

- "Machine learning algorithms require substantial computing resources."
- "Machine learning enables computers to learn from data."

Abstractive Summarization: Attention Mechanism Calculation

Example for Sequence-to-Sequence with Attention

Consider we have encoded our input document into 5 hidden states h_1 to h_5, each with dimension 3:

Hidden State	Vector Representation
h_1	[0.2, 0.5, -0.3]
h_2	[0.6, 0.1, 0.4]
h_3	[-0.1, 0.7, 0.2]
h_4	[0.5, -0.2, 0.3]
h_5	[0.3, 0.4, -0.5]

Current decoder state s_1 = [0.4, 0.3, 0.2]

Step 1: Calculate Attention Scores

Let's use a simplified attention scoring: $e_{1,j} = \tanh(W_1 h_(+ W_2 s_1)$

Where W_1 = [[0.1, 0.2, 0.3], [0.2, 0.1, 0.5], [0.4, 0.3, 0.1]] and W_2 = [[0.5, 0.1, 0.3], [0.2, 0.4, 0.1], [0.3, 0.2, 0.5]]

For h_1: $W_1 h_1$ = [[0.1, 0.2, 0.3], [0.2, 0.1, 0.5], [0.4, 0.3, 0.1]] × [0.2, 0.5, -0.3] = [0.1×0.2 + 0.2×0.5 + 0.3×(-0.3), 0.2×0.2 + 0.1×0.5 + 0.5×(-0.3), 0.4×0.2 + 0.3×0.5 + 0.1×(-0.3)] = [0.02 + 0.1 - 0.09, 0.04 + 0.05 - 0.15, 0.08 + 0.15 - 0.03] = [0.03, -0.06, 0.20]

$W_2 s_1$ = [[0.5, 0.1, 0.3], [0.2, 0.4, 0.1], [0.3, 0.2, 0.5]] × [0.4, 0.3, 0.2] = [0.5×0.4 + 0.1×0.3 + 0.3×0.2, 0.2×0.4 + 0.4×0.3 + 0.1×0.2, 0.3×0.4 + 0.2×0.3 + 0.5×0.2] = [0.2 + 0.03 + 0.06, 0.08 + 0.12 + 0.02, 0.12 + 0.06 + 0.1] = [0.29, 0.22, 0.28]

$e_{1,1} = \tanh(W_1 h_1 + W_2 s_1) = \tanh([0.03, -0.06, 0.20] + [0.29, 0.22, 0.28]) = \tanh([0.32, 0.16, 0.48]) = [0.31, 0.16, 0.45]$

The attention score for h_1 is the sum = 0.31 + 0.16 + 0.45 = 0.92

(Similar calculations for h_2 through h_5)

Let's say we get these attention scores:

- $e_{1,1} = 0.92$
- $e_{1,2} = 1.54$
- $e_{1,3} = 0.78$
- $e_{1,4} = 1.21$
- $e_{1,5} = 0.65$

Step 2: Apply Softmax to Get Attention Weights

Softmax: $\alpha_{1,j} = \exp(e_{1,j}) / \Sigma^{k} \exp(e_{1,k})$

Total = exp(0.92) + exp(1.54) + exp(0.78) + exp(1.21) + exp(0.65) = 2.51 + 4.66 + 2.18 + 3.35 + 1.92 = 14.62

Hidden State	Attention Score	exp(score)	Attention Weight
h_1	0.92	2.51	2.51/14.62 = 0.172
h_2	1.54	4.66	4.66/14.62 = 0.319
h_3	0.78	2.18	2.18/14.62 = 0.149
h_4	1.21	3.35	3.35/14.62 = 0.229
h_5	0.65	1.92	1.92/14.62 = 0.131

Step 3: Calculate Context Vector

The context vector is a weighted sum of hidden states: $c_1 = \Sigma_j \alpha_{1,j} h_($

$c_1 = 0.172 \times [0.2, 0.5, -0.3] + 0.319 \times [0.6, 0.1, 0.4] + 0.149 \times [-0.1, 0.7, 0.2] + 0.229 \times [0.5, -0.2, 0.3] + 0.131 \times [0.3, 0.4, -0.5]$

$c_1 = [0.034, 0.086, -0.052] + [0.191, 0.032, 0.128] + [-0.015, 0.104, 0.030] + [0.115, -0.046, 0.069] + [0.039, 0.052, -0.066]$

$c_1 = [0.364, 0.228, 0.109]$

This context vector is then combined with the decoder state to predict the next word in the summary.

Evaluation Metrics: ROUGE Calculation

Example:

Original text: "Deep learning models require large datasets. They use neural networks with multiple layers to extract features. Computation is typically performed on GPUs."

Reference summary: "Deep learning uses multi-layer neural networks and needs large datasets and GPUs."

System summary: "Deep learning models use neural networks with multiple layers and require large datasets."

ROUGE-1 (Unigram) Calculation

Metric	Calculation	Result
Reference unigrams	{deep, learning, uses, multi-layer, neural, networks, and, needs, large, datasets, gpus}	11 unique words
System unigrams	{deep, learning, models, use, neural, networks, with, multiple, layers, and, require, large, datasets}	13 unique words
Overlap	{deep, learning, neural, networks, large, datasets}	6 words
Precision	6/13 = 0.462	46.2%
Recall	6/11 = 0.545	54.5%
F1-score	2 × (0.462 × 0.545) / (0.462 + 0.545) = 0.500	50.0%

ROUGE-2 (Bigram) Calculation

Metric	Calculation	Result
Reference bigrams	{deep learning, learning uses, uses multi-layer, multi-layer neural, neural networks, networks and, and needs, needs large, large datasets, datasets and, and gpus}	11 bigrams
System bigrams	{deep learning, learning models, models use, use neural, neural networks, networks with, with multiple, multiple layers, layers and, and require, require large, large datasets}	12 bigrams
Overlap	{deep learning, neural networks, large datasets}	3 bigrams
Precision	3/12 = 0.250	25.0%
Recall	3/11 = 0.273	27.3%

Metric	Calculation	Result
F1-score	2 × (0.25 × 0.273) / (0.25 + 0.273) = 0.261	26.1%

Transformer Self-Attention Calculation

For a simplified example with 3 input tokens with embeddings:

Token	Embedding (dim=3)
x_1	[0.5, 0.1, 0.3]
x_2	[0.2, 0.6, 0.4]
x_3	[0.8, 0.2, 0.7]

Step 1: Calculate Query, Key, and Value matrices

Using weight matrices (simplified):

- W^Q = [[0.1, 0.2, 0.1], [0.3, 0.1, 0.2], [0.2, 0.3, 0.1]]
- W^K = [[0.2, 0.1, 0.3], [0.1, 0.3, 0.1], [0.3, 0.2, 0.1]]
- W^V = [[0.3, 0.2, 0.1], [0.1, 0.3, 0.2], [0.2, 0.1, 0.3]]

Query matrix Q = XW^Q:

- Q_1 = [0.5, 0.1, 0.3] × W^Q = [0.13, 0.20, 0.14]
- Q_2 = [0.2, 0.6, 0.4] × W^Q = [0.22, 0.22, 0.22]
- Q_3 = [0.8, 0.2, 0.7] × W^Q = [0.29, 0.37, 0.21]

Similarly for K and V matrices.

Step 2: Calculate Attention Scores

QK^T = Q × K^T (matrix multiplication)

	K_1	K_2	K_3
Q_1	0.098	0.089	0.127
Q_2	0.142	0.165	0.176
Q_3	0.201	0.207	0.244

Scaling: QK^T/√d_k = QK^T/√3

	K_1	K_2	K_3
Q_1	0.057	0.051	0.073
Q_2	0.082	0.095	0.102
Q_3	0.116	0.119	0.141

Step 3: Apply Softmax

	K_1	K_2	K_3
Q_1	0.329	0.318	0.353
Q_2	0.321	0.333	0.346
Q_3	0.320	0.323	0.357

These are the attention weights showing how each token attends to all other tokens.

Step 4: Calculate Output

Output = softmax(QK^T/√d_k) × V

The final output for each position is a weighted combination of value vectors according to attention weights.

This self-attention mechanism allows tokens to attend to relevant parts of the input sequence regardless of distance, making it effective for summarization tasks.