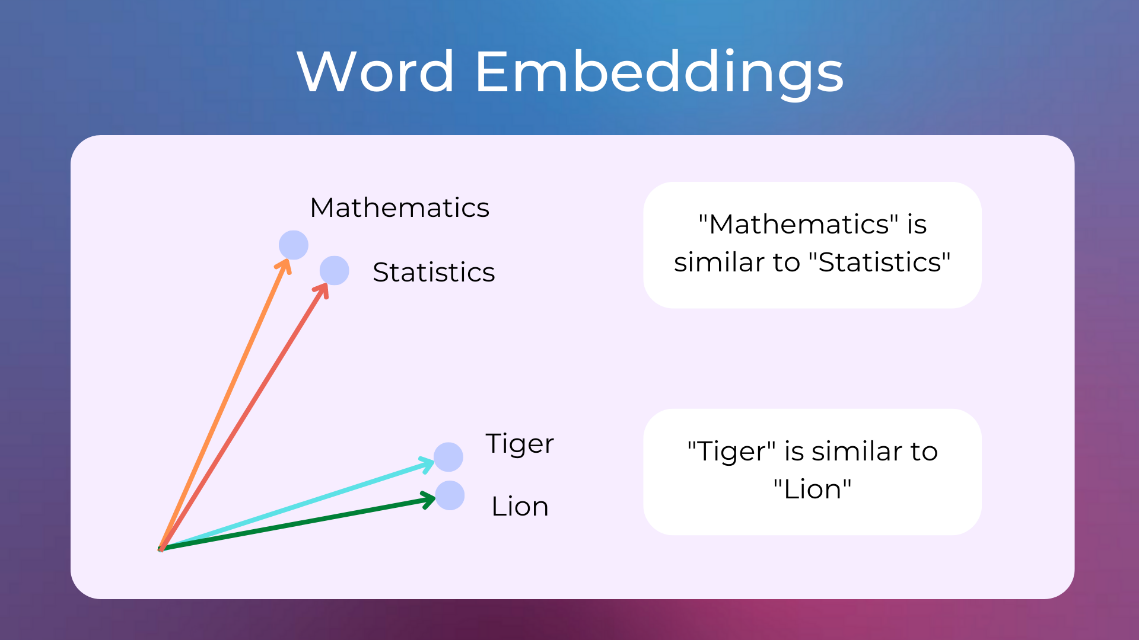
**EMBEDDING AND SELF ATTENTION MECHANISM**

Embeddings are numerical representations of real-world objects that [machine learning (ML)](https://aws.amazon.com/what-is/machine-learning/) and [artificial intelligence (AI)](https://aws.amazon.com/what-is/artificial-intelligence/) systems use to understand complex knowledge domains like humans do. As an example, computing algorithms understand that the difference between 2 and 3 is 1, indicating a close relationship between 2 and 3 as compared to 2 and 100. However, real-world data includes more complex relationships.

For example, a bird-nest and a lion-den are analogous pairs, while day-night are opposite terms. Embeddings convert real-world objects into complex mathematical representations that capture inherent properties and relationships between real-world data. The entire process is automated, with AI systems self-creating embeddings during training and using them as needed to complete new tasks.

****

**Why are embeddings important?**

Embeddings enable deep-learning models to understand real-world data domains more effectively. They simplify how real-world data is represented while retaining the semantic and syntactic relationships. This allows machine learning algorithms to extract and process complex data types and enable innovative AI applications. The following sections describe some important factors.

Reduce data dimensionality

Data scientists use embeddings to represent high-dimensional data in a low-dimensional space. In data science, the term *dimension* typically refers to a feature or attribute of the data. Higher-dimensional data in AI refers to datasets with many features or attributes that define each data point. This can mean tens, hundreds, or even thousands of dimensions. For example, an image can be considered high-dimensional data because each pixel color value is a separate dimension.

When presented with high-dimensional data, deep-learning models require more computational power and time to learn, analyze, and infer accurately. Embeddings reduce the number of dimensions by identifying commonalities and patterns between various features. This consequently reduces the computing resources and time required to process raw data.

Train large language models

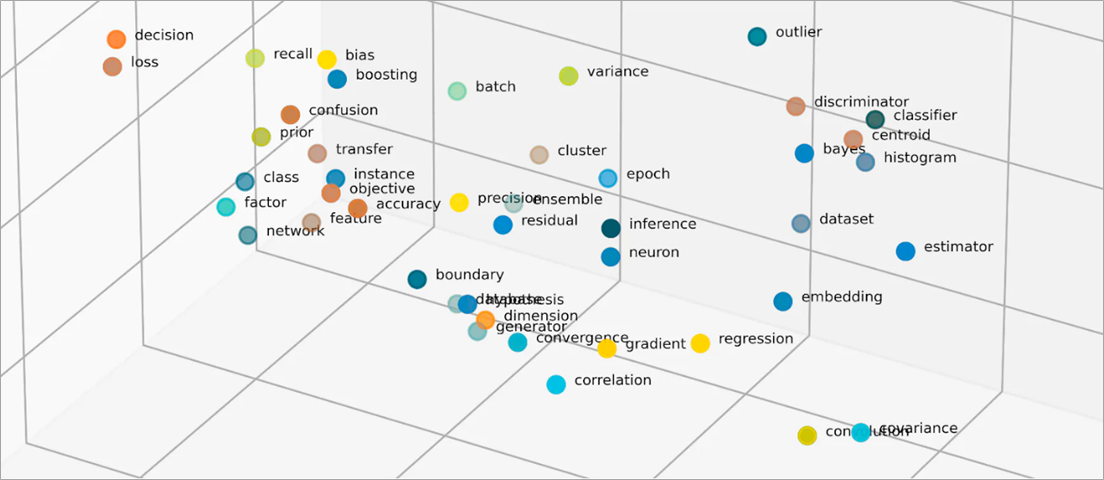
Embeddings improve data quality when training [large language models (LLMs)](https://aws.amazon.com/what-is/large-language-model/). For example, data scientists use embeddings to clean the training data from irregularities affecting model learning. ML engineers can also repurpose pre-trained models by adding new embeddings for transfer learning, which requires refining the foundational model with new datasets. With embeddings, engineers can fine-tune a model for custom datasets from the real world.

Build innovative applications

Embeddings enable new [deep learning](https://aws.amazon.com/what-is/deep-learning/) and generative artificial intelligence (generative AI) applications. Different embedding techniques applied in neural network architecture allow accurate AI models to be developed, trained, and deployed in various fields and applications. For example:

* With image embeddings, engineers can build high-precision computer vision applications for object detection, image recognition, and other visual-related tasks.
* With word embeddings, natural language processing software can more accurately understand the context and relationships of words.
* Graph embeddings extract and categorize related information from interconnected nodes to support network analysis.

Computer vision models, [AI chatbots](https://aws.amazon.com/what-is/chatbot/), and AI recommender systems all use embeddings to complete complex tasks that mimic human intelligence.



**What are vectors in embeddings?**

ML models cannot interpret information intelligibly in their raw format and require numerical data as input. They use neural network embeddings to convert real-word information into numerical representations called vectors. Vectors are numerical values that represent information in a multi-dimensional space. They help ML models to find similarities among sparsely distributed items.

Every object an ML model learns from has various characteristics or features. As a simple example, consider the following movies and TV shows. Each is characterized by the genre, type, and release year.

*The Conference (Horror, 2023, Movie)*

*Upload (Comedy, 2023, TV Show, Season 3)*

*Tales from the Crypt (Horror, 1989, TV Show, Season 7)*

*Dream Scenario (Horror-Comedy, 2023, Movie)*

ML models can interpret numerical variables like years, but cannot compare non-numerical ones like genre, types, episodes, and total seasons. Embedding vectors encode non-numerical data into a series of values that ML models can understand and relate. For example, the following is a hypothetical representation of the TV programs listed earlier.

*The Conference (1.2, 2023, 20.0)*

*Upload (2.3, 2023, 35.5)*

*Tales from the Crypt (1.2, 1989, 36.7)*

*Dream Scenario (1.8, 2023, 20.0)*

The first number in the vector corresponds to a specific genre. An ML model would find that *The Conference* and *Tales from the Crypt*share the same genre. Likewise, the model will find more relationships between *Upload*and *Tales from the Crypt*based on the third number, representing the format, seasons, and episodes. As more variables are introduced, you can refine the model to condense more information in a smaller vector space.

A blue and white rectangular signs with white text

AI-generated content may be incorrect.

**How do embeddings work?**

Embeddings convert raw data into continuous values that ML models can interpret. Conventionally, ML models use one-hot encoding to map categorical variables into forms they can learn from. The encoding method divides each category into rows and columns and assigns them binary values. Consider the following categories of produce and their price.

|  |  |
| --- | --- |
| Fruits | Price |
| Apple | 5.00 |
| Orange | 7.00 |
| Carrot | 10.00 |

Representing the values with one-hot encoding results in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| Apple | Orange | Pear | Price |
| 1 | 0 | 0 | 5.00 |
| 0 | 1 | 0 | 7.00 |
| 0 | 0 | 1 | 10.00 |

The table is represented mathematically as vectors [1,0,0,5.00], [0,1,0,7.00], and [0,0,1,10.00].

One-hot encoding expands dimensional values of 0 and 1 without providing information that helps models relate the different objects. For example, the model cannot find similarities between *apple*and *orange*despite being fruits, nor can it differentiate *orange*and *carrot* as fruits and vegetables. As more categories are added to the list, the encoding results in sparsely distributed variables with many empty values that consume enormous memory space.

Embeddings vectorize objects into a low-dimensional space by representing similarities between objects with numerical values. Neural network embeddings ensure that the number of dimensions remains manageable with expanding input features. Input features are traits of specific objects an ML algorithm is tasked to analyze. Dimensionality reduction allows embeddings to retain information that ML models use to find similarities and differences from input data. Data scientists can also visualize embeddings in a two-dimensional space to better understand the relationships of distributed objects.

A screenshot of a computer

AI-generated content may be incorrect.

**What are embedding models?**

Embedding models are algorithms trained to encapsulate information into dense representations in a multi-dimensional space. Data scientists use embedding models to enable ML models to comprehend and reason with high-dimensional data. These are common embedding models used in ML applications.

Principal component analysis

*Principal component analysis* (*PCA*) is a dimensionality-reduction technique that reduces complex data types into low-dimensional vectors. It finds data points with similarities and compresses them into embedding vectors that reflect the original data. While PCA allows models to process raw data more efficiently, information loss may occur during processing.

Singular value decomposition

*Singular value decomposition* (*SVD*) is an embedding model that transforms a matrix into its singular matrices. The resulting matrices retain the original information while allowing models to better comprehend the semantic relationships of the data they represent. Data scientists use SVD to enable various ML tasks, including image compression, text classification, and recommendation.

Word2Vec

Word2Vec is an ML algorithm trained to associate words and represent them in the embedding space. Data scientists feed the Word2Vec model with massive textual datasets to enable natural language understanding. The model finds similarities in words by considering their context and semantic relationships.

There are two variants of Word2Vec—Continuous Bag of Words (CBOW) and Skip-gram. CBOW allows the model to predict a word from the given context, while Skip-gram derives the context from a given word. While Word2Vec is an effective word embedding technique, it cannot accurately distinguish contextual differences of the same word used to imply different meanings.

BERT

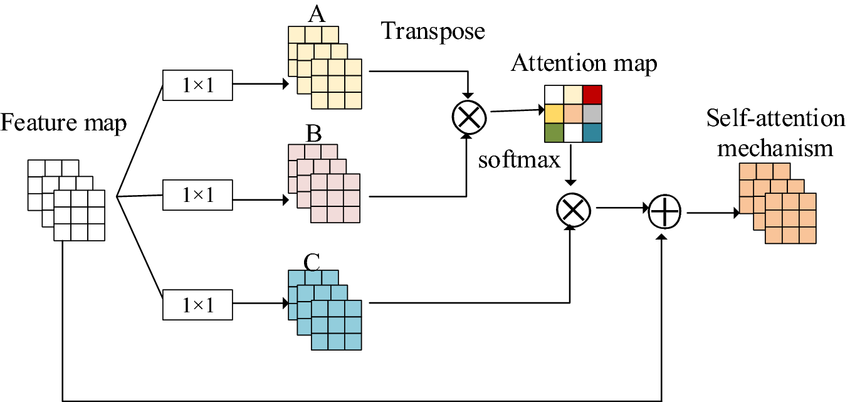
BERT is a transformer-based language model trained with massive datasets to understand languages like humans do. Like Word2Vec, BERT can create word embeddings from input data it was trained with. Additionally, BERT can differentiate contextual meanings of words when applied to different phrases. For example, BERT creates different embeddings for ‘play’ as in *“I went to a play”* and *“I like to play.”*

How are embeddings created?

Engineers use [neural networks](https://aws.amazon.com/what-is/neural-network/) to create embeddings. Neural networks consist of hidden neuron layers that make complex decisions iteratively. When creating embeddings, one of the hidden layers learns how to factorize input features into vectors. This occurs before feature processing layers. This process is supervised and guided by engineers with the following steps:

1. Engineers feed the neural network with some vectorized samples prepared manually.
2. The neural network learns from the patterns discovered in the sample and uses the knowledge to make accurate predictions from unseen data.
3. Occasionally, engineers may need to fine-tune the model to ensure it distributes input features into the appropriate dimensional space.
4. Over time, the embeddings operate independently, allowing the ML models to generate recommendations from the vectorized representations.
5. Engineers continue to monitor the performance of the embedding and fine-tune with new data.

**Self – Attention Mechanism**

****

Self-attention is a mechanism that helps machine learning models, especially transformers, understand relationships between words in a sentence by weighing their importance. It overcomes the limitations of CNNs and RNNs by allowing parallel processing instead of step-by-step computation. This enables faster training, efficient use of computational resources, and better handling of long sequences. Self-attention is key to modern large language models (LLMs) and is widely used in NLP tasks like translation, sentiment analysis, and summarization.

The self-attention mechanism was introduced by means of the transformer, a model [neural network](https://www.ibm.com/think/topics/neural-networks) architecture proposed by researchers. The aim of the proposed architecture was to address the challenges of traditional machine learning models that use convolution neural networks (CNNs) and recurrent neural networks (RNNs).[1](https://www.ibm.com/think/topics/self-attention#f01)

Traditional sequential models follow the same encoder-decoder architecture as [transformer models](https://www.ibm.com/think/topics/transformer-model) but process data step-by-step or sequence-to-sequence (seq2seq). This function poses a challenge for parallelization, which is the ability to reduce computation time and enhance output generation by calculating attention weights across all parts of the input sequence simultaneously.

**How does self-attention work?**

Self-attention in machine learning models is similar to the human behavioral concept in that they both involve focusing on relevant elements within a larger context to accurately process information. In psychology, it is about focusing on your own thoughts or behaviors, while in [deep learning](https://www.ibm.com/think/topics/deep-learning), it is about focusing on the relevant parts of an input sequence.

The transformer architecture includes a self-attention layer where the attention process is integrated. The steps are explained as presented in the paper by Ashish Vaswani et al. introducing the self-attention layer “Attention is All You Need.”

* Embedding the input sequence

An input sequence is a series of data points vectorized into embeddings, or numerical representations, that the machine learning algorithm can use to calculate attention scores needed to produce an output sequence.

In machine translation, a sentence would be considered an input sequence, where each part of the sentence is considered a data point or input token. Tokens are converted into embeddings that act as semantic units that the model can process.[2](https://www.ibm.com/think/topics/self-attention#f02) The embeddings are used to calculate the attention weights that help the model prioritize (or attend to) the most relevant input data.

* Generate vectors for the attention function

The model uses these embeddings to generate three key vectors for each token: query (Q), key (K) and value (V). These values will be used to help the model make the strongest semantic matches within the input sentence.

Matrix multiplications are performed to obtain the query, key and value vectors. The attention mechanism calculates a weighted sum of the values based on the respective query, key and value components’ weight matrices and embedded inputs.[1](https://www.ibm.com/think/topics/self-attention#f01) This process is known as linear transformation.

* Compute the attention scores

After the embeddings are transformed, attention scores for each element in the sequence are calculated. The attention scores are obtained by taking the scaled dot product attention scores between the query vectors and key vectors. These attention weights represent how much focus (or attention) a specific token should give to other tokens in a sequence.

Next, the attention score is scaled by the square root of the dimensionality of the key vectors. This process helps to stabilize the gradients and prevent them from growing too large to compute efficiently as the dimensionality of the vectors increases.

* Transform the attention scores into probabilities

The attention scores obtained through the dot product of the query vectors and key vectors are transformed into probabilities by using the softmax function. This process is called normalization.

With these normalized probabilities, the softmax attention block allows the transformer architecture the ability to evaluate the importance of individual input elements during output generation.[3](https://www.ibm.com/think/topics/self-attention#f03) These probabilities are used to find the relative importance of each element in the sequence. The attention model uses these normalized weights to decide which parts of the input to focus on.

Finally, the attention weights derived from this process contribute to the final weighted sum of the value vector. The higher the attention score, the more attention weight the sequence has. This means it will have more influence on the final output of the value vector’s weighted sum.

**How Attention models improve context understanding?**

Attention-based transformer models dramatically enhance a system’s understanding of context by capturing long-range dependencies, even when related words are far apart in a sentence. The multi-head attention mechanism extends this capability by making the model “look at” different relationships simultaneously — one head might focus on syntax, another on semantic meaning, and another on relationships across words. This parallel processing allows the model to draw deeper context from text, improving performance across tasks like translation, summarization, and reasoning.[Medium](https://medium.com/%40tungvu_37498/understanding-transformer-architecture-the-brains-behind-modern-ai-ec3c99f0baed?utm_source=chatgpt.com)

Furthermore, different architectures leverage attention in unique ways. Bidirectional models like BERT treat context from both directions, offering a more complete understanding of each word. Generative models like GPT-3 expand context windows, enabling the processing of much longer sequences and enhancing text generation quality. This flexibility — combining attention, bidirectionality, and expanded context — empowers modern transformer-based models to understand and generate human-like language with unprecedented accuracy.

A diagram of a diagram

AI-generated content may be incorrect.

**Use cases**

**NLP tasks:**The self-attention mechanism enhances the linguistic capabilities of machine learning models by allowing the efficient and complete analysis of an entire text. Research has shown advancements in sentiment classification.[6](https://www.ibm.com/think/topics/self-attention#f06) Models can perform NLP tasks well because the attention layer allows it to compute the relation between words regardless of the distance between them.[7](https://www.ibm.com/think/topics/self-attention#f07)

[**Computer vision**](https://www.ibm.com/think/topics/computer-vision): Self-attention mechanisms are not exclusive to NLP tasks. It can be used to focus on specific parts of an image. Developments in image-recognition models suggest that self-attention is a crucial component to increase their robustness and generalization.