Generative Artificial Intelligence

Prof. James L. Crowley[1]

Grenoble Institut Polytechnique

Univ. Grenoble Alpes

Abstract:

Generative Artificial Intelligence is a technology for building systems that exhibit or exceed human level performance at a large variety of tasks. The first generation of such systems have included tools for Natural Language Processing that empower conversational agents such as ChatGPT, Llama, Claude, CoPilot and similar systems. These systems are increasingly embedded in systems that combine speech recognition and computer vision with realistic speech and image generation to provide multimodal spoken language interaction with graphically rendered artificial agents. However, this is only the beginning of a growing wave of revolutionary new technologies that will soon include personal digital assistants on mobile devices, fully-autonomous self-driving automobiles, humanoid robots, and many other autonomous intelligent systems.

This chapter provides an introduction to the key enabling technologies that make such systems possible. This introduction covers the history and variations of the autoencoder, embeddings for natural language, residual learning, and attention, as well as a discussion of the architectures of Transformers and Large Language Models. Each of these topics is accompanied by historical references as well as summaries of mathematical principles on which each technology is based.

This introduction is based on the contents of a course on machine learning taught to 3rd year engineering students at the ENSIMAG engineering school of Informatics and Applied Mathematics at the Univ. Grenoble Alpes in Grenoble, France. It assumes a familiarity with training neural networks using back-propagation, as well as a basic understanding of probability and statistics. This chapter is intended to provide a foundation for understanding the potential impacts and risks associated with human centered artificial intelligence.

A note on the mathematics: The linear algebra used in this introduction is expressed using column-vector notation, in order to align more naturally with the classical formulations found in engineering disciplines such as signal processing, control theory, robotics, and computer vision. In this convention, linear transformations act on vectors from the left, supporting a clear interpretation of transformations as compositions of functions. Column-vector notation also preserves consistency with standard mathematical notation and simplifies reasoning about coordinate systems, changes of basis, and hierarchical transformations.

This contrasts with much of the modern machine learning literature, which adopts row-vector conventions. Row-vector representations align naturally with data organized as batches of samples to enable highly efficient operations on modern hardware, particularly GPUs.

A glossary of mathematical symbols may be found at the end of the chapter.

[1] James L. Crowley
Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, 38000 Grenoble, France

Contents

1. Introduction 3
2. Key Enabling Technologies 3
3. The Autoencoder 4
3.1 The Key Idea 4
3.2 The Denoising Autoencoder 5
3.3 The Sparse Autoencoder 5
3.4 Variational Autoencoders 7
4. Embeddings 8
4.1 The Key Idea 8
4.2 Embeddings for Linguistic Data. 9
5. Attention 12
5.1 The Key Idea 12
5.2 Attention in Machine Learning 13
5.3 Self-Attention in Transformers 13
6. Residual Learning 14
6.1 The Key Idea 15
6.2 Gradient Stability 15
7. Transformers 16
7.1 The Key Idea 16
7.2 Architectural Details 17
7.3 BERT - Bidirectional Encoder Representations from Transformers 18
7.4 GPT and Large Language Models 19
7.5 ChatGPT 20
7.6 Vision Transformer (ViT) 21
7.7 Auditory Transformers 21
7.8 Multimodal Transformers 21
8. Conclusions 22
Bibliography 23
Index 25
Mathematical Notation 26

Keywords:

Machine Learning, Generative Artificial Intelligence, Autoencoders, Embeddings, Attention, Transformers, Large Language Models

1. Introduction

Generative Artificial Intelligence (Generative AI) is a class of artificial intelligence systems that learn patterns from data and use the result to generate new content. Generative AI is increasingly recognized as a rupture technology with potential for profound economic and social impact. The most immediate impacts have been in the area of language technologies, with near-term applications including real-time machine translation for multi-lingual interaction, natural language interfaces for machines and systems, and intelligent assistants for creating prose, sounds, images, movies, computer programs, or other content in reaction to natural language requests. However, natural language processing is only the first of many potential application areas.

Generative AI makes it possible to build artificial systems that can interpret and generate realistic images, video, sounds, music, tactile and haptic signals, and support multimodal interaction that complements natural language with visual, auditory, tactile and haptic sensory-motor modalities. Generative AI is well suited for use with reinforcement learning, particularly in intelligent robotics applications involving manipulation and assembly, as well as driving, piloting, and navigation for autonomous ground, aerial, and underwater vehicles. Generative AI can be used for mechanical design, chemical synthesis, material science and even genetic engineering.

A practical technology for Generative AI became available with the invention of the Transformer by a team at Google in 2017 (Vaswani et al 2017). Transformers combine a discriminative component (an encoder) with a generative component (a decoder), communicating through a small number of hidden latent variables (a code). In most useful systems, the components are implemented as neural networks trained with self-supervised learning, a form of unsupervised learning in which the system learns by reconstructing raw, unlabeled training data. The hidden latent variables are trained using a loss function that encourages statistical independence. As a result, the latent variables learn to represent independent phenomena.

Most useful generative systems combine multiple layers of encoders and/or decoders, using learned attention functions to focus the encoding and decoding components on relevant phenomena. While many architectures employ both encoders and decoders, autoregressive Transformer models such as GPT that use only stacks of decoders have proven particularly effective for natural language conversation. In either case, the result is a learned hierarchy of latent variables that can represent ever more abstract information over ever-larger spatial, temporal and conceptual contexts. By training on massive amounts of raw unstructured data, such systems can learn to interpret and react to novel input, to create meaningful original output, and to exhibit human-level performance for a variety of tasks.

2. Key Enabling Technologies

Generative AI is made possible by several key-enabling technologies. These include the autoencoder, domain-specific encodings (embeddings) for symbolic and numerical data, residual learning, learned self-attention for data association, and Transformer architectures trained with self-supervised learning. Each of these key-enabling technologies is briefly described below, with pointers to the literature for readers interested in a more complete treatment. This is intended to provide a quick introduction for use as background for later chapters on applications and ethical challenges for human centered Artificial Intelligence.

The most important key enabling technology for Generative Artificial Intelligence is the use of backpropagation for training neural networks. However, machine learning with neural networks is now a well-established technology, and tutorials on neural networks and backpropagation are widely available on the internet. For example, a tutorial entitled Machine Learning with Neural Networks may be found in (Crowley 2021). The following section provides a brief overview, with historical background, on the other enabling technologies. We begin with a key idea that illustrates the potential for generative systems: The Autoencoder.

3. The Autoencoder

Neural networks were originally invented for recognition. When used for recognition, a network is trained to map a signal into a category label taken from a discrete set of possible categories. Such networks are called discriminative networks. However, with sufficient data and computing, networks can also be trained to generate signals from category labels. This is called a generative network. Generative networks can be trained to generate realistic examples for any signal including natural language text, images, sounds, video, and motor commands for coordinated robot motions. A discriminative network can be paired with a generative network to create an autoencoder. Autoencoders are a key enabling technology for Transformers and Large Language Models.

3.1 The Key Idea

The key idea for an autoencoder is to combine a discriminative network with a generative network, communicating via a compact latent code vector, as shown in figure 1. The discriminative network encodes an input vector as a compact code vector $\vec{Z}$. The latent code vector is then used to drive a generative network to produce an associated output vector $\vec{Y}$. Classically, the generator is trained to generate an output vector $\vec{Y}$ that is a reconstruction of the input, $\vec{X}$. However generators can be trained to associate any useful output with an input. Autoencoders can be used to associate words with images, sounds with words, or even translate words between languages. This provides a powerful tool for multimodal perception and interaction and a key enabling technology for artificial intelligence.

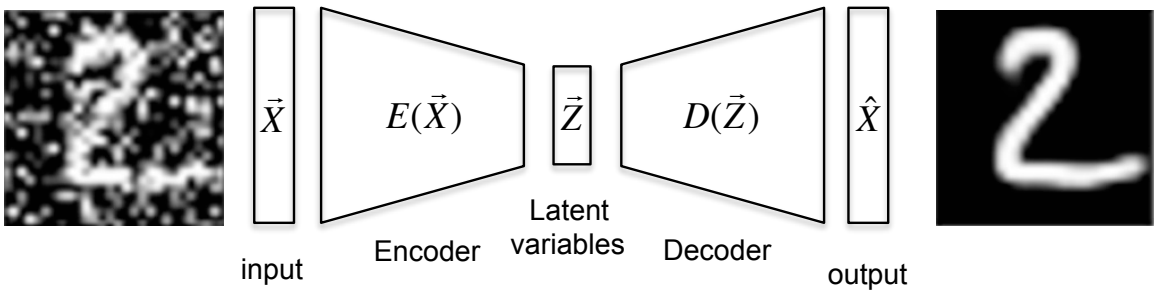


Fig. 1 A denoising autoencoder combines a discriminative encoder with a generative decoder to create a clean (denoised) copy of an input signal.

Autoencoders are a form of Latent-Variable Model in which the model learns to estimate values for a small number of latent code variables that can be used to reconstruct data in a training set. The code variables are referred to as latent because they encode what remains when random variations are removed. The latent variables preserve the patterns that are common to all of the training data while ignoring random variations (noise). Latent variable models are easily implemented as neural networks trained with backpropagation.

Autoencoders can be trained with a form of unsupervised learning in which a model is trained by reconstructing the desired output from corrupted version of training samples. The process of learning by systematically corrupting samples of training data is referred to as self-supervised learning. Any data set can be used for self-supervised learning without the need for additional ground truth labeling. The data is its own ground truth.

In the early days of research on perceptron learning, autoencoders were used to compensate for a lack of labeled training data for experiments with the backpropagation algorithm (Ackley et al 1985). Autoencoders were trained using a loss function, $L(\vec{X}_m)$, based on the squared-difference between the input vector, $\vec{X}_m$, and the output, $\vec{Y}_m$.

$$L(\vec{X}_m) = \frac{1}{2}(\vec{Y}_m - \vec{X}_m)^2$$

This is referred to as a least-squares loss function. When trained with a least-squares loss function, the discriminative network converges towards a form of principal components analysis, and the latent variables represent the principal components of the training data. This provides a form of autoencoder referred to as a denoising autoencoder. Neural autoencoders provided a new, incremental method for estimating the principal components for a data set, allowing the results to be updated as new training data was acquired.

3.2 The Denoising Autoencoder

The denoising autoencoder learns to represent data in a training set with a small compact code vector that represents the variations (or energy) in the training set with as few latent variables as possible. This is accomplished by minimizing the squared-error between an input vector $\vec{X}$ and an estimated output vector $\hat{X}$. This is easily implemented with a two layer fully connected neural network trained with a classic least-squares loss function.

Consider a simple fully-connected 2-layer autoencoder that learns to reconstruct data from a training set $\{\vec{X}_m\}$ of M training samples each with d features. Assume a code vector, $\vec{Z}$, of $n<<d$ latent variables, z_i. For any input vector $\vec{X}_m$, the n coefficients of the code vector, $\vec{Z}_m$, are computed by a fully connected encoder layer, $E(\vec{X}_m)$.

$$\vec{Z}_m = E(\vec{X}_m) = f(W^{(1)}\vec{X}_m + \vec{B}^{(1)})$$

where $W^{(1)}$ is a matrix of weights and $\vec{B}^{(1)}$ is a vector of biases for the encoder, and *f(-)* is a non-linear activation function, such as sigmoid or RELU. The autoencoder reconstructs an approximation with a second, generator layer, $D(\vec{Z}_m)$.

$$\hat{X}_m = D(\vec{Z}_m) = f(W^{(2)}\vec{Z}_m + \vec{B}^{(2)})$$

The error for using $\hat{X}_m$ to represent $\vec{X}_m$ is $\delta_m = \hat{X}_m - \vec{X}_m$. The square of this error can be used to define a cost or loss function for training the network with back-propagation.as shown in Figure 2.

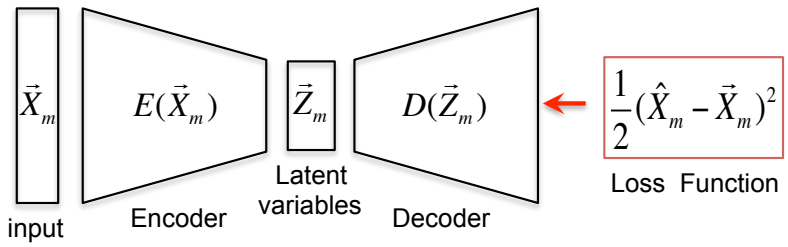


Fig 2. A denoising autoencoder uses a least-square loss function to learn a latent variable model for data using stochastic gradient descent.

When trained with a squared-error loss function, the autoencoder converges to a form of Principal Components Analysis (PCA) of the data in the training set, and the latent variables express the principal components of the training data. Principal components analysis seeks to represent the training data by concentrating energy in the smallest number of latent variables. However, this distributes the energy for each training sample over multiple latent variables. While this is useful for noise removal, the latent variables cannot be used as independent categories. For category learning, a more useful code can be obtained by forcing the network to learn independent categories, providing a minimum description-length code for training data (Hinton and Zemel 1993). This is now referred to as a sparse autoencoder.

3.3 The Sparse Autoencoder

A sparse autoencoder can be obtained by adding an additional term to the loss function that includes an information theoretic measure for the uniqueness of each code variable. This additional term is

referred to as sparsity. Sparsity measures the independence of the latent variables by computing the average absolute value of the activation energy for each of the N hidden latent units when trained with a batch of M training samples.

The activation for the n^{th} latent variable for m^{th} data points is $z_m(n)$. When trained with a batch of M data samples, the average absolute value of the n^{th} latent variable is

$$\hat{\rho}(n) = \frac{1}{M}\sum_{m=1}^{M}\left|z_m(n)\right|$$

Independent latent variables will be close to zero for most inputs, responding only to examples of their associated category. If the vector of latent variables is normalized to sum to 1 with a soft-max, then the sparsity will reflect the percentage of training samples for each category. When trained with a balanced training set containing the same number of examples of each category, independent latent variables will tend towards a constant distribution of sparsities. Thus sparsity can be used as a regulariser to bias the loss towards independent latent variables during training. The absolute value (L_1 norm) is used rather than a squared value (L_2 norm) to prevent sparsity from dominating the loss function.

A sparse autoencoder can be created from a denoising autoencoder by adding an additional term to the loss function using the Kullback-Leibler (KL) divergence. The KL divergence is used to compare the distribution of sparsity values $\hat{\rho}(n)$ for the N latent variables to a constant target value, ρ.

$$KL(\rho \,\|\, \hat{\rho}(n)) = \sum_{n=1}^{N}\rho\log\left(\frac{\rho}{\hat{\rho}(n)}\right) + (1-\rho)\log\left(\frac{1-\rho}{1-\hat{\rho}(n)}\right)$$

The KL divergence measures the relative entropy between two probability distributions (tables of probabilities). This gives a loss function $L(\vec{X}_m; w)$ for training with a data sample, $\vec{X}_m$ as:

$$L(\vec{X}_m; w) = \frac{1}{2}(\hat{X}_m - \vec{X}_m)^2 + \beta \cdot KL\left(\rho \,\|\, \hat{\rho}(n)\right)$$

where $w = \left\{W^{(1)}, W^{(2)}, \vec{B}^{(1)}, \vec{B}^{(2)}\right\}$ represents the parameters of the network, and β is a hyper-parameter that sets the weight of penalty for absence of sparsity.

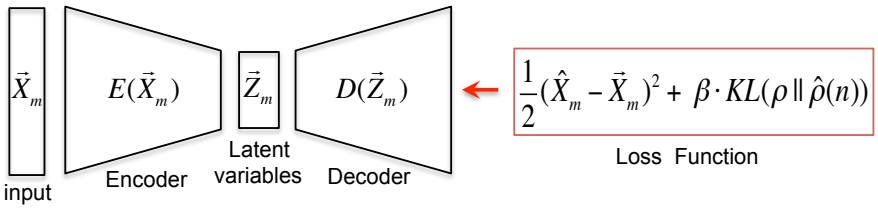


Fig. 3 A sparse autoencoder uses batch learning with a loss function that includes a comparison of the distribution of sparsities $\hat{\rho}(n)$ of the latent variables to a target sparsity, ρ.

Including the loss term for absence of sparsity of the latent variables is slightly more expensive than simple least-squares error, as it requires a pass through the N latent variables to determine the distribution of sparsities.

Combining loss terms for sparsity and least-squares error of reconstruction as a weighted sum makes it possible to use the sparse autoencoder for unsupervised category learning with unlabeled training data. However, this is not strictly necessary. When trained with labeled training data, the encoder can first be trained with supervised learning to recognize each category as a latent variable,

using the known category as the true latent variable. The resulting encoder can then be used to train a decoder using the least-squares error of the reconstruction as a loss function for back-propagation.

Consider, for example, the case of the popular MNIST data set composed of handwritten samples of the 10 digits sampled on a 28 by 28 pixel grid. An encoder can be trained using supervised learning to represent each of the 10 digits as one of ten latent variables. The decoder can then be trained to reconstruct the handwritten digits for each digit using the least-squares reconstruction error. Training in this way is much more efficient than with unsupervised learning. In addition, because the identity of each of the latent variables is known, it is possible to train both the encoders and decoders for multiple modalities (Vazquez-Rodriguez et al 2022). This can be used, for example, to recognize and generate spoken digits in multiple languages associated with images of each handwritten digit.

With sufficient computing and data, the encoders and decoders can be stacked hierarchically, with each layer of the hierarchical encoder using the latent variables from lower layers to learn more abstract encodings, and each layer of the decoder generating patterns that are based on a larger context. Each new layer learns categories of categories, and can be used to represent concepts with a form of deep concept learning. This is the idea behind the Transformer. An interesting challenge is to use this hierarchy to generate original output. This can be accomplished with a variational autoencoder.

3.4 Variational Autoencoders

For some problems the challenge is to learn to generate original output that is similar to, but not identical to the training data. This can be useful, for example, in computer graphics for generating examples of leaves for an image of a tree, or of trees for generating a forest. It can also be used to generate smooth motion between configurations of a complex figure, such as a dancer. This can be provided by a variational autoencoder (Kingma and Welling 2014).

Variational Autoencoders (VAEs) are a form of generative model that uses a probability distribution $P(\vec{Z},\vec{X})$ to represent how likely each latent variable z_n, is for an input $\vec{X}$. This distribution can then be used to generate random approximations $\vec{Y}$ of the input. For example, $\vec{X}$ may be an image patch, in which each pixel is an independent dimension, and $P(\vec{Z},\vec{X})$ describes a lower-dimensional manifold representing a set of imagettes with similar appearance. In a variational autoencoder, the encoder learns to approximate this joint distribution as a density function, and the output is created by sampling this density. This is commonly modeled as a Gaussian or Normal density function, using estimated distributions for the mean $\mu(\vec{Z},\vec{X})$ and Covariance $\Sigma(\vec{Z},\vec{X})$ of the latent variables. These can be computed from the joint probability distribution $P(\vec{Z},\vec{X})$ by fully-connected networks, implemented as matrix multiplications. Other density functions are also possible. For example, binary patterns can be represented with Bernoulli functions. A Dirac function gives an exact deterministic reconstruction.

For the problem of the recognizing handwritten digits of the MNIST data set, each of the 10 digits corresponds to low-dimensional manifold in a 28 x 28 = 784 dimensional input space. The variational autoencoder learns to model these manifolds as 784 dimensional probability distributions, representing each digit as a mean and a covariance. Examples of each category can then be generated by randomly sampling the corresponding density. The encoder serves to estimate the mean and covariance, and the decoder projects these onto an output, $\hat{Y}$, that approximates the digit as shown in figure 4.

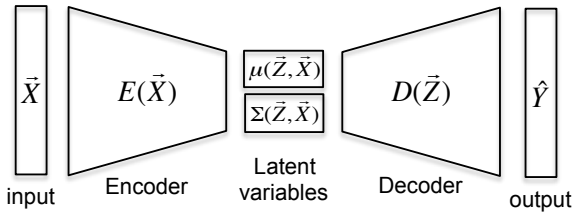


Fig. 4 A variational autoencoder learns parameters for a joint probability density function that represents the joint distribution $P(\vec{Z},\vec{X})$. This density can then be sampled and used by a generator to generate an output vector, $\hat{Y}$ that approximates a desired output, $\vec{Y}$.

Variational autoencoders can be used to generate new images of handwritten digits, or to generate a representation in another domain such as written or spoken word. Variational autoencoders have been trained to generate examples of original faces, hands and bodies, render images in the style of an artist, and generate possible motions from static images. They can also be used to generate natural language text and even to fabulate stories with large language models. However, this requires a way to express natural language as numbers. This can be done using an embedding for natural language.

4. Embeddings

Embeddings are encodings for data that enable machine-learning models to efficiently recognize and generate information. Embeddings exist for both numeric and symbolic data and have been developed for a variety of domains including Natural Language Processing, Speech Recognition and Computer Vision. Embeddings for symbolic information, in particular, have made it possible to apply machine learning to solve many classically hard problems in Natural Language Processing.

In classic statistical pattern recognition, an embedding is a form of feature space (Duda and Hart 1973). Much of the pattern recognition literature of the 20th century has been devoted to discovery of feature spaces that can reveal meaningful patterns in data. Classically, feature spaces were developed for numerical data. Development of effective embedding for symbolic and textual data have made it possible to develop machine learning and artificial intelligence technologies that can understand text, write computer programs and converse with humans using natural language.

4.1 The Key Idea

An embedding is a numerical representation that projects a high dimensional representation for data into a lower dimensional space that reveals properties or relations that facilitate interpretation. Embeddings make it possible to detect and classify patterns in data and to associate information from data with information acquired at other times or places.

As a classic example of an embedding, consider a black and white (gray-scale) image. Each pixel represents an independent dimension. An 8 x 8 image patch from the image represents a point in a 64 dimensional space. The position of this point is a unique signature for the appearance of the patch, and similar patches will generally map to nearby points in this 64 dimensional space. However, most of this space will be empty.

Principal Components Analysis (PCA) can be used to learn a low dimensional linear subspace that preserves similarity of appearance and can reveal meaningful information such as changes in shape, illumination or camera position. In the early 1990's, Computer Vision researchers demonstrated that PCA could be used compute low-dimensional linear sub-space representations for collections of images to reveal information about objects in the scene. PCA analysis was used for face detection and recognition (Turk and Pentland 1991), facial expression analysis (Schwerdt et al 2000), robot position estimation (Crowley et al 1998), and many other image analysis problems.

Histograms

Histograms are a classic feature space used for data analysis. A histogram is a simple count of the frequency of occurrence for the possible values for a set of data. Frequency of occurrence can be used to compute the probability of encountering other data with a similar value. The idea of using the histograms of frequency of occurrence for computing probability distributions appears in Claude Shannon's foundational paper on the mathematical theory of communications in 1948 (Shannon 1948).

Histograms can be computed for any data composed of elements with a finite set of values, and are often used for symbolic data. Histograms can be used with scalar or vector quantities with integer values over some finite range including words, or symbols from a finite vocabulary as well as integers from a finite range. For example, consider a vocabulary composed of N unique words. A hash *h(w)* can be used to associate each word, *w*, with a count of how many times the word in a document. Given the number of words in the document, *M*, the hash can be used to compute the probability of occurrence, *P(w)*, for each word:

$$P(w) = \frac{1}{M} h(w)$$

Given *D* documents, each with M_d words, the hash can be extended to give a count, *h(w,d)*, of the number of times each word occurs in each document *d.* This can be used to compute the conditional probability of encountering a word *w* in document *d* as:

$$P(w \mid d) = \frac{1}{M_d} h(w,d)$$

With these values we can compute the probability of having a document given a word:

$$P(d \mid w) = \frac{h(w,d)}{h(w)} = \frac{h(w,d)}{\sum_{d=1}^{D} h(w,d)}$$

Given a word, *w*, the vector of probabilities in a set of *D* documents provides a unique feature vector, or fingerprint, $\vec{K}$ that captures the contexts in which the word is used.

$$\vec{K}_w = \begin{pmatrix} P(d=1 \mid w) \\ \vdots \\ P(d=D \mid w) \end{pmatrix}$$

This feature vector can be used to associate words with similar meanings. Words with similar contexts have similar fingerprints. This can be used to determine context for interpreting a sentence or paragraph.

4.2 Embeddings for Linguistic Data.

Embeddings for linguistic data have made it possible to apply generative systems to generation and translation of natural language. This has made possible the recent revolutionary progress in Natural Language Processing systems with Large Language Models. Advances began with classic technique known as a Bag-of-Words (BoW).

Bag-of-Words

For natural language processing, symbolic text must be transformed into a numerical representation in order to apply machine-learning techniques. Classically this has been performed with a Bag-of-Words representation (Harris 1954). A Bag-of-Words (BoW) is an orderless statistical representation, where each word is associated with a count of the frequency of occurrence of the

word in a text. This allows the use of Bayes rule to compute the likelihood that a query text is part of a larger context. The likelihood that the word is associated with a vector of multiple contexts provides a unique fingerprint (feature vector) for the meaning of a word as described above. A variety of statistical measures have been developed for use as embeddings, and the choice depends on task domain and problem to be solved. Classic examples are provided by Term Frequency and Inverse Document Frequency.

Term Frequency (TF) is the number of times a term (or token) occurs in a Document. This is simply the frequency of occurrence, *h(w,d)* as described above. Inverse Document Frequency (IDF) is an information theoretic metric computed as the logarithmically scaled inverse fraction of the documents that contain the word. This provides a measure of how common or rare a word (or token) is across all documents. The product of TF and IDF is sometimes used as a measure of the specificity of a word. In practice many different variations have been developed for these metrics. The method can easily be applied to any symbolic vocabulary including word fragments (word roots, with prefixes and suffixes etc.) and even genetic codes.

The statistics of most individual words are relatively poor in information. Sequences of words provide a much richer source of information. Short sequences of words (or any symbolic tokens) are called N-Grams. An N-gram is a contiguous sequence of N items taken from a longer sequence and used to model statistical patterns in language. N-Grams can provide a semantically rich set of tokens for text classification, but cannot account for minor changes in word order, misspellings of different grammatical forms for the same word.

Word2Vec

In 2013, a team of researchers at Google invented and patented a technique called Word2Vec for converting text into numerical vectors, making it possible to address many natural language processing problems with deep learning (Mikolov et al 2013a), (Mikolov et al 2013b). This led to very rapid progress in machine translation and natural language process. Mobile apps for translating spoken language commonly use such technology. There are stories of doctoral students translating poorly written text in their native language to English and then back to their native language to obtain high quality scientific text.

Word2vec takes as its input a corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located close to one another in the space.

Word2Vec can be implemented with two different encodings to produce a distributed representation of words: continuous Bag-of-Words or continuous Skip-Gram. Continuous Bag-of-Words is faster while Continuous Skip-Gram does a better job for text with infrequent words.

Continuous Bag of Words (CBOW)

A continuous Bag-of-Words (CBoW) learns an embedding that can predict the likelihood of a word based on its immediate context. With CBoW, Word2Vec predicts the current word from a window of surrounding context words. Only the set of surrounding words is used for the encoding. The order of the words does not influence prediction. A key component is the size of the context window, that is, how many words before and after a word are included as context of each word. Most researchers have found that a value of 10 works best for skip-gram and 5 for CBoW.

CBoW is often used with sequences of 5 tokens. For example, the sentence "The quick brown fox jumped over the lazy dog." With 9 words, this sentence generates 5 bags:

{ (The, quick, brown, fox, jumped),
(quick, brown, fox, jumped, over),
(brown, fox, jumped, over, the)
(fox, jumped, over, the, lazy)

(jumped, over, the, lazy, dog)

}.

Each bag is an unordered set that forms an element in the encoding. The order of the word is not preserved, so that "jumped over quick brown fox" and "fox jumped over quick brown" are equivalent.

For each sequence, CBOW can be used to predict the most likely center word (target) given the 4 other context words.

[(quick, brown, –, jumped, over) => fox]

CBoW is used with smaller data sets because it treats the context of the sentence as a single observation. In practice this becomes very inefficient when working with large sets of words.

Skip-N-Grams

N-Grams are sequences of N words or tokens. Skip-N-Grams are a generalization of N-Grams in which the components may have gaps between the tokens. The tokens in a Skip-N-Grams may be separated by as many as *K* words that are not included in the sequence. This provides additional robustness and also augments the number of elements in the encoding as there are many more skip-N-grams available in a text to use as training data.

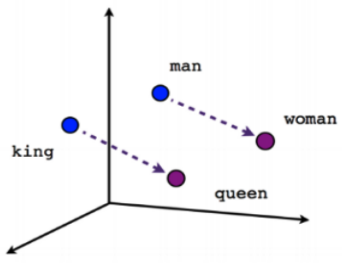
A continuous Skip-N-Gram can predict the surrounding words given a current word. For example, with the sentence "The quick brown fox jumped over the lazy dog" the Skip-N-Gram model will break the sentence into (context, targets) pairs resulting in pairs such as ([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox). This enables a prediction such as

Fox => (quick, brown, jumped, over)

In practice, Skip-N-Grams are commonly implemented with sequences of 10 tokens.

CBoW and Skip-N-Gram are examples of semantic vector space models of language in which words are represented as real-valued vectors. These vectors can then be used for tasks such as information retrieval, document classification, question answering, named entity recognition, and parsing. Such models capture the finer structure of the word vector space using the dimensions of difference rather than the scalar distance between word vectors. Skip-N-Gram models are used to predict a word's context given the word. CBoW can be used to predict a word given its context. Both techniques can be computed with a simple single-layer, fully-connected, neural architecture that computes the inner product between two word vectors.

Words that share semantic or syntactic relationships are represented by vectors of the same number of dimensions that are mapped in close proximity to each other. Such models learn linguistic patterns as linear relationships between the word vectors and can thus be used to compute analogies as a simple arithmetic between vectors. For example: King – Man + Woman = Queen



Queen is to King as Woman is to Man.

Source: http://www.tensorflow.org/tutorials/representation/word2vec

Semantic vector space models, trained on separate local context windows, do not fully use the statistics of a corpus. An alternative is to train the model on a global co-occurrence statistics. This is the approach used in log-bilinear regression models such as GloVe.

GloVe (Global Vectors for Word Representation)

GloVe (Global Vectors for Word Representation) embeddings are a type of word embedding that encodes the co-occurrence probability ratio between two words as vector differences (Pennington et al 2014). GloVe uses a weighted least-squares objective that minimizes the difference between the dot product of the vectors of two words and the logarithm of their number of co-occurrences:

GloVe provides a global log-bilinear regression model that combines the advantages of local context window methods such as CBoW and Skip-N-Gram with global matrix factorization. Such models capture the nonzero elements in a "word to word" co-occurrence matrix rather than individual context windows in a large corpus. The result is a vector space that is useful for named entity recognition that can capture the similarity of meanings of words.

A problem with both vector space models and global vector models is that some relevant words may occur beyond the scope of the immediate context. A surprising solution to this problem has been provided by Attention, as described below. Attention greatly extends the scope of the skip-N-gram to a much larger duration, while suppressing association with irrelevant tokens, encoding only sets of relevant tokens as context.

5. Attention

Attention has long been studied in both human and machine vision as a mechanism to focus processing on the relevant parts of a signal or scene. The human vision system is known to make extensive use of top-down and bottom-up attention processes to suppress irrelevant visual stimuli and limit interpretation to the most salient or the most relevant visual phenomena. Similar processes are known to play a key role in acoustic perception, speech recognition and multimodal interaction. Learning to attend to co-occurring information turns out to be a key technology for learning from unlabeled data.

5.1 The Key Idea

Attention is a filter for associating relevant information. Attention combines three concepts: selection, association, and relevance. Attention operates as a filter, enhancing a small set of sensory input while attenuating or suppressing the rest. The selected set must be small enough to remain within the system's capacity for interpretation. The selected stimuli are interpreted by association with a model. In cognitive psychological terms, this is referred to as association with memory. Interpretation requires that the selected stimuli be relevant, or meaningful, for the model. In psychological terms, relevance is with reference to an action, task or objective.

Attention is fundamental to human perception. For example, the entire human visual system can be seen as succession of filters for attention. Each filter implements a form of attention that allows attended (or selected) information to pass while suppressing irrelevant information to protect the perceptual system from distraction and overload.

In classic AI, and in cognitive psychology, attention has been seen as a balance between a top-down goal directed search for information in a much larger set of stimuli, and bottom-up reaction to salient stimuli. Much of the classic AI literature on attention was focused on methods for enhancing salience to generate triggers for bottom up selection of models. Once selected, the model was assumed to control further processing in a top-down manner, for example looking for stimuli to "fill slots" as with the Frames theory proposed by Marvin Minsky (Minsky 1974).

In cognitive psychology, it is widely accepted that human cognition operates with a set of 7+/-2 working-memory elements. Working memory associates salient perceptual stimuli with concepts

and episodes recalled from long-term memory. This process can be illustrated with two small organs in the brain that are central to visual perception: The Superior Colliculus (SC) that directs visual fixation, and the Lateral Geniculate Nucleus that fuses visual fields from the left and right eyes, and selects relevant visual phenomena for interpretation by the visual cortex.

5.2 Attention in Machine Learning

The idea of using attention to speed up processing for machine learning began to gain credibility within the Machine Learning community in 2010 following a paper discussing the concept of a glimpse (Larochelle and Hinton 2010). The idea was to enlarge the scope of the input without enlarging the quantity of data by suppressing irrelevant text. This idea was rapidly adopted in Natural Language Processing associate relevant words in a sentence through word highlighting. Attention was used to enhance the important parts of the input data while fading out the rest, allowing the network to devote more power on a small but relevant part of the data. The ability to identify and select relevant information depends on context and was learned during training.

Interest in attention for Machine Learning increased following a demonstration that attention at multiple resolutions could significantly speed up processing for tasks such as object tracking (Minh et al 2014). The situation changed dramatically when a team of researchers at Google showed that the convolutional and recurrent layers of deep networks could be completely replaced with a simple fully-connected network using learned self-attention. The result was a revolutionary new technology that has greatly accelerated all areas of artificial intelligence.

5.3 Self-Attention in Transformers

Self-Attention is an operation that suppresses irrelevant data allowing a system to associate relevant information from a signal. Modern generative AI systems make extensive use of learned attention functions to associate information by matching a Query (***Q***) to Keys (***K***). The resulting associations are then used to suppress and enhance information from a Value (**V**) vector in order to generate the latent variables to interpret a signal. This *QKV* association mechanism is highly parallelizable and has been shown to outperform or match deep convolutional and recurrent architectures on a wide range of tasks in both discrimination and generation.

Self-attention with Keys, Queries and Values works in a manner that is similar to associative memory retrieval. The Keys (***K***) act as a form of associative memory address for each token. The Queries (**Q**) define requests for information that can be associated with each token. Values (**V**) define the information that can be recovered by associations. Similarity between **Q** and **K** determines how strongly **V** contributes to the output variable for each token.

In the following, we use traditional column vector notation. Assume a sequence of n tokens, $\mathbf{X} = (\vec{x}_1, \vec{x}_2, ..., \vec{x}_n)$ that compose a context where each token, $\vec{x}_i$, is a d-dimensional vector. At the input level, tokens may represent words, word fragments (subword units) or any other sequential signals. At deeper levels, tokens represent latent variables. For each token, $\vec{x}_i$, the system computes three vectors $\vec{K}_i$, $\vec{Q}_i$, and $\vec{V}_i$ by left-multiplication with learned matrices $\boldsymbol{W}_K$, $\boldsymbol{W}_Q$ and $\boldsymbol{W}_V$.

$$\vec{K}_i = \mathbf{W}_K \vec{x}_i$$

$$\vec{Q}_i = \mathbf{W}_Q \vec{x}_i$$

$$\vec{V}_i = \mathbf{W}_V \vec{x}_i$$

Keys ($\vec{K}_i$) and queries ($\vec{Q}_i$) are vectors with d_K components, while values ($\vec{V}_i$) are vectors of dimension d_V. The matrices $\boldsymbol{W}_K$, and $\boldsymbol{W}_Q$ are of dimension d_k x d and the matrix $\boldsymbol{W}_V$ is of dimension d_V x d. Although d_K and d_V may be different, in many systems they are the same.

The product of a query $\vec{Q}_i$ from the i^{th} token with the key $\vec{K}_j$ of the j^{th} token produces a scalar score, a_{ij}, that indicates how strongly token i attends to the token j. In most implementations, the product of the query and key, $\vec{Q}_i^T \vec{K}_j$ is divided by $\sqrt{d_k}$ to enhance stability during learning.

$$a_{ij} = \frac{Q_i^T K_j}{\sqrt{d_k}}$$

Dividing by $\sqrt{d_k}$ keeps the attention scores in a reasonable numerical range so that learning with backpropagation remains effective. The attention scores are organized into an n x n matrix, $\boldsymbol{A}$, that tells how much each token, $\vec{x}_i$, attends to every other token, $\vec{x}_j$. Each row of $\boldsymbol{A}$ is normalized to sum to *1* with a softmax function. The result is an n x n matrix of attention weights where each row sums to 1, and that describes how much each token, $\vec{x}_i$, attends to every other token, $\vec{x}_j$.

When expressed as matrices, the Keys ($\boldsymbol{K}$), Queries ($\boldsymbol{Q}$) and Values ($\boldsymbol{V}$) for a context $\boldsymbol{X}$ of n tokens are computed as:

$$\boldsymbol{K} = \boldsymbol{W}_K \boldsymbol{X}$$
$$\boldsymbol{Q} = \boldsymbol{W}_Q \boldsymbol{X}$$
$$\boldsymbol{V} = \boldsymbol{W}_V \boldsymbol{X}$$

Where $\boldsymbol{K}$ and $\boldsymbol{Q}$ are $(d_K$ x $n)$ while $\boldsymbol{V}$ is $(d_V$ x $n)$. This is typically written as:

$$\boldsymbol{A} = softmax(\frac{\boldsymbol{Q}^T \boldsymbol{K}}{\sqrt{d_k}})$$

Each output variable, $\vec{Z}_i$, is then computed as a sum over the n value vectors $\vec{V}_j$ weighted by the attention scores a_{ij} to produce n latent output variables of dimension d_V.

$$\vec{Z}_i = \sum_{j=1}^{N} a_{ij} \vec{V}_j$$

In matrix form, this is written:

$$\boldsymbol{Z} = \boldsymbol{V}\, \boldsymbol{A}^T$$

The output $\boldsymbol{Z}$ is a d_V x n matrix. The columns of $\boldsymbol{Z}$ are the n tokens of the output, each expressed in an embedding as a d_v-dimensional vector.

Attention makes it possible to replace a deep recurrent or convolutional network with a simple fully-connected layer that provides a similar error rate with a substantial reduction in the cost of training and run-time operation. This substantial reduction has made possible very deep hierarchical learning in the form of transformers. However training very deep hierarchies raises problems with the stabilities of the gradients used for backpropagation. This is commonly solved using Residual Learning.

6. Residual Learning

Training very deep neural networks with gradient descent poses a fundamental problem: as the number of layers increases, gradients propagated through the network during training with backpropagation may vanish or explode, making learning unstable. A solution to this problem has been provided by an architectural innovation referred to as a Residual Network (ResNet) (He et al 2016).

6.1 The Key Idea

In a traditional neural network layer, each layer can be seen as learning a function $H(\vec{X})$ that maps the input $\vec{X}$ to the output $\vec{Y}$.

$$\vec{Y} = H(\vec{X})$$

The network must learn the entire transformation from input to output. In a residual network, instead of learning a direct mapping $H(\vec{X})$ from the input layer to its output layer, the network learns a correction function, $F(\vec{X})$, referred to as a residual, that applies a minor change to the input vector.

$$F(\vec{X}) = H(\vec{X}) - \vec{X}$$

The layer then computes the output as an element-wise addition of the Residual to the input.

$$\vec{Y} = \vec{X} + F(\vec{X})$$

This formulation allows each layer to learn only the correction required to refine the representation rather than a complete transformation.

A key property of residual networks is that every layer has the same number of variables. The resulting output layer is the same size as the input. This property is crucial for training very deep networks because it allows gradients to propagate backward without being repeatedly multiplied by shrinking or exploding weight matrices.

6.2 Gradient Stability

When gradients propagate backward through a deep network, each layer contributes a Jacobian matrix. Without residual connections, the gradient becomes a long product of Jacobian matrices, which can easily cause vanishing or exploding gradients. With Residual networks, the output of each layer contains a copy of the input. As a result, during backpropagation the gradient includes an identity component,

$$\frac{\partial \vec{Y}}{\partial \vec{X}} = I + \frac{\partial F(\vec{X})}{\partial \vec{X}}$$

which helps stabilize gradient propagation across many layers. This makes it possible to train very deep architectures with tens or even hundreds of layers. The presence of the identity matrix ensures that gradients can pass through many layers without collapsing or exploding.

Maintaining the same latent dimensionality allows this identity mapping to exist naturally. This design makes it feasible to train transformers with dozens or hundreds of layers, something that would otherwise be extremely difficult. Another reason the dimensionality remains constant is that the network is not trying to compress the representation layer by layer, as an autoencoder might. Instead, it is trying to refine a predictive representation of the sequence.

From the perspective of generative learning, residual networks can be interpreted as performing a sequence of incremental refinements of a latent predictive representation. This iterative refinement perspective aligns naturally with the information-theoretic interpretation of self-supervised learning discussed earlier: each layer modifies the latent state to reduce the prediction error of the model.

In early residual networks, residual variables were calculated with classic multi-layer perceptrons or convolutional networks. In a transformer architectures (Vaswani et al 2017), residual variables are calculated with self-attention as described below. In these models each layer refines the latent representation by adding attention and feed-forward corrections to the current state, enabling stable optimization of extremely deep networks.

7. Transformers

A Transformer is a neural network architecture that transforms an input sequence into an output sequence. The Transformer was originally developed as a machine translation technology that could translate complete sentences between natural languages (Vaswani et al 2017). Natural languages have complex grammatical structures and often express concepts that do not directly match to words in the target language. A sentence level machine translation system must be able to identify the different grammatical elements in the input sentence, often from words or word segments that are spread throughout the input sentence and then translate these elements to a grammatically correct sentence in the target language expressing the same or similar meaning.

The Transformer addressed natural language translation with radical new approach for machine learning that replaced deep convolutional and recurrent networks with a bank of trained attention heads. Recognition of grammatical structures with self-attention provided a substantial reduction in both the computational cost of the network, and in the computational cost and quantity of data required for training, compared to earlier approaches with deep recurrent and convolutional networks. Initially this was used to reduce the cost of training and run-time execution, but it was soon understood that the reduction in computational cost enabled construction of hierarchical networks with a much larger scope of context. Extending the context window enables transformers to represent significantly more information leading to substantial gains in the depth and complexity of associations. The computational gains provided by self-attention made possible hierarchies of encoders and decoders that could not only translate words and sentences, but also extract meaning and translate paragraphs, chapters or even entire documents.

The introductory paper "Attention Is All You Need" (Vaswani et al 2017) documented a revolutionary new technology for natural language processing. Significant advances soon appeared with the emergence of Bidirectional Encoder Representations from Transformers (BERT), Generative Pre-trained Transformers (GPT) and other similar architectures. Transformers were developed for creating images from textual descriptions (DALL-E), for analyzing images (ViT), and speech and music, and for associating natural language with multiple interaction modalities including visual, auditory or tactile perception. New architectural variations continue to emerge with significant gains in training efficiency and task effectiveness. Multiple different transformer architectures have been developed for natural language tasks as well as computer vision, spoken language and multi-modal interaction.

7.1 The Key Idea

Most transformers combine the following key elements:

1) Clever use of input and output embeddings for representing input data.

2) The use of stacked layers of encoders and decoders to provide a hierarchy of contextual descriptions for the input and output data.

3) The use of multiple attention heads that use trained self-attention to identify and associate grammatical elements in the input and output signals.

4) Residual structures in which the input signal is added element-wise to latent variables to allow associations at deeper levels.

5) An auto-regressive structure in which the output decoders are driven by a combination of the latent variables from the input decoders and the previous output.

6) The use of self-supervised learning in which the transformer trains itself by learning to reconstruct artificially corrupted (partially masked) input data, and to predict previous and next input data from massive amounts of unlabeled training data.

7.2 Architectural Details

The Transformer is an auto-regressive architecture, consuming the previously generated symbols as additional input when generating the next output. Self-attention allows each word in a sentence or paragraph to look at other tokens to better know which token relates to the current token. Transformers consist of multiple layers where the output encoding of each layer acts as input to the next layer. Each encoder is composed of a parallel set of trained attention heads for self-attention, followed by a fully-connected layer that maps the selected tokens to a set of latent variables for use as input to the next layer. The attention heads in each layer select relevant parts of an input coding for association and inference by the next encoder.

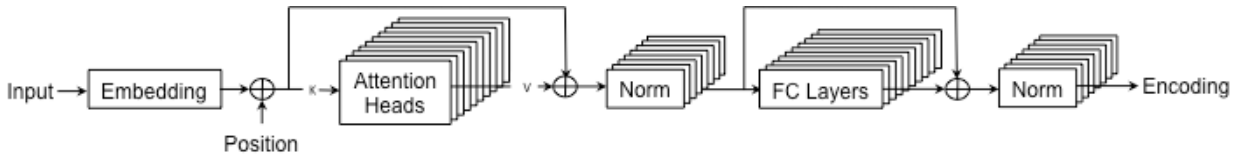


Figure 2. Encoder Architecture for a vanilla transformer.

The Transformer encoder operates on a set of tokens created by encoding an input signal with an embedding. Positional encodings are added to the tokens to capture the relative position of each element in the input. Multiple attention heads associate relevant tokens using the Key-Query-Value computations as described above. Each attention head uses a set of three learned matrices that determine which tokens are related, outputting a value that represents the associated tokens. The original embedding is appended to the output of each attention head in an inception style computation and the result is normalized to sum to one. The associations from each attention head are then processed by a fully-connected neural layer and added to its input to produce the encoding from each attention head.

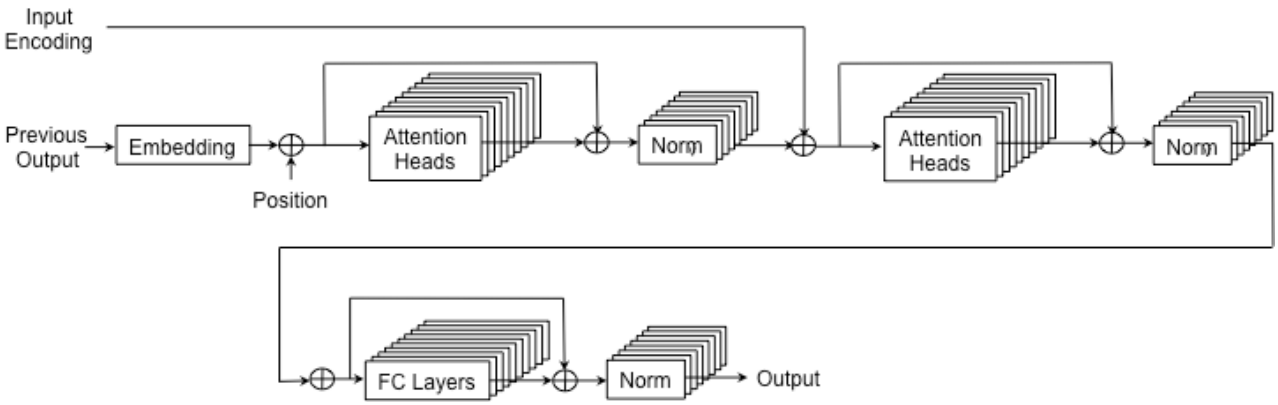


Figure 3. The decoder architecture for a vanilla transformer.

The decoder in a Transformer combines the input encoding produced by the encoder to an encoding of a shifted version of the previous output to produce a new output. When processing temporal sequences such as spoken language or text, the previous output is a shifted copy so that the new output is appended in the next position. When processing non-temporal signals such as images, the input is typically a 2D window, and the previous output may be a partial description such as a low-resolution image to which the decoder is adding detail. Some systems for processing text treat the text as a temporal sequence and generate the next output token to the right from the previous tokens on the left. Some more recent text based systems consider both the right and left context in producing a successively refined version of the output.

The above description provides a very approximate description of a generic transformer. Different transformer architectures differ from this generic model in various details. The real power of

transformers results from composing multiple layers of encoders and decoders providing increasingly deep context for both the input and the output. The classic example is the architecture that was used to produce the first example of a useful transformer, BERT.

7.3 BERT - Bidirectional Encoder Representations from Transformers

Shortly after the Vaswani paper on attention was published, researchers at Google AI language developed a Transformer-based architecture for natural language processing named BERT: Bidirectional Encoder Representations from Transformers (Devlin 2018). The BERT transformer architecture was a significant advancement in natural language processing, demonstrating the effectiveness of bidirectional context in understanding language, and has since influenced many subsequent developments.

Prior to BERT, the Google AI Language team had developed a model named ELMo (Devlin 2018) using a bi-directional Long-Short Term Memory (LSTM) (Gers 2000) that took into account the context from both the left and the right context when interpreting a token. ELMo produced contextualized word embeddings, improving on research using embedding techniques such as Bag-of-Words and Word2Vec. With BERT, they adapted this bidirectional approach to a transformer-style architecture, using self-attention to replace the deep recurrent networks.

An important innovation introduced by BERT was the use of self-supervised learning, in which a system trains itself by corrupting and reconstructing data. With self-supervised learning, a system learns to reconstruct missing data and to guess associated data from examples. With self-supervised learning, the data is its own ground truth. The key idea for self-supervised learning is learning to reconstruct corrupted versions of raw data from local neighborhoods at multiple levels of abstraction. A system trains itself by masking small parts of the training data and learning to reconstruct the missing parts from the local neighbors. Ironically, this is reminiscent of the way young children learn from memorizing data by filling in the blanks.

BERT was trained using two training techniques: masked-token completion and next sentence prediction. With Masked-Token Completion a percentage of randomly selected input tokens are masked during training and the system is trained to predict the missing tokens using the surrounding context. With Next Sentence Prediction the system is trained to predict an association between two sentences, enabling the system to predict the next sentence (or sequence of tokens) based on the current sentence.

BERT was pre-trained by self-supervised learning using 3.3 billion tokens of unlabeled text extracted from the English Language Wikipedia (2.5 billion words) and the BooksCorpus with (800 million words). In the Masked-Token Completion task, BERT learned to predict the identities of words that have been masked-out of the input text. In the Next Sentence Prediction task, BERT was trained to predict whether the second half of the input follows the first half of the input in the corpus, or is a random paragraph. Further training the model on supervised data results in impressive performance across a variety of tasks ranging from sentiment analysis to question answering.

BERT is useful for tasks requiring understanding and analysis of text rather than generation. It excels in tasks such as text classification, sentiment analysis, Named Entity Recognition, Question Answering, and Sentence Similarity tasks. BERT uses context and is thus effective in situations where nuanced comprehension of language is needed. Because of the use of bidirectional context, the system performs well on tasks that require understanding relationships and contexts. For example, BERT is very well suited for Google-style web search, in which the system is provided with a poorly structured inquiry and asked to provide a sorted list of relevant web pages.

In October 2019, Google began using BERT to process search queries. In 2020, Google Translate replaced the previous Recurrent Neural Network based encoder–decoder model by a Transformer based encoder combined with a Recurrent Neural Network based decoder. However, the system is not designed for generating text, making it less suitable for tasks like story or dialogue generation.

Self-supervised learning makes it possible to pre-train a general-purpose system with a very large collection of unlabeled training data, followed by transfer learning using supervised learning or reinforcement learning to specialize for a particular problem. The BERT architecture is easily extended to multimodal perception and interaction by simply concatenating encodings of different modalities. Each layer uses multiple Self-Attention Heads to associate multiple mutually relevant entities to be interpreted at that next level. Thus BERT can be trained to complete missing data with multiple modalities, or to predict an appropriate reaction in all modalities for a stimulus in one or more modalities.

7.4 GPT and Large Language Models

In 2018 OpenAI proposed a novel Transformer architecture named GPT (Generative Pre-trained Transformer) that used residual learning and minimized the use of encoders in order to focus training on the generative components (Radford et al 2018). This allowed GPT to greatly expand the number of parameters used for learning to generate output. Training on an extremely large training corpus gave GPT an exceptional power at modeling and generating natural language. Since 2018 the GPT series of decoder-only Transformers have defined the state of the art in natural language generation.

GPT is an example of an autoregressive language model. In autoregressive language modeling, a network learns to predict the next token x_T from a sequence of tokens $x_1, x_2, ..., x_{T-1}$. from a large corpus of training data. The sequence of tokens $x_1, x_2, ..., x_{T-1}$ is referred to as a context. This is equivalent to learning a factorization of the joint probability of the sequence

$$P(x_1, ..., x_T) = \prod_{t=1}^{T} P_\theta(x_T \mid x_1, ..., x_{T-1})$$

where the ground truth for each prediction is simply the next token that appears in the training corpus. Learning proceeds by maximizing the likelihood of the observed sequences, or equivalently by minimizing the cross-entropy loss

$$L = -\sum_{t=1}^{T} \log P_\theta(x_T \mid x_{T-C}, ..., x_{T-1})$$

This forces the model to approximate the empirical distribution of language (Bengio et al 2003).

In information-theoretic terms, minimizing cross-entropy is equivalent to minimizing the expected description length of the data under the model, connecting modern self-supervised learning with the Minimum Description Length (MDL) principle introduced by Rissanen (Rissanen 1978). From this perspective, learning can be interpreted as constructing latent representations that provide increasingly compact predictive descriptions of the observed data.

The use of residual learning made it possible for OpenAI implement GPT using very deep networks using very large numbers of parameters. GPT-1 was initially created in 2018 with 117 million parameters spread over twelve layers with a context window of 512 tokens, dwarfing competing models in both size and computational cost. In February 2019 GPT-2 extended this size to 1.5 billion parameters extending processing to 48 layers and a context window of 1024 tokens, corresponding to about a page of text. In June 2020, OpenAI released GPT-3 with 175 billion parameters, 96 layers and a context window of 2048 tokens. This was more than 10 times the capacity of the next largest Transformer model for Natural Language Processing available at the time. And yet, GPT-3 was not suitable for human interaction.

7.5 ChatGPT

Large Language Models (LLMs) based on Transformers would appear to be ideal for creating chatbots that use natural language to provide information and services to users. However, early experiments with using LLMs to create chatbots by Meta and IBM led to a number of highly embarrassing failures, as users learned to prompt such systems to generate fake news, racist insults and toxic advice. GPT showed similar weaknesses. Despite its enormous size, GPT-3 would make simple mistakes, fail to follow instructions, make up facts, give long indirect hedging answers and fail to detect questions based on false premises.

OpenAI presented this problem as a lack of alignment between transformer output and human expectations and proposed that the problem could be solved by an additional fine-tuning stage to teach the system to interact with humans. OpenAI formed a human-machine interaction team, known as the alignment team, to explore ways to align model outputs with human intentions. The team constructed a lightweight version of GPT, called InstructGPT, with only 1.3 Billion parameters. InstructGPT was subjected to a post-training phase that used supervised learning and reinforcement learning to avoid generating outputs that displeased human trainers. In early 2022, the alignment team published results of early experiments with InstructGPT, demonstrating that humans clearly preferred interaction with the lightweight version of InstructGPT rather than the much more capable full version of GPT-3 (Ouyang et al 2022). The experiments showed that fine-tuning large language models using human feedback could significantly improve a model's ability to provide outputs that were helpful, safe, and aligned with human expectations.

The team developed a three-stage training pipeline that combined supervised fine-tuning and Reinforcement Learning from Human Feedback (RLHF). These methods were specifically designed to improve the truthfulness of GPT's output. This fine-tuning pipeline was composed using 3 stages

1. Building a data set of data for fine-tuning using human-written demonstrations.
2. Training a Reward Model using examples of outputs by human labelers.
3. Using the Reward Model for Reinforcement Learning from Human Feedback to adapt outputs to human expectations.

These techniques were then used to fine-tune GPT-3 for human interaction in order to create a chatbot named "Chat with GPT". In late 2022, the team proposed an experiment in which an alpha version of Chat with GPT would be made publicly available for a short period to gauge public reaction. Just before release, the name was changed to ChatGPT.

Previous failures of such experiments by Meta and IBM led the team to be very low-key, announcing the experiment with a simple blog post and a tweet by OpenAI CEO Sam Altman. The response was spectacular. Within 24 hours, the system had been used by over 100,000 users. Within 5 days the number of users surpassed 1 million. The system began to demonstrate unexpected abilities such as an ability to interact in Japanese and an ability to write original computer programs. People began using ChatGPT for medical advice, travel planning and writing letters.

GPT was the first and most prominent example of a Large Language Model using decoder-only Transformers. Since the appearance in 2022 of ChatGPT, a number of competing large language models such as LLaMA, Gemini, Claude, Mistral, and Grok have been announced. OpenAI has released a series of GPT foundational models, labeled GPT-*n*, based on increasingly larger numbers of parameters and the use of substantially larger training corpus.

The release of ChatGPT in November 2022 triggered a revolutionary advance in generative AI and a new wave of interest in the power of Generative AI and Large Language models. However, applications for Transformers extend well beyond Natural Language Processing. Transformers are increasingly used for computer vision, audio perception, and multimodal interaction. Foundation models based on transformers are now finding applications in autonomous vehicles, humanoid robotics, manufacturing and material science.

7.6 Vision Transformer (ViT)

In 2020, a team at Google Brain adapted the transformer model to processing images for computer vision (Dosovitskiy et al 2020). The Vision Transformer (ViT) uses an architecture that follows the original Transformer architecture as closely as possible. Small Image patches (16x16 or 8x8 pixels) were flattened to 1D vectors and used in place of the 1D token embeddings used in the original transformer. Because images have a 2D structure, a position embedding was added to the patches using a sine-based code. The resulting vectors were then used as token embeddings. This allowed the team to use much of the original transformer code "out of the box". As with the original transformer, multiple attention heads use self-attention to capture relations between different image patches.

ViT was pretrained on extremely large dataset using self-supervised learning. The resulting model was shown to achieve performance similar to deep CNNs when trained on sufficiently large datasets. ViT has been published as a trained foundational model that can be adapted to a large variety of image analysis and computer vision tasks with subsequent fine-tuning. Applications include image classification, object detection and segmentation, category discovery in images and videos.

7.7 Auditory Transformers

A number of Transformer architectures have been developed for audio processing for tasks such as speech recognition, music generation, and sound classification. The Audio Transformer developed by Google Research adapts the original transformer architecture for audio tasks by using spectrograms (time-frequency representations) as an embedding. The Audio Transformer can be used for various tasks, including music genre classification and sound event detection. The Speech Transformer, developed by Google Research is designed for speech recognition tasks, using a transformer architecture that can effectively model long-range dependencies in audio signals. The Speech Transformer performs well in end-to-end speech recognition setups. A Music Transformer, developed by Google Brain focuses on music generation and is capable of capturing musical structures like melody and harmony. It uses a modified transformer architecture that incorporates attention mechanisms specifically tailored for musical contexts.

7.8 Multimodal Transformers

CLIP (Contrastive Language–Image Pre-training), developed by OpenAI is a multimodal transformer that learns to connect images and text through a contrastive learning approach. CLIP is designed to understand and relate images and text in a unified way. It can understand and relate text descriptions to images, making it highly effective for tasks like zero-shot image classification, image generation from text prompts.

CLIP's training methodology is based on contrastive learning, using a contrastive loss function, referred to as InfoNCE loss. With this approach, the model learns to maximize the similarity between the representations of paired images and texts while minimizing the similarity between non-matching pairs.

CLIP employs a dual-encoder architecture that consists of a Vision Encoder, based on ViT, that processes images a combined with a transformer based Text Encoder that processes textual descriptions. Both encoders project their outputs into a shared embedding space, enabling direct comparisons between image and text representations.

A very large dataset of image-text pairs is used to learn representations that can be applied to various tasks involving both modalities. The training process involves learning to associate images with their textual descriptions, allowing the model to understand visual concepts in the context of language. Once trained, CLIP can generalize to various tasks without additional fine-tuning. For example, it can classify images based on textual prompts (e.g., "a cat," "a dog") even if it hasn't

seen specific classes during training. This flexibility is due to the model's ability to interpret text descriptions in the context of the visual content.

CLIP can be used for applications such as Image Classification, Image Search and Retrieval, Image Generation and Cross-Modal Tasks, and can classify images into categories based on text prompts without needing labeled training data for those specific categories. Users can search for images using natural language queries, and obtain relevant images based on the learned associations. When combined with other generative models, CLIP can guide the generation of images that match specific text descriptions.

8. Conclusions

Since the initial publication of the attention paper in 2017, Generative Systems have become the dominant approach for natural language processing (NLP) with systems such as BERT (Devlin et al., 2019) and GPT-*n* (Brown et al., 2020) rapidly displacing more established RNN and CNN structures with an architecture composed of stacked encoder-decoder modules using self-attention. Generative AI has had a revolutionary impact on Computer Vision, with a flood of transformer inspired computer vision techniques that have come to dominate the major conferences and journals. Similar impact has been observed in Speech Recognition and recent results have shown that transformers are well suited for multimodal perception combining language and computer vision. The result is a widespread availability of source code and data for research on multimodal perception with transformers. The websites arXiv and "Papers with Code" and Code examples in Keras present the most salient examples of this movement and an excellent educational and tutorial resource.

Bibliography

(Crowley 2021) J. L. Crowley, Machine Learning with Neural Networks. In *ECCAI Advanced Course on Artificial Intelligence*, pp. 39-54, Springer International Publishing, 2021.

(Ackley et al 1985) D. H. Ackley, G. E. Hinton and T. J. Sejnowski, "A Learning Algorithm for Boltzmann Machines", *Cognitive Science*, vol 9, pp 147-169, 1985.

(Hinton and Zemel 1993) G. E. Hinton and R. Zemel, Autoencoders, minimum description length and Helmholtz free energy, *Advances in neural information processing systems* 6, 1993.

(Vazquez-Rodriguez et al 2022) J. Vazquez-Rodriguez, G. Lefebvre, J. Cumin, and J. L. Crowley, Emotion recognition with pre-trained transformers using multimodal signals. In *2022 10th International Conference on Affective Computing and Intelligent Interaction (ACII)*. IEEE. Oct. 2022.

(Kingma and Welling 2014) D. P. Kingma and M. Welling, Auto-encoding Variational Bayes. ICLR 2014, *2014 International Conference on Learning Representations*, Banff, AB, Canada, April 14-16, 2014.

(Duda and Hart 1973) R. O. Duda, and P. E. Hart, *Pattern Recognition and Scene Analysis*. Wiley New York, 1973.

(Turk and Pentland 1991) M. A. Turk and A. Pentland. Face Recognition Using Eigenfaces. In Proceedings of the 1991 *IEEE conference on Computer Vision and Pattern Recognition*, pages 586–587. IEEE Computer Society, 1991.

(Schwerdt et al 2000) K. Schwerdt, D. Hall, J. L. Crowley, "Visual Recognition of Emotional States", The Third International Conference on Multimodal Interfaces, ICMI-2000, p41-48, Beijing China, October 2000.

(Crowley et al 1998) J. L. Crowley, F. Wallner and B. Schiele, "Position Estimation Using Principal Components of Range Data", Robotics and Autonomous Systems, Vol 23, no 4, pp 267-276, 1998.

(Shannon 1948) C. Shannon, The Mathematical Theory of Communication, *Bell System Technical Journal*, Vol. 27, No. 4, pp. 379-423 and pp. 623 - 656, July and Oct, 1948.

(Harris 1954) Z. S. Harris, Distributional Structure. *Word*, 10(2–3), 146–162, 1954.

(Mikolov et al 2013a) T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient Estimation of Word Representations in Vector Space, arXiv preprint arXiv:1301.3781, 16 January 2013.

(Mikolov et al 2013b) T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 2013.

(Pennington et al 2014) J. Pennington, R. Socher, and C. Manning, GloVe: Global Vectors for Word Representation, In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (*EMNLP*), pp 1532–1543, Doha, Qatar, 2014.

(He et al 2016) Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

(Minsky 1974) M. Minsky, A Framework for Representing Knowledge, MIT AI Lab memo 306, June 1974.

(Larochelle and Hinton 2010) H. Larochelle and G. E. Hinton, "Learning to combine foveal glimpses with a third-order Boltzmann machine," in *Proc. NIPS*, pp. 1243–1251, 2010.

(Minh et al 2014) V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, “Recurrent models of Visual Attention,” in *Neural Information Processing*, NIPS-2014, pp. 2204–2212, Curran Associates Red Hook, NY, USA, 2014,

(Vaswani et al 2017) A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, Attention is all you need, *31st Conference on Neural Information Processing Systems (NIPS 2017),* Long Beach, CA., USA, 2017.

(Radford et al 2018) A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, Improving Language Understanding by Generative Pre-Training, 2018.

(Devlin 2018) Devlin, J., Chang, M. W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

(Bengio et al 2003) Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

(Rissanen 1978) Jorma Rissanen, Modeling by shortest data description, *Automatica*, 14(5):465–471, 1978.

(Gers 2000) F. A. Gers, J. Schmidhuber, and F. Cummins, Learning to forget: Continual prediction with LSTM. *Neural computation*, vol. 12, no 10, p. 2451-2471, 2000

(Radford 2021) A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763, July, 2021.

(Brown 2020) Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., and Amodei, D. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165.

(Ouyang et al 2022) Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S. Slama, K., Ray, A. Schulman, J. Hilton, J. Kelton, F., Miller, L., Simens, M. Askell A., Welinder, P. Christiano, P. Leike J., and Lowe, R. (2022). Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35, 27730-27744.

(Dosovitskiy 2021) A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai,T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, and J. Uszkoreit, (2021) An image is worth 16x16 words: Transformers for image recognition at scale. *9th International Conference on Learning Representations, ICLR*. 2021.

Index

A

Attention, 3, 12, 13, 15
Attention head, 16, 17
Audio Transformer, 21
Autoencoder, 3, 4, 5

B

Backpropagation, 3
Bag-of-Words, 9, 10, 18
BERT, 16, 18
BoW, 9, 10

C

CBoW, 10, 12
CLIP, 22
Continuous Bag-of-Words, 10
Continuous Skip-Gram, 10

D

DALL-E, 16
Decoder, 3, 16, 17, 18
Denoising Autoencoder, 5

E

embedding, 8
Embedding, 3, 8
Encoder, 3, 16, 17

F

Frames, 13

G

Generative AI, 3
Generative Pre-trained Transformers, 16, 19
GloVe, 12
GPT, 16, 19

H

Histograms, 9

I

Inverse Document Frequency, 10

K

Key, 13
KL divergence, 6
Kullback-Leibler divergence, 6

L

Latent code vector, 4
Latent variables, 3, 5, 6, 7, 17
Latent-variable model, 4
Lateral Geniculate Nucleus, 13
Least-squares loss, 5
Loss function, 5
LSTM, 18

M

Machine Learning, 13
Masked-Token Completion, 19
MNIST data set, 7

N

Natural Language Processing, 13
Next Sentence Prediction, 19
N-Gram, 10
N-Grams, 11

P

PCA, 8
Principal Components Analysis, 5, 8

Q

Query, 13

R

Reinforcement Learning from Human Feedback, 20
Residual Network, 15
ResNet, 15

S

Self-attention, 3, 13, 16, 17
Self-supervised learning, 3, 4, 17, 18, 19
Skip-N-Gram, 11, 12
Sparse Autoencoder, 6
Sparsity, 6
Superior Colliculus, 13

T

Term Frequency, 10
Transformer, 3, 7, 15, 16, 17, 18

V

Value, 13
Variational Autoencoders, 7
Vision Transformer, 21
ViT, 16, 21

W

Word2Vec, 10, 18

Mathematical Notation

Note: All vectors and matrices are written with classic column vector notation. Vectors are indicated with an arrow symbol.

X_i	An observed or measured value.
$\vec{X}$	A vector of d measurements.
$\vec{Z}$	A vector of n latent variables computed by an Encoder $\vec{Z} = E(\vec{X})$
$\hat{X} = D(\vec{Z})$	An estimated vector computed from Latent Variables by a Decoder $D(\vec{Z})$
$L(\vec{X})$	A loss function or cost function used to train a network with back-propagation.
$W^{(l)}$	A matrix of weights for the units of layer l.
$B^{(l)}$	A vector of biases for the units of layer l.
ρ	The sparsity parameter (desired average independence)
$\hat{\rho}(n)$	The estimated sparsity (independence) of the n^{th} latent variable
$P(\vec{Z},\vec{X})$	A joint probability distribution for a vector $\vec{X}$ and a vector of latent variables $\vec{Z}$
w	A word or token in a sequence
$h(w)$	A histogram (table of frequency of occurrence) for a document of M words
$P(w) = \frac{1}{M} h(w)$	The probability of occurrence of a word w in a document of M words
$P(w \mid d)$	The probability of the word w given the document d.
$P(d \mid w)$	The probability of the document d given the word w.
$\vec{Y} = H(\vec{X})$	A Function that transformed a vector $\vec{X}$ to a vector $\vec{Y}$
$F(\vec{X}) = \vec{Y} - \vec{X}$	The residual vector for the transformation from a vector $\vec{X}$ to a vector $\vec{Y}$
$\vec{K}_i = \mathbf{W}_K \vec{x}_i$	A Key vector that serves as a hash code to recall a token $\vec{x}_i$
$\vec{Q}_i = \mathbf{W}_Q \vec{x}_i$	A Query vector for tokens to associate with a token $\vec{x}_i$
$a_{ij} = \frac{\vec{Q}_i^T \vec{K}_j}{\sqrt{d_k}}$	a score that indicates how strongly token $\vec{x}_i$ attends towards the token $\vec{x}_j$.
$\boldsymbol{A} = softmax(\frac{\boldsymbol{Q}^T \boldsymbol{K}}{\sqrt{d_k}})$	An attention matrix that tells how much each token $\vec{x}_i$ attends to token, $\vec{x}_j$.
$\vec{V}_i = \mathbf{W}_V \vec{x}_i$	A Value vector contributed by an association with a token $\vec{x}_i$
$\vec{Z}_i = \sum_{j=1}^{N} a_{ij} \vec{V}_j$	A latent output variable computed as a weighted sum of n value vectors $\vec{V}_j$