Understanding Large Language Models: A Tutorial - Part 1

Kenneth Zhang

August 2023

Word Embeddings: A basis for LLM necessity

Word embeddings represent a foundational concept in Natural Language Processing (NLP), converting words into vectors in a continuous space. This allows similar words or words used in similar contexts to be mapped close to each other. Let's explore the mathematical and machine learning concepts behind it.

1. Traditional Representation: One-Hot Encoding

Traditionally, a vocabulary V of size $|V|$ would use a one-hot encoding. Each word w is represented as a vector of size $|V|$ where the index corresponding to w is set to 1, and all others are set to 0.

For instance, for a vocabulary $V = \{\text{apple}, \text{orange}, \text{banana}\}$, the word "apple" could be represented as:

$$\text{apple} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

However, this representation is sparse and doesn't capture any semantic information about words.

2. Motivation for Dense Representations

We need a representation that:

- Reduces dimensionality
- Captures semantic meaning

Dense vectors (where most elements are non-zero) of a fixed size, regardless of vocabulary size, are preferred. These dense vectors are what we refer to as "word embeddings."

3. Creating Word Embeddings: Context Matters

The fundamental hypothesis driving word embeddings is the Distributional Hypothesis:

A word is characterized by the company it keeps.

In essence, words appearing in similar contexts tend to have similar meanings.

3.1. Co-occurrence Matrix

Given a corpus, for each word w and context c, count how often c appears in the vicinity of w. This results in a $|V| \times |V|$ matrix where each element X_{ij} represents how often word i appears near word j.

3.2. Dimensionality Reduction

The co-occurrence matrix can be extremely large and sparse. Dimensionality reduction techniques, like Singular Value Decomposition (SVD), can reduce this matrix to a more manageable size while retaining the most important semantic information.

$$X \approx U\Sigma V^T$$

Where:

- U and V are orthogonal matrices.
- Σ is a diagonal matrix with singular values in descending order.

The rows of U (or V) give dense word vectors. Typically, we might reduce our matrix to have a few hundred dimensions.

4. Prediction-Based Methods

Prediction models, notably Word2Vec, train embeddings by predicting a word from its context (or vice versa). This contrasts with the count-based methods of co-occurrence matrices.

4.1. Skip-Gram and Continuous Bag Of Words (CBOW)

Word2Vec offers two architectures:

- Skip-Gram: Predict context words from a target word.
- CBOW: Predict a target word from its context.

By training on these tasks, the model learns to represent words in a way that can predict their context, thus learning the semantic meaning of words.

5. Advantages of Word Embeddings

1. **Semantic Relationships**: Word embeddings can capture complex semantic relationships. For instance, the vector operations: vector("king")−vector("man")+vector("woman") results in a vector close to vector("queen").
2. **Dimensionality Reduction**: Reduces the dimensionality from $|V|$ (in tens of thousands or more) to a few hundred.
3. **Transfer Learning**: Trained embeddings on one task can be used on another. For instance, embeddings trained on a large corpus can be used on a different, smaller dataset.

1 Attention Mechanism

1.1 Background: Neural Machine Translation

Translation, from a neural machine perspective, is equivalent to finding a target sentence y that maximizes the conditional probability of y given a source sentence x. Simply put, we maximize the conditional probability by fitting a parameterized model on the sentence pairs using a training corpus. Using previous approaches by Kalchbrenner and Blunsom, 2013, and Cho et al., 2014, and Forcada and Neco, 1997, the RNN encoder-decoder neural machine translation approach consistents of two components: an encoder for x and a decoder to decode to target sentence y.

1.2 RNN Encoder-Decoder

The focus of the RNN Encoder-Decoder model is to align and translate simultaneously. This process consists of the following:

1. **Encoder**: reads in input sentence (represented by a sequence of vectors) $x = (x_1, ..., x_{T_x})$, into a context vector c, such that:

   $h_t = f(x_t, h_{t-1})$ represents the hidden state at time t and c is a vector generated from the sequence of hidden states.
   f and g are non-linear functions, which could be, for example, an LSTM for f and $q(h_T, ..., h_T)$.

2. **Decoder**: network that is trained to predict the next word y'_t given the context vector c and all previously predicted target words $y_1, ..., y_{t'-1}$

   The decoder defines a probability over y by decomposing the joint probability into ordered conditionals, which can be expressed as such:

   $p(y) = \prod_{t=1}^{T} p(y_t|\{y_1, ..., y_{t-1}\}, c)$ where $y = (y_1, ..., y_{T_y})$

   For RNNs, each conditional probability is modelled as:
   $p(y_t|y_1, ..., y_{t-1}, c) = g(y_{t-1}, s_t, c)$ where g is a nonlinear function, and is a function that outputs the probability of y_T and s_t is the hidden state of the RNN.

1.3 Learning: Alignment and Translation

A novel architecture proposed by Bahdanau et al, 2015, proposed a new network that consists of a Bidirectional RNN as an encoder and a decoder that emulates searching through a source sentence while decoding a translation.

1.3.1 Decoder: General Architecture

Define each conditional probability as $p(y_i|y_1, ..., y_{i-1}) = g(y_{i-1}, s_i, c_i)$. Where, s_i is an RNN hidden state for time i computed by $s_i = f(s_{i-1}, y_{i-1}, c_i)$.

It is worth noting: unlike traditional encoder-decoder approaches, probabilities for the novel architecture are conditioned on a distinct context vector c_i for each target word y_i.

The context vector, c_i, depends on a sequence of annotations $(h_1, ..., h_{T_x})$ which an encoder maps the input sequence. Each annotation, h_i, contains information about the entire input sequence with a focus on the parts surrounding the i-th word of the input sequence. (attention mechanism)

The context vector, c_i, is then computed as a weighted sum of these annotations from the input sequence:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

where the weight α_{ij} of each annotation h_i is computed by

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})}$$

where the alignment model $e_{ij} = a(s_{i-1}, h_j)$.

The alignment model scores how well the inputs around position j and the output at position i match, and is based on the hidden state s_{i-1} and j-th annotation. α, the alignment model, is a feedforward neural network, jointly trained with all the other components of the proposed system.

Unlike traditional machine translation, the alignment is not considered to be a latent variable and directly computes the soft alignment, which allows the gradient of the cost function to be backpropagated so that it can be used to train the alignment model as well as the whole translation model, jointly.

1.4 Attention Mechanism

Taking a weighted sum of all the annotations as computing an expected annotation is equivalent to the expectation over all possible alignments. Let α_{ij} be the probability that the target word y_i is aligned to or translated from, a source word x_j. The i-th context vector c_i is the expected annotation over all the annotations with probabilities α_{ij}. Probability α_{ij}, or its associated energy e_{ij}, reflects the importance of the annotation h_j with respect to the previous hidden state s_{i-1}, in deciding the next state s_i and generating the target word, y_i. The decoder decides on parts of the source sentence to pay attention to. The decoder attention mechanism relieves the encoder of the burden of having to encode all information in the source sentence into a fixed-length vector.

1.5 Types of Attention

In summary, the attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as the weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

1.5.1 Scaled Dot-Product Attention

Scaled Dot-Product Attention (Vaswani et al., 2023), has inputs that consist of queries and keys of dimension d_k and values of dimension d_v. The dot product of the queries and keys are computed, then divided by $\sqrt{d_k}$ and applied softmax to obtain weights of the values. The attention function is computed simultaneously on a set of queries and packed in a Q matrix. The keys and values are packed into matrices K and V resulting in the matrix of outputs:

$$\text{Attention(Q, K, V)} = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

1.5.2 Scaled Dot-Product Attention Process

The Scaled Dot-Product Attention Mechanism requires multiple processes:

1. Input: Queries (Q), Keys (K), and Values (V) as matrices
2. Dot Product of Q and K: Give a measure of similarity (higher dot product implies more similarity between the given query and specific key). The Resultant is a matrix where each row represents a query and each column represents a key.
3. Scaling: Scaled by the root of the depth of the key vectors (helps the dot products avoid growing to larger magnitudes, leading to softer softmax, resulting in smaller gradients.

4. Apply Softmax: Scaled scores for each query then passed through the softmax function, converting them into weights that sum to one, representing the attention scores
5. Compute Weighted Sum: Attention weights are used to compute the weighted sum of values. If the input value has high a attention score for a query, it will have a higher representation in the output.
6. Masking: Sometimes, not all key-value pairs are useful, so we will need to mask so that the softmax function assigns a zero weight to those positions.

1.5.3 Multi-Head Attention

Instead of performing a single attention function with d_{model}-dimension keys, values, and queries, it is beneficial to linearly project the queries, keys, and values h times with different, learned, linear projections to d_k, d_k, and d_v dimensions, respectively. On each of these projected versions of queries, keys, and values, we then perform the attention function in parallel, yielding d_v-dimensional output values, which are concatenated and projected again to result in the final values. Multi-head attention allows for the model to jointly attend to information from different representation subspaces at different positions.

$$\text{MultiHead(Q, K, V)} = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$
$$\text{where, head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

2 Models: The Applications

2.1 Bidirection RNN Encoder, Attention-Mechanism Decoder

Bahdanau et al, ICLR 2015, published their novel architecture for neural machine translation, which consisted of a bidirectional RNN as an encoder, and a decoder that emulates searching through a source sentence during decoding a sequence translation.

The bidirectional RNN encoder does the following:

1. Consists of Forward RNN (start to end) and Backward RNN (end to start)
2. encoder processes source sentence: for each word, the encoder produces a set of annotations by taking into account both the preceding words and the following words
3. Annotations from forward and backward RNNs concatenated at each time step to form combined annotation for each word in the source sentence
4. Core Concept: capture info from both past and future context when representing each word

The RNN Decoder with Attention Mechanism does the following:

1. Responsible for generating target translation (unidirectional)
2. Focuses on different parts of the source sentence at different time steps instead of relying solely on a fixed-size context vector
3. Attention Mech. calculates context vector for each time step by taking the weighted average of all the source annotations
4. Aligns source and target sequences and decides which source words are most relevant for predicting the next word in the target sequence
5. context vector + previous state of decoder + previously generated word, used to produce next word in translation

2.2 Transformers: Attention Mechanism

Transformers use multi-head attention in three different ways:

1. In the encoder-decoder layer, queries come from the previous decoder layer, and memory keys and values come from the output of the encoder. Allows the decoder to attend to all positions in the input sequence and mimics encoder-decoder attention mechanisms in Seq2Seq models.
2. Each position in the encoder layer can attend to all positions in the previous layer of the encoder and the Self-attention layers in the decoder allow the position in the decoder to attend to all positions in the decoder up to and including that position.
3. Prevention of leftward information flow in the decoder to preserve autoregressive property uses scaled dot-product attention by masking out all values in the input of the softmax which correspond to illegal connections.

2.3 Position-wise Feed-Forward Neural Networks

Each of the layers in the encoder and decoder contains a fully connected feed-forward neural network which is applied to each position individually and identically. This consists of two linear transformations with a ReLU activation in between each and use variable parameter quantities across different positions.

2.4 Embeddings and Softmax

The model uses pre-learned embeddings to convert input tokens to output tokens to vectors of dimension d_{model}. It also uses learned linear transformations and softmax to convert the output to predicted next-token probabilities. In the embedding layers, the weight matrix is shared between the embeddings and the pre-softmax linear transformation. The embeddings are then multiplied by $\sqrt{d_{\text{model}}}$ in the embedding layers.

2.5 Positional Encoding

Positional Encodings give the model some information about the relative positions of the tokens or the absolute position of the tokens in the sequence. They have the same dimensions d_{model} as the embeddings so that two can be summed. When the positional information is added to the token embedding, both the token's meaning and its position in the sequence.

$$PE_{(\text{pos, 2i})} = \sin(\text{pos}/10000^{2i/d_{\text{model}}})$$
$$PE_{(\text{pos, 2i + 1})} = \cos(\text{pos}/10000^{(2i/d_{\text{model}})})$$

3 BERT: Pre-training of Deep Bidirectional Transformer Models

The world of Natural Language Processing (NLP) has witnessed dramatic shifts in methodologies and paradigms, especially with the advent of deep learning. While models like RNNs and CNNs paved early paths to success, there emerged a challenge of deeply understanding context from both left-to-right and right-to-left simultaneously in text data. Enter BERT (Bidirectional Encoder Representations from Transformers) – an innovative approach that harnesses the power of transformers and a bidirectional mechanism to capture context in a way never achieved before. Devlin et al.'s groundbreaking paper in 2018 introduced BERT, presenting not just a model, but a shift in perspective about how machines understand and interpret human language. By pre-training on vast amounts of text and then fine-tuning for specific tasks, BERT not only demonstrated unparalleled prowess on benchmarks but also set a new standard for NLP research. The motivation behind BERT is clear: to achieve deeper language understanding by holistically analyzing context, breaking away from the unidirectional constraints of the past.

3.1 BERT: Bidirectional Encoder Representations from Transformers

For the BERT framework, there are two parts: pre-training and fine-tuning.
In pre-training, BERT is trained on unlabeled data over different pre-training tasks. For fine-tuning, the BERT model is first initialized with pre-trained parameters and then fine-tuned on labeled data from downstream tasks (feature-based and fine-tuning). Each of the downstream tasks have separate fine-tuned models, however, are both initialized with the same initial pre-trained parameters. The general model of the BERT framework is based on a multi-layer bidirectional transformer encoder in Vaswani et al, 2017.

We define two types of transformer blocks for the sake of simplicity and define the following components, where L is the number of layers, H is the hidden size, and A is the number of self-attention heads.

1. BERT$_{\text{base}}$ is a model block that contains 12 layers, a hidden size of 768, and 12 self-attention heads, as well as 110 trainable parameters.

2. BERT$_{\text{large}}$ is a model block that contains 24 layers, a hidden size of 1024, and 16 self-attention heads, with 340 million trainable parameters.

BERT uses bidirectional self-attention whereas OpenAI's GPT uses constrained self-attention, meaning that every token can only attend to the context on its left (leftward context movement).

3.2 BERT: Input-Output Representations

Input representation is able to unambiguously represent both a single sentence and a pair of sentences (Question, Answer) in one token sequence. A sentence can be an arbitrary span of contiguous text rather than an actual linguistic sentence which refers to the input token sequence to BERT.

Using the WordPiece Embeddings with a 30,000 token vocabulary (corpus), the **first token** of every sequence is always a special classification token. The **final hidden state** corresponding to the **first token** is used as the aggregate sequence representation for classification tasks.

Sentence pairs are packed together into a single sequence.
There are two ways to differentiate the sentences:
1. Separate them with a special token [(SEP)]
2. Add a learned embedding to every token indicating whether it belongs to sentence A or sentence B

For a given token, its input representation is constructed by summing the corresponding token segment, and position embeddings.

3.3 Pre-training BERT

Left-to-right or right-to-left language models are not used to pre-train the BERT model. Instead, the pre-training process involves two unsupervised tasks.

3.3.1 Task 1: Masked LM

Mask a percentage of input tokens at random, then predict those masked tokens. The final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary as in a standard LM.

This allows a bidirectional pre-trained model to be obtained, however, it creates a mismatch between pre-training and fine-tuning since the [MASK] token does not appear during fine-tuning.

Instead, we will not always replace the masked words with the actual [MASK] token. If the i-th token is chosen, we replace the i-th token with (1) the [MASK] 80% of the time (2) a random token 10% of the time, (3) the unchanged i-th token 10% of the time. The token, T_i, is then used to predict the original token with cross-entropy loss.

3.3.2 Task 2: BERT Next Sentence Prediction

Quantion Answering and Natural Language Inference are two important down-stream tasks that are based on the understanding of the relationships between two sentences. Pre-training a binarized next-sentence prediction task can help us train a model that understands sentence relationships, which requires a mono-lingual corpus. When choosing sentences A and B for each pre-training example, 50% of the time, B is the actual next sentence that follows A, labeled as IsNext. The other 50% of the time it is a random sentence from the corpus labeled as NotNext. Pre-training towards Question Answering, however, only sentence embeddings are transferred to down-stream tasks where BERT transfers all parameters to initialize end-task model parameters.

3.3.3 Pre-Training Data

The data used to pre-trained BERT is the BooksCorpus (800M words) and the English Wikipedia which consists of (2500M words). Only text passages are used, and lists, tables, and headers are ignored.

3.4 Fine-Tuning BERT

BERT uses the self-attention mechanism to unify the following two stages: swapping out the appropriate inputs and outputs, and encoding text pairs before applying bidirectional cross-attention.
The self-attention mechanism unifies the two stages by encoding a concatenated text pair with self-attention effectively including bidirectional cross-attention between two sentences.

For each task, task-specific inputs and outputs are plugged into BERT and fine-tuning takes place on all the parameters end-to-end. At the input, sentence A and sentence B from pre-training are analogous to (1) sentence pairs in paraphrasing, (2) hypothesis-premise pairs in entailment (3) quest-passage pairs in question answering and (4) degenerate text-øpair in text classification or sequence tagging. At the output the token representations are fed into an output layer for token-level tasks such as sequence tagging or question answering and the [CLS] representation is fed into an output layer for classification such as an entailment or sentiment analysis.

4 Experiments

In the ”Experiments” section, the authors present a comprehensive evaluation of BERT on various benchmark tasks. They discuss the pre-training procedure, tokenization, and model architectures used in their experiments.

4.1 Pre-training and Fine-tuning

The authors describe the pre-training process of BERT on a large corpus of text data. They train the model using masked language modeling (MLM) objectives, where a certain percentage of words in a sentence are randomly masked and the model is trained to predict these masked words based on the surrounding context. They also discuss the fine-tuning process on downstream tasks, where BERT's pre-trained weights are fine-tuned on task-specific data.

4.2 Tokenization

The authors emphasize the importance of subword tokenization using WordPiece tokenization. This allows BERT to handle out-of-vocabulary words effectively and capture morphological nuances. They provide details on the tokenization process and demonstrate its advantages through experiments.

4.3 Model Architectures

The authors present two model sizes, $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$, with varying numbers of layers, hidden dimensions, and self-attention heads. They discuss how these model sizes affect performance and training time.

4.4 Benchmark Tasks

The authors evaluate BERT on various natural language understanding tasks, including question answering, sentence pair classification, and named entity recognition. They report competitive or state-of-the-art results across multiple benchmarks, demonstrating BERT's effectiveness in capturing contextual information.

5 Ablation Studies

In the "Ablation Studies" section, the authors perform ablation experiments to analyze the impact of different components and design choices on BERT's performance.

5.1 Effect of Masked LM

The authors investigate the contribution of the masked language modeling (MLM) objective during pre-training. They compare BERT's performance when trained with MLM alone and when trained with both MLM and next sentence prediction (NSP). Results show that the MLM objective plays a more significant role in learning contextual representations.

5.2 Ablation of Layers

The authors study the effect of the number of layers in $\text{BERT}_{\text{BASE}}$ by training models with different layer combinations. They observe that a minimal configuration of layers still performs well, indicating that deep representations are not always necessary for strong performance.

5.3 Ablation of Attention Heads

The authors analyze the impact of attention heads in BERT's self-attention mechanism. They find that reducing the number of attention heads has a limited impact on performance, suggesting that BERT is robust to variations in this aspect.

5.4 Comparison with ELMo and OpenAI GPT

The authors compare BERT with previous models, ELMo and OpenAI GPT, on multiple tasks. BERT consistently outperforms these models, showcasing its ability to capture bidirectional contextual information.

5.5 Effect of Model Size

The authors discuss the trade-off between model size and performance. They show that increasing the model size leads to performance gains, but the improvement saturates beyond a certain point due to data limitations.

5.6 Effect of Training Data Size

The authors examine the impact of training data size on BERT's performance. They observe that increasing the amount of pre-training data significantly improves performance, highlighting the importance of large-scale training corpora.

5.7 Effect of Task-specific Layers

The authors analyze the use of task-specific layers during fine-tuning. They find that task-specific layers are crucial for achieving optimal performance on downstream tasks.

5.8 Analysis of Attention Patterns

The authors visualize attention patterns in BERT and provide insights into how attention heads capture different linguistic phenomena and relationships.

Citation List

1. Word Embeddings
Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26.

2. Attention Mechanism
Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473.*

3. Transformers and Self-Attention
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

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