Chapter 3 Deep learning fundamentals for NL

In this chapter, we will explore the foundational principles of deep learning that have revolutionized the field of Natural Language Processing (NLP). Deep learning plays a crucial role in modern NLP, enabling machines to process, understand, and generate human language with unprecedented accuracy. This chapter provides the essential knowledge necessary to understand transformer-based architectures and the Hugging Face Diffusion library.

First, we will delve into the basics of deep learning, covering neural networks and their significance in NLP. We will then explore key concepts such as tokenization and word embeddings, which are critical for text processing tasks. Additionally, we will introduce attention mechanisms, a key innovation that enhances model performance by allowing models to focus on relevant parts of input sequences. Lastly, we will examine transformer-based architectures, which represent the state-of-the-art in NLP models today.

By the end of this chapter, you will have a solid understanding of the core components of deep learning applied to NLP and will be well-prepared to explore more advanced models and techniques in the subsequent chapters. This knowledge will help you apply deep learning techniques to real-world NLP tasks, such as text classification, named entity recognition, and machine translation, all while leveraging the capabilities of Hugging Face Diffusion.

In the next illustration, a schematic diagram of a deep neural network: architecture

1. an architecture of DNN model comprised of input, hidden, and output layers
2. dropout regularization method that controls the connection of the neurons in the hidden layers.

A diagram of a neural network

Description automatically generated

Figure 1 Schematic diagram of a deep neural network: architecture

Deep Learning Basics for NLP

Deep learning forms the backbone of modern NLP by using neural networks to process language data. Neural networks consist of multiple layers of interconnected neurons that can learn patterns from data. Unlike traditional rule-based approaches, deep learning models can learn hierarchical representations from vast datasets, which makes them ideal for understanding the complexities of human language.

In this section, we will introduce neural networks and their relevance to NLP, differentiating them from earlier approaches. We will also explain why deep learning has been so effective in advancing NLP, including improvements in tasks like text classification, sentiment analysis, and translation.

Tokenization and Word Embeddings

Tokenization fundamentally divides raw text into smaller units, such as words or sub-words, allowing models to interpret language at a granular level. Word embeddings then convert these words into dense vector representations, capturing semantic relationships between words based on their context.

We will explore different tokenization techniques and introduce popular word embedding methods like Word2Vec, GloVe, and FastText, all of which have significantly enhanced the way machines understand text. These techniques are critical for downstream NLP tasks and are the foundation for training transformer-based models.

Attention Mechanisms in NLP

Attention mechanisms are a breakthrough in NLP, allowing models to focus on specific parts of input sequences when making predictions. This innovation has improved the performance of NLP tasks, such as machine translation, sentiment analysis, and summarization.

In this section, we will explain the concept of attention, including self-attention, and how it enhances model accuracy. We will also discuss practical applications of attention mechanisms in real-world tasks, providing a clear understanding of why attention plays a key role in modern NLP models.

Transformer-Based Architectures

Transformers are the backbone of modern NLP, enabling parallel processing of entire sequences of text, unlike previous architectures that relied on sequential processing (such as RNNs and LSTMs). Transformers use self-attention mechanisms to capture dependencies between words, regardless of their position in a sequence.

In this section, we will break down the components of transformer models and explain their advantages over traditional architectures. We will also introduce popular transformer models like BERT, GPT, and T5, which play a crucial role in a wide range of NLP tasks. Understanding these models will be critical for leveraging the Hugging Face Diffusion library in subsequent chapters.

Basics of deep learning for NLP

Deep learning has become a cornerstone in modern Natural Language Processing (NLP) by enabling machines to process and understand human language with remarkable accuracy. Before diving deeper into its applications in NLP, it is essential to grasp the fundamental concepts that form the backbone of deep learning models. These concepts not only explain how deep learning works but also highlight its relevance to solving complex language tasks in NLP.

Introduction to deep learning concepts

Deep learning represents a change in basic assumptions in artificial intelligence, enabling machines to learn from vast amounts of data and automatically discover intricate patterns. At the heart of deep learning are neural networks, computational models inspired by the human brain's interconnected neurons.

Neural networks consist of layers of interconnected nodes (neurons) that process information hierarchically. Each neuron applies a weighted sum of inputs, followed by an activation function to produce an output. Through backpropagation and gradient descent, neural networks adjust these weights to minimize prediction errors during training, making them highly adaptable to complex tasks like natural language processing (NLP).

Overview of neural networks and their relevance to NLP

In the context of NLP, neural networks play a pivotal role in learning representations of language. Traditional methods relied on handcrafted rules and statistical models, which often struggled to capture semantic nuances and context-dependent meanings present in human language. Neural networks, however, excel at automatically extracting features from raw text, making them suitable for tasks such as sentiment analysis, machine translation, and text generation.

**Example: Sentiment Analysis** Consider a neural network designed for sentiment analysis of movie reviews. By training on a dataset of labeled reviews (positive or negative sentiment), the network learns to classify recent reviews based on learned features like word frequencies, syntactic structures, and sentiment-bearing phrases.

Differences between traditional NLP methods and deep learning-based approaches

Traditional NLP methods, such as rule-based systems and statistical models like Hidden Markov Models (HMMs) or Conditional Random Fields (CRFs), relied heavily on linguistic rules and hand-engineered features. While effective for specific tasks, these approaches often required extensive manual effort to adapt to new languages or domains and struggled with complex language phenomena.

Figure 2 below pprovides a comparative chart showing the evolution from traditional NLP methods to deep learning approaches, illustrating key differences in methodology, performance metrics, and adaptability.

A diagram of a machine learning process

Description automatically generated

Figure 2 Comparison between classical NLP model and deep learning-based NLP model.

In contrast, deep learning-based approaches leverage neural networks to automatically learn representations directly from data, without the need for explicit rule definitions. This data-driven approach enables more flexible and adaptive models capable of handling diverse linguistic patterns and domains.

**Comparison Example: Machine Translation** Traditional statistical machine translation systems relied on predefined translation rules and statistical models trained on parallel corpora. Deep learning-based systems like transformer models, such as Google's Neural Machine Translation (GNMT) and subsequent architectures like BERT and GPT, have demonstrated superior performance by learning to map sequences directly from one language to another based on large-scale multilingual corpora.

Tokenization and word embeddings

Before we dive into the practical applications of tokenization and word embeddings, it is essential to grasp why these concepts are critical to Natural Language Processing (NLP). Tokenization is the first step in processing text, transforming raw data into a format that models can interpret. Word embeddings take this process further by mapping words to vectors that capture semantic meaning. Together, these processes form the bedrock of modern NLP tasks, enabling models to understand and analyze language effectively.

Understanding tokenization and its importance in NLP

Tokenization serves as a foundational preprocessing step in NLP, breaking down raw text into smaller units called tokens. These tokens might be words, sub-words, or even characters, depending on the tokenization strategy used. The main objective of tokenization is to segment the text into meaningful units for further processing by NLP models.

Figure below illustrates that choosing the right tokenization method depends on the specific NLP task at hand.

A diagram of a diagram

Description automatically generated

Figure 3 Different tokenization methods.

**Importance Example: Named Entity Recognition** Consider a sentence: "Apple is planning to open a new store in Tokyo." Tokenization here would segment the sentence into tokens like "Apple", "is", "planning", "to", "open", "a", "new", "store", "in", "Tokyo". This tokenized representation enables downstream tasks such as named entity recognition to identify entities like "Apple" (organization) and "Tokyo" (location) accurately.

Different tokenization techniques

Tokenization is a critical step in NLP, as it breaks down text into manageable units for processing. The choice of tokenization technique directly impacts the model’s ability to understand and represent the underlying structure of the text. Depending on the task at hand, different tokenization methods can be employed to capture varying levels of granularity, from words to sub words and even individual characters. Each technique has its strengths and is better suited for specific tasks, making it essential to select the right one to maximize model performances:

On the next figure, we illustrate tokenization In natural language processing: Tokenization is breaking down a sentence into smaller phrases/units called “tokens” The types of tokenization:

A diagram of a type of tokenization

Description automatically generated

Figure 4 Types of Tokenization in NLP

1. **Word tokenization:** Splits text into words based on whitespace or punctuation boundaries. Example: "Hello, world!" → ["Hello", ",", "world", "!"]
2. **Subword tokenization:** Splits words into smaller units that may or may not correspond to meaningful subword components. Example: "unbelievable" → ["un", "believable"]
3. **Character-level Tokenization:** Treats each character in the text as a separate token. Example: "Hello" → ["H", "e", "l", "l", "o"]

Each technique offers trade-offs in terms of granularity and information retention, depending on the language and specific NLP task requirements.

Introduction to word embeddings

Word embeddings represent words as dense vectors in a continuous vector space. Each word corresponds to a high-dimensional vector determined by its context and usage within a text corpus. These embeddings capture semantic meaning, positioning words with similar contexts closer together in the vector space.

The figure below Includes visualizations of word embeddings in a vector space, demonstrating how words with similar meanings cluster together.

A diagram of a graph

Description automatically generated

Figure 5 Visualizations of word embeddings in a vector space.

**Semantic Meaning Example: Word2Vec** Word2Vec, a popular word embedding model, learns representations by predicting the surrounding words (skip-gram model) or predicting a word given its context (continuous bag of words model). This contextual learning approach allows Word2Vec to capture nuanced semantic relationships, such as similarity and analogy (Mikolov et al., 2013).

Next, the figure graphically explains Word2Vec, a shallow two-layer neural network designed to predict the surrounding linguistic context of words. It processes a diverse collection of text as input and generates a high-dimensional vector space. Each word in the corpus corresponds to a unique vector in this space, with words that appear in similar contexts positioned closely together. Word2Vec efficiently generates word embeddings from raw text and offers two models: Continuous Bag-of-Words (CBOW) and Skip-Gram, which share similar algorithmic foundations. (Gilyadov, 2017)

A diagram of a diagram

Description automatically generated

Figure 6 Word2Vec Explained.

How word embeddings capture semantic meaning

Word embeddings encode semantic meaning through distributional semantics, where words with similar meanings tend to have similar vector representations. This property enables NLP models to generalize better across different contexts and tasks, improving performance on tasks like sentiment analysis, machine translation, and document classification.

**Illustrative Example: GloVe** Global Vectors for Word Representation (GloVe) leverages co-occurrence statistics from a large corpus to create embeddings that emphasize global word-word relationships. This statistical approach captures both syntactic and semantic information, making GloVe embeddings effective for a wide range of NLP tasks (Pennington et al., 2014).

Attention mechanisms in NLP

As we continue to explore the essential components of NLP models, it becomes clear that not all input data carries the same level of importance in each context. This is where attention mechanisms come into play. These mechanisms allow models to focus on the most relevant parts of the input sequence, significantly improving their ability to process complex language tasks.

Introduction to the concept of attention

Attention mechanisms in NLP mimic human cognitive processes, allowing models to selectively focus on various parts of input data (e.g., words in a sentence) when making predictions or generating outputs. This selective focus enhances the handling of long-range dependencies and improves NLP performance by assigning varying degrees of importance to different elements.

**Example: Neural Machine Translation:** In neural machine translation (NMT), attention mechanisms enable the model to align and selectively address relevant words in the source sentence when generating each word in the target sentence. This dynamic attentional focus improves translation accuracy by ensuring that the model incorporates contextually relevant information (Bahdanau et al., 2015).

How attention mechanisms improve model performance in NLP Tasks

Attention mechanisms different advantages in NLP tasks:

1. **Improved Contextual Understanding:** By focusing on relevant parts of the input sequence, attention mechanisms enhance the model's ability to capture nuanced relationships and context, crucial for tasks like sentiment analysis and question answering.
2. **Efficient Handling of Long Sequences:** Traditional models like RNNs struggle with long-range dependencies due to vanishing or exploding gradients. Attention mechanisms mitigate this issue by linking relevant tokens, reducing the reliance on sequential processing.
3. **Enhanced Interpretability:** Attention weights provide insights into which parts of the input are most influential for model predictions, aiding interpretability, and model debugging.

Types of attention mechanisms

Attention mechanisms have become an integral part of modern NLP architectures, allowing models to focus on relevant parts of input sequences. The adaptability and efficiency of these mechanisms have spurred the development of various forms, each tailored to meet specific task requirements. In this section, we will explore two commonly used types of attention mechanisms in NLP—self-attention and cross-attention—and examine how they play crucial roles in improving model performance:

1. **Self-Attention (Intra-Attention):** Within a single sequence, self-attention computes attention weights based on the relationships between various positions of the same input sequence. This mechanism is foundational in transformer architectures (Vaswani et al., 2017). Next figure below diagrams illustrates self-attention and multi-head attention mechanisms within transformer architectures.

A diagram of a machine

Description automatically generated

Figure 7 Self-attention and Multi-head mechanism within Transformer architecture.

1. **Cross-Attention (Inter-Attention):** In tasks involving multiple sequences (e.g., machine translation), cross-attention allows the model to address relevant parts of one sequence (source) when processing another sequence (target).

Practical examples of attention in NLP models

Attention mechanisms have not only reshaped how NLP models process information but have also laid the foundation for state-of-the-art models. By allowing models to dynamically focus on relevant parts of the input data, attention has improved performance on a variety of tasks. In this section, we will explore prominent models that harness the power of attention, demonstrating how these mechanisms contribute to achieving innovative results in NLP. Through practical examples like BERT and the Transformer architecture, we will highlight how attention mechanisms facilitate more accurate language understanding and generation.

**Example: BERT (Bidirectional Encoder Representations from Transformers)** BERT utilizes self-attention to capture bidirectional contextual information from the entire input sequence. During pre-training, BERT learns to predict masked words using contextual embeddings derived from self-attention layers, achieving state-of-the-art results on various NLP benchmarks (Devlin et al., 2019).

**Example: Transformer Architecture** The transformer architecture employs multi-head attention, where attention spans across multiple projections (heads) of both the query and key vectors. This parallel processing enhances the model's capacity to capture diverse dependencies and improves computational efficiency (Vaswani et al., 2017).

Transformer-based architectures

As natural language processing (NLP) tasks have grown increasingly complex, traditional sequential models like RNNs and LSTMs began to show limitations, particularly in handling long-range dependencies and large datasets. Transformer-based architectures have since emerged as a groundbreaking alternative, addressing these challenges with their ability to process input data in parallel and capture global context efficiently. In the following section, we will explore the fundamental components of transformer models and the key innovations that distinguish them from earlier architectures.

Refer also to fugue seven above illustrating the architecture of transformers, with emphasis on the encoder-decoder structure, self-attention mechanism, and flow of information through layers.

Overview of transformer models and their components

Transformer models revolutionized NLP by introducing a novel architecture based entirely on self-attention mechanisms, eschewing sequential processing like RNNs and LSTMs. Central to transformers are attention mechanisms that allow capturing global dependencies and context in parallel, leading to significant improvements in various NLP tasks.

**Example: Transformer Architecture** The transformer architecture consists of encoder and decoder stacks. Each stack comprises multiple layers, with each layer having two main components: self-attention mechanisms and feed-forward neural networks. The self-attention mechanism enables the model to weigh the importance of each input token based on its relation to other tokens, capturing dependencies across the entire sequence simultaneously (Vaswani et al., 2017).

Key innovations introduced by transformer architectures

Transformer models have revolutionized NLP by introducing novel techniques that overcome the limitations faced by previous architectures like RNNs and LSTMs. The innovations at the core of transformers allow for improved efficiency, scalability, and performance on a wide variety of NLP tasks. By enabling parallel processing and incorporating advanced mechanisms such as self-attention, transformers have set new benchmarks in language modeling, text generation, and translation tasks. The following key innovations are what make transformer architectures so impactful in the field of NLP.

1. **Self-Attention Mechanism:** Unlike traditional architectures relying on sequential processing, transformers employ self-attention to compute representations of input tokens based on their relationships with other tokens. This mechanism allows transformers to capture long-range dependencies efficiently.
2. **Parallelization:** Transformers facilitate parallel computation of attention across tokens, enhancing efficiency compared to sequential models. This parallelization is critical for scaling to larger datasets and longer sequences without increasing computational cost disproportionately.
3. **Layer Normalization and Residual Connections:** To stabilize training and enable deeper architectures, transformers employ layer normalization and residual connections between layers. These techniques mitigate issues like vanishing gradients and enable effective training of very deep networks (Vaswani et al., 2017).

Comparison with previous architectures (e.g., RNNs, LSTMs)

Transformer architectures have transformed the landscape of NLP, primarily due to their ability to manage long-range dependencies and scale efficiently, aspects that traditional architectures such as RNNs and LSTMs struggled with. By eliminating the sequential bottlenecks of earlier models, transformers enable parallel processing and significantly boost performance on tasks that require a deeper understanding of global context. The comparison below highlights the key differences that make transformers a superior choice for modern NLP challenges:

**Long-range Dependencies:** While RNNs and LSTMs process sequences sequentially, transformers can capture dependencies across the entire sequence simultaneously through self-attention, making them more effective for tasks requiring understanding of global context.

**Scalability:** Transformers' parallelizable nature allows them to manage larger datasets and longer sequences more efficiently compared to sequential models, which suffer from computational constraints as sequence length increases.

Understanding the transformer’s self-attention mechanism

Self-attention lies at the heart of transformer models, enabling them to process entire sequences of data in parallel while efficiently capturing dependencies between distant words in a text. This mechanism assigns different attention scores to input tokens based on their relevance to one another, allowing the model to prioritize contextually important tokens during both language understanding and generation tasks. To illustrate this, we can examine how BERT, a well-known transformer model, utilizes self-attention to achieve state-of-the-art results in various NLP applications.

**Example: BERT (Bidirectional Encoder Representations from Transformers)** BERT utilizes multi-head self-attention to capture bidirectional dependencies in language modeling tasks. Each attention head attends to various aspects of the input, allowing BERT to model complex relationships within and between sentences effectively (Devlin et al., 2019).

Introduction to Popular Transformer Models

1. **BERT (Bidirectional Encoder Representations from Transformers):** Introduced by Google, BERT pre-trains a bidirectional transformer encoder on large text corpora for various downstream tasks like question answering and sentiment analysis (Devlin et al., 2019).
2. **GPT (Generative Pre-trained Transformer):** Developed by OpenAI, GPT uses an autoregressive decoder for generating coherent text and applies to various tasks such as text completion and dialogue generation (Radford et al., 2018).
3. **T5 (Text-to-Text Transfer Transformer):** Introduced by Google, T5 unifies diverse NLP tasks into a single text-to-text format, demonstrating robust performance across translation, summarization, and question answering tasks (Raffel et al., 2019).

**Practical application example: Using an LSTM network for text generation**

To demonstrate how to apply an LSTM network for text generation, we will walk through setting up a simple LSTM model using TensorFlow. This model simulates a basic version of a task typically managed by more complex models like transformers. In this example, we will work with a small text corpus, tokenize it at the character level, and prepare the dataset for training.

First, the text data undergoes tokenization, mapping each character in the text to an integer for model input. The dataset is prepared by sliding a window across the text, capturing sequences of characters (sentences) and the next character that the model will predict (next\_chars).

Next, we define and build the LSTM model. The model consists of an embedding layer that transforms input characters into dense vectors, an LSTM layer that processes the sequences, and a dense output layer that predicts the next character in the sequence. We compile the model with the categorical cross-entropy loss function and the Adam optimizer, then train it for multiple epochs.

After training the model, we can generate new text by choosing a random starting point from the input text. The model then predicts subsequent characters, generating a new sequence based on the learned patterns.

Here is the code that demonstrates this process:

``python

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Embedding

# Sample text data (could be expanded to a large corpus)

text = "Hello world! Welcome to the world of deep learning for natural language processing."

# Character-level tokenization

chars = sorted(set(text))

char\_to\_int = dict((c, i) for i, c in enumerate(chars))

int\_to\_char = dict((i, c) for i, c in enumerate(chars))

# Prepare the dataset

max\_length = 10

step = 1

sentences = []

next\_chars = []

for i in range(0, len(text) - max\_length, step):

sentences.append(text[i: i + max\_length])

next\_chars.append(text[i + max\_length])

x = np.zeros((len(sentences), max\_length, len(chars)), dtype=np.float32)

y = np.zeros((len(sentences), len(chars)), dtype=np.float32)

for i, sentence in enumerate(sentences):

for t, char in enumerate(sentence):

x[i, t, char\_to\_int[char]] = 1

y[i, char\_to\_int[next\_chars[i]]] = 1

# Build the LSTM model

model = Sequential([

Embedding(input\_dim=len(chars), output\_dim=50, input\_length=max\_length),

LSTM(128),

Dense(len(chars), activation='softmax')

])

# Compile the model

model.compile(loss='categorical\_crossentropy', optimizer='adam')

# Train the model

model.fit(x, y, batch\_size=128, epochs=60)

# Generate text

def generate\_text(length):

start\_index = np.random.randint(0, len(text) - max\_length - 1)

generated = ''

sentence = text[start\_index: start\_index + max\_length]

generated += sentence

for i in range(length):

x\_pred = np.zeros((1, max\_length, len(chars)))

for t, char in enumerate(sentence):

x\_pred[0, t, char\_to\_int[char]] = 1.

preds = model.predict(x\_pred, verbose=0)[0]

next\_index = np.argmax(preds)

next\_char = int\_to\_char[next\_index]

generated += next\_char

sentence = sentence[1:] + next\_char

return generated

# Example of generated text

print(generate\_text(50))

``

This example offers a basic introduction to using LSTM models for sequence prediction and text generation tasks, demonstrating their ability to capture patterns within sequences. By understanding how these models function, you will build a foundation for tackling more complex text generation tasks in NLP.

Wrap-up for Chapter 3

In this chapter, we set out to establish a solid foundation in deep learning concepts essential for natural language processing (NLP). We explored how deep learning has revolutionized NLP by shifting from traditional, rule-based methods to neural network-driven models. We examined tokenization and word embeddings, understanding their importance in breaking down and representing text meaningfully. We also discussed how attention mechanisms, particularly in transformer models, have enhanced the ability of neural networks to capture long-range dependencies and improve model performance across various tasks. Finally, we introduced transformer architectures, which have significantly outperformed older models like RNNs and LSTMs in both scalability and accuracy.

By addressing these key concepts, we have met our goal of providing the theoretical and practical knowledge necessary for understanding the core technologies behind modern NLP systems. This groundwork will serve as a steppingstone as we transition into more advanced topics in subsequent chapters.

Lead-in to Chapter 4

In Chapter 4, we will build upon the deep learning fundamentals discussed here and dive into the practical application of transformer models using the Hugging Face Diffusion library. We will explore how to leverage pre-trained models for NLP tasks through transfer learning, and you will learn how to fine-tune these models for domain-specific applications. Additionally, we will cover strategies for deploying these models effectively in real-world environments, ensuring they are scalable and efficient. By the end of Chapter 4, you will have direct experience with state-of-the-art NLP tools, positioning you to implement advanced deep learning techniques in practical scenarios.

Let us move forward to see how these foundational principles translate into innovative solutions in NLP with Hugging Face Diffusion.

References

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. *preprint*.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics, 5*, 135–146.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., & Amodei, D. (2020). Language Models are Few-Shot Learners. *preprint*.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research, 12*, 2493–2537.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* *1*, pp. 4171–4186. Long and Short Papers.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., & Houlsby, N. (2020). An Image is Worth 16x16 Words. *preprint*.

Gilyadov, J. (2017, March 12). *Vector Representations of Words.* Retrieved from Hacker's Blog: https://israelg99.github.io/2017-03-22-Vector-Representations-of-Words/

Goldberg, Y. (2016). A Primer on Neural Network Models for Natural Language Processing. *Journal of Artificial Intelligence Research, 57*, 345–420.

Jurafsky, D., & Martin, J. H. (2019). *Speech and Language Processing* (3rd ed.). Pearson.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., & Stoyanov, V. (2020). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *preprint*.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *preprint*.

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 (NIPS)*, 3111–3119.

Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1532-1543.*

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving Language Understanding by Generative Pre-training. *Open AI Research*. Retrieved from https://cdn.openai.com/research-covers/language-unsupervised/language\_understanding\_paper.pdf

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., & Liu, P. J. (2019). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *preprint*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., & Polosukhin, I. (2017). Attention is All You Need. 5998–6008.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. (2019). XLNet: Generalized Autoregressive Pretraining for Language Understanding. *Advances in Neural Information Processing Systems (NeurIPS)*, 5754–5764.