Language Modeling with Deep Neural Networks

Project of AI3611 Intelligent Perception and Cognition Practice, 2024 Spring, SJTU

Xiangyuan Xue (521030910387)

School of Electronic Information and Electrical Engineering

Abstract

Language modeling is a fundamental task in natural language processing. In this project, we research on language modeling with deep neural networks, where we explore multiple architectures including RNNs, LSTM, GRU, transformers and GPTs. We introduce the principles of the models and train the models on the given dataset to compare the performance of different architectures. We also conduct ablation studies to investigate the effects of different hyperparameters on the performance. Besides, we give an explanation to the anomaly about the inadequate performance of the Transformer-based models.

1 Introduction

A language model is a probabilistic model of a natural language, which gives the probability that a sequence of words is a valid sentence. To be specific, given a sequence of words $w_1, w_2, \ldots, w_n$, we expect the model to predict the joint probability $P(w_1, w_2, \ldots, w_n)$. Language modeling is first applied in speech recognition [8] to avoid generating unreasonable sentences. Until now, language modeling has been used in extensive fields such as machine translation [1], information retrieval [12], grammar induction [6] and natural language generation [10].

Traditionoal language modeling tends to use statistical models such as n-gram models [11] and skip-gram models [4]. With the advent of deep learning, these models are superseded by neural networks, which are capable of learning complex representations. Feedforward Neural Networks (FNNs) [14] are first applied to language modeling, but they can only accept fixed-length inputs. Recurrent Neural Networks (RNNs) [3] are then proposed to handle variable-length sequences, which are widely used in sequence modeling. However, RNNs suffer from the vanishing gradient problem, which makes it hard to capture long-range dependencies, so Long Short-Term Memory (LSTM) [5] and Gated Recurrent Unit (GRU) [2] are proposed to improve the performance of RNNs. Recently, transformers [15] have emerged as a novel architecture which makes it possible to capture long-range dependencies and model complex relationships. Generative Pretrained Transformers (GPTs) [13] have achieved recognized state-of-the-art results in language modeling.

In this project, we will explore multiple architectures for language modeling, including RNNs, LSTM, GRU, transformers and GPTs. We will train the models on the given dataset and compare the performance of different architectures. We will research how the hyperparameters affect the performance and try to minimize the perplexity within specific numbers of parameters. In Section 2, we will introduce the principles of the models. In Section 3, we will describe our experiments and analyze the results. In Section 4, we will summarize our findings and provide some insights.

2 Method

Recurrent Neural Networks (RNNs) and Transformers are typical architectures for sequence modeling. In this project, we explore multiple architectures, including RNNs, LSTM, GRU, transformers and GPTs. All the models share the same training process, where we feed them with a sequence of words and expect them to predict the next word. The models output the log-likelihood of each word in the vocabulary, and we use the negative log-likelihood as the loss function, namely

$$\mathcal{L}_{\text{NLL}}(x) = -\log P(x) \tag{1}$$

In the following parts, we will introduce the principles of different architectures respectively.

2.1 Recurrent Neural Networks

Proposed in the 1980s, RNNs [3] are one of the most popular models for sequence modeling. The key idea of RNNs is to use the hidden state to capture the context information, which is updated at each time step. The hidden state at time step t is computed as

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) \tag{2}$$

where $\boldsymbol{x}_t$ is the input at time step t, $\boldsymbol{h}_{t-1}$ is the hidden state at time step $t-1$, and f is a non-linear function. A common choice of f is the hyperbolic tangent function, which is computed as

$$f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) = \tanh(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{b}) \tag{3}$$

where $\boldsymbol{W}_h$, $\boldsymbol{W}_x$, and $\boldsymbol{b}$ are the parameters to be learned. Note that the hidden state at the last time step is typical of being used as the context representation.

We also try to use Long Short-Term Memory (LSTM) [5] and Gated Recurrent Unit (GRU) [2] as the improved units of RNNs. LSTM applies multiple gates to control the information flow, maintaining a long-term memory. GRU modifies the design of the gates to simplify the structure of LSTM, bringing faster training speed while maintaining comparable performance.

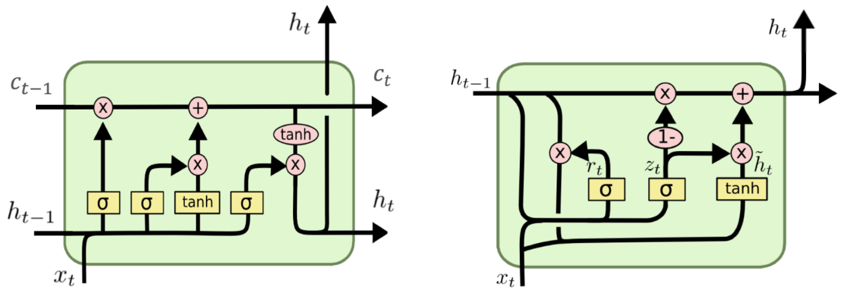

Figure 1: The structure of the LSTM and GRU cells. LSTM applies multiple gates to control the information flow, maintaining a long-term memory. GRU modifies the design of the gates to simplify the structure of LSTM, bringing faster training speed while maintaining comparable performance.

2.2 Transformers

Recently, transformers [15] have emerged as a novel architecture for sequence modeling, achieving state-of-the-art results in multiple domains. Transformers show strong capabilities in capturing long-range dependencies and modeling complex relationships, which makes learning from tremendous amounts of data feasible. The core of transformers is the attention mechanism, which computes the weighted sum of the values based on the similarity between the query and the keys. The attention mechanism can be formulated as

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}})\boldsymbol{V} \tag{4}$$

where $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ are the query, key and value matrices respectively, and d_k is the dimension of the key. The transformer encoder is composed of multiple layers of multi-head attention and feed-forward networks. The key idea of multi-head attention is to project the query, key and value into multiple subspaces and extract the dependencies from different representations, which enhances the representation ability of the model. Note that we only use the transformer encoder here to build a basic language model. The causal transformer decoder will be explored in GPTs.

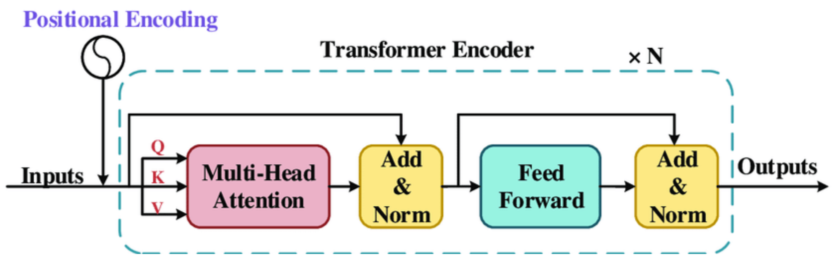

Figure 2: The structure of the transformer encoder. The transformer encoder is composed of multiple layers of multi-head attention and feed-forward networks. Positional embedding is applied to encode the positional information of the inputs.

We should notice that the transformer encoder is not a recurrent architecture, which cannot capture accurate temporal information. To solve this problem, positional embedding is applied to encode the positional information of the inputs. We follow the original paper [15] to use sine and cosine functions with different frequencies as the positional embedding.

2.3 Generative Pretrained Transformers

Generative Pretrained Transformers (GPTs) [13] are a series of models that extend the transformer architecture to autoregressive language modeling. The key idea of GPTs is to use the transformer decoder as the backbone and train the model to predict the next word given the previous words. Different from vanilla transformers, GPTs use a decoder-only architecture and apply a causal self-attention mechanism to ensure that the model only attends to the previous words, which avoids the information leakage from the future. Researchers believe that the decoder-only architecture works better in generative tasks due to the causal training objective. Much industrial practice has proved that GPTs have stronger scalability and are capable of generating high-quality text [10].

Note that it is unnecessary to implement the full training process of GPTs since we focus on simple language modeling, where the number of parameters is strictly limited. We follow the MinGPT paradigm to build a small-scale GPT model, which consists of a stack of transformer decoder blocks. Therefore, the training process will be exactly the same as the transformer encoder.

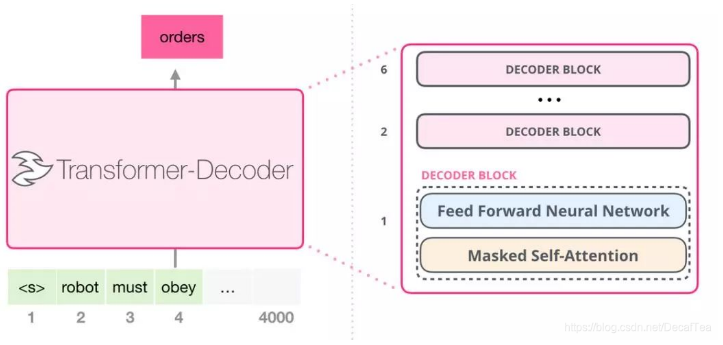

Figure 3: The structure of the GPT-2 model. Different from vanilla transformers, GPTs use a decoder-only architecture and apply a causal self-attention mechanism to ensure that the model only attends to the previous words, which avoids the information leakage from the future.

3 Experiments

In this section, we will implement and train the language models on the given dataset and compare the performance of different architectures. We will research how the hyperparameters affect the performance and try to minimize the perplexity within specific numbers of parameters. Considering the scalability of the code, we refactor the framework to support more features including advanced architectures, flexible hyperparameters and efficient optimizers. We introduce packages like `hydra` and `tensorboard` to manage the experiment configurations and visualize the training process. In the following parts, we will present the settings and analyze the results.

3.1 Experiment Setup

We employ the given GigaSpeech dataset to train the language models. The dataset contains 908506 sentences in the training set, 5713 sentences in the validation set and 19884 sentences in the test set. The vocabulary size is 27710 and the maximum length of the input sequence is limited to 1024.

In the default setting, we use an embedding size of 256, a hidden size of 512 and a layer number of 2. All the layers apply dropout with a rate of 0.2. The length of the input sequence is set to 32, which determines the maximum time step for back propagation. We train the model for 30 epochs with a batch size of 32 using Adam [7] or AdamW [9] optimizer with a learning rate of 10^{-4}. Note that a larger learning rate may work in some cases, but may also lead to divergence in other cases. We clip the gradient norm to 1.0 to prevent gradient explosion and reduce the learning rate by a factor of 0.5 if the validation perplexity fails to decrease in a single epoch.

To evaluate the performance of the language models, we employ the perplexity as the primary metric. Perplexity is a common metric in language modeling tasks, which measures the uncertainty of the prediction. Mathematically, it can be formulated as

$$\text{PPL} = \exp\left[-\frac{1}{N}\sum_{i=1}^{N}\log P(w_i|w1, w2, \ldots, w_{i-1})\right] \tag{5}$$

where N is the number of words in the sentence. Since we use the log-likelihood as the loss function, the perplexity is equivalent to the exponential of the loss. Typically, a lower perplexity indicates a better performance of the model, which effectively measures the quality of the language modeling.

3.2 Model Performance

To compare the performance of the language models with different architectures, we implement the RNN, LSTM, GRU, Transformer and GPT models with various layer numbers and hidden sizes. All the hyperparameters follow the default setting except for the layer number and the hidden size. The number of parameters, the loss and perplexity on the test set are presented in Table 1.

From the table we can see that the GRU model with 4 hidden layers and 512 hidden units achieves the best performance with a perplexity of 142.11. The GPT model with 8 hidden layers and 1024 hidden units also performs well with a perplexity of 144.40. Such results indicate that both the RNN-based and Transformer-based models can achieve satisfactory performance on language modeling tasks. For the RNN-based models, the LSTM and GRU models outperform the vanilla RNN model, which demonstrates the effectiveness of the gating mechanism. In addition, increasing the number of hidden layers does not always lead to better performance for the RNN-based models, but increasing the hidden size can effectively reduce the perplexity. The Transformer-based models, however, show a better scalability with the number of parameters. Both increasing the number of hidden layers and the hidden size can effectively reduce the perplexity. In addition, the GPT model slightly outperforms the Transformer model, which supports the theory that the causal learning strategy can improve the performance of generative models, especially in natural language processing tasks.

3.3 Ablaition Study

To investigate the impact of other hyperparameters on the performance of the language models, we conduct an ablation study on the embedding size, dropout rate and sequence length. We fix the architecture to 4 hidden layers and 512 hidden units for the RNN, LSTM and GRU models, and 8 hidden layers and 1024 hidden units for the Transformer and GPT models. Other hyperparameters follow the default setting. We vary the embedding size in the set $\{32, 64, 128, 256, 512\}$, the dropout rate in the set $\{0.1, 0.2, 0.3, 0.4, 0.5\}$, and the sequence length in the set $\{4, 8, 16, 32, 64\}$ to observe how the performance changes. The results are presented in Figure 4, 5 and 6.

From the figures we can see that the embedding size, dropout rate and sequence length have different impacts on the performance of the language models. Typically, a larger embedding size provides efficient representation ability for the word embeddings, which contains more useful information

Table 1: The performance of the language models with different architectures. The numbers in the architecture name represent the layer number and the hidden size (e.g., RNN-2-128 denotes an RNN model with 2 hidden layers and 128 hidden units in each layer). The number of parameters, the loss and perplexity on the test set are presented to reveal the actual performance.

Method	**Architecture**	**Parameter**	**Loss** $\downarrow$	**Perplexity** $\downarrow$
RNN	RNN-2-128	10.75M	5.24	189.36
	RNN-2-256	14.48M	5.16	174.33
	RNN-4-256	14.74M	5.17	175.71
	RNN-4-512	23.28M	5.07	159.55
LSTM	LSTM-2-128	11.00M	5.15	172.15
	LSTM-2-256	15.27M	5.01	150.44
	LSTM-4-256	16.32M	5.07	159.40
	LSTM-4-512	29.19M	5.02	151.46
GRU	GRU-2-128	10.92M	5.11	165.71
	GRU-2-256	15.00M	5.00	149.08
	GRU-4-256	15.79M	5.04	155.16
	GRU-4-512	27.22M	**4.96**	**142.11**
Transformer	Transformer-2-128	14.88M	5.17	176.35
	Transformer-2-256	15.01M	5.13	169.07
	Transformer-4-256	15.80M	5.08	160.13
	Transformer-4-512	16.32M	5.04	154.83
	Transformer-8-512	18.43M	5.01	149.17
	Transformer-8-1024	20.53M	4.98	145.90
GPT	GPT-2-128	14.88M	5.15	173.25
	GPT-2-256	15.27M	5.11	166.01
	GPT-4-256	16.06M	5.05	155.89
	GPT-4-512	17.64M	5.02	152.16
	GPT-8-512	20.80M	4.99	146.73
	GPT-8-1024	27.11M	4.97	144.40

for the model to learn. Hence, the perplexity decreases in general as the embedding size increases. The dropout rate, however, demonstrates a trade-off between the regularization degree and the model capacity. A moderate dropout rate around 0.2 tends to achieve the best performance for most models. A smaller dropout rate may lead to overfitting, while a larger dropout rate may hinder the learning process. The sequence length is also critical for the performance of the language models. A longer sequence length provides more context information for the model to learn, but also increases the computational complexity and memory consumption. For most models, the perplexity decreases as the sequence length increases, but the improvement becomes marginal when the sequence length exceeds a certain threshold.

3.4 Anomaly Explanation

In the previous experiments, we observe an abnormal phenomenon that the performance of the Transformer-based models fail to defeat the RNN-based models. Such results can be explained from multiple perspectives. For one thing, we notice that the texts in the given dataset are composed of short spoken sentences, containing wrong taggings and obscure words, which makes the Transformer-based models fail to capture the context information effectively. For another, the Transformer-based models only have small numbers of parameters, which cannot fully exploit the scalability of the architecture compared to the RNN-based models.

However, we should note that the anomaly is mainly caused by the unfair training strategy. For the RNN-based models, the hidden states computed from the last batch are feeded together with the current batch, so they can break the limitation of the sequence length and capture information from longer history. In contrast, the Transformer-based models are only feeded with the current batch, which limits the context information they can fetch. We call this unfair mechanism *inheritance*.

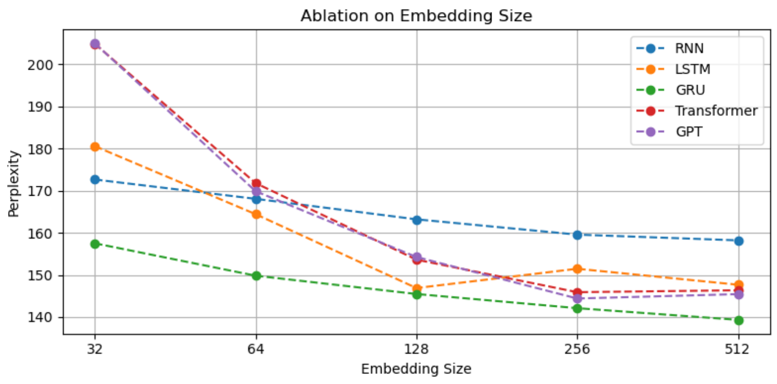

Figure 4: The performance of the language models with different embedding sizes. We present the perplexity on the test set for the RNN, LSTM, GRU, Transformer and GPT models with various embedding sizes in the set $\{32, 64, 128, 256, 512\}$ to reveal its impact on the performance.

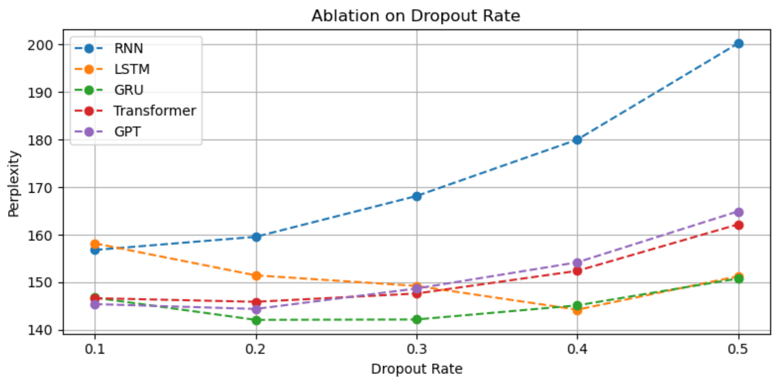

Figure 5: The performance of the language models with different dropout rates. We present the perplexity on the test set for the RNN, LSTM, GRU, Transformer and GPT models with various dropout rates in the set $\{0.1, 0.2, 0.3, 0.4, 0.5\}$ to reveal its impact on the performance.

To verify our hypothesis, we conduct an experiment to compare the performance of the language models with and without inheritance. We train the RNN, LSTM, GRU, Transformer and GPT models with the same architecture as the ablation study. The results are presented in Table 2.

Table 2: The performance of the language models with and without inheritance. We present the loss and perplexity on the test set for the RNN, LSTM, GRU, Transformer and GPT models with and without inheritance to reveal the impact of the inheritance mechanism.

Model	**w/ Inheritance**		**w/o Inheritance**	
	Loss $\downarrow$	**Perplexity** $\downarrow$	**Loss** $\downarrow$	**Perplexity** $\downarrow$
RNN	5.07	159.55	5.17	175.06
LSTM	5.02	151.46	5.14	170.55
GRU	**4.96**	**142.11**	5.13	168.50
Transformer	-	-	4.98	145.90
GPT	-	-	**4.97**	**144.40**

From the table we can see that the performance of the RNN-based models degrades when the inheritance mechanism is removed. With a fair training strategy, the Transformer-based models can actually outperform the RNN-based models, which supports our hypothesis. In addition, we try to scale up the GPT model with more hidden layers and hidden units to achieve a better performance.

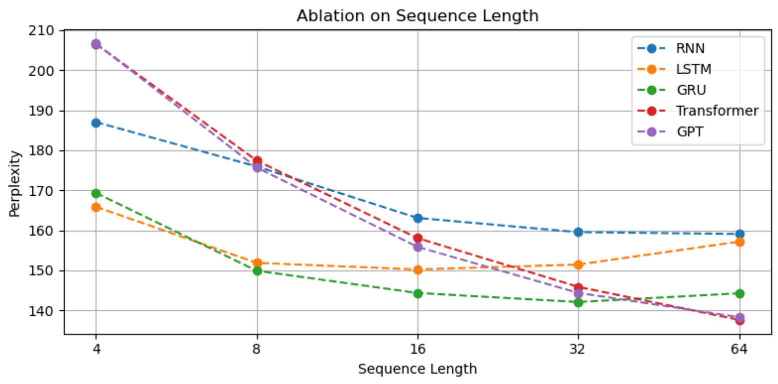

Figure 6: The performance of the language models with different sequence lengths. We present the perplexity on the test set for the RNN, LSTM, GRU, Transformer and GPT models with various sequence lengths in the set $\{4, 8, 16, 32, 64\}$ to reveal its impact on the performance.

The best performance is finally achieved by the GPT model with 384 embedding size, 2048 hidden size, 16 attention heads, 8 decoder layers and 128 sequence length, whose number of parameters is 59.54M and test perplexity is 130.57. Other hyperparameters follow the default setting.

4 Conclusion

In this project, we research on language modeling with deep neural networks, where we explore multiple architectures including RNNs, LSTM, GRU, transformers and GPTs. We introduce the principles of the models and our training strategies. Then we implement the models and train them on the given dataset. We compare the performance of different architectures and analyze the results. We find that the GRU model with specific hyperparameters achieves the best performance in terms of perplexity, and the advanced GPT model can achieve close performance.

Besides, we conduct ablation studies to investigate the effects of different hyperparameters on the performance. We mainly focus on the embedding size, dropout rate and sequence length. We find that both the embedding size and the sequence length have a positive correlation with the performance. A larger embedding size and a longer sequence length can help the model capture more context information and achieve lower test perplexity. However, the dropout rate controls the trade-off between the regularization degree and the model capacity. A moderate dropout rate can effectively prevent overfitting while improving the generalization ability of the model.

In addition, we observe and explain the anomaly that the performance of the Transformer-based models fail to defeat the RNN-based models. We suspect that such results are caused by the unfair training strategy. Then we conduct another experiment to train the RNN-based models with the same strategy as the Transformer-based models, which verifies our hypothesis.

References

[1] Jacob Andreas, Andreas Vlachos, and Stephen Clark. Semantic parsing as machine translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 47–52, 2013.

[2] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[3] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[4] David Guthrie, Ben Allison, Wei Liu, Louise Guthrie, and Yorick Wilks. A closer look at skip-gram modelling. In *LREC*, volume 6, pages 1222–1225, 2006.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[6] Phu Mon Htut, Kyunghyun Cho, and Samuel R Bowman. Grammar induction with neural language models: An unusual replication. *arXiv preprint arXiv:1808.10000*, 2018.

[7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] Roland Kuhn and Renato De Mori. A cache-based natural language model for speech recognition. *IEEE transactions on pattern analysis and machine intelligence*, 12(6):570–583, 1990.

[9] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[10] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[11] Constituency Parsing. Speech and language processing. *Power Point Slides*, 2009.

[12] Jay M Ponte and W Bruce Croft. A language modeling approach to information retrieval. In *ACM SIGIR Forum*, pages 202–208. ACM New York, NY, USA, 2017.

[13] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[14] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.