Week 3 : Generative Pre-trained Transformer

120220210 고재현

2023년 3월 29일

1 Introduction

The GPT class of language models has attracted global attention as they can produce text resembling that written by humans. This article focuses on the OpenAI GPT model, explaining it intuitively. It utilizes semi-supervised learning, pretraining on large unlabeled corpora and fine-tuning on small labeled ones. The decoder segment of the original Transformer is used as the base architecture and hyperparameters are covered. Tasks the GPT model was fine-tuned on are also discussed. Takeaways include the effect of 'locking' layers, zero-shot behavior, and the unsupervised language model's ability to recognize linguistic patterns. The performance of Transformer-based[1] architectures for semi-supervised learning is compared to that of LSTMs.

2 Model Architecture

The architecture of the GPT-1 model can be expressed mathematically as follows.

$$h_0 = U\mathbf{W}_e + \mathbf{W}p \tag{1}$$

$$h_l = \text{transformerblock}(h_{l-1}) \quad \forall l \in [1, n] \tag{2}$$

$$P(u) = \text{softmax}(h_n^T \mathbf{W}_e) \tag{3}$$

Given an input sequence of tokens $x_1, x_2, ..., x_T$, the GPT-1 model computes a sequence of hidden representations $h_1, h_2, ..., h_T$ using a stack of L transformer layers. Each layer consists of two sub-layers.

$h_0 = U\mathbf{W}_e + \mathbf{W}p$: This equation computes the initial hidden representation h_0 by taking the dot product of the learned word embedding matrix W with the one-hot encoded input sequence e, and then applying a linear transformation with learned weights U and Wp. This operation is equivalent to applying a linear layer to the input embeddings.

$h_l = \text{transformerblock}(h_{l-1}) \quad \forall l \in [1, n]$: This equation applies a series of

transformer blocks to the input sequence. Each transformer block consists of a multi-head self-attention mechanism and a position-wise feedforward neural network. The output of each block is passed as input to the next block.
$P(u) = \text{softmax}(h_n^T \mathbf{W}_e)$: This equation computes the final probability distribution over the next token in the sequence. It takes the dot product of the final hidden representation h_n with a learned output weight matrix W_e, and applies a softmax function to obtain a probability distribution over the vocabulary. The probability distribution is used to select the next token in the sequence during training and generation.

3 Example of foward pass of GPT-1

1. Initialize the input sequence as a list of token IDs, where each ID corresponds to a token in the vocabulary.

2. Embed the input sequence using the learned embedding matrix. The embedding matrix maps each token ID to a learned vector representation of the token.

3. Add positional encodings to the embedded sequence. The positional encodings are added to the embedded sequence to encode the relative position of each token in the sequence.

4. Pass the input sequence through a stack of transformer layers. Each layer consists of a multi-head self-attention mechanism and a feedforward neural network.

5. Compute the probability distribution over the vocabulary for the next token in the sequence. The final hidden state of the transformer model is used to compute the logits for each token in the vocabulary, and the logits are then passed through a softmax function to obtain a probability distribution.

```
input_sequence = [28, 312, 92, 745, 2]
embedding = embedding_matrix(input_sequence)  # shape: (5, 768)
```

```
positional_encoding =
↪  get_positional_encoding(input_sequence.shape[0],
↪  embedding.shape[1])
input_sequence_with_pos_enc = embedding + positional_encoding  #
↪  shape: (5, 768)
hidden_states = input_sequence_with_pos_enc
for layer in transformer_layers:
    hidden_states = layer(hidden_states)  # shape: (5, 768)
```

4 GPT Training

To understand how GPT works, it's important to understand the approach it utilizes, which is semi-supervised learning. This approach includes an unsupervised component, pretraining, which uses an unlabeled corpus of tokenized text to find a good initialization point for learning a specific task, and a supervised component, fine-tuning, which uses a labeled corpus of tokenized text tailored to a specific language task. This approach improves the performance of language models significantly, as it utilizes the abundance of unlabeled text to extract linguistic patterns, which serves as a better starting point for specialization using smaller labeled datasets. GPT falls under the semi-supervised learning approach and utilizes pretraining and fine-tuning.

The GPT model utilizes both the masked multi-head attention segment and the feed forward segment, along with the residuals and their corresponding addition and layer normalization steps. This involves position embedding of the learned embedding, followed by unidirectional self-attention through a masked multi-head attention segment. Residual is added and the result is layer normalized. The result then passes through a position-wise feedforward network and the residual is again added and layer normalized. The outcome either passes to the next decoder segment or is the output of the model as a whole.

Model	Context Window Size	Batch Size	Word Embedding Vector
GPT-1	512	64	768
GPT-2	Variable up to 2048	1 or 8	768
GPT-3	Variable up to 2048	Up to 8192	768, 1024, or 2048

〈표 1〉 Summary of the hyperparameter values in GPT-1, GPT-2, and GPT-3.

4.1 Important Hyperparameters

This table summarizes the differences in context window size, batch size, and word embedding vector across the three models. The values for each hyperparameter are listed for each model in their respective columns.

4.1.1 Context Window size

Context window size refers to the number of tokens that the model takes into consideration when predicting the next token in the sequence. A larger context window size allows the model to consider more contextual information when making its predictions, which can improve the accuracy and coherence of the generated text. In the case of GPT-1, the context window size is fixed at 512 tokens. GPT-2 and GPT-3, on the other hand, use a variable context window size that can be up to 2048 tokens. The larger context window size in GPT-2 and GPT-3 allows the models to capture longer-range dependencies and context in the input sequence, which can be beneficial for tasks such as summarization, question-answering, and dialogue generation. However, using larger context window sizes also requires more memory and computation, which can make the training and inference slower and more computationally intensive. Additionally, using too large a context window size can lead to overfitting on the training data, as the model may memorize long-range dependencies in the training set that do not generalize well to new inputs. In practice, choosing an appropriate context window size depends on the specific requirements of the task and the available hardware resources. A larger context window size may be preferred when generating long-form text, while a smaller context window size may be sufficient for tasks such as sentiment analysis or text classification. Overall, the appropriate context window size

depends on the complexity of the task and the available hardware resources, and should be carefully tuned to avoid overfitting and achieve optimal performance.

4.1.2 Word Embedding size

The main advantage of larger word embedding vectors is that they can capture more nuanced semantic relationships between words. By representing each word with a higher-dimensional vector, the model has more degrees of freedom to capture complex semantic relationships, such as synonyms, antonyms, and related concepts. This allows the model to better understand the underlying meaning and context of the words, resulting in more accurate language modeling and downstream language tasks. However, using larger word embedding vectors also comes with some drawbacks. Larger embeddings require more memory and computation, which can increase the training time and computational cost of the model. Additionally, using very large word embedding vectors may lead to overfitting on the training data, as the model may memorize the training set instead of learning generalizable patterns in the data. Overall, a larger word embedding vector allows the model to capture more nuanced semantic relationships between words, but also requires more computational resources and careful tuning to prevent overfitting.

4.1.3 Batch size

Batch size is the number of training samples that the model processes in a single forward/backward pass. Larger batch sizes can lead to faster training times, as more samples can be processed in parallel. However, increasing the batch size also requires more memory and computation, which can lead to longer training times and may require larger hardware resources. In the case of GPT-1, the batch size is set to 64 sequences. GPT-2 uses a smaller batch size of 1 or 8 sequences, while GPT-3 can use larger batch sizes of up to 8192 sequences. The larger batch sizes used in GPT-3 can significantly reduce training times, especially when using large models and datasets. However, they also require larger hardware resources, such as GPUs or TPUs, to

fit the models in memory and to achieve high parallelism during training. In practice, choosing an appropriate batch size depends on the available hardware resources and the complexity of the model and dataset. A smaller batch size may be preferred when training on smaller datasets or when using less powerful hardware, as it requires less memory and computation. Conversely, a larger batch size may be preferred when training on larger datasets or when using more powerful hardware, as it can reduce training times and improve parallelism. Overall, the appropriate batch size depends on the specific requirements of the task, the available hardware resources, and the complexity of the model and dataset.

4.2 Pretraining

During pretraining, there are no labels to guide the training process, making it unsupervised. However, we can utilize our large corpus of tokens $T_1, ..., T_n$ by applying a (sliding) context window of length k. This allows us to structure the text into windows, such as T_1, T_2, T_3, T_2, T_3, T_4, and so on, with $k = 3$. By feeding a context window to the GPT model, we can predict the next token.

4.3 Fine-Tuning

After pretraining, the GPT model can be fine-tuned using a labeled dataset C, as explained in paper [2]. Each instance in the dataset contains a sequence of tokens $x_1, x_2, ..., x_m$ and a corresponding label y. The sequence is passed through the pretrained Transformer architecture and then through a linear layer with weights W_y and Softmax activation for multiclass prediction.

4.3.1 Auxilary objective in fine-tuning

Mathematically, the auxiliary objective can be expressed as:

$$\mathcal{L}_{aux} = \lambda ||C||_1 \tag{4}$$

where lambda is the regularization parameter and $||C||_1$ is the L1 norm of the weight vector C.

The auxiliary objective, $\lambda * \mathrm{L1}(C)$, is used to improve the model's generalization ability. This objective function performs L1 regularization on the weight vector C and uses it as an additional loss function multiplied by lambda. In this way, the model can explore a wider range of parameter values and improve generalization performance while preventing overfitting. Therefore, it is reasonable to use pretrained neural network parameter values as the initial values for fine-tuning neural network parameters. This allows the model to leverage the information from the previously learned language model to start optimizing for the new task-specific dataset.

참고 문헌

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[2] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.