GPT Pre-training Summary

1_GPT.pdf
PDF

Extract and explain contents

The document **"Improving Language Understanding by Generative Pre-Training"** by Alec Radford and others from OpenAI describes the foundational work behind Generative Pre-trained Transformer (GPT). Here's a summary of its contents:

1. Introduction

- NLP models often require large labeled datasets, which are expensive to create.
- The paper proposes a **two-step approach**:
  1. **Unsupervised Pre-training**: Train a language model on a large corpus of unlabeled text.
  2. **Supervised Fine-tuning**: Adapt the model to specific tasks using labeled data.
- A key contribution is the use of **task-aware input transformations**, making fine-tuning more effective.

2. Related Work

- **Semi-Supervised Learning**: Previous approaches mainly relied on word embeddings; this work captures more contextual information.
- **Unsupervised Pre-training**: Compared to prior LSTM-based approaches, transformers allow better generalization across tasks.
- **Auxiliary Training Objectives**: Some models add extra tasks like Part-of-Speech tagging, but this paper shows that pre-training alone captures enough useful features.

3. Framework

3.1 Unsupervised Pre-training

- Uses a **Transformer decoder** with a standard language modeling objective:

$$L1(U) = \sum_i \log P(u_i | u_{i-k}, ..., u_{i-1})$$

  where k is the context window.

- The model is trained on the **BooksCorpus dataset**, which contains long-form text.

3.2 Supervised Fine-tuning

- Uses labeled datasets for tasks like **natural language inference, question answering, and sentiment analysis**.
- The final representation from the transformer is passed through a linear+softmax layer to predict output labels.

3.3 Task-Specific Input Transformations

- Converts structured tasks into sequences that the pre-trained model can process:
  - **Textual entailment**: Concatenates premise and hypothesis with a separator.
  - **Semantic similarity**: Processes two sentence orderings separately.
  - **Question answering**: Concatenates document, question, and each possible answer.

4. Experiments

4.1 Setup

- The model is trained on **BooksCorpus** (7,000 books).
- Uses a **12-layer Transformer decoder**, **768-dimensional embeddings**, **12 attention heads**, and a **40,000-token vocabulary**.

4.2 Supervised Fine-tuning

- The model is tested on multiple tasks:
  - **Natural Language Inference (NLI)**: Outperforms previous methods on MNLI, SciTail, QNLI.
  - **Question Answering (RACE, Story Cloze Test)**: Achieves an 8.9% improvement in commonsense reasoning.
  - **Semantic Similarity**: Improves on the STS benchmark.
  - **Text Classification (SST-2, CoLA)**: Shows better linguistic understanding.

5. Analysis

- **Zero-shot Learning**: The pre-trained model performs well even without fine-tuning.
- **Ablation Studies**: Removing pre-training or using LSTMs reduces performance significantly.

6. Conclusion

- A **task-agnostic** Transformer can generalize across multiple NLP tasks.
- Pre-training on **long, contiguous text** leads to better understanding.
- The approach surpasses state-of-the-art methods on 9 out of 12 NLP benchmarks.

This paper was one of the key steps leading to models like **GPT-2, GPT-3, and beyond**, demonstrating the power of large-scale pre-training in NLP. 🚀

Explain it's architecture

GPT Architecture Explained 🚀

The **Generative Pre-trained Transformer (GPT)** is based on the **Transformer** architecture, specifically using a **decoder-only** model. Below is a detailed breakdown of its components and how it works.

---

1. Overall Structure

GPT follows a **two-stage** training approach:

1. **Unsupervised Pre-training**
   - Trains on a large corpus of text using a **language modeling objective** (predict the next word in a sentence).
2. **Supervised Fine-tuning**
   - Adapts the pre-trained model for specific NLP tasks like classification, translation, or question answering.

GPT is a **unidirectional (left-to-right) Transformer decoder**, meaning it **only attends to previous tokens** when generating text.

---

2. Transformer Decoder Block

GPT is built using a **stack of Transformer decoder layers**. Each layer consists of:

- **Masked Multi-Head Self-Attention**
- **Layer Normalization**
- **Feedforward Neural Network**
- **Residual Connections**

🔹 Transformer Decoder Block Structure

Each decoder block follows this flow:

1. **Token + Positional Embeddings**
   - Converts input words into vector representations and adds positional information.
2. **Masked Multi-Head Self-Attention (MHA)**
   - The model can only attend to past tokens (unlike bidirectional transformers like BERT).
   - Uses multiple attention heads to capture different aspects of the input.
3. **Feedforward Neural Network (FFN)**
   - Applies two linear transformations with a **non-linearity (ReLU or GELU)** in between.
4. **Layer Normalization & Residual Connections**
   - Helps in stabilizing training and improves gradient flow.

Mathematical Formulation:

- The self-attention mechanism computes a weighted sum of input token embeddings:

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

  where:

  - Q (Query), K (Key), and V (Value) are derived from input embeddings.
  - d_k is the dimensionality of key vectors.
  - The **masking mechanism** ensures that each token only attends to previous tokens.

---

3. Token and Positional Embeddings

Since transformers do not have an inherent understanding of token order, GPT includes:

- **Token embeddings**: Convert words/subwords into numerical vectors.
- **Positional embeddings**: Learn a representation of token positions in a sentence.

These are summed together to form the final input representation:

$$h_0 = UW_e + W_p$$

where:

- U = input tokens
- W_e = token embedding matrix
- W_p = positional embedding matrix

---

4. Masked Multi-Head Attention (MHA)

Unlike standard self-attention, **GPT uses causal (masked) attention**, meaning:

- Each word can **only attend to previous words**, preventing the model from "cheating" and looking ahead.

This allows GPT to **generate text step by step**, predicting one word at a time.

---

5. Final Output Layer

The output of the last decoder layer goes through:

1. **Linear Projection** to map hidden states to vocabulary size.
2. **Softmax Layer** to produce probabilities for each token.

Prediction at time step t:

$$P(y_t|y_1, ..., y_{t-1}) = \text{softmax}(W_o h_t)$$

where W_o is the output weight matrix.

6. Training Objective

GPT is trained using **language modeling (causal modeling)**:

$$L(U) = \sum_i \log P(u_i | u_{i-k}, ..., u_{i-1})$$

where k is the context window size.

The model learns to predict the next token **based on the previous tokens**.

---

7. Differences Between GPT and BERT

Feature	GPT (Decoder)	BERT (Encoder)
Architecture	Transformer Decoder	Transformer Encoder
Training Objective	Predict next word (causal)	Masked language model (fill in missing words)
Attention Mask	Masked (causal)	Full (bidirectional)
Use Case	Text Generation, Summarization, Dialogue	Classification, Q&A, Sentence Embeddings

---

8. Hyperparameters (GPT-1)

- **12 Transformer layers**
- **12 Attention heads**
- **768 Hidden size**
- **3072 Feedforward size**
- **40,000 Vocabulary size**
- **Byte-Pair Encoding (BPE) for tokenization**

Conclusion

- **GPT is a unidirectional Transformer-based language model.**
- **Uses causal self-attention for text generation.**
- **Pre-training on large datasets improves downstream NLP tasks.**
- **Foundation for future models like GPT-2, GPT-3, and GPT-4.**

This architecture laid the groundwork for the **modern era of generative AI**, influencing how **chatbots, code generation, and creative AI** work today! 🚀

how infrence/prediction will be

Inference/Prediction in GPT 🚀

During inference, GPT **generates text** by predicting one token at a time, using **causal (auto-regressive) decoding**. The process follows these steps:

---

1. Overview of GPT Inference

- Given an **input prompt**, GPT **predicts the next token**.
- The predicted token is **appended** to the input.
- The process **repeats iteratively** until a stopping condition is met (e.g., end token, max length).

---

2. Step-by-Step Inference Process

Let's say we start with the prompt:
👉 **"Artificial intelligence is"**

Step 1: Tokenization

- The input sentence is converted into **tokens** using **Byte-Pair Encoding (BPE)**.

```css
["Artificial", " intelligence", " is"]
```

- These tokens are mapped to **embedding vectors**.

Step 2: Positional Encoding

- Positional embeddings are added to the input embeddings.

Step 3: Transformer Processing

- The input is passed through **multiple Transformer decoder layers**:
  - **Masked Self-Attention**: Ensures each token can only attend to previous tokens.
  - **Feedforward Network**: Processes token relationships.
  - **Residual Connections & Layer Normalization**: Improves stability.

Step 4: Output Token Prediction

- The model computes **probabilities** over the vocabulary using **Softmax**:

$$P(y_t|y_1, ..., y_{t-1}) = \text{softmax}(W_o h_t)$$

- Example probability distribution:

```arduino
{"learning": 0.25, "theory": 0.18, "a": 0.10, "and": 0.07, ...}
```

- The highest probability token (**"learning"**) is selected.

Step 5: Append the Predicted Token

- The new token **"learning"** is added to the sequence:

```arduino
"Artificial intelligence is learning"
```

Step 6: Repeat Until Stopping Condition

- The process **repeats**, generating one token at a time:

```arduino
"Artificial intelligence is learning rapidly in"
"Artificial intelligence is learning rapidly in various"
"Artificial intelligence is learning rapidly in various fields."
```

- The generation stops when:
  - A **stop token** (like `<EOS>`) is generated.
  - The **maximum length** is reached.

---

3. Sampling Strategies in Inference

To **avoid repetitive or deterministic outputs**, different sampling methods are used:

Method	Description	Example Output
Greedy Decoding	Selects the token with the highest probability.	*Predictable but repetitive*
Beam Search	Expands multiple possible sequences and selects the most likely one.	*More coherent but may lack diversity*
Top-k Sampling	Samples from the top-k most probable tokens.	*Adds randomness*
Temperature Scaling	Adjusts token probabilities (higher temp = more randomness).	*Diverse outputs*
Nucleus Sampling (Top-p)	Chooses from a **cumulative probability** threshold p.	*Balances coherence & randomness*

---

4. Example: Generating Text with GPT

Let's say GPT is asked:

📝 **Input Prompt:** `"The future of AI is"`

🔄 **Step-by-Step Predictions:**

1. `"The future of AI is"` → **"promising"** (highest probability)
2. `"The future of AI is promising"` → **"due"**

3. `"The future of AI is promising due"` → **"to"**
4. `"The future of AI is promising due to rapid"`, etc.

💡 **Final Output:** `"The future of AI is promising due to rapid advancements in deep learning and neural networks."`

---

5. Summary

✅ **Auto-regressive Generation**: Predicts one token at a time.
✅ **Masked Self-Attention**: Ensures past context influences predictions.
✅ **Sampling Techniques**: Adjust randomness for better quality.
✅ **Efficient Decoding**: Uses optimized algorithms for fast inference.

GPT's **powerful inference** makes it ideal for chatbots, text completion, and content generation! 🚀

how infrence/prediction in case of BERT

Inference/Prediction in BERT 🧠

Unlike GPT, which is an **autoregressive** model that generates text token by token, **BERT (Bidirectional Encoder Representations from Transformers)** is an **autoencoding** model primarily used for **classification, question answering, and sentence embeddings** rather than text generation.

---

1. Overview of BERT Inference

- BERT is **bidirectional** (sees both left and right context).
- It is **not a generative model** but excels at tasks like:
  - **Text classification** (Sentiment analysis, spam detection)
  - **Named Entity Recognition (NER)**
  - **Question Answering** (SQuAD)

- **Sentence Pair Tasks** (Entailment, similarity)
- **Masked Language Model (MLM) filling missing words)

---

2. Step-by-Step BERT Inference Process

Let's assume we use BERT for **sentence classification** (e.g., sentiment analysis).

Step 1: Tokenization

- Input text is tokenized using **WordPiece Tokenization**:

```arduino
"BERT is amazing!" → ["[CLS]", "bert", "is", "amazing", "!", "[SEP]"]
```

  - **[CLS]** → Special token for classification tasks.
  - **[SEP]** → Used to separate two sentences in pairwise tasks.

Step 2: Embedding Layer

- The input tokens are converted into **three embeddings**:
  1. **Token embeddings** (word representations)
  2. **Segment embeddings** (sentence A/B distinction)
  3. **Positional embeddings** (word order information)

Step 3: Transformer Encoder Layers

- BERT processes input through **multiple self-attention layers**.
- Unlike GPT, **self-attention is bidirectional**, meaning it can see both **left and right context**.

  Example:

  - In GPT: `"The cat sat on the"` → Predict `"mat"` (only sees left context).
  - In BERT: `"The [MASK] sat on the mat"` → Predict `"cat"` (sees both sides).

Step 4: Output Representation

- The **[CLS] token's output vector** is used for classification tasks.
- For **question answering**, BERT outputs **start and end token positions** for the answer span.
- For **MLM**, BERT predicts the missing word.

---

3. BERT's Inference for Different Tasks

1️⃣ Text Classification (Sentiment Analysis)

- Use the **[CLS] token's** representation.
- Apply a **fully connected layer + softmax** for classification.

  Example Task: Classify **"BERT is amazing!"**

```scss
[CLS] BERT is amazing ! [SEP]  →  Positive (0.98 probability)
```

2️⃣ Named Entity Recognition (NER)

- Each token's embedding is classified into entity categories.

  Example Task: Extract entities from **"Elon Musk founded Tesla."**

```css
[CLS] Elon Musk founded Tesla . [SEP]
 B-PER   I-PER      O     B-ORG  O
```

3️⃣ Question Answering (SQuAD)

- BERT identifies **start and end positions** of the answer in the passage.

  Example Task:
  Context: "Albert Einstein was born in Germany."
  Question: "Where was Einstein born?"
  Answer: `"Germany"`

```pgsql
```

```
Start Token: "Germany" (index 6)
End Token: "Germany" (index 6)
```

4️⃣ Masked Language Modeling (MLM)

- BERT predicts **masked words** in a sentence.

  Example: `"BERT is [MASK]!"`
  Prediction: `"BERT is powerful!"`

---

4. Key Differences Between BERT & GPT Inference

Feature	BERT (Encoder)	GPT (Decoder)
Architecture	Transformer Encoder	Transformer Decoder
Training Task	MLM (fill in the blanks)	Autoregressive (predict next word)
Directionality	**Bidirectional** (sees full sentence)	**Unidirectional** (left-to-right)
Primary Use Case	Classification, Q&A, NER	Text generation, completion
Output	Contextual embeddings, predictions	Generated text
Example Task	Sentiment: "Good" or "Bad"	Autocomplete: "The cat sat on the..."

---

5. Summary

✅ **BERT is bidirectional** → Uses **entire context** for predictions.
✅ **Predictions vary by task** → Classification, question answering, sentence similarity, etc.
✅ **Not used for text generation** → Instead, it provides **contextual word representations**.
✅ **Inference is faster than GPT** because all words are processed in parallel.

🔍 **In short:**

- GPT **generates** text.
- BERT **understands** text.

Hope that clarifies it! 🚀