GPT Transformer

The GPT transformer research paper, particularly OpenAI's original "Improving Language Understanding by Generative Pre-Training" (2018), laid the foundation for the GPT family of models.

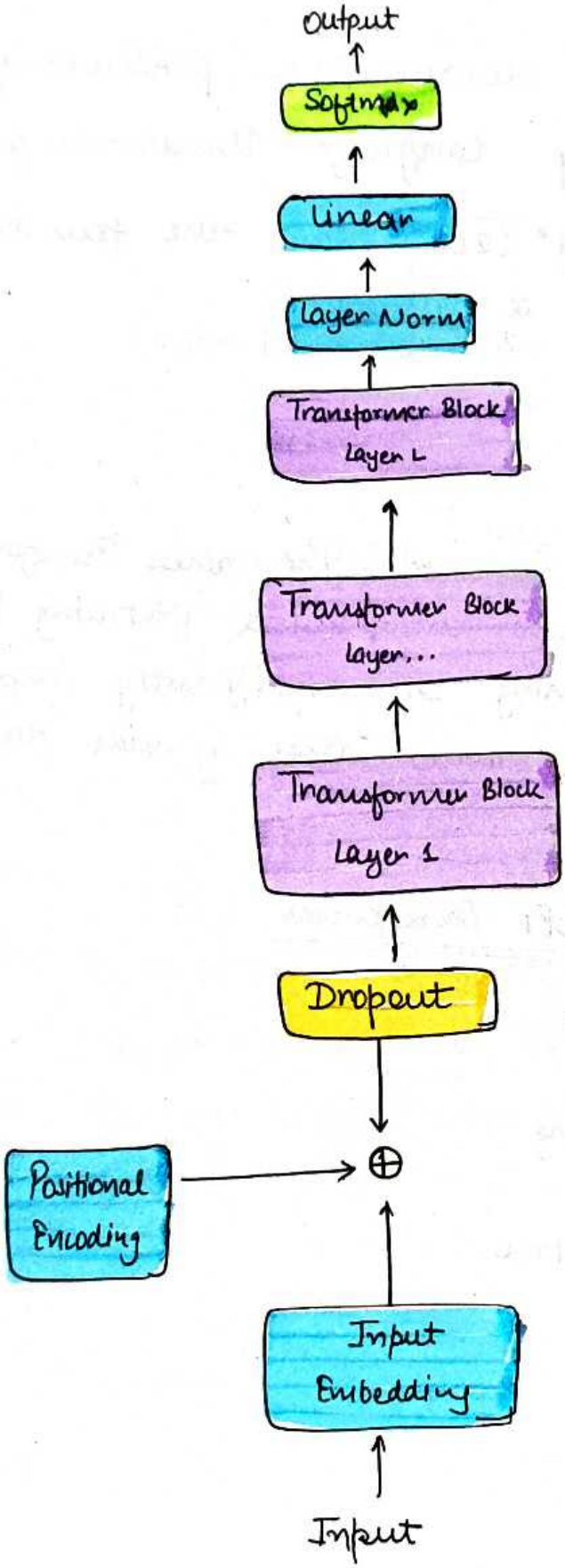
Core Idea of GPT

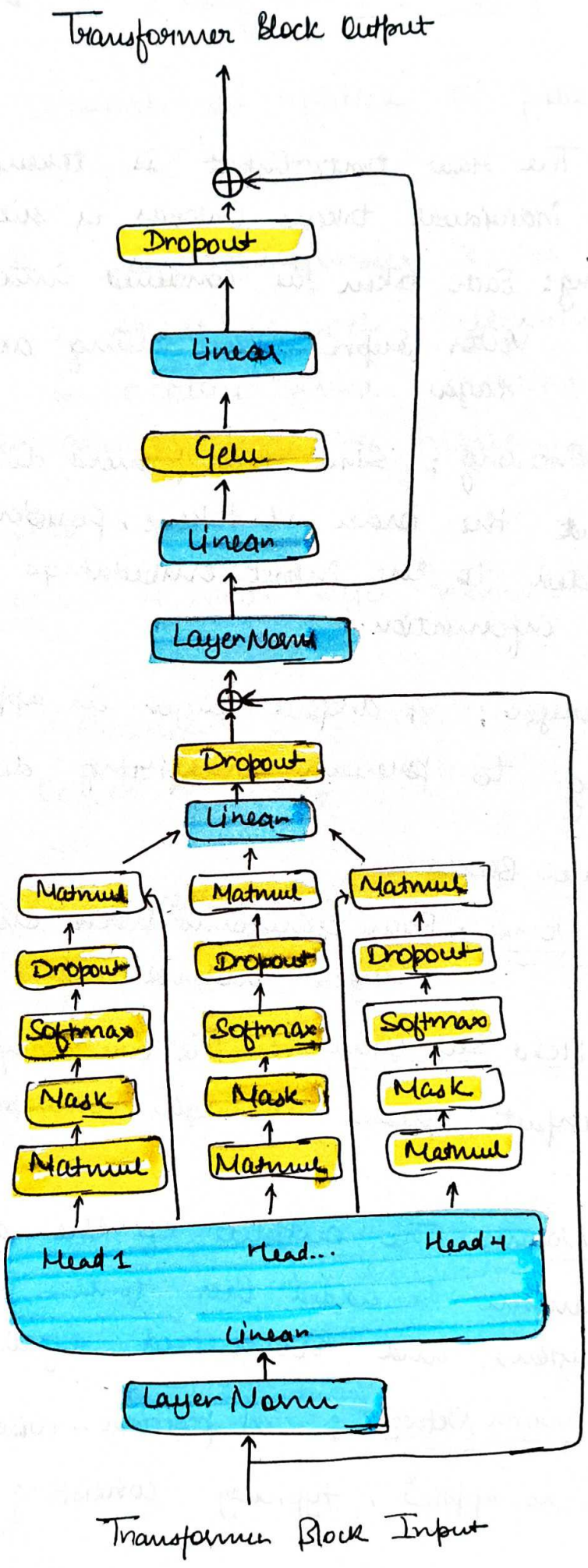
The paper introduces a Generative Pre-trained Transformer (GPT) models, showcasing how unsupervised pretraing followed by supervised fine-tuning can significantly improve language model performance across a wide range of NLP tasks.

Key Contribution in GPT Transformers

1. Unsupervised Pretraining
2. Supervised fine-tuning
3. Transformer Architecture
4. Scalability

GPT Architecture (wikipedia)




Transformer Block Output
Dropout
Linear
Gelu
Linear
LayerNorm
Dropout
Linear
Matmul
Matmul
Matmul
Dropout
Dropout
Dropout
Softmax
Softmax
Softmax
Mask
Mask
Mask
Matmul
Matmul
Matmul
Head 1
Head...
Head 4
Linear
LayerNorm
Transformer Block Input

Quick review of Architecture (geeks for Geeks)

1. **Input Embedding**
   - Input: The raw text input is tokenized into individual tokens (words or subwords).
   - Embedding: Each token is converted into a dense vector representation using an embedding layer.
2. **Positional Encoding**: Since transformers do not inherently understand the order of tokens, positional encoding are added to the input embeddings to retain the sequence information.
3. **Dropout Layer**: A dropout layer is applied to the embedding to prevent overfitting during training.
4. **Transformer Blocks**
   - Layer Norm: Each transform block starts with a layer normalization.
   - Multi-Head Self-Attention: The core component, where the input passes through multiple attention heads.
   - Add & Norm: The output of the attention mechanism is added back to the input (residual connection) and normalized again.
   - Feed-Forword Network: A position-wise feed-forward network is applied, typically consisting of two

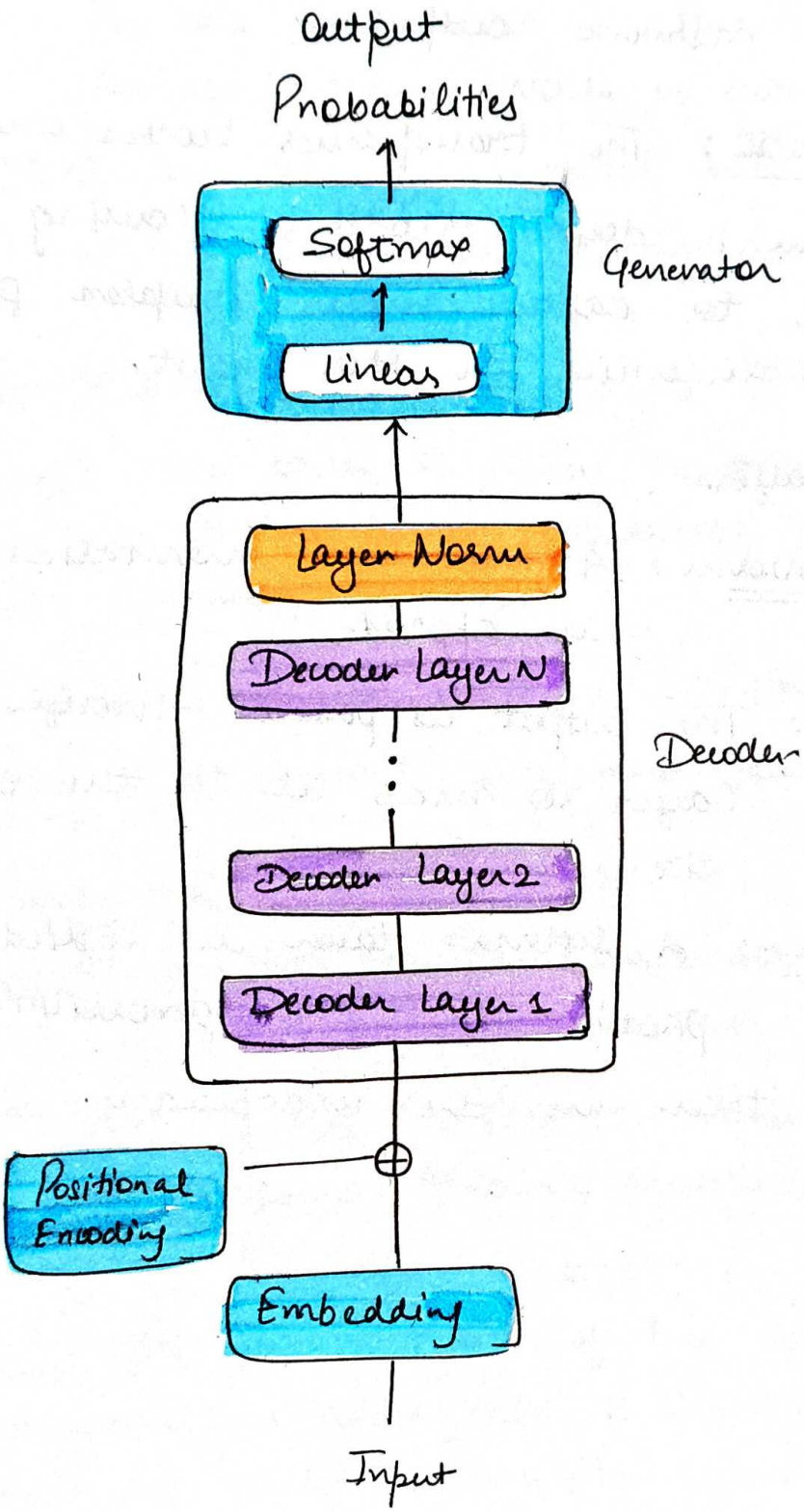
linear transformation with a GELU activation in between.

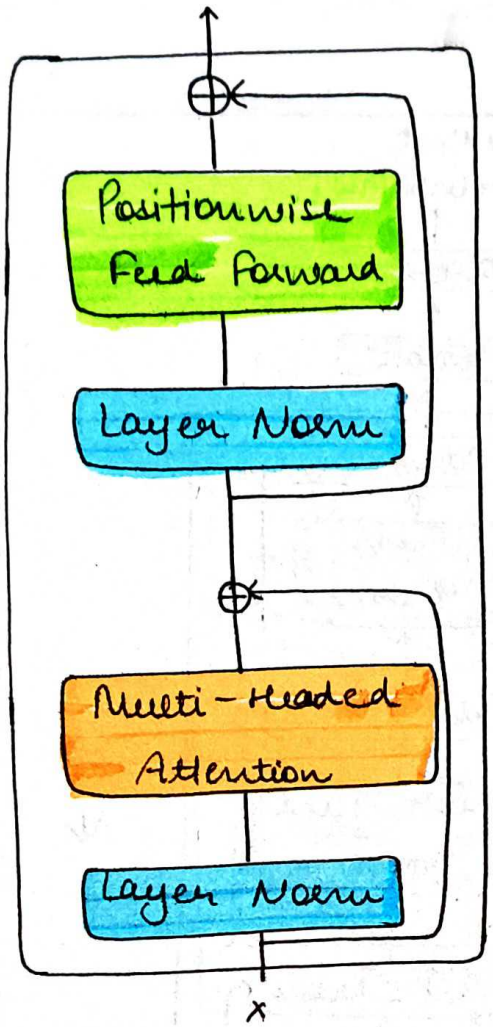
- Dropout: Dropout is applied to the feed-forward network output.

5. Layer Stack: The transformer blocks are stacked to form a deeper model, allowing the network to capture more complex pattern and dependencies in the input.

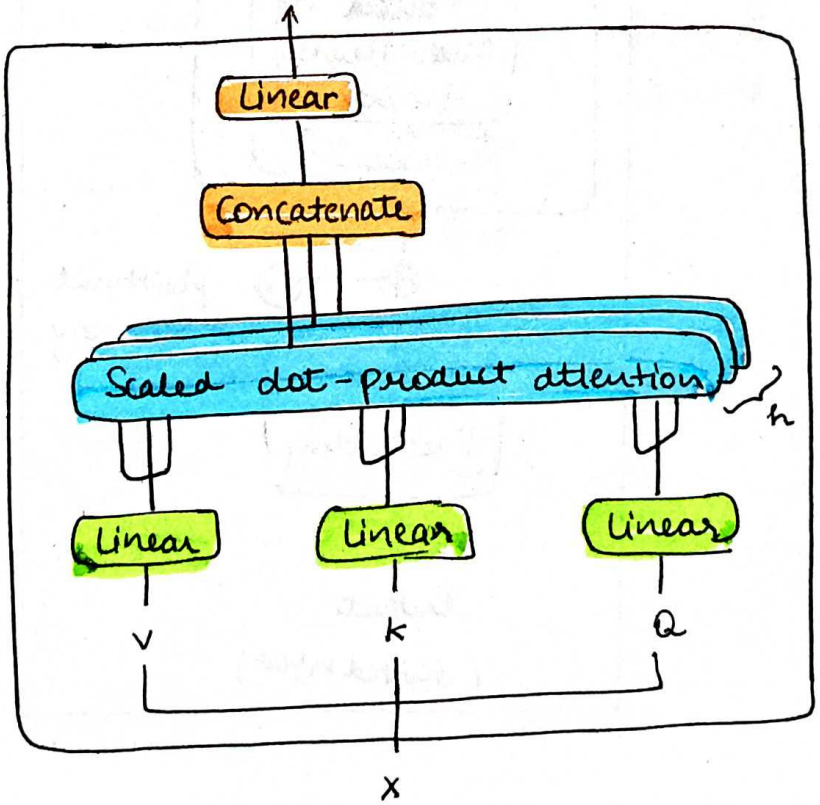
6. Final Layers
   - Layer Norm: A final layer normalization is applied.
   - Linear: The output is passed through a linear layer to map it to the vocabulary size.
   - Softmax: A softmax layer is applied to produce that final probabilities for each token in the vocabulary.

Overview of Transformer Architecture



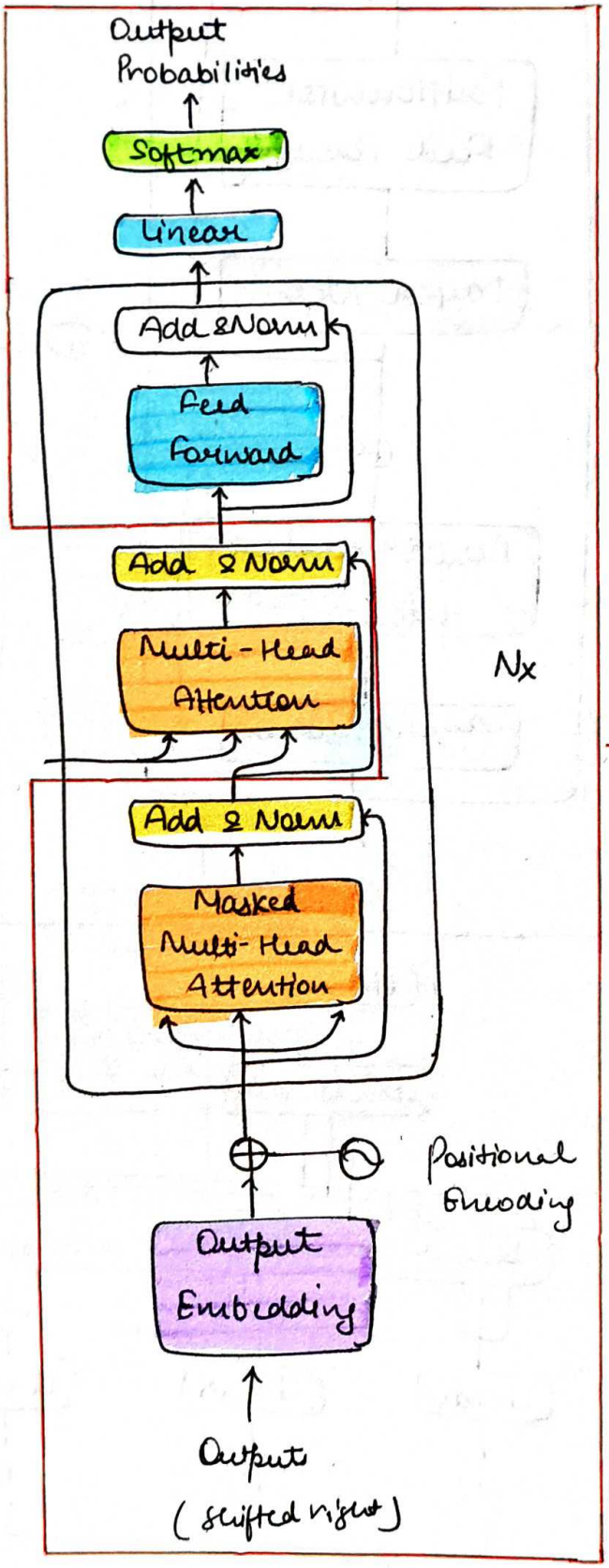


Decoder Layer

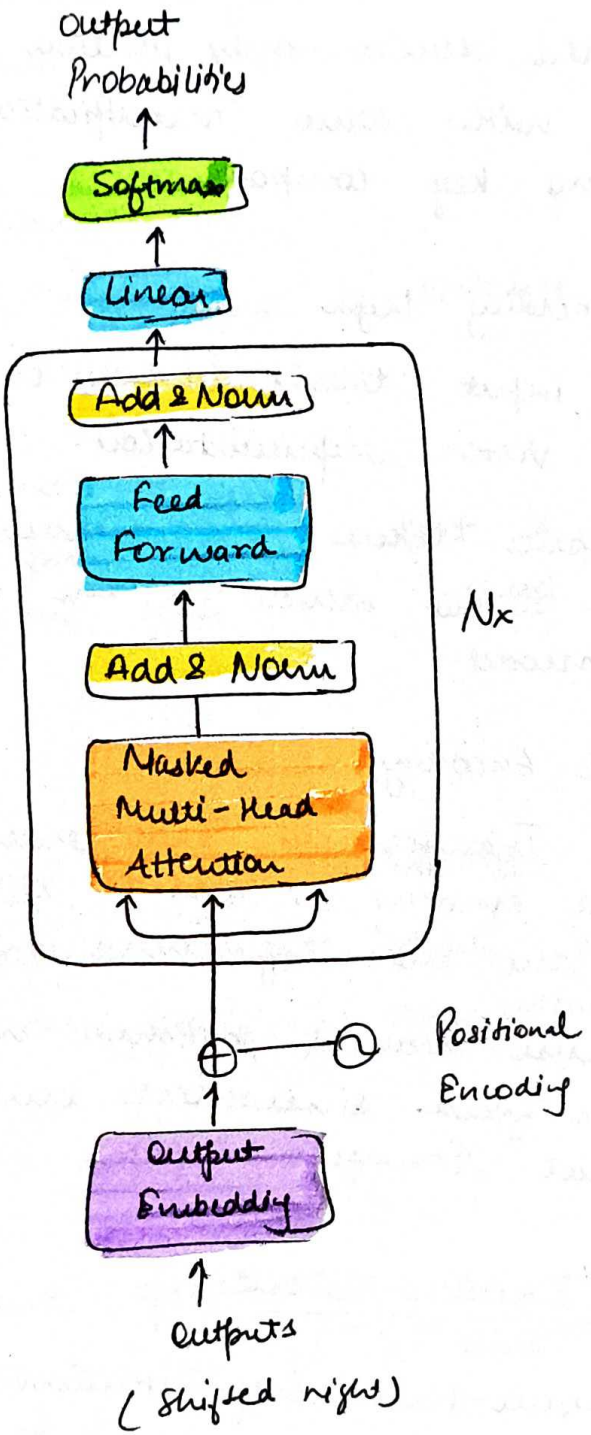


Multi-Head Attention

GPT-Style Transformer



Final GPT Architecture



GPT Architecture and Working

GPT adopts the decoder-only portion of the Transformer architecture with some modifications. It consists of the following key components:

1. Input Embedding layer
   - Converts input tokens (words or subwords) into dense vector representation.
   - The input tokens are represented as embedding of size d_{model} which is the dimensionality of the model.

2. Positional Encoding
   - Since Transformers lack recurrence or convolution, positional encoding is added to token embeddings to encode the order of tokens in a sequence.
   - GPT use learned positional embedding as opposed to the fixed sinusoidal embeddings in the original Transformer paper.

Transformer Decoder Block

1. Masked Multi-Head Self-Attention:
   - Computes the relationships (attention scores) between tokens in the sequence.
   - Masking ensure that the model only attends to previous tokens, preserving the casual nature of language generation.

2. Feed forward Neural Network (FFN):

- A fully connected feedforward network with two layers and an activation function (Relu or Gelu).

3. Layer Normalization:

- Normalization the input to stabilize training and improve convergence.

4. Residual Connections:

- Add the input to the output of each sub-layer to ease gradient flow and prevent vanishing gradients.

Output Layer

- Outputs the logits for the vocabulary.
- A softmax function convert these logits into probabilities for each token in the vocabulary.

Working of GPT

GPT is trained and operates in an autoregressive manner, predicting one token at a time, conditioned on the tokens seen so far.

Training Process

1. Tokenization

- Input text is tokenization into smaller units (words or subwords) using a tokenizer like Byte

Pair Encoding (BPE).

2. Embedding:
   - Tokens are mapped to dense vectors using the input embedding layer.

3. Positional Encoding:
   - Positional embeddings are added to the token embeddings to provide positional information.

4. Masked Multi-Head Self-Attention:
   - Attention is applied with masking to ensure only tokens up to the current position influence predictions.
   - This casual masking enables autoregressive generation.

5 Feedforward Network
   - The outputs from the attention mechanism are passed through the feedforward network to extract non-linear features.

6. Output Layer
   - A linear transformation followed by a softmax layer produces probabilities over the vocabulary.

7. Loss function
   - The model minimizes the cross-entropy loss, which measures how well the predicted probabilities match the target tokens.

Inference Process (Text Generation)

1. Initialization:
   - Start with a prompt or seed text, such as "The weather today is".

2. Token-by-Token Generation:
   - Predict the next token by sampling from the output probability distribution.
   - Append the predicted token to the input sequence and repeat until a stopping condition (e.g., end-of-sequence token or length limit) is met.

Key Features of GPT

1. Autoregressive Modelling
   - GPT predicts the next token $P(t_i / t_1, t_2, \ldots t_{i-1})$, ensuring the output depends only on preceding tokens.

2. Multi-Head Self-Attention
   - Captures relationships between all tokens in a sequence, enabling content-aware token

3. Scalability
   - GPT scales effectively by increasing the number of layers, hidden dimensions and attention heads, as demonstrated by models like GPT-2 and GPT-3.

4. Transfer Learning
   - Pretrained on massive unlabeled corpora using unsupervised learning, then fine-tuned for specific tasks with with supervised learning.

Difference between GPT and Transformer

1. Casual Masking:
   - GPT uses casual masking in attention layers to ensure tokens cannot attend to future tokens.

2. Decoder-Only Architecture.
   - GPT retains only the decoder stack of the Transformer, discarding the encoder.

Advantages of GPT

1. Efficient Training
   - Uses self-supervised learning on vast amounts of text.

2. Generative Capabilities
   - Excels at generating coherent, contextually relevant text.

3. Versatality
   - Can be fine-tuned for a variety of NLP tasks.

Challenges

1. Resource Intensive:
   - Requires significant computational resources for pretraining.

2. Bias:
   - May propagate biases present in the training data.

3. Lack of Explicit Task Supervision
   - Relies heavily on fine-tuning to adapt to specific tasks.

Example: Predicting the Next Word

Input Text: "The weather today is very"

Step-by-Step Walkthrough

1. Tokenization

The input text is tokenized into subword tokens using Byte Pair Encoding (BPE) or a similar tokenizer.

For example:

- Tokens: [The, weather, today, is, very]
- Token IDs: [1203, 4507, 620, 15, 879]

2. Positional Encoding

Positional embeddings are added to token embeddings to encodings to encode the order of tokens.

For instance:

- Token + Positional embeddings: $E_{total} = E_{token} + E_{position}$

3. Masked Multi-Head Self-Attention

The Masked Self-Attention computes relationships between tokens while ensuring each token can only attend to itself and the previous tokens.

For example:

- Attention considers:

$$P(t_1),\ P(t_2|t_1),\ P(t_3|t_1, t_2) \ldots\ P(t_5|t_1, t_2, t_3, t_4)$$

- Masks ensure the model does not "look ahead".

Key output:

The model outputs content-aware embeddings for each token, capturing relationships between tokens.

5. Feedforward Network

The outputs of the attention layer are passed through a fully connected feedforward (FFN) to extract non-linear features.

Example:

- Input: Content embeddings from the attention layer.
- Output: Enhanced features for each token.

6. Output Layer

The final layer produces logits for all tokens in the vocabulary.

For example

- Vocabulary size $V = 50,000$
- Logits: $[0.1, -0.2, \ldots, 0.9, 0.3]$

A softmax function converts these logits into probabilities.
For example: $P_{vocabulary} = \text{softmax}(\text{logits})$

Predicted Probabilities

Token	Probability
sunny	0.45
rainy	0.30
cloudy	0.20
snowy	0.05

7. Sampling

From the predicted probabilities, a token is selected.

For example:

- Next Token: "sunny" (highest probability).

8. Update Sequence

The model appends the predicted token to the input sequence:

New Sequence: "The weather today is very sunny"

Autoregressive Loop

The model repeats the process, predicting one token at a time, until a stopping condition is met (e.g. reaching a maximum length or generating an end-of-sequence token).

Visual Flow of Example

1. Input: "The weather today is very"
2. Tokenize → Embed → Add Positional Encoding.
3. Pass through Transformer Decoder Blocks.
   - Masked Attention: Captures content while respecting sequence order.
   - Feedforward: Enhances forward features.
4. Output logits → Apply Softmax → Generate next token ("sunny")
5. Update input and repeat for further predictions.

GPT train on unsupervised data without label

GPT trains on unsupervised data without requiring explicit labels by using a self-supervised learning approach, specially a casual language modeling (CLM) objective. Here's how this works in detail:

Self-Supervised learning in GPT

Self-Supervised learning generates supervision signals directly from the input data. For GPT, the task is to predict the next token in a sequence based on the preceding tokens. This is known as next-token prediction or casual language modeling.

How it works:

- The model takes a sequence of tokens (e.g words or subwords) as input.
- It predicts the probability of the next token in the sequence.
- The input data serves as both the "features" and the "labels":

→ Feature (input): The tokens up to the current ~~location~~ position.

→ Labels (output): The token immediately following the input sequence.

Example : Suppose we have a sentence:

"I love machine learning".

- Input : "I love machine"
- Label : "learning"

This process is repeated across the entire corpus, where every token becomes a target for prediction.

Data Preparation

1. Tokenization :
   - The text data is divided into tokens (e.g, words or subwords) using a tokenizer like Byte Pair Encoding (BPE). Each token is assigned a unique num. ID.
2. Padding and Truncation:
   - Sequences are padded or truncated to a fixed length (e.g. 512 tokens) for batch processing.
3. Sliding Window:
   - To ensure all tokens in the dataset are used, a sliding window technique is applied:
     → First window: "I love machine" → Predict "learning".
     → Second window: "love machine learning" → Predict "."

Masked Self-Attention Mechanism

GPT uses masked self-attention during training to ensure the model can only "see" tokens that occur before the current position. This maintains

the casual nature of the tasks.

- For each token, the self-attention mechanism masks out future tokens to prevent them from influencing the prediction.
- This ensures the model trains in an autoregressive manner, mimicking how it will generate text during inference.

Why No Labels Are Needed

1. Unsupervised Nature:
   - The dataset (eg: Books Corpus, web text) is just raw text, with no explicit labels provided.
   - The task itself (predicting the next token) inherently provides the labels.

2. Self-Generated Targets:
   - For every token in the dataset, the preceding tokens form the input, and the token itself is the target.
   - The model learns from the structure and patterns in the data without needing external annotations.

Advantages of Unsupervised Training

1. Scalability:
   - Large-scale, unlabeled datasets (e.g., Book Corpus, Common Crawl) are abundant and easy to collect.

2. Pretraining Generalizes Across Tasks:

- The model learns general linguistic knowledge (grammer, syntax, semantics) from diverse text data.

3. Minimized Human Effort:
   - No manual labeling or task-specific dataset preparation is required.

Optimization Process

1. Loss Calculation: • The model adjusts its weights to min. the loss.
   - The cross-entropy loss measures how far the predicted token probabilities are from the true token.

2. • Backpropagation:
   - Gradients are computed for all trainable parameters.
   - The weights are updated using an optimizer like Adam with learning rate schedule.

3. Epochs and Iterations:
   - The model proces the datasets multiple time (epochs) until convergence.
   - Each iterations processes a batch of tokenized sequences.

Key Takeways

- GPT doesn't need labels because the next-token prediction task naturally creates its own "labels".
- The model learns by adjusting its predictions to match the next token in the sequence, based solely on the input text.
- This approach leverages the predicitive structure inherent data.

Example of Training GPT (Unsupervised data)

In the unsupervised pretraining phase, GPT is trained on a massive corpus of unlabeled text data. The objective is to predict the next token in a sequence, which is often called the casual language modelling (CLM) task. Here's an example of how GPT trains on unsupervised data:

1. Unsupervised Dataset Example

- Source: Text data from books, articles, website, etc.
- Example Text: "The sun rises in the east and sets in the west. This cycle happens every day"

2. Pretraining Objective

The model's goal is to predict the next word (or token) given a sequence of preceding words. This is done without any explicit labels or annotations.

For example:

- Input Sequence: "The sun rises in the ~~east~~ east and sets in the"
- Target output: "west"

3. Training Steps

Step 1: Tokenization

The raw text is tokenized into subword units using a tokenizer like Byte Pair Encoding (BPE).

For example:

- Input text: "The sun rises in the east and sets in the"

- Tokenized: [The, sun, rises, in, the, east, and, sets, in, the]

Step2: Embedding and Positional Encoding

The token is mapped to a dense vector (via the embedding layer), and positional encoding is added to preserve word order.

Step3: Transformer Decoder Blocks

The token embeddings are passed through the GPT architecture.

- Masked Multi-Head Self-Attention ensures the model only attends to tokens up to the current position (causality).
- Feedforward layers extract deeper relationships.

Step4: Output Prediction

The output layer predicts the probability distribution over the vocabulary for the next token.

for example:

- Input: "The sun rises in the east and sets in the"
- output Probabilities:

$$P(\text{west}) = 0.92,\ P(\text{sky}) = 0.05,\ P(\text{day}) = 0.03$$

- Predicted Token: "west"

Iterative Learning

The model processes the text sequentially, learning patterns syntax, semantics and content from millions or billions of such sequences. It updates its weights using Cross-Entropy loss to maximum the probability of the correct tokens.

Key Benefits of Unsupervised Training

1. No Labels Required : The model learns directly from Raw text, making it scalable to vast datasets.
2. General knowledge : GPT acquires a deep understanding of language structure and relationship
3. Transferability : The pretraind model can be fine - tuned for specific tasks with labeled data.

Supervised fine-Tuning in GPT

Supervised fine-tuning in the GPT transformer involves adapting the pre-trained model for specific downstream tasks by training it on labeled data. Here's a detailed breakdown of the process:

1. Purpose of Supervised Fine-Tuning

Fine-tuning builds on the general linguistick knowledge GPT acquires unsupervised pretraining. The goal is to:

- Specialize the model for a specific task (eg., text classification, summarization, or question answering).
- Use a relatively small labeled dataset for this adaption.

Fine-tuning ensures the model retains its pretrained capabilities while optimizing its performance for ~~that~~ task at hand.

2. Input Preparation for Fine-Tuning

The way inputs and output are structured depends on the downstream task. Let's look at some.

Examples:

Task 1: Task classification

- Input A sequence of tokens representing the text to be classified.

  Example: "This product is fantastic"

- Label: The category or sentiment of the text (eg positive or negative)

Task 2: Question Answering

- Input: A concatenation of the question and content.

  Example: "Question: what is the capital of France? content: France is a country in Europe, and its capital is Paris."

- Label: The span of text or token(s) in the content corresponding to the answer (e.g. "Paris").

Task 3: Summarization

- Input: A sequence of tokens representing the document to summarize.

  Example: "The long document text goes here..."

- Label: A sequence of tokens representing the summary

  Example: "This is summary".

Modification to the Pretrained Model

1. **Adding Task-Specific Layers:**
   - For tasks like classification, a new linear layer is added to map the model's output (hidden states) to the number of task-specific labels.
   - Ex: If there are 3 sentiment categories, the output dimension of the new layer is 3.
2. **Adjusting the input Format:**
   - Special tokens may be introduced to help the model understand task structure.
     → [CLS]: For classification, the embedding of the token is used as the input to the

task specific layer.

→ [SEP]: for seperating different components in the input (eg. question and content).

3. **Reinitializing Task-Specific Parameters**:
   - Weights in the new task-specific layers are initialized randomly, while the pretrained parameters remain intact.

Training Objective

During fine-tuning, the loss function is tailored to the specific task:

- **Classification**: Use cross-entropy loss to compare the model's predict probabilities with the true label.
- **Question Answering**: Use span prediction objective:
  → Predict the start and end position of the answer span in the content.
- **Summarization or Text Generation**: Use the casual language modelling loss (similar to pretraining) to maximize the probabilities of the target sequence (Summary).

Optimization Process

1. **Freezing Pre-trained Weights (Optional)**:
   - Some layer may be frozen to retain pretrained knowledge and avoid overfitting on the

task - specific data.

- Fine-tuning can also adjust all weights (full fine-tuning) if the task is complex.

2. **Gradient Updates:**

- Gradients are computed for the trainable parameters using backpropagation.
- The model's weight are updated using an optimizer like adam.

3. **Regularization Techniques:**

- To prevent overfitting on small datasets, techniques like dropout and weight decay are applied.
- Early stopping is often used to halt training when validation performance stops improving.

Training Process

- The labeled dataset is split into training, validation and test sets.
- **During fine-tuning:**

1. Forward Pass

→ Input sequence pass through the model to generate predictions.

2. Loss computations:

→ Compare prediction with ground-truth labels to compute the task-specific loss.

- **Backward pass:**
  → Compute gradients and updates the model weights to minimize the loss.

Example: Fine-Tuning for text classification

Suppose we want to fine-tune GPT for sentiment analysis (positive or negative sentiments):

1. **Input format:**
   - **Input text:** "I absolutely loved this movie!"
   - **Tokenized input:** [I, absolutely, loved, this, movie!]

2. **Model Adaption**
   - Add a classification head (linear layer) on the top of the GPT model to output two logits (for positive and negative).

3. **Training Objective:**
   - Cross-entropy loss:

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

Where:
→ C: No. of classes (2 in this cases)
→ y_i: Ground-truth label (one-hot encoded)

- $\hat{y}_i$: Prediction probabilities for class i.

4. **Optimization** :
   - Train the model using label examples to minimize the loss.

Evaluation

After fine-tuning the model is evaluated on the test set to measure task-specific performance:

- Metrics:
  - → Classification Accuracy: F1 score, Precision, recall
  - → Question Answering: Exact match (EM), F1 score
  - → Summarization: ROUGE, BELU

Advantage of Fine-Tuning

1. **Task Efficiency**:
   - Fine-tuning allows leveraging a pretrained model's general knowledge, reducing the need for large labeled datasets.

2. **Flexibility**:
   - A single model can be fine-tuned for various task with minimal task-specific changes.

3. **Improved Performance**:
   - Fine-tuned models often outperform task-specific models trained from scratch.

Example: Supervised Fine-Tuning in GPT

Supervised fine-tuning adapts a pre-trained GPT model for a specific Task. (e.g., sentiment analysis, question answering) by training it on a labeled dataset. Let's walk through an example where GPT is fine-tuned for sentiment classification.

1. Task Overview

- Objective: Classify reviews as positive or negative.
- Dataset: A labeled dataset of movie reviews.

  Example entries:

  - Input: "The movie was fantastic! I loved it." → Label: "positive"
  - Input: "The plot was boring and predictable." → Label: "Negative"

2. Fine-Tuning Steps

Step 1: Prepare the dataset

- Input: Sentence (reviews).
- Labels: Sentiment (Positive or Negative)

Convert the dataset into a suitable format:

- Combine input text and label into single sequence with special separator.

eg:- "[CLS] The movie was fantastic! I loved it. [SEP] Positive"

Here:

- [CLS]: Start of sequence token.
- [SEP]: Separator between input text and label.

Step 2: Modify the model Architecture.

- Add a classification head (fully connected layer) on top of the GPT model.
- This head maps the final hidden state of the [CLS] token to output logits for each label (Positive or Negative).

Step 3: Define Loss function

- Use Cross-Entropy Loss to compare the predicted probabilities with the true labels.

Step 4: Fine-Tuning Process

1. Pretrained GPT weights: Load the GPT model with weights from unsupervised pretraining.
2. Task-Specific Training:
   - Forward pass: Pass the input sequence through GPT.
   - Compute Loss: Use cross-entropy loss on the predicted label probabilities.
   - Backward pass: Update weights using backpropagation
3. Optimizer: Use Adam optimizer with a small learning rate to prevent overfitting.

Example workflow

Input:

- Training example:

  "The movie was fantastic! I loved it".

  True label: Positive

Steps:

1. Tokenizer:

   input text is tokenized: [The, movie, was, fantastic, !, I, loved, it, .]

2. Embedding:

   Each token is converted to dense vectors.

3. Model Processing:
   - Pass through Transformer decoder blocks.
   - The final hidden state of the [CLS] token is extracted.

4. Classification Head:
   - The [CLS] hidden state is passed through the classification head.
   - Outputs logit for "positive" and "Negative".

5. Prediction:
   - Apply softmax to logits to get probabilities:

     $P(\text{positive}) = 0.85$, $P(\text{Negative}) = 0.15$

   - Predicted label: "Positive".

6. Loss Computation:

- Use the cross-entropy loss between predicted probabilities and the true label.

7. Backpropagation:

- Update model weights to minimize the loss.

Output

- After fine-tuning on the dataset, the GPT model can classify unseen reviews accurately.
- Input "The acting was terrible."
- Predicted label: "Negative"

Key Points

- Supervised fine-tuning allows the GPT model to specialize in specific tasks.
- It starts from pre-trained weights, leveraging prior knowledge of language structure.
- This process is resources-efficient compared to training a model from scratch.

Dropout Layer in Architecture

Yes. The GPT architecture includes dropout layers, primarily to prevent overfitting and improve generalization during training. Here's where dropout is used in the GPT models:

Dropout in Transformer Decoder Blocks

Dropout is applied in the following components of each decoder block.

- **Self-Attention Mechanism:** Dropout is applied to the output of the multi-head self attention layer before it is added to the residual connection.
- **Feedforward NN:** Dropout is applied after the activation function in the feedforward layers.
- **Residual Connections:** Dropout is also applied to the outputs of sublayers before they are added to the residual connection and normalized.

Configurable Dropout Rates

- The dropout rate is a hyperparameter and is typically set during training (e.g 0.1 or 0.2). It controls the fraction of neurons that are randomly "dropped" (i.e, set to zero) during each forward pass in training.

Why Dropout in GPT?

- **Pretraining**: Dropout helps regularize the model while training on massive unsupervised datasets.
- **Fine-tuning**: Dropout prevents overfitting on smaller labelled datasets during supervised fine-tuning.