

Module 13: Transformers and Attention

Overview

Transformers have revolutionized natural language processing and are increasingly used for vision and other domains. This module explains the attention mechanism that enables transformers to process long sequences effectively, the complete transformer architecture, and how pre-trained models like BERT and GPT enable state-of-the-art performance through transfer learning.

1. The Problem with Recurrent Networks

Sequential Processing

RNNs/LSTMs process sequences one step at a time:

- Cannot parallelize across sequence positions
- Long training times on GPUs

Long-Range Dependencies

Information must propagate through many steps:

- Vanishing gradients over long sequences
- Difficult to remember early tokens when processing later ones

The Attention Solution

Attention allows direct connections between any positions:

- Parallel processing of entire sequence
- Constant path length between any two positions
- Better gradient flow

2. The Attention Mechanism

Intuition

“Pay attention” to relevant parts of the input when producing each output.

For example, when translating “The cat sat on the mat” to French:

- When generating “chat” (cat), attend strongly to “cat”
- When generating “tapis” (mat), attend strongly to “mat”

Attention as Soft Retrieval

- **Query:** What am I looking for?
- **Keys:** What do I have available?
- **Values:** What information do I retrieve?

Attention computes a weighted sum of Values, where weights depend on how well each Key matches the Query.

3. Self-Attention

The Idea

In self-attention, queries, keys, and values all come from the same sequence. Each position attends to all positions (including itself).

Computation

For input sequence X (n positions \times d dimensions):

1. **Project** to queries, keys, values:

$$\begin{aligned} Q &= X \times W_Q \\ K &= X \times W_K \\ V &= X \times W_V \end{aligned}$$

2. **Compute attention scores:**

$$\text{Scores} = Q \times K^T / \sqrt{d_k}$$

3. **Apply softmax** to get weights:

$$\text{Weights} = \text{softmax}(\text{Scores})$$

4. **Weighted sum** of values:

$$\text{Output} = \text{Weights} \times V$$

Scaled Dot-Product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}(Q \times K^T / \sqrt{d_k}) \times V$$

The scaling by $\sqrt{d_k}$ prevents dot products from growing too large.

4. Multi-Head Attention

Why Multiple Heads?

Different heads can learn to attend to different things: - Syntactic relationships - Semantic relationships - Different distance patterns

Computation

Run h attention functions in parallel:

$$\text{head}_i = \text{Attention}(Q \times W_{Q^i}, K \times W_{K^i}, V \times W_{V^i})$$

Concatenate and project:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) \times W_O$$

Typical: 8-16 attention heads.

5. The Transformer Architecture

Encoder Block

```
Input
  ↓
Multi-Head Self-Attention
  ↓
Add & Layer Norm (residual connection)
  ↓
Feed-Forward Network
  ↓
Add & Layer Norm
  ↓
Output
```

Key Components

Residual Connections: Add input to output (helps gradient flow)

Layer Normalization: Normalize activations for training stability

Feed-Forward Network: Position-wise fully connected layers

$$\text{FFN}(x) = \text{ReLU}(x \times W_1 + b_1) \times W_2 + b_2$$

Positional Encoding: Since attention is permutation-invariant, add position information

$$\text{PE}(\text{pos}, 2i) = \sin(\text{pos} / 10000^{(2i/d)})$$

$$\text{PE}(\text{pos}, 2i+1) = \cos(\text{pos} / 10000^{(2i/d)})$$

6. Encoder vs Decoder

Encoder-Only (BERT-style)

- Bidirectional: each position sees all positions
- Good for: classification, NER, question answering
- Pre-trained with masked language modeling

Decoder-Only (GPT-style)

- Autoregressive: each position only sees previous positions
- Uses causal masking in attention
- Good for: text generation
- Pre-trained with next-token prediction

Encoder-Decoder (T5, BART)

- Encoder processes input (bidirectional)
 - Decoder generates output (autoregressive)
 - Cross-attention: decoder attends to encoder
 - Good for: translation, summarization
-

7. BERT: Bidirectional Encoder

Pre-training Objectives

Masked Language Modeling (MLM): - Randomly mask 15% of tokens - Predict original tokens - Learns bidirectional context

Next Sentence Prediction (NSP): - Predict if sentence B follows sentence A - Learns sentence relationships

Fine-tuning

Add task-specific layer on top: - Classification: Use [CLS] token embedding - Token classification: Use per-token embeddings - Question answering: Predict start/end positions

8. GPT: Generative Pre-Training

Pre-training

Predict next token given previous tokens:

$$P(x_t \mid x_1, \dots, x_{t-1})$$

Causal masking ensures no future information leaks.

Scaling

GPT-3 has 175 billion parameters. Larger models show: - Better few-shot learning - Emergent capabilities - General-purpose understanding

Prompting

Instead of fine-tuning, provide task instructions:

Translate English to French:

"The cat sat on the mat" →

9. Using Pre-trained Transformers

Hugging Face Transformers

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

# Load pre-trained model
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = AutoModelForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels=2
)

# Tokenize input
inputs = tokenizer("This movie was great!", return_tensors="pt")

# Forward pass
outputs = model(**inputs)
predictions = outputs.logits.argmax(dim=-1)
```

Fine-tuning Pipeline

1. Load pre-trained model + tokenizer
 2. Add task-specific head
 3. Prepare dataset with tokenization
 4. Train with small learning rate
 5. Evaluate on held-out data
-

10. Beyond Text

Vision Transformers (ViT)

Split images into patches, treat as sequence: - Each patch becomes a token - Self-attention over patches - Competitive with CNNs at scale

Multimodal Transformers

- CLIP: Vision + text alignment
 - DALL-E: Text to image generation
 - GPT-4V: Vision understanding
-

Key Takeaways

1. **Attention** allows direct connections between any positions
2. **Self-attention** computes Query-Key-Value within a sequence
3. **Multi-head attention** learns diverse attention patterns

4. **Transformers** stack attention with FFN and layer norm
 5. **BERT** (encoder) is bidirectional, good for understanding
 6. **GPT** (decoder) is autoregressive, good for generation
 7. **Pre-training + fine-tuning** enables efficient transfer learning
 8. **Hugging Face** provides easy access to pre-trained models
-

Course Conclusion

This module completes our journey through applied machine learning: - Fundamentals (Modules 1-5) - Neural networks and deep learning (Modules 6, 10, 13) - Classical methods (Module 7) - Data representation (Modules 8-9) - Practical skills (Module 11) - Responsible AI (Module 12)

Good luck with your capstone projects!

Further Reading

- [Links to be added]
- “Attention Is All You Need” (Vaswani et al., 2017)
- Illustrated Transformer (Jay Alammar)
- Hugging Face documentation