DL Demystified 10 - Transformers Extended - Additional Insights

January 29, 2025

[1]:
```
######################################################################
#                                                                    #
#  CS435 Generative AI: Security, Ethics and Governance              #
#                                                                    #
#  Instructor: Dr. Adnan Masood                                      #
#  Contact:    adnanmasood@gmail.com                                 #
#                                                                    #
#  Notebook is MIT Licensed                                          #
######################################################################
```

1 Transformers and Attention

Welcome to today's lesson! In this notebook, we will explore **Transformers and Attention Mechanisms** in detail. This will include everything from basic intuition to building a simple transformer from scratch in PyTorch. We'll go step by step to understand the concepts, math, and code behind these technologies.

1.1 Building an Intuitive Understanding

To make it easier to grasp the material, we'll explain it at five levels: 1. What is it and why does it matter? 2. Where it came from and how it works at a high level. 3. Understanding the math behind attention and transformers. 4. Writing PyTorch code to implement and experiment. 5. Building a transformer from scratch.

1.2 1. Attention in Transformers?

Imagine reading a book and trying to summarize it. A transformer is like a super-smart assistant that not only reads the book but also pays attention to the important parts, understands their relationships, and creates a summary that's easy to understand.

Why is this important? Because machines need to understand relationships between words to translate languages, generate text, or even answer questions like humans!

Transformers were introduced in 2017 in a research paper titled "Attention is All You Need." Before transformers, models like RNNs and LSTMs struggled to handle long-term dependencies in text. Transformers solved this by using "self-attention" to focus on the most relevant parts of the input.

Key ideas of transformers: 1. **Self-Attention**: Helps the model focus on important words in a sentence. 2. **Positional Encoding**: Adds information about the order of words. 3. **Multi-Head Attention**: Looks at the input from multiple perspectives. 4. **Feedforward Networks**: Processes attention outputs for prediction. 5. **Layer Normalization**: Stabilizes training.

1.3 Understanding Attention

Let's break down the self-attention mechanism step by step.

1.3.1 Self-Attention Formula

The main idea is to compute a weighted sum of values where the weights are based on pairwise word similarity. For each word in a sentence, we compute:

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

Where: - Q: Query - K: Key - V: Value - d_k: Dimension of the key vectors

---

1.3.2 Breaking it Down

1. Q and K are compared to measure similarity.
2. The result is scaled by $\sqrt{d_k}$ to stabilize gradients.
3. We apply Softmax to ensure weights sum to 1.
4. Multiply weights with V to get the output.

---

[2]:
```
import torch
import torch.nn.functional as F

# Example Inputs
queries = torch.tensor([[1.0, 0.0], [0.0, 1.0]])  # Q
keys = torch.tensor([[1.0, 0.0], [0.0, 1.0]])     # K
values = torch.tensor([[10.0, 0.0], [0.0, 10.0]]) # V

# Compute Attention Scores
scores = torch.mm(queries, keys.T) / torch.sqrt(torch.tensor(keys.shape[-1],␣
 ↪dtype=torch.float32))
weights = F.softmax(scores, dim=-1)
output = torch.mm(weights, values)

print("Attention Weights:", weights)
print("Output:", output)
```

```
Attention Weights: tensor([[0.6698, 0.3302],
        [0.3302, 0.6698]])
Output: tensor([[6.6976, 3.3024],
        [3.3024, 6.6976]])
```

1.4 Building Multi-Head Attention

Below, we create a simple multi-head attention layer in PyTorch.

[3]:
```python
class MultiHeadAttention(torch.nn.Module):
    def __init__(self, embed_size, num_heads):
        super(MultiHeadAttention, self).__init__()
        assert embed_size % num_heads == 0, "Embedding size must be divisible by number of heads."

        self.head_dim = embed_size // num_heads
        self.num_heads = num_heads
        self.query = torch.nn.Linear(embed_size, embed_size)
        self.key = torch.nn.Linear(embed_size, embed_size)
        self.value = torch.nn.Linear(embed_size, embed_size)
        self.fc_out = torch.nn.Linear(embed_size, embed_size)

    def forward(self, x):
        N, seq_length, embed_size = x.shape
        Q = self.query(x)
        K = self.key(x)
        V = self.value(x)

        # Reshape for multi-heads
        Q = Q.view(N, seq_length, self.num_heads, self.head_dim).transpose(1, 2)
        K = K.view(N, seq_length, self.num_heads, self.head_dim).transpose(1, 2)
        V = V.view(N, seq_length, self.num_heads, self.head_dim).transpose(1, 2)

        # Scaled Dot-Product Attention
        attention_scores = torch.matmul(Q, K.transpose(-2, -1)) / torch.sqrt(torch.tensor(self.head_dim, dtype=torch.float32))
        attention_weights = F.softmax(attention_scores, dim=-1)
        output = torch.matmul(attention_weights, V)

        # Concatenate heads
        output = output.transpose(1, 2).contiguous().view(N, seq_length, embed_size)

        return self.fc_out(output)
```

[4]:
```
import os, sys, platform, datetime, uuid, socket

def signoff(name="Anonymous"):
    colab_check = "Yes" if 'google.colab' in sys.modules else "No"
    mac_addr = ':'.join(format((uuid.getnode() >> i) & 0xff, '02x') for i in␣
 ↪reversed(range(0, 48, 8)))
    print("+++ Acknowledgement +++")
    print(f"Executed on: {datetime.datetime.now()}")
    print(f"In Google Colab: {colab_check}")
    print(f"System info: {platform.system()} {platform.release()}")
    print(f"Node name: {platform.node()}")
    print(f"MAC address: {mac_addr}")
    try:
        print(f"IP address: {socket.gethostbyname(socket.gethostname())}")
    except:
        print("IP address: Unknown")
    print(f"Signing off, {name}")

signoff("Ali Muhammad Asad")
```

```
+++ Acknowledgement +++
Executed on: 2025-01-29 01:29:33.567065
In Google Colab: No
System info: Linux 6.8.0-51-generic
Node name: alimuhammad-Inspiron-7559
MAC address: 20:47:47:74:94:05
IP address: 127.0.1.1
Signing off, Ali Muhammad Asad
```

[ ]: