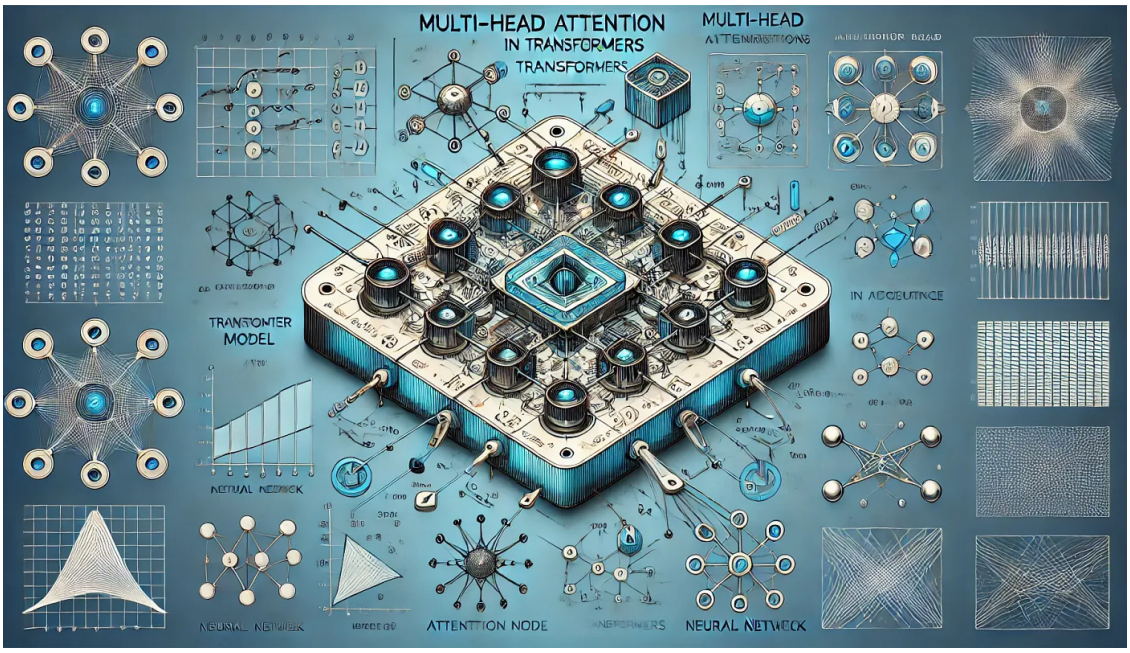
Multi-Head Attention

July 18, 2024

0.0.1 Multi-Head Attention From Scratch

By Cristian Leo

Link to Article



0.0.2 Required Libraries

[1]:
```
import numpy as np
```

0.0.3 Multi-Head Attention Class

[2]:
```
class MultiHeadAttention:
    """
    Multi-head attention.

    Parameters:
    num_hiddens: int
        Number of hidden units.
    num_heads: int
```

```
        Number of attention heads.
    dropout: float
        Dropout rate.
    bias: bool
        Whether to include bias parameters in the model.
    """
    def __init__(self, num_hiddens, num_heads, dropout=0.0, bias=False):
        self.num_heads = num_heads
        self.num_hiddens = num_hiddens
        self.d_k = self.d_v = num_hiddens // num_heads

        self.W_q = np.random.rand(num_hiddens, num_hiddens)
        self.W_k = np.random.rand(num_hiddens, num_hiddens)
        self.W_v = np.random.rand(num_hiddens, num_hiddens)
        self.W_o = np.random.rand(num_hiddens, num_hiddens)

        if bias:
            self.b_q = np.random.rand(num_hiddens)
            self.b_k = np.random.rand(num_hiddens)
            self.b_v = np.random.rand(num_hiddens)
            self.b_o = np.random.rand(num_hiddens)
        else:
            self.b_q = self.b_k = self.b_v = self.b_o = np.zeros(num_hiddens)

    def transpose_qkv(self, X):
        """
        Transposition for batch processing

        Parameters:
        X: np.ndarray
            Input tensor

        Returns:
        np
            Transposed tensor
        """
        X = X.reshape(X.shape[0], X.shape[1], self.num_heads, -1)
        X = X.transpose(0, 2, 1, 3)
        return X.reshape(-1, X.shape[2], X.shape[3])

    def transpose_output(self, X):
        """
        Transposition for output

        Parameters:
        X: np.ndarray
            Input tensor
```

```
        Returns:
        np
            Transposed tensor
        """
        X = X.reshape(-1, self.num_heads, X.shape[1], X.shape[2])
        X = X.transpose(0, 2, 1, 3)
        return X.reshape(X.shape[0], X.shape[1], -1)

    def scaled_dot_product_attention(self, Q, K, V, valid_lens):
        """
        Scaled dot product attention

        Parameters:
        Q: np.ndarray
            Query tensor
        K: np.ndarray
            Key tensor
        V: np.ndarray
            Value tensor
        valid_lens: np.ndarray
            Valid lengths for the query

        Returns:
        np
            Output tensor
        """
        d_k = Q.shape[-1]
        scores = np.matmul(Q, K.transpose(0, 2, 1)) / np.sqrt(d_k)
        if valid_lens is not None:
            mask = np.arange(scores.shape[-1]) < valid_lens[:, None]
            scores = np.where(mask[:, None, :], scores, -np.inf)
        attention_weights = np.exp(scores - np.max(scores, axis=-1,␣
↪keepdims=True))
        attention_weights /= attention_weights.sum(axis=-1, keepdims=True)
        return np.matmul(attention_weights, V)

    def forward(self, queries, keys, values, valid_lens):
        """
        Forward pass

        Parameters:
        queries: np.ndarray
            Query tensor
        keys: np.ndarray
            Key tensor
        values: np.ndarray
```

```
            Value tensor
        valid_lens: np.ndarray
            Valid lengths for the query

        Returns:
        np
            Output tensor
        """
        queries = self.transpose_qkv(np.dot(queries, self.W_q) + self.b_q)
        keys = self.transpose_qkv(np.dot(keys, self.W_k) + self.b_k)
        values = self.transpose_qkv(np.dot(values, self.W_v) + self.b_v)

        if valid_lens is not None:
            valid_lens = np.repeat(valid_lens, self.num_heads, axis=0)

        output = self.scaled_dot_product_attention(queries, keys, values,␣
 ↪valid_lens)
        output_concat = self.transpose_output(output)
        return np.dot(output_concat, self.W_o) + self.b_o
```

[3]:
```
# Define dimensions and initialize multi-head attention
num_hiddens, num_heads = 100, 5
attention = MultiHeadAttention(num_hiddens, num_heads, dropout=0.5, bias=False)

# Define sample data
batch_size, num_queries, num_kvpairs = 2, 4, 6
valid_lens = np.array([3, 2])
print(valid_lens)
```

```
[3 2]
```

[4]:
```
X = np.random.rand(batch_size, num_queries, num_hiddens)  # Use random data to␣
 ↪simulate input queries
Y = np.random.rand(batch_size, num_kvpairs, num_hiddens)  # Use random data to␣
 ↪simulate key-value pairs
print("Query data shape:", X.shape)
print("Key-value data shape:", Y.shape)
```

```
Query data shape: (2, 4, 100)
Key-value data shape: (2, 6, 100)
```

[5]:
```
# Apply multi-head attention
output = attention.forward(X, Y, Y, valid_lens)
print("Output shape:", output.shape)  # Expected shape: (batch_size,␣
 ↪num_queries, num_hiddens)

# Output sample data
```

```
print(output[0][0])
```

```
Output shape: (2, 4, 100)
[1205.50019525 1318.40294837 1202.74795967 1455.57568921 1265.41910134
 1412.39503534 1265.6051992  1105.08395357 1241.69740915 1284.52168046
 1288.35849879 1308.93026427 1248.64984445 1312.26703139 1317.89851446
 1339.90026899 1407.58060749 1303.10053768 1297.05734393 1088.51859258
 1220.89544194 1257.68897785 1260.03487276 1220.67801893 1318.56893146
 1404.3526966  1348.78886246 1400.71298358 1379.90853954 1182.99515963
 1294.67726698 1276.89072956 1365.85742282 1355.08934127 1393.73412172
 1297.58821378 1199.39144261 1202.4538607  1379.05487506 1337.54203429
 1260.82261499 1349.45802824 1279.10416123 1257.93569313 1250.30616378
 1322.03127264 1186.5407581  1237.53190197 1359.0163692  1241.83989194
 1309.08616704 1309.18611101 1284.6503605  1402.85942108 1272.66630628
 1306.35012271 1384.70001294 1343.42940202 1282.52728987 1295.70366797
 1138.04981365 1229.61215801 1325.80515729 1331.06053924 1311.59336619
 1364.84760246 1357.24049113 1339.48407166 1287.65925432 1334.2958758
 1452.94043944 1312.22185253 1255.75561698 1240.39891042 1282.61245247
 1133.35154662 1374.52161694 1241.82226361 1248.0423815  1322.6988869
 1229.3635741  1282.01347991 1317.57304151 1335.60310279 1302.06795861
 1263.22080951 1332.02077941 1324.73558321 1327.52108457 1301.3535082
 1292.27089407 1206.04247744 1297.88020996 1189.30013174 1306.96166691
 1360.9462695  1312.06452925 1198.68776006 1297.10786271 1353.59958087]
```