كلية الحوسبة والمعلوماتية |
College of Computing and Informatics
الجامعة السعودية الإلكترونية
SAUDI ELECTRONIC UNIVERSITY
2011-1432

Second Semester – 2021/2022

Course Code	DS660
Course Name	Deep Learning Techniques
Assignment type	Critical Thinking
Module	10
Total Points	105 Points

Student ID	G200007615
Student Name	Abdulaziz Aqlumayzi
CRN	21604

Solutions:

Critical Thinking Assignment 3

Using Additive Attention as the Scoring Function

Introduction

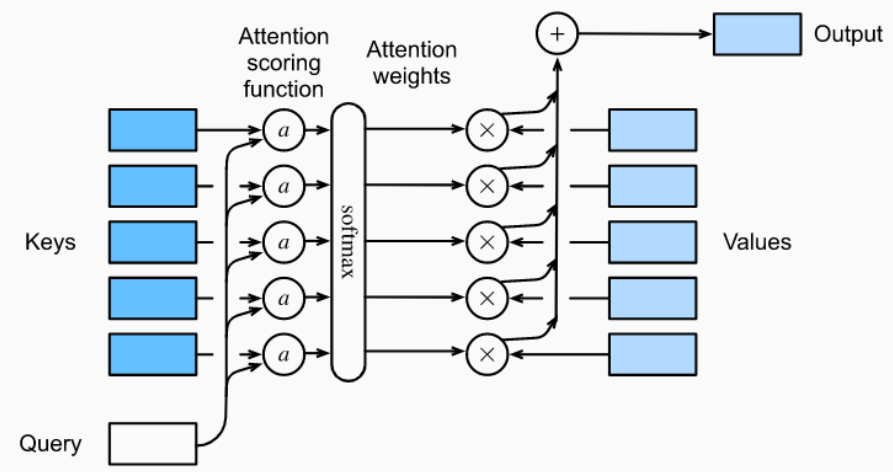
In this activity, we will explain when we can utilize additive attention as the scoring function. Create a Python class application that demonstrates additive attention. Show the AdditiveAttention class in action with a toy example in which the shapes (batch size, number of steps or sequence length in tokens, feature size) of queries, keys, and values are (2, 1, 20), (2, 10, 2), and (2, 10, 4), respectively. Lastly, a heatmaps can be used to display attention weights.

Additive Attention as the Scoring Function

To begin, let's define the **Scoring Function** (or **Attention Scoring Function**), the outcomes of which were effectively fed into a softmax operation. As a consequence, we derived a probability distribution (attention weights) over values that are paired with keys. Finally, the output of the attention pooling is just a weighted sum of the values depending on these attention weights.

Figure 1

Computing of Attention scoring function



Let us now explain when we can use **Additive Attention**. When queries and keys are vectors of varying lengths, we may utilize additive attention as the scoring function. Given a query $q \in R^q$ and a key $k \in R^k$, the additive attention scoring function is $\alpha(q,k) = w_v^T \tanh(W_q q + W_k k) \in R$ where learnable parameters $W_q \in R^{h \times q}$, $W_k \in R^{h \times k}$, and $w_v \in R^h$. Equivalent to $\alpha(q,k) = w_v^T \tanh(W_q q + W_k k) \in R$, the query and key are concatenated and put into an MLP with a single hidden layer with ***h*** hidden units, a hyperparameter. Using ***tanh*** as the activation function and turning off bias terms.

Full Python Programming Code

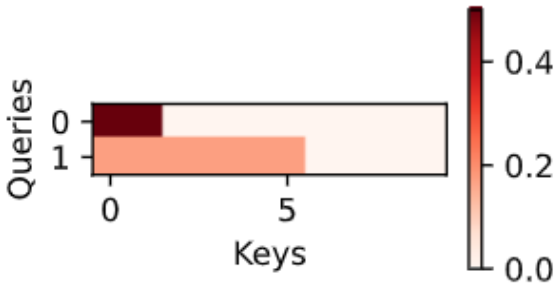
```
# importing packages
import tensorflow as tf
from d2l import tensorflow as d2l


# building the masked softmax class
def masked_softmax(X, valid_lens):
    """Perform softmax operation by masking elements on the last axis."""
    if valid_lens is None:
        return tf.nn.softmax(X, axis=-1)
    else:
        shape = X.shape
        if len(valid_lens.shape) == 1:
            valid_lens = tf.repeat(valid_lens, repeats=shape[1])

        else:
            valid_lens = tf.reshape(valid_lens, shape=-1)
        X = d2l.sequence_mask(tf.reshape(X, shape=(-1, shape[-1])),
valid_lens, value=-1e6)
        return tf.nn.softmax(tf.reshape(X, shape=shape), axis=-1)


# building the additive attention class
class AdditiveAttention(tf.keras.layers.Layer):
    """Additive attention."""
    def __init__(self, key_size, query_size, num_hiddens, dropout,
**kwargs):
        super().__init__(**kwargs)
        self.W_k = tf.keras.layers.Dense(num_hiddens, use_bias=False)
        self.W_q = tf.keras.layers.Dense(num_hiddens, use_bias=False)
        self.w_v = tf.keras.layers.Dense(1, use_bias=False)
        self.dropout = tf.keras.layers.Dropout(dropout)

    def call(self, queries, keys, values, valid_lens, **kwargs):
        queries, keys = self.W_q(queries), self.W_k(keys)

        features = tf.expand_dims(queries, axis=2) + tf.expand_dims(
            keys, axis=1)
        features = tf.nn.tanh(features)
        scores = tf.squeeze(self.w_v(features), axis=-1)
```

```
        self.attention_weights = masked_softmax(scores, valid_lens)
        return tf.matmul(self.dropout(
            self.attention_weights, **kwargs), values)


# the toy example is used to demonstrate the AdditiveAttention class.
queries, keys = tf.random.normal(shape=(2, 1, 20)), tf.ones((2, 10, 2))
values = tf.repeat(tf.reshape(tf.range(40, dtype=tf.float32), shape=(1, 10,
4)), repeats=2, axis=0)
valid_lens = tf.constant([2, 6])
attention = AdditiveAttention(key_size=2, query_size=20, num_hiddens=8,
dropout=0.1)
attention(queries, keys, values, valid_lens, training=False)


# a heatmaps to show the attention weights
d2l.show_heatmaps(tf.reshape(attention.attention_weights, (1, 1, 2, 10)),
                  xlabel='Keys', ylabel='Queries')
```

Demonstrating the AdditiveAttention class with a toy example result:

```
<tf.Tensor: shape=(2, 1, 4), dtype=float32, numpy=
array([[[ 2.,  3.,  4.,  5.]],

       [[10., 11., 12., 13.]]], dtype=float32)>
```

A heatmaps attention weights result:



References

Zhang, A., Lipton, Z., Li, M., Smola, A., Werness, B., Hu, R., Zhang, S., & Tay, Y. (2022). *Dive into Deep Learning*. https://d2l.ai/index.html.