TensorFlow2教程-遮罩和填充

在構建深度學習模型，特別是進行序列的相關任務時，經常會用到遮罩和填充技術

In [1]:
```
from __future__ import absolute_import, division, print_function, unicode_literal

import numpy as np

import tensorflow as tf

from tensorflow.keras import layers
```

1 填充序列數據

處理序列資料時，常常會遇到不同長度的文本，例如處理某些句子時：

In [2]:
```
[
  ["The", "weather", "will", "be", "nice", "tomorrow"],
  ["How", "are", "you", "doing", "today"],
  ["Hello", "world", "!"]
]
```

Out[2]:
```
[['The', 'weather', 'will', 'be', 'nice', 'tomorrow'],
 ['How', 'are', 'you', 'doing', 'today'],
 ['Hello', 'world', '!']]
```

電腦無法理解字元，我們一般將詞語轉換為對應的id

In [3]:
```
[
  [83, 91, 1, 645, 1253, 927],
  [73, 8, 3215, 55, 927],
  [71, 1331, 4231]
]
```

Out[3]:
```
[[83, 91, 1, 645, 1253, 927], [73, 8, 3215, 55, 927], [71, 1331, 4231]]
```

由於深度資訊模型的輸入一般是固定的張量，所以我們需要對短文本進行填充（或對長文本進行截斷），使每個樣本的長度相同。 Keras提供了一個API，可以通過填充和截斷，獲取等長的資料：
tf.keras.preprocessing.sequence.pad_sequences

In [4]:
```
raw_inputs = [
  [83, 91, 1, 645, 1253, 927],
  [73, 8, 3215, 55, 927],
  [711, 632, 71]
]
# 默認左填充
padded_inputs = tf.keras.preprocessing.sequence.pad_sequences(raw_inputs)
print(padded_inputs)
# 右填充需要設 padding='post'
padded_inputs = tf.keras.preprocessing.sequence.pad_sequences(raw_inputs,
                                                              padding='post')
print(padded_inputs)
```

```
[[  83   91    1  645 1253  927]
 [   0   73    8 3215   55  927]
 [   0    0    0  711  632   71]]
[[  83   91    1  645 1253  927]
 [  73    8 3215   55  927    0]
 [ 711  632   71    0    0    0]]
```

2 遮罩

現在所有樣本都獲得了同樣的長度，在求損失函數或輸出的時候往往需要知道那些部分是填充的，需要忽略。這種忽略機制就是遮罩。

keras中三種添加遮罩的方法：

- 添加一個keras.layer.Masking的網路層
- 配置keras.layers.Embedding層的mask_zero=True
- 在支援mark的網路層中，手動傳遞參數

2.1 遮罩生成層：Embedding和Masking

Embedding和Masking會創建一個遮罩張量，附加到返回的輸出中。

In [8]:
```
embedding = layers.Embedding(input_dim=5000, output_dim=16, mask_zero=True)
mask_output= embedding(padded_inputs)
print(mask_output._keras_mask)
```

```
tf.Tensor(
[[ True  True  True  True  True  True]
 [ True  True  True  True  True False]
 [ True  True  True False False False]], shape=(3, 6), dtype=bool)
```

In [5]:
```
# 使用遮罩層
masking_layer = layers.Masking()
unmasked_embedding = tf.cast(
tf.tile(tf.expand_dims(padded_inputs, axis=-1), [1, 1, 16]),
tf.float32)
masked_embedding = masking_layer(unmasked_embedding)
print(masked_embedding._keras_mask)
```

```
tf.Tensor(
[[ True  True  True  True  True  True]
 [ True  True  True  True  True False]
 [ True  True  True False False False]], shape=(3, 6), dtype=bool)
```

mark是帶有2D布林張量(batch_size, sequence_length)，其中每個單獨的False指示在處理過程中應忽略相應的時間步資料。

2.2 函數API和序列API中使用遮罩

使用函數式API或序列API時，由Embedding或Masking層生成的遮罩將通過網路傳播到能夠使用它們的任何層（例如RNN層）。Keras將自動獲取與輸入相對應的遮罩，並將其傳遞給知道如何使用該遮罩的任何層。

請注意，在子類化模型或圖層的call方法中，遮罩不會自動傳播，因此將需要手動將mask參數傳遞給需要的圖層。

下面展示了LSTM層怎麼自動接受遮罩

In [6]:
```
# 序列式API
model = tf.keras.Sequential([
  layers.Embedding(input_dim=5000, output_dim=16, mask_zero=True),
  layers.LSTM(32),
])
```

In [7]:
```
# 函數式API
inputs = tf.keras.Input(shape=(None,), dtype='int32')
x = layers.Embedding(input_dim=5000, output_dim=16, mask_zero=True)(inputs)
outputs = layers.LSTM(32)(x)

model = tf.keras.Model(inputs, outputs)
```

也可以直接在層間傳遞遮罩參數

可以處理遮罩的圖層（例如LSTM圖層）在其圖層中的**call**方法中有一個mask參數。

同時，產生遮罩的圖層（例如Embedding）會公開compute_mask(input, previous_mask)，以獲取遮罩。

因此，可以執行以下操作：

In [8]:
```
class MyLayer(layers.Layer):

    def __init__(self, **kwargs):
        super(MyLayer, self).__init__(**kwargs)
        self.embedding = layers.Embedding(input_dim=5000, output_dim=16, mask_zer
        self.lstm = layers.LSTM(32)

    def call(self, inputs):
        x = self.embedding(inputs)
        # 也可手動構造遮罩
        mask = self.embedding.compute_mask(inputs)
        output = self.lstm(x, mask=mask)  # The layer will ignore the masked valu
        return output

layer = MyLayer()
x = np.random.random((32, 10)) * 100
x = x.astype('int32')
layer(x)
```

Out[8]:
```
<tf.Tensor: shape=(32, 32), dtype=float32, numpy=
array([[ 0.00290916,  0.00114209,  0.00177287, ..., -0.00501184,
        -0.00386718,  0.00442083],
       [ 0.00179124, -0.00290007,  0.00513888, ...,  0.00875426,
        -0.00810401, -0.00435879],
       [ 0.00025985, -0.00370021, -0.00526272, ..., -0.00189606,
         0.00106883,  0.00700225],
       ...,
       [-0.00540151, -0.01014827,  0.00695395, ..., -0.00501114,
        -0.00454895,  0.00930379],
       [-0.00488472,  0.00584594, -0.00532092, ...,  0.00173811,
        -0.00213989,  0.0004227 ],
       [-0.00273294,  0.00197935, -0.00656161, ...,  0.01354779,
        -0.00077409,  0.0051595 ]], dtype=float32)>
```

在自訂網路層中支援遮罩

有時，可能需要編寫生成遮罩的圖層（例如Embedding），或需要修改當前遮罩的圖層。

例如，產生張量的時間維度與其輸入不同的任何層，都需要修改當前遮罩，以便下游層能夠適當考慮遮罩的時間步長。

為此，自建網路層應實現layer.compute_mask()方法，該方法將根據輸入和當前遮罩生成一個新的遮罩。

大多數圖層都不會修改時間維度，因此無需擔心掩蓋。compute_mask()在這種情況下，預設行為是僅通過當前蒙版。

TemporalSplit修改當前蒙版的圖層的示例。

In [9]:
```
# 數據拆分時的遮罩
class TemporalSplit(tf.keras.layers.Layer):

    def call(self, inputs):
        # 將資料沿時間維度一分為二
        return tf.split(inputs, 2, axis=1)

    def compute_mask(self, inputs, mask=None):
        # 將遮罩也一分為二
        if mask is None:
            return None
        return tf.split(mask, 2, axis=1)

first_half, second_half = TemporalSplit()(masked_embedding)
print(first_half._keras_mask)
print(second_half._keras_mask)
```

```
tf.Tensor(
[[ True  True  True]
 [ True  True  True]
 [ True  True  True]], shape=(3, 3), dtype=bool)
tf.Tensor(
[[ True  True  True]
 [ True  True False]
 [False False False]], shape=(3, 3), dtype=bool)
```

自己構建遮罩，把為 0 的掩掉

In [10]:
```
class CustomEmbedding(tf.keras.layers.Layer):

    def __init__(self, input_dim, output_dim, mask_zero=False, **kwargs):
        super(CustomEmbedding, self).__init__(**kwargs)
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.mask_zero = mask_zero

    def build(self, input_shape):
        self.embeddings = self.add_weight(
        shape=(self.input_dim, self.output_dim),
        initializer='random_normal',
        dtype='float32')

    def call(self, inputs):
        return tf.nn.embedding_lookup(self.embeddings, inputs)

    def compute_mask(self, inputs, mask=None):
        if not self.mask_zero:
            return None
        return tf.not_equal(inputs, 0)

layer = CustomEmbedding(10, 32, mask_zero=True)
x = np.random.random((3, 10)) * 9
x = x.astype('int32')

y = layer(x)
mask = layer.compute_mask(x)

print(mask)
```

```
tf.Tensor(
[[ True False  True  True  True False  True  True  True  True]
 [ True  True  True False  True  True  True False  True False]
 [ True  True  True  True  True  True  True  True  True  True]], shape=(3, 10),
dtype=bool)
```

直接用網路層實現遮罩功能：call函數中接受一個mask參數，並用它來確定是否跳過某些時間步。

要編寫這樣的層，只需在call函數中添加一個mask=None參數即可。只要有可用的輸入，與輸入關聯的遮罩將被傳遞到圖層。

In [11]:
```
class MaskConsumer(tf.keras.layers.Layer):

    def call(self, inputs, mask=None):
        pass
```

In [ ]:
```
```