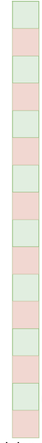
```python
def masked_softmax(X, valid_lens):  #@save

    def _sequence_mask(
            X,
            valid_len,
            value=0
        ):
        maxlen = X.size(1)
```

$\text{maxlen} = \text{length}_k$



$(B*h*\text{length}_q, \text{length}_k)(16,2)$ $(1,2)$ $(16,1)$

$\text{mask} =$ $<$

torch.arange(maxlen)[None,:] valid_lens[:,None]

$(1, \text{length}_k)$ $(B*h*\text{length}_q, 1)$

```python
        mask = torch.arange(
            (maxlen),
            dtype=torch.float32,
            device=X.device
        )[None, :] < valid_len[:, None]

        X[~mask] = value

        return X
```

let ~mask position pad by 0

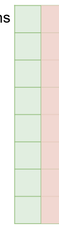
$(B*h*\text{length}_q, \text{length}_k)(16,2)$

```python
    if valid_lens is None:
        return nn.functional.softmax(X, dim=-1)
    else:
        shape = X.shape
```

$\text{shape} = (B*h,\ \text{length}_q,\ \text{length}_k)\ \ (8,2,2)$

$(B*h, \text{length}_q)$ $(8,2)$

valid_lens

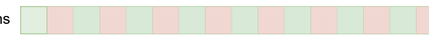


```python
        if valid_lens.dim() == 1:
            valid_lens = torch.repeat_interleave(
                valid_lens, shape[1]
            )
        else:
            valid_lens = valid_lens.reshape(-1)
```

reshape(-1)

$(B*h*\text{length}_q,)\ (16,)$

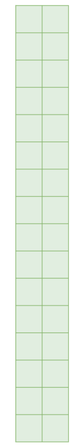
valid_lens



```python
        X = _sequence_mask(
            X.reshape(-1, shape[-1]),
            valid_lens,
            value=-1e6
        )
```

$(B*h*\ \text{length}_q,\ \text{length}_k)\,(16,2)$

X



```python
        return nn.functional.softmax(
            X.reshape(shape), dim=-1
        )
```

$(B*h,\ \text{length}_q,\ \text{length}_k)\,(8,2,2)$

X

softmax to the last dim (length_k)

```python
class DotProductAttention(nn.Module):  #@save
    def __init__(self, dropout):
        super().__init__()
        self.dropout = nn.Dropout(dropout)


    def forward(self, queries, keys, values, valid_lens=None):
```

queries=keys=values=X $d = d_{model}/h$

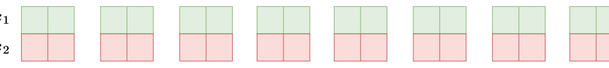
```python
        d = queries.shape[-1]
```

$(B*h,\ \text{length}, d_{model}/h)\,(8,2,2)$

X

x_1

x_2



$(B*h,\ \text{length}_q,\ \text{length}_k)\,(8,2,2)$

```python
        scores = torch.bmm(
            queries,
            keys.transpose(1, 2)
        ) / math.sqrt(d)
```

queries keys.transpose(1,2)

$\text{scores} = \dfrac{(8,\ \ ) \times (8,\ \ )}{\sqrt{d}}$

$=$


```python
        self.attention_weights = masked_softmax(
            scores,
            valid_lens
        )
```

the dim is the same as scores,
we do the mask at length_k dim

```python
        return torch.bmm(
            self.dropout(self.attention_weights),
            values
        )
```

$(B*h,\ \text{length}_q,\ \text{length}_k)\ \ (B*h,\ \text{length}_k, d_{model}/h)$

$\text{out} = \dfrac{(8,\ \ ) \times (8,\ \ )}{\sqrt{d}}$

$(B*h,\ \text{length}_q, d_{model}/h)(8,2,2)$