Attention & 电力负荷预测

2021.10.13

张文韬

jasonwtz@shu.edu.cn

北京大学
PEKING UNIVERSITY

目录》
CONTENTS

01
PART

引言

Attention

Attention模型表述：

当人在看一样东西的时候，我们当前时刻关注的一定是我们当前正在看的这样东西的某一地方，换句话说，当我们目光移到别处时，注意力随着目光的移动也在转移。

这意味着，当人们注意到某个目标或某个场景时，该目标内部以及该场景内每一处空间位置上的注意力分布是不一样的。当我们试图描述一件事情，我们当前时刻说到的单词和句子和正在描述的该事情的对应某个片段最相关，而其他部分随着描述的进行，相关性也在不断地改变。

Attention机制：是**一种能让模型对重要信息重点关注并充分学习吸收的技术**，它不算是一个完整的模型，而是是一种技术，能够作用于任何序列模型中。

Attention

为什么要加入Attention：

在Seq2seq模型中，原始编解码模型的encode过程会生成一个中间向量C，用于保存原序列的语义信息。但是这个向量长度是固定的，当输入原序列的长度比较长时，向量C无法保存全部的语义信息，上下文语义信息受到了限制，这也限制了模型的理解能力。所以使用Attention机制来打破这种原始编解码模型对固定向量的限制。

> “一个潜在的问题是，采用编码器-解码器结构的神经网络模型需要将输入序列中的必要信息表示为一个固定长度的向量，而当输入序列很长时则难以保留全部的必要信息（因为太多），尤其是当输入序列的长度比训练数据集中的更长时。”
> — Dzmitry Bahdanau, et al., Neural machine translation by jointly learning to align and translate, 2015

Attention

Treisman 和 Geldade 提出注意力机制，它可以模拟人脑给不同的输入赋予不同的权重，权重的大小可以突出输入对输出的影响大小。

Bahdanau 等最先将注意力机制应用于机器翻译，输入的隐藏状态序列上的注意力信号由多层感知器通过解码器的最后一个隐藏状态来确定，通过在每个解码步骤中可视化输入序列上的注意力信号，可以清晰地对翻译语句和目标语句进行校准。

Luong等构造了基于全局注意力机制和局部注意力机制的两种神经网络模型，注意力机制可以解释输入句子和输出句子之间的对齐关系，解释模型到底学到了什么知识，打开了深度学习的黑箱

Parikh等基于注意力机制，将几个句子直接的关系转换成几个句子中关键词之间的关系，减少了需要学习的参数，提高了模型的训练效率

Li等表示尽管神经网络已经较好地应用到自然语言处理中，但是却缺乏解释力，而加入注意力机制的模型不仅可以解释模型的决策，而且可以辅助分析模型的误差

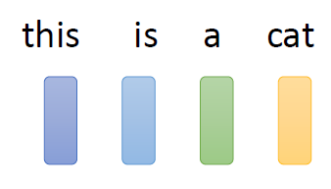
Golub等引入注意力机制，改进了之前基于Word-level的Encoder-Decoder中存在的训练参数较多的问题

02
PART

Seq2seq

Seq2Seq

A



One-hot Encoding

apple = [1 0 0 0 0 .....]

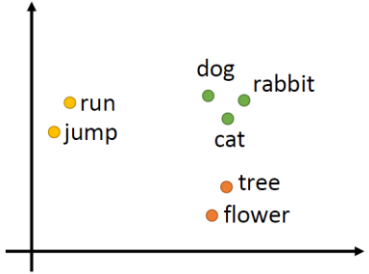
bag = [0 1 0 0 0 .....]

cat = [0 0 1 0 0 .....]

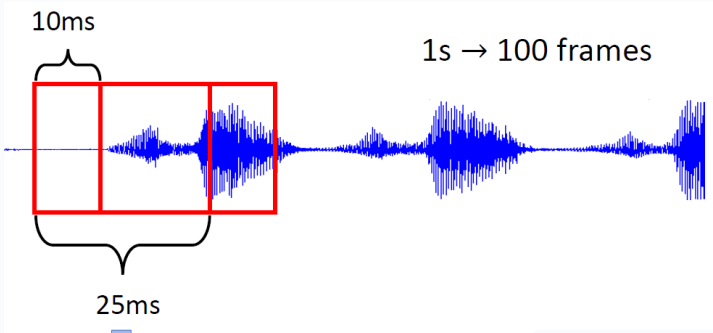
dog = [0 0 0 1 0 .....]

elephant = [0 0 0 0 1 .....]

Word Embedding



B



frame

400 sample points (16KHz)

39-dim MFCC

80-dim filter bank output

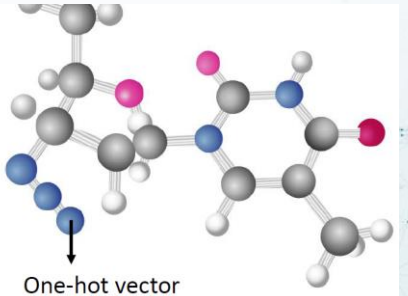
C

D

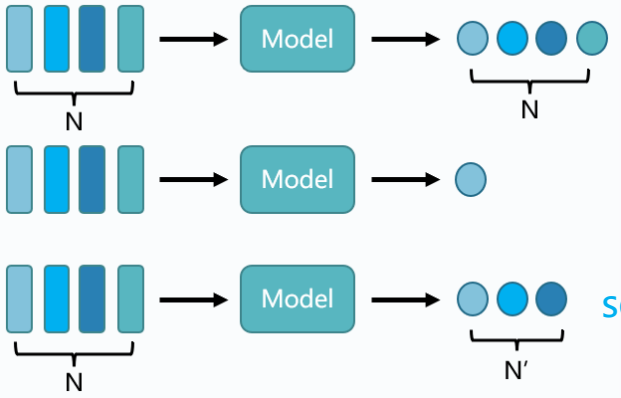
H = [1 0 0 0 0 .....]

C = [0 1 0 0 0 .....]

O = [0 0 1 0 0 .....]

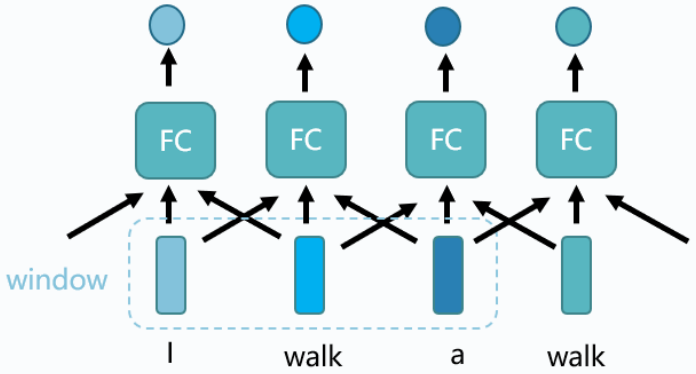


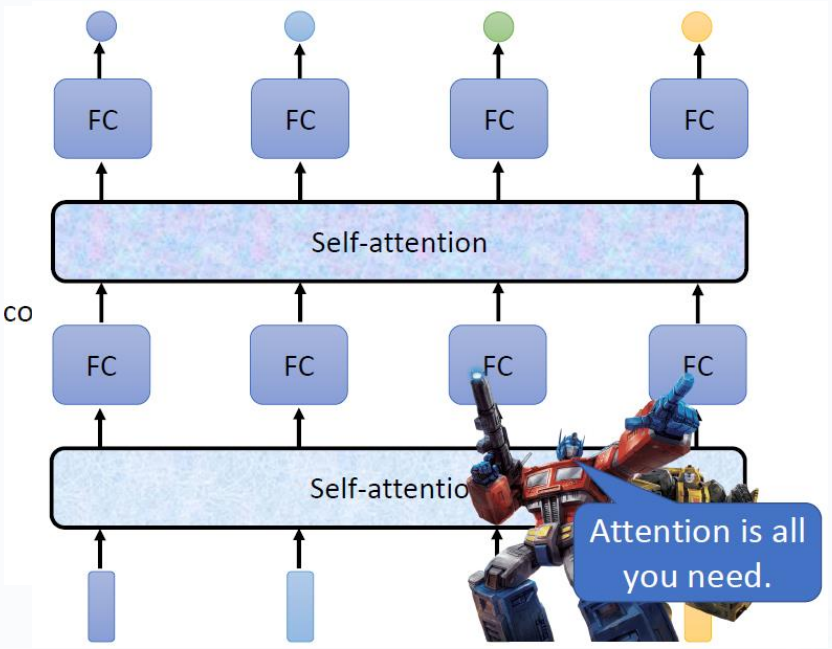
Seq2Seq



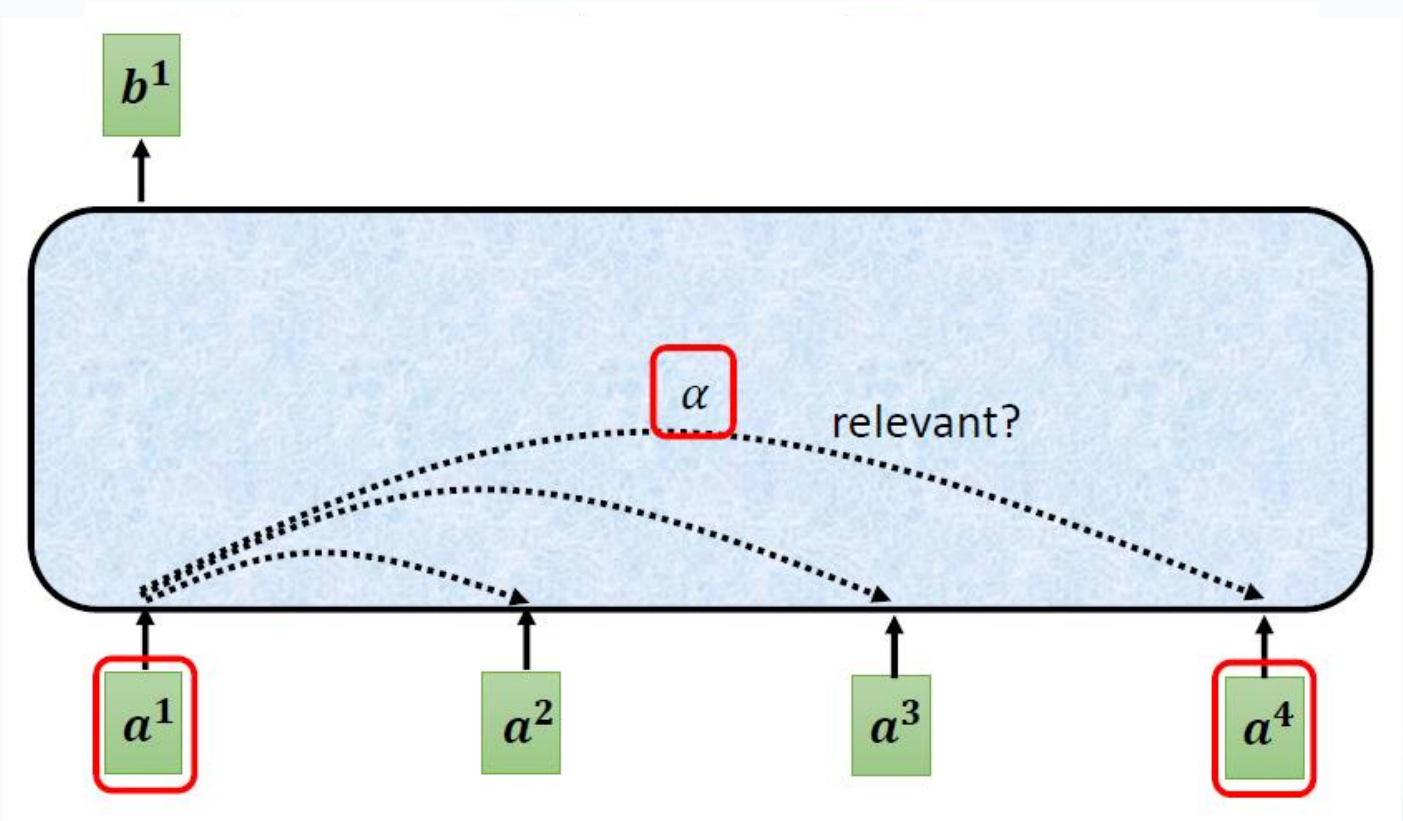
seq2seq

Sequence Labeling





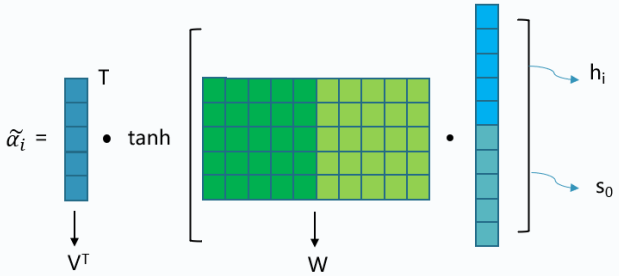
Seq2Seq



Self-Attention

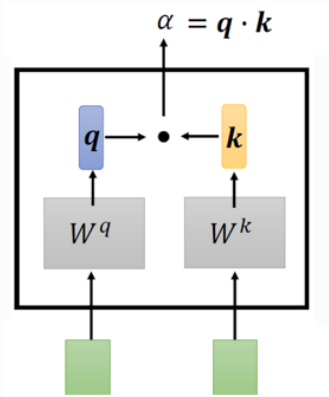
权重函数计算

1. Bilinear方法



$$a(q, k) = q^T W k$$

2. Dot-product



$$a(q, k) = q^T k$$

$$k_i = \mathrm{Wk} \bullet hi$$
$$q_0 = \mathrm{Wq} \bullet s_0$$

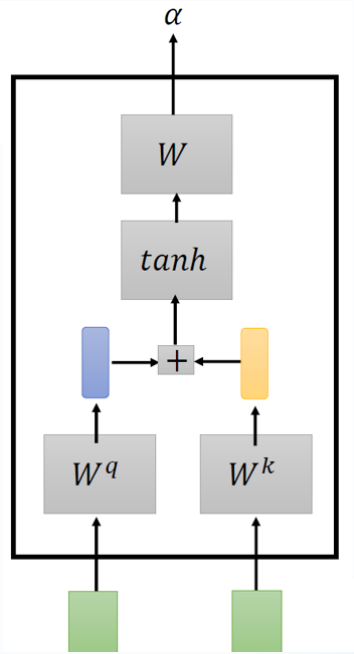
$$\widetilde{\alpha_i} = \mathrm{ki}^{\mathrm{T}} \bullet q_0$$

$$\alpha_i, \ldots, \alpha_m = \mathrm{Softmax}([\widetilde{\alpha_i}, \ldots, \widetilde{\alpha_m}])$$

3. 多层感知机方法

Additive

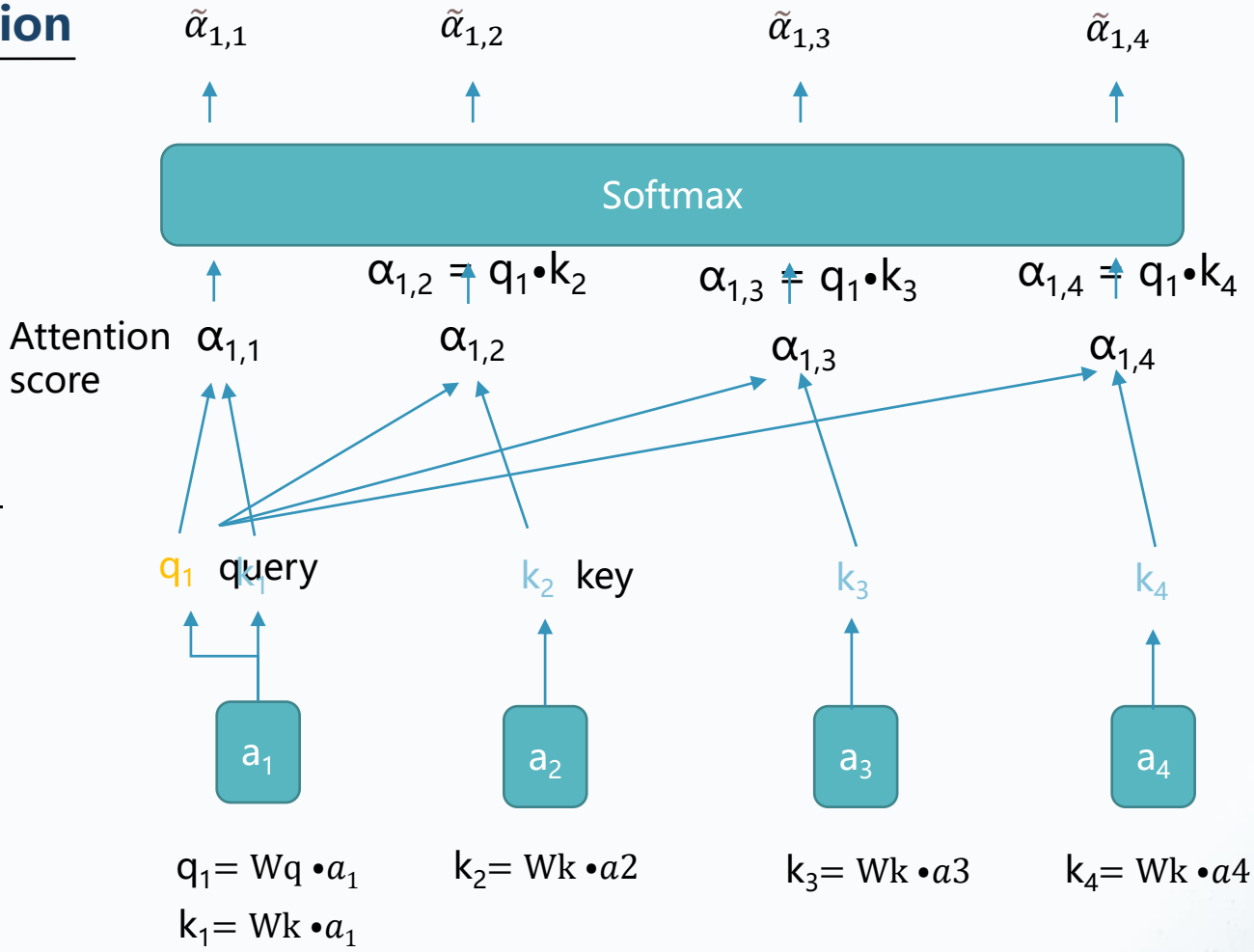
$$a(q, k) = w_2^T \tanh(W_1[q; k])$$



Self-Attention

α 计算

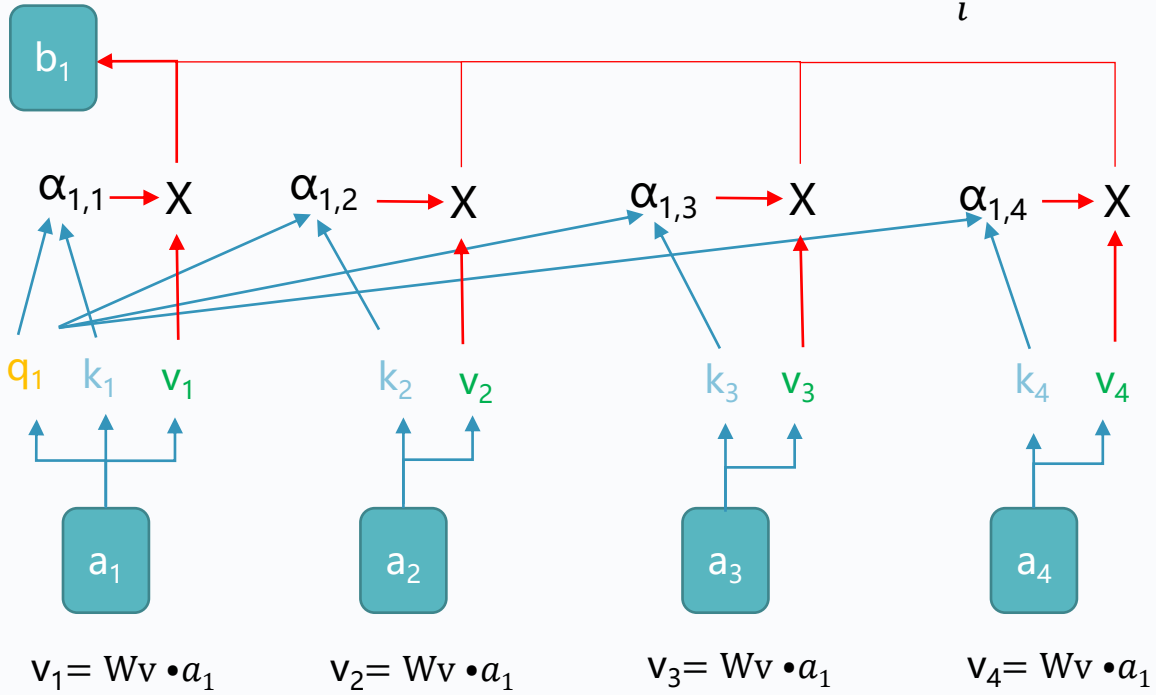
$$\tilde{\alpha}_{1,i} = \frac{\exp(\alpha_{1,i})}{\sum \exp(\alpha_{1,j})}$$



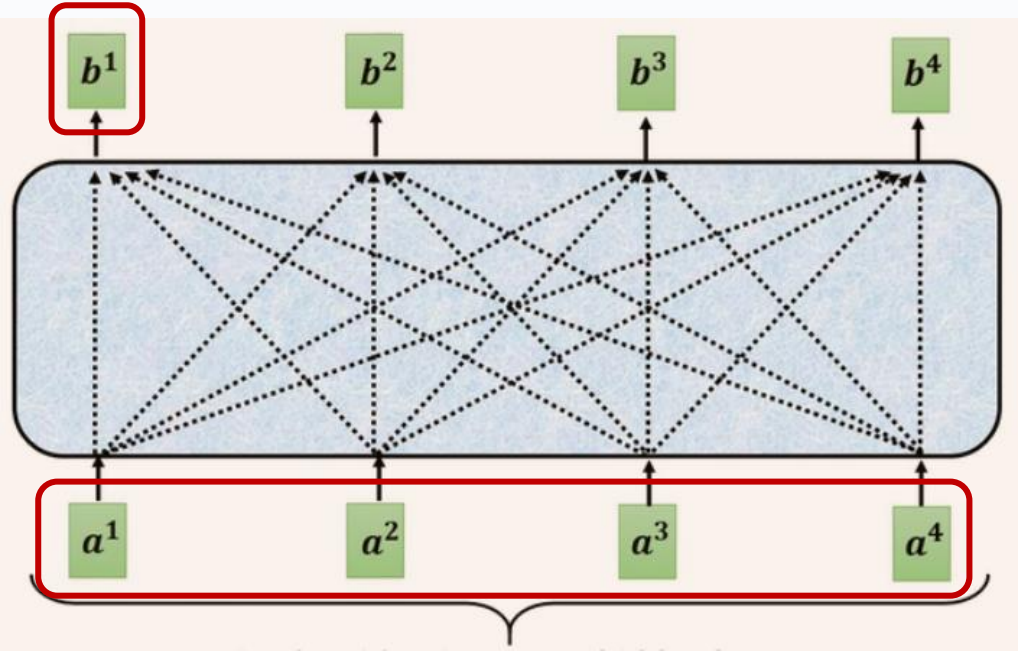
Self-Attention

q , k 计算

$$b^1 = \sum_i \alpha'_{1,i} v^i$$



Self-Attention



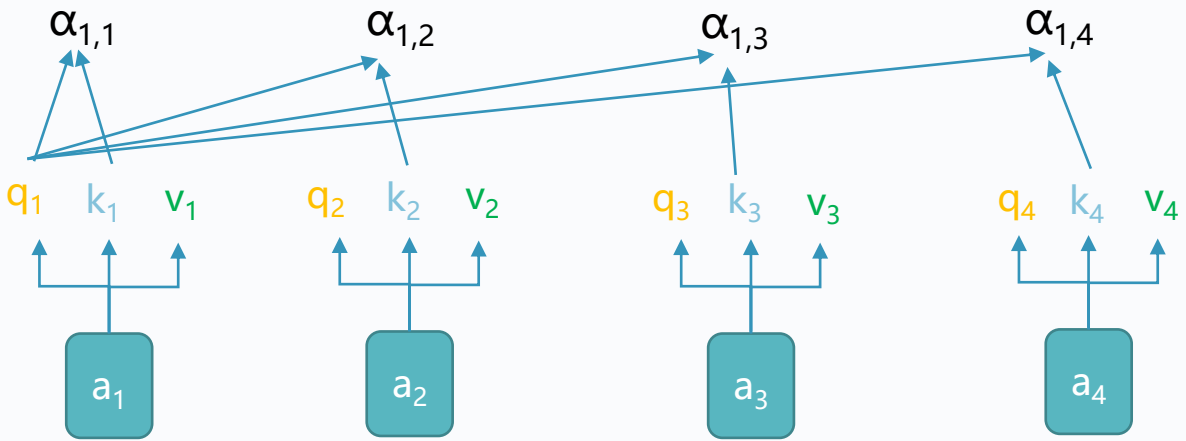
Self-Attention

数据传递

$$\alpha_{1,1} = k_1 \cdot q_1$$
$$\alpha_{1,2} = k_2 \cdot q_1$$
$$\alpha_{1,3} = k_3 \cdot q_1$$
$$\alpha_{1,4} = k_4 \cdot q_1$$

$$q_i = Wq \cdot ai$$
$$k_i = Wk \cdot ai$$
$$v_i = Wv \cdot ai$$

$$q^1 q^2 q^3 q^4 = W^q \, a^1 a^2 a^3 a^4$$
$$Q \qquad\qquad I$$
$$k^1 k^2 k^3 k^4 = W^k \, a^1 a^2 a^3 a^4$$
$$K \qquad\qquad I$$
$$v^1 v^2 v^3 v^4 = W^v \, a^1 a^2 a^3 a^4$$
$$V \qquad\qquad I$$



Self-Attention

$$\alpha_{1,1} = k_1 \cdot q_1$$
$$\alpha_{1,2} = k_2 \cdot q_1$$
$$\alpha_{1,3} = k_3 \cdot q_1$$
$$\alpha_{1,4} = k_4 \cdot q_1$$

$$\begin{matrix}\alpha_{1,1}\\ \alpha_{1,2}\\ \alpha_{1,3}\\ \alpha_{1,4}\end{matrix} = \begin{matrix}k_1\\ k_2\\ k_3\\ k_4\end{matrix} \cdot q_1$$

$$\begin{matrix}\alpha'_{1,1} & \alpha'_{2,1} & \alpha'_{3,1} & \alpha'_{4,1}\\ \alpha'_{1,2} & \alpha'_{2,2} & \alpha'_{3,2} & \alpha'_{4,2}\\ \alpha'_{1,3} & \alpha'_{2,3} & \alpha'_{3,3} & \alpha'_{4,3}\\ \alpha'_{1,4} & \alpha'_{2,4} & \alpha'_{3,4} & \alpha'_{4,4}\end{matrix}$$

A'

← Softmax

$$\begin{matrix}\alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} & \alpha_{4,1}\\ \alpha_{1,2} & \alpha_{2,2} & \alpha_{3,2} & \alpha_{4,2}\\ \alpha_{1,3} & \alpha_{2,3} & \alpha_{3,3} & \alpha_{4,3}\\ \alpha_{1,4} & \alpha_{2,4} & \alpha_{3,4} & \alpha_{4,4}\end{matrix}$$

A

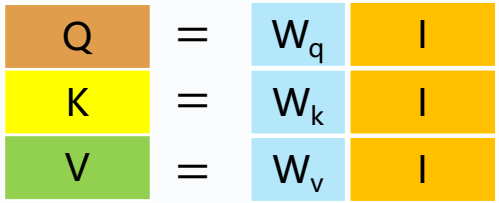
$$= \begin{matrix}k_1\\ k_2\\ k_3\\ k_4\end{matrix} \cdot \begin{matrix}q_1 & q_2 & q_3 & q_4\end{matrix}$$

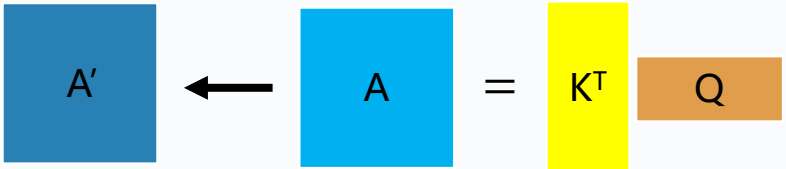
K^T $\quad$ Q

$$\begin{matrix}b_1 & b_2 & b_3 & b_4\end{matrix} = \begin{matrix}v_1 & v_2 & v_3 & v_4\end{matrix} \cdot \begin{matrix}\alpha'_{1,1} & \alpha'_{2,1} & \alpha'_{3,1} & \alpha'_{4,1}\\ \alpha'_{1,2} & \alpha'_{2,2} & \alpha'_{3,2} & \alpha'_{4,2}\\ \alpha'_{1,3} & \alpha'_{2,3} & \alpha'_{3,3} & \alpha'_{4,3}\\ \alpha'_{1,4} & \alpha'_{2,4} & \alpha'_{3,4} & \alpha'_{4,4}\end{matrix}$$

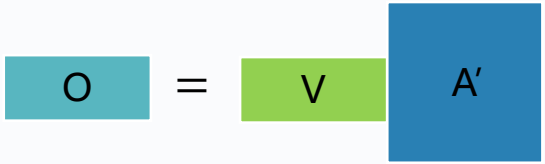
O $\quad$ V $\quad$ A'

Self-Attention

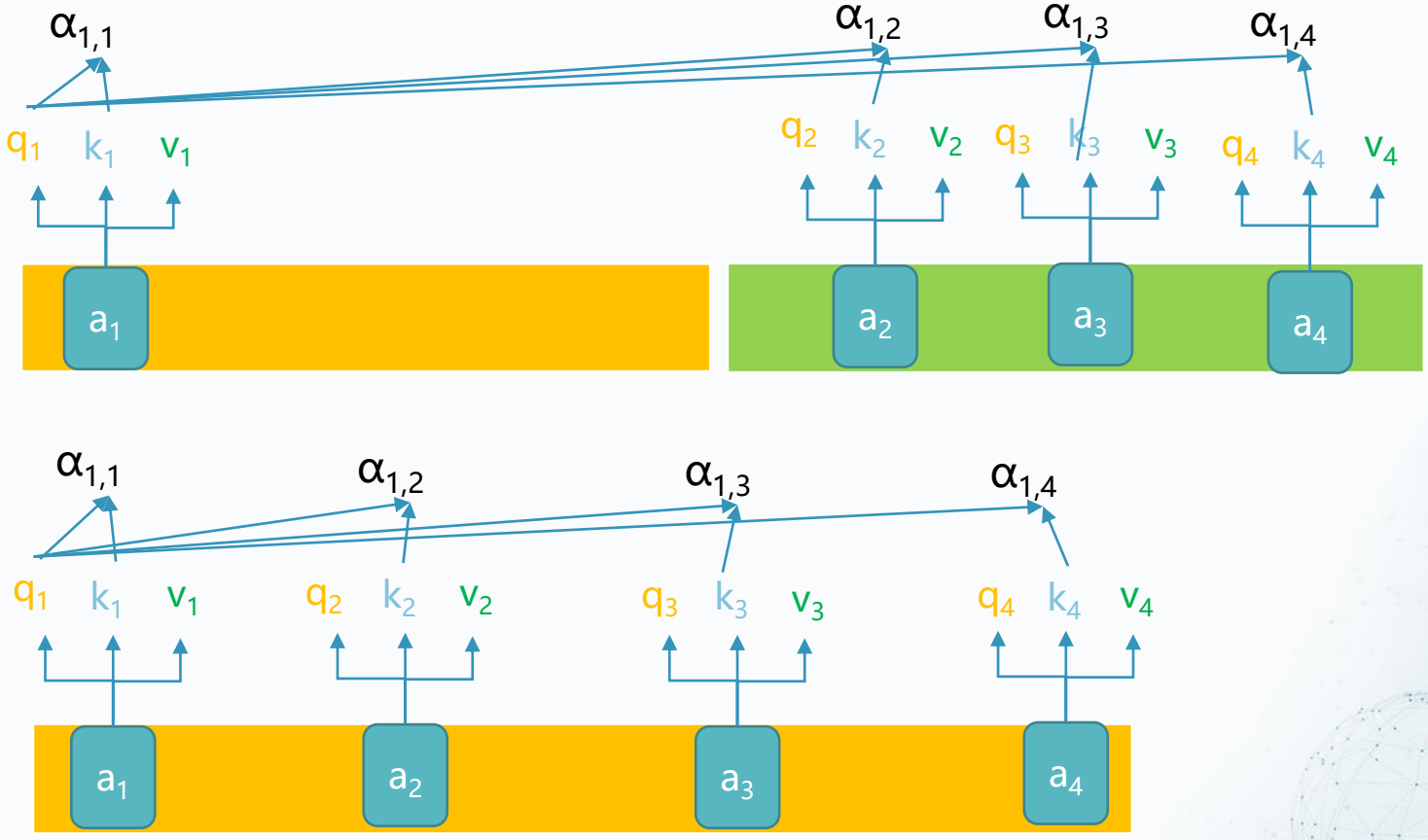




Attention matrix



Self-Attention



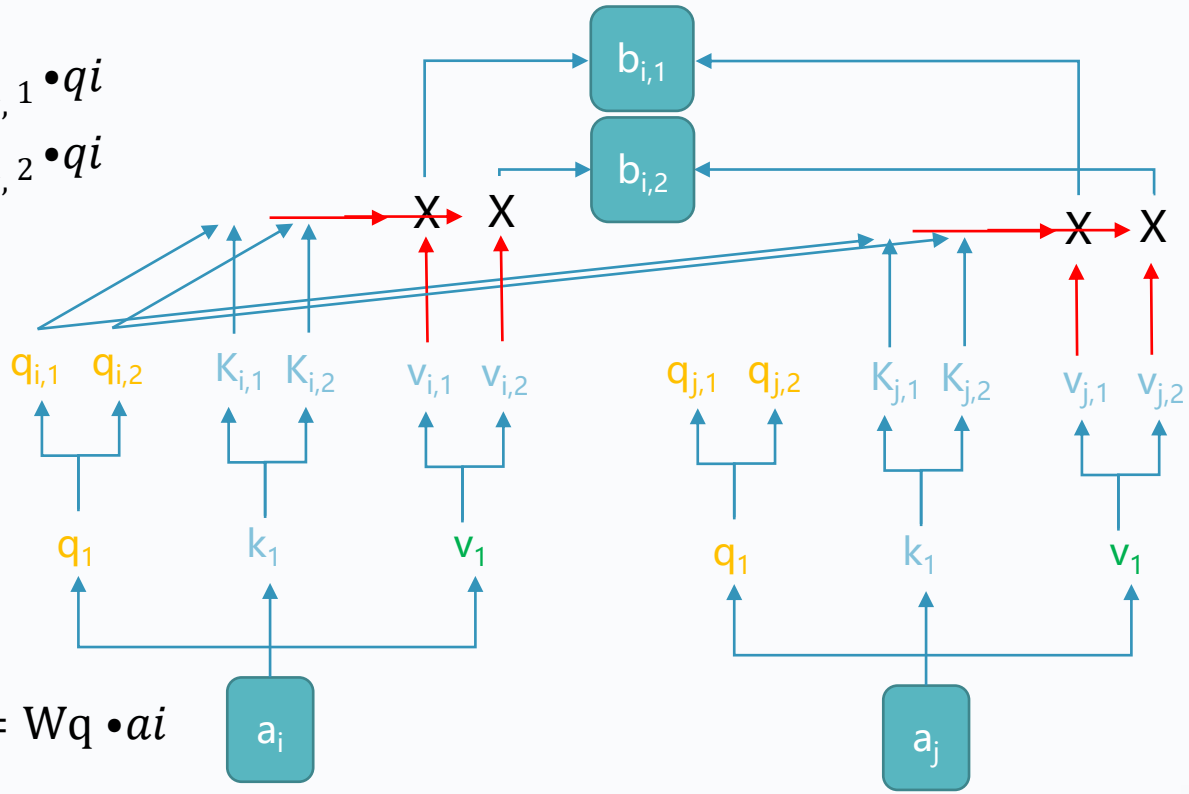
Self-Attention

$$b_i = W_0 \bullet \begin{matrix} b_{i,1} \\ b_{i,2} \end{matrix}$$

Multi-head Self-attention

$$q_{i,1} = Wq_{,1} \bullet qi$$

$$q_{i,2} = Wq_{,2} \bullet qi$$

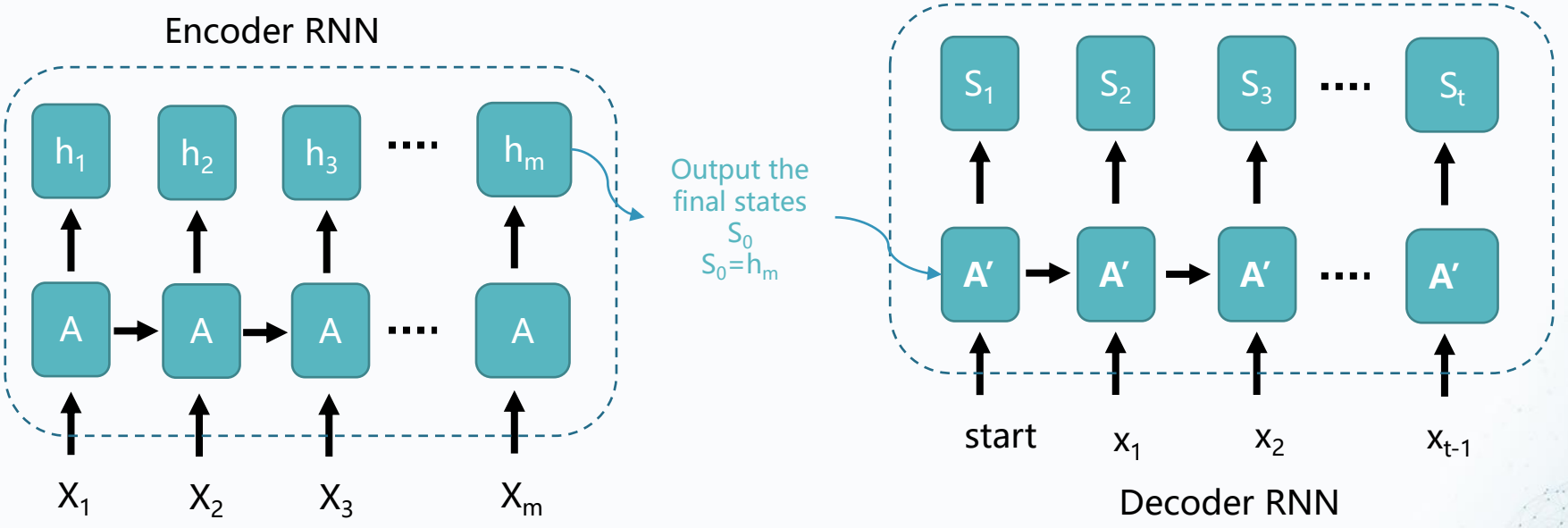


$$q_i = Wq \bullet ai$$

Self-Attention Attention & RNN

Attention & RNN

Seq2seq



Attention & RNN

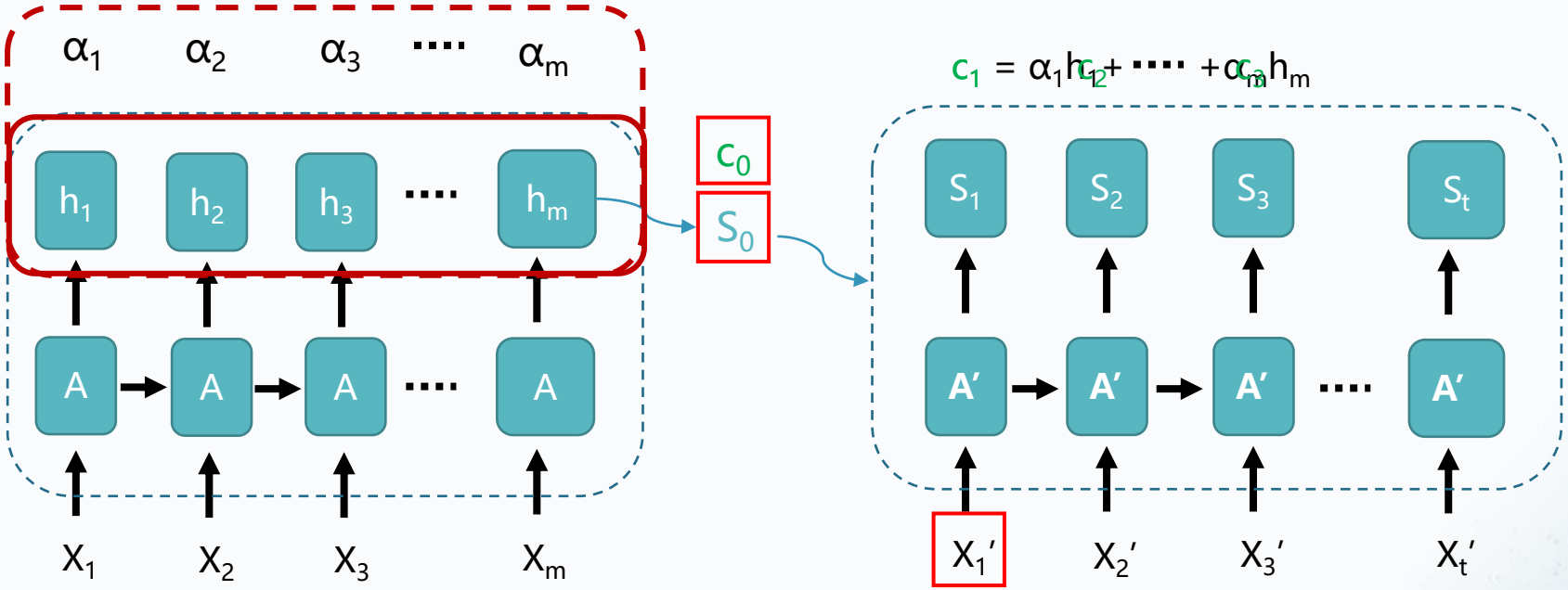
Seq2seq model with attention

Weight: $\alpha_i = \text{align}(h_i, s_0)$

Context vector: $c_0 = \alpha_1 h_1 + \cdots + \alpha_m h_m$

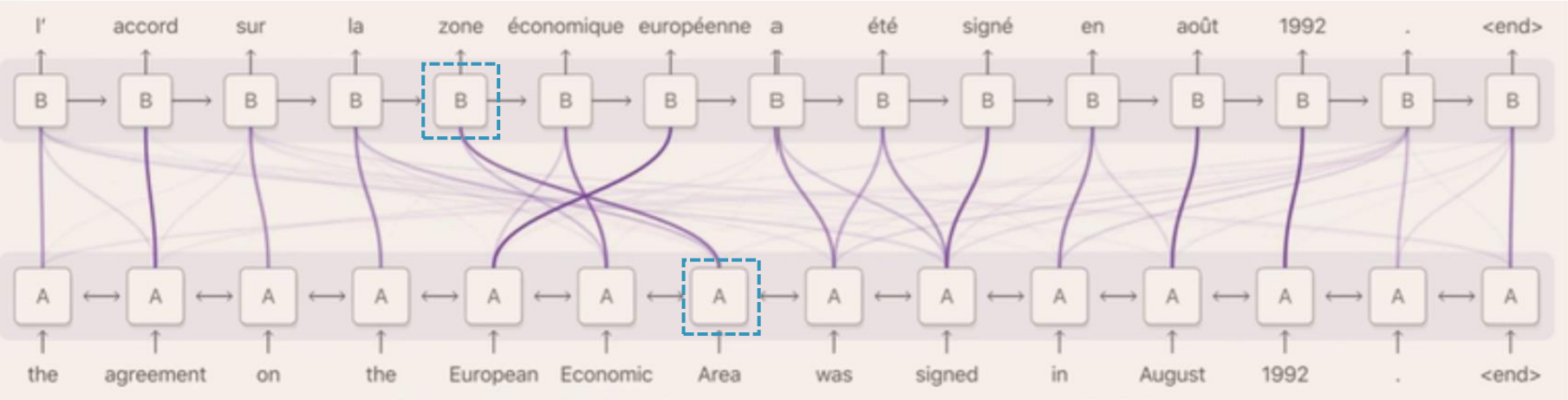
SimpleRNN: $\mathbf{s}_1 = \tanh\left(\mathbf{A}' \cdot \begin{bmatrix}\mathbf{x}_1' \\ \mathbf{s}_0\end{bmatrix} + \mathbf{b}\right)$

SimpleRNN + Attention : $\mathbf{s}_1 = \tanh\left(\mathbf{A}' \cdot \begin{bmatrix}\mathbf{x}_1' \\ \mathbf{s}_0 \\ \mathbf{c}_0\end{bmatrix} + \mathbf{b}\right)$



Attention & RNN

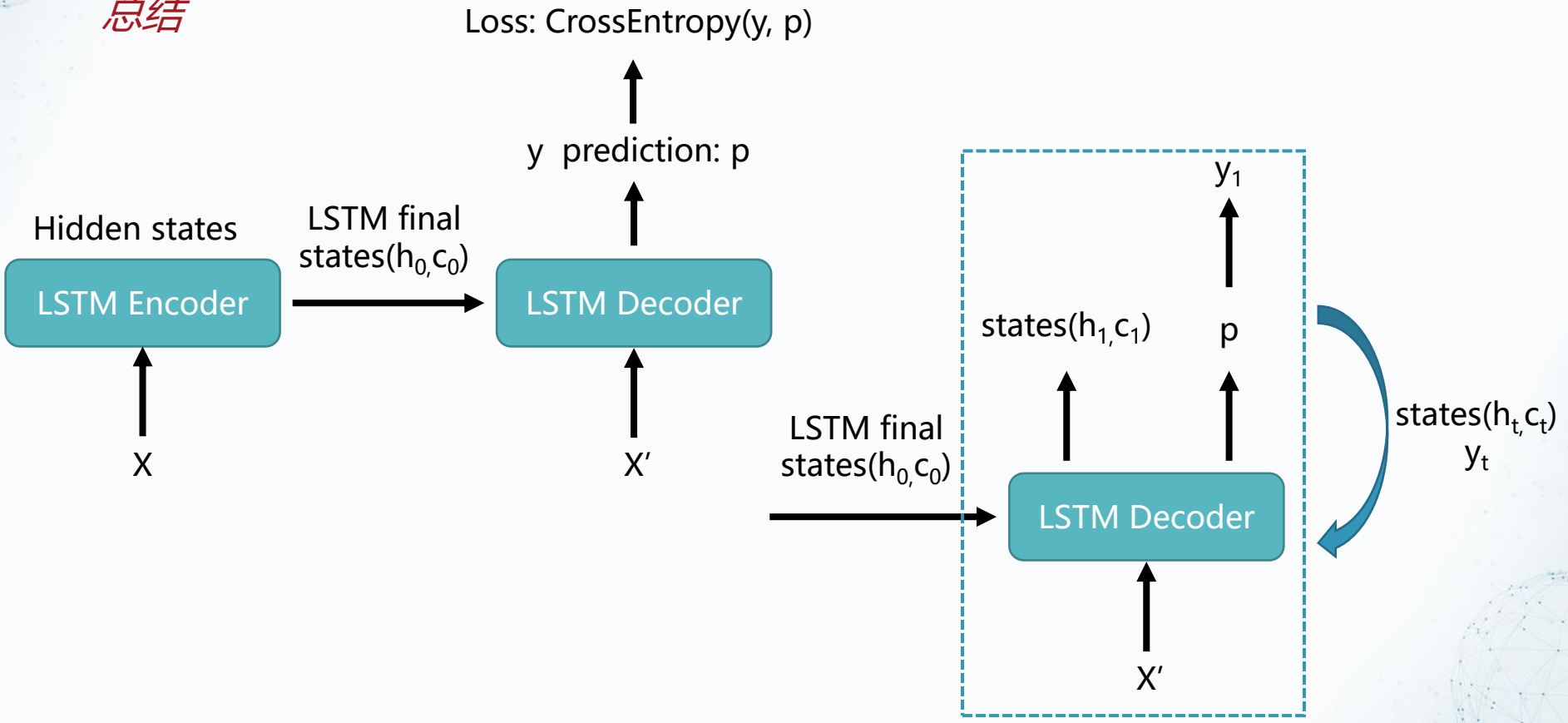
Decoder RNN (target labguage)



Encoder RNN (source language)

Attention & RNN

总结

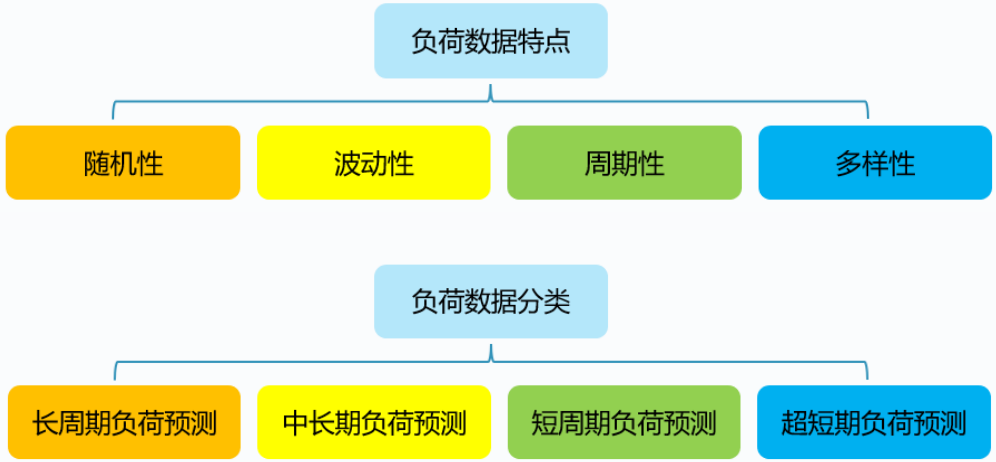


04
PART

电力负荷预测

背景&理论基础

电力负荷预测对于保证电网安全、高效、经济地运行十分重要。由于电力负荷数据受到经济、气候等很多因素的影响，导致电力负荷变化难以预估，如果想要精确的对电力负荷进行预测，就需要对这些影响因素进行分析研究，发现它们之间的潜在关系，才能取得较好的预测效果。



电力负荷预测流程

先需要确定预测的对象，如前文所述不同国家、不同地区的负荷数据具有差异性，需要具体情况具体分析，所以要选定合适的预测目标

预测目标

收集所选择地区的电力负荷数据

数据收集

对所选数据集进行分析，如果是多维度特征的数据集，则还需要将多种特征结合采集地的实际情况进行分析，发掘这些特征与负荷之间的关系

数据分析

需要使用多种模型在所选定的数据集上进行建模，寻找对数据亲和力较好的模型

建模调优

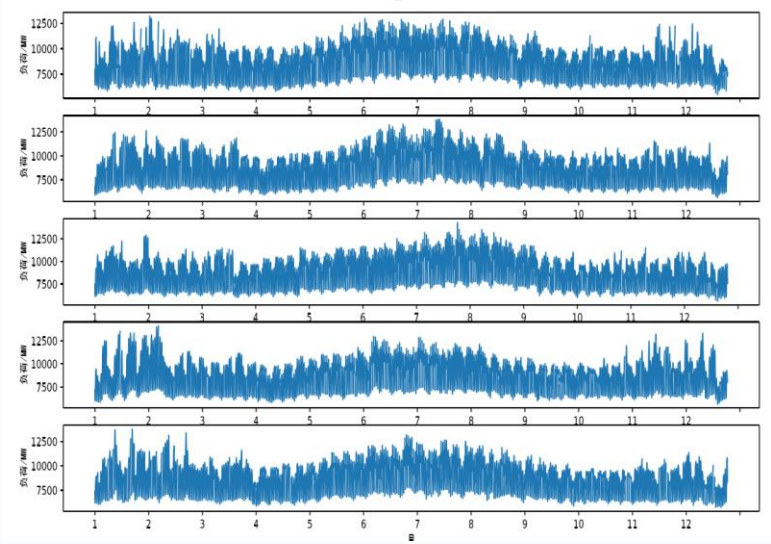
建模调优后就可以使用模型对负荷数据进行预测，并对预测结果进行分析

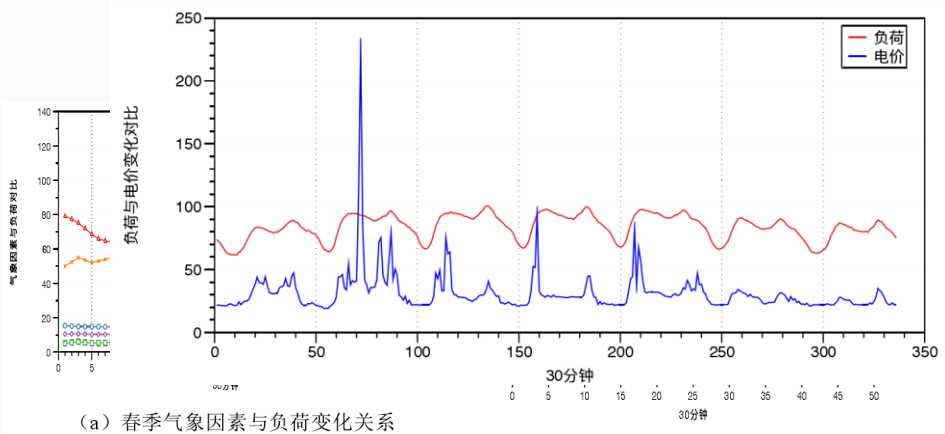
预测目标

电力负荷影响因素分析

电力负荷数据变化受多种因素共同作用，影响负荷的因素分为周期规律性因素与随机因素。

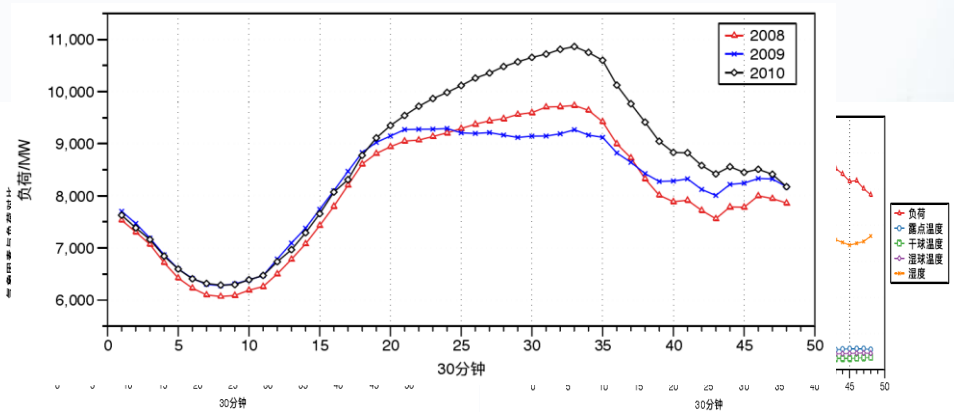
随机因素主要包含气象、经济、自然灾害等，这些因素都具有一定的不可预见性，它们会使负荷数据产生异常波动，因此这些因素也是影响负荷预测精度的主要原因。





（a）春季气象因素与负荷变化关系

（b）夏季气象因素与负荷变化关系



（c）秋季气象因素与负荷变化关系

（d）冬季气象因素与负荷变化关系

05
PART

代码实战

代码

```python
def attention_3d_block(inputs):
    input_dim = int(inputs.shape[2])
    a = inputs
    a = Dense(input_dim, activation='softmax')(a)
    if SINGLE_ATTENTION_VECTOR:
        a = Lambda(lambda x: K.mean(x, axis=1), name='dim_reduction')(a)
        a = RepeatVector(input_dim)(a)
    a_probs = Permute((1, 2), name='attention_vec')(a)

    output_attention_mul = merge.multiply([inputs, a_probs])
    return output_attention_mul
```

```python
def attention_model():
    inputs = Input(shape=(TIME_STEPS, INPUT_DIMS))

    lstm_out = Bidirectional(LSTM(lstm_units, return_sequences=True, input_shape=(train_x.shape[1], 1)))(inputs)
    lstm_out = Dropout(0.3)(lstm_out)

    attention_mul = attention_3d_block(lstm_out)
    attention_mul = Flatten()(attention_mul)

    output = Dense(1, activation='sigmoid')(attention_mul)
    model = Model(inputs=[inputs], outputs=output)
    return model
```

代码

```python
def normalization(data, min, max):
    range = max - min
    m = data.shape[0]
    normData = data - np.tile(min, (1, m))
    normData = normData / np.tile(range, (1, m))
    return normData
```

```python
def create_new_dataset(dataset, seq_len):
    X = []  # 初始特征数据集为空列表
    y = []  # 初始标签数据集为空列表

    start = 0  # 初始位置
    end = dataset.shape[0] - seq_len  # 截止位置

    for i in range(start, end):  # for循环构造特征数据集
        sample = dataset[i: i + seq_len]  # 基于时间跨度seq_len创建样本
        label = dataset[i + seq_len]  # 创建sample对应的标签
        X.append(sample)  # 保存sample
        y.append(label)  # 保存label

    # 返回特征数据集和标签集
    return np.array(X), np.array(y)
```

代码

```
# 加载数据
data = pd.read_csv("./MT1_2.csv", usecols=[0], engine='python')
data = data.values

len_data = len(data)
INPUT_DIMS = 1
TIME_STEPS = 24
lstm_units = 128
train_data_len = int(len_data * 0.8)

# 数据集划分
train_set = data[: train_data_len]  # 训练集
test_set = data[train_data_len - TIME_STEPS:]
```

```
# 数据集划分
train_set = data[: train_data_len]  # 训练集
test_set = data[train_data_len - TIME_STEPS:]

# 归一化
scaler = MinMaxScaler(feature_range=(0, 1))
train_norm_1 = scaler.fit_transform(train_set)
max = scaler.data_max_
min = scaler.data_min_

test_norm_1 = copy.deepcopy(test_set)
for i in range(len(test_set)):
    tmp = test_norm_1[i]
    test_norm_1[i] = normalization(tmp, min, max)

train_x, train_y = create_new_dataset(train_norm_1, TIME_STEPS)
test_x, test_y = create_new_dataset(test_norm_1, TIME_STEPS)
test_x_, test_y_ = create_new_dataset(test_set, TIME_STEPS)
```

代码

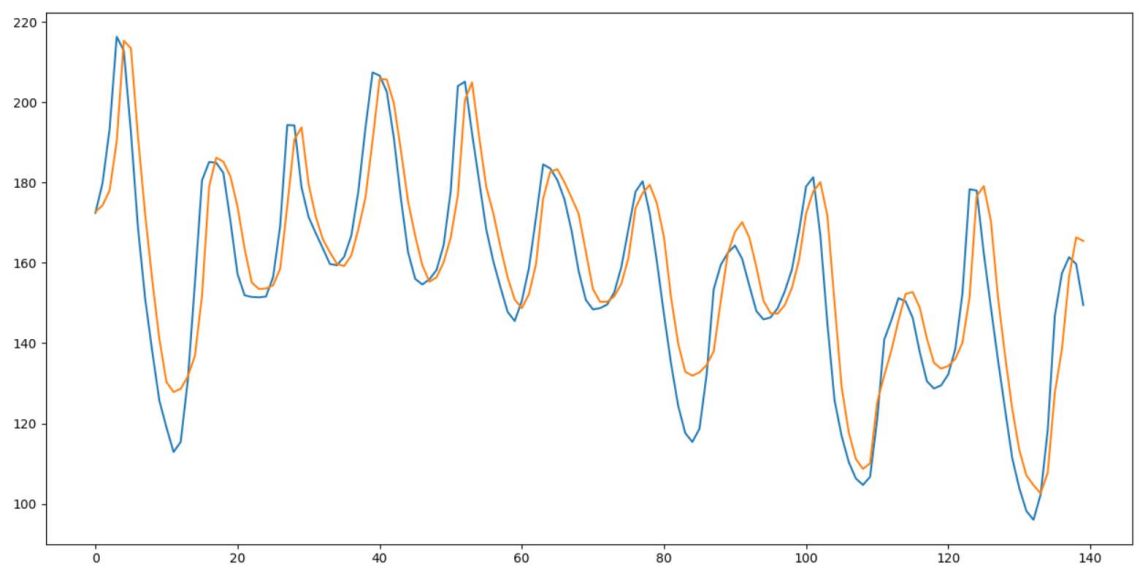
```
# 训练&预测
model = attention_model()
model.compile(optimizer='adam', loss='mse')
model.fit(train_x, train_y, epochs=150, batch_size=64, validation_split=0.1)
test_pred_2 = model.predict(test_x)
```

```
# 反归一化
true_predictions = scaler.inverse_transform(np.array(test_pred_2).reshape(-1, 1))

# 指标数据
score_2 = r2_score(test_y_, true_predictions)
MAE_2 = mean_absolute_error(test_y_, true_predictions)
MSE_2 = mean_squared_error(test_y_, true_predictions)
RMSE_2 = math.sqrt(MSE_2)
print('MAE_1: ', MAE_2)
print('MSE_1:', MSE_2)
print('RMSE_1:', RMSE_2)
print("r^2_1 的值:", score_2)
```

```
# 绘制图形
len_data = len(data)
x1 = np.arange(0, len_data - train_data_len, 1)

plt.figure(2)
plt.plot(x1, test_y_)
plt.plot(x1, true_predictions)
plt.show()
```


220
200
180
160
140
120
100
0
20
40
60
80
100
120
140

THANKS

谢谢