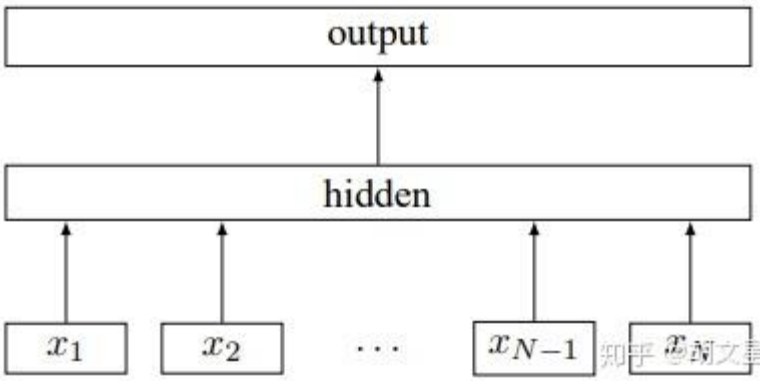
FastText



简单粗暴的FastText

0.用哈希算法将 2-gram、3-gram 信息分别映射到两张表内。

1.模型输入： **[batch_size, seq_len]**

2.embedding 层：随机初始化，词向量维度为 embed_size，2-gram 和 3-gram 同理：

word： **[batch_size, seq_len, embed_size]**

2-gram：**[batch_size, seq_len, embed_size]**

3-gram：**[batch_size, seq_len, embed_size]**

3.拼接 embedding 层：

[batch_size, seq_len, embed_size * 3]

4.求所有 seq_len 个词的均值

[batch_size, embed_size * 3]

5.全连接+非线性激活：隐层大小 hidden_size

[batch_size, hidden_size]

6.全连接+softmax 归一化：

[batch_size, num_class]==>[batch_size, 1]

分析：

不加 N-Gram 信息，就是词袋模型，准确率 89.59%，加上 2-gram 和 3-gram 后准确率 92.23%。N-Gram 的哈希映射算法见 **utils_fasttext.py** 中注释------gram-------处。

N-Gram 的词表我设的 25W，相对于前面几个模型，这个模型稍慢一点，FastText 被我搞成 SlowText 了。。？但是效果不错呀。。哈哈

有人私信问我这个 N-Gram，我大体讲一下。对于 N-Gram，我们设定一个词表，词表大小自己定，当然越大效果越好，但是你得结合实际情况，对于 2-Gram，5000 个字(字表大小)的两两组合有多少种你算算，3-Gram 的组合有多少种，组合太多了，词表设大效果是好了，但是机器它受不了啊。

所以 N-Gram 词表大小的设定是要适中的，那么适中的词表就放不下所有 N-Gram 了，不同的 N-Gram 用哈希算法可能会映射到词表同一位置，这确实是个弊端，但是问题不大：5000 个字不可能每两个字都两两组合出现，很多两个字是永远不会组成 2-Gram 的，所以真正出现的 N-Gram 不会特别多，映射到同一词表位置的 N-Gram 也就不多

了。

N-Gram 词表大小我定的 25w，比较大了，比小词表训练慢了很多，效果也提升了。这么形容 N-Gram 词表大小对效果的影响吧：一分价钱 1 分货，十分价钱 1.1 分货。