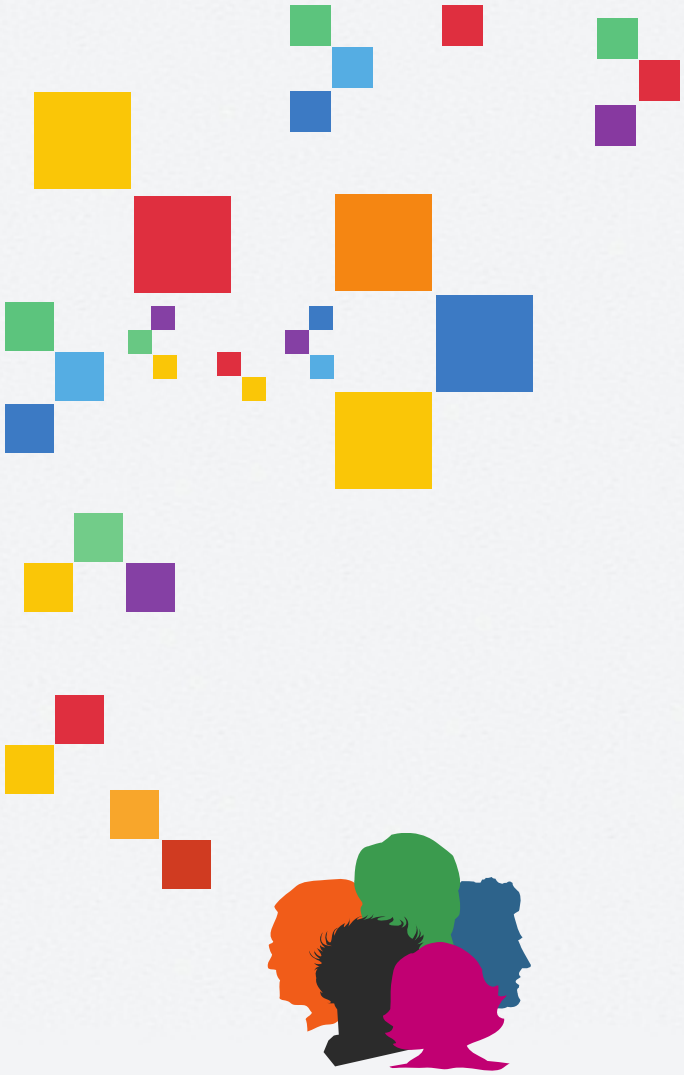
TensorFlow与自然语言处理模型的应用

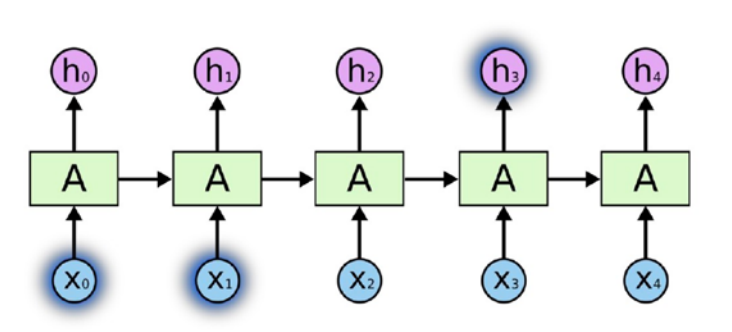
李嘉璇 《TensorFlow技术解析与实战》作者



主题大纲

- 自然语言处理的模型原理和常见模型的演进
- TensorFlow框架下的RNN实践小结
- 生成式聊天机器人的原理及TensorFlow实现

Recurrent Neural Networks



BPTT

- 对于t时刻的误差信号计算如下：

$$\vartheta_j(t) = f_j'(net_j(t)) \sum_i w_{ij}\vartheta_i(t+1).$$

- 这样权值的更新方式如下：

$$\Delta w_{jl} = \alpha \vartheta_j(t) y^l(t-1)$$

- 误差传递信号的关系可以写成如下的递归式：

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \begin{cases} f_v'(net_v(t-1))w_{uv} & q=1 \\ f_v'(net_v(t-q)) \sum_{l=1}^{n} \frac{\partial \vartheta_l(t-q+1)}{\partial \vartheta_u(t)} w_{lv} & q>1 \end{cases}$$

存在问题1

- gradient
  - BPTT学习算法存在梯度爆炸和消失问题(gradient blow up or vanish)，简单通过local error flow分析如下：

$$l_q = v, l_0 = u$$

$$T = \prod_{m=1}^{q} f'_{l_m}(net_{l_m}(t-m))w_{l_m l_{m-1}} = f'_{l_1}(net_{l_1}(t-1))w_{l_1 u} \cdot f'_{l_2}(net_{l_2}(t-2))w_{l_2 l_1} \cdots f'_{v}(net_{v}(t-q))w_{v l_{q-1}}$$

- 整个结果式对T求和的次数是n^(q-1), 即T有n^(q-1)项：
- 如果|T| > 1, 误差就会随着q的增大而呈指数增长，那么网络的参数更新会引起非常大的震荡。
- 如果|T| < 1，误差就会消失，导致学习无效，一般激活函数用simoid函数，它的倒数最大值是0.25, 权值最大值要小于4才能保证不会小于1。
- 误差呈指数增长的现象比较少，误差消失在BPTT中很常见。

存在问题2

- conflict
  - input weight conflict

    假设wji表示输入层到隐层之间的连接，对于有些输入希望尽可能通过，也就是wji比较大，但是另外一些无关的输入可能希望尽可能屏蔽掉，也就是wji尽可能为0。而实际网络中的wji参数是跟输入无关，对于所有的输入，它的大小是一致的。由于缺少这种自动调节功能，从而导致学习比较困难。
  - output weight conflict

    同理，隐层到输出层之间也存在放行和屏蔽的conflict。

TensorFlow的实现

- 特点：
  - Duck typing

```
def _like_rnncell(cell):
  """Checks that a given object is an RNNCell by using duck typing."""
  conditions = [hasattr(cell, "output_size"), hasattr(cell, "state_size"),
                hasattr(cell, "zero_state"), callable(cell)]
  return all(conditions)
```

- 每个`RNNCell`都必须具有以上属性，并使用`call`来实现（output，next_state）= call（input，state）。

Base interface for all RNN Cells

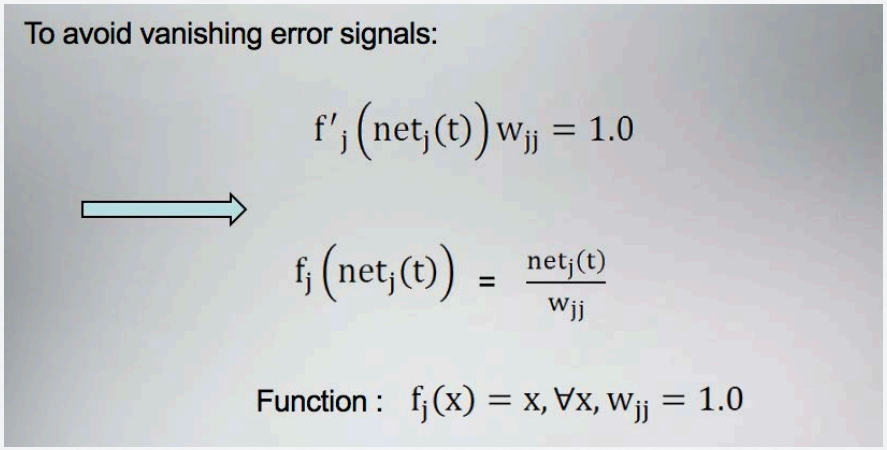
- `tf.contrib.rnn.RNNCell`

Original LSTM

- 1997年Hochreiter和Schmidhuber首先提出了LSTM的网络结构，解决了传统RNN的上面两个问题。
- 问题一的solution:
  - 通过引入CEC(constant error carrousel)单元解决了梯度沿时间尺度unfolding带来的问题。
  - 首先梯度的递推关系如下：

$$\vartheta_j(t) = f_j'(net_j(t)) \sum_i w_{ij} \vartheta_i(t+1)$$

Original LSTM

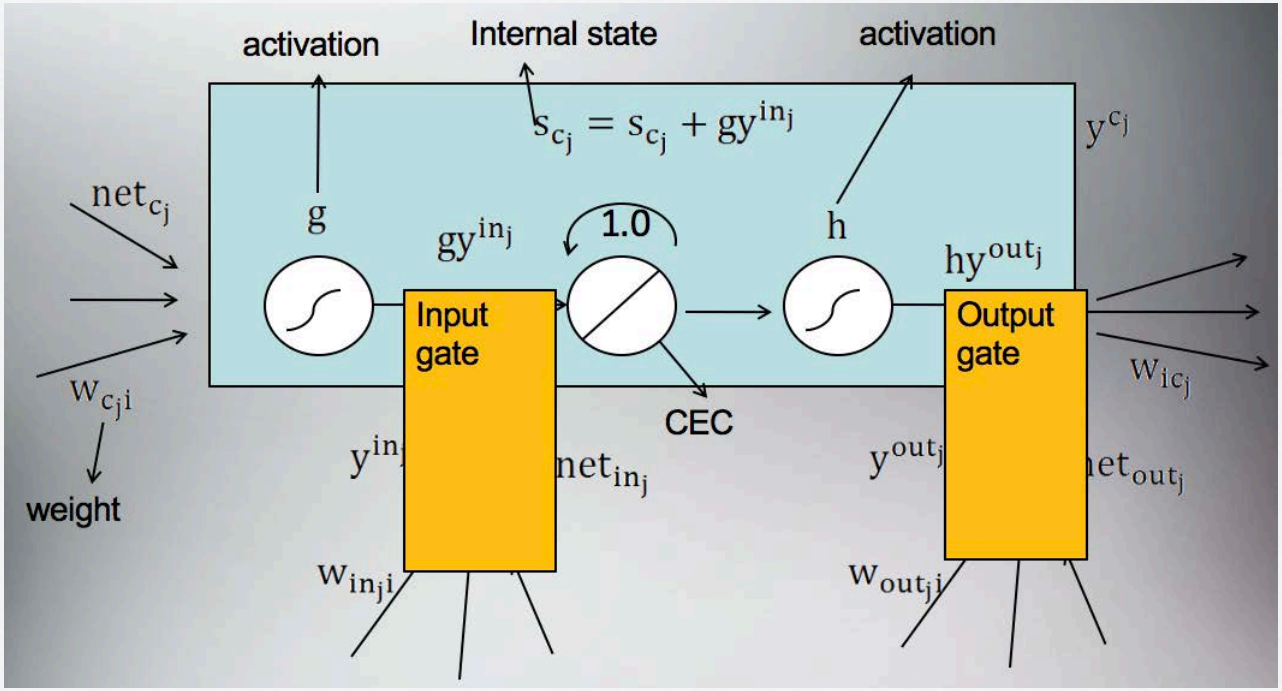


- 相比之前的RNN结构，做了如下改变：
  - 1. 矩阵Whh简化为对角矩阵，也就是只允许节点的自旋，不允许隐层的其他节点连接到本节点
  - 2. 激活函数sigmoid替换为了线性函数f(x)=x
- 以上两点保证了error可以无损由t时刻传递到t-1时刻，CEC是lstm的核心部件

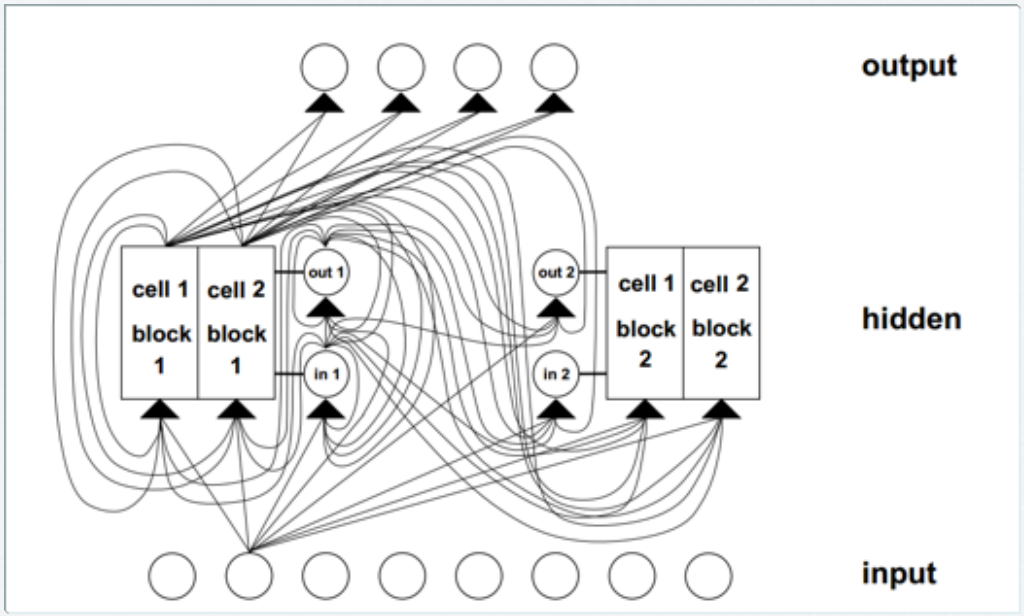
Original LSTM

- 问题二的solution:
- LSTM引入了两个gate：
  - input gate（对应图中的inj）可以控制某些输入进入cell（对应图中的cj）更新原来存储的信息，或者屏蔽输入以保持cell存储的信息不变；
  - output gate（对应图中的outj）以控制cell的信息对输出产生多大程度的影响。

Architecture of Original LSTM



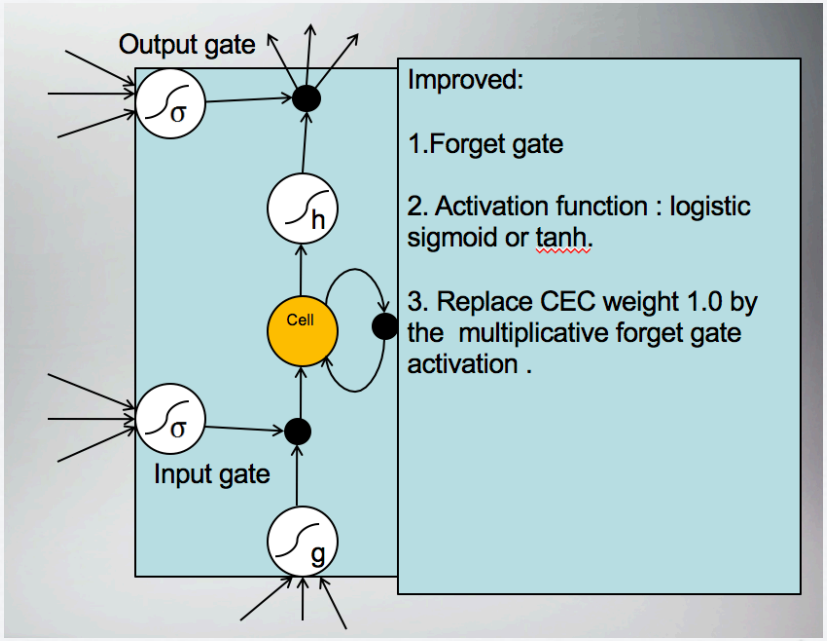
Architecture of Original LSTM

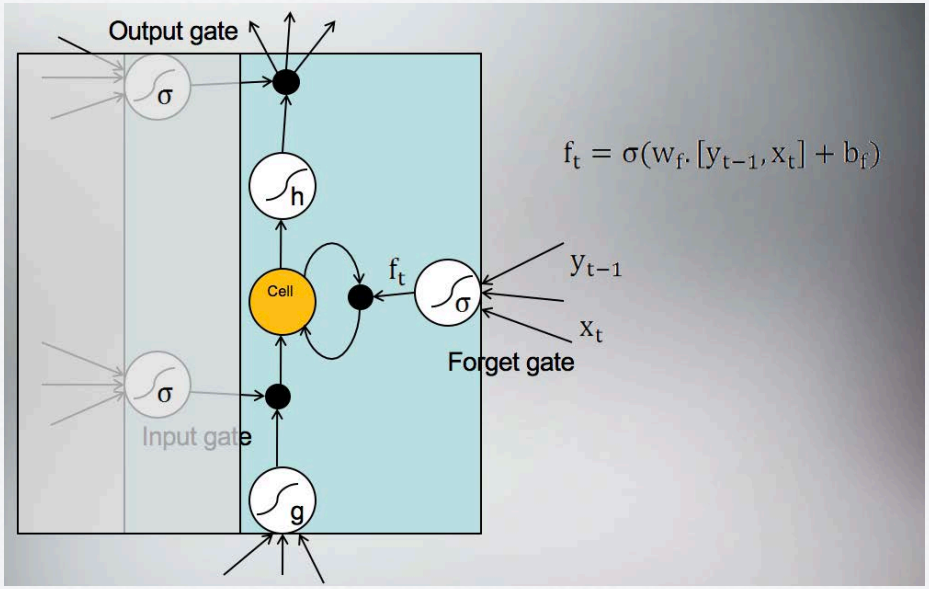


Standard LSTM

- 问题：
- 传统的LSTM存在一个问题：随着时间序列的增多，LSTM网络没有重置的机制（比如两句话合成一句话作为输入的话，希望是在第一句话结束的时候进行reset），从而导致cell state容易发生饱和，进一步会导致cell state的输出h（趋近于1）的梯度很小（sigmoid函数在x值很大的时候梯度趋向于0），阻碍了error的传播；
- 另一方面输出h趋近于1，导致cell的输出近似等于output gate的输出，意味着网络丧失了memory的功能。

Standard LSTM





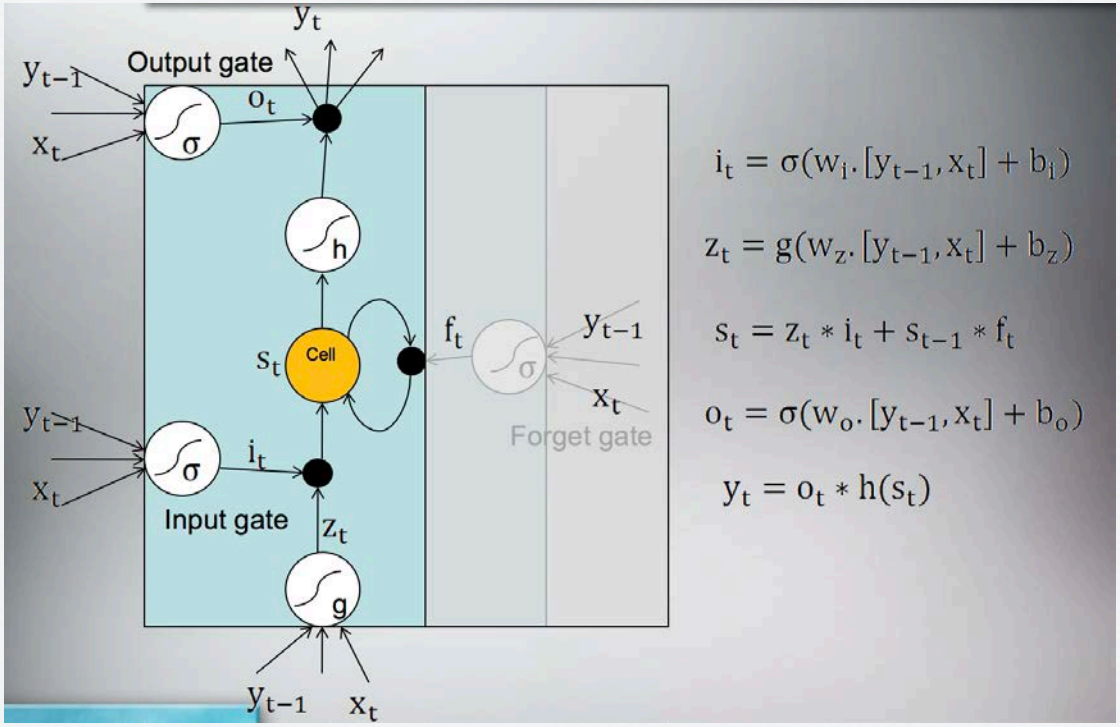
Standard LSTM

- 在传统LSTM的基础之上，引入了forget gate。使用这种结构可以让网络自动学习什么时候应该reset。具体做法即为使用yφ替换原来的CEC的常量1.0，定义如下：

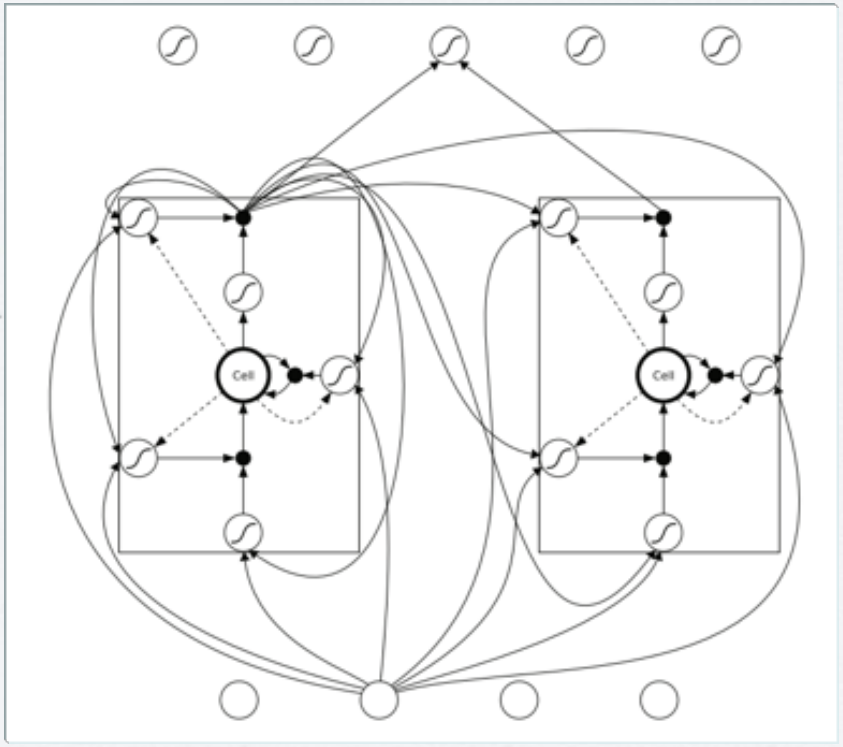
$$net_{\varphi_j}(t) = \sum_m w_{\varphi_j m} y^m(t-1)$$

$$y^{\varphi_j}(t) = f_{\varphi_j}(net_{\varphi_j}(t))$$

Standard LSTM



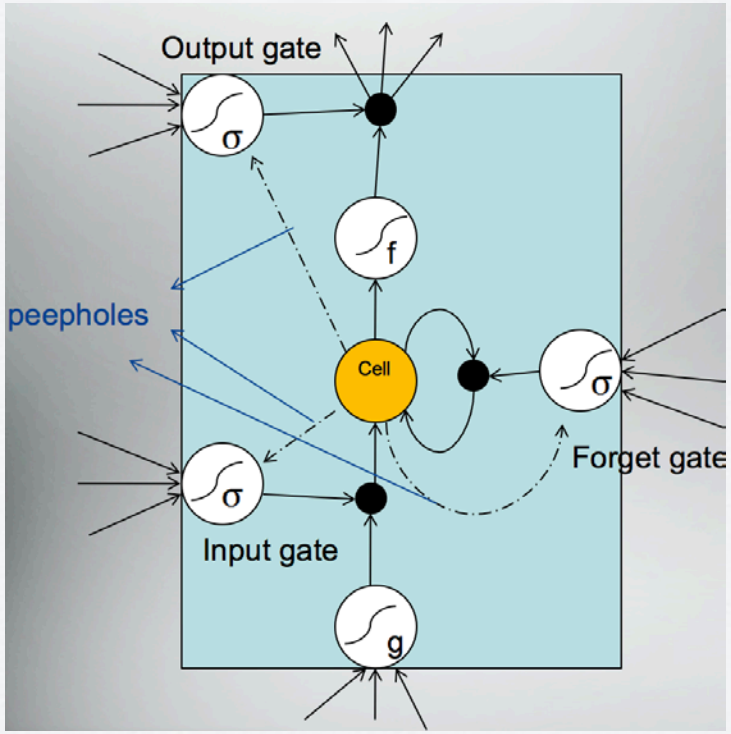
Standard LSTM



Variants:LSTM+peephole

- 问题：
- LSTM的gate的输入包含两个部分，网络输入和上一时刻（t-1）网络的输出。
- 此时如果output gate关闭（值接近0）的话，网络的输出（t时刻）将为0，下一时刻（t+1）网络gate将完全跟网络输入有关，就会丢失历史信息。
- 解决：
- 增加CEC到各个gate之间的连线，使得CEC(const error carrousels)和gate之间存在双向的关联，CEC受到当前时刻gate的限制，同时又会影响下一时刻的gate。

Variant1:LSTM+peephole Architecture



input gate和forget gate的输入增加一项 s(t−1)

- output gate的输入增加一项s(t)

$$f_t = \sigma(w_f.[s_{t-1}, y_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(w_i.[s_{t-1}, y_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(w_o.[s_t, y_{t-1}, x_t] + b_o)$$

TensorFlow的实现

Core RNN Cells for use with TensorFlow's core RNN methods

- `tf.contrib.rnn.BasicRNNCell`
- `tf.contrib.rnn.BasicLSTMCell`
- `tf.contrib.rnn.GRUCell`
- `tf.contrib.rnn.LSTMCell`
- `tf.contrib.rnn.LayerNormBasicLSTMCell`

Core RNN Cell wrappers (RNNCells that wrap other RNNCells)

- `tf.contrib.rnn.MultiRNNCell`
- `tf.contrib.rnn.LSTMBlockWrapper`
- `tf.contrib.rnn.DropoutWrapper`
- `tf.contrib.rnn.EmbeddingWrapper`
- `tf.contrib.rnn.InputProjectionWrapper`
- `tf.contrib.rnn.OutputProjectionWrapper`
- `tf.contrib.rnn.DeviceWrapper`
- `tf.contrib.rnn.ResidualWrapper`

TensorFlow的实现

- Block RNNCells
  - `tf.contrib.rnn.LSTMBlockCell`
  - `tf.contrib.rnn.GRUBlockCell`

融合的RNNCell表示在时间维度上扩展的整个RNN。

Fused RNNCells

- `tf.contrib.rnn.FusedRNNCell`
- `tf.contrib.rnn.FusedRNNCellAdaptor`
- `tf.contrib.rnn.TimeReversedFusedRNN`
- `tf.contrib.rnn.LSTMBlockFusedCell`

Unlike rnn_cell_impl.LSTMCell, this is a monolithic op and should be much faster. The weight and bias matrices should be compatible as long as the variable scope matches.

Variant2

- Another variation is to use coupled forget and output gates.
- Instead of separately deciding what to forget and what we should add
- new information to, we make those decisions together.

$$s_t = (1 - f_t) * z_t + s_{t-1} * f_t$$

LSTM-like cells

- `tf.contrib.rnn.CoupledInputForgetGateLSTMCell`

Variant3

- A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014).

- They used neither peephole connections nor output activation functions, and coupled the input and the forget gate into an update gate.
- Finally, their output gate (called reset gate) only gates the recurrent connections to the block input (Wz).

TensorFlow框架下的RNN实践小结

- TF的RNN APIs主要集中tensorflow/python/ops中的rnn和rnn_cell两个模块。其中，后者定义了一些常用的RNN cells，包括RNN和优化的LSTM、GRU等等；前者则提供了一些helper方法。
- 创建一个基础的RNN：
  - from tensorflow.contrib.rnn import rnn_cell
  - cell = rnn_cell.BasicRNNCell(inputs, state)
- 创建一个LSTM或者GRU的cell？
  - cell = rnn_cell.BasicLSTMCell(num_units) #最最基础的，不带peephole。
  - cell = rnn_cell.LSTMCell(num_units, input_size) #可以设置peephole等属性。
  - cell = rnn_cell.GRUCell(num_units)

TensorFlow框架下的RNN实践小结

- 调用呢？
  - output, state = cell(input, state)
- 按timestep调用需要设置variable_scope的reuse属性为True，因此可写为：
  - state = cell.zero_state(batch_size, dtype=tf.float32)
  - outputs, states = rnn.rnn(cell, inputs, initial_state=state)
- 怕overfit，加个Dropout如何？
  - cell = rnn_cell.DropoutWrapper(cell, input_keep_prob=0.5, output_keep_prob=0.5)

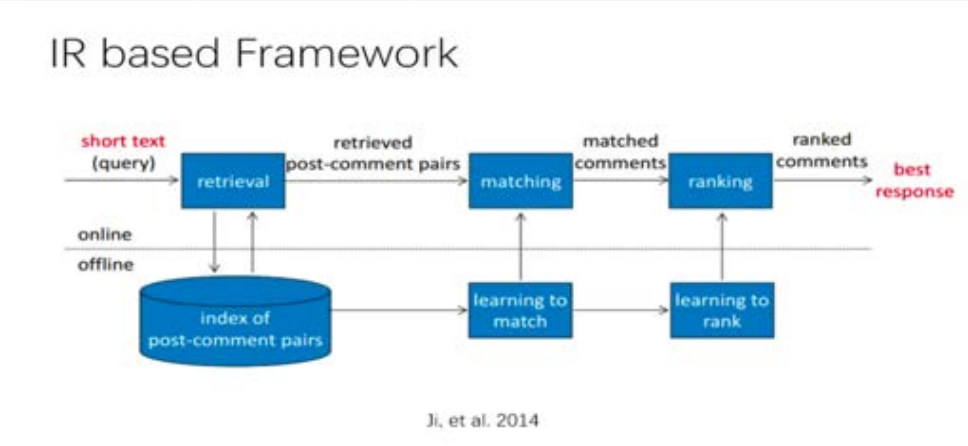
TensorFlow框架下的RNN实践小结

- 做个三层的带Dropout的网络？
  - cell = rnn_cell.DropoutWrapper(cell, output_keep_prob=0.5)
  - cell = rnn_cell.MultiRNNCell([cell] * 3)
  - inputs = tf.nn.dropout(inputs, 0.5) #给第一层单独加个Dropout。

TensorFlow在简易智能聊天机器人的实现

智能聊天机器人的实现方式

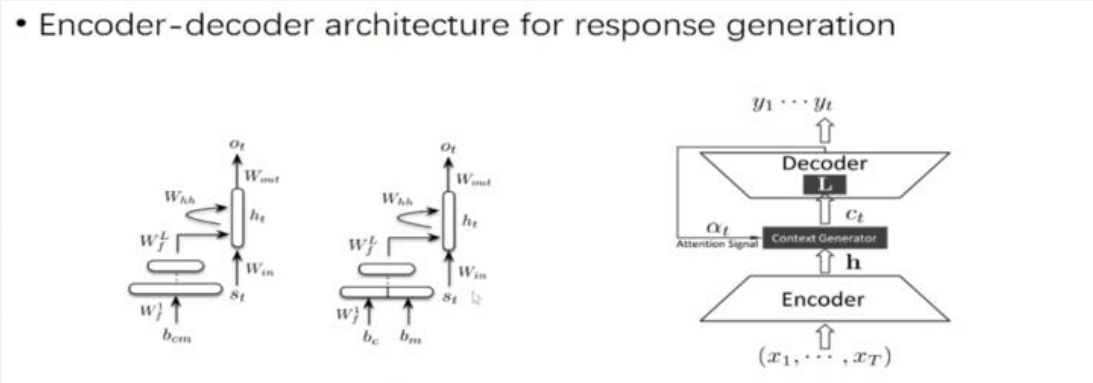
- 基于检索的模型（Retrieval-Based Models）



- 优势：
  - 答句可读性好
  - 答句多样性强
  - 出现不相关的答句，容易分析、定位bug
- 劣势：需要对候选的结果做排序，进行选择

智能聊天机器人的实现方式

- 生成式模型（Generative Models）



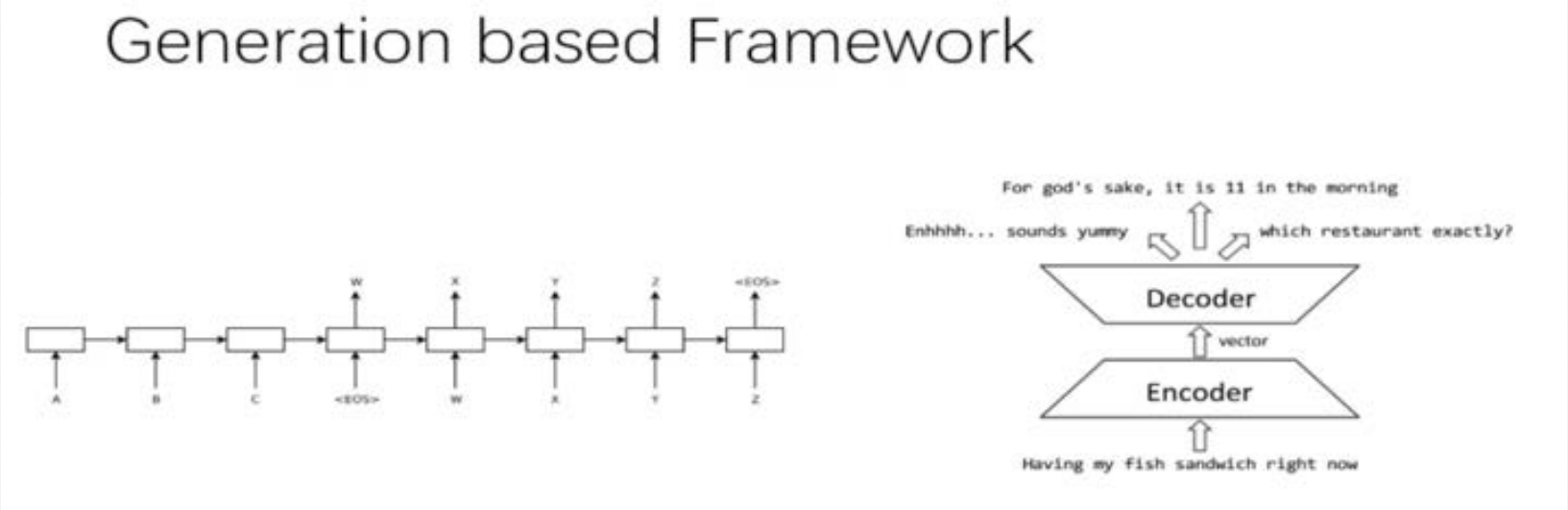
- 优势：
  - 端到端的训练，比较容易实现
  - 避免维护一个大的Q-A数据集
  - 不需要对每一个模块额外进行调优，避免了各个模块之间的误差级联效应
- 劣势：难以保证生成的结果是可读的，多样的。

评价一个聊天机器人的好坏？

- 问句和答句的相关性。本质是：短文本相关度计算
  - 词语共现的统计
  - 基于机器翻译的相关度计算
  - 主题模型（LDA）的相似度计算
- 智能聊天机器人的目标：
  - 和人类能够进行持续的沟通
  - 对不同的提问能够给出合适的回答
  - 考虑到人类不同个性化的差异性，给出差异性的回答
  - （例如，同一个问题，对男女老少不同群体的回答应该略有差异）

Sequence to Sequence+Attention

- seq2seq模型是一个翻译模型，主要是把一个序列翻译成另一个序列。
- 用两个RNNLM，一个作为编码器，另一个作为解码器，组成RNN编码器-解码器。



聊天机器人的TensorFlow实现

- 数据集：
- 康奈尔大学的Corpus数据集（Cornell Movie Dialogs Corpus）
- 含有600多部电影的对白。对白示例如下：

- L1045 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ They do not!
- L1044 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ They do to!

聊天机器人的TensorFlow实现

- 数据预处理：
  - 把数据集整理成“问”和“答”的文件，生成.enc（问句）和.dec（答句）文件，如下：
    - ├—— test.dec # 测试集答句
    - ├—— test.enc # 测试集问句
    - ├—— train.dec # 训练集答句
    - ├—— train.enc # 训练集问句
- train.enc问句示例如下：
  - Gosh, if only we could find Kat a boyfriend...
  - C'esc ma tete. This is my head
  - How is our little Find the Wench A Date plan progressing?

聊天机器人的TensorFlow实现

- 创建词汇表，词汇表的文件里有2万个词汇，如下：
- ├—— vocab20000.dec # 答句的词汇表
- └—— vocab20000.enc # 问句的词汇表
-
- 词汇表的内容如下：
- _PAD
- _GO
- _EOS
- _UNK
- .
- '
- ,
- I
- ?
- you
- the
- to
- it

其中_GO、_EOS、_UNK、_PAD是在seq2seq模型中使用的特殊标记，用来填充标记对话：
_GO 标记对话开始；
_EOS标记对话结束；
_UNK标记未出现在词汇表中的字符，用来替换稀有词汇；
_PAD是用来填充序列，保证批次中的序列有相同的长度。

聊天机器人的TensorFlow实现

- 把问句答句文件转换成的ids文件，如下：
-
- ├—— test.enc.ids20000
- ├—— train.dec.ids20000
- └—— train.enc.ids20000
-
- 问句和答句转换成的ids文件中，每一行是一个问句或答句，
- 每一行中的每一个id代表问句或答句中对应位置的词，格式如下：
- 185 4 4 4 146 131 5 1144 39 313 53 102 1176 12042 4 2020 9 2691 9
- 792 15 4
- 7518 4
- 2993 49 88 109 54 13 765 466 252 4 4 4

聊天机器人的TensorFlow实现

采用编码器-解码器框架进行训练。

1．定义训练参数

这里，我们将参数写到一个专门的文件seq2seq.ini中，如下：

```
[strings]
# 模式: train，test，serve
mode = train
train_enc = data/train.enc
train_dec = data/train.dec
test_enc = data/test.enc
test_dec = data/test.enc
# 模型文件和词汇表的存储路径
working_directory = working_dir/
[ints]
# 词汇表大小
enc_vocab_size = 20000
dec_vocab_size = 20000
# LSTM层数
num_layers = 3
# 每层大小，可以取值：128, 256, 512, 1024
layer_size = 256

max_train_data_size = 0
batch_size = 64
# 每多少次迭代存储一次模型
steps_per_checkpoint = 300
[floats]
learning_rate = 0.5 # 学习速率
learning_rate_decay_factor = 0.99 # 学习速率下降系数
max_gradient_norm = 5.0
```

聊天机器人的TensorFlow实现

- 2．定义网络模型
- 下面来定义seq2seq模型，该模型的代码在seq2seq_model.py中。
- 定义一个seq2seq+Attention模型类，里面主要包含3个函数：

- （1）初始化模型的函数（__init__）；
- （2）训练模型的函数（step）；
- （3）获取下一批次训练数据的函数（get_batch）。

聊天机器人的TensorFlow实现

- 4．训练结果

- 训练了417次后，生成了大小为209 MB的seq2seq.ckpt-417.data-00000-of-00001模型文件，开始进行测试，结果如下（行首有“>”的是我的输入，没有的是机器人的输出）：

- > Hello
- Hi .
- > I love you.
- Yeah .
- > What
- What?
- > Sunny day
- What?

Make Professional Discovery

MPD

msup

感谢您参加本届MPD！

www.mpd.org.cn

400-812-8020