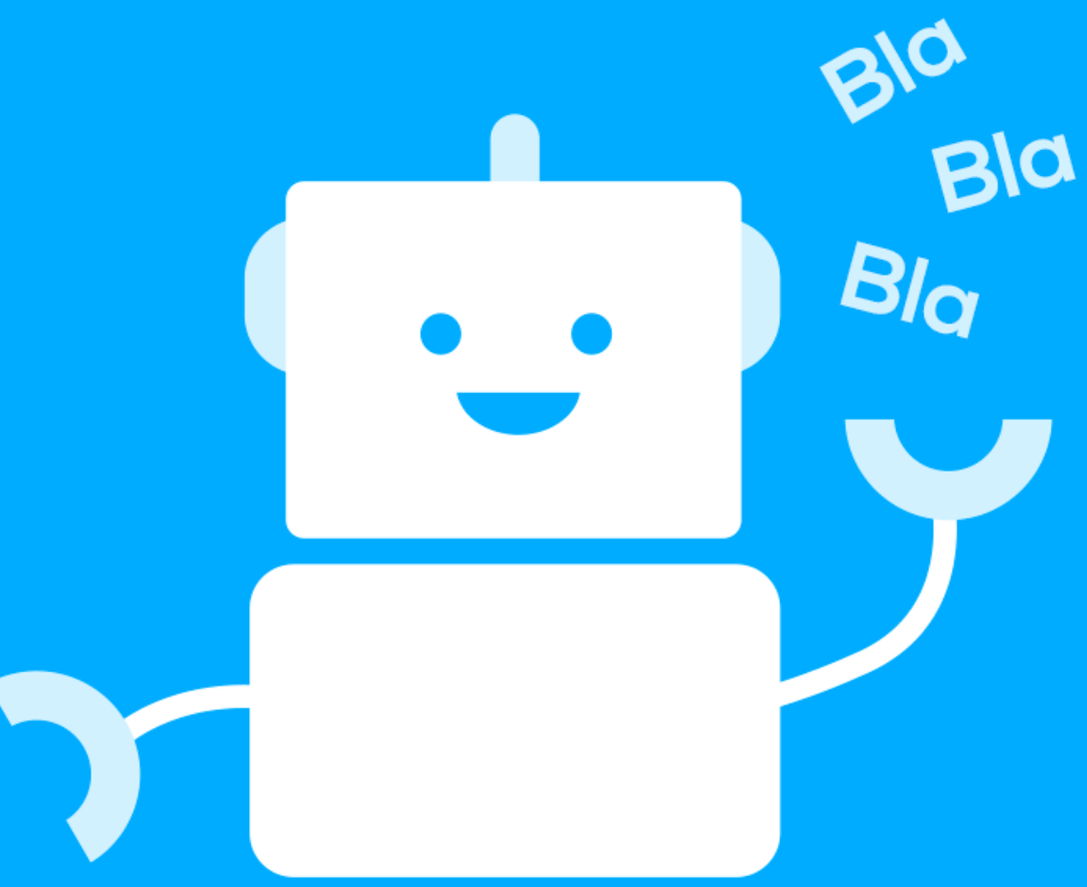
Building Task-Oriented Neural Dialog System

Head First Theory and Practice

Yanran Li
The Hong Kong Polytechnic University

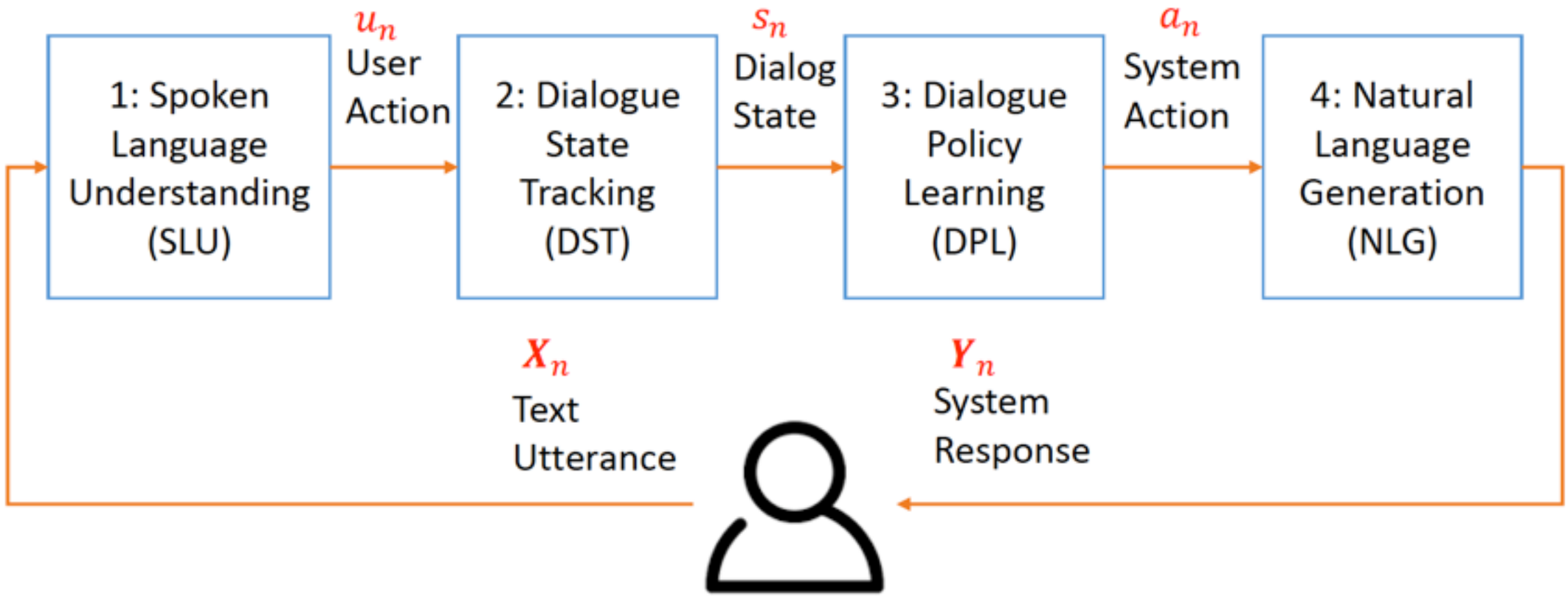


VS.

Task-oriented | Dialog System

Task-oriented Dialog System

- Modular: often 4 modules

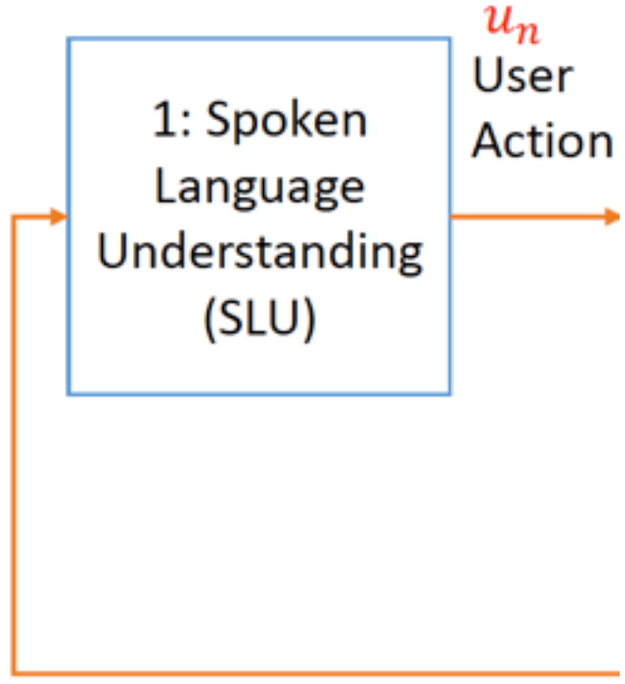


Task-oriented Dialog System

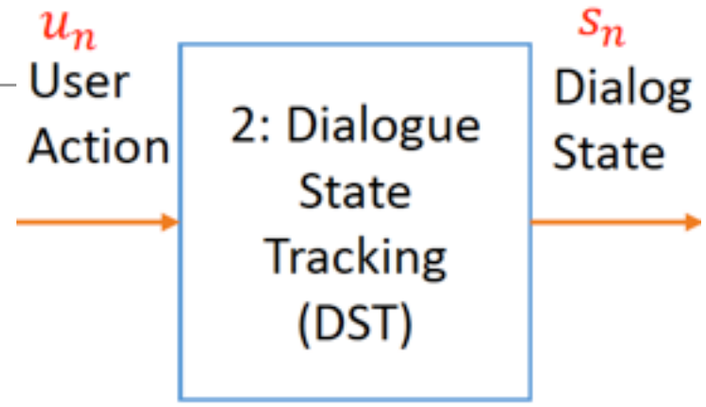
- Modular: often 4 modules
- Sub-modules, sub-tasks:
  - Spoken Language Understanding (SLU)
  - Dialogue State Tracking (DST)
  - Dialogue Policy Learning (DPL)
  - Natural Language Generation (NLG)

Task-oriented Dialog System

- Modular: often 4 modules
- Sub-modules, sub-tasks:
  - Spoken Language Understanding (SLU)
    - SLU turns natural language into user intention and slot-values, and it takes input and outputs structured user action
- Dialogue State Tracking (DST)
- Dialogue Policy Learning (DPL)
- Natural Language Generation (NLG)

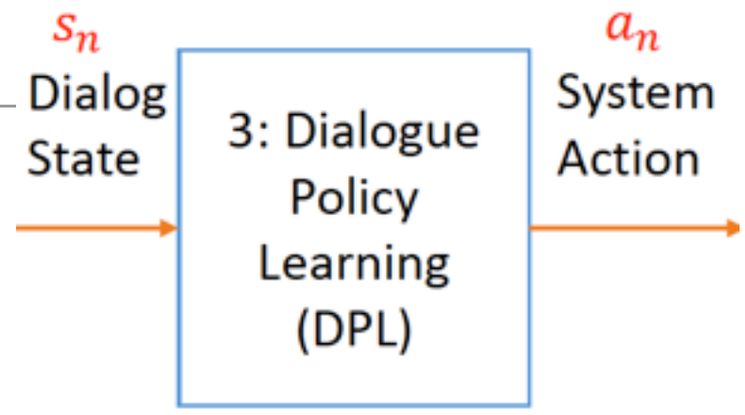


Task-oriented Dialog System



- Modular: often 4 modules
- Sub-modules, sub-tasks:
  - Spoken Language Understanding (SLU)
  - Dialogue State Tracking (DST)
    - DST tracks the current dialogue state, and outputs dialogue state
  - Dialogue Policy Learning (DPL)
  - Natural Language Generation (NLG)

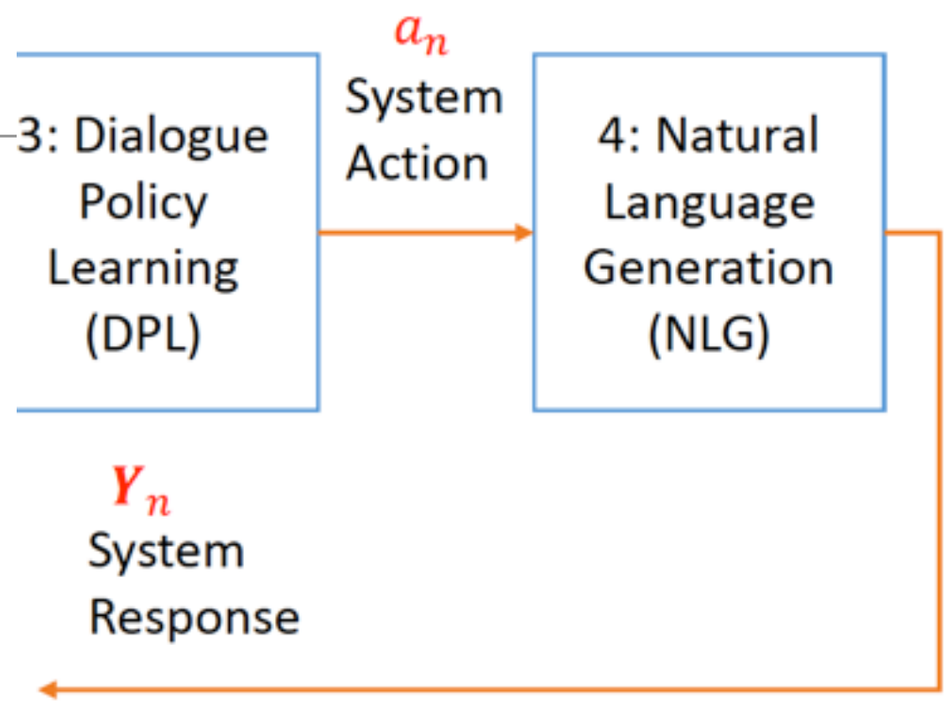
Task-oriented Dialog System

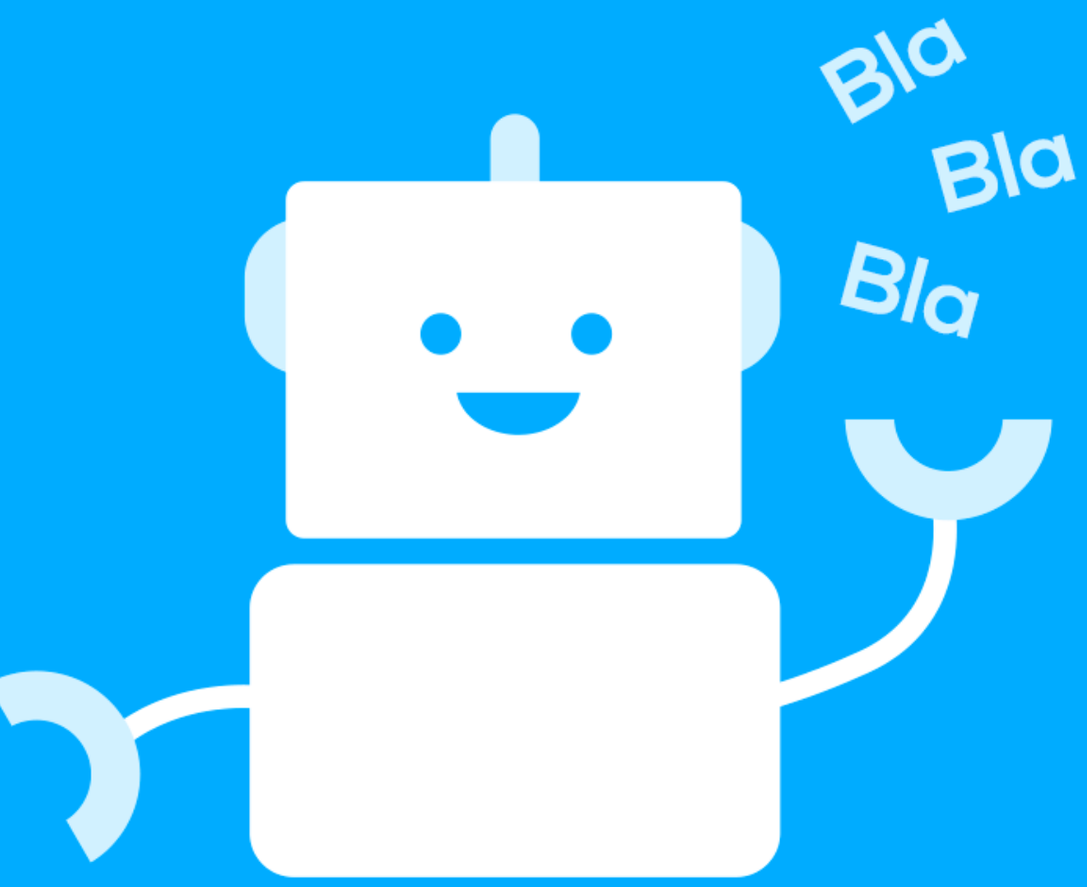


- Modular: often 4 modules
- Sub-modules, sub-tasks:
  - Spoken Language Understanding (SLU)
  - Dialogue State Tracking (DST)
  - Dialogue Policy Learning (DPL)
    - Policy decides which system action to take based on the dialogue state, and it takes dialogue state as input and outputs system action
  - Natural Language Generation (NLG)

Task-oriented Dialog System

- Modular: often 4 modules
- Sub-modules, sub-tasks:
  - Spoken Language Understanding (SLU)
  - Dialogue State Tracking (DST)
  - Dialogue Policy Learning (DPL)
  - Natural Language Generation (NLG)
    - NLG turns a system action into natural language, and it takes the system action as input and outputs the system response





VS.

Task-oriented | An Example

Example

- Wen et al., “A Network-based End-to-End Trainable Task-oriented Dialogue System”. EACL 2017.[7]

- https://github.com/shawnwun/NNDIAL

Task-oriented Neural Dialog System [7]

Can I have Korean

Little Seoul serves great Korean .

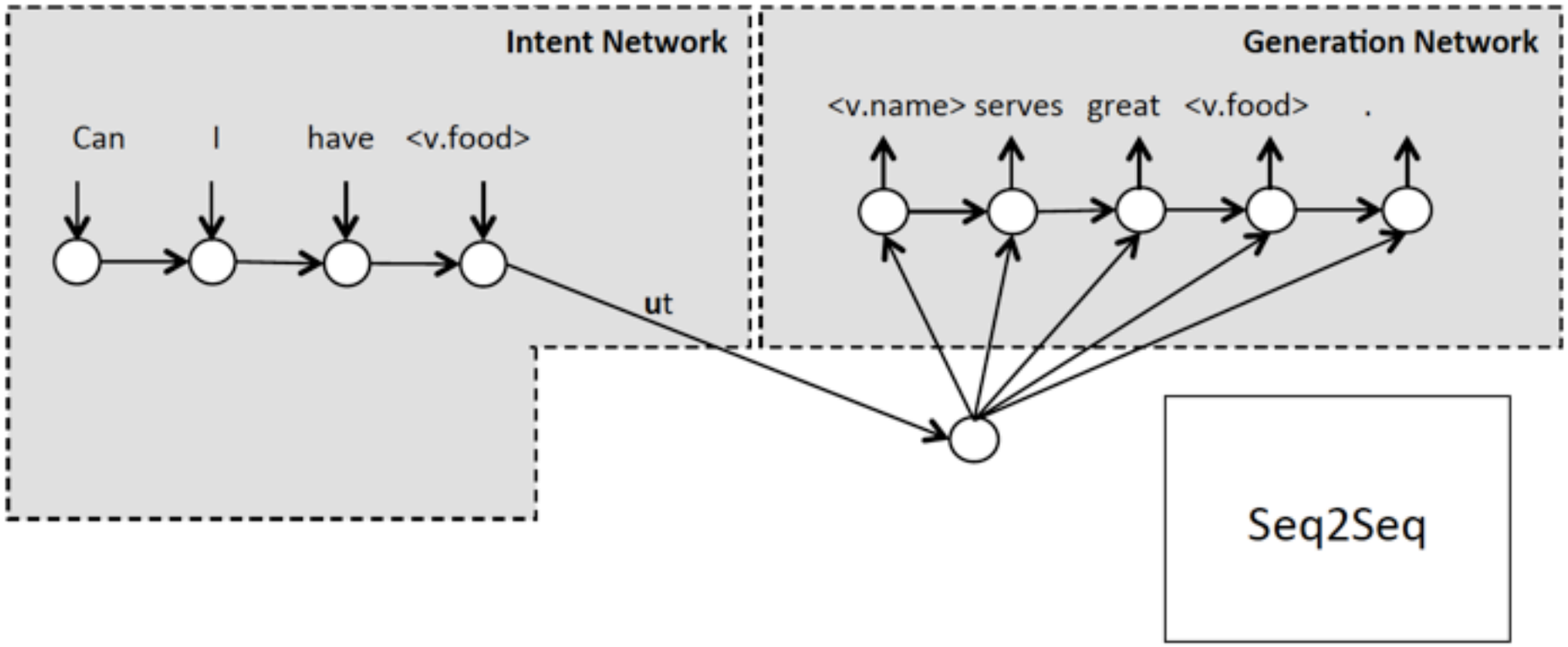
Task-oriented Neural Dialog System

Can I have <v.food>

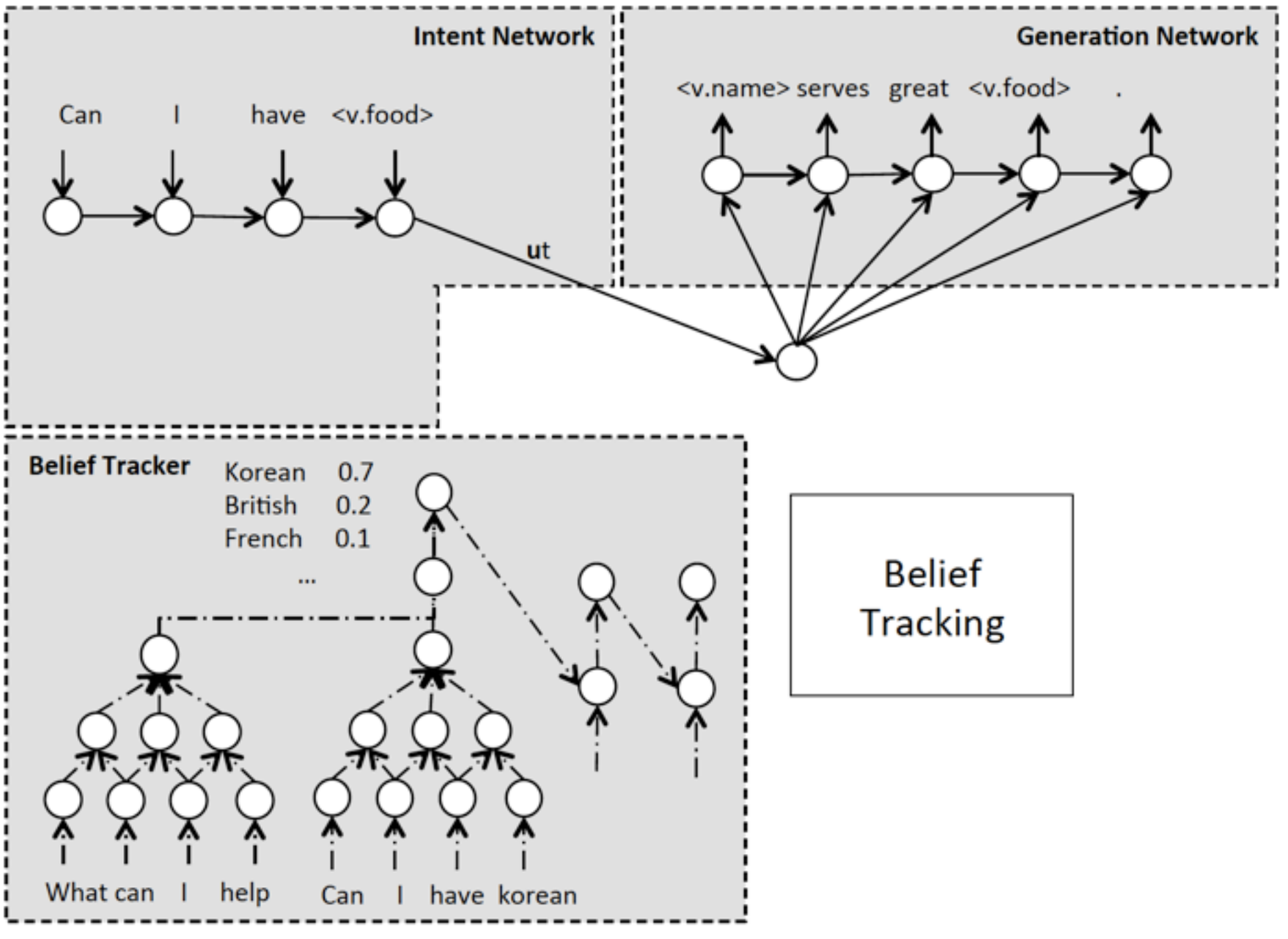
<v.name> serves great <v.food> .

Delexicalisation

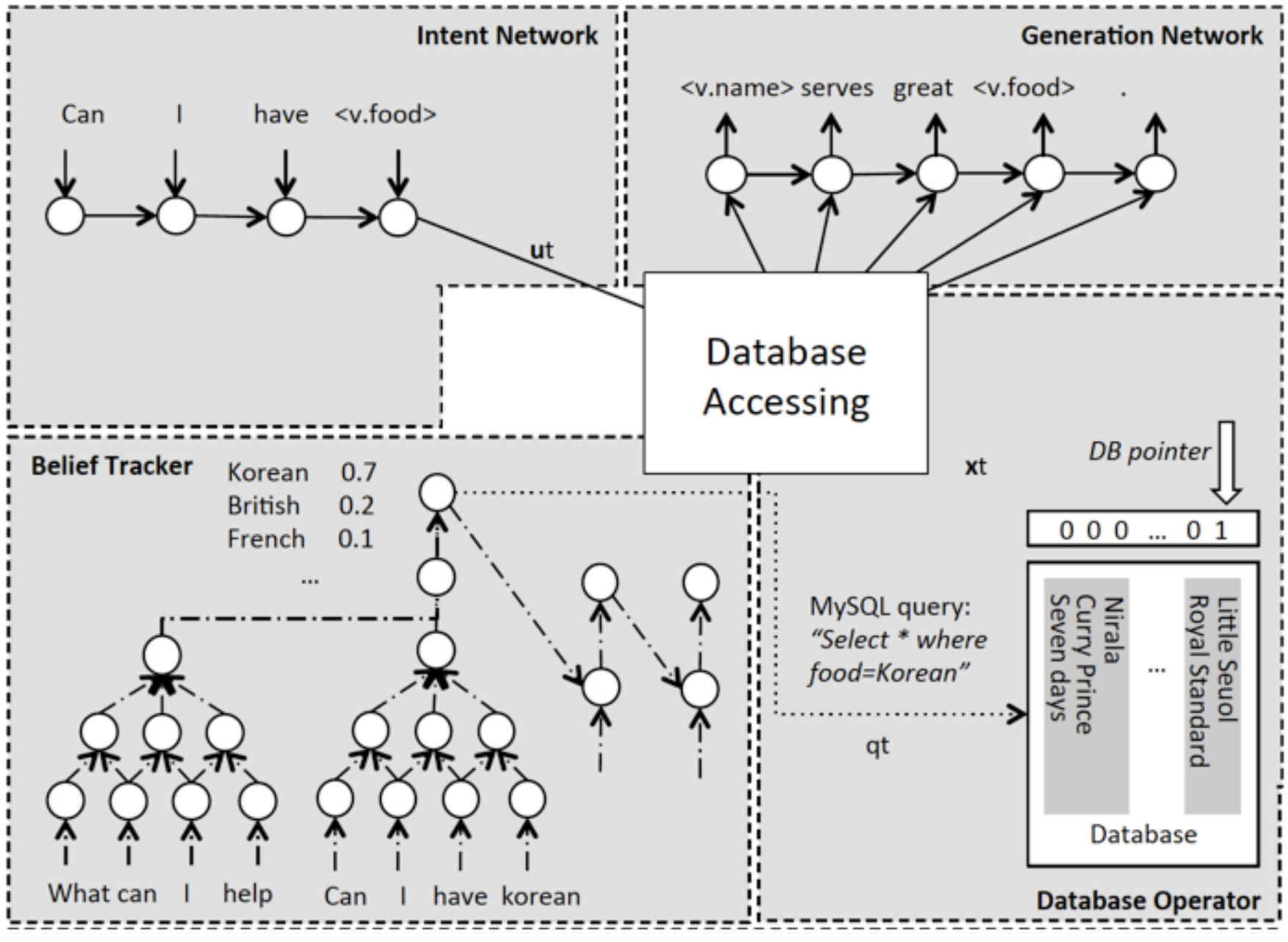
Task-oriented Neural Dialog System



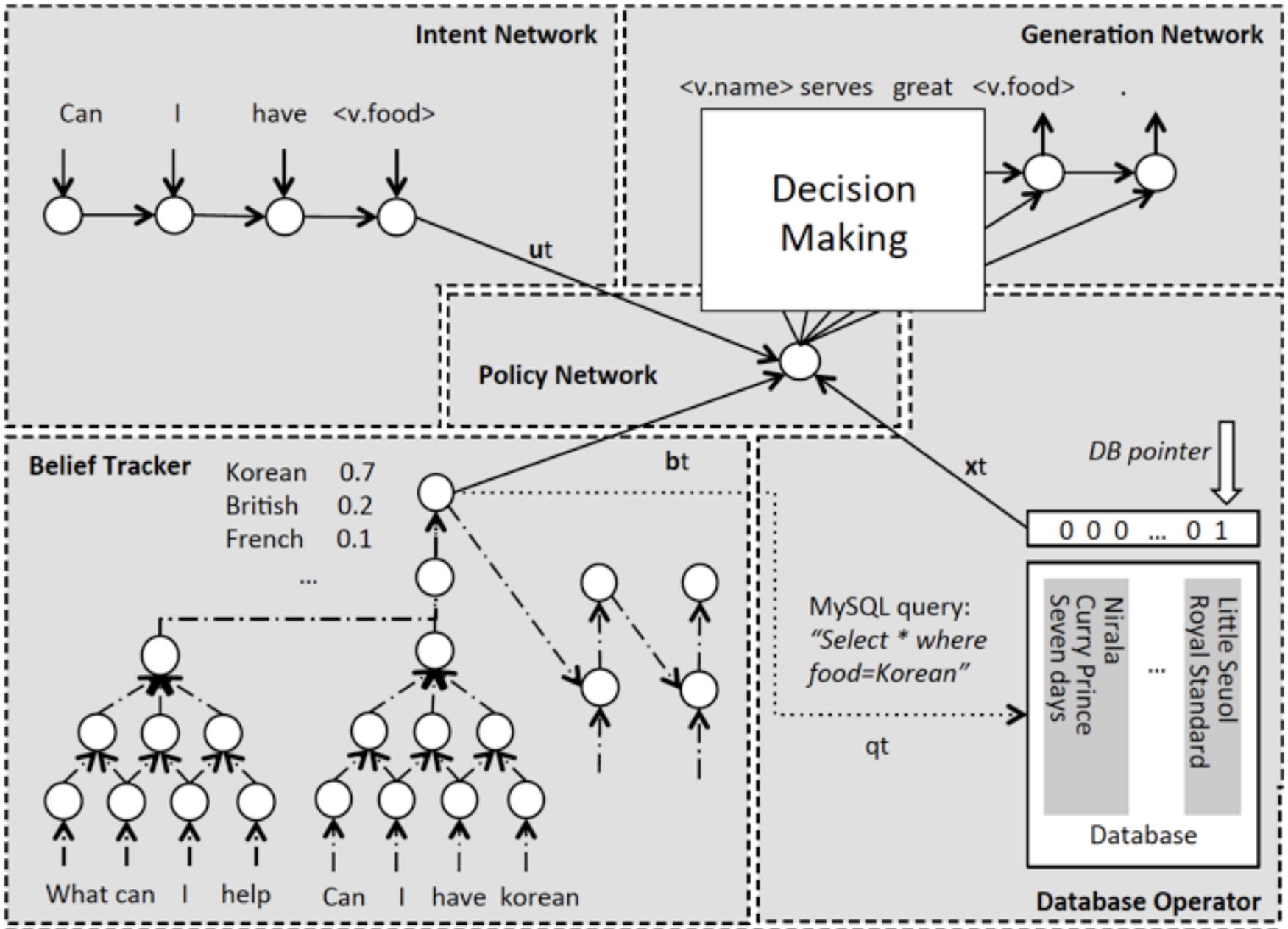
Task-oriented Neural Dialog System



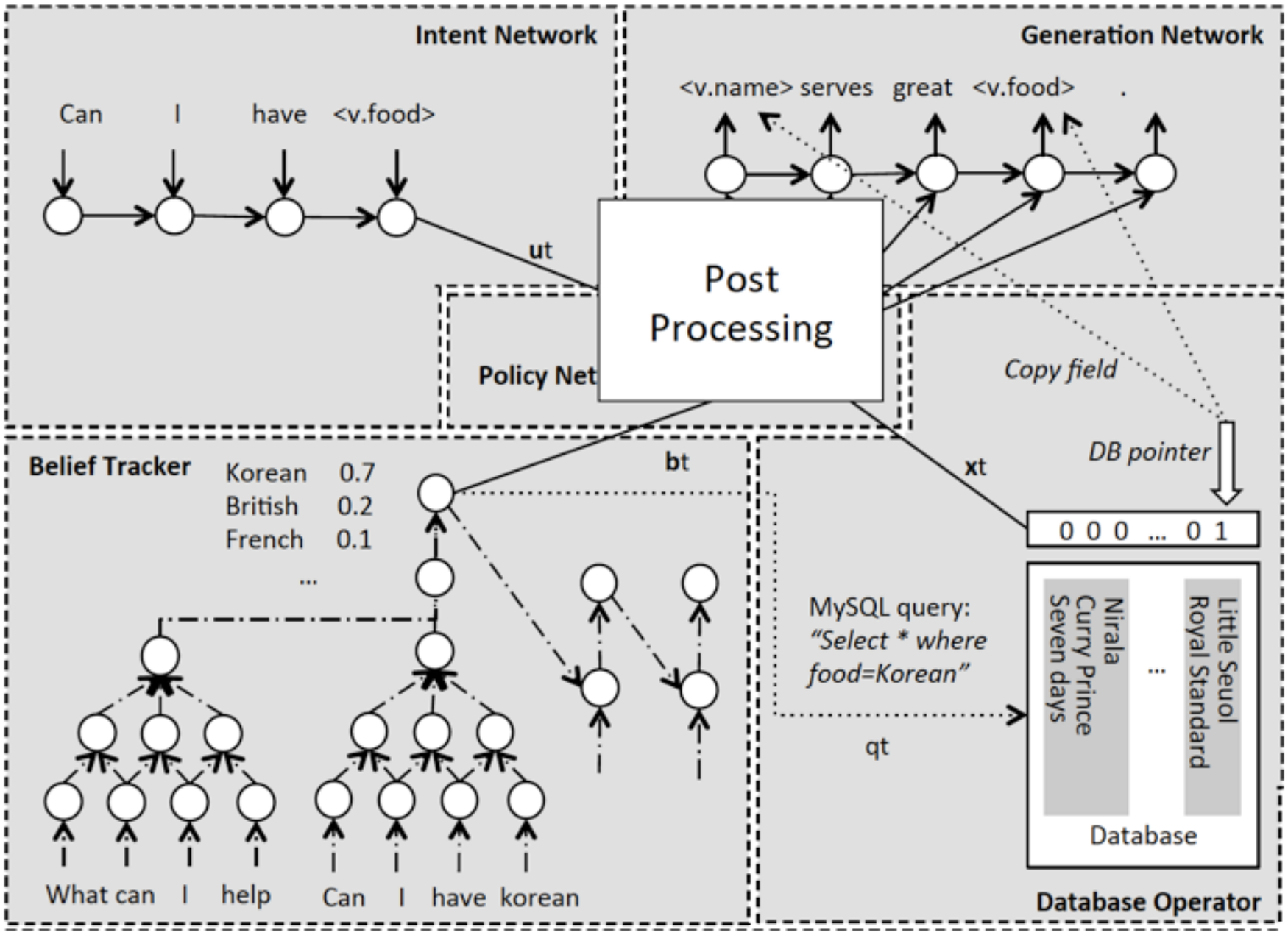
Task-oriented Neural Dialog System



Task-oriented Neural Dialog System



Task-oriented Neural Dialog System



Task-oriented Neural Dialog System [7]

Can I have Korean

Little Seoul serves great Korean .

Task-oriented Neural Dialog System

Can I have <v.food>

<v.name> serves great <v.food> .

Delexicalisation

Delexicalisation

- Replace slot values by a symbol representing its corresponding slot tokens. In the final postprocessing phase, these slot symbols are converted back to the corresponding slot values.

- A small knowledge graph defining the slot-value pairs the system can talk about for a particular task.

- Features and corpus can be found in [12].

- *In this paper, the system was designed to assist users to find a restaurant in the Cambridge, UK area. There are three informable slots (food, pricerange, area) that users can use to constrain the search and six requestable slots (address, phone, postcode plus the three informable slots) that the user can ask a value for once a restaurant has been offered. There are 99 restaurants in the DB.* [7]

Delexicalisation

- Replace slot values by a symbol representing its corresponding slot tokens. In the final postprocessing phase, these slot symbols are converted back to the corresponding slot values.

- A small knowledge graph defining the slot-value pairs the system can talk about for a particular task.

- Features and corpus can be found in [12].

Delexicalised token	Examples
informable slot token	<s.food>, <s.area>,...
informable value token	<v.food>, <v.area>,...
requestable slot token	<s.phone>,<s.address>,...
requestable value token	<v.phone>,<v.address>,...

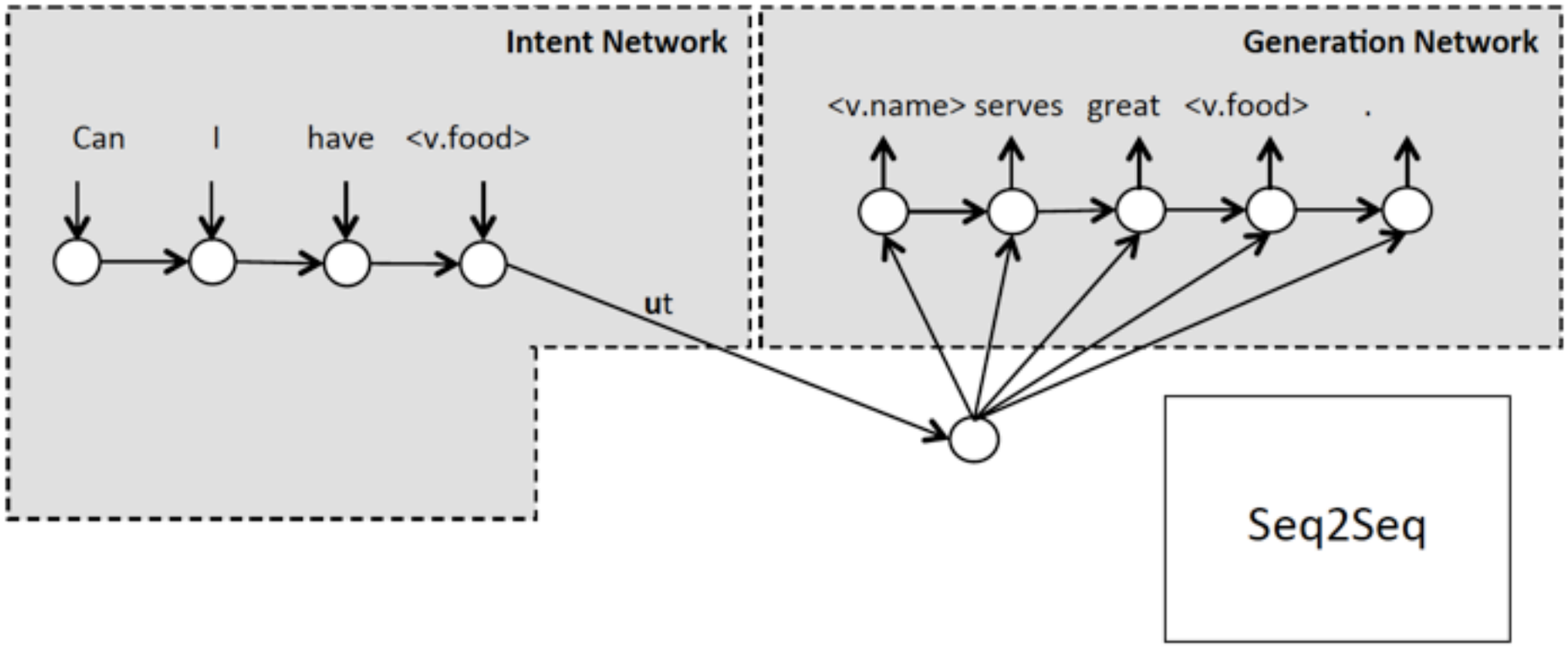
Delexicalisation

```
549     def prepareSlotValues(self):
550
551         print '\tprepare slot value templates ...'
552         # put db requestable values into s2v
553         for e in self.db:
554             for s,v in e.iteritems():
555                 if self.s2v['requestable'].has_key(s):
556                     self.s2v['requestable'][s].append(v.lower())
557                 if self.s2v['other'].has_key(s):
558                     self.s2v['other'][s].append(v.lower())
559         # sort values
560         for s,vs in self.s2v['informable'].iteritems():
561             self.s2v['informable'][s] = sorted(list(set(vs)))
562         for s,vs in self.s2v['requestable'].iteritems():
563             self.s2v['requestable'][s] = sorted(list(set(vs)))
564         for s,vs in self.s2v['other'].iteritems():
565             self.s2v['other'][s] = sorted(list(set(vs)))
566
567         # make a 1-on-1 mapping for delexicalisation
568         self.supervalues = []
569         self.values = []
570         self.slots  = []
```

Delexicalisation

```
519     def delexicalise(self,utt,mode='all'):
520         inftoks =   ['[VALUE_'+s.upper()+']' for s in self.s2v['informable'].keys()] + \
521                     ['[SLOT_' +s.upper()+']' for s in self.s2v['informable'].keys()] + \
522                     ['[VALUE_DONTCARE]','[VALUE_NAME]'] +\
523                     ['[SLOT_' +s.upper()+']' for s in self.s2v['requestable'].keys()]
524         reqtoks =   ['[VALUE_'+s.upper()+']' for s in self.s2v['requestable'].keys()]
525         for i in range(len(self.values)):
526             # informable mode, preserving location information
527             if mode=='informable'and self.slots[i] in inftoks:
528                 tok = self.slots[i]+'::'+(self.supervalues[i]).replace(' ','-')
529                 utt = (' '+utt+' ').replace(' '+self.values[i]+' ',' '+tok+' ')
530                 utt = utt[1:-1]
531             # requestable mode
532             elif mode=='requestable' and self.slots[i] in reqtoks:
533                 utt = (' '+utt+' ').replace(' '+self.values[i]+' ',' '+self.slots[i]+' ')
534                 utt = utt[1:-1]
535             elif mode=='all':
536                 tok = self.slots[i]+'::'+(self.supervalues[i]).replace(' ','-') \
537                         if self.slots[i] in inftoks else self.slots[i]
538                 utt = (' '+utt+' ').replace(' '+self.values[i]+' ',' '+tok+' ')
539                 utt = utt[1:-1]
540         utt = re.sub(digitpat,'[VALUE_COUNT]',utt)
541         return utt
```

Task-oriented Neural Dialog System



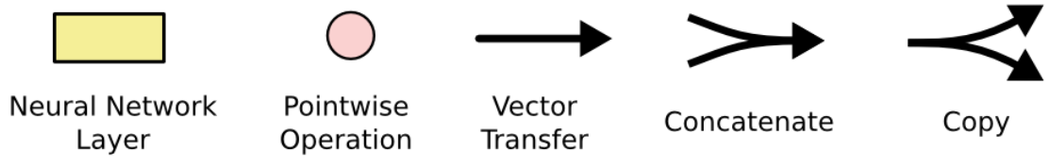
Intent Network

- convert user input (utterances) into distributed representation at every speaker turn.

- can be viewed as the encoder in the sequence-to-sequence learning framework, e.g. LSTM or CNN. The example code repository adopts LSTM.

```
* The encoder modules contain:
- LSTM encoder      : an LSTM network that encodes the user utterance.
- RNN+CNN tracker   : a set of slot trackers that keep track of each slot/value pair across turns.
- DB operator       : a discrete database accessing component.

* The decoder modules contain:
- Policy network    : a decision-making module that produces the conditional vector for decoding.
- LSTM decoder      : an LSTM network that generates the system response.
```

Recall Long-Short Term Memory (LSTM)



- Three sigmoid gates and One cell

$$\mathbf{i}_t = \sigma(\mathbf{W}_{wi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1})$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{wf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1})$$

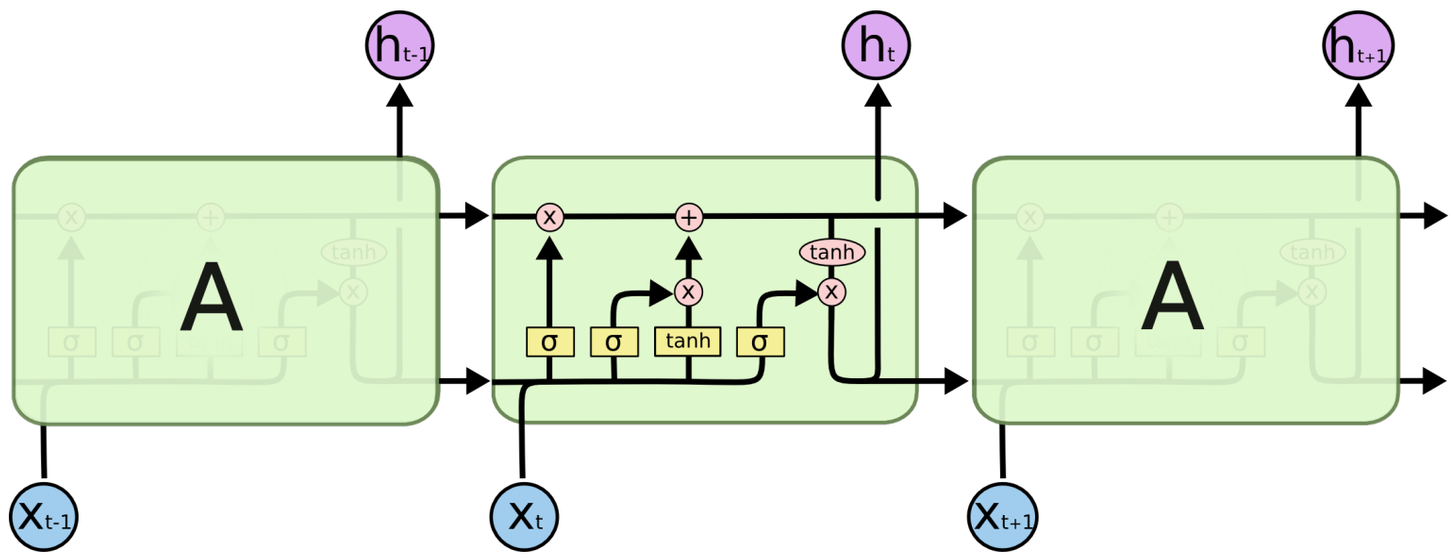
$$\mathbf{o}_t = \sigma(\mathbf{W}_{wo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1})$$

$$\hat{\boldsymbol{c}}_t = \tanh(\mathbf{W}_{wc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1})$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\boldsymbol{c}}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

- LSTM

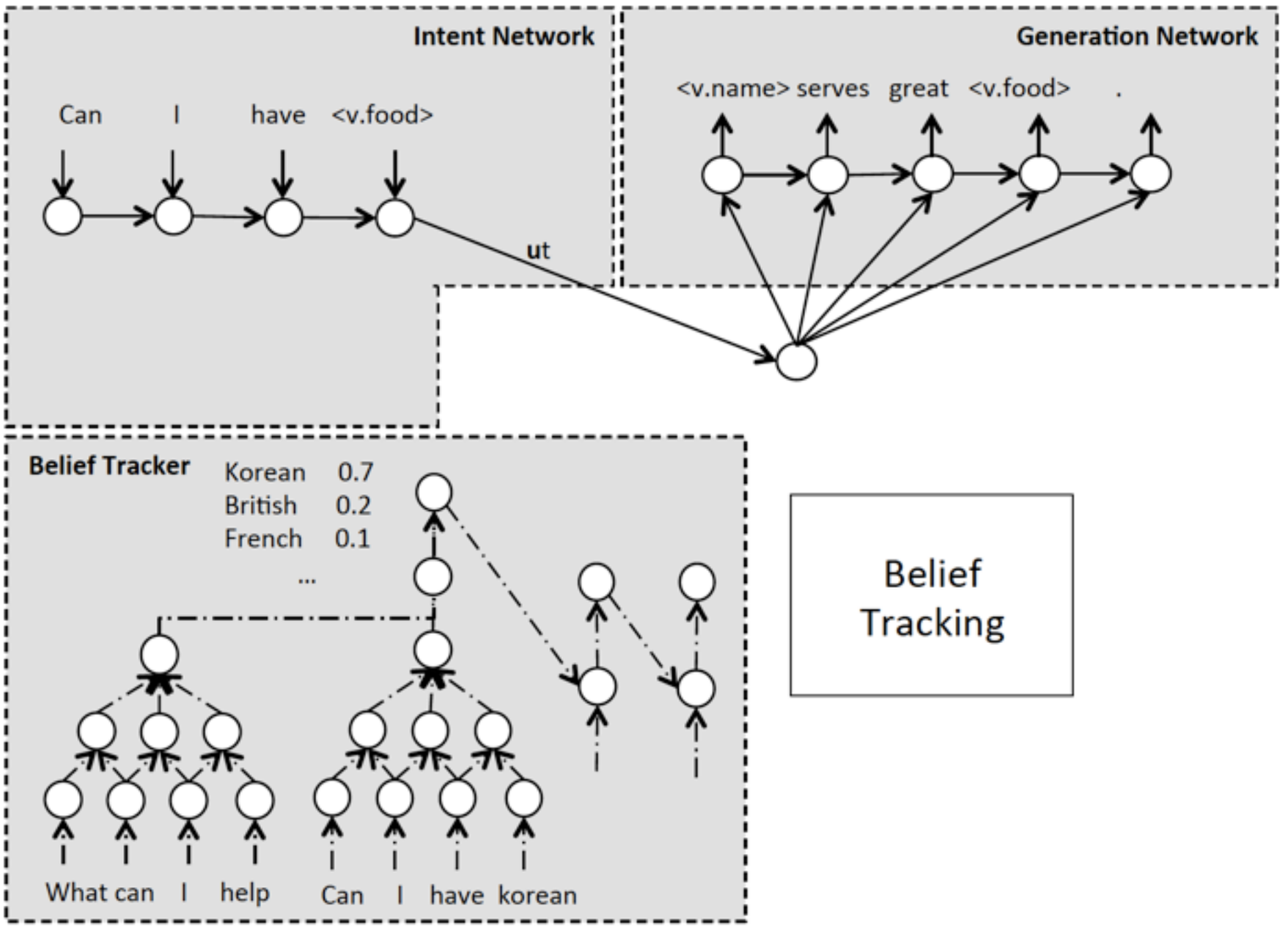


Intent Network

```
# LSTM Encoder
class LSTMEncoder(BaseNNModule):

    def __init__(self, vocab_size, ihidden_size):

        # parameters for encoder biLSTM
        self.dih = ihidden_size
        self.di  = vocab_size
        self.iWgate = theano.shared(0.3 * np.random.uni
                (self.dih,self.dih*4)).astype(theano.c
        self.iUgate = theano.shared(0.3 * np.random.uni
                (self.dih,self.dih*4)).astype(theano.c
        self.Wemb   = theano.shared(0.3 * np.random.uni
                (self.di,self.dih)).astype(theano.confi
        self.bf = theano.shared(2.0*np.ones((ihidden_si
                astype(theano.config.floatX))

        self.params = [
                self.iWgate, self.iUgate,   self.Wemb,  self.bf ]

        # initial states
        self.ih0 = theano.shared(np.zeros((self.dih),\
                dtype=theano.config.floatX))
        self.ic0 = theano.shared(np.zeros((self.dih),\
                dtype=theano.config.floatX))
```

```
# Bidirectional encoding
def bidirectional_encode(fEncoder,bEncoder,sent,leng):
    fw_intent_t = fEncoder.encode(sent,leng)
    bw_intent_t = bEncoder.encode(T.concatenate(
        [sent[:leng][::-1],sent[leng:] ],axis=0),leng)
    intent_t = T.concatenate(
            [fw_intent_t,bw_intent_t],axis=0)
    return intent_t

def bidirectional_read(fEncoder, bEncoder, sent):
    fw_intent_t = fEncoder.read(sent)
    bw_intent_t = bEncoder.read(sent[::-1])
    intent_t = np.concatenate([fw_intent_t,bw_intent_t],axis=0)
    return intent_t
```

Task-oriented Neural Dialog System



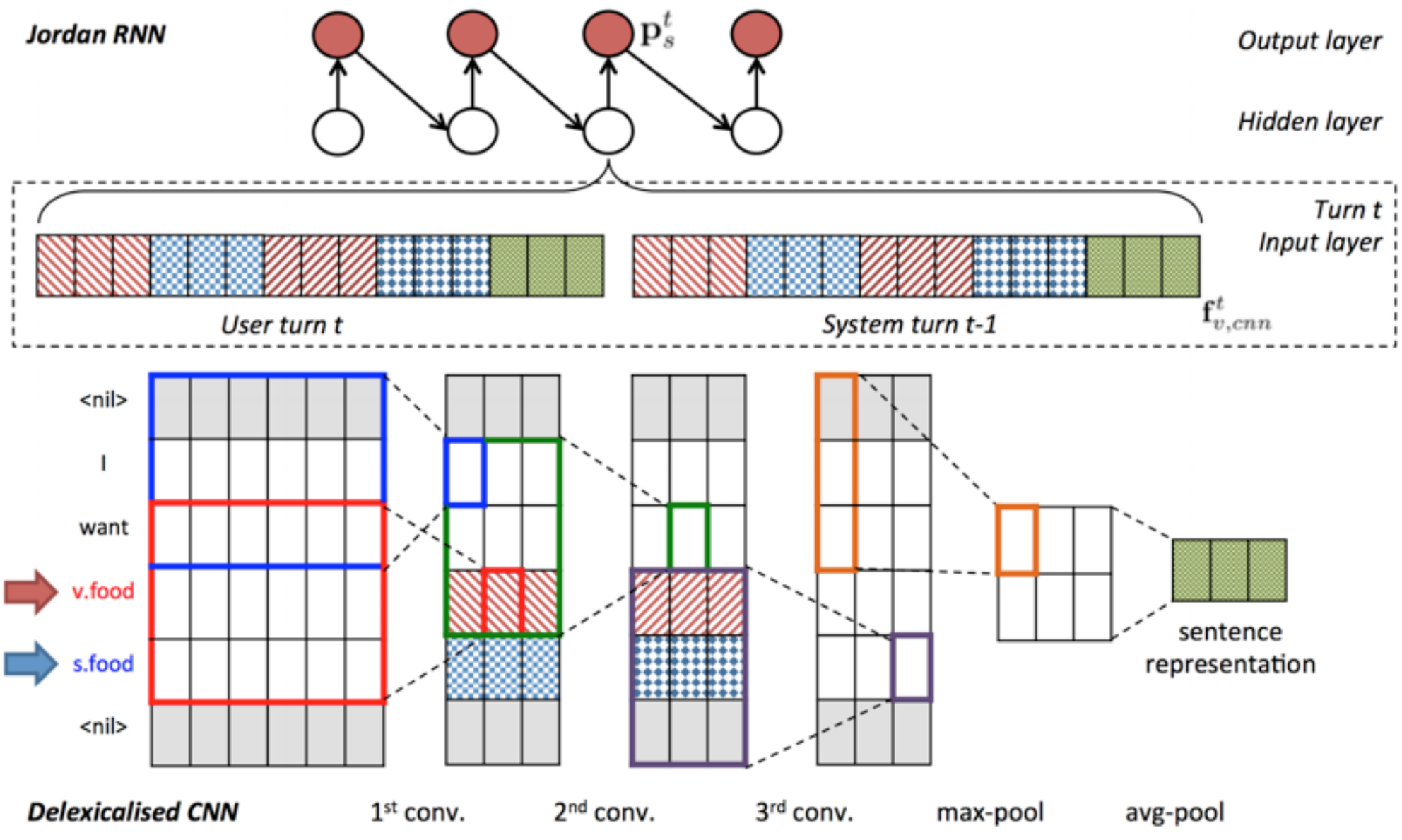
Belief Tracker

- convert user input (utterances) into a probability distribution over slot-value pairs, which is called the belief states, a multinomial distribution for each informable slot, and a binary distribution for each requestable slot.

- a set of belief trackers over a set of slot-value pairs.

- Belief tracking (also called Dialogue State tracking) provides the core of a task-oriented spoken dialogue system.

Jordan RNN--CNN Belief Trackers



Visualize CNN [13]

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	1	2	1	1	0
0	0	0	1	2	2	0
0	2	0	0	0	2	0
0	0	1	0	2	0	0
0	2	1	2	1	2	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	1	2	2	0	1	0
0	0	0	2	2	0	0
0	1	2	2	0	1	0
0	0	2	0	1	2	0
0	0	1	1	2	2	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	1	1	1	1	2	0
0	0	0	0	0	0	0
0	2	0	2	1	0	0
0	1	1	2	2	0	0
0	1	1	2	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	1	1
-1	-1	1
-1	0	0

w0[:,:,1]

-1	-1	0
-1	1	0
-1	-1	-1

w0[:,:,2]

0	1	1
0	-1	-1
0	1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	0	0
0	1	0
1	0	0

w1[:,:,1]

1	1	0
-1	1	0
-1	-1	-1

w1[:,:,2]

0	0	1
0	0	-1
0	0	-1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

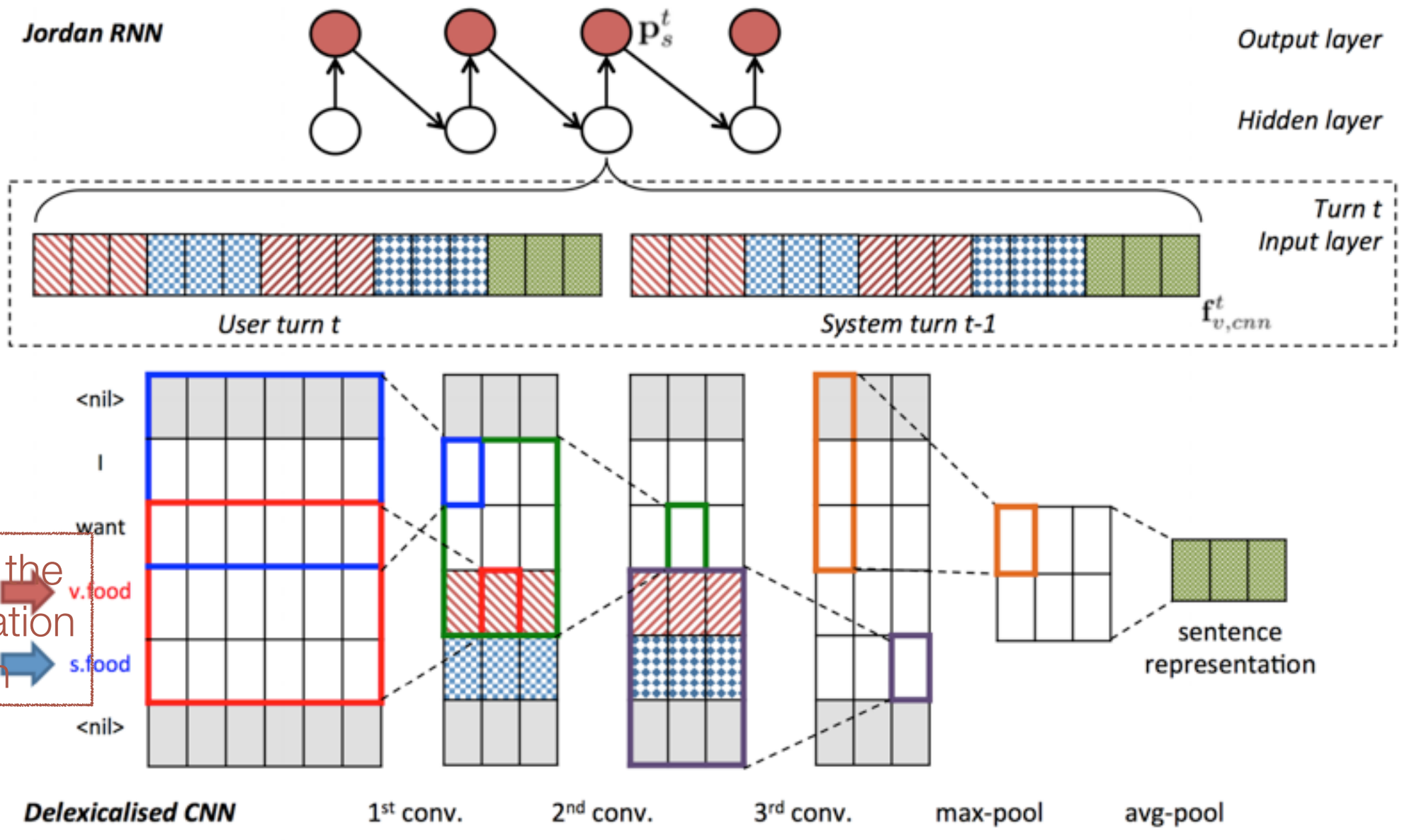
-1	-7	-6
-4	-5	-7
1	-1	-8

o[:,:,1]

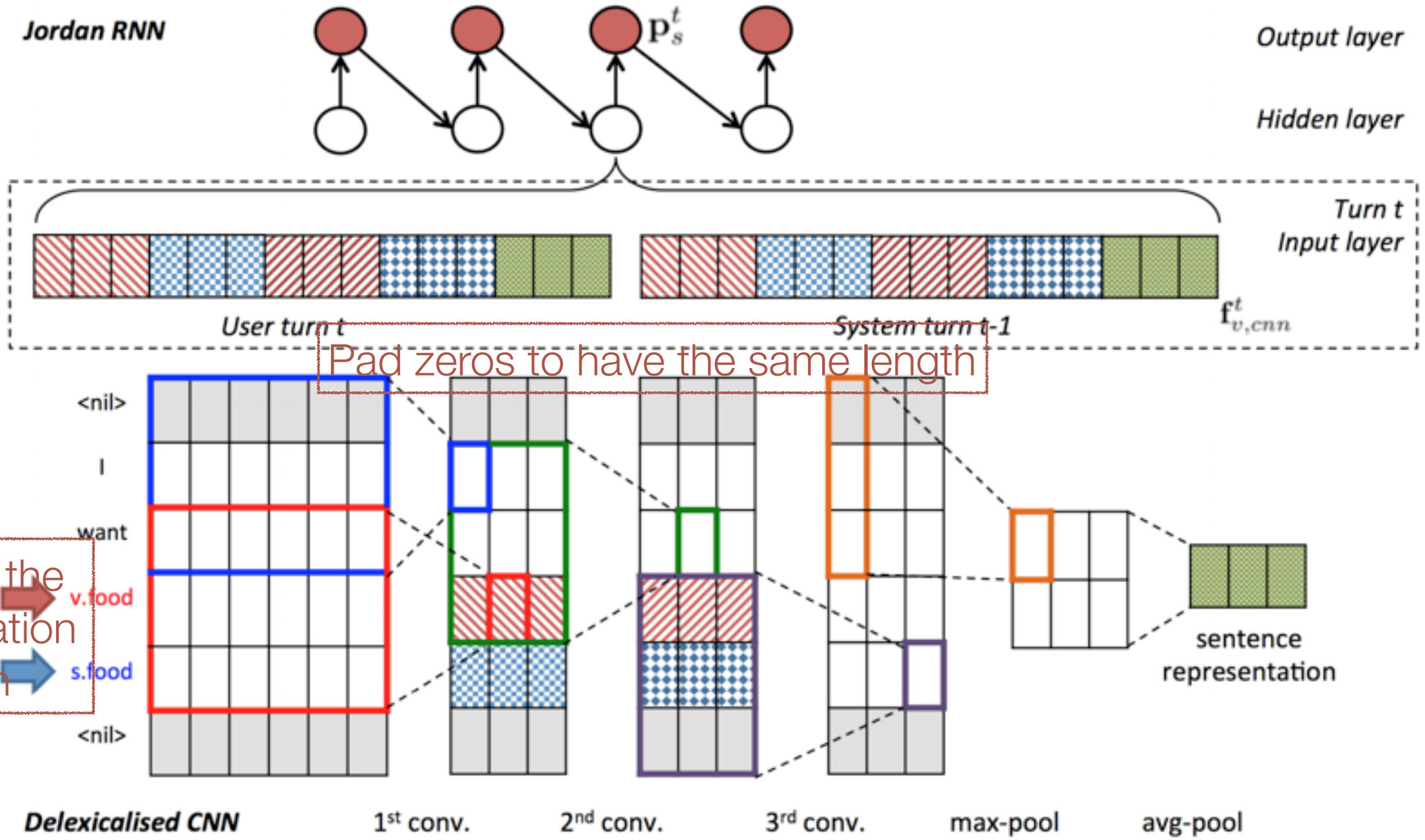
2	-3	2
0	-3	4
2	5	5

toggle movement

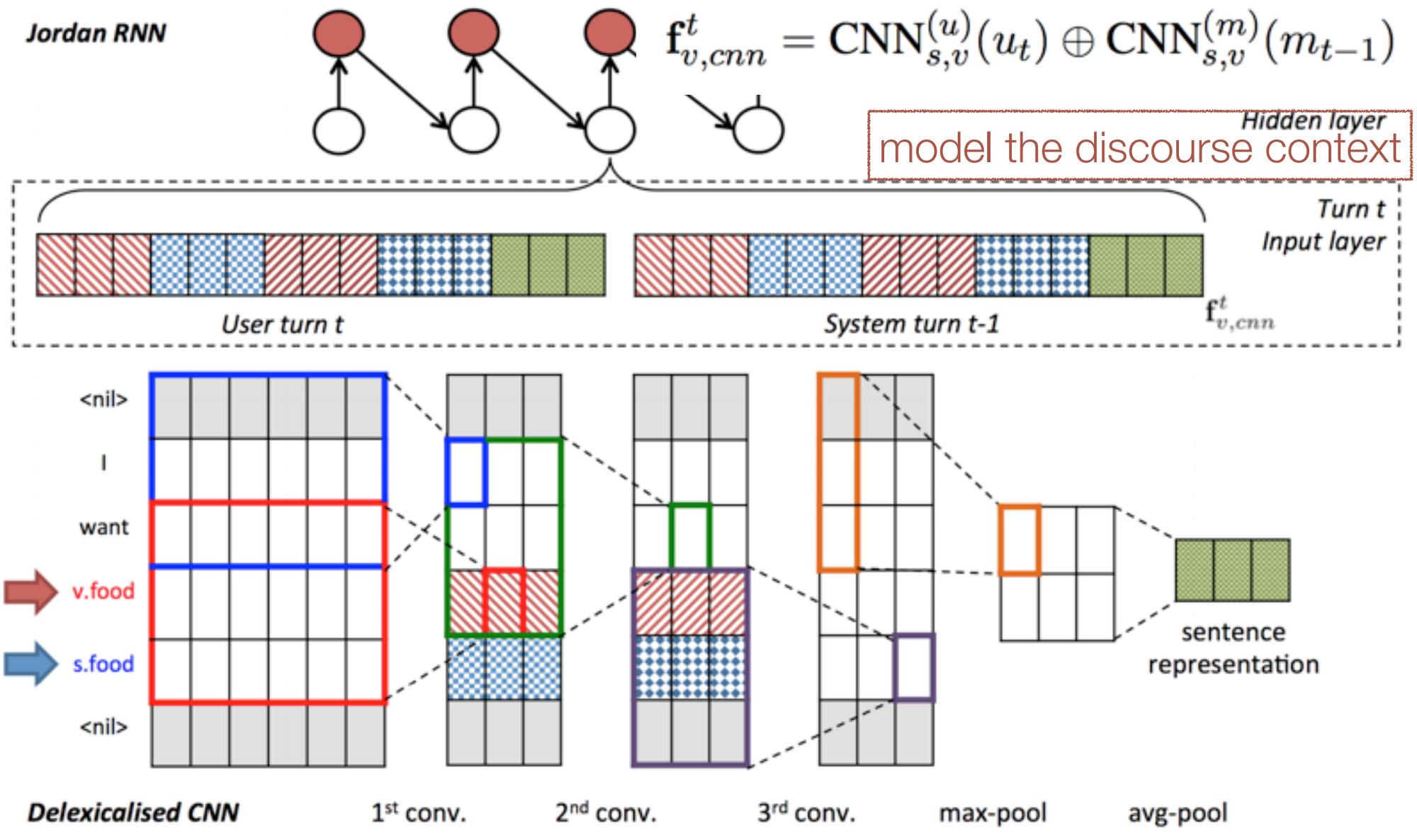
Jordan RNN--CNN Belief Trackers



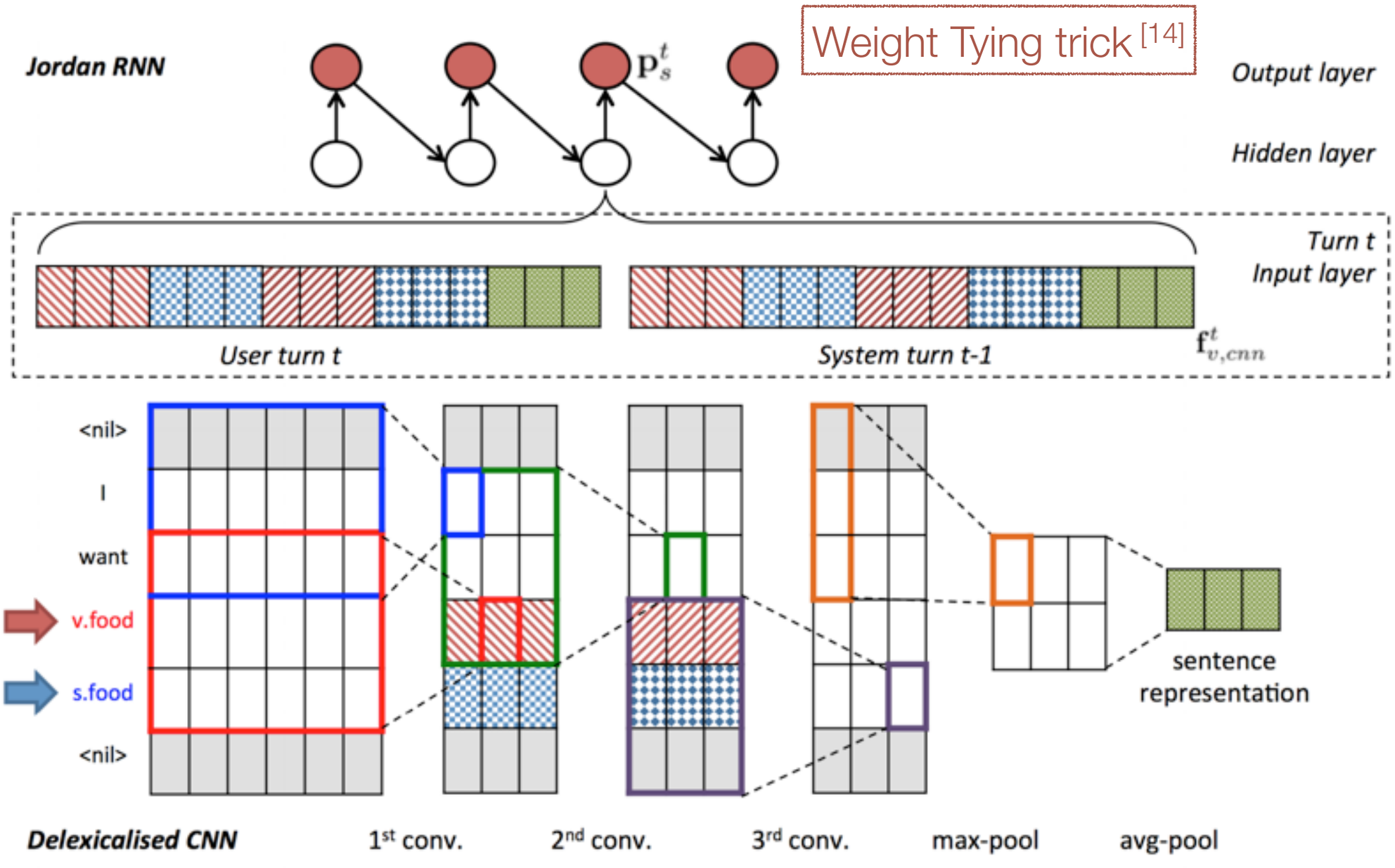
Jordan RNN--CNN Belief Trackers



Jordan RNN--CNN Belief Trackers



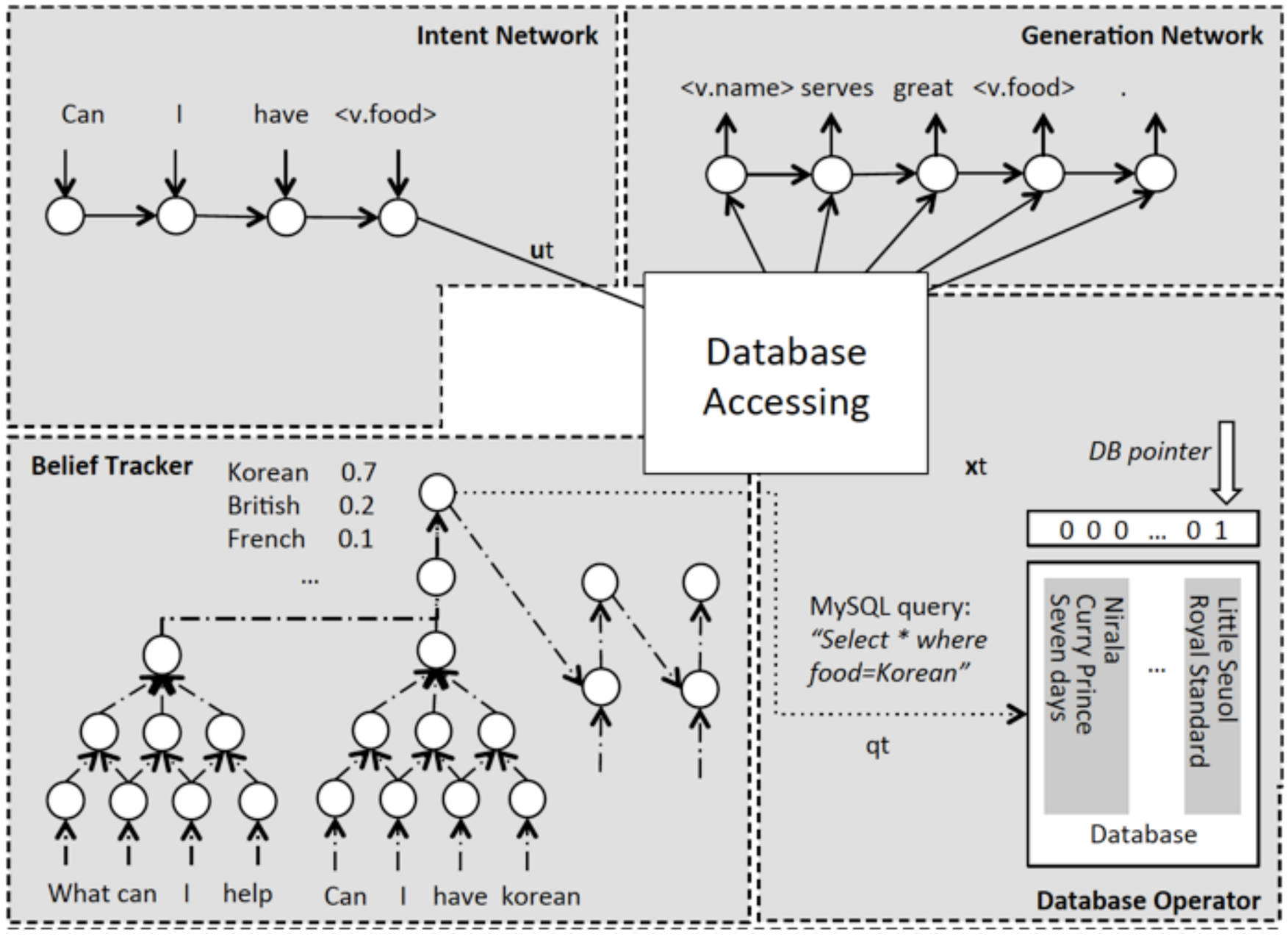
Jordan RNN--CNN Belief Trackers



Jordan RNN--CNN Belief Trackers

```
13  # Informable slot tracker
14  class CNNInformableTracker(BaseNNModule):
15
16      def __init__(self, belief_size,
17              ivocab_size, ihidden_size,
18              ovocab_size, ohidden_size):
19
20          # parameters for RNN tracker
21          # tracker specific CNN encoder
22          self.sCNN = CNNEncoder(ivocab_size, ihidden_size,
23                  pool=(False,False,True), level=3)
24          self.tCNN = CNNEncoder(ovocab_size, ihidden_size,
25                  pool=(False,False,True), level=3)
26
27          # handling feature to belief mapping
28          self.dbm1 = belief_size-1
29          self.db   = belief_size
30          self.dh   = int(belief_size/1.5)
31          self.Wfbs = theano.shared(0.3 * np.random.uniform(-1.0,1.0,\
32                  (ihidden_size*5,self.dh)).astype(theano.config.floatX))
33          self.Wfbt = theano.shared(0.3 * np.random.uniform(-1.0,1.0,\
34                  (ihidden_size*5,self.dh)).astype(theano.config.floatX))
35          self.Whb  = theano.shared(0.3 * np.random.uniform(-1.0,1.0,\
36                  (self.dh)).astype(theano.config.floatX))
```

Task-oriented Neural Dialog System

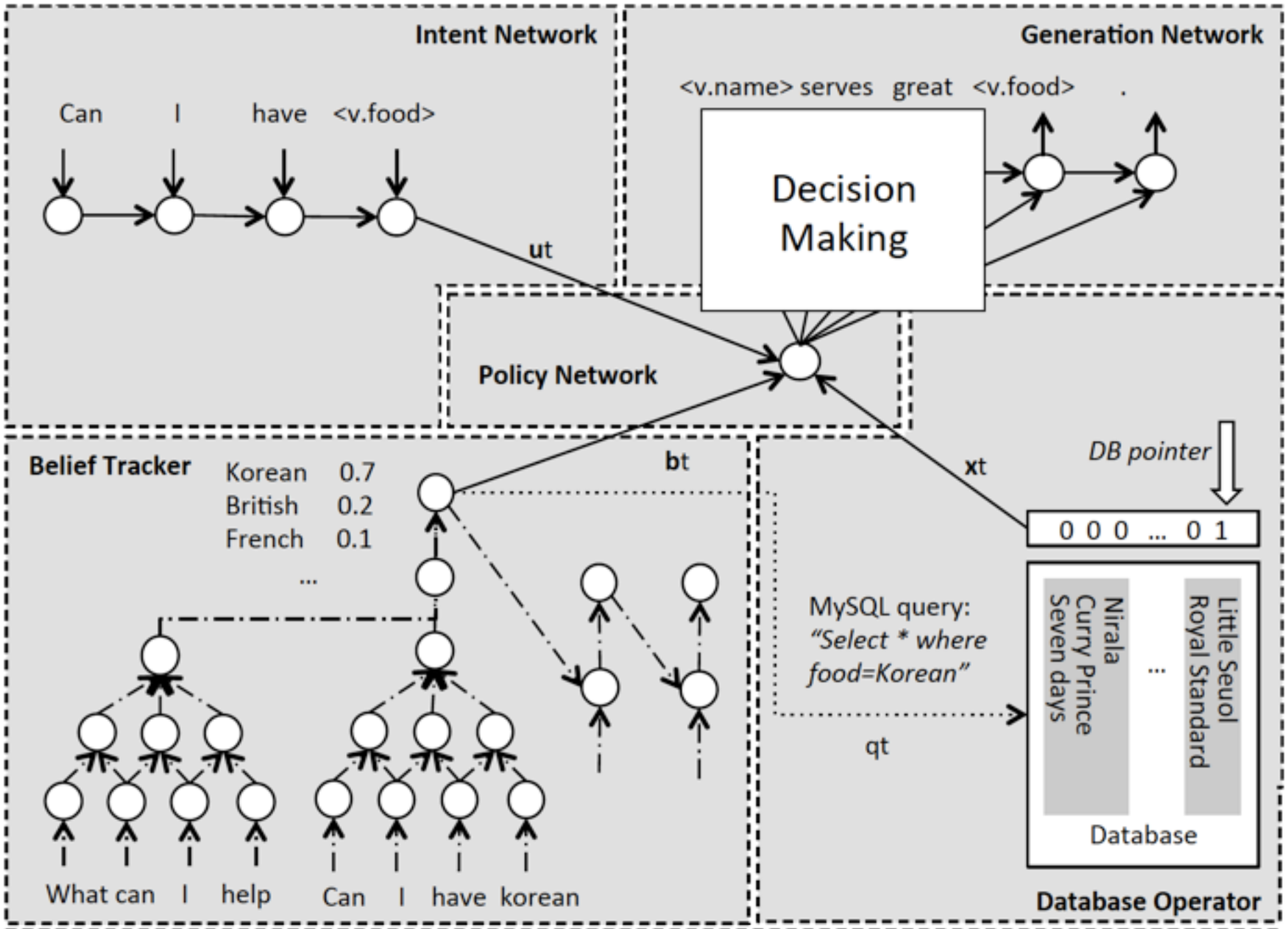


Database Operator

```
948     # search database functio
949     def _searchDB(self,b):
950
951         query = []
952         q = []
953         db_logic = []
954         # formulate query for
955         if self.trkinf==True:
956             for i in range(len(self.inf_dimensions)-1):
957                 b_i = b[self.inf_dimensions[i]:self.inf_dimensions[i+1]]
958                 idx = np.argmax(np.array(b_i)) + self.inf_dimensions[i]
959                 # ignore dont care case
960                 s2v = self.reader.infovs[idx]
961                 if '=dontcare' not in s2v and '=none' not in s2v:
962                     query.append(idx)
963                 q.append(idx)
964         # search through db by query
965         for entry in self.reader.db2inf:
966             if set(entry).issuperset(set(query)):
967                 db_logic.append(1)
968             else:
969                 db_logic.append(0)
970         # form db count features
971         dbcount = sum(db_logic)
```

```
289                     # choose venue
290                     venues = [i for i, e in enumerate(db_degree_t[:-6]) if e != 0 ]
291                     # keep the current venue
292                     if selected_venue in venues: pass
293                     else: # choose the first match as default index
294                         if len(venues)!=0:  selected_venue = random.choice(venues)
295                         # no matched venues
296                         else: selected_venue = None
```

Task-oriented Neural Dialog System



Policy Network

- can be viewed as a glue which outputs a single action vector
- an additional ScoreTable and additional matching degrees/counts

Scoring Table

Table 5: Additional R_t term for delexicalised tokens when using weighted decoding (Equation 14). *Not observed* means the corresponding tracker has a highest probability on either *not mentioned* or *dontcare* value, while *observed* mean the highest probability is on one of the categorical values. A positive score encourages the generation of that token while a negative score discourages it.

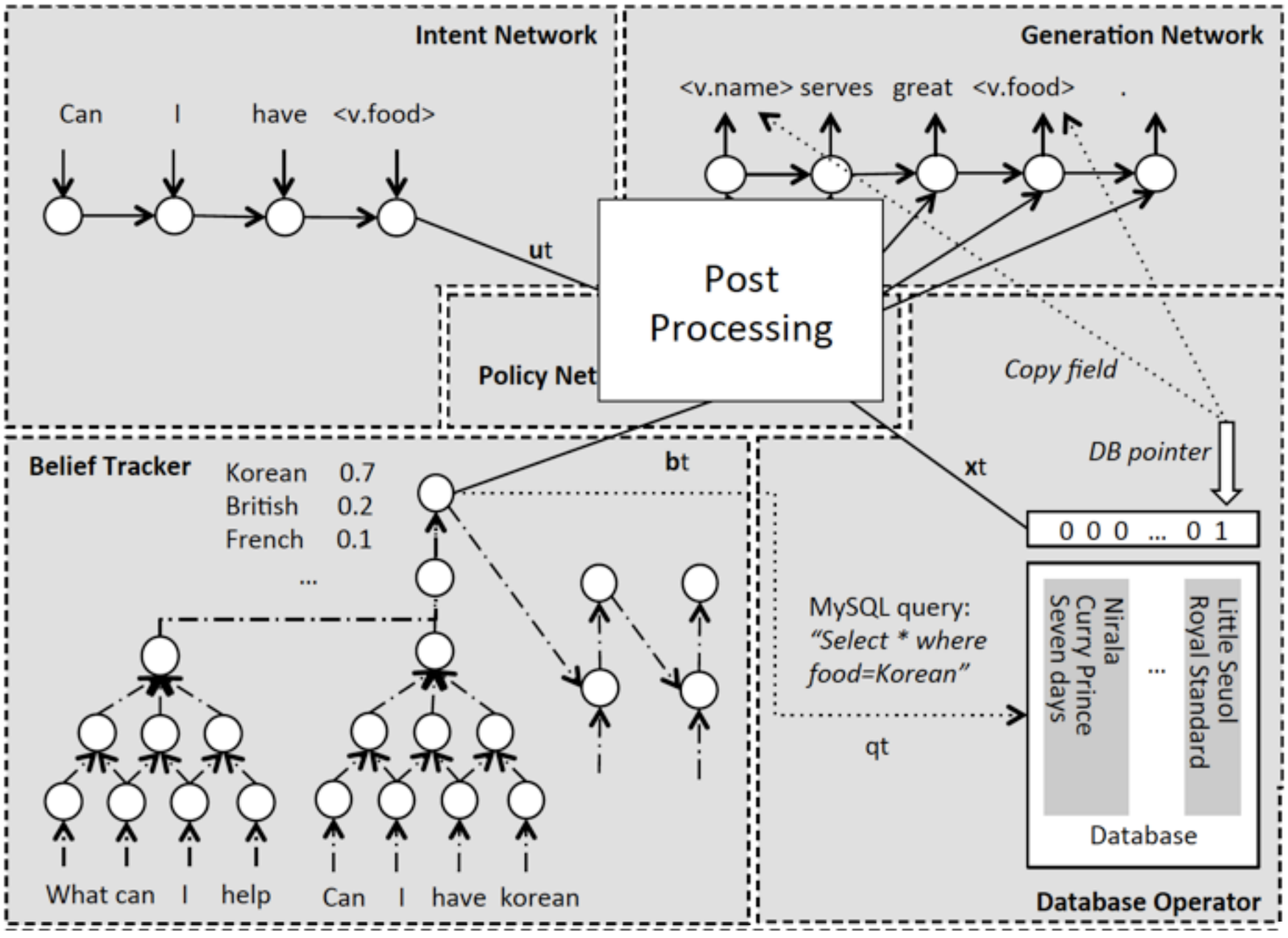
Delexicalised token	Examples	R_t (*observed*)	R_t (*not observed*)
informable slot token	<s.food>, <s.area>,...	0.0	0.0
informable value token	<v.food>, <v.area>,...	+0.05	-0.5
requestable slot token	<s.phone>,<s.address>,...	+0.2	0.0
requestable value token	<v.phone>,<v.address>,...	+0.2	0.0

Policy Network

```
1168        def _genScoreTable(self, sem_j)
1169            scoreTable = {}
1170            # requestable tracker score
1171            if self.trk=='rnn' and self.
1172                infbn = 3 if self.trkin
1173                for i in range(len(self.
1174                    bn = self.req_dimen
1175                    # prediction for th
1176                    psem = self.reader.
1177                        np.argmax(np.array(sem_j[infbn+i])) +\
1178                        self.req_dimensions[i] ]
1179                #print psem
1180                # slot & value
1181                s,v = psem.split('=')
1182                if s=='name': # skip name slot
1183                    continue
1184                # assign score, if exist, +reward
1185                score = -0.05 if v=='none' else 0.2
1186                # slot value indexing
1187                vidx = self.reader.vocab.index('[VALUE_'+s.upper()+']')
1188                sidx = self.reader.vocab.index('[SLOT_'+s.upper()+']')
1189                scoreTable[sidx] = score
1190                scoreTable[vidx] = score # reward [VALUE_****] if generate
1191        # informable tracker scoreTable
1192        if self.trk=='rnn' and self.trkinf==True:
```

```
280                 # search DB
281                 db_degree_t, query = self._searchDB(flatten_belief_t)
282                 # score table
283                 scoreTable = self._genScoreTable(full_belief_t)
284                 # generation
285                 generated,sample_t,_ = self.model.talk(
286                         masked_intent_t,belief_t, db_degree_t,
287                         masked_source_t, masked_target_t, scoreTable)
288
```

Task-oriented Neural Dialog System



Generation Network

- generates template-like sentences token by token based on the language model probabilities and the system action output by policy network.

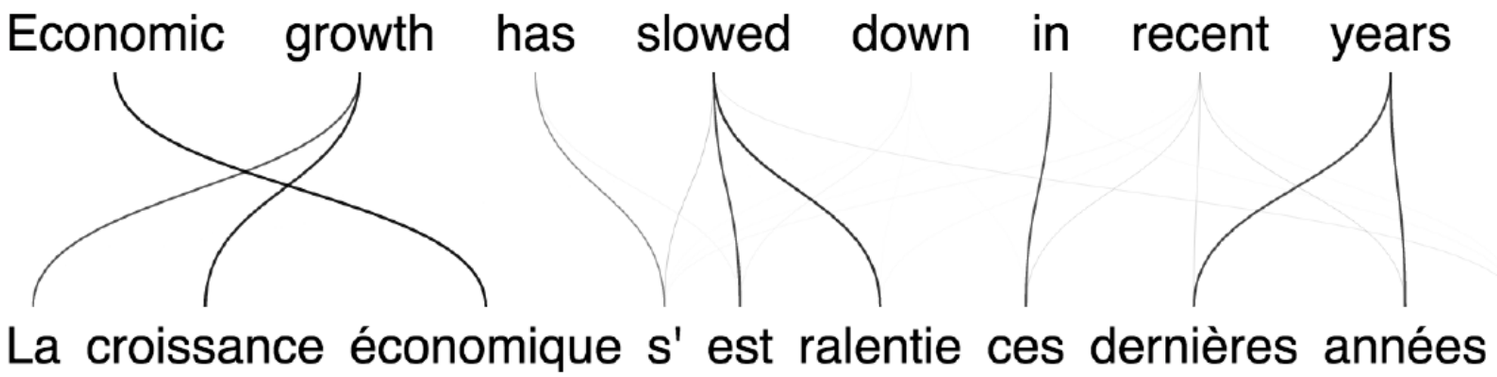
- Once the output token sequence has been generated, the generic tokens are replaced by their actual values: (1) replacing delexicalised slots by random sampling from a list of surface forms, e.g. <s.food> to food or type of food, and (2) replacing delexicalised values by the actual attribute values of the entity currently selected by the DB pointer. [7]

- attentive generation network.

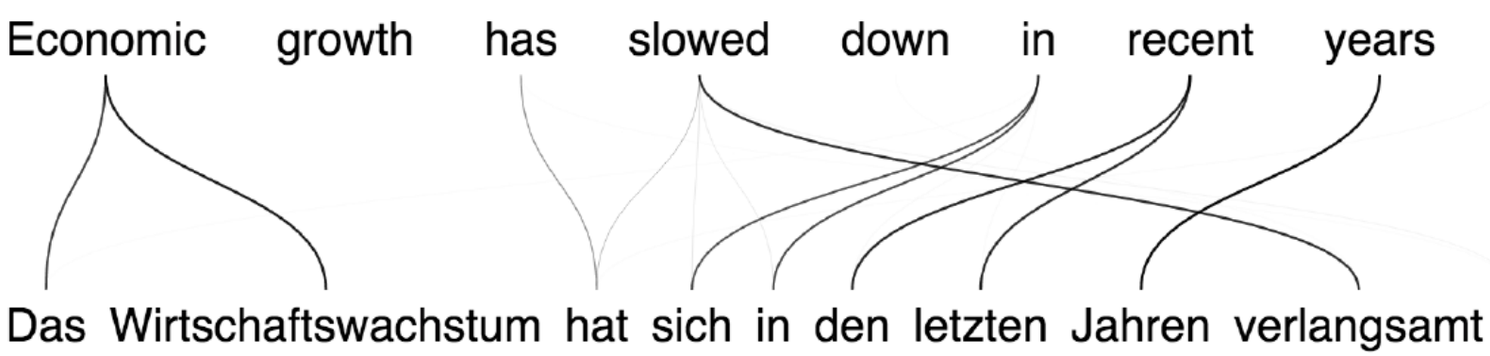
Attention-based Neural Machine Translation

- ✦ fixed representation of source sentence —> soft and dynamic[5]

English-French



English-German



Recall Attention Mechanism

- At every generation step t [5]
  - Score source h_j by

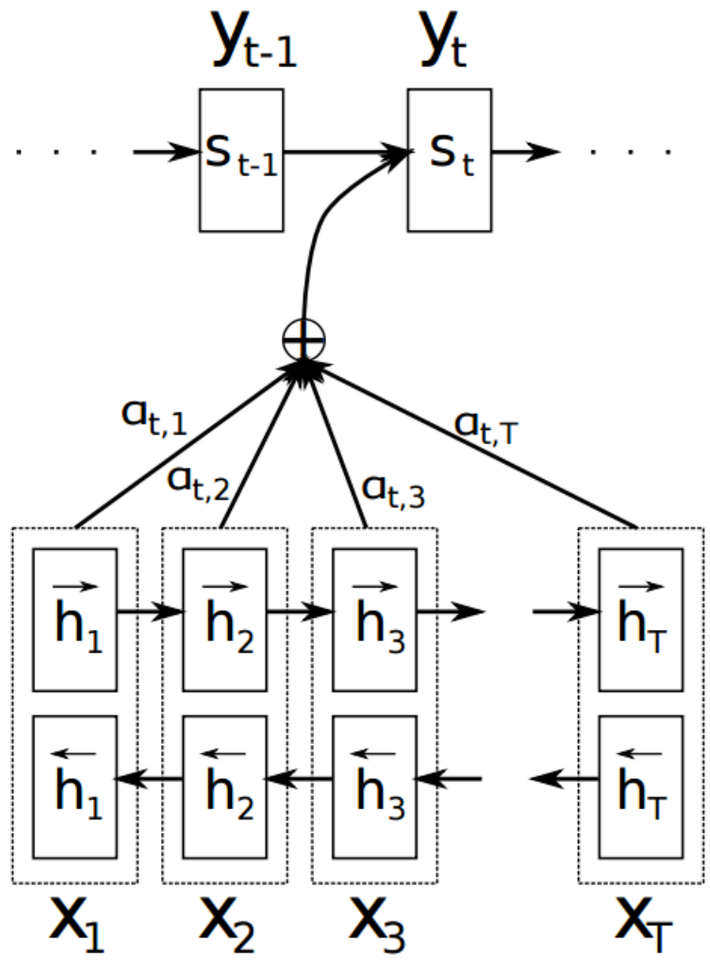
$$e_{tj} = \mathbf{v}^{\mathrm{T}} \tanh(\mathbf{W} \cdot \mathbf{s}_{t-1} + \mathbf{U} \cdot \mathbf{h}_j)$$

$$\alpha_{tj} = \mathrm{softmax}(e_{tj})$$

  - Take an expectation over sources

$$\mathbf{c}_t = \sum_j \alpha_{tj} \, \mathbf{h}_j$$

- Everything is differentiable.
  Back-prop end-to-end!



Attentive Generation Network

- use attention mechanism to combine the tracker belief states. In fact, attentive policy.

$$e_{tj} = \mathbf{v}^{\mathrm{T}} \tanh(\mathbf{W} \cdot \mathbf{s}_{t-1} + \mathbf{U} \cdot \mathbf{h}_j)$$

$$\alpha_{tj} = \mathrm{softmax}(e_{tj})$$

states, i.e. $\mathbf{o}_t$ is computed at each output step j by,

$$\mathbf{o}_t^{(j)} = \tanh(\mathbf{W}_{zo}\mathbf{z}_t + \hat{\mathbf{p}}_t^{(j)} + \mathbf{W}_{xo}\hat{\mathbf{x}}_t) \quad (10)$$

where for a given ontology $\mathbb{G}$,

$$\hat{\mathbf{p}}_t^{(j)} = \sum_{s \in \mathbb{G}} \alpha_s^{(j)} \tanh(\mathbf{W}_{po}^s \cdot \hat{\mathbf{p}}_s^t) \quad (11)$$

and where the attention weights $\alpha_s^{(j)}$ are calculated by a scoring function,

$$\alpha_s^{(j)} = \mathrm{softmax}\big(\mathbf{r}^\intercal \tanh(\mathbf{W}_r \cdot \mathbf{u}_t)\big) \quad (12)$$

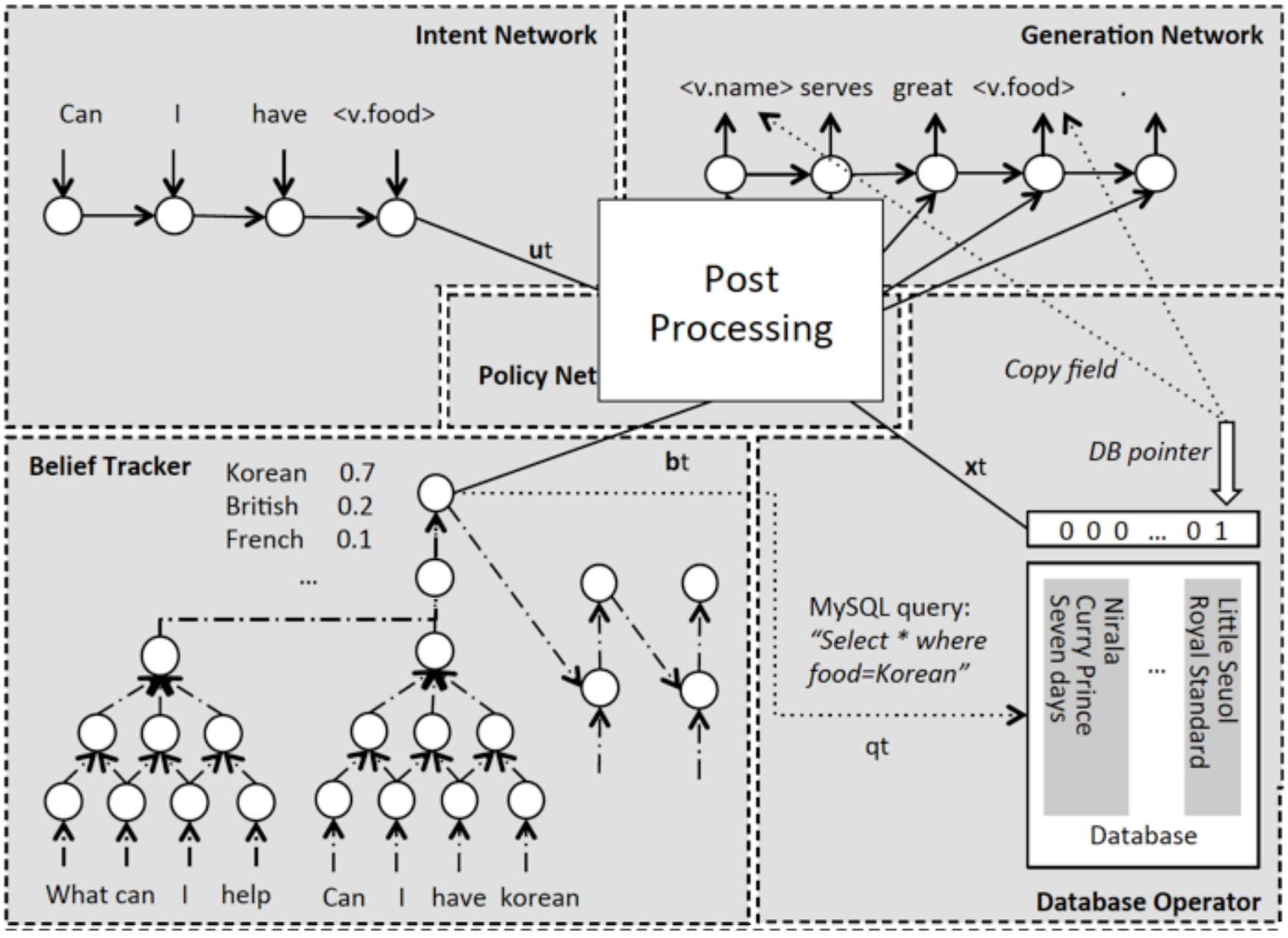
where $\mathbf{u}_t = \mathbf{z}_t \oplus \hat{\mathbf{x}}_t \oplus \hat{\mathbf{p}}_s^t \oplus \mathbf{w}_j^t \oplus \mathbf{h}_{j-1}^t$, matrix $\mathbf{W}_r$, and vector $\mathbf{r}$ are parameters to learn and $\mathbf{w}_j^t$ is the embedding of token w_j^t.

Attentive Generation Network

```
402     def prepareBelief(self,belief_t):
403         # belief vectors
404         beliefs_t = []
405         bn = 0
406         for bvec in belief_t:
407             size = bvec.shape[0]
408             beliefs_t.append( T.tanh(T.dot(bvec,self.Ws1[bn:bn+size,:])).\
409                     dimshuffle('x',0) )
410             bn += size
411         beliefs_t = T.concatenate(beliefs_t,axis=0)
412         return beliefs_t
```

```
209     # decoding function
210     def decode(self, masked_source_t, masked_source_len_t,
211             masked_target_t, masked_target_len_t,
212             intent_t, belief_t, degree_t, utt_group_t,
213             snapshot_t, sample_t):
214         # decide policy type
215         if self.ply=='attention':
216             # set dummpy, not used
217             actEmb_t = T.concatenate(belief_t,axis=0)
218             prior_t = posterior_t = z_t = base_t = debug_t = None
219             # attentive policy
220             belief_t = self.policy.prepareBelief(belief_t)
```

Task-oriented Neural Dialog System



Task-oriented Neural Dialog System

```
248             # for each turn
249             reqs = []
250             generated_utt_tm1 = ''
251             for t in range(len(source)):
252                 if self.verbose>0:
253                     print '-'*28 + ' Turn '+ str(t) +' '+ '-'*28
254                 # extract source and target sentence for that turn
255                 source_t        = source[t][:source_len[t]]
256                 masked_source_t = masked_source[t][:masked_source_len[t]]
257                 masked_target_t = masked_target[t][:masked_target_len[t]]
258                 # this turn features
259                 srcfeat_t   = srcfeat[t]
260
261                 # previous target
262                 masked_target_tm1, target_tm1, starpos_tm1, vtarpos_tm1, offer = \
263                     self.reader.extractSeq(generated_utt_tm1,type='target')
264
265                 tarfeat_tm1 = [starpos_tm1,vtarpos_tm1]
266
267                 # utterance preparation
268                 source_utt = ' '.join([self.reader.vocab[w] for w in source_t])
269                 masked_source_utt= ' '.join([self.reader.vocab[w]
270                         for w in masked_source_t])
271                 masked_target_utt= ' '.join([self.reader.vocab[w]
272                         for w in masked_target_t])
273
```

Task-oriented Neural Dialog System

```
# read and understand user sentence
masked_intent_t = self.model.read(masked_source_t)
full_belief_t, belief_t = self.model.track(
        flatten_belief_tm1, masked_source_t, masked_target_tm1,
        srcfeat_t, tarfeat_tm1 )
flatten_belief_t = np.concatenate(full_belief_t,axis=0)
# search DB
db_degree_t, query = self._searchDB(flatten_belief_t)
# score table
scoreTable = self._genScoreTable(full_belief_t)
# generation
generated,sample_t,_ = self.model.talk(
        masked_intent_t,belief_t, db_degree_t,
        masked_source_t, masked_target_t, scoreTable)

# choose venue
venues = [i for i, e in enumerate(db_degree_t[:-6]) if e != 0 ]
# keep the current venue
if selected_venue in venues: pass
else: # choose the first match as default index
    if len(venues)!=0:  selected_venue = random.choice(venues)
    # no matched venues
    else: selected_venue = None
```

Intent Network

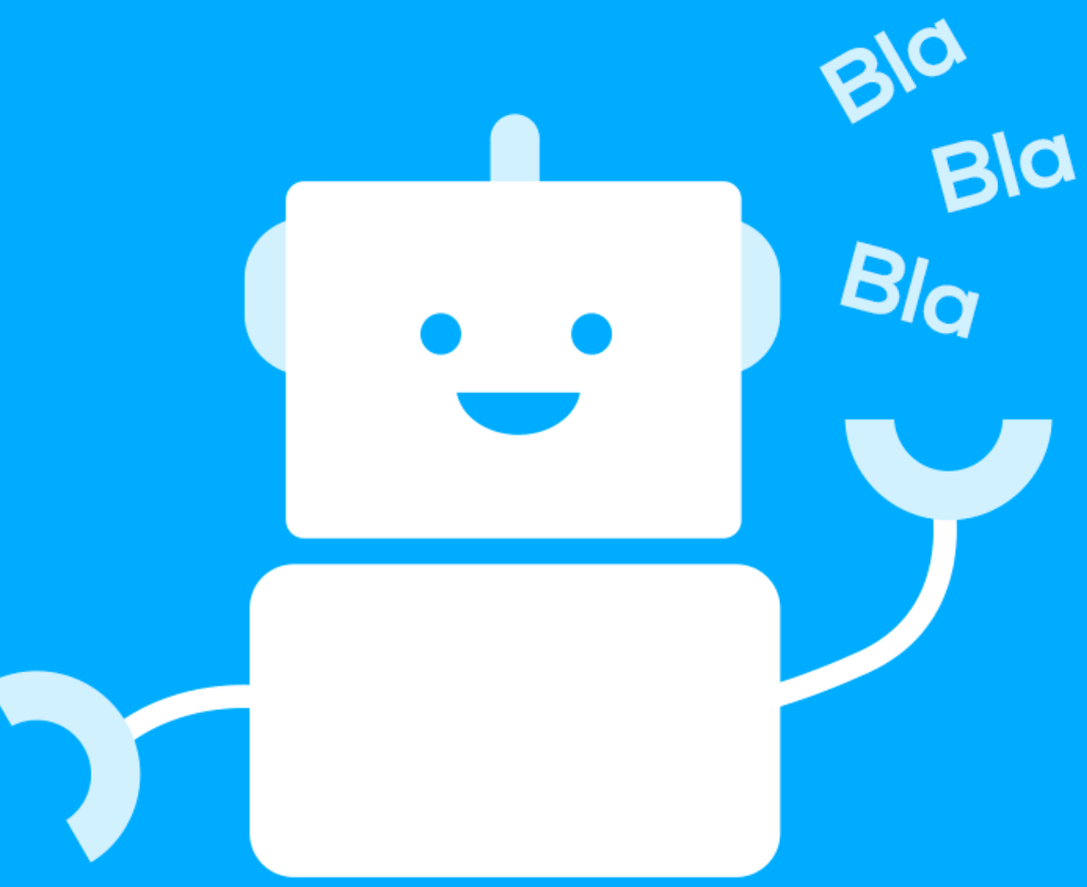
Belief Tracker

DB Pointer

Policy Network and Generation Network

Policy Network and Generation Network

```
298                 # lexicalise generated utterance
299                 generated_utts = []
300                 for gen in generated:
301                     generated_utt = ' '.join([self.reader.vocab[g] for g in gen[0]])
302                     generated_utts.append(generated_utt)
303                 gennerated_utt = generated_utts[0]
304
305                 # calculate semantic match rate
306                 twords = [self.reader.vocab[w] for w in masked_target_t]
307                 for gen in generated:
308                     gwords = [self.reader.vocab[g] for g in gen[0]]
309                     for gw in gwords:
310                         if gw.startswith('[VALUE_') or gw.startswith('[SLOT_'):
311                             if gw in twords: # match target semi token
312                                 stats['approp'][0] += 1.0
313                                 stats['approp'][1] += 1.0
314                     #gstats += np.mean(np.array(gen[2][1:]),axis=0 )
315                     num_sent += 1
316
317                 # update history belief
318                 flatten_belief_tm1 = flatten_belief_t[:self.inf_dimensions[-1]]
319
320                 # for calculating success: check requestable slots match
321                 requestables = ['phone','address','postcode','food','area','pricerange']
322                 for requestable in requestables:
323                     if '[VALUE_'+requestable.upper()+']' in gennerated_utt:
324                         reqs.append(self.reader.reqs.index(requestable+'=exist'))
325                 # check offered venue
326                 if '[VALUE_NAME]' in generated_utt and selected_venue!=None:
327                     venue_offered = self.reader.db2inf[selected_venue]
```



VS.

Task-oriented | Dataset

CamRest676 Dataset

- https://www.repository.cam.ac.uk/handle/1810/260970
- Cambridge restaurant dialogue domain. In-domain human-human conversation.
- Users can look for restaurant with particular food type, price range, and area (informable), and ask for phone number, address and postcode (requestable).
- Including coarse slot-value labels per turn.
- Evaluation: *Dialogue success, BLEU

CamRest676 Dataset

Task:
Find a ~~**restaurant**~~,
~~**cheap**~~, ~~**Korean**~~, ~~**North**~~
Ask ~~**phone number**~~

*Hi, I want a **cheap Korean restaurant**.*

*What **area** are you looking for ?*

*Somewhere in the **north**.*

***Little Seoul** is nice one in the north.*

*What is the **phone number**?*

*Its phone number is **01223456789**.*

Thank you very much, good bye.

Thank you for using the system.

What user wants?	
Food	Korean
Price	Cheap
Area	North

Search Table	
Little Seoul	...

Dataset Statistics

- Ontology:
  - Cambridge restaurant domain, 99 venues.
  - 3 informable slots: area, price range, food type
  - 3 requestable slots: address, phone, postcode
- Dataset:
  - 676 dialogues, ~2750 turns
  - 3000 HITS, takes 3 days, costs ~400 USD
  - Data cleaning takes 2-3 days for one person

Training Details

- Two stage training:
  - Training trackers with label cross entropy
  - Training other parts with response cross entropy
- Decoding
  - Beam search w/ beam width 10
  - Decode with average word likelihood

Experiment Results

Table 1: Tracker performance in terms of Precision, Recall, and F-1 score.

Tracker type	Informable			Requestable		
	Prec.	Recall	F-1	Prec.	Recall	F-1
cnn	99.77%	96.09%	97.89%	98.66%	93.79%	96.16%
ngram	99.34%	94.42%	96.82%	98.56%	90.14%	94.16%

Table 3: Human assessment of the NN system. The rating for comprehension/naturalness are both out of 5.

Metric	NN
Success	98%
Comprehension	4.11
Naturalness	4.05
# of dialogues:	245

Example Dialogues

Human	: hello
Machine	: what cuisine would you like ?
Human	: afghan
Machine	: i am sorry , there are no afghan restaurants . would you like something else ?
Human	: korean
Machine	: little seoul serves korean food , it is located at 108 regent street city centre .
Human	: is it expensive?
Machine	: little seoul serves korean food and is in the expensive price range .
Human	: what area is that
Machine	: little seoul is located in the centre area of town . would you like their number ?
Human	: yes
Machine	: the telephone number is 01223 308681 . is there anything else i can help you with today ?
Human	: no goodbye
Machine	: thank you for using the cambridge restaurant system , goodbye .

Homework

- Paper Reading:
  - Lifeng Shang, Zhengdong Lu, Hang Li. “Neural Responding Machine for Short-Text Conversation”. EMNLP 2015.
  - Zongcheng Ji, Zhengdong Lu, Hang Li. “An Information Retrieval Approach to Short Text Conversation”. arXiv preprint 2014.

- Code Reading:
  - Multi-turn Response Selection: https://github.com/MarkWuNLP/MultiTurnResponseSelection
  - A Neural Conversational Model: https://github.com/Conchylicultor/DeepQA
  - Attention-based Conversational Model: https://github.com/pbhatia243/Neural_Conversation_Models

Thanks for your attention!

Q&A

References

- [1] Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." International Conference on Machine Learning. 2013.
- [2] Mikolov, Tomas, et al. "Recurrent neural network based language model." Interspeech. Vol. 2. 2010.
- [3] Colah. “Understanding LSTMs”. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- [4] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.
- [5] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).
- [6] Vinyals, Oriol, and Quoc Le. "A neural conversational model." arXiv preprint arXiv:1506.05869 (2015).
- [7] Wen, Tsung-Hsien, et al. "A network-based end-to-end trainable task-oriented dialogue system." EACL 2017.
- [8] Bowman, Samuel R., et al. "Generating sentences from a continuous space." arXiv preprint arXiv: 1511.06349 (2015).

References

- [9] Hu, Baotian, et al. "Convolutional neural network architectures for matching natural language sentences." Advances in neural information processing systems. 2014.

- [10] Qiu, Xipeng, and Xuanjing Huang. "Convolutional Neural Tensor Network Architecture for Community-Based Question Answering." IJCAI. 2015.

- [11] Wu, Yu, et al. "Sequential Match Network: A New Architecture for Multi-turn Response Selection in Retrieval-based Chatbots." arXiv preprint arXiv:1612.01627 (2016).

- [12] Matthew Henderson, Blaise Thomson, and Steve Young. 2014. Robust dialog state tracking using delexicalised recurrent neural networks and unsupervised adaptation. In Proceedings of IEEE Spoken Language Technology.

- [13] http://cs231n.github.io/convolutional-networks/

- [14] Nikola Mrkšic, Diarmuid Ó Séaghdha, Blaise Thomson, ´ Milica Gašic, Pei-Hao Su, David Vandyke, Tsung- ´ Hsien Wen, and Steve Young. 2015. Multi-domain dialog state tracking using recurrent neural networks. In ACL, pages 794–799, Beijing, China, July. ACL.