Word-Based Dialog State Tracking with Recurrent Neural Networks

Matthew Henderson, Blaise Thomson and Steve Young
Department of Engineering,
University of Cambridge, U.K.
{mh521, brmt2, sjy}@eng.cam.ac.uk

Abstract

Recently discriminative methods for tracking the state of a spoken dialog have been shown to outperform traditional generative models. This paper presents a new word-based tracking method which maps directly from the speech recognition results to the dialog state without using an explicit semantic decoder. The method is based on a recurrent neural network structure which is capable of generalising to unseen dialog state hypotheses, and which requires very little feature engineering. The method is evaluated on the second Dialog State Tracking Challenge (DSTC2) corpus and the results demonstrate consistently high performance across all of the metrics.

1 Introduction

While communicating with a user, statistical spoken dialog systems must maintain a distribution over possible dialog states in a process called *dialog state tracking*. This distribution, also called the *belief state*, directly determines the system's decisions. In MDP-based systems, only the most likely dialog state is considered and in this case the primary metric is dialog state accuracy (Bohus and Rudnicky, 2006). In POMDP-based systems, the full distribution is considered and then the shape of the distribution as measured by an L2 norm is equally important (Young et al., 2009). In both cases, good quality state tracking is essential to maintaining good overall system performance.

Typically, state tracking has assumed the output of a Spoken Language Understanding (SLU) component in the form of a semantic decoder, which maps the hypotheses from Automatic Speech Recognition (ASR) to a list of semantic hypotheses. This paper considers mapping directly from ASR hypotheses to an updated belief state at each turn in the dialog, omitting the intermediate SLU processing step. This *word-based* state tracking avoids the need for an explicit semantic representation and also avoids the possibility of information loss at the SLU stage.

Recurrent neural networks (RNNs) provide a natural model for state tracking in dialog, as they are able to model and classify dynamic sequences with complex behaviours from step to step. Whereas, most previous approaches to discriminative state tracking have adapted stationary classifiers to the temporal process of dialog (Bohus and Rudnicky, 2006; Lee and Eskenazi, 2013; Lee, 2013; Williams, 2013; Henderson et al., 2013b). One notable exception is Ren et al. (2013), which used conditional random fields to model the sequence temporally.

Currently proposed methods of discriminative state tracking require engineering of feature functions to represent the turn in the dialog (Ren et al., 2013; Lee and Eskenazi, 2013; Lee, 2013; Williams, 2013; Henderson et al., 2013b). It is unclear whether differences in performance are due to feature engineering or the underlying models. This paper proposes a method of using simple n-gram type features which avoid the need for feature engineering. Instead of using inputs with a select few very informative features, the approach is to use high-dimensional inputs with all the information to potentially reconstruct any such hand-crafted feature. The impact of significantly increasing the dimensionality of the inputs is managed by careful initialisation of model parameters.

Accuracy on unseen or infrequent slot values is an important concern, particularly for discriminative classifiers which are prone to overfitting training data. This is addressed by structuring the recurrent neural network to include a component which is independent of the actual slot value in question. It thus learns general behaviours for specifying slots enabling it to successfully decode

ASR output which includes previously unseen slot values.

In summary, this paper presents a word-based approach to dialog state tracking using recurrent neural networks. The model is capable of generalising to unseen dialog state hypotheses, and requires very little feature engineering. The approach is evaluated in the second Dialog State Tracking Challenge (DSTC2) (Henderson et al., 2014) where it is shown to be extremely competitive, particularly in terms of the quality of its confidence scores.

Following a brief outline of DSTC2 in section 2, the definition of the model is given in section 3. Section 4 then gives details on the initialisation methods used for training. Finally results on the DSTC2 evaluation are given in 5.

2 The Second Dialog State Tracking Challenge

This section describes the domain and methodology of the second Dialog State Tracking Challenge (DSTC2). The challenge is based on a large corpus collected using a variety of telephone-based dialog systems in the domain of finding a restaurant in Cambridge. In all cases, the subjects were recruited using Amazon Mechanical Turk.

The data is split into a train, dev and test set. The train and dev sets were supplied with labels, and the test set was released unlabelled for a one week period. At the end of the week, all participants were required to submit their trackers' output on the test set, and the labels were revealed. A mis-match was ensured between training and testing conditions by choosing dialogs for the evaluation collected using a separate dialog manager. This emulates the mis-match a new tracker would encounter if it were actually deployed in an end-to-end system.

In summary, the datasets used are:

- **dstc2_train** - Labelled training consisting of 1612 dialogs with two dialog managers and two acoustic conditions.
- **dstc2_dev** - Labelled dataset consisting of 506 calls in the same conditions as dstc2_train, but with no caller in common.
- **dstc2_test** - Evaluation dataset consisting of 1117 dialogs collected using a dialog manager not seen in the labelled data.

In contrast with DSTC1, DSTC2 introduces dynamic user goals, tracking of requested slots and tracking the restaurant search method. A DSTC2 tracker must therefore report:

- **Goals**: A distribution over the user's goal for each slot. This is a distribution over the possible values for that slot, plus the special value *None*, which means no valid value has been mentioned yet.
- **Requested slots**: A reported probability for each requestable slot that has been requested by the user, and should be informed by the system.
- **Method**: A distribution over methods, which encodes how the user is trying to use the dialog system. E.g. 'by constraints', when the user is trying to constrain the search, and 'finished', when the user wants to end the dialog.

A tracker may report the goals as a joint over all slots, but in this paper the joint is reported as a product of the marginal distributions per slot.

Full details of the challenge are given in Henderson et al. (2013a), Henderson et al. (2014). The trackers presented in this paper are identified under 'team4' in the reported results.

3 Recurrent Neural Network Model

This section defines the RNN structure used for dialog state tracking. One such RNN is used per slot, taking the most recent dialog turn (user input plus last machine dialog act) as input, updating its internal memory and calculating an updated belief over the values for the slot. In what follows, the notation $\mathbf{a} \oplus \mathbf{b}$ is used to denote the concatenation of two vectors, $\mathbf{a}$ and $\mathbf{b}$. The i^{th} component of the vector $\mathbf{a}$ is written $\mathbf{a}|_i$.

3.1 Feature Representation

Extracting n-grams from utterances and dialog acts provides the feature representations needed for input into the RNN. This process is very similar to the feature extraction described in Henderson et al. (2012), and is outlined in figure 1.

For n-gram features extracted from the ASR N-best list, unigram, bigram and trigram features are calculated for each hypothesis. These are then weighted by the N-best list probabilities and summed to give a single vector.

Dialog acts in this domain consist of a list of component acts of the form `acttype(slot=value)` where the `slot=value` pair is optional. The n-gram type features extracted from each such component act are `'acttype'`, `'slot'`, `'value'`, `'acttype slot'`, `'slot value'` and `'acttype slot value'`, or just `'acttype'` for the act `acttype()`. Each feature is given weight 1, and the features from individual component acts are summed.

To provide a contrast, trackers have also been implemented using the user dialog acts output by an SLU rather than directly from the ASR output. In this case, the SLU N-best dialog act list is encoded in the same way except that the n-grams from each hypothesis are weighted by the corresponding probabilities, and summed to give a single feature vector.

Consider a word-based tracker which takes an ASR N-best list and the last machine act as input for each turn, as shown in figure 1. A combined feature representation of both the ASR N-best list and the last machine act is obtained by concatenating the vectors. This means that in figure 1 the food feature from the ASR and the `food` feature from the machine act contribute to separate components of the final vector $\mathbf{f}$.

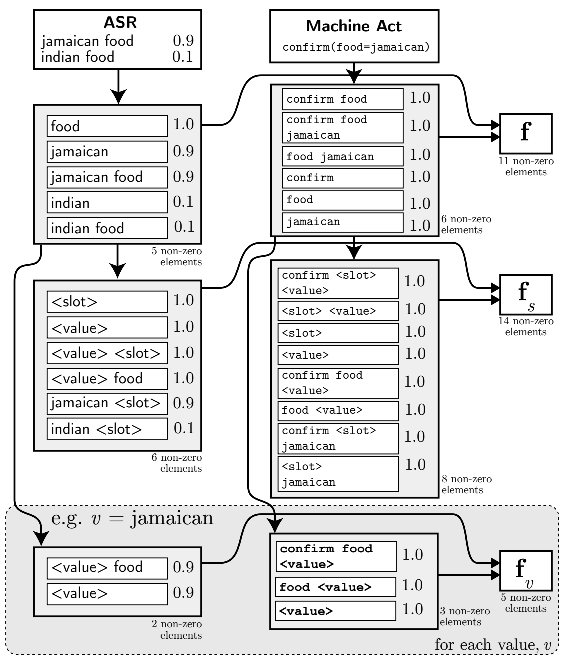

Figure 1: Example of feature extraction for one turn, giving $\mathbf{f}$, $\mathbf{f}_s$ and $\mathbf{f}_v$. Here s = food. For all $v \notin$\{indian, jamaican\}, $\mathbf{f}_v = \mathbf{0}$.

Note that all the methods for tracking reported in DSTC1 required designing feature functions. For example, suggested feature functions included the SLU score in the current turn, the probability of an 'affirm' act when the value has been confirmed by the system, the output from baseline trackers etc. (e.g. Lee and Eskenazi (2013), Williams (2013), Henderson et al. (2013b)). In contrast, the approach described here is to present the model with all the information it would need to reconstruct any feature function that might be useful.

3.2 Generalisation to Unseen States

One key issue in applying machine learning to the task of dialog state tracking is being able to deal with states which have not been seen in training. For example, the system should be able to recognise any obscure food type which appears in the set of possible food types. A naïve neural network structure mapping n-gram features to an updated distribution for the food slot, with no tying of weights, would require separate examples of each of the food types to learn what n-grams are associated with each. In reality however n-grams like '<value> food' and 'serving <value>' are likely to correspond to the hypothesis food='<value>' for any food-type replacing '<value>'.

The approach taken here is to embed a network which learns a generic model of the updated belief of a slot-value assignment as a function of 'tagged' features, i.e. features which ignore the specific identity of a value. This can be considered as replacing all occurrences of a particular value with a tag like '<value>'. Figure 1 shows the process of creating the tagged feature vectors, $\mathbf{f}_s$ and $\mathbf{f}_v$ from the untagged vector $\mathbf{f}$.

3.3 Model Definition

In this section an RNN is described for tracking the goal for a given slot, s, throughout the sequence of a dialog. The RNN holds an internal memory, $\mathbf{m} \in \mathbb{R}^{N_{\text{mem}}}$ which is updated at each step. If there are N possible values for slot s, then the probability distribution output $\mathbf{p}$ is in $\mathbb{R}^{N+1}$, with the last component $\mathbf{p}|_N$ giving the probability of the *None* hypothesis. Figure 2 provides an overview of how $\mathbf{p}$ and $\mathbf{m}$ are updated in one turn to give the new belief and memory, $\mathbf{p}'$ and $\mathbf{m}'$.

One part of the neural network is used to learn a mapping from the untagged inputs, full memory and previous beliefs to a vector $\mathbf{h} \in \mathbb{R}^N$ which goes directly into the calculation of $\mathbf{p}'$:

$$\mathbf{h} \;=\; \text{NNet}\left(\mathbf{f} \oplus \mathbf{p} \oplus \mathbf{m}\right) \in \mathbb{R}^N$$

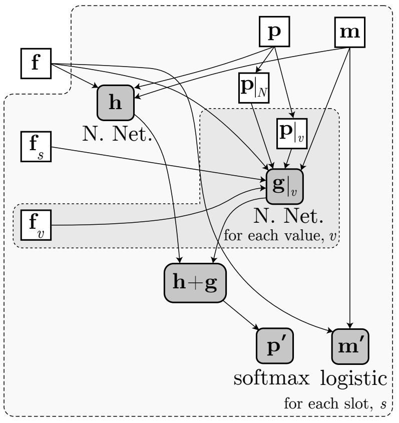

Figure 2: Calculation of $\mathbf{p}'$ and $\mathbf{m}'$ for one turn

where NNet($\cdot$) denotes a neural network function of the input. In this paper all such networks have one hidden layer with a sigmoidal activation function.

The sub-network for $\mathbf{h}$ requires examples of every value in training, and is prone to poor generalisation as explained in section 3.2. By including a second sub-network for $\mathbf{g}$ which takes tagged features as input, it is possible to exploit the observation that the string corresponding to a value in various contexts is likely to be good evidence for or against that value. For each value v, a component of $\mathbf{g}$ is calculated using the neural network:

$$\mathbf{g}|_v = \mathrm{NNet}\left(\begin{array}{c} \mathbf{f} \oplus \mathbf{f}_s \oplus \mathbf{f}_v \oplus \\ \{\mathbf{p}|_v,\ \mathbf{p}|_N\} \oplus \mathbf{m} \end{array} \right) \in \mathbb{R}$$

By using regularisation, the learning will prefer where possible to use the sub-network for $\mathbf{g}$ rather than learning the individual weights for each value required in the sub-network for $\mathbf{h}$. This sub-network is able to deal with unseen or infrequently seen dialog states, so long as the state can be tagged in the feature extraction. This model can also be shared across slots since $\mathbf{f}_s$ is included as an input, see section 4.2.

The sub-networks applied to tagged and untagged inputs are combined to give the new belief:

$$\mathbf{p}' = \mathrm{softmax}\left([\mathbf{h} + \mathbf{g}] \oplus \{B\}\right) \in \mathbb{R}^{N+1}$$

where B is a parameter of the RNN, contributing to the *None* hypothesis. The contribution from $\mathbf{g}$ may be seen as accounting for general behaviour of tagged hypotheses, while $\mathbf{h}$ makes corrections due to correlations with untagged features and value specific behaviour e.g. special ways of expressing specific goals and fitting to specific ASR confusions.

Finally, the memory is updated according to the logistic regression:

$$\mathbf{m}' = \sigma\left(W_{m_0}\mathbf{f} + W_{m_1}\mathbf{m}\right) \in \mathbb{R}^{N_{\mathrm{mem}}}$$

where the W_{m_i} are parameters of the RNN.

3.4 Requested Slots and Method

A similar RNN is used to track the requested slots. Here the v runs over all the requestable slots, and requestable slot names are tagged in the feature vectors $\mathbf{f}_v$. This allows the neural network calculating $\mathbf{g}$ to learn general patterns across slots just as in the case of goals. The equation for $\mathbf{p}'$ is changed to:

$$\mathbf{p}' = \sigma\left(\mathbf{h} + \mathbf{g}\right)$$

so each component of $\mathbf{p}'$ represents the probability (between 0 and 1) of a slot being requested.

For method classification, the same RNN structure as for a goal is used. No tagging of the feature vectors is used in the case of methods.

4 Training

The RNNs are trained using Stochastic Gradient Descent (SGD), maximizing the log probability of the sequences of observed beliefs in the training data (Bottou, 1991). Gradient clipping is used to avoid the problem of exploding gradients (Pascanu et al., 2012). A regularisation term is included, which penalises the $l2$ norm of all the parameters. It is found empirically to be beneficial to give more weight in the regularisation to the parameters used in the network calculating $\mathbf{h}$.

When using the ASR N-best list, $\mathbf{f}$ is typically of dimensionality around 3500. With so many weights to learn, it is important to initialise the parameters well before starting the SGD algorithm. Two initialisation techniques have been investigated, the denoising autoencoder and shared initialisation. These were evaluated by training trackers on the dstc2_train set, and evaluating on dstc2_dev (see table 1).

4.1 Denoising Autoencoder

The denoising autoencoder (dA), which provides an unsupervised method for learning meaningful

		Joint Goals		Method		Requested	
Shared init.	dA init.	Acc	L2	Acc	L2	Acc	L2
		0.686	0.477	0.913	0.147	0.963	0.059
	✓	0.688	0.466	0.915	0.144	0.962	0.059
✓		0.680	0.479	0.910	0.152	0.962	0.059
✓	✓	**0.696**	**0.463**	**0.915**	**0.144**	**0.965**	**0.057**
	Baseline:	0.612	0.632	0.830	0.266	0.894	0.174

Table 1: Performance on the dev set when varying initialisation techniques for word-based tracking. Acc denotes the accuracy of the most likely belief at each turn, and L2 denotes the squared $l2$ norm between the estimated belief distribution and correct (delta) distribution. For each row, 5 trackers are trained and then combined using score averaging. The final row shows the results for the *focus*-based baseline tracker (Henderson et al., 2014).

underlying representations of the input, has been found effective as an initialisation technique in deep learning (Vincent et al., 2008).

A dA is used to initialise the parameters of the RNN which multiply the high-dimensional input vector $\mathbf{f}$. The dA learns a matrix W_{dA} which reduces $\mathbf{f}$ to a lower dimensional vector such that the original vector may be recovered with minimal loss in the presence of noise.

For learning the dA, $\mathbf{f}$ is first mapped such that feature values lie between 0 and 1. The dA takes as input $\mathbf{f}_{\mathrm{noisy}}$, a noisy copy of $\mathbf{f}$ where each component is set to 0 with probability p. This is mapped to a lower dimensional hidden representation $\mathbf{h}$:

$$\mathbf{h} = \sigma\left(W_{\mathrm{dA}}\mathbf{f}_{\mathrm{noisy}} + b_0\right)$$

A reconstructed vector, $\mathbf{f}_{\mathrm{rec}}$, is then calculated as:

$$\mathbf{f}_{\mathrm{rec}} = \sigma\left(W_{\mathrm{dA}}^{\mathrm{T}}\mathbf{h} + b_1\right)$$

The cross-entropy between $\mathbf{f}$ and $\mathbf{f}_{\mathrm{rec}}$ is used as the objective function in gradient descent, with an added $l1$ regularisation term to ensure the learning of sparse weights. As the ASR features are likely to be very noisy, dense weights would be prone to overfitting the examples. [1]

When using W_{dA} to initialise weights in the RNN, training is observed to converge faster. Table 1 shows that dA initialisation leads to better solutions, particularly for tracking the goals.

4.2 Shared Initialisation

It is possible to train a slot-independent RNN, using training data from all slots, by not including $\mathbf{h}$ in the model (the dimensionality of $\mathbf{h}$ is dependent on the slot). In *shared initialisation*, such an RNN is trained for a few epochs, then the learnt parameters are used to initialise slot-dependent RNNs for each slot. This follows the shared initialisation procedure presented in Henderson et al. (2013b).

Table 1 suggests that shared initialisation when combined with dA initialisation gives the best performance.

4.3 Model Combination

In DSTC1, the most competitive results were achieved with model combination whereby the output of multiple trackers were combined to give more accurate classifications (Lee and Eskenazi, 2013). The technique for model combination used here is score averaging, where the final score for each component of the dialog state is computed as the mean of the scores output by all the trackers being combined. This is one of the simplest methods for model combination, and requires no extra training data. It is guaranteed to improve the accuracy if the outputs from the individual trackers are not correlated, and the individual trackers operate at an accuracy > 0.5.

Multiple runs of training the RNNs were found to give results with high variability and model combination provides a method to exploit this variability. In order to demonstrate the effect, 10 trackers with varying regularisation parameters were trained on dstc2_train and used to track dstc2_dev. Figure 3 shows the effects of combining these trackers in larger groups. The mean accuracy in the joint goals from combining m trackers is found to increase with m. The single output from combining all 10 trackers outperforms any single tracker in the group.

The approach taken for the DSTC2 challenge was therefore to train multiple trackers with vary-

[1]The state-of-the-art in dialog act classification with very similar data also uses sparse weights Chen et al. (2013).

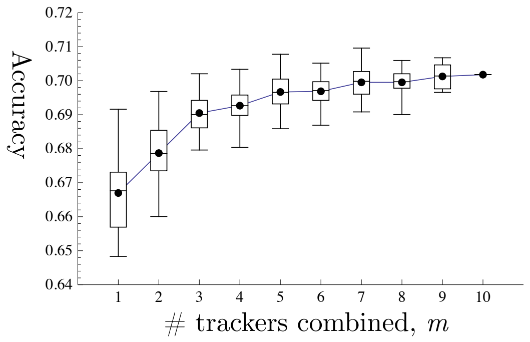

Figure 3: Joint goal accuracy on dstc2_dev from system combination. Ten total trackers are trained with varying regularisation parameters. For each $m = 1 \ldots 10$, all subsets of size m of the 10 trackers are used to generate ${}^{10}C_m$ combined results, which are plotted as a boxplot. Boxplots show minimum, maximum, the interquartile range and the median. The mean values are plotted as connected points.

ing model hyper-parameters (e.g. regularisation parameters, memory size) and combine their output using score averaging. Note that maintaining around 10 RNNs for each dialog state components is entirely feasible for a realtime system, as the RNN operations are quick to compute. An unoptimised Python implementation of the tracker including an RNN for each dialog state component is able to do state tracking at a rate of around 50 turns per second on an Intel® Core™ i7-970 3.2GHz processor.

5 Results

The strict blind evaluation procedure defined for the DSTC2 challenge was used to investigate the effect on performance of two contrasts. The first contrast compares word-based tracking and conventional tracking based on SLU output. The second contrast investigates the effect of including and omitting the sub-network for $\mathbf{h}$ in the RNN. Recall $\mathbf{h}$ is the part of the model that allows learning special behaviours for particular dialog state hypotheses, and correlations with untagged features. These two binary contrasts resulted in a total of 4 system variants being entered in the challenge.

Each system is the score-averaged combined output of 12 trackers trained with varying hyperparameters (see section 4.3). The performance of the 4 entries on the featured metrics of the challenge are shown in table 2.

It should be noted that the live SLU used the word confusion network, not made available in the challenge. The word confusion network is known to provide stronger features than the N-best list for language understanding (Henderson et al., 2012; Tür et al., 2013), so the word-based trackers using N-best ASR features were at a disadvantage in that regard. Nevertheless, despite this handicap, the best results were obtained from word-based tracking directly on the ASR output, rather than using the confusion network generated SLU output. Including $\mathbf{h}$ always helps, though this is far more pronounced for the word-based trackers. Note that trackers which do not include $\mathbf{h}$ are value-independent and so are capable of handling new values at runtime.

The RNN trackers performed very competitively in the context of the challenge. Figure 4 visualises the performance of the four trackers relative to all the entries submitted to the challenge for the featured metrics. For full details of the evaluation metrics see Henderson et al. (2014). The box in this figure gives the entry IDs under which the results are reported in the DSTC (under the team ID '**team4**'). The word-based tracker including $\mathbf{h}$ ($\mathbf{h}$-ASR), was top for joint goals L2 as well as requested slots accuracy and L2. It was close to the top for the other featured metrics, following closely entries from team 2. The RNN trackers performed particularly well on measures assessing the quality of the scores such as L2.

There are hundreds of numbers reported in the DSTC2 evaluation, and it was found that the $\mathbf{h}$-ASR tracker ranked top on many of them. Considering L2, accuracy, average probability, equal error rate, log probability and mean reciprocal rank across all components of the the dialog state, these give a total of 318 metrics. The $\mathbf{h}$-ASR tracker ranked top of all trackers in the challenge in 89 of these metrics, more than any other tracker. The ASR tracker omitting $\mathbf{h}$ came second, ranking top in 33 of these metrics.

The trackers using SLU features ranked top in all of the featured metrics among the trackers which used only the SLU output.

6 Conclusions

The RNN framework presented in this paper provides very good performance in terms of both accuracy and the quality of reported probability distributions. Word-based tracking is shown to be one of the most competitive approaches submitted to DSTC2. By mapping straight from the ASR output to a belief update, it avoids any information

	Tracker Inputs			Joint Goals			Method			Requested		
entry	Include **h**	Live ASR	Live SLU	Acc	L2	ROC	Acc	L2	ROC	Acc	L2	ROC
0	✓	✓		**0.768**	**0.346**	0.365	**0.940**	**0.095**	0.452	**0.978**	**0.035**	0.525
1		✓		0.746	0.381	0.383	0.939	0.097	0.423	0.977	0.038	0.490
2	✓		✓	0.742	0.387	0.345	0.922	0.124	0.447	0.957	0.069	0.340
3			✓	0.737	0.406	0.321	0.922	0.125	0.406	0.957	0.073	0.385

Table 2: Featured metrics on the test set for the 4 RNN trackers entered to the challenge.

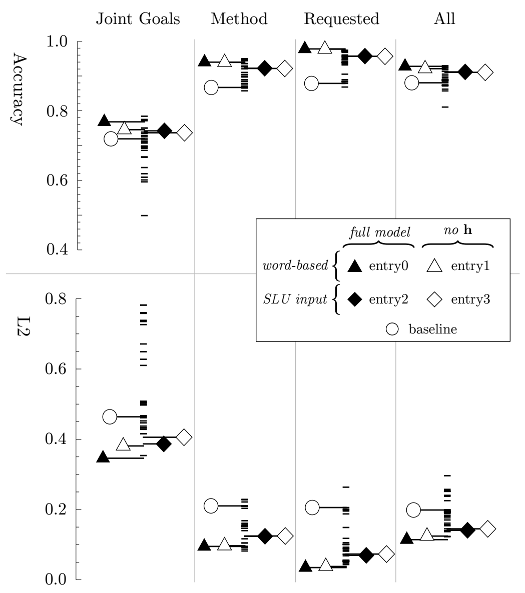

Figure 4: Relative performance of RNN trackers for featured metrics in DSTC2. Each dash is one of the 34 trackers evaluated in the challenge. Note a lower L2 is better. ROC metric is only comparable for systems of similar accuracies, so is not plotted. The *focus* baseline system is shown as a circle.

lost in the omitted SLU step.

In general, the RNN appears to be a promising model, which deals naturally with sequential input and outputs. High dimensional inputs are handled well, with little feature engineering, particularly when carefully initialised (e.g. as here using denoising autoencoders and shared initialisation).

Future work should include making joint predictions on components of the dialog state. In this paper each component was tracked using its own RNN. Though not presented in this paper, no improvement could be found by joining the RNNs. However, this may not be the case for other domains in which slot values are more highly correlated. The concept of tagging the feature functions allows for generalisation to unseen values and slots. This generalisation will be explored in future work, particularly for dialogs in more open-domains.

Acknowledgements

Matthew Henderson is a Google Doctoral Fellow.

References

Dan Bohus and Alex Rudnicky. 2006. A K-hypotheses+ Other Belief Updating Model. *Proc. of the AAAI Workshop on Statistical and Empirical Methods in Spoken Dialogue Systems.*

Léon Bottou. 1991. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nîmes, France. EC2.

Yun-Nung Chen, William Yang Wang, and Alexander I Rudnicky. 2013. An empirical investigation of sparse log-linear models for improved dialogue act classification. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on.*

Matthew Henderson, Milica Gašić, Blaise Thomson, Pirros Tsiakoulis, Kai Yu, and Steve Young. 2012. Discriminative Spoken Language Understanding Using Word Confusion Networks. In *Spoken Language Technology Workshop, 2012. IEEE.*

Matthew Henderson, Blaise Thomson, and Jason Williams. 2013a. Dialog State Tracking Challenge 2 & 3 Handbook. `camdial.org/~mh521/dstc/`.

Matthew Henderson, Blaise Thomson, and Steve Young. 2013b. Deep Neural Network Approach for the Dialog State Tracking Challenge. In *Proceedings of SIGdial*, Metz, France, August.

Matthew Henderson, Blaise Thomson, and Jason Williams. 2014. The second dialog state tracking challenge. In *Proceedings of the SIGdial 2014 Conference*, Baltimore, U.S.A., June.

Sungjin Lee and Maxine Eskenazi. 2013. Recipe for building robust spoken dialog state trackers: Dialog state tracking challenge system description. In *Proceedings of the SIGDIAL 2013 Conference*, Metz, France, August.

Sungjin Lee. 2013. Structured discriminative model for dialog state tracking. In *Proceedings of the SIGDIAL 2013 Conference*, Metz, France, August.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2012. Understanding the exploding gradient problem. *CoRR*.

Hang Ren, Weiqun Xu, Yan Zhang, and Yonghong Yan. 2013. Dialog state tracking using conditional random fields. In *Proceedings of the SIGDIAL 2013 Conference*, Metz, France, August.

Gökhan Tür, Anoop Deoras, and Dilek Hakkani-Tür. 2013. Semantic parsing using word confusion networks with conditional random fields. In *INTERSPEECH*.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland.

Jason Williams. 2013. Multi-domain learning and generalization in dialog state tracking. In *Proceedings of the SIGDIAL 2013 Conference*, Metz, France, August.

Steve Young, Milica Gašić, Simon Keizer, François Mairesse, Jost Schatzmann, Blaise Thomson, and Kai Yu. 2009. The Hidden Information State model: A practical framework for POMDP-based spoken dialogue management. *Computer Speech & Language*.