Context Aware Conversational Understanding for Intelligent Agents with a Screen

Vishal Ishwar Naik * [1], **Angeliki Metallinou** [2], **Rahul Goel** [2]
[1] Arizona State University, [2] Amazon Alexa Machine Learning
vnaik1@asu.edu, {ametalli, goerahul }@amazon.com

Abstract

We describe an intelligent context-aware conversational system that incorporates screen context information to service multimodal user requests. Screen content is used for disambiguation of utterances that refer to screen objects and for enabling the user to act upon screen objects using voice commands. We propose a deep learning architecture that jointly models the user utterance and the screen and incorporates detailed screen content features. Our model is trained to optimize end to end semantic accuracy across contextual and non-contextual functionality, therefore learns the desired behavior directly from the data. We show that this approach outperforms a rule-based alternative, and can be extended in a straightforward manner to new contextual use cases. We perform detailed evaluation of contextual and non-contextual use cases and show that our system displays accurate contextual behavior without degrading the performance of non-contextual user requests.

1 Introduction

Voice powered personal assistants like Amazon Alexa, Google Now and Apple Siri have become powerful among consumers as they enable intuitive user-interfaces that resemble human communication. Natural human interaction is to a large extend contextual, where interlocutors' successful communication often depends on a shared understanding of their environment, that influences the semantics of the dialog. For example, interlocutors may refer to or act upon objects or imply shared knowledge. Building intelligent assistants that feel and act natural requires enriching conversational understanding systems with the ability to incorporate context while processing a spoken user request.

Here we focus on screen content as the shared environment context for intelligent devices with access to a screen. Our goal is to enable multimodal functionality where screen content is used to interpret and disambiguate the user's intention and the objects they are referring to. For example, a request like 'select harry potter' may refer to a book or a movie, and the screen content can be used for disambiguation. 'play the first one' is a similarly ambiguous anaphoric request that could refer to music, movies, news items on screen etc. We describe the system design and spoken language understanding (SLU) methodologies that enable screen integration functionality, where the user can act on screen objects using voice commands. We propose joint modeling of text and screen context in a single SLU model trained to optimize end to end semantic accuracy of the spoken request. We show that our model is able to learn the desired behavior directly from contextual training instances and outperforms an alternative system with hand engineered context rules. While there is some relevant multimodal work on combining text with speech emphasis features for conversational understanding (Ning et al. 2017) and on using gaze and linguistic information for coreference resolution (Iida, Yasuhara, and Tokunaga 2011), to our knowledge there is no prior work on developing a multimodal conversational system for screen integration.

In summary, the main contributions of this work are:

- We describe the design of a context aware virtual agent that allows understanding and executing multimodal user requests. We use a flexible representation of the semantic content of the user's request and corresponding screen content cues that cover a wide range of visual use cases.
- We propose deep learning architectures that jointly model the input user utterance and the screen information and are trained to optimize end to end semantic accuracy. We show that our models can learn from data the desired behavior and can outperform a hand engineered rule-based system for screen integration.
- We perform detailed analysis of our system for contextual and non-contextual use cases to ensure improvement in the former without degradation in the latter, and discuss the effect of different types of screen features.

2 Related Work

Deep learning advances on Recurrent Neural Networks (RNNs) and Long Short Term Memory networks (LSTMs) (Hochreiter and Schmidhuber 1997), (Gers, Schraudolph, and Schmidhuber 2002), and the availability of fast GPU computing resources, have made such deep learning methods state of the art for many natural language processing tasks (NLP), such as sequence tagging (Chung et al. 2014), (Ma and Hovy 2016), sequence to sequence modeling (Sutskever, Vinyals, and Le 2014), and sentence classi-

*Work done during an internship at Amazon Alexa

fication (Liu et al. 2015) (Socher et al. 2013). The multitask learning paradigm is also very common in the literature, where a network is jointly trained to optimize performance for multiple related tasks, and learns to exploit beneficial correlations across tasks (Collobert and Weston 2008), (Liu, Qiu, and Huang 2017). Finally, convolutional neural networks (CNNs) are common for image classification (LeCun et al. 1998) (Krizhevsky, Sutskever, and Hinton. 2012), and text processing and classification (Kim 2014) (Kalchbrenner, Grefenstette, and Blunsom 2014).

Regarding the use of contextual information, related work in language modeling (LM) succesfully uses topics (Lau, Baldwin, and Cohn 2017), document history (Ji et al. 2015) or discourse information (Ji, Haffari, and Eisenstein 2016). These different types of context are used as an additional conditioning input to an LSTM-LM, leading to increased accuracy. Hierarchical LSTMs have also been proposed for modeling conversational context within and between utterances in a dialog (Sordoni et al. 2015), (Serban et al. 2016).

Regarding semantic understanding for intelligent assistants, recent work includes the use of deep convex neural networks (Deng et al. 2012), LSTMs with attention (Liu and Lane 2016), and multitask LSTMs (Ning et al. 2017). Finally, there is little prior work on multimodal conversational understanding systems, including combining speech content and speech emphasis for recognizing the user's intent (Ning et al. 2017), combining speech content and gaze for coreference resolution (Iida, Yasuhara, and Tokunaga 2011) and combining speech content and hand gestures for coreference resolution (Eisenstein and Davis 2006).

3 Context Aware Conversational Understanding

We focus on Amazon Alexa, a voice operated conversational agent that interacts with the user to fulfill requests for various functionality. Alexa is part of a speech powered device with a screen, where the screen can be used to display content relevant to the user's request. The user interacts with Alexa through spoken commands, which are recognized by our Automatic Speech Recognition (ASR) module, and their semantic content is parsed by our Spoken Language Understanding (SLU) module. Requests are routed to the appropriate application layer according to their semantic content. For example, a request like 'search for harry potter showtimes in palo alto' would be routed to the Movie application layer to service the request. As a result, Alexa will respond to the user verbally and relevant movie showtimes will be displayed on the screen. Additionally, the application layer will return to the SLU module information about the current screen content. This enables building a screen-aware SLU and extending the agent's functionality to service essentially multimodal requests. For example, the user can refer to objects on the screen ('select the second one') or request particular screen content ('play harry potter'). The SLU would consider screen content to disambiguate the utterance semantics and route the request appropriately to the Book, Movie or other application layer.

Specifically, the SLU module performs *intent* detection and *slot* tagging of the user's request. Considering the earlier example, the intent detection unit performs utterance classification for intent categories such as SearchAction<MovieScreening>. The sequence tagging unit tags relevant semantic tokens, called here *slots* or *entities*, e.g., Movie.name='harry potter' and City.name='palo alto'. Our semantic representation is detailed in Section 4.1.

Assume an utterance $\overline{X} = x_1, x_2, \cdots x_T$, a set of intent labels $I \in \{I_i\}$, and a set of slot labels $S \in \{S_i\} \cup \{Other\}$. We model $P(I|\overline{X})$ and $P(\overline{S}|\overline{X})$, where $\overline{S} = S_1, S_2, \ldots S_T$ Additionally, assume screen context information C as a set of M key-value pairs, i.e., $C = \{(C^m_{key}, C^m_{value})\}^M_{m=1}$, where M is the number of items on screen. The key is a contextual entity type identifier, such as 'Onscreen_Movie' while the value is the contextual entity name, like 'harry potter'. For example, for a screen that displays two movies, the context could be $(`Onscreen\_Movie', `harry\ potter')$ and $(`Onscreen\_Movie', `interstellar')$. The number M is variable and application specific, and the order of the items is not available to the SLU. This work proposes methods for context-aware semantic understanding, that allow us to model $P(I|\overline{X}, C)$ and $P(\overline{S}|\overline{X}, C)$.

Importantly, our model should adapt to the availability of relevant screen context, i.e., Alexa should learn to consider screen context when it is present and relevant, but ignore screen context when it is absent or irrelevant. This is desirable because the screen may be turned off or disconnected, or the user's request may be irrelevant to the screen content. The enriched context-aware functionality of our agent should not come at the expense of non-contextual requests. We consider both contextual and non contextual use cases, detailed in Table 1, where our goal is to improve performance on the former without degrading the latter.

Table 1: Contextual and non-contextual example use cases.

Description	User command	Desired behavior
Contextual Example Use Case		
Context is present and relevant	Screen shows Movies including Harry Potter 'play harry potter'	Alexa plays movie Harry Potter
Non Contextual Example Use Cases		
Context is not present	No screen content 'what is the weather'	Alexa gives weather forecast
Context is not relevant	Screen shows movies 'what is the weather'	Alexa gives weather forecast
	Screen shows books 'play interstellar'	Alexa plays movie Interstellar

4 Data and Representation

4.1 User Interaction Data and the AMRL

Our dataset contains user interactions with Alexa for various functionality, including playing music, movies, videos

or books, movie showtimes, local search queries, weather forecast, calendar information. For annotating our data we developed the Alexa Meaning Representation Language (AMRL), an internal formalism for rich semantic annotation. Examples of AMRL annotation are presented in Table 2 for both contextual and non contextual use cases.

Alexa MRL Examples	
utterance 1	play the movie harry potter
intent	PlayAction<Movie>
entities	'movie'→ Movie.type
	'harry potter'→ Movie.name
utterance 2	play the first one
screen	screen shows movies
intent	PlayAction<Movie>
entities	'first'→ Ordinal.value
	'one'→ Movie.reference
utterance 3	select the first one
screen	no screen content
intent	ChooseAction<Thing>
entities	'first'→ Ordinal.value
	'one'→ Thing.reference

Table 2: Examples of AMRL annotation for non-contextual and contextual use cases

Intents are represented as actions applied on entities. For entity annotation, our representation distinguishes between entity types (Movie.type='movie'), entity names (Movie.name='harry potter'), and entity references (Movie.reference='one'). Utterances 2 and 3 of Table 2 are examples of anaphora, where screen content is used to disambiguate the reference 'one' in the former utterance, while the reference in the latter utterance is under-specified and we tag it with the generic entity label $Thing$.

4.2 Screen Content Information

We examine two groups of visual use cases: selecting an object on the screen (ChooseAction$< \cdot >$) and playing an object on screen (PlayAction$< \cdot >$), where applicable. These actions can cover functionality from various domains depending on the type of object that is displayed. In Table 3, we give a few examples of contextual functionality and corresponding screen information. Our screen object data range across domains including music, books, movies and showtimes, videos, shopping, calendar, local search.

5 Methodology

5.1 Non Contextual Multitask BiLSTM

We first describe our non contextual architecture for intent classification and slot tagging. We employ Bidirectional Long Short Term Memory neural networks (BiLSTMs) for both the slot tagging and the intent classification tasks of our SLU unit. The LSTM unit, proposed by (Hochreiter and Schmidhuber 1997), is an RNN extension, that is designed to handle long term dependencies through the use of an input gate, an output gate, a forget gate and a memory cell. Here, we use the LSTM variant with peephole connections (Gers, Schraudolph, and Schmidhuber 2002). Given a sentence input $\overline{X} = x_1, \cdots x_T$, the LSTM computes a representation r_t at each t, which is denoted as $r_t = \phi(x_t, r_{t-1})$.

Table 3: Examples of contextual functionality per domain and corresponding example screen content.

Domain and Screen	Intents and Entities
Movies (Onscreen_Movie, 'harry potter')	ChooseAction<Movie> PlayAction<Movie> Movie (.name, .type, .reference)
Books (Onscreen_Book, 'harry potter')	ChooseAction<Book> PlayAction<Book> Book (.name, .type, .reference)
LocalSearch (Point_Of_Interest, 'starbucks')	ChooseAction<LocalBusiness> LocalBusiness (.name, .type, .reference)

In our setup, each input word x_t is embedded into a $E = 300$ dimensional embedding, where the embeddings are estimated from our data. Additionally, we use pre-trained word embeddings as a separate input, that allows incorporating unsupervised word information from much larger corpora. Our network structure is shown in Fig. 1b (inputs (a)).

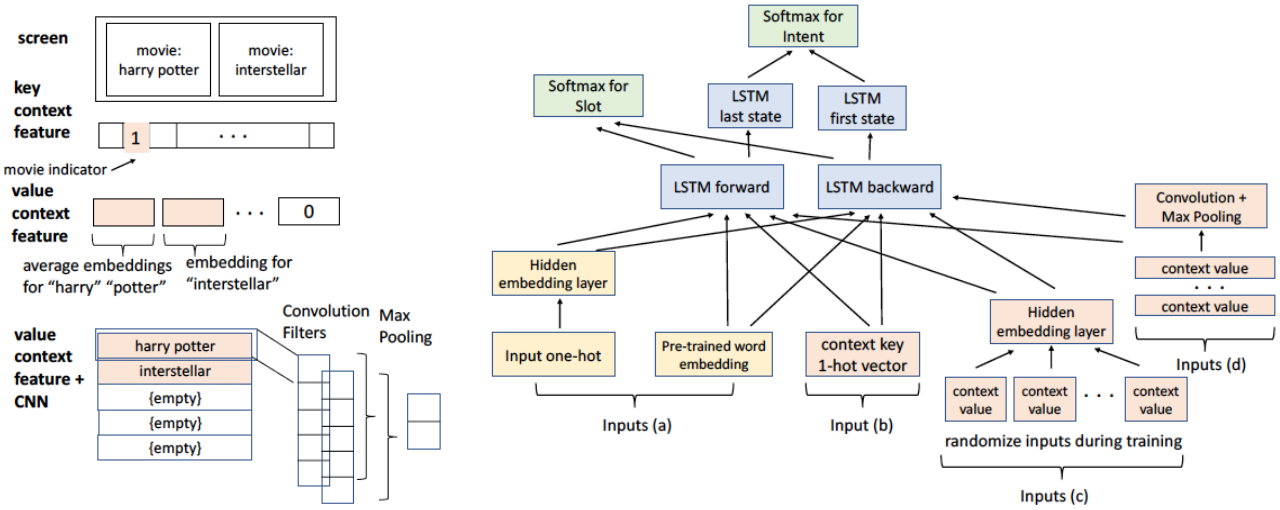
We use BiLSTM, which consists of a forward and a backward LSTM. For slot tagging, we concatenate the two LSTM representations at each time step t before passing to a softmax. For intent classification, we concatenate the full sentence representation obtained from the final timestep of the forward LSTM and the first timestep of the backward LSTM and pass them through a softmax for classification:

$$r_t^{slot} = r_t^{forw} \oplus r_t^{back}, r^{intent} = r_T^{forw} \oplus r_1^{back}$$
$$\hat{S}_t = softmax(W_s r_t^{slot} + b_s)$$
$$\hat{I} = softmax(W_i r^{intent} + b_i)$$

where $\oplus$ is the concatenation operator.

While we can train two separate BiLSTMs for intent classification and slot tagging, we use multitask learning to jointly learn a shared representation for both tasks. Multitask learning leverages correlations between related tasks to learn joint representations that are informative across the tasks. It is relevant in our case since the slots and the intent are related sub-components of the underlying semantics of the user's request. Therefore, we use a single B-LSTM to learn both tasks, i.e, $\phi = \phi^{intent} = \phi^{slot}$.

Empirically, we have observed that this multitask setup achieves better results than separately training intent and slot models, with the added advantage of having a single model, and a smaller total parameter size. Going forward, we will use this architecture as our base model, which will be augmented with contextual features.



(a) Example screen content features

(b) Model architecture with different context inputs

Figure 1: Left: Example of screen content showing two movies and the corresponding key content indicator feature, and value content indicator features. Right: Model architecture. Utterance inputs (non-contextual) are denoted as inputs (a). Key context features are input (b). Value context features are inputs (c), and value context features processed by a CNN are inputs (d).

5.2 Context Aware Multitask BiLSTM

Encoding entity type context using key context features The screen context information C is provided as a set of M unordered key-value pairs, where M is variable, i.e., $C = \{(C^m_{key}, C^m_{value})\}^M_{m=1}$. The keys C^m_{key} come from a limited set of values, each representing a distinct visual functionality and typically originating from a separate application (domain) layer, e.g., 'Onscreen_Movie' for the Movies domain. For many use cases, such as anaphora to a screen object, the keys encode all the the visual information required to disambiguate a user's request

We assume a set of known possible keys of size K. This is not restrictive, because adding a new key essentially corresponds to adding new visual functionality, which requires re-training the SLU model anyway, to support the added functionality. We create an indicator feature vector of size K, where ones denote the presence, at least once, of the corresponding key in the screen context C, while zeros denote absence of that key. This feature is illustrated in Fig. 1a, and will be referred to as *Key context feature*: c_{key}. We pass c_{key} at each t as an additional input to the BiLSTM model, as shown in Fig. 1b with input (b), therefore computing the context-aware sentence representation $r_t = \phi(x_t, r_{t-1}, c_{key})$:

Encoding entity name context using value context features The key values C^m_{value} contain the entity names that appear on the screen, such as book titles, movie names, etc. This additional information could benefit the SLU model for non anaphoric requests, where the user refers to a screen object by name. It could also increase robustness to avoid contextual false positives. Consider the request 'play stone roses', for the music band 'stone roses', while the screen is showing a list of movies. A model that only relies on the key context feature might be too biased to tag 'stone roses' as a movie. The additional value information could help the model ignore the irrelevant screen content, which does not match the utterance text.

Given the limited screen size, we assume a maximum number N of items that can appear simultaneously on the screen, here $N = 5$. Therefore, our feature allocates maximum N positions, each one filled by C^m_{value}. Some positions will be empty, if there are less than N items on screen. In our architecture, we use pre-trained word embeddings for C^m_{value}, where multi-word values are computed by averaging the embeddings per word. For embedding size E, the total feature vector size is $N \times E$. The corresponding *Value context feature* c_{value} is illustrated in Fig. 1a, and added as an extra input to the model $r_t = \phi(x_t, r_{t-1}, c_{key}, c_{value})$:

The order of the values in the feature does not matter, as we only care about the presence of a string value anywhere on the screen. To make our model ignore the order, we randomize the order of the N positions of the value context feature for a training instance, across different epochs of model training. Empirically, we observe that this randomization process leads to better performance. Our model is shown in Fig. 1b (inputs c).

5.3 Context Aware Multitask CNN+BiLSTM

Encoding entity name context using value context features Using c_{value} as described earlier requires computing weights for each of the N input positions. However, for our contextual use cases, there can be at most one value position that is relevant to the utterance. For example, for 'play interstellar', there may be at most one relevant matching screen value C^m_{value}='interstellar'. Ideally, we want to scan through the screen values, and the select the one with maximum relevance to our task.

We note that this operation can be implemented through a standard convolutional layer, followed by a max pooling layer. The typical convolution operation for images involves a 2D filter that is applied consecutively across the image to compute location invariant features, keeping filter weights shared. Similarly, our context values can be represented by a N-by-E matrix, where each row is a screen entity value. As before, some rows will be zero if there are less than N items on screen, and we average the word embeddings of multi-word values. We apply convolutional filters of size 1-by-E, with a stride of one, that scan each value consecutively, followed by a max-over-N max pooling operation to select the most relevant value. Our strategy is illustrated in Fig. 1a (bottom). The extracted context features are added as an extra input to our BiLSTM model: $r_t = \phi(x_t, r_{t-1}, c_{key}, \psi^{Conv}(c_{value}))$, where ψ^{Conv} denotes the convolution and max pooling operation. The model is illustrated in Fig. 1b (inputs d). Our method is similar to convolution for text classification tasks (Kim 2014), where convolution is applied across consecutive words in a sentence. One difference is that our filters need to be 1-dimensional instead of spanning more than one values, because entity values on screen are unrelated and in random order. In contrast, convolution over a sentence can span many words to convey consecutive word information.

5.4 BiLSTM + Contextual Reranker

We consider an alternative strategy for context-aware SLU using a two-stage pipeline where a contextual reranker is applied on top of the base non-contextual BiLSTM multi-task model. The reranker module contains hand engineered rules that encode the desired behavior for each of the contextual use cases. This represents a reasonable alternative where contextual logic is directly encoded through rules. To strengthen this approach, we associate each rule with a weight optimized on a subset of the data. We will compare our context-aware BiLSTM architectures with this rule based alternative, to understand to what extend our joint utterance-context models can learn the desired behavior directly from the data, and whether they can reach and potentially outperform the hand designed contextual rules.

The reranker takes the n-best lists from the slot tagging and the intent classification results of the non-contextual BiLSTM model, and increases the scores of certain intent or slot labels depending on the screen context keys $\{C^m_{key}\}^M_{m=1}$. The logic is that given a screen content ('Onscreen_Movie'), some slot or intent labels should be upweighted (Movie). An example rule for Movie contextual functionality is shown in Example 1. To avoid false positives that would cause degradation of non-contextual functionality, as defined in Table 1, we also include rules to decrease the default probabilities of certain contextual intent and slot labels. The logic here is that the probability of certain contextual labels, e.g., ChooseAction<Movie>, is low by default, unless a related entity appears on screen (it does not make sense to choose an object if the screen is empty). The increase and decrease weights, α and β respectively, are parameters that are estimated through grid search on a validation set. Regarding the choice of 'n' in the n-best list, we set $n = 10$, which empirically gave good performance. The contextual re-ranker computes updated scores and re-ranks the slot and intents n-bests lists accordingly. The top reranker output is used for final SLU recognition.

Example 1: Example contextual reranking rule for Movie

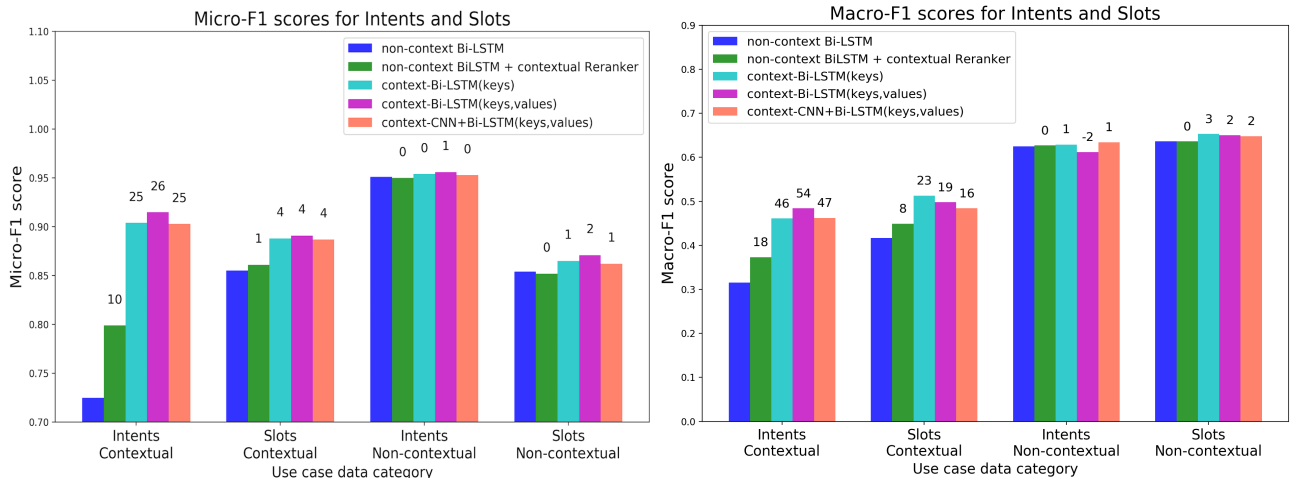
```
# default down-weight rule
P(ChooseAction<Movie>)
    ← P(ChooseAction<Movie>) −β
# context dependent up-weight rule
if screen keys contain Onscreen_Movie then
  P(Movie.name) ← P(Movie.name) +α
  P(Movie.type) ← P(Movie.type) +α
  P(Movie.reference) ← P(Movie.reference) +α
  P(PlayAction<Movie>)
      ← P(PlayAction<Movie>) +α
  P(ChooseAction<Movie>)
      ← P(ChooseAction<Movie>) +α
end if
```

6 Experiments

6.1 Data preparation

We use a large set of non contextual utterances from user interactions with Alexa on devices without a screen, and a smaller dataset of contextual utterances from user interactions where a screen is available. The train and test data sizes for non contextual use cases are on the order of 100K utterances, while for contextual use cases they are on the order of 10K utterances. The dataset used for these experiments is a fraction of our production data and covers a range of domain functionality including music, books, movies and showtimes, videos, calendar events, local search, shopping, and general commands. In total there are $|S| = 105$ distinct slot labels for the slot tagging task and $|I| = 108$ distinct intent labels for the intent classification task. The number of screen content keys in our study is $K = 8$. We used a train-dev split of 70%-30%, where the dev set was used for optimizing the deep learning models as well as tuning the reranker parameters α, β.

All our deep learning models were trained on the same amount of data. For the non-contextual model we don't use any screen content features. For contextual models, we used the relevant screen information that is available for the contextual utterances to compute c_{key} and c_{value}. For non-contextual utterances, we applied the following preprocessing strategy. For half of the utterances, we set all contextual features to zero to simulate the case where a screen is not available (turned off, disconnected, etc.). For the other half, we paired the utterance with a random screen content $C = \{(C^m_{key}, C^m_{value})\}^M_{m=1}$ and extracted the contextual key and value features from it. This is to simulate the case of irrelevant screen context. Context values on screen and entity names spoken by the user both have a potentially unlimited number of values, therefore we can assume that the likelihood of the screen content randomly matching the user command is very low. The random context facts were sampled



(a) Micro-F1 for intents and slots.

(b) Macro-F1 for intents and slots.

Figure 2: Numbers on bar graphs show relative % improvements compared to non-contextual baseline

from a pool of much larger, un-annotated user interactions with Alexa on a device with screen.

6.2 Experimental Setup

We examine performance of the non-contextual model, as well as four contextual models, all described in section 5:

- **non-context BiLSTM** Described in Section 5.1, and illustrated in Figure 1b, inputs (a)
- **non-context BiLSTM + contextual Reranker**. Two stage strategy with reranker described in Section 5.4
- **context-BiLSTM(keys)** Described in Section 5.2, using only the key contextual features. Illustrated in Fig. 1b, inputs (a)+(b)
- **context-BiLSTM(keys,values)** Described in Section 5.2, using both the key and value contextual features. Illustrated in Fig. 1b, inputs (a)+(b)+(c)
- **context-CNN+BiLSTM(keys,values)** Described in Section 5.3, using both the key and value contextual features. Illustrated in Fig. 1b, inputs (a)+(b)+(d)

All our deep learning models were trained end-to-end using stochastic gradient descent. Models were regularized using dropout and L2. We used word2vec pre-trained embeddings of size E=300 downloaded from (Mikolov). For the **context-BiLSTM(keys,values)** model we use a hidden layer of size H between the context value feature vector and the BiLSTM, see Fig. 1b. Empirically, we chose $H = 200$ while the input value vector is of size $N \times E = 5 \times 300$. Value inputs at each of the 5 positions are shuffled between training epochs, as described in 5.2, which led to better performance. For the **context-CNN+BiLSTM(keys,values)**, we used $F = 100$ filters for the convolutional layer. We also found that adding a ReLu non-linearity after the max pooling operation performs slightly better. For the **non-context BiLSTM + contextual Reranker**, the reranker parameters α and β were chosen based on grid search in $[0, 1]$, to optimize the dev set performance for slots and intents. Performance for slot tagging and intent classification was measured using the F1 score. We report both micro-F1 and macro-F1, the former giving a picture of our overall system accuracy while the latter giving a picture of performance at the long tail of infrequent intents and slots.

6.3 Results and Discussion

Results for all models are presented in Figures 2a and 2b, for micro-F1 and macro-F1 respectively. Each plot contains F1 for intents and slots separately. Looking from left to right, plots show performance for contextual and non-contextual test sets separately. We want the contextual models to improve the performance of the contextual test sets, without degrading the performance of the non contextual test sets. For all contextual models, the numbers above the bar plots show the percentage of relative change of the model with respect to the non-contextual BiLSTM model. Positive changes indicate performance improvement.

Looking at the contextual use cases, on the left of Fig. 2a, where we would expect to see the effect of contextual modeling, we indeed notice a significance performance increase, up to 26% rel. improvement for intents and 4% rel. improvement for slots, compared to the non-contextual model. The improvement is most pronounced for contextual intents, for which the base non-contextual model has low accuracy. This is because of the large number of possible valid intent labels for a given ambiguous contextual request ('select number two' can be labeled as any of ChooseAction$< \cdot >$), which makes screen content necessary for disambiguation. Finally, we observe that the performance of non-contextual intents and slots does not degrade, see Fig. 2a (right), which means that non contextual functionality is not negatively affected. The overall micro-F1 performance over both contextual and non-contextual test sets, is around 95% for intents and 86% for slots across all contextual and non-contextual models, hence does not degrade when introducing contextual model-

ing. Looking at the macro-F1 performance of Fig. 2b, we notice similar trends. The relative improvements of our contextual architectures are significantly more pronounced, which indicates that our context modeling benefits the most the long tail of infrequent contextual intents and slots.

Compared to the two-stage strategy with the rule-based reranker, our contextual BiLSTMs achieve higher performance in context use cases. Our models are able to learn more complex contextual behavior that is optimized directly on the data, compared to the hand designed reranker logic. Additionally, it is straightforward to scale our modeling to new functionality by adding new contextual use cases to our data, while a rule based system requires more manual maintenance. While we could have pushed reranker performance a bit higher by optimizing separate weights per rule (α_i, β_i) such a strategy would not be scalable as contextual functionality increases (grid search on this space would become unscalable), and was therefore out of scope.

Comparing the use of the simpler key context features, over combination of both key and value context features, we observe that the key context feature is sufficient for achieving most of the performance increase (e.g., see BiLSTM(keys) vs BiLSTM(keys,values)). Indeed, the majority of our contextual data are requests for screen objects by reference ('select the second one') or by order ('select number two'), which can be disambiguated through the key feature alone. Requests where the value feature is important for disambiguation are quite rare, and typically include cases where the user requests an entity name which is irrelevant to the objects that appear on screen (see example utterance 2 in Table 4 in the next section). As a result, the use of value context features results in a small performance increase over using just key context features, and adds more robustness to our models. We would expect the relative importance of the value feature to increase, as contextual user interactions with Alexa become richer and more complex, including more frequent references by object name. Finally, between the context-BiLSTM(keys,values) and the context-CNN+BiLSTM(keys,values) models, the former has a small advantage.

6.4 Example Contextual Results

In Table 4, we illustrate examples of the behavior of the proposed contextual joint BiLSTM models, and the effect of key and value screen features. Utterance 1 illustrates a case where screen context successfully disambiguates a reference to a screen object, while the non-contextual model just assigns the most frequent viable intent label from the training data. Utterance 2 is an example music request while the screen is showing movies. While the non-contextual model correctly classifies the intent, the contextual BiLSTM with key features is misleaded by the screen context and misclassifies the intent as a movie request. Adding value context features to our BiLSTM models helps recover from this error by capturing the fact that screen content is irrelevant. This illustrates the usefulness of value features in adding robustness and reducing false positives. Utterance 3 shows an example of successful slot tagging for the Book 'redemption' using screen context. Utterance 4 is a request for selecting an event that does not appear on screen. The non contextual model defaults to correctly tagging Event.name='surgery', while the contextual models, misleaded by the empty screen, fail to tag 'surgery' as a slot. The user intention is unclear, and may refer to a calendar event creation earlier in the dialog. This illustrates a failure for our contextual model, that might be remedied by adding more contextual information such as conversation history.

Intent classification examples	
Utterance 1	play the third one
Screen	screen shows movies
Ground Truth	PlayAction<Movie>
Non-context BiLSTM	PlayAction<MusicWork>
Context BiLSTM(k)	PlayAction<Movie>
Context BiLSTM(k, v)	PlayAction<Movie>
Utterance 2	play cameron
Screen	screen shows movies
Ground Truth	PlayAction<MusicWork>
Non-context BiLSTM	PlayAction<MusicWork>
Context BiLSTM(k)	PlayAction<Movie>
Context BiLSTM(k, v)	PlayAction<MusicWork>
Slot tagging examples	
Utterance 3	play redemption
Screen	screen shows books including redemption
Ground Truth	redemption→Book.name
Non-context BiLSTM	redemption →MusicWork.name
Context BiLSTM(k)	redemption→Book.name
Context BiLSTM(k, v)	redemption→Book.name
Utterance 4	select surgery
Screen	empty screen
Ground Truth	surgery→Event.name
Non-context BiLSTM	surgery→Event.name
Context BiLSTM(k)	surgery→ not tagged as slot
Context BiLSTM(k, v)	surgery→ not tagged as slot

Table 4: Results for non-contextual and contextual models. (k) denotes (keys) and (k,v) denotes (keys, values)

7 Conclusions and Future Work

We presented the design of a context aware conversational agent that enables multimodal user requests for screen content integration. We proposed deep learning architectures that jointly model utterance content and the screen context, and are trained end-to-end to maximize semantic accuracy for slot and intent recognition. We compared our end-to-end approach with a hand engineered rule-based method, where the rules encode the desired user experience, and showed that our system outperforms this rule-based alternative. Our approach is naturally extensible to new visual use cases, without requiring manual rule writing. Finally, we extensively evaluated the performance of our proposed method in both contextual and non contextual use cases, and verified

that the contextual awareness of our models does not cause a degradation of non contextual functionality. In future, we plan to extend our visual features to encode screen object locations for multiple object types displayed simultaneously on screen (e.g., both books and movies). We will also explore additional context cues such as conversation context.

References

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning,*.

Collobert, R., and Weston, J. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proc. of International Conference of Machine Learning (ICML) 2008*.

Deng, L.; Tur, G.; He, X.; and Hakkani-Tur, D. 2012. Use of kernel deep convex networks and end-to-end learning for spoken language understanding. In *Proc of SLT 2012*.

Eisenstein, J., and Davis, R. 2006. Gesture improves coreference resolution. In *Proc. of the Human Language Technology Conference of the NAACL 2006*.

Gers, F. A.; Schraudolph, N. N.; and Schmidhuber, J. 2002. Learning precise timing with lstm recurrent networks. *Journal of Machine Learning Research* 3:115–143.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Iida, R.; Yasuhara, M.; and Tokunaga, T. 2011. Multi-modal reference resolution in situated dialogue by integrating linguistic and extra-linguistic clues. In *Proc of International Joint Conference on Natural Language Processing (IJCNLP) 2011*.

Ji, Y.; Cohn, T.; Kong, L.; Dyer, C.; and Eisenstein, J. 2015. Document context language models. In *Proc of CoRR 2015*.

Ji, Y.; Haffari, G.; and Eisenstein, J. 2016. A latent variable recurrent neural network for discourse relation language models. In *Proc. of NAACL-HLT 2016*.

Kalchbrenner, N.; Grefenstette, E.; and Blunsom, P. 2014. A convolutional neural network for modelling sentences. In *In Proceedings of ACL 2014*.

Kim, Y. 2014. Convolutional neural networks for sentence classification. In *Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Krizhevsky, A.; Sutskever, I.; and Hinton., G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Proc of In NIPS 2012*.

Lau, J. H.; Baldwin, T.; and Cohn, T. 2017. Topically driven neural language model. In *Proc. of the 55th Annual Meeting of the Association for Computational Linguistics (ACL) 2017*.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proc. of the IEEE* 86(11):2278–2324.

Liu, B., and Lane, I. 2016. Attention-based recurrent neural network models for joint intent detection and slot filling. In *In Proc of Interspeech 2016*.

Liu, P.; Qiu, X.; Chen, X.; Wu, S.; and Huang., X. 2015. Multi-timescale long short-term memory neural network for modelling sentences and documents. In *In Proceedings of the Conference on EMNLP 2015*.

Liu, P.; Qiu, X.; and Huang, X. 2017. Adversarial multi-task learning for text classification. In *Proc. of the 55th Annual Meeting of the Association for Computational Linguistics (ACL) 2017*.

Ma, X., and Hovy, E. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (ACL) 2016*.

Mikolov, T. Word2Vec: https://code.google.com/archive/p/word2vec/.

Ning, Y.; Jia, J.; Wu, Z.; Li, R.; An, Y.; Wang, Y.; and Meng, H. 2017. Multi-task deep learning for user intention understanding in speech interaction systems. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*.

Serban, I. V.; Sordoni, A.; Bengio, Y.; Courville, A.; and Pineau, J. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*.

Socher, R.; Perelygin, A.; Wu, J. Y.; Chuang, J.; Manning, C. D.; Ng, A. Y.; and Potts, C. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc of EMNLP 2013*.

Sordoni, A.; Bengio, Y.; Vahabi, H.; Lioma, C.; Simonsen, J. G.; and Nie, J.-Y. 2015. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *Proc. of the 24th ACM International on Conference on Information and Knowledge Management (CIKM) 2015*.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Proc. of the 27th International Conference on Neural Information Processing Systems (NIPS) 2014*.