Globally Coherent Text Generation with Neural Checklist Models

Chloé Kiddon Luke Zettlemoyer Yejin Choi
Computer Science & Engineering
University of Washington
{chloe, lsz, yejin}@cs.washington.edu

Abstract

Recurrent neural networks can generate locally coherent text but often have difficulties representing what has already been generated and what still needs to be said – especially when constructing long texts. We present the *neural checklist model*, a recurrent neural network that models global coherence by storing and updating an agenda of text strings which should be mentioned somewhere in the output. The model generates output by dynamically adjusting the interpolation among a language model and a pair of attention models that encourage references to agenda items. Evaluations on cooking recipes and dialogue system responses demonstrate high coherence with greatly improved semantic coverage of the agenda.

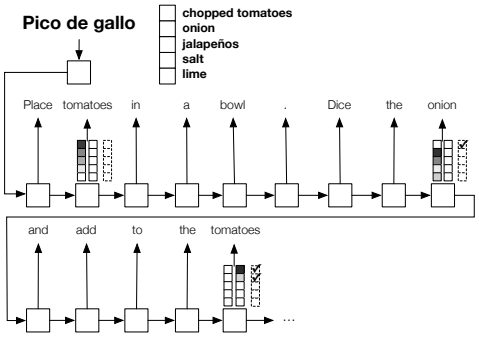

Figure 1: Example checklist recipe generation. A checklist (right dashed column) tracks which agenda items (top boxes; "salt," "lime," etc.) have already been used (checked boxes). The model is trained to interpolate an RNN (e.g., encode "pico de gallo" and decode a recipe) with attention models over new (left column) and used (middle column) items that identify likely items for each time step (shaded boxes; "tomatoes," etc.).

1 Introduction

Recurrent neural network (RNN) architectures have proven to be well suited for many natural language generation tasks (Mikolov et al., 2010; Mikolov et al., 2011; Sordoni et al., 2015; Xu et al., 2015; Wen et al., 2015; Mei et al., 2016). Previous neural generation models typically generate locally coherent language that is on topic; however, overall they can miss information that should have been introduced or introduce duplicated or superfluous content. These errors are particularly common in situations where there are multiple distinct sources of input or the length of the output text is sufficiently long. In this paper, we present a new recurrent neural model that maintains coherence while improving coverage by globally tracking what has been said and what is still left to be said in complete texts.

For example, consider the challenge of generating a cooking recipe, where the title and ingredient list are provided as inputs and the system must generate a complete text that describes how to produce the desired dish. Existing RNN models may lose track of which ingredients have already been mentioned, especially during the generation of a long recipe with many ingredients. Recent work has focused on adapting neural network architectures to improve coverage (Wen et al., 2015) with application to generating customer service responses, such as hotel information, where a single sentence is generated to describe a few key ideas. Our focus is instead on developing a model that maintains coherence while producing longer texts or covering longer

input specifications (e.g., a long ingredient list).

More specifically, our *neural checklist model* generates a natural language description for achieving a goal, such as generating a recipe for a particular dish, while using a new checklist mechanism to keep track of an agenda of items that should be mentioned, such as a list of ingredients (see Fig. 1). The checklist model learns to interpolate among three components at each time step: (1) an encoder-decoder language model that generates goal-oriented text, (2) an attention model that tracks remaining agenda items that need to be introduced, and (3) an attention model that tracks the used, or checked, agenda items. Together, these components allow the model to learn representations that best predict which words should be included in the text and when references to agenda items should be checked off the list (see check marks in Fig. 1).

We evaluate our approach on a new cooking recipe generation task and the dialogue act generation from Wen et al. (2015). In both cases, the model must correctly describe a list of agenda items: an ingredient list or a set of facts, respectively. Generating recipes additionally tests the ability to maintain coherence in long procedural texts. Experiments in dialogue generation demonstrate that our approach outperforms previous work with up to a 4 point BLEU improvement. Our model also scales to cooking recipes, where both automated and manual evaluations demonstrate that it maintains the strong local coherence of baseline RNN techniques while significantly improving the global coverage by effectively integrating the agenda items.

2 Task

Given a goal $\mathbf{g}$ and an agenda $E = \{\mathbf{e}_1, \ldots, \mathbf{e}_{|E|}\}$, our task is to generate a goal-oriented text $\mathbf{x}$ by making use of items on the agenda. For example, in the cooking recipe domain, the goal is the recipe title ("pico de gallo" in Fig. 1), and the agenda is the ingredient list (e.g., "lime," "salt"). For dialogue systems, the goal is the dialogue type (e.g., inform or query) and the agenda contains information to be mentioned (e.g., a hotel name and address). For example, if $\mathbf{g}$ ="inform" and $E = \{$name(Hotel Stratford), has_internet(no)$\}$, an output text might be $\mathbf{x}$ ="Hotel Stratford does not have internet."

3 Related Work

Attention models have been used for many NLP tasks such as machine translation (Balasubramanian et al., 2013; Bahdanau et al., 2014), abstractive sentence summarization (Rush et al., 2015), machine reading (Cheng et al., 2016), and image caption generation (Xu et al., 2015). Our model uses new types of attention to record what has been said and to select new agenda items to be referenced.

Recently, other researchers have developed new ways to use attention mechanisms for related generation challenges. Most closely related, Wen et al. (2015) and Wen et al. (2016) present neural network models for generating dialogue system responses given a set of agenda items. They focus on generating short texts (1-2 sentences) in a relatively small vocabulary setting and assume a fixed set of possible agenda items. Our model composes substantially longer texts, such as recipes, with a more varied and open ended set of possible agenda items. We also compare performance for our model on their data.

Maintaining coherence and avoiding duplication have been recurring challenges when generating text using RNNs for other applications, including image captioning (Jia et al., 2015; Xu et al., 2015) and machine translation (Tu et al., 2016b; Tu et al., 2016a). A variety of solutions have been developed to address infrequent or out-of-vocabulary words in particular (Gülçehre et al., 2016; Jia and Liang, 2016). Instead of directly copying input words or deterministically selecting output, our model can learn how to generate them (e.g., it might prefer to produce the word "steaks" when the original recipe ingredient was "ribeyes"). Finally, recent work in machine translation models has introduced new training objectives to encourage attention to all input words (Luong et al., 2015), but these models do not accumulate attention while decoding.

Generating recipes was an early task in planning (Hammond, 1986) and generating referring expression research (Dale, 1988). These can be seen as key steps in classic approaches to generating natural language text: a formal meaning representation is provided as input and the model first does content selection to determine the non-linguistic concepts to be conveyed by the output text (i.e., *what to say*) and then does realization to describe those concepts

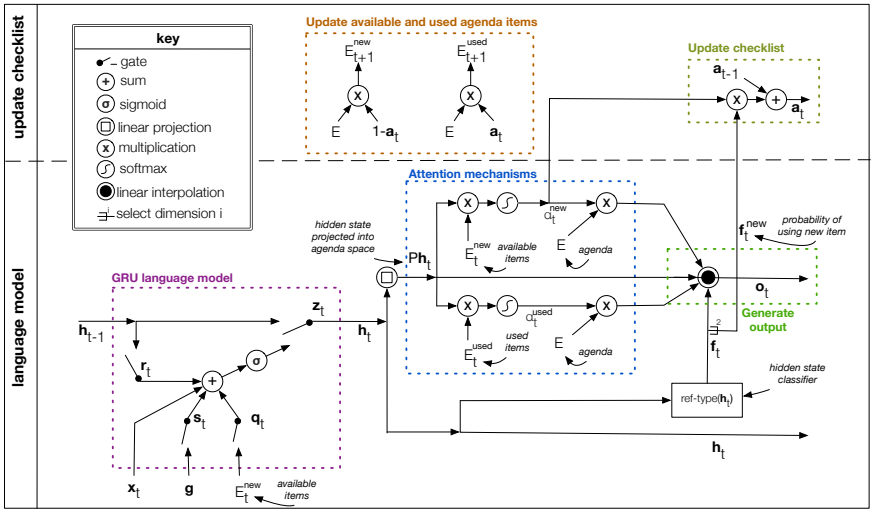

Figure 2: A diagram of the neural checklist model. The bottom portion depicts how the model generates the output embedding $\mathbf{o}_t$. The top portion shows how the checklist and available/used agenda item matrices are updated.

in natural language text (i.e., *how to say it*) (Thompson, 1977; Reiter and Dale, 2000). More recently, machine learning methods have focused on parts of this approach (Barzilay and Lapata, 2005; Liang et al., 2009) or the full two-stage approach (Angeli et al., 2010; Konstas and Lapata, 2013). Most of these models shorter texts, although Mori et al. (2014) did consider longer cooking recipes. Our approach is a joint model that instead operates with textual input and tries to cover all of the content it is given.

4 Model

Fig. 2 shows a graphical representation of the neural checklist model. At a high level, our model uses a recurrent neural network (RNN) language model that encodes the goal as a bag-of-words and then generates output text token by token. It additionally stores a vector that acts as a soft checklist of what agenda items have been used so far during generation. This checklist is updated every time an agenda item reference is generated and is used to compute the available agenda items at each time step. The available items are used as an input to the language model and to constrain which agenda items can still be referenced during generation. Agenda embeddings are also used when generating item references.

4.1 Input variable definitions

We assume the goal $\mathbf{g}$ and agenda items E (see Sec. 2) are each defined by a set of tokens. Goal tokens come from a fixed vocabulary $\mathcal{V}_{goal}$, the item tokens come from a fixed vocabulary $\mathcal{V}_{agenda}$, and the tokens of the text $\mathbf{x}_t$ come from a fixed vocabulary $\mathcal{V}_{text}$. In an abuse of notation, we represent each goal $\mathbf{g}$, agenda item $\mathbf{e}_i$, and text token $\mathbf{x}_t$ as a k-dimensional word embedding vector. We compute these embeddings by creating indicator vectors of the vocabulary token (or set of tokens for goals and agenda items) and embed those vectors using a trained $k \times |\mathcal{V}_z|$ projection matrix, where $z \in \{goal, agenda, text\}$ depending whether we are generating a goal, agenda item, or text token.

Given a goal embedding $\mathbf{g} \in \mathbb{R}^k$, a matrix of L agenda items $E \in \mathbb{R}^{L \times k}$, a checklist soft record of what items have been used $\mathbf{a}_{t-1} \in \mathbb{R}^L$, a previous hidden state $\mathbf{h}_{t-1} \in \mathbb{R}^k$, and the current input word embedding $\mathbf{x}_t \in \mathbb{R}^k$, our architecture computes the next hidden state $\mathbf{h}_t$, an embedding used to generate the output word $\mathbf{o}_t$, and the updated checklist $\mathbf{a}_t$.

4.2 Generating output token probabilities

To generate the output token probability distribution (see "Generate output" box in Fig. 2), $\mathbf{w}_t \in \mathbb{R}^{|V_{text}|}$, we project the *output hidden state* $\mathbf{o}_t$ into the vocabulary space and apply a softmax:

$$\mathbf{w}_t = \text{softmax}(W_o \mathbf{o}_t),$$

where $W_o \in \mathbb{R}^{|V| \times k}$ is a trained projection matrix. The output hidden state is the linear interpolation of (1) content $\mathbf{c}_t^{gru}$ from a Gated Recurrent Unit

(GRU) language model, (2) an encoding $\mathbf{c}_t^{new}$ generated from the new agenda item reference model (Sec. 4.3), and (3) and an encoding $\mathbf{c}_t^{used}$ generated from a previously used item model (Sec. 4.4):

$$\mathbf{o}_t = f_t^{gru}\mathbf{c}_t^{gru} + f_t^{new}\mathbf{c}_t^{new} + f_t^{used}\mathbf{c}_t^{used}.$$

The interpolation weights, f_t^{gru}, f_t^{new}, and f_t^{used}, are probabilities representing how much the output token should reflect the current state of the language model or a chosen agenda item. f_t^{gru} is the probability of a non-agenda-item token, f_t^{new} is the probability of an new item reference token, and f_t^{used} is the probability of a used item reference. In the Fig. 1 example, f_t^{new} is high in the first row when new ingredient references "tomatoes" and "onion" are generated; f_t^{used} is high when the reference back to "tomatoes" is made in the second row, and f_t^{gru} is high the rest of the time.

To generate these weights, our model uses a three-way probabilistic classifier, $ref\text{-}type(\mathbf{h}_t)$, to determine whether the hidden state of the GRU $\mathbf{h}_t$ will generate non-agenda tokens, new agenda item references, or used item references. $ref\text{-}type(\mathbf{h}_t)$ generates a probability distribution $\mathbf{f}_t \in \mathbb{R}^3$ as

$$\mathbf{f}_t = ref\text{-}type(\mathbf{h}_t) = softmax(\beta S\mathbf{h}_t),$$

where $S \in \mathbb{R}^{3\times k}$ is a trained projection matrix and β is a temperature hyper-parameter. $f_t^{gru} = \mathbf{f}_t^1$, $f_t^{new} = \mathbf{f}_t^2$, and $f_t^{used} = \mathbf{f}_t^3$. $ref\text{-}type()$ does not use the agenda, only the hidden state $\mathbf{h}_t$: $\mathbf{h}_t$ must encode when to use the agenda, and $ref\text{-}type()$ is trained to identify that in $\mathbf{h}_t$.

4.3 New agenda item reference model

The two key features of our model are that it (1) predicts which agenda item is being referred to, if any, at each time step and (2) stores those predictions for use during generation. These components allow for improved output texts that are more likely to mention agenda items while avoiding repetition and references to irrelevant items not in the agenda.

These features are enabled by a *checklist vector* $\mathbf{a}_t \in \mathbb{R}^L$ that represents the probability each agenda item has been introduced into the text. The checklist vector is initialized to all zeros at $t = 1$, representing that all items have yet to be introduced. The checklist vector is a soft record with each $\mathbf{a}_{t,i} \in [0, 1]$.[1]

We introduce the remaining items as a matrix $E_t^{new} \in \mathbb{R}^{L\times k}$, where each row is an agenda item embedding weighted by how likely it is to still need to be referenced. For example, in Fig. 1, after the first "tomatoes" is generated, the row representing "chopped tomatoes" in the agenda will be weighted close to 0. We calculate E_t^{new} using the checklist vector (see "Update [...] items" box in Fig. 2):

$$E_t^{new} = ((\mathbf{1}_L - \mathbf{a}_{t-1}) \otimes \mathbf{1}_k) \circ E,$$

where $\mathbf{1}_L = \{1\}^L$, $\mathbf{1}_k = \{1\}^k$, and the outer product $\otimes$ replicates $\mathbf{1}_L - \mathbf{a}_{t-1}$ for each dimension of the embedding space. $\circ$ is the Hadamard product (i.e., element-wise multiplication) of two matrices with the same dimensions.

The model predicts when an agenda item will be generated using $ref\text{-}type()$ (see Sec. 4.2 for details). When it does, the encoding $\mathbf{c}_t^{new}$ approximates which agenda item is most likely. $\mathbf{c}_t^{new}$ is computed using an attention model that generates a learned soft alignment $\boldsymbol{\alpha}_t^{new} \in \mathbb{R}^L$ between the hidden state $\mathbf{h_t}$ and the rows of E_t^{new} (i.e., available items). The alignment is a probability distribution representing how close $\mathbf{h}_t$ is to each item:

$$\boldsymbol{\alpha}_t^{new} \propto \exp(\gamma E_t^{new} P\mathbf{h}_t),$$

where $P \in \mathbb{R}^{k\times k}$ is a learned projection matrix and γ is a temperature hyper-parameter. In Fig. 1, the shaded squares in the top line (i.e., the first "tomatoes" and the onion references) represent this alignment. The attention encoding $\mathbf{c}_t^{new}$ is then the attention-weighted sum of the agenda items:

$$\mathbf{c}_t^{new} = E^T\boldsymbol{\alpha}_t^{new}.$$

At each step, the model updates the checklist vector based on the probability of generating a new agenda item reference, $\mathbf{f}_t^{new}$, and the attention alignment $\boldsymbol{\alpha}_t^{new}$. We calculate the update to checklist, $\mathbf{a}_t^{new}$, as $\mathbf{a}_t^{new} = \mathbf{f}_t^{new} \cdot \boldsymbol{\alpha}_t^{new}$. Then, the new checklist $\mathbf{a}_t$ is $\mathbf{a}_t = \mathbf{a}_{t-1} + \mathbf{a}_t^{new}$.

[1] By definition, $\mathbf{a}_t$ is non-negative. We truncate any values greater than 1 using a hard tanh function.

4.4 Previously used item reference model

We also allow references to be generated for previously used agenda items through the previously used item encoding $\mathbf{c}_t^{used}$. This is useful in longer texts – when agenda items can be referred to more than once – so that the agenda is always responsible for generating its own referring expressions. The example in Fig. 1 refers back to tomatoes when generating to what to add the diced onion.

At each time step t, we use a second attention model to compare $\mathbf{h}_t$ to a used items matrix $E_t^{used} \in \mathbb{R}^{L \times k}$. Like the remaining agenda item matrix E_t^{new}, E_t^{used} is calculated using the checklist vector generated at the previous time step:

$$E_t^{used} = (\mathbf{a}_{t-1} \otimes \mathbf{1}_k) \circ E.$$

The attention over the used items, $\boldsymbol{\alpha}_t^{used} \in \mathbb{R}^L$, and the used attention encoding $\mathbf{c}_t^{used}$ are calculated in the same way as those over the available items (see Sec. 4.3 for comparison):

$$\boldsymbol{\alpha}_t^{used} \propto \exp(\gamma E_t^{used} P \mathbf{h}_t),$$
$$\mathbf{c}_t^{used} = E^T \boldsymbol{\alpha}_t^{used}.$$

4.5 GRU language model

Our decoder RNN adapts a Gated Recurrent Unit (GRU) (Cho et al., 2014). Given an input $\mathbf{x}_t \in \mathbb{R}^k$ at time step t and the previous hidden state $\mathbf{h}_{t-1} \in \mathbb{R}^k$, a GRU computes the next hidden state $\mathbf{h}_t$ as

$$\mathbf{h}_t = (\mathbf{1} - \mathbf{z}_t)\mathbf{h}_{t-1} + \mathbf{z}_t \tilde{\mathbf{h}}_t.$$

The *update gate*, $\mathbf{z}_t$, interpolates between $\mathbf{h}_{t-1}$ and new content, $\tilde{\mathbf{h}}_t$, defined respectively as

$$\mathbf{z}_t = \sigma(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1}),$$
$$\tilde{\mathbf{h}}_t = \tanh(W \mathbf{x}_t + \mathbf{r}_t \odot U \mathbf{h}_{t-1}).$$

$\odot$ is an element-wise multiplication, and the *reset gate*, $\mathbf{r}_t$, is calculated as

$$\mathbf{r}_t = \sigma(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1}).$$

W_z, U_z, W, U, W_r, $U_r \in \mathbb{R}^{k \times k}$ are trained projection matrices.

We adapted a GRU to allow extra inputs, namely the goal $\mathbf{g}$ and the available agenda items E_t^{new} (see "GRU language model" box in Fig. 2). These extra inputs help guide the language model stay on topic. Our adapted GRU has a change to the computation of the new content $\tilde{\mathbf{h}}_t$ as follows:

$$\begin{aligned}\tilde{\mathbf{h}}_t \;=\; & \tanh(W_h x_t + \mathbf{r}_t \odot U_h \mathbf{h}_{t-1} \\ & + \mathbf{s}_t \odot Y \mathbf{g} + \mathbf{q}_t \odot (\mathbf{1}_L^T Z E_t^{new})^T,\end{aligned}$$

where $\mathbf{s}_t$ is a *goal select* gate and $\mathbf{q}_t$ is a *item select* gate, respectively defined as

$$\mathbf{s}_t = \sigma(W_s \mathbf{x}_t + U_s \mathbf{h}_{t-1}),$$
$$\mathbf{q}_t = \sigma(W_q \mathbf{x}_t + U_q \mathbf{h}_{t-1}).$$

$\mathbf{1}_L$ sums the rows of the available item matrix E_t^{new}. Y, Z, W_s, U_s, W_q, $U_q \in \mathbb{R}^{k \times k}$ are trained projection matrices. The goal select gate controls when the goal should be taken into account during generation: for example, the recipe title may be used to decide what the imperative verb for a new step should be. The item select gate controls when the available agenda items should be taken into account (e.g., when generating a list of ingredients to combine). The GRU hidden state is initialized with a projection of the goal: $\mathbf{h}_0 = U_g \mathbf{g}$, where $U_g \in \mathbb{R}^{k \times k}$.

The content vector $\mathbf{c}_t^{gru}$ that is used to compute the output hidden state $\mathbf{o}_t$ is a linear projection of the GRU hidden state, $\mathbf{c}_t^{gru} = P\mathbf{h}_t$, where P is the same learned projection matrix used in the computation of the attention weights (see Sections 4.3 and 4.4).

4.6 Training

Given a training set of (goal, agenda, output text) triples $\{(\mathbf{g}^{(1)}, E^{(1)}, \mathbf{x}^{(1)}), \ldots, (\mathbf{g}^{(J)}, E^{(J)}, \mathbf{x}^{(J)})\}$, we train model parameters by minimizing negative log-likelihood: $NLL(\theta) =$

$$-\sum_{j=1}^{J} \sum_{i=2}^{N_j} \log p(\mathbf{x}_i^{(j)} | \mathbf{x}_1^{(j)}, \ldots, \mathbf{x}_{i-1}^{(j)}, \mathbf{g}^{(j)}, E^{(j)}; \theta),$$

where $\mathbf{x}_1^{(j)}$ is the start symbol. We use mini-batch stochastic gradient descent, and back-propagate through the goal, agenda, and text embeddings.

It is sometimes the case that weak heuristic supervision on latent variables can be easily gathered to improve training. For example, for recipe generation, we can approximate the linear interpolation weights $\mathbf{f}_t$ and the attention updates $\mathbf{a}_t^{new}$ and $\mathbf{a}_t^{used}$ using string match heuristics comparing tokens in

the text to tokens in the ingredient list.[2] When this extra signal is available, we add mean squared loss terms to $NLL(\theta)$ to encourage the latent variables to take those values; for example, if $\mathbf{f}_t^*$ is the true value and $\mathbf{f}_t$ is the predicted value, a loss term $-(\mathbf{f}_t^* - \mathbf{f}_t)^2$ is added. When this signal is not available, as is the case with our dialogue generation task, we instead introduce a mean squared loss term that encourages the final checklist $\mathbf{a}_{N_j}^{(j)}$ to be a vector of 1s (i.e., every agenda item is accounted for).

4.7 Generation

We generate text using beam search, which has been shown to be fast and accurate for RNN decoding (Graves, 2012; Sutskever et al., 2014). When the beam search completes, we select the highest probability sequence that uses the most agenda items. This is the count of how many times the three-way classifier, $ref\text{-}type(\mathbf{h}_t)$, chose to generate an new item reference with high probability (i.e., $> 50\%$).

5 Experimental setup

Our model was implemented and trained using the Torch scientific computing framework for Lua.[3]

Experiments We evaluated neural checklist models on two natural language generation tasks. The first task is cooking recipe generation. Given a recipe title (i.e., the name of the dish) as the goal and the list of ingredients as the agenda, the system must generate the correct recipe text. Our second evaluation is based on the task from Wen et al. (2015) for generating dialogue responses for hotel and restaurant information systems. The task is to generate a natural language response given a query type (e.g., informing or querying) and a list of facts to convey (e.g., a hotel's name and address).

Parameters We constrain the gradient norm to 5.0 and initialize parameters uniformly on $[-0.35, 0.35]$. We used a beam of size 10 for generation. Based on dev set performance, a learning rate of 0.1 was chosen, and the temperature hyperparameters (β, γ) were $(5, 2)$ for the recipe task and $(1, 10)$ for the dialogue task. The models for the recipe task had a hidden state size of $k = 256$; the models for the dialogue task had $k = 80$ to compare to previous models. We use a batch size 30 for the recipe task and 10 for the dialogue task.

Recipe data and pre-processing We use the Now You're Cooking! recipe library: the data set contains over 150,000 recipes in the Meal-Master$^{\text{TM}}$ format.[4] We heuristically removed sentences that were not recipe steps (e.g., author notes, nutritional information, publication information). 82,590 recipes were used for training, and 1,000 each for development and testing. We filtered out recipes to avoid exact duplicates between training and dev (test) sets.

We collapsed multi-word ingredient names into single tokens using word2phrase[5] ran on the training data ingredient lists. Titles and ingredients were cleaned of non-word tokens. Ingredients additionally were stripped of amounts (e.g., "1 tsp"). As mentioned in Sec. 4.6, we approximate true values for the interpolation weights and attention updates for recipes based on string match between the recipe text and the ingredient list. The first ingredient reference in a sentence cannot be the first token or after a comma (e.g., the bold tokens cannot be ingredients in "**oil** the pan" and "in a large bowl, **mix** [...]").

Recipe data statistics Automatic recipe generation is difficult due to the length of recipes, the size of the vocabulary, and the variety of possible dishes. In our training data, the average recipe length is 102 tokens, and the longest recipe has 814 tokens. The vocabulary of the recipe text from the training data (i.e., the text of the recipe not including the title or ingredient list) has 14,103 unique tokens. About 31% of tokens in the recipe vocabulary occur at least 100 times in the training data; 8.6% of the tokens occur at least 1000 times. The training data also represents a wide variety of recipe types, defined by the recipe titles. Of 3793 title tokens, only 18.9% of the title tokens in the title vocabulary occur at least 100 times in the training data, which demonstrates the large variability in the titles.

Dialogue system data and processing We used the hotel and restaurant dialogue system corpus and the same train-development-test split from Wen et al. (2015). We used the same pre-processing, sets

[2] Similar to $\mathbf{a}_t^{new}$, $\mathbf{a}_t^{used} = \mathbf{f}_t^{used} \cdot \boldsymbol{\alpha}_t^{used}$.

[3] http://torch.ch/

[4] Recipes and format at http://www.ffts.com/recipes.htm

[5] See https://code.google.com/p/word2vec/

of reference samples, and baseline output, and we were given model output to compare against.[6] For training, slot values (e.g., "Red Door Cafe") were replaced by generic tokens (e.g., "NAME_TOKEN"). After generation, generic tokens were swapped back to specific slot values. Minor post-processing included removing duplicate determiners from the relexicalization and merging plural "-s" tokens onto their respective words. After replacing specific slot values with generic tokens, the training data vocabulary size of the hotel corpus is 445 tokens, and that of the restaurant corpus is 365 tokens. The task has eight goals (e.g., *inform*, *confirm*).

Models Our main baseline *EncDec* is a model using the RNN Encoder-Decoder framework proposed by Cho et al. (2014) and Sutskever et al. (2014). The model encodes the goal and then each agenda item in sequence and then decodes the text using GRUs. The encoder has two sets of parameters: one for the goal and the other for the agenda items. For the dialogue task, we also compare against the *SC-LSTM* system from Wen et al. (2015) and the handcrafted rule-based generator described in that paper.

For the recipe task, we also compare against three other baselines. The first is a basic attention model, *Attention*, that generates an attention encoding by comparing the hidden state $\mathbf{h}_t$ to the agenda. That encoding is added to the hidden state, and a nonlinear transformation is applied to the result before projecting into the output space. We also present a nearest neighbor baseline (*NN*) that simply copies over an existing recipe text based on the input similarity computed using cosine similarity over the title and the ingredient list. Finally, we present a hybrid approach (*NN-Swap*) that revises a nearest neighbor recipe using the neural checklist model. The neural checklist model is forced to generate the returned recipe nearly verbatim, except that it can generate new strings to replace any extraneous ingredients.

Our neural checklist model is labeled *Checklist*. We also present the *Checklist+* model, which interactively re-writes a recipe to better cover the input agenda: if the generated text does not use every agenda item, embeddings corresponding to missing items are multiplied by increasing weights and a new recipe is generated. This process repeats until the new recipe does not contain new items.

[6] We thank the authors for sharing their system outputs.

Model	BLEU-4	METEOR	Avg. % given items	Avg. extra items
Attention	2.8	8.6	22.8%	3.0
EncDec	3.1	9.4	26.9%	2.0
NN	**7.1**	12.1	40.0%	4.2
NN-Swap	**7.1**	**12.8**	58.2%	2.1
Checklist	3.0	10.3	67.9%	**0.6**
- $\mathbf{o}_t = \mathbf{h}_t$	2.1	8.3	29.1%	2.4
- no used	3.0	10.4	62.2%	1.9
- no supervision	3.7	10.1	38.9%	1.8
Checklist+	3.8	11.5	**83.4%**	0.8

Table 1: Quantitative results on the recipe task. The line with $\mathbf{o}_t = \mathbf{h}_t$ has the results for the non-interpolation ablation.

We also report the performance of our checklist model without the additional weak supervision of heuristic ingredient references (- *no supervision*) (see Sec. 4.6).[7] we also evaluate two ablations of our checklist model on the recipe task. First, we remove the linear interpolation and instead use $\mathbf{h}_t$ as the output (see Sec. 4.2). Second, we remove the previously used item reference model by changing $ref\text{-}type()$ to a 2-way classifier between new ingredient references and all other tokens (see Sec. 4.4).

Metrics We include commonly used metrics like BLEU-4,[8] and METEOR (Denkowski and Lavie, 2014). Because neither of these metrics can measure how well the generated recipe follows the input goal and the agenda, we also define two additional metrics. The first measures the percentage of the agenda items corrected used, while the second measures the number of extraneous items incorrectly introduced. Both these metrics are computed based on simple string match and can miss certain referring expressions (e.g., "meat" to refer to "pork"). Because of the approximate nature of these automated metrics, we also report a human evaluation.

6 Recipe generation results

Fig. 1 results for recipe generation. All BLEU and METEOR scores are low, which is expected for long texts. Our checklist model performs better than both neural network baselines (Attention and EncDec) in all metrics. Nearest neighbor baselines (NN and NN-Swap) perform the best in terms of BLEU and

[7] For this model, parameters were initialized on [-0.2, 0.2] to maximize development accuracy.

[8] See Moses system (http://www.statmt.org/moses/)

Model	Syntax	Ingredient use	Follows goal
Attention	4.47	3.02	3.47
EncDec	**4.58**	3.29	3.61
NN	4.22	3.02	3.36
NN-Swap	4.11	3.51	3.78
Checklist	**4.58**	3.80	3.94
Checklist+	4.39	**3.95**	**4.10**
Truth	4.39	4.03	4.34

Table 2: Human evaluation results on the generated and true recipes. Scores range in $[1, 5]$.

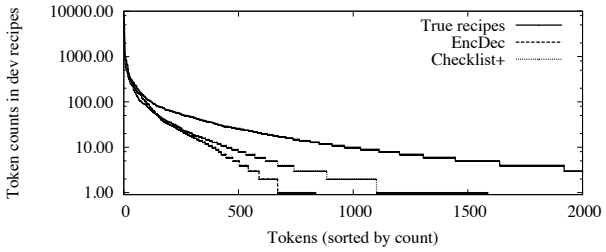

Figure 3: Counts of the most used vocabulary tokens (sorted by count) in the true dev set recipes and in generated recipes.

METEOR; this is due to a number of recipes that have very similar text but make different dishes.

However, NN baselines are not successful in generating a goal-oriented text that follows the given agenda: compared to Checklist+ (83.4%), they use substantially less % of the given ingredients (40% - 58.2%) while also introducing extra ingredients not provided. EncDec and Attention baselines similarly generate recipes that are not relevant to the given input, using only 22.8% - 26.9% of the agenda items. Checklist models rarely introduce extraneous ingredients not provided (0.6 - 0.8), while other baselines make a few mistakes on average (2.0 - 4.2).

The ablation study demonstrates the empirical contribution of different model components. ($\mathbf{o}_t = \mathbf{h}_t$) shows the usefulness of the attention encodings when generating the agenda references, while (*-no used*) shows the need for separate attention mechanisms between new and used ingredient references for more accurate use of the agenda items. Similarly, (*-no supervision*) demonstrates that the weak supervision encourages the model to learn more accurate management of the agenda items.

Human evaluation Because neither BLEU nor METEOR is suitable for evaluating generated text in terms of their adherence to the provided goal and the agenda, we also report human evaluation using Amazon Mechanical Turk. We evaluate the generated recipes on (1) grammaticality, (2) how well the recipe adheres to the provided ingredient list, and (3) how well the generated recipe accomplishes the desired dish. We selected 100 random test recipes. For each question we used a Likert scale ($\in [1, 5]$) and report averaged ratings among five turkers.

Table 2 shows the averaged scores over the responses. The checklist models outperform all baselines in generating recipes that follow the provided agenda closely and accomplish the desired goal, where NN in particular often generates the wrong dish. Perhaps surprisingly, both the Attention and EncDec baselines and the Checklist model beat the true recipes in terms of having better grammar. This can partly be attributed to noise in the parsing of the true recipes, and partly because the neural models tend to generate shorter, simpler texts.

Fig. 3 shows the counts of the most used vocabulary tokens in the true dev set recipes compared to the recipes generated by EncDec and Checklist+. Using the vocabulary from the training data, the true dev recipes use 5206 different tokens. The EncDec's vocabulary is only $\sim$16% of that size, while the Checklist+ model is a third of the size.

An error analysis on the dev set shows that the EncDec baseline over-generates catch-all phrases like "all ingredients" or "the ingredients," used in 21% of the generated recipes, whereas only 7.8% of true recipes use that construction. This phrase type simplifies the recipe, but using all ingredients in one step reduces the chance of accomplishing the desired dish correctly. The Checklist model only generates those phrases 13% of the time.

Qualitative analysis Fig. 4 shows two dev set recipes with generations from the EncDec and Checklist+ models. The EncDec model is much more likely to both use incorrect ingredients and to introduce ingredients more than once (e.g., "baking power" and "salt" in the bottom example are not in the ingredient list, and "milk" in the top example is duplicated). In the top example, the Checklist+ model refers to both Parmesean and Swiss cheese as "cheese"; generating more precise referring expressions is an important area for future work. The Checklist+ recipes generate the correct dishes to an extent: for example, the top recipe makes a casserole but does not cook the ingredients together before baking and mixes in biscuits instead of putting

Title: Casino royale casserole
Ingredients: 10 oz chopped broccoli, 2 tbsp butter, 2 tbsp flour, 1/2 tsp salt, 1/4 tsp black pepper, 1/4 tsp ground nutmeg, 1 cup milk, 1 1/2 cup shredded swiss cheese, 2 tsp lemon juice, 2 cup cooked cubed turkey, 4 oz mushrooms, 1/4 cup grated Parmesan cheese, 1 can refrigerated biscuits

Truth	EncDec	Checklist+
Preheat oven to 375. Melt butter in saucepan. Blend in flour, salt, pepper, and nutmeg; cook 1-2 minutes. Gradually add milk; cook, stirring, until slightly thickened. Stir in frozen vegetables. Remove from heat; stir in cheese until melted. Add lemon juice, turkey, mushrooms, and broccoli. Pour mixture into a lightly greased 8"x8" baking dish; sprinkle with parmesean and top with biscuits. Bake 20 minutes, or until biscuits are golden brown.	Cook broccoli according to package directions; drain. In medium saucepan, melt butter. Stir in flour, salt and pepper. Gradually add milk, stirring constantly. Cook over medium heat, stirring constantly, until mixture comes to a boil. Reduce heat to low. Cook, stirring constantly, until mixture boils and thickens, about 5 minutes. Stir in **milk**. Gradually heat to boiling; reduce heat. Cover; simmer 10 minutes or until heated through. Serve immediately.	Mix the turkey with the flour, salt, pepper, milk and lemon juice. Stir in the cheese. Add broccoli, mushrooms, nutmeg, and cheese. Mix well. Add the butter and mix well. Add the biscuits and mix well. Spread in a greased 13x9 inch baking pan. Bake in a 350f oven for 35 to 40 minutes, or until pick inserted in the center comes out clean. Serve hot.

Title: Kathie Frieman's cheese cake
Ingredients: 2 lb cream cheese, 1 3/4 cups sugar, 5 eggs, 15 oreo cookies, 1/2 tsp vanilla, 1/2 tsp almond extract

Truth	EncDec	Checklist+
Butter pan well. Whip cheese until soft and add sugar, combine until fluffy. Add extracts and eggs. Chop oreos into pieces and fold into mixture. Pour into pan and place in a pan of 1 or more inches of water. Bake at 350 1 1/2 hours (check after 1 1/4 hours) until center is set. Cool in pan of water. When cool turn upside-down on plate. Sprinkle with cookie crumbs.	Beat cream cheese and sugar together until light and fluffy. Add eggs, one at a time, beating well after each addition. Blend in vanilla. Sift flour, measure, and sift with **baking power** and **salt**. Add to creamed mixture alternately with **milk**. Fold in stiffly beaten **egg whites**. Pour into 9x13 pan. Bake at 350 degrees for 25 to 30 minutes.	Cream sugar and cream cheese until smooth. Add eggs, one at a time, beating well after each addition. Add vanilla, almond extract, and cookies. Mix well. Pour into greased and floured tube pan. Bake at 350 degrees for 30 minutes.

Figure 4: Example dev set generated recipes. Tokenization, newlines, and capitalization changed for space and readability. Bolded ingredient references are either ingredients not in the list and/or duplicated initial ingredient references.

Model	Hotel		Restaurant	
	BLEU	**METEOR**	**BLEU**	**METEOR**
HDC	55.52	48.10	44.39	43.42
SC-LSTM	86.53	60.84	74.49	54.31
Checklist	**90.61**	**62.10**	**77.82**	**54.42**

Table 3: Quantitative evaluation of the top generations in the hotel and restaurant domains

them on top. Future work could better model the full set of steps needed to achieve the overall goal.

7 Dialogue system results

Figure 3 shows our results on the hotel and restaurant dialogue system generation tasks. HDC is the rule-based baseline from Wen et al. (2015). For both domains, the checklist model achieved the highest BLEU-4 and METEOR scores, but both neural systems performed very well. The power of our model is in generating long texts, but this experiment shows that our model can generalize well to other tasks with different kinds of agenda items and goals.

8 Future work and conclusions

We present the neural checklist model that generates globally coherent text by keeping track of what has been said and still needs to be said from a provided agenda. Future work includes incorporating referring expressions for sets or compositions of agenda items (e.g., "vegetables"). The neural checklist model is sensitive to hyperparameter initialization, which should be investigated in future work. The neural checklist model can also be adapted to handle multiple checklists, such as checklists over composite entities created over the course of a recipe (see Kiddon (2016) for an initial proposal).

Acknowledgements

This research was supported in part by the Intel Science and Technology Center for Pervasive Computing (ISTC-PC), NSF (IIS-1252835 and IIS-1524371), DARPA under the CwC program through the ARO (W911NF-15-1-0543), and gifts by Google and Facebook. We thank our anonymous reviewers for their comments and suggestions, as well as Yannis Konstas, Mike Lewis, Mark Yatskar, Antoine Bosselut, Luheng He, Eunsol Choi, Victoria Lin, Kenton Lee, and Nicholas FitzGerald for helping us read and edit. We also thank Mirella Lapata and Annie Louis for their suggestions for baselines.

References

Gabor Angeli, Percy Liang, and Dan Klein. 2010. A simple domain-independent probabilistic approach to generation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 502–512.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. In *ICLR 2015*.

Niranjan Balasubramanian, Stephen Soderland, Mausam, and Oren Etzioni. 2013. Generating coherent event schemas at scale. In *Proceedings of the 2013 Conference on Empirical Methods on Natural Language Processing*, pages 1721–1731.

Regina Barzilay and Mirella Lapata. 2005. Collective content selection for concept-to-text generation. In *Proceedings of the 2005 Conference on Empirical Methods in Natural Language Processing*, pages 331–338.

Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

Robert Dale. 1988. *Generating Referring Expressions in a Domain of Objects and Processes*. Ph.D. thesis, Centre for Cognitive Science, University of Edinburgh.

Michael Denkowski and Alon Lavie. 2014. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the EACL 2014 Workshop on Statistical Machine Translation*, pages 376–380.

Alex Graves. 2012. Sequence transduction with recurrent neural networks. *Representation Learning Worksop, ICML*.

Çaglar Gülçehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 140–149.

Kristian J. Hammond. 1986. CHEF: A model of case-based planning. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 267–271.

R. Jia and P. Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 12–22.

Xu Jia, Efstratios Gavves, Basura Fernando, and Tinne Tuytelaars. 2015. Guiding long-short term memory for image caption generation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2407–2415.

Chloé Kiddon. 2016. *Learning to Interpret and Generate Instructional Recipes*. Ph.D. thesis, Computer Science & Engineering, University of Washington.

Ioannis Konstas and Mirella Lapata. 2013. A global model for concept-to-text generation. *Journal of Artificial Intelligence Research (JAIR)*, 48:305–346.

Percy Liang, Michael I. Jordan, and Dan Klein. 2009. Learning semantic correspondences with less supervision. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, pages 91–99.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, September.

Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. 2016. What to talk about and how? Selective generation using lstms with coarse-to-fine alignment. In *The 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 720–730.

Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proceedings of INTERSPEECH 2010, the 11th Annual Conference of the International Speech Communication Association*, pages 1045–1048.

Tomas Mikolov, Stefan Kombrink, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, (ICASSP 2011)*, pages 5528–5531.

Shinsuke Mori, Hirokuni Maeta, Tetsuro Sasada, Koichiro Yoshino, Atsushi Hashimoto, Takuya Funatomi, and Yoko Yamakata. 2014. FlowGraph2Text: Automatic sentence skeleton compilation for procedural text generation. In *Proceedings of the 8th International Natural Language Generation Conference*, pages 118–122.

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation Systems*. Cambridge University Press, New York, NY, USA.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 379–389.

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Meg Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. 2015. A neural network approach to context-sensitive generation of conversational responses. In *Conference of the North American Chapter of the Association for Computational Linguistics Human Language Technologies (NAACL-HLT)*, pages 196–205.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Henry S. Thompson. 1977. Strategy and tactics: a model for language production. In *Papers from the Thirteenth Regional Meeting of the Chicago Linguistics Society*, pages 89–95. Chicago Linguistics Society.

Zhaopeng Tu, Yang Liu, Zhengdong Lu, Xiaohua Liu, and Hang Li. 2016a. Context gates for neural machine translation. *CoRR*, abs/1608.06043.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016b. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 76–85.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-hao Su, David Vandyke, and Steve J. Young. 2015. Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1711–1721.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina Maria Rojas-Barahona, Pei-hao Su, David Vandyke, and Steve J. Young. 2016. Multi-domain neural network language generation for spoken dialogue systems. In *Proceedings of the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 120–129.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. *Proceedings of the 32nd International Conference on Machine Learning*, pages 2048–2057.