Modeling Recipe Steps Using Skip-Thought Vectors

Nicholas Hynes, Mandy Korpusik
MIT Computer Science and Artificial Intelligence Laboratory
{nhynes, korpusik}@mit.edu

Abstract

The generation of sequences of instructions remains an open problem in NLP. A large part of the challenge is to represent each step in a way that enables the construction of a robust model that supports operations like reordering and inference. To this end, we investigate the use of sentence-level embeddings known as skip-thought vectors in predicting the next and previous steps in a recipe. Trained on a corpus of 150k recipes with a total of 1.7m sentences, skip-thought vectors perform reasonably well as features for discriminating the next step, achieving an accuracy of 66.1% when given five candidate steps, but fare poorly when tasked with actually generating the next step. Additional experiments show that, while skip-thoughts vectors can be accurately decoded to the encoded step, there may be limits to their representational power.

1. Introduction

When preparing food under constraints–whether they be dietary restrictions, a sparse pantry, a desire for novelty, or even lack of any information about the dish other than its appearance–it is helpful to have a deep knowledge of how ingredients and steps can be combined and rearranged to make a recipe. As most people do not have the ability or time to acquire such knowledge, it would be advantageous if a machine could be trained to generate recipes automatically. Not only would such a system enable the generation of novel recipes, it could also aid in diet tracking and meal planning if provided with the necessary nutritional information and quantities. Going beyond the domain of food, though, the general techniques for sequence modeling could be generalized and applied to other problems including planning for autonomous agents–perhaps including those that actually *prepare* the food.

To this end, we take initial steps toward creating a model that can be used to generate a new recipe. Specifically, we investigate the use of sentence-level vector embeddings known as *skip-thought* vectors [3] as a basis for modeling recipe steps. Evaluating the embeddings on several tasks including identifying the next step and finding the location of a held-out instruction, we find that skip-thought vectors outperform credible baselines. We encounter limitations of the model, however, when attempting the most difficult task of generating recipe directions, which suggest that the encoded representations are too tightly coupled with the method by which they are created, thereby limiting general applicability.

Thus, the remainder of this paper presents related work in section 2, explains our approach to the problem of modeling recipes in section 3, describes the experimental setup and results in section 4, and concludes in section 5.

2. Related work

This section presents related work on skip-thoughts and recipe modeling, in general, while making clear the novel contributions of this work.

2.1. Skip-thoughts

Prior work has demonstrated the utility of deep neural networks for learning vector embeddings of words, sentences, and paragraphs through unsupervised training. One popular and effective formulation of the learning goal is the *skip-gram* model [9, 8], in which a held-out word is used to predict its surrounding context words. Skip-thought vectors [3] are the result of extending the skip-grams approach to the sentence level by training an encoder-decoder pair to predict the previous and next sentences. Additionally, it has been shown that paragraph vectors [4], representing variable-length documents, can also be learned in a similar manner; in this case, however, instead of using an encoder-decoder pair, the paragraph vector and several word vectors are concatenated and then used to predict the next word.

Recent work has explored the applications of skip-thought vectors in training neural memory networks [1] on a weakly-supervised question answering task by encoding the questions and statements with skip-thoughts.

Although the performance is lower than that of end-to-end systems, it nonetheless presents an interesting application of the technique.

In another line of inquiry, three multi-task learning approaches [5] were explored for sequence to sequence learning: one-to-many encoders to decoders, many-to-one, and many-to-many. It was then shown that machine translation results can be improved by the addition of skip-thought vectors.

This work also explores the application of skip-thought vectors but focuses on the problem of sequence modeling instead of question answering or translation.

2.2. Instruction sequence modeling

Previously, recipes have been modeled using graph-based methods. One example in this domain used a directed acyclic graph as a basis for a semantic parser for recipe text [6]. In the larger space of instruction sequence modeling, an end-to-end recurrent neural network (RNN) was shown to yield state-of-the-art results on mapping route-navigation instructions to actions [7]. Our work is set apart in that it uses vector representations, as opposed to graphs, that are encoded and decoded by *dis*jointly trained models.

3. Approach

Our approach for modeling recipe steps is based on skip-thought vectors [3], which are distributed representations of sentences learned by training an encoder-decoder model to predict the text surrounding a sentence. Thus, this section presents the skip-thoughts encoder and decoder and their application to the specific task of recipe modeling.

3.1. Skip-thoughts

The general concept of the skip-thoughts model is that of a disjointly trained encoder-decoder pair which operates on vector representations of entire sentences. The desired end results are sentence vectors that can be used in much the same way as the off-the-shelf embeddings produced by word2vec. For clarity we describe the model in detail here, but it will suffice to familiarize oneself with only the high-level introduction presented in each subsection describing the encoder, decoder, and training methods.

3.1.1 Encoder

The purpose of the encoder is to transform an input sentence into a fixed-dimensional vector that, ideally, captures the semantic content of both the sentence and its constituent words. At its core, the encoder is a RNN implemented by a gated recurrent unit (GRU), which is similar to LSTM but conceptually simpler. In a GRU, the output after each step is the new hidden state, h^t, which is a combination the previous hidden state h^{t-1} and a proposed state update, $\overline{h}^t$. $\overline{h}^t$, in turn, is derived from the input x and previous state. The contributions of the input, previous, and proposed states are mediated by the outputs of a reset gate, r^t, and a forget gate, z^t. Formally, the update equations are as follows:

$$r^t = \sigma(W_r x^t + U_r h^{t-1}) \tag{1}$$

$$z^t = \sigma(W_z x^t + U_z h^{t-1}) \tag{2}$$

$$\overline{h}^t = \tanh\left(W x^t + U(r^t \odot h^{t-1})\right) \tag{3}$$

$$h^t = (1 - z^t) \odot h^{t-1} + z^t \odot \overline{h}^t \tag{4}$$

where $\odot$ is the elementwise product and $W_{\{r,z,\cdot\}}$ and $U_{\{r,z,\cdot\}}$ are matrices of learned weights. A skip-thought vector, then, is simply h after observing every token in the input sentence.

3.1.2 Decoder

Unlike the encoder which bundles a sequence of tokens into a single vector, the decoder unrolls a single vector into a sequence of tokens, namely a sentence. Like the encoder, however, the decoder also incorporates a GRU, but is modified slightly to also take into account a context vector, h_i, when determining the gate outputs and proposed next state. While an obvious choice for h_i is the output of the encoder, we experiment with variants in which additional data are appended. Again, to be concrete, the state updates are given by

$$r^t = \sigma(W_r^d x^{t-1} + U_r^d h^{t-1} + C_r h_i) \tag{5}$$

$$z^t = \sigma(W_z^d x^{t-1} + U_z^d h^{t-1} + C_z h_i) \tag{6}$$

$$\overline{h}^t = \tanh\left(W^d x^{t-1} + U^d(r^t \odot h^{t-1}) + C h_i\right) \tag{7}$$

$$h_{i+1}^t = (1 - z^t) \odot h^{t-1} + z^t \odot \overline{h}^t \tag{8}$$

where $C_{\{r,z,\cdot\}}$ is a matrix of learned weights applied to the context vector.

3.1.3 Training

Since training the encoder also requires concurrently training decoders for the next and time-reversed previous steps, the training procedure for the decoder will be described first. For simplicity we focus on the forward decoder; training the backwards decoder proceeds analogously. The decoder's hidden state is used to represent the probability of the target step's words, conditioned on the context vector:

$$P(w^t|w^{<t}, h_i) \propto \exp\left(v_{w^t} h_i^t\right) \tag{9}$$

where v_{w^t} corresponds to the row of the vocabulary matrix associated with word w^t. The training objective is then to maximize the sum of the log-probabilities of each word in the sentence:

$$\sum_t \log P(w^t|w^{<t}, h_i) \tag{10}$$

When training the encoder, in the manner of skip-grams, the learning objective is to maximize the probability of decoding the previous and next step using the encoding of the current step. This is simply the sum of the objectives for the forward and backward decoders.

We now proceed to elaborate on the use of the encoded step vectors and decoder architecture in the task of modeling recipe steps.

3.2. Modeling recipe steps

Our ability to model recipes is based on three main assumptions: the first is that the vector encodings of semantically-similar steps are spatially related, next is that an encoded step can be decoded with reasonable accuracy to the original step, and the last, and perhaps most important, is that it is possible to learn a mapping from the encoding of a step to that of the next. In the remainder of this section, we validate the first two assumptions, leaving the third for more detailed treatment in the following sections on experiments and results.

The assumption of spatial organization is primarily related to the sub-task of predicting the next recipe step since, without this property, there would be no easy way to learn the required mappings. We qualitatively evaluate the skip-thought vector representations of recipe steps by encoding several test sentences and then finding their nearest neighbors based on cosine similarity. As shown in Figure 1, the results are intuitively correct, which serves to justify the assumption that skip-thought vectors, like word embeddings, transform semantic similarity into spatial similarity.

Successful recipe generation using skip-thoughts requires not only the ability to predict the next step's encoding, but also to undo the vectorization to produce human-readable text. To assess the capability of skip-thought vectors the latter regard, we perform a simple evaluation in which the steps in the test set (which is described in the next section) are encoded and subsequently decoded. For comparison, we construct a nearest-neighbor baseline that returns the text associated with the skip-thought vector nearest to the evaluated one, as measured by cosine similarity. The accuracy of the generated text with respect to the original is assessed using BLEU [10], a modified n-gram precision metric that is commonly used in similar machine translation tasks. In Table 1, below, we present the average BLEU score using uni-, bi-, and tri-grams for both text production methods.

Cut the chicken thighs into bite-sized pieces.
Cut chicken into 3/8-inch-square pieces.
Cut chicken and pineapple into bite-sized pieces.
Cut the chicken into cubes.

Heat the olive oil in a large saucepan.
Heat the olive oil in a large saucepan.
Heat the oil in a large saucepan.

Add the tomato, chillies, pine nuts, currants and parsley .
Add the bay leaves, thyme and parsley.
Add the cloves, bay leaf, thyme and parsley.
Add the crawfish tails, cheese, green onions and parsley.

Figure 1. Examples of nearest neighbors to three test sentences when encoded as skip-thoughts vectors.

In a medium bowl, whisk together the flour, baking powder, cinnamon, nutmeg, baking soda, salt and ginger.
In a medium bowl, whisk together the flour, baking powder, baking soda, salt, ginger and cinnamon.

Pour the wet ingredients into the dry and fold to blend well.
Pour the wet ingredients into the dry and fold together well.

Fill the cups of the muffin tin about halfway with the batter.
Fill the muffin tin halfway with the batter.

Figure 2. Examples of original and decoded recipe steps.

Judging from the results in Figure 2 and Table 1 the skip-thoughts encoder-decoder pair performs well at its intended task. Subsequent sections, however, will present evidence suggesting that even slight modifications to the setup yield significantly degraded results.

Model	n-grams	Avg. BLEU
baseline	1	.671
	2	.609
	3	.576
decoder	1	.781
	2	.692
	3	.610

Table 1. BLEU scores for unigrams, bigrams, and trigrams averaged of each encoded-decoded sentence pair in the test set. The baseline is the text of the nearest encoded neighbor.

4. Experiments & Results

In this section, we present the dataset, the models trained on it, tasks in which the models were used, and the results by which we evaluate the effectiveness of skip-thought vectors for modeling recipe steps.

4.1. Dataset

For this work, we have assembled a corpus of 148k recipes and their 337k photos by scraping several recipe websites. A minimal recipe consists of a title, set of ingredients—possibly segmented into quantity, unit, and name—and an ordered list of directions. On average, a recipe contains ten ingredients and twelve instructions. Furthermore, most (93k) include at least one image, and almost all (128k) list the cooking time. To maintain general applicability of models trained on this dataset, the only pre-processing that has been done is to replace unicode characters with ASCII equivalents (e.g. $\frac{1}{4}$ becomes 1/4).

Additionally, the data were randomly partitioned by recipe into training, validation, and test subsets using a 4:1:1 ratio. Care was taken to avoid data contamination.

4.2. Models

In the following experiments, we use 600-dimensional skip-thought vectors trained using a vocabulary containing the most frequent 30,000 words (out of 55k), each of which which is embedded as a 300-dimensional word vector learned separately for each encoder and decoder.

On top of the step encodings we use linear regression models. Neural and regularized linear models were also evaluated but found to perform similarly to the simpler, linear model.

4.3. Experiment 1: Identify previous/next step

In the first experiment, a linear model is used to predict the embedding of either the previous or next step from the embedding of the current step. The resulting vector is then compared using cosine similarity to the actual next-step vector and several other vectors sampled from either within the same recipe or from different recipes. The highest scoring vector is then taken as the previous/next step. Evaluation is done using the metrics of mean accuracy and mean rank of the target step in the list of steps sorted by similarity (1 is best).

Task	Acc.	Mean rank
next, 4 same (baseline)	32.2	2.9
next, 4 same	56.6	1.8
prev, 4 same	57.5	1.9
next, 4 different	66.1	1.6
prev, 4 different	68.9	1.6
next, 9 same	42.2	2.8
prev, 9 same	44.3	2.9
next, 9 different	52.1	2.3
prev, 9 different	57.5	2.3
next, 100 different	0.26	15.3
prev, 100 different	0.06	30.5

Table 2. Accuracy and mean rank for identifying the adjacent step among four or nine candidates from within the same recipe or from different recipes. The baseline uses bag-of-words features instead of skip-thought vectors.

Compared to the baseline predictor, the linear models fare reasonably well. Moreover, the data suggest that performance begins to plateau when more candidate steps are added. These findings, taken together, support the use of skip-thought vectors in modeling recipe steps.

The results of this experiment also suggest that the relationship between the encodings of adjacent steps is largely linear. As the third experiment will demonstrate, though, much of the latent semantic content is contained within the non-linear kernel.

4.4. Experiment 2: Locate missing step

This next task involves identifying the position in a recipe from which a step was omitted and is motivated by the task of filling in omitted details in a recipe. We compare two methods of prediction:

- context - select the location where the next-step prediction of step $i-1$ and the previous-step prediction of step $i+1$ are most similar
- held-out - using the models trained in the previous experiment, select the location where the next-step and previous-step predictions of the held-out step best match (via cosine similarity) the actual previous and next step encodings

Again, we evaluate the models and prediction methods using accuracy and mean rank.

Model	Acc.	Mean rank
baseline, 5 locs	20.7	3.0
context, 3 locs	54.9	1.6
context, 5 locs	37.4	2.3
context, 7 locs	29.5	3.0
held-out, 3 locs	71.7	1.4
held-out, 5 locs	62.9	1.6
held-out, 7 locs	56.7	2.0

Table 3. Accuracy and mean rank for identifying the location of a missing step from among three, five, or seven candidate locations using either the context steps or the next/previous-step predictions of held-out step, itself. The baseline uses a random encoding.

While the models are less capable of positioning a missing step than simply identifying an adjacent step, they still show improvement over the baseline. We hypothesize that the discrepancy between the context and held-out variants of the task is due to systematic dissimilarity between the predictions in the forward and backward directions. This effect may also be observed in the results of the previous experiment which is manifested as previous-step prediction performance being generally superior to the reverse. In both cases, however, the exact cause remains to be determined.

4.5. Experiment 3: Generate next step

The third experiment involves generating the next step by directly transforming the current step vector into the next using the previously-described linear model and then decoding the result. This is a direct evaluation of the third assumption of skip-thoughts made in the Approach section: that a mapping from current to next step vector can be learned. Again, the metric used here is BLEU.

n-grams	Avg. BLEU
1	0.23
2	0.11
3	0.00

Table 4. BLEU scores for steps obtained by decoding next-step encoding predicted by linear model.

From the results, as shown in Table 4, it is clear that, although a prediction made by the linear model is near the target skip-thought vector, this is not enough to produce the correct decoded step. This suggests that the function learned by the encoder is highly non-linear and motivates yet further work investigating the use of a neural next-step predictor.

4.6. Experiment 4: Decode next step

The fourth and final experiment seeks to determine whether the skip-thoughts architecture, itself, can be modified to support generation of the next step by training the decoder to output not the encoded step, but rather the following step. Results are evaluated qualitatively since BLEU scores would be near-meaningless given the divergence between the target and generated steps. Example outputs are shown in Figure 3 below.

Form the beef into twelve 4-ounce patties, about 1/2 inch thick
Brush the grates of the grill with oil.

Reduce oven temperature to 300 degrees F (150 degrees C)
Store in an airtight container.

In a small bowl, mix milk with 2 tablespoons warm water
Preheat oven to 350 degrees F (150 degrees C)

Stir and cook until zucchini is tender, about 5 minutes more.
Cook, stirring, until vegetables are tender, about 5 minutes.

Figure 3. Example output of decoder trained to decode the next step. In bold is the target output; below is the step generated by the decoder.

Somewhat surprisingly, the decoder is largely unable to generate the target outputs despite that being half of the objective when encoding the step. Moreover, the tendency is for the decoder to yield the most common steps like "Preheat the oven" or "Remove from heat and let cool," which parallels the failure mode of an overfit n-gram model.

We also experimented briefly with appending a 100-dimensional continuous bag-of-words vector over the recipe ingredients to the decoder context vector in hopes of improving performance. This offered no benefit to the next-step decoder and even *reduced* the performance of the same-step decoder, as measured by a BLEU score drop from 0.75 to 0.71. Thus, it seems as though, while skip-thoughts do an excellent job of encoding and decoding the current step, they struggle to do much else.

5. Conclusions and Future Work

In this work, we have demonstrated the utility of skip-thought vector encodings for modeling recipe steps by first verifying that they can be accurately decoded to the encoded step, evaluating their use as features for

predicting adjacent recipe steps and the location of a missing step, and finally exploring their ability to be transformed and decoded into the next recipe step.

While the lack of positive results for the last set of tasks suggests that there may be a fairly low limit to the representational power of skip-thought vectors, further work involving neural architectures for next-step generation may show that this is not the case.

Alternatively, it may also be more feasible to try other, more established methods taken from domains such as machine translation [11, 2] that rely on jointly training an encoder-decoder pair end-to-end, sequence to sequence.

As a final note, all code related to this project is made publicly available at `https://github.mit.edu/nhynes/pomme`.

6. Acknowledgements

We thank the 6.864 course staff for their attentiveness to both identifying and subsequently attempting to remedy any issues that we encountered.

References

[1] E. Caballero. Skip-thought memory networks. *arXiv preprint arXiv:1511.06420*, 2015.

[2] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[3] R. Kiros, Y. Zhu, R. Salakhutdinov, R. Zemel, A. Torralba, R. Urtasun, and S. Fidler. Skip-thought vectors. *arXiv preprint arXiv:1506.06726*, 2015.

[4] Q. Le and T. Mikolov. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053*, 2014.

[5] M. Luong, Q. Le, I. Sutskever, O. Vinyals, and L. Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

[6] H. Maeta, T. Sasada, and S. Mori. A framework for procedural text understanding. In *Proc. IWPT*, pages 50–60, 2015.

[7] H. Mei, M. Bansal, and M. Walter. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. *arXiv preprint arXiv:1506.04089*, 2015.

[8] T. Mikolov, K. Chen, and J. Dean. word2vec (2013).

[9] T. Mikolov, W. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *Proc. HLT-NAACL*, pages 746–751, 2013.

[10] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[11] I. Sutskever, O. Vinyals, and Q. Le. Sequence to sequence learning with neural networks. In *Proc. NIPS*, pages 3104–3112, 2014.