Building Hierarchically Disentangled Language Models for Text Generation with Named Entities

Yash Agarwal*
NSUT, New Delhi
yasha.co.17@nsit.net.in

Devansh Batra*
NSUT, New Delhi
devanshb.it.17@nsit.net.in

Ganesh Bagler
IIIT-Delhi, New Delhi
bagler@iiitd.ac.in

Abstract

Named entities pose a unique challenge to traditional methods of language modeling. While several domains are characterised with a high proportion of named entities, the occurrence of specific entities varies widely. Cooking recipes, for example, contain a lot of named entities — viz. ingredients, cooking techniques (also called processes), and utensils. However, some ingredients occur frequently within the instructions while most occur rarely. In this paper, we build upon the previous work done on language models developed for text with named entities by introducing a Hierarchically Disentangled Model. Training is divided into multiple branches with each branch producing a model with overlapping subsets of vocabulary. We found the existing datasets insufficient to accurately judge the performance of the model. Hence, we have curated 158,473 cooking recipes from several publicly available online sources. To reliably derive the entities within this corpus, we employ a combination of Named Entity Recognition (NER) as well as an unsupervised method of interpretation using dependency parsing and POS tagging, followed by a further cleaning of the dataset. This unsupervised interpretation models instructions as action graphs and is specific to the corpus of cooking recipes, unlike NER which is a general method applicable to all corpora. To delve into the utility of our language model, we apply it to tasks such as graph-to-text generation and ingredients-to-recipe generation, comparing it to previous state-of-the-art baselines. We make our dataset (including annotations and processed action graphs) available for use, considering their potential use cases for language modeling and text generation research.

1 Introduction

Language Modeling is an important task in Natural Language Processing with several applications (Wiseman et al., 2017) including auto complete (Arnold et al., 2017) among others. A key application of language modeling is text generation (Semeniuta et al., 2017). Recent advances in language models have led to impressive generation of programming code in languages such as Java and C (Hindle et al., 2016; Yin and Neubig, 2017; Hellendoorn and Devanbu, 2017; Rabinovich et al., 2017). Similarly, efforts have been made towards application of language models in the context of cooking recipes (Parvez et al., 2018; Kiddon et al., 2016). Both these tasks, recipe and code generation, deal with corpora heavily populated with named entities. In case of cooking recipes these entities are thousands of ingredients used in recipes, hundreds of cooking processes which can be applied upon them and tens of utensils used across the different cuisines. These numerous entities immensely increase the vocabulary of the language model. Typically, the vocabulary of corpora of cooking recipes is of the order of tens of thousands of tokens. The vocabulary of the corpus we prepared (more details will follow later in the manuscript) has

*Yash Agarwal and Devansh Batra contributed equally to the work presented in this manuscript.

Proceedings of the 28th International Conference on Computational Linguistics, pages 26–38
Barcelona, Spain (Online), December 8-13, 2020

about 54 thousand tokens and 6 thousand ingredients, which form the single largest entity type. It is amply clear that the language model has to deal with the language semantics comprising of thousands of English language tokens along with the way ingredients, utensils and processes interact with each other.

Parvez et al. (2018) provide credence to this argument. For instance, fruits occur 27,260 times in their corpus but the fruit 'apple' occurs only 720 times. And fruit is just one kind of ingredient. To some extent, all fruits can be modeled as one entity, as all of them largely behave in the same manner when written in a natural language. The processes like cutting and peeling are common to all fruits. Thus (Parvez et al., 2018) propose using one entity level language model which predicts whether a fruit or a vegetable should occur at a given place, followed by a model specific to the entity type which predicts the probabilistic distribution of which exact fruit fits well. They show that this method greatly influences the quality of text generation for both recipe and code generation owing to the reduction in vocabulary and complexity for the models being trained.

We note that programming languages have far less ambiguities and variations than natural language because they are meant to be compiled using a detailed protocol. Due to this reason, the language models perform far better on code generation than natural language generation tasks even though the former has a lot of data types and variable names (which work like the entity types and their name). These models have already achieved a BLEU score as high as 42.39 (Wei et al., 2019) and models by (Parvez et al., 2018) achieved a validation perplexity as low as 2.39 (a perplexity of 1.0 would mean that the cross-entropy loss is zero and is theoretically the best result). Thus, we feel that any further improvement in the language model would be hard to quantitatively estimate for programming languages by existing metrics, which is why we focused on corpora of cooking recipes.

There have been several advances in the field of recipe text generation. The first implemented system was called EPICURE (Dale, 2019) and was a purely rule based generation algorithm. (Cromwell et al., 2015) developed a system for generating salads using statistical search ingredient combinations. (Kiddon et al., 2016) developed a modern neural recipe generation system by maintaining a state of available and used state. Despite that it had a low BLEU score. More recently, the release of Recipe1M+ (Marin et al., 2019), has led to the development a related domain of recipe retrieval — retrieving recipes from a large scale dataset (Zhu et al., 2019; Salvador et al., 2017). In the past year, we have seen two new applications — recipe generation from image (Salvador et al., 2019), and personalised recipe generation (recommendation) based on user reviews (Majumder et al., 2019). The problem of ingredient to recipe generation is also tackled by many of these papers.

Aligned with the arguments made earlier, we decided to create our own dataset (which was further cleaned and annotated) for the development of the language model. Taking inspiration from (Kiddon et al., 2015), we apply an unsupervised recipe interpretation pipeline to model the instructions as action graphs along with Named Entity Recognition models which describe in detail in Section 2.

The language model is itself described in Section 3. We further study some of the applications of our language models, testing it for tasks such as graph-to-recipe generation and ingredients-to-recipe generation. Image-to-recipe generation as beyond the scope of this paper since (Salvador et al., 2019) shows that clues from images are a big help for the language model in predicting the recipe.

2 Dataset

2.1 Need for a new dataset

The existing research explorations have used a variety of datasets for their experiments. (Kiddon et al., 2016) and (Parvez et al., 2018) employ the 'Now You're Cooking!' dataset containing 150,000 recipes. However, as also pointed out by (Kiddon et al., 2016), many of them do not follow a uniform format, have missing instructions or are duplicates. This leads to only 84,590 usable recipes. Both these papers provide their preprocessed datasets. However, (Parvez et al., 2018) removes all instances of numerals within the text without citing a reason. While this may been done to avoid the serial number of the steps in recipes, removing numbers takes away important instances of quantities of ingredients. In our experiments with this dataset, we observed several instances like "keep boiling for minutes". (Kiddon et al., 2016) outrightly stripped away all amount information, including the unit, (e.g. '1 ts') Clearly, this

step makes the modified dataset unusable for generation of viable recipes.

We believe that the original dataset is heavily based on AllRecipes and is insufficient in itself to get all entities. In this paper, we aimed to make a plug and play language model for cooking recipes which can be used for several applications such as ingredient-to-recipe generation, image-to-recipe generation, action-graphs-to-recipe generation, translation of recipe between languages etc., such that only the encoder would differ in all these use cases. For this purpose, we require a dataset with a wide coverage of different types of entities as well as the entities themselves. We believe that a dataset of 84k recipes is far too insufficient for such a purpose.

A recently released related dataset by (Majumder et al., 2019) with 180,000 recipes may be of more value; but all of these recipes come from a single source - viz. Food.com. We may have used it for our purpose. However, by the time the paper and the dataset were made public, we were already done with the gathering and annotation of our dataset.

2.2 Entities in a Cooking Recipe

The modified dataset provided by (Parvez et al., 2018) set the entity types as categories of ingredients - fruits, vegetables, proteins, etc. However, we have strong contentions against such categorisation. All of these entities are ingredients, and are likely to behave in similar way. We delve into this more in a following section 2.3. In an instruction, generally ingredients are subjected to cooking processes inside cooking utensils in some quantity for a specific duration of time. Thus ingredients, cooking processes, utensils, quantity, and duration are the fundamental level of entities we encounter. This is also how cooking recipes have been modeled traditionally (Mori et al., 2014; Kiddon et al., 2015; Diwan et al., 2020).

We also note that the number of ingredient entities (537) considered by (Parvez et al., 2018) is very low compared to the number of ingredients occurring in large scale databases such as RecipeDB (Batra et al., 2019), FlavorDB (Garg et al., 2018) which record around 3 to 4 thousand ingredients clearly pointing to incomplete entity recognition.

2.3 Data collection and processing

We compiled 158,473 recipes from 8 websites and 70 geo-cultural cuisines, as detailed in Table 1. To cover all entities, we attempted to maximise the number of cuisines as ingredients, processes and utensils are often specific to certain cuisines. This greatly increased the number of entities.

Pre-processing ingredients: Diwan et al. (2020) argue that the Ingredients section may contains additional information about the freshness, state, temperature and quantity. Since our focus is on generating recipes from cooking ingredients, we filter out these extraneous qualifiers to keep only ingredient names so as to avoid confusing the model. To accomplish this, we used pretrained models made available by the authors which accurately predict freshness, state, temperature and quantity with F1 scores of over 0.95. This was partly motivated by the fact that the baseline model for our experiments (Majumder et al., 2019) also uses only ingredient names (and no other information about freshness, quantity, etc).

Identifying entities in instructions: For identifying the five entities in instructions section, we train Stanford's Named Entity Recogntion (NER) Tagger (Finkel et al., 2005) on a manually annotated corpus. The training set comprised of 33,000 words. We tested it on a manually annotated test set containing 4,236 words. We make the pre-trained models available to use with the evaluation summarised in Table 2.

Interpreting Action Graphs Another possible use case of the language model could be data-to-text, (Puduppully et al., 2019) or graph-to-text generation (Guo et al., 2019; Koncel-Kedziorski et al., 2019). As mentioned before, instructions in cooking recipes can be modeled of as action graphs. Each action (or event) is identified by a cooking process. These actions are applied upon entities such as ingredients and utensils or a combination thereof (several ingredients processed and stored in a utensil). The relationship is usually given by an edge between the process and these entities. Such action graphs closely resemble Resource Description Framework (RDF) Graphs, which are an active research interest for graph-to-text applications.

Source	Number of Recipes
AllRecipes	17508
Epicurious	11165
Food Network	15917
Food.com	101992
Kraft Recipes	195
My Korean Kitchen	203
Taste (Australia)	7637
The Spruce	3856

Table 1: Breakdown of sources from where cooking recipes were compiled

Entity	F1 score
Ingredients	0.944
Cooking Processes	0.912
Utensils	0.945
Quantity	0.962
Duration	0.968

Table 2: Evaluation of NER inference

We applied a semi-supervised method to interpret our recipes as action graphs, largely inspired by the excellent work done by (Kiddon et al., 2015) with a few adjustments. A detailed explanation of the full method which deals with several probabilistic models is out of the scope of this paper; we highly recommend referring to the mentioned paper. Following are the changes we made to the method:

1. The paper reports an F1 score of 95.6%, learnt and tested on a limited set of 2456 recipes, for identification of verbs which were used to define the actions. On our manually annotated test set for identifying cooking process entities, the method achieves an F1 score of only 78.6%. We swapped the dependency parser for a more recent Shift-Reduce parser in the Segmentation Model. All verbs were identified using a POS tagger along with all words which were also tagged as cooking processes by our NER model. This raised the F1 score to 93.7%, significantly higher than using only NER inference (91.2%) for Cooking Processes. These verbs were then used as processes identifying the actions to form the action graphs.

2. For identification of 'ingredient spans' in the pre-processing phase, we rely upon the NER tags. Some of the missed occurrences of ingredient names were retrieved from the ingredient names retrieved from the preprocessed ingredients section, as described above, due to absence of evaluation metrics for the heuristic-based method proposed in the above paper.

3. While we do provide the action graphs obtained, as DOT (.gv) files, for all the recipes in the dataset to aid further research, the GCN based data-to-text model is trained sentence-by-sentence to match the baseline. So we only make use of 'partial' action graphs - those obtained before the sequential connections across consecutive sentences are joined. The action graphs and partial action graphs are demonstrated in the supplementary section.

3 Language Model

As mentioned before, we build upon the work done by (Parvez et al., 2018) pertaining to the domain of building language models for natural language text containing high proportions of named entities. Intuitively, the reason why language models suffer for such text can be understood through the following observations -

- Named entities can be very high in number, with a small proportion of common English words.

- The common words in English language follow set patterns with limited deviations. These patterns, formalised by the English grammar, do not change with training documents. Thus they are easy to model if we have a corpus of millions of words of naturally occurring text. Consider the case when the documents on which the model is being trained upon are cooking recipes. Every document would have different entities appearing in them. And the patterns would change too. Very few recipes involve lining up a dough on a baking sheet and putting it in a preheated oven. Fewer recipes would involve spreading a pizza dough on a grill.

- Thus, a few hundred entities can co-occur together in millions of different ways and complex relationships. It can be seen that the problem arises from the fact that the text has a lot of named entities to learn about, but not sufficient occurrences of these individual entities to successfully learn all the patterns from their context.

A way to mitigate this problem would be to separate the learning of entity type (ex: fruit) specific knowledge, from entity (ex: apple) specific knowledge. In more intuitive words - all entities of the same type behave in certain similar ways. While it may be uncommon for a recipe to include "heat pizza dough on a grill", if we were to label all ingredients, cooking process and utensils, a pattern like "COOKING_PROCESS INGREDIENT on a UTENSIL" is far more likely to be found. Thus it would be easier for a language model to learn such combinations of patterns because of increased occurrences and a decreased vocabulary since it doesn't have to consider the thousands of individual tokens belonging to these categories. (Parvez et al., 2018) introduced a "Type Model" to learn this behaviour.

In the subsequent operation, the model (or a separate one) can try to learn what specific entity should appear at a place. Given the context, it is easier to tell what cooking process or utensil or ingredient one should use. Consider a recipe starting with "COOKING_PROCESS a UTENSIL before you begin". A model can easily learn by seeing several examples that in such cases when a COOKING_PROCESS is applied at the start of the instructions section, the most frequent process is "Preheat". The input sentence to the language model now becomes "Preheat a UTENSIL". Strictly speaking, an oven would be a kitchen appliance, and not a utensil. But in the context of discussion, it has been deliberately marked as a utensil and is undoubtedly the best fit here. (Parvez et al., 2018) introduced an "Entity Composite Model" to learn this behaviour. During inference, it relies on the output of the "Type Model".

For simplicity, the trivial example above includes only three entities, we later explain our experiments with the higher more number of entities and their effects. Also, only backward context was considered here, which may indeed be the case when, say, a forward LSTM (Hochreiter and Schmidhuber, 1997) is used for learning the language model. However a bi-directional LSTM (Graves et al., 2005) allows much better performance, as also suggested by by (Parvez et al., 2018).

3.1 Type Disentangled Language Models

We appreciate the improvement in perplexity achieved by learning the 'entity type specific knowledge' and 'entity specific knowledge' separately. However, unlike (Parvez et al., 2018), we attempted to avoid learning two different language models and took a joint inference for the following reasons

1. Learning two different models from scratch for a single task is computationally expensive.

2. Also, the entity composite model makes use of the output of type model to give the joint inference. So, the parameters of type model are learnt separately and the loss function (used for learning the type model) does not directly affect this joint inference in any way (which is the desired result to be optimised in the first place).

3. A two model framework would be difficult to extend or interpret as a knowledge base in itself (Petroni et al., 2019). Similarly, a rigid framework which requires two models with related vocabularies would be difficult to tune for transfer-learning on a different corpus. We believe transfer learning could be helpful in this context, as a lot of tasks include corpora with named entities. Recently, transfer-learning to adapt pretrained representations for different tasks has been highly encouraged with the release of algorithms like ELMo (Peters et al., 2019) and benchmarks like GLUE (Wang et al., 2018).

Inspired by (Vondrick and Torralba, 2017), we see the entity composite model as a transformation for predicting the next word, using the context from previous tokens and the type of the next word being predicted (given by the type model). Since the entity composite model itself is differentiable, we decide to change the architecture into a single model like in (Vondrick and Torralba, 2017) which would allow training the whole architecture together.

We also change the nomenclature to type prediction network (instead of type model) and entity transformation network (instead of entity composite model) to further reinforce the fact that we aren't training two separate language models. Formally, the two networks work as follows:

We want to model the probability distribution of the next word w when we know the previous context $\bar{w}$:

$$P\left(w|\bar{w};\theta_t,\theta_e\right), \tag{1}$$

here, θ_t and θ_e are the learnable parameters of the type prediction network and entity transformation network respectively.

More accurately, the Type Prediction Network network models the probability distribution of types $s(w)$ where w is the next word, given the context and type information of the previous words. Thus, the overall problem becomes the estimation of

$$P\left(w, s(w)|\bar{w};\theta_t,\theta_e\right). \tag{2}$$

In our model, the prediction of w and $s(w)$ is done by two separate networks. Thus, the problem simplifies as follows:

$$P\left(s(w)|\bar{w};\theta_t\right) \times P\left(w|\bar{w}, s(w);\theta_e\right) \tag{3}$$

Whereas the first term in Equation 3 is approximated by the Type Prediction Network, the second term is approximated by the Entity Transformation Network.

NOTE: This is in contrast to (Parvez et al., 2018) who approximate both of these terms differently in their "Type Model". They treat the Type Model as a language model working on types, taking the entity types of the previous tokens as input to predict the entity type of the next word. They also use the type of information of all the previous words to predict the next word in the Entity Composite Model while we only use the predicted type of the next word received from the Type Prediction Network. In our evaluations, this change greatly improves the performance of our network 3. The approximation for the first term used by (Parvez et al., 2018) is -

$$P(s(w)|\bar{w}, s(\bar{w});\theta_t,\theta_v) \approx P(s(w)|s(\bar{w}),\theta_t) \tag{4}$$

and that of the second term is -

$$P(w|\bar{w}, s(\bar{w}), s(w),\theta_v) \tag{5}$$

where θ_t and θ_v are the parameters of their type and entity composite models respectively. Interestingly, both the estimations would behave identically for all tokens which do not belong to a type (in most corpora, normal English words). In that case both the methods use the approximation $s(w) = w$. This change causes subtle difference in all the equations derived below.

Type Prediction Network: This network predicts the entity type of the next word. In case the next word does not belong to an entity, we output $s(w) = w$. It estimates the first term in the equation 3.

$$P(s(w)|\bar{w}, s(\bar{w});\theta_t) \approx P(s(w)|\bar{w},\theta_t) \tag{6}$$

Stacked Hierarchical Type Prediction Network: To handle cases where entities may be hierarchically classified into two or more levels, we experimented with a modification of the Type Prediction Networks. This will be helpful in dealing with entity types containing large number of entities. A recipe may have entities like Ingredients and Utensils. Our corpus contains 3022 ingredients and 140 utensils. Ingredients may further be divided into fruits, vegetables, proteins, etc. Fruits may further be divided into sub-entity types and so on until we finally arrive at a named entity. This way, at each level of categorisation, the vocabulary of tokens to be considered is decreased. To accomplish this, we stack Type Prediction Models with different vocabularies one after the other to go down a level each time and finally output an entity type at the lowest level. For a 3-layered Entity Hierarchy,

$$P(s(w)|\bar{w};\theta_t) \approx P(s_1(w)|\bar{w},\theta_{t_1}) \times P(s_2(w)|\bar{w}, s_1(w),\theta_{t_2}) \times P(s_3(w)|\bar{w}, s_2(w),\theta_{t_3}) \tag{7}$$

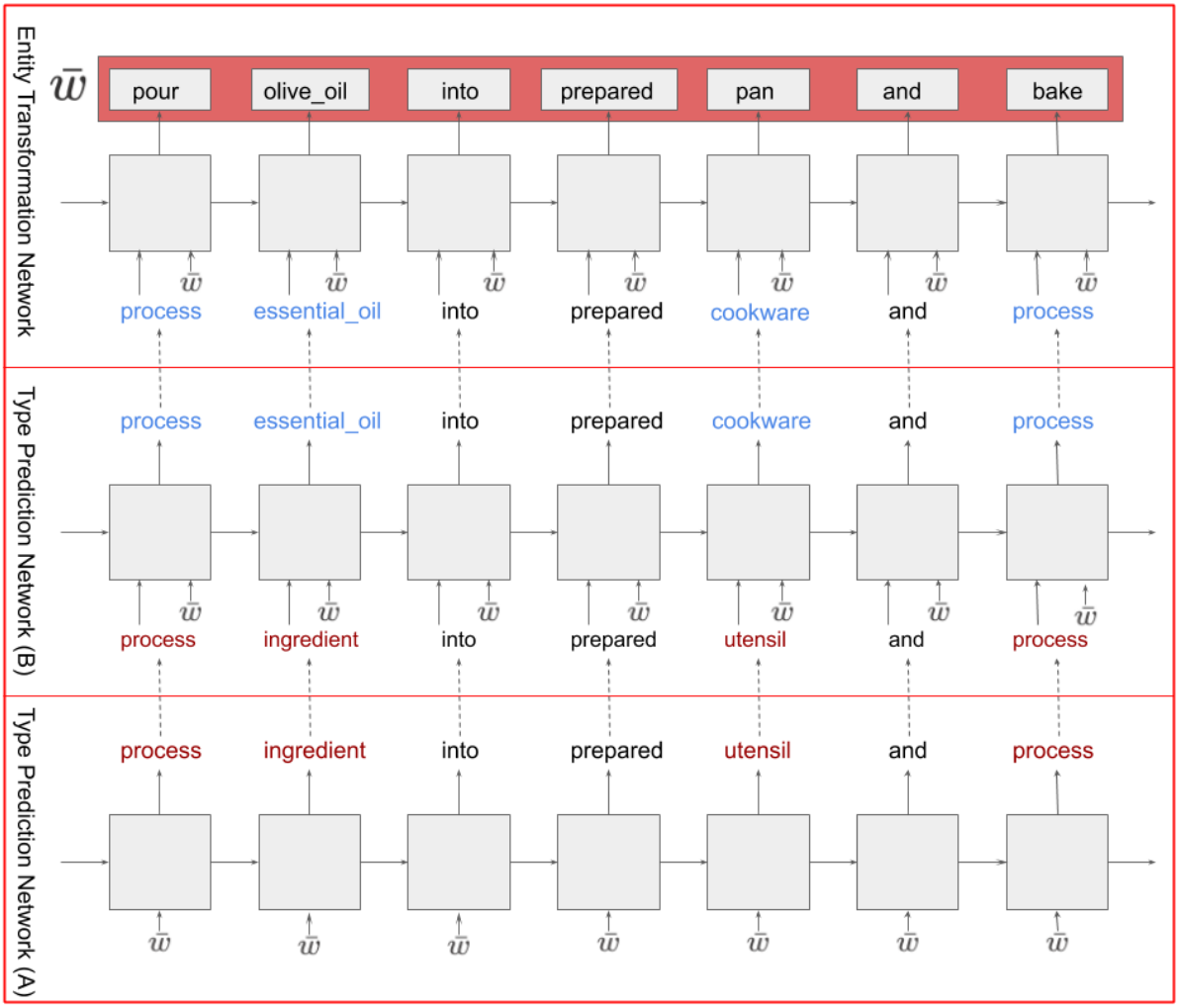

Figure 1: The inputs and outputs of the proposed networks during pretraining are illustrated. The two Type Prediction networks together form the Stacked Hierarchical Type Prediction Network. During fine-tuning and inference, the tokens are directly fed to the subsequent network as denoted by dotted lines.

Entity Transformation Network: This network also uses RNN layers (in our experiments, we have utilised the QRNNs (Bradbury et al., 2017)) to predict the next word, given the predicted type for this word $s(w)$ and the context $\bar{w}$. It estimates the second term in the Equation 3.

$$\frac{P(w|\bar{w}, s(w); \theta_e)}{\sum_{w_s \in \Omega(s(w))} P(w_s|\bar{w}, s(\bar{w}); \theta_e)}, \tag{8}$$

where $\Omega(s(w))$ is the set of words of the same type with w. Figure 1 graphically illustrates the networks.

3.2 Training

Unlike (Parvez et al., 2018), all networks use the same vocabulary. In the gathered corpus of cooking recipes, we first train both networks separately using Cross Entropy Loss on QRNNs so that the models learn to give the right output individually. For this purpose, we provide datasets tagged with entity type, subtype and ground truth. We finally fine-tune the whole architecture. For this, we follow the approach used by (Xu et al., 2019); training one sub-network at a time at a low learning rate and freezing the other sub-networks. This is done for many iterations. The performance gain is significant, as documented in Table 3. In all experiments involving the proposed dataset, a uniform training-validation-test split was used (126,778 recipes in training set and 15,847 recipes in both validation and test sets). For reproducibility, we have provided more details in the appendix.

Training Phase	Ppl.
Pretrained Type Prediction N/W	5.20
Pretrained Subtype Prediction N/W	1.34
Pretrained Entity Prediction N/W	1.44
Fine-tuned End-to-end Model	9.25

Table 3: Test perplexity while training the proposed stacked hierarchical type model on our dataset. Two instances of type prediction models were stacked. The highest level entities are Ingredients, Utensils, Cooking Processes, Duration, and Quantity. Ingredients and Utensils are further classified into 22 and 3 different sub types respectively.

Model	Dataset	Number of Named Entities	Perplexity
Parvez et al (Baseline)	*Parvez et al*	537	9.67
Our model	*Parvez et al*	537	9.22
Parvez et al	ours	3608	14.89
Our model with hierarchical types	ours	3608	**9.25**

Table 4: Comparison of the performance of our language model with the baseline. All the results are on the test set of the corresponding corpus. Lower perplexity is better.

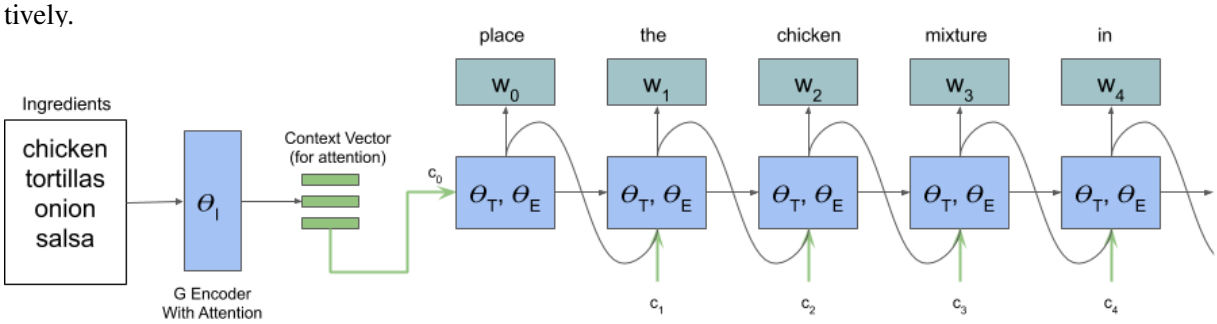

Figure 2: Architecture for the model used for ingredient-to-text generation (Experiment 4.1).

3.3 Evaluation

Since our work heavily builds upon (Parvez et al., 2018), we treat it as the baseline. We evaluate both models on both the datasets - the one provided by them as well as the one we procure. While evaluating on their dataset, we use the entity types (8 superingredients) they provide. On our dataset, we compare it with a model trained on 28 entity types (22 categories of ingredients, 3 categories of utensils, cooking processes, quantity and duration). No hierarchical types were included since their architecture does not support hierarchical entity classification. The results have been summarised in Table 4. To the best of our knowledge, there are no other language models which aim to improve the perplexity for text with named entities.

4 Experiments

4.1 Ingredient to Recipe Generation

We apply our language model on the practical task of generating cooking recipes from given ingredients. This task is covered by (Kiddon et al., 2016), (Majumder et al., 2019) and (Salvador et al., 2019). The latter two papers point out that their models, which use user review history and recipe images for generating the recipe (and not just ingredients), result in a smaller perplexity than simple encoder decoder frameworks. Because they require these additional information, we cannot use these models as baselines.

Majumdar et al. (2019) do not use (Kiddon et al., 2016) as their baseline because an Encoder Decoder by (Cho et al., 2014) framework, incorporating attention, achieves a similar performance at a much lower complexity. Due to this, we have used the baseline (GRU encoder + GRU decoder) provided by them.

We modify our architecture, as illustrated in Figure 2, to use the same ingredient GRU encoder as the baseline, followed by the pretrained Type Prediction and Entity Transformation Networks. We then incorporate Bahdanau attention (Bahdanau et al., 2015) to add the embeddings from the encoder as an additional weighted input to both the Type Prediction and Entity Transformation Networks and finally fine-tune the whole architecture for an ingredient-to-recipe generator. For simplicity, we only used a

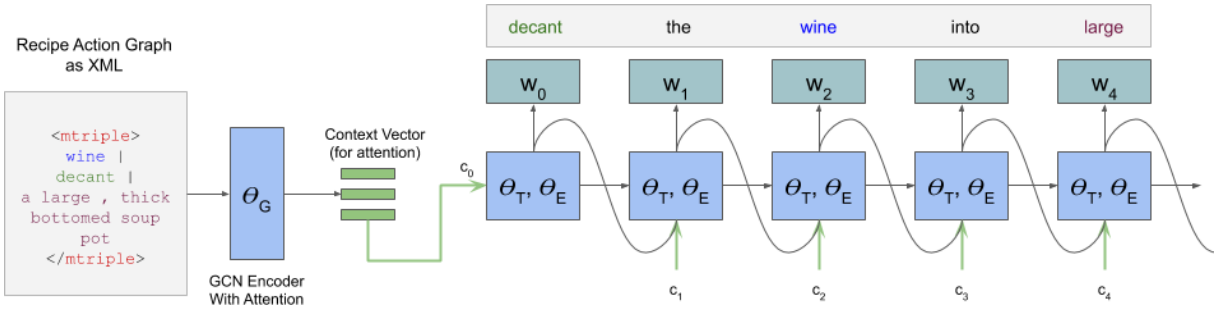

Figure 3: Architecture for the model used for action-graph-to-text generation (Experiment 4.2).

Model	Dataset (Recipe Corpus)	Number of Recipes	Perplexity
Baseline (EncDec)	Ours	158k	10.09
GRU Encoder with our model	Ours	158k	8.99
Personalised Recipe	FOOD.com	180k	9.52
Image to Recipe	Recipe1M	1M	8.66

Table 5: Comparison of the performance of ingredient-to-recipe generation task.

Model	BLEU Score
Seq-to-seq baseline *LSTM EncDec*	43.79
GCN Encoder LSTM Decoder	62.14
GCN Encoder Our Decoder	**63.57**

Table 6: Comparison of the performance of graph to recipe generation task. Higher BLEU score is better.

single Type Prediction Network with the 28 entity types as explained before. This task was chosen as it allows us to demonstrate the Transfer Learning ability of our Language Model. Both networks were individually pretrained and have been applied for a different task with some adjustments. As evident from Table 5, our model clearly outperforms the baseline on our dataset. We could not incorporate (Salvador et al., 2019) and (Majumder et al., 2019) on our dataset because of lack of images and user reviews in our corpus, but have listed them for fair comparison. We attribute the lower perplexity by (Salvador et al., 2019) to the much larger corpus size and clues from images..

4.2 Action Graph to Recipe Step Generation

This experiment builds upon ingredient-to-text generation to explore the problem of data to text generation. We use the partial Action Graphs obtained during dataset preparation and parse them into XML format, similar to the source dependency graphs from SR11Deep generation task (Belz et al., 2011) and the RDF graphs from WebNLG Task. Each partial Action Graph has a recipe step associated with it. We now attempt to predict the correct recipe step given a partial Action Graph. Due to high training times even on a P100 GPU, we could only conduct this experiment using action graphs from 56,676 recipes. Unlike other experiments in the manuscript, action graphs of instructions from 51,280 recipes were used as training set, 2,698 each as validation and test sets.

The proposed model borrows the Graph Convolution Encoder from (Marcheggiani and Perez-Beltrachini, 2018), we use the same encoder as the one used in the SR11 task but to delexicalise the named entities in the input sentence instead of treating multi-word entities as being composed of separate nodes. During training and inference these are relexicalised to get the same embeddings as the pretrained on the corpus. The architecture is illustrated in Figure 3.

We apply the same training procedure by using pretrained networks on the dataset. Again, a soft attention is applied over the GCN induced representation and fed into the Type Prediction and Entity Transformation Networks. We use the same baseline as them, a seq-to-seq LSTM encoder decoder framework, which takes a linear version of the source graph as input. We also compare our model with their proposed GCN Encoder + LSTM Decoder model. Although our architecture outperforms both the baselines, as detailed in Table 6, it's only a marginal improvement when compared with the GCN Encoder. Nevertheless, it shows the use of these pretrained models in transfer learning on different tasks.

4.3 Recipe Generation

The language model was used to generate recipes given the first sentence of a genuine cooking recipe using beam search with a beam width of 15. We generated 107 such cooking recipes. We collected another 108 genuine recipes which were then tokenised and converted into strings of space separated tokens, such that they were indistinguishable from the unformatted generated recipes output by the language model. This set was then shuffled and distributed among 13 adults with varied cooking experience. They were told about the composition of the set, that it is evenly comprised of generated and genuine recipes, and were asked to predict which recipes were generated and which were genuine. Out of the 107 generated recipes, 60 were marked correctly as having been generated while 47 were incorrectly marked as genuine. Similarly of the 108 genuine recipes, 32 were marked incorrectly as having been generated while 76 were correctly marked as genuine. We can see that when the recipes are seen as a whole, it was possible to distinguish them from the fact 76 genuine recipes were marked as genuine but only 47 generated recipes were marked as genuine. However, this does not evaluate the model well as a single giveaway is sufficient to identify a fake recipe, as evident from the examples in appendix. We further asked the respondents to give a score (from 0 to 10) to the quality of generated text for all the recipes they believed were generated. They were asked to give at least of 4 minutes in making their decisions. Interestingly, genuine recipes identified as generated recipes received an average score of 6.46 but generated recipes were close behind with an average score of 6.43. This shows that the quality of text being generated is of approximately the same quality as that found on online sources.

Example of Cooking Recipe Generation using Beam Search

Preheat oven to 350 degrees F. In a large skillet, heat oil over medium heat heat . Add chicken and cook until brown on all sides, about 5 minutes per side. Transfer chicken to a baking dish and keep warm in oven oven. Heat oil in the same skillet over medium high heat. Add onion and cook until translucent, about 3 minutes. Add garlic and cook, stirring, until fragrant, about 1 minute. Add broth and bring to a boil. Add chicken, reduce heat to medium , and simmer until chicken is cooked through, about 10 minutes. Transfer chicken to a serving dish and cover with foil to keep warm. Bring to a boil, reduce to a simmer and cook until thickened, about 5 minutes. Season with salt and pepper to taste. Spoon sauce over chicken.

5 Conclusion

The Hierarchically Disentangled Language Models proposed in this manuscript clearly are an incremental improvement over (Parvez et al., 2018), across the three settings we evaluated upon.

First, we evaluated it intrinsically in terms of perplexity on two datasets on which it out-performs the primary baseline. We also show that the original dataset with lower number of distinct named entities is comparatively easier for both the models. Whereas, the original baseline performed poorly over the newly procured, more complex dataset. This highlights the importance of fine-tuning the subnetworks as well as simplifying the input to Entity Transformation network – since knowledge about the previous types is already present within the RNN, explicity including it is not only redundant, but also makes the modeling task more complex. Secondly, we used our Language Model as a decoder in an ingredient-to-recipe generator. Here it got lower perplexity than a baseline encoder-decoder model. Finally, we used the language model as the decoder in an action graph-to-recipe model. In this setting the generated recipes had higher BLEU compared with reference recipes than the same model using an LSTM decoder. The latter two experiments together provide the evidence to the utility of such language models by Transfer Learning, in more complex tasks.

An issue with the proposed language models is the annotation of named entities, which is domain specific and hard to automate. We also did not see much improvement by increasing levels of entity hierarchy, which only made the overall annotation tasks more complex. We encourage future contributions in further testing such Language Models in different text domains and improve upon our training strategy and architecture - whether specific to certain domains or for general purpose language modeling.

References

Kenneth Arnold, Kai-Wei Chang, and Adam Kalai. 2017. Counterfactual language model adaptation for suggesting phrases. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 49–54, Taipei, Taiwan, November. Asian Federation of Natural Language Processing.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Devansh Batra, Nirav Diwan, Utkarsh Upadhyay, Jushaan Singh Kalra, Tript Sharma, Aman Kumar Sharma, Dheeraj Khanna, Jaspreet Singh Marwah, Srilakshmi Kalathil, Navjot Singh, et al. 2019. Recipedb: A resource for exploring recipes. *Available at SSRN 3482237*.

Anja Belz, Mike White, Dominic Espinosa, Eric Kow, Deirdre Hogan, and Amanda Stent. 2011. The first surface realisation shared task: Overview and evaluation results. In *Proceedings of the 13th European Workshop on Natural Language Generation*, pages 217–226, Nancy, France, September. Association for Computational Linguistics.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2017. Quasi-Recurrent Neural Networks. *International Conference on Learning Representations (ICLR 2017)*.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, 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, Doha, Qatar, October. Association for Computational Linguistics.

Erol Cromwell, Jonah Galeota-Sprung, and Raghuram Ramanujan. 2015. Computational creativity in the culinary arts.

Robert Dale. 2019. Generating recipes: An overview of epicure. In Christopher Mellish Robert Dale and Michael Zock, editors, *Current Research in Natural Language Generation*. Academic Press, London.

Nirav Diwan, Devansh Batra, and Ganesh Bagler. 2020. A named entity based approach to model recipes. In *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*, pages 88–93. IEEE.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 363–370, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Neelansh Garg, Apuroop Sethupathy, Rudraksh Tuwani, Rakhi Nk, Shubham Dokania, Arvind Iyer, Ayushi Gupta, Shubhra Agrawal, Navjot Singh, Shubham Shukla, et al. 2018. Flavordb: a database of flavor molecules. *Nucleic acids research*, 46(D1):D1210–D1216.

Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. 2005. Bidirectional lstm networks for improved phoneme classification and recognition. In *Proceedings of the 15th International Conference on Artificial Neural Networks: Formal Models and Their Applications - Volume Part II*, ICANN'05, page 799–804, Berlin, Heidelberg. Springer-Verlag.

Zhijiang Guo, Yan Zhang, Zhiyang Teng, and Wei Lu. 2019. Densely connected graph convolutional networks for graph-to-sequence learning. *Transactions of the Association for Computational Linguistics*, 7:297–312, March.

Vincent J. Hellendoorn and Premkumar Devanbu. 2017. Are deep neural networks the best choice for modeling source code? In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, page 763–773, New York, NY, USA. Association for Computing Machinery.

Abram Hindle, Earl T. Barr, Mark Gabel, Zhendong Su, and Premkumar Devanbu. 2016. On the naturalness of software. *Commun. ACM*, 59(5):122–131, April.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Chloé Kiddon, Ganesa Thandavam Ponnuraj, Luke Zettlemoyer, and Yejin Choi. 2015. Mise en place: Unsupervised interpretation of instructional recipes. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 982–992, Lisbon, Portugal, September. Association for Computational Linguistics.

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 329–339, Austin, Texas, November. Association for Computational Linguistics.

Rik Koncel-Kedziorski, Dhanush Bekal, Yi Luan, Mirella Lapata, and Hannaneh Hajishirzi. 2019. Text Generation from Knowledge Graphs with Graph Transformers. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2284–2293, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Bodhisattwa Prasad Majumder, Shuyang Li, Jianmo Ni, and Julian McAuley. 2019. Generating personalized recipes from historical user preferences. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5976–5982, Hong Kong, China, November. Association for Computational Linguistics.

Diego Marcheggiani and Laura Perez-Beltrachini. 2018. Deep graph convolutional encoders for structured data to text generation. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 1–9, Tilburg University, The Netherlands, November. Association for Computational Linguistics.

Javier Marin, Aritro Biswas, Ferda Ofli, Nicholas Hynes, Amaia Salvador, Yusuf Aytar, Ingmar Weber, and Antonio Torralba. 2019. Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images. *IEEE Trans. Pattern Anal. Mach. Intell.*

Shinsuke Mori, Hirokuni Maeta, Yoko Yamakata, and Tetsuro Sasada. 2014. Flow graph corpus from recipe texts. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 2370–2377, Reykjavik, Iceland, May. European Language Resources Association (ELRA).

Md Rizwan Parvez, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2018. Building language models for text with named entities. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2373–2383, Melbourne, Australia, July. Association for Computational Linguistics.

Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 7–14, Florence, Italy, August. Association for Computational Linguistics.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China, November. Association for Computational Linguistics.

Ratish Puduppully, Li Dong, and Mirella Lapata. 2019. Data-to-text generation with entity modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2023–2035, Florence, Italy, July. Association for Computational Linguistics.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1139–1149, Vancouver, Canada, July. Association for Computational Linguistics.

A. Salvador, N. Hynes, Y. Aytar, J. Marin, F. Ofli, I. Weber, and A. Torralba. 2017. Learning cross-modal embeddings for cooking recipes and food images. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3068–3076.

Amaia Salvador, Michal Drozdzal, Xavier Giro-i Nieto, and Adriana Romero. 2019. Inverse cooking: Recipe generation from food images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June.

Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. 2017. A hybrid convolutional variational autoencoder for text generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 627–637, Copenhagen, Denmark, September. Association for Computational Linguistics.

C. Vondrick and A. Torralba. 2017. Generating the future with adversarial transformers. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2992–3000.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November. Association for Computational Linguistics.

Bolin Wei, Ge Li, Xin Xia, Zhiyi Fu, and Zhi Jin. 2019. Code generation as a dual task of code summarization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 6563–6573. Curran Associates, Inc.

Sam Wiseman, Stuart M. Shieber, and Alexander M. Rush. 2017. Challenges in data-to-document generation. *CoRR*, abs/1707.08052.

Kai Xu, Longyin Wen, Guorong Li, Liefeng Bo, and Qingming Huang. 2019. Spatiotemporal cnn for video object segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. *ArXiv*, abs/1704.01696.

B. Zhu, C. Ngo, J. Chen, and Y. Hao. 2019. Rgan: Cross-modal recipe retrieval with generative adversarial network. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11469–11478.