Chapter 4

Conditional Long Short-Term Memory Network

This chapter describes our text generating network. The goal of this network is to generate realistic yet novel customer reviews. However, instead of generating arbitrary customer reviews, we want to have control over the characteristics of the generated output. We enforce this by making the generation of a review conditional on a deliberately chosen cluster.

Recall from the previous stage, that we have clustered the reviews into meaningful groups. That is, by now, we have found groups of reviews that share linguistic characteristics. For example, one cluster describes reviews that are short and positive. Another cluster of reviews describes reviews that are longer, have a neutral sentiment, and tend to include references to family members. Example reviews from these clusters can be seen in Table 4.1 and Table 4.2.

Using our text generating network, we want to generate novel reviews conditional on a chosen cluster. This way we can generate reviews that are conform to our preferences, since we have control over the linguistic style. For example, we may want to generate reviews that are short and positive, similar to the ones depicted in 4.1. To achieve this, we pass the network information on the chosen cluster, such that it generates a review conditional on this information. We will describe in detail how this is achieved in the subsequent sections.

The network used for our task of generating text, namely a Long Short-Term Memory, is a variant of a Recurrent Neural Network. The subsequent section gives an introduction into these kind of networks.

Reviews
looks good .
works great .
looks great .
great option to get you started .
very high quality . i love it .
perfect love it
great product

Table 4.1: Example reviews of cluster 2. The reviews of this cluster tend to be longer and tend to have a neutral sentiment.

Reviews
bought this for my boyfriend for his birthday i do n t use it . he seems to really like it .
i ve been looking for replacement heads for my norelco xl which i bought years ago
i bought this as a gift for my husband years ago . he was never able to get it tuned .
i bought this stand for my yamaha psr keyboard . i thought it would fit but it does n t .
i bought two of these for my boys . one silver and one blue .

Table 4.2: Example reviews of a single cluster. The reviews of this cluster tend to be longer, have a neutral sentiment, and frequently include references to family members.

4.1 Recurrent Neural Networks

A Long Short-Term Memory is a variant of a Recurrent Neural Network. A Recurrent Neural Network (RNN) is a dynamic model that processes sequences of input one step at a time, learning to predict the next step. At each step, the model updates an internal state that stores information of previous inputs, such that each prediction is also based upon what the model has seen before. This characteristic makes these type of networks suitable for the task of generating sequences of data. Recurrent Neural Networks are used in a variety of domains, including image generation (Gregor et al., 2015), handwriting generation (Graves, 2013), text generation (Sutskever et al., 2011) and poetry generation (Yi et al., 2017).

When generating novel sequences with a trained RNN, one iteratively samples from the network's output distribution, then feeding in the sample as an input in the next step (Graves, 2013). Since the prediction at each step depends on previous inputs, the output distribution is conditional (Graves, 2013).

This allows us to build a conditional RNN, where we deliberately feed the network inputs of a certain kind, priming its internal state. After priming, we let the primed network continue generating a novel sequence. Since future outputs are produced based on what the network has seen before, the generated sequence will be conditional on the input we have shown it at the beginning. The idea of primed sampling

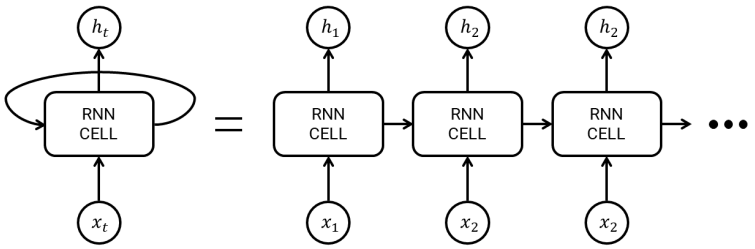


Figure 4.1: An illustration of the recurrent structure of a Recurrent Neural Network.

stems from work one a generative network that is able to generate handwriting conditional on a particular writer's style (Graves, 2013).

In theory, standard Recurrent Neural Networks are able to generate sequences of arbitrary complexity (Graves, 2013). However, researchers have shown that in practice, RNNs have difficulties to learn long-term dependencies (Hochreiter et al., 2001). Most RNNs are trained using gradient descent and backpropagation through time (Rumelhart et al., 1986), which suffers from vanishing or exploding gradient problem: the gradients computed tend to either vanish or explode during the backward pass (Hochreiter et al., 2001).

Long-Short-Term Memory (Hochreiter & Schmidhuber, 1997) networks are a variant of a Recurrent Neural Network designed to improve the gradient flow, therefore avoiding the vanishing gradient problem (Hochreiter et al., 2001). In addition, gradient clipping (Pascanu et al., 2013) can be used to tackle the exploding gradient problem to further stabilize the training of LSTMs (Trinh et al., 2018).

The subsequent section will describe LSTMs and how they work more formally.

4.2 Long Short-Term Memory Networks

For our use case we use a LSTM with a so-called *many-to-many* architecture. That is, when processing a sequence of length τ, at each time step t, where t ranges from 1 to τ , the network will receive an input $\vec{x}_t$ and also produce an output $\vec{y}_t$.

A LSTM has a recurrent structure, which allows the network to store information about previous inputs in a way such that a prediction $\vec{y}_t$ not only depends on the input $\vec{x}_t$, but also on information about previous inputs. Previous information is stored and passed on from time step to time step via a so-called hidden state $\vec{h}_t$ and cell state $\vec{c}_t$. To better understand the output of a LSTM, one may look at the mathematical formulation. A LSTM can be described via the following composite

function (Graves, 2013):

$$\vec{i}_t = \sigma(W_{xi}\vec{x}_t + W_{hi}\vec{h}_{t-1} + W_{ci}\vec{c}_{t-1} + \vec{b}_i) \tag{4.1}$$

$$\vec{f}_t = \sigma(W_{xf}\vec{x}_t + W_{hf}\vec{h}_{t-1} + W_{cf}\vec{c}_{t-1} + \vec{b}_f) \tag{4.2}$$

$$\vec{c}_t = \vec{f}_t \cdot \vec{c}_{t-1} + \vec{i}_t \tanh(W_{xc}\vec{x}_t + W_{hc}\vec{h}_{t-1} + \vec{b}_c) \tag{4.3}$$

$$\vec{o}_t = \sigma(W_{xo}\vec{x}_t + W_{ho}\vec{h}_{t-1} + W_{co}\vec{c}_t + \vec{b}_o) \tag{4.4}$$

$$\vec{h}_t = \vec{o}_t \tanh(\vec{c}_t) \tag{4.5}$$

where σ is the logistic sigmoid function, $\vec{i}_t$ is called *input gate*, $\vec{f}_t$ is called *forget gate*, $\vec{c}_t$ is called *cell state*, $\vec{o}_t$ is called *output gate* and h_t is called *hidden state*. All of $\vec{i}_t, \vec{f}_t, \vec{c}_t, \vec{o}_t$ are vectors of the same size as the hidden vector $\vec{h}_t$. Weight matrices are denoted with capital W, with subscripts used to clarify their usage: For example W_{xi} is the weight matrix that is multiplied with the input $\vec{x}_t$ when computing the input gate $\vec{i}_t$. Biases are denoted with the lowercase letter $\vec{b}$.

Looking at the equations, one notices that the output $\vec{h}_t$ of the LSTM cell depends on the input $\vec{x}_t$, the previous hidden state $\vec{h}_{t-1}$ and the previous cell state $\vec{c}_{t-1}$, which shows how information is passed on from one time step to another.

4.3 LSTM for language modeling

In the previous section we have described the inner workings of a LSTM more formally. In the following, we explain how we have set up the network to be able to conditionally generate text in form of customer reviews.

4.3.1 Word-Based Language Models

In the context of language modeling, text is commonly split up into a sequence of words, where each word is presented to the network as a one-hot input vector (Graves, 2013). That is, using a dictionary with a total of K words, we represent the k^{th} word, where $k \in \{1, \ldots, K\}$, as a vector of zeros except for a one at the k^{th} position.

There are other promising approaches where text is modeled as a sequence of characters (Sutskever et al., 2011; Mikolov et al., 2012). However, there are two main reasons as to why we have not followed such an approach. First, word-based models tend to yield a slightly better performance at the task of text generation (Graves, 2013). Second, we later wish to focus on evaluating the conditional aspect of our LSTM. To do this, we will (1) generate reviews conditional on a cluster, (2) embed each review into the SBERT vector space and (3) see whether the trained k-Means model is able to predict the correct cluster. The problem with this procedure has to do with the SBERT embedding: SBERT has problems with misspelled words that potentially impact the embedding (Sun et al., 2020). Noting that a character-based model, especially one that is not fully trained until convergence, will frequently

produce misspelled words, this fact would bias the embedding and hence bias our evaluation of the conditional LSTM. Hence, we have chosen a word-based model.

Using the word-based approach to language modeling, we can formulate the goal of the LSTM as follows: At each time step t, the goal is to predict the respective next word $\vec{x}_{t+1}$ of the sequence. An illustration is provided in Figure 4.2.

This problem can be viewed as a classification task with K possible classes (words). As is common for classification tasks, we included a softmax layer that is connected to the output layer of the LSTM. The softmax layer will receive the hidden vector $\vec{h}_t$ of size H as input, and produce an output vector $\hat{\vec{y}}_t$ of size K. Since the latter vector is put through a softmax function, the elements of $\hat{\vec{y}}_t$ correspond to multinomial distribution defined over the set of all K classes.

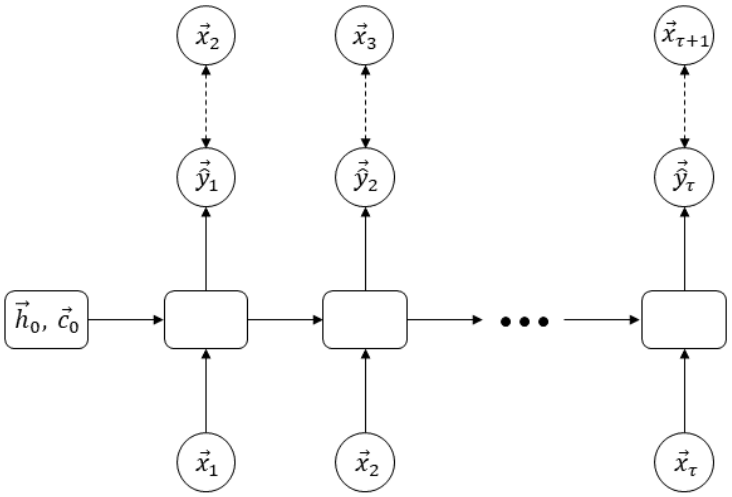


Figure 4.2: Illustration of a many-to-many architecture of a LSTM. At each time step t, the network will receive the hidden state $\vec{h}_{t-1}$ and cell state $\vec{c}_{t-1}$ from the previous time step as well as a input $\vec{x}_t$, and will output a prediction vector $\hat{\vec{y}}_t$. The prediction is then compared with the target, which corresponds to the next input $\vec{x}_{t+1}$.

At this point it is important to highlight that the number of classes K, or the number of words in the dictionary, is critical for the set up of the model. There is a problem when having a high number of classes (ie, words in a dictionary). Since words are represented as a one-hot vectors, these vectors have the same size as the dictionary. If K is very large, then the one-hot input vectors will be large and hence the weight matrices that are multiplied by those input vectors will also be large. For example, if there are 300 000 unique words in the corpus (which is easily the case with our data), the one-hot vectors would have 300 000 elements. If one further chooses the size of the hidden state $\vec{h}_t$ (and therefore also the size of $\vec{i}_t$) to be 100, which is common in word-based language models (Graves, 2013), then the weight matrix W_{xi} that is multiplied with the one-hot input $\vec{x}_t$ will be of size $100 \times 300,000$. In other words, the matrix W_{xi} would store 30 million weights. Training a network with such

a large number of weights is computationally expensive, as it would require a ton of training data. Therefore, we have followed the approach of (Graves, 2013) and limit the number of words in the dictionary to only include the most 10 000 most frequent words of our corpus. If a word appears in our corpus that is not included in the dictionary, it is mapped to a special Unknown token.

4.3.2 Input Sequences

In the following, we will describe how we have designed the input sequences of our LSTM in order to cope with the problem of generating reviews conditional on a cluster.

First, in our set up each input sequence $\mathbf{x} = (x_1, \ldots, x_\tau)$ corresponds to a whole review. For example, let "we love this product" be a review in our corpus. Then, splitting it up into individual words, we will get a sequence of length 4.

Second, we append a special End-Of-Review token to the end of each sequence. This way, the network learns to predict the end of a review, which is helpful when we generate coherent, novel reviews.

Third, we append a special Start-Of-Review token to the front of each sequence. This token includes the label of the cluster to which the review was assigned to in the previous stage. Later, when generating reviews conditional on a cluster, we will feed the network a Start-Of-Review token with a chosen cluster ID. The model will process this token, updating its hidden and cell state accordingly. If we then let the network continue generating a review, all generated words will be generated conditional on having seen the token. This way, we are able to generate a review conditional on a cluster. The text generated will share the linguistic characteristics of the particular cluster.

Putting all steps together, the example review "we love this product" will be presented to the LSTM as the input sequence

$$\mathbf{x} = (x_1, x_2, x_3, x_4, x_5) = (\text{<SOR Cluster-ID>, we, love, this, product}) \tag{4.6}$$

with corresponding targets

$$\mathbf{y} = (y_1, y_2, y_3, y_4, y_5) = (\text{we, love, this, product, <EOR>}) \tag{4.7}$$

4.3.3 Minibatches

There is one programmatic issue that arises due to choosing to model each review as a single input sequence. Namely, since most reviews are of different lengths, the LSTM needs to process sequences of variable length.

In theory, this is not a problem. The recurrent structure of LSTMs allows for processing sequences of arbitrary length (Graves, 2013).

In practice however, one wishes to train the model using minibatches, where each minibatch includes a number of sequences (here: reviews). The sequences of a minibatch are stored in a matrix with fixed dimensions. In order to store multiple reviews in a single matrix, we append special padding-tokens to all reviews that are shorter than the longest review in a given batch. After this preprocessing step, every review (sequence) will be of the same length as the longest review in the batch.

The padding-tokens are only used such that the data can conveniently be processed in minibatches. We do not want the LSTM to predict padding-tokens when we generate novel reviews. Therefore, padding-tokens are ignored when computing the loss.

4.3.4 Hyperparameters

There are two important hyperparameters when setting up a LSTM: The size H of the hidden vector $\vec{h}_t$ and the number of layers or depth.

Looking at comparable word-based models for text generation, we have found that well-performing values for H lie in the range between 90 and 700. For example, Mikolov, Karafiát, Burget, Černockỳ, and Khudanpur (2010) find that a word-based Recurrent Neural Network with $H = 90$ for a speech recognition task yields the best performance. Graves (2013) use a word-based LSTM with $H = 700$ for text generation of Wikipedia data. Kawthekar, Rewari, and Bhooshan (2017) compare a variety of text generating models, finding that within a group of word-based LSTMs, the LSTM with $H = 200$ performs best on validation data.

Surely, the size H of the hidden vector $\vec{h}_t$ depends on the particular use case as well as other modeling choices such as the size of the input vectors. However, looking at these studies we at least get a reference point.

For our case, we have trained models with $H = 100$ and $H = 500$. The latter model performed better at all evaluation metrics, which are presented in section 4.4.

Concerning the number of layers of our LSTMs, we have only used single-layer networks. The reasons for this are as follows. First, researchers have found that performance improvements of making a Recurrent Neural Net deeper than two layers are marginal (Reimers & Gurevych, 2017). Second, since our training data includes only rather short and independent reviews, such that our model doesn't have to process long and complex sequences, we think that a single layer network is enough to be able to capture most patterns in our data.

4.3.5 Learnable hidden state

Any time the network sees a new sequence during training, we want it to have a reset hidden state. That is, at the time where the network sees a new sequence, it should not hold any information of previous reviews in its internal state. Therefore,

both the hidden state and cell state are reset before each sequence.

There a few options one has with regards to setting the initial hidden and cell state. The standard way is to set the initial hidden and cell state to be zero-vectors (Graves et al., 2013). However, researchers have found that using nonzero initial values can boost the performance of the network (Haykin et al., 2007). In our case, we have decided to treat the initial hidden and cell state as parameters that are updated during training. In other words, our model learns a good initial hidden and cell state.

4.3.6 Loss

The network is trained using backpropagation through time. The loss function follows naturally from the fact that we modelled text as a sequence of words, where the goal at time step t is to predict the respective next word $\vec{x}_{t+1}$. Given a sequence $\mathbf{x} = (\vec{x}_1, \ldots, \vec{x}_\tau)$, the sequence loss used to train the model is the cross entropy loss, defined by:

$$L(\mathbf{x}) = -\sum_{t=1}^{\tau-1} \log\left(\frac{e^{p_{\vec{x}_{t+1}}}}{\sum_{k=1}^{K} e^{p_k}}\right)$$

where $p_{\vec{x}_{t+1}}$ is the predicted probability of the target word $\vec{x}_{t+1}$ at time step t. The sequence losses are averaged across all sequences in a given minibatch.

4.4 Evaluation

We have trained three models with different configurations. First, to compare models with a larger size H of the hidden vector $\vec{h}_t$, we have trained a model with $H = 100$ and one with $H = 500$. For both models, the initial hidden and cell state were learnable parameters. Second, to compare the effect of making the initial hidden and cell state learnable, we have trained another LSTM with $H = 100$ but without a learnable hidden state. For this model, hidden and cell state are reset to zero vectors before each sequence is processed.

It has taken approximately 20-30 epochs for the models to converge. Figure 4.3 plots training and test loss of the models with learnable hidden and cell state. Both training and test loss of the more model with $H = 500$ are lower than the respective loss of the model with $H = 100$.

However, looking at the test loss, one notices that from Epoch 12 onwards, the test loss for the more complex model with $H = 500$ rises, whereas at the same time the training loss of the same model decreases. This is an indication that the model has enough parameters to be able to start memorizing the training data. In other words, the more complex model tends to overfit after 12 Epochs.

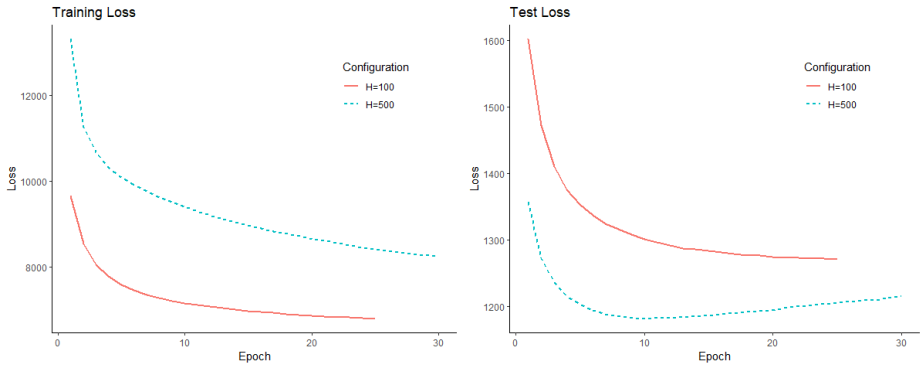


Figure 4.3: Training and test loss of the models with learnable hidden and cell state.

Analyzing the training and test loss is not really insightful with respect to the conditional aspect of the LSTM. To evaluate the conditional aspect, we have come up with two ways to get insights on how well our model generate reviews, conditional on a cluster.

The first way uses a metric that we call *Cluster Accuracy*. Given a LSTM, Cluster Accuracy is defined as the percentage of conditionally generated reviews whose cluster labels are correctly predicted when passed into our trained k-Means model.

We have computed the Cluster Accuracy after each epoch. After each epoch, we have generated 100 reviews for each cluster respectively, and have passed those to the trained k-Means model. Figure 4.4 shows how the Cluster Accuracies of our three models develop over epochs. First, one notices that after around 12 epochs, the Cluster Accuracies seem to converge or only improve marginally.

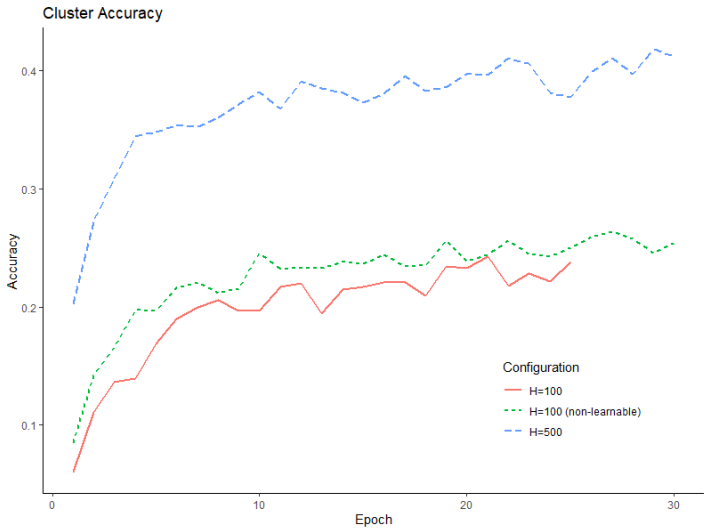


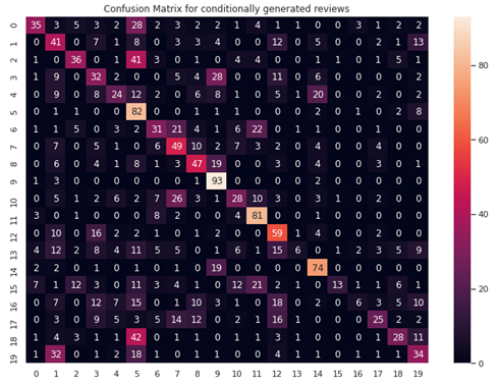
Figure 4.4: Cluster Accuracies of our trained models.

Further, it is notable that the more complex model with $H = 500$ performs significantly better than the other models. In fact, out of all conditionally generated

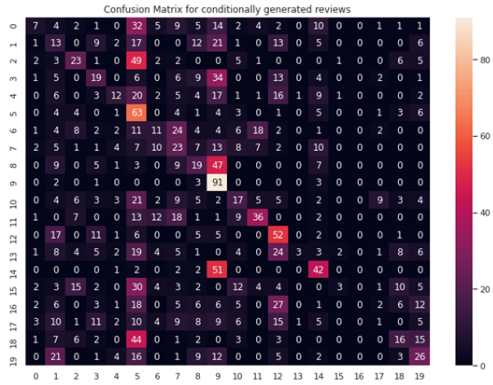
reviews, around 40 percent are correctly classified by the k-Means model. In contrast, the analogous model with $H = 100$ only achieves only a 25 percent Cluster Accuracy.

Comparing both models with $H = 100$, but where the on has a learnable hidden and cell state and the other one has not, one notices that actually the model without a learnable initial hidden and cell state performed marginally better at every epoch.

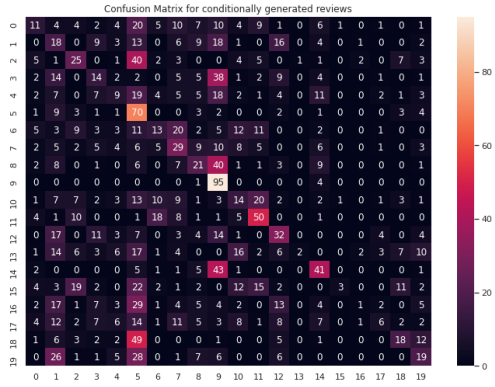
The second way we used to evaluate the conditional aspect is by looking at confusion matrices. Taking the 100 conditionally reviews that were generated for each cluster after every epoch, we can compute a confusion matrix that yields for each cluster the number of conditionally generated reviews that were assigned to the individual clusters. The diagonal of the confusion matrix corresponds to the number of correctly assigned reviews. Note that since we have generated 100 reviews for each cluster, the entries in the confusion matrix can be interpreted as probabilities. Figure 4.5 shows the results.



(a) LSTM with $H = 500$ and learnable hidden and cell state



(b) LSTM with $H = 100$ and learnable hidden and cell state



(c) LSTM with $H = 100$ and no learnable hidden and cell state

Figure 4.5: Confusion Matrices

Looking at the confusion matrices, one finds that there is one cluster, namely cluster 9, for which all models have a 90 percent Cluster Accuracy. That is, more than 90 percent of the conditionally generated reviews were correctly classified by the k-Means model. To find an explanation for this cluster, we have looked into the linguistic style of the actual reviews as well as the generated ones. We find that

this cluster has a very small within-cluster variance: Almost all reviews are very short and positive (e.g. "great product"). Surely, every one of our tested LSTM configurations is able to generate such short sequences. Further, since the short and positive review style is so characteristic for cluster 9, the k-Means will almost always assign the conditionally generated reviews to this cluster. An excerpt of the actual as well as generated reviews of cluster 9 can be found in Table 4.3

It is also interesting to look at clusters where the models do not perform too well. One example is cluster 2. In cluster 2, for example, the more complex LSTM with $H = 500$ achieves only a 36 percent Cluster Accuracy. Table 4.4 depicts the actual reviews as well as the generated reviews from all models. Looking at the actual reviews, we notice a larger within-cluster variance. That is, there is not a clearly identifiable, single linguistic style of the reviews within the cluster. However, the reviews of this cluster seem to be longer (mostly 2 sentences), have a rather neutral sentiment and frequently include mentions of family members. The lower Cluster Accuracy can be explained by two things. First, since the reviews generated are longer, there is more room for making a mistake when sampling a word from the output distribution. If during sampling a false prediction is made early in the sequence, the subsequent predictions will also more likely be wrong. The LSTM has a very small chance to recover from previous mistakes (Graves, 2013). Second, the reviews within cluster 2 have a larger within-cluster variance. That is, the reviews in that cluster do not share a single dominant linguistic style. This makes it difficult for the k-Means model to correctly label the generated reviews: Other clusters' reviews look not too different.

Actual Reviews
"great quality product ."
"beautiful and as expected"
"looks good"
"works great"
"perfect love it"
"great product"

(a) Actual Reviews

Generated Reviews
"great product"
"love this stuff <UNK>"
"good product"
"i really love this product <UNK>"
"love the smell . "
"works great"

(b) LSTM with $H = 500$; learnable $\vec{h}_0, \vec{c}_0$

Generated Reviews
"great product and it is great"
"great product"
"love this product . "
"love this stuff"
"good product"

(c) LSTM with $H = 100$; learnable $\vec{h}_0, \vec{c}_0$

Generated Reviews
"great product"
"great product ."
"good product"
"great deal <UNK>"
"good quality"

(d) LSTM with $H = 100$; non-learnable $\vec{h}_0, \vec{c}_0$

Table 4.3: Actual and conditionally generated reviews of cluster 9.

Actual Reviews
"i bought two of these for my boys . one silver and one blue."
"i never used it yet . but my husband tried it out <UNK>"
"i never thought i d find a replacement head my razor is years old . works great ."
"my son loved it . spent hours learning instead of playing xbox ."
"i bought this software for my year old grandson . i m sorry i did ."

(a) Actual Reviews

Generated Reviews
"i bought this for my husband for christmas . he was using a different brand before he got it ."
"this product is a good product . i have been buying this for years ."
"this product has been a very nice for my year old son . he s used to the <UNK> for years ."
"my husband uses this every night and he loves to use it ."
"i bought this product for my daughter . it does n t seem to last long . "
"i have a very thick beard . this has been a long time to purchase ."

(b) LSTM with $H = 500$; learnable $\vec{h}_0, \vec{c}_0$

Generated Reviews
"i bought this product as my wife uses it . i am very happy with it ."
"i ve used this product for many years . i have tried many different brands ."
"i ve been using this shampoo for years . i had been using for years . "
"love this stuff"
"i bought this for my husband . it was a good purchase ."
"i ve used the product for years for the first time . the product was not as good as the original ."

(c) LSTM with $H = 100$; learnable $\vec{h}_0, \vec{c}_0$

Generated Reviews
"the best product i ve tried . it has a lot of different <UNK> but not a great price <UNK>"
"i have a few years and i am very impressed . i have tried this brand ."
"i have used this stuff for years . it is the best <UNK> . "
"i have used this for about years and the price is not a little more than a blade . it works well for me ."
"i was using it for a year now . it s a good product for me to get a lot of money ."

(d) LSTM with $H = 100$; non-learnable $\vec{h}_0, \vec{c}_0$

Table 4.4: Actual and conditionally generated reviews of cluster 2.