UNIVERSITY OF CALIFORNIA

Los Angeles

Merging Meaning for Product Attribute Extraction

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Applied Statistics

by

Nicholas Carbone

2022

ABSTRACT OF THE THESIS

Merging Meaning for Product Attribute Extraction

by

Nicholas Carbone

Master of Applied Statistics
University of California, Los Angeles, 2022
Professor Yian Nian Wu, Chair

Companies that can leverage their product descriptions to find meaningful insights have a competitive advantage in reaching and understanding their consumer. In practice, however, this is a challenging task. Product descriptions may be written by a number of different individuals with inconsistent formatting. They may contain typos, inaccurate representations and complex terminology so unique to a single product as to be meaningless to analyze alongside other products.

For this reason, the study of Product Attribute Extraction (PAE) aims to extract meaningful attributes from product descriptions with the goal of producing a structured dataset of attribute-value pairs. Prior work has focused largely on conventional natural language processing (NLP) techniques to identify such relationships. In this paper, we combine the architecture of transformers and variational autoencoders (VAE) to merge and format semantically similar product descriptions with the goal of decreasing the complexity of PAE. We demonstrate examples of the feasibility of this approach and assess its potential, shortcomings and application to the field.

The thesis of Nicholas Carbone is approved.

Frederic R. Paik Schoenberg

Tao Gao

Yian Nian Wu, Committee Chair

Unversity of California, Los Angeles

2022

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Product attribute extraction (PAE) aims to improve the quality of a business's understanding of its own products. Often times, product descriptions are handled by marketers, merchandisers and product managers who possess detailed knowledge and insight on the products, but might not have the technical ability to organize their descriptions into a unified style. Admittedly, this would be challenging for any one individual as descriptions, especially in an online marketplace, may come from multiple companies and within those companies there may be multiple people on different teams writing descriptions.

This leads to a scenario where, while individual people at a company may understand a product, a company is unable to analyze its products with maximum granularity. Consider a company that sells t-shirts with a number of different fabric blends. Its customer base may decide subconsciously that their favorite blend is a 90/10 mix of cotton and polyester. The business sees 2 of its t-shirts with this blend selling very well. The descriptions contain phrases like "90% cotton / 10% polyester" and "ninety percent cotton with ten percent polyester fabric blend". While we can read these and understand them to mean the same thing, there's no direct mapping between them to associate them with each other. Thus, the business doesn't learn a key quality about its customers and cannot capitalize by selling products tuned to their preferences.

In this paper, we seek to provide a consistent framework to extract attributes from products. Our goal is not to directly extract the attributes, but to develop a system to take much guesswork out of parsing sentences to identify attributes. To achieve this end, we combine transformer models with variational autoencoders to provide a smooth distribution over contextual embeddings. Our findings show that it is possible to recreate quality embeddings with a VAE by making use of a few different contributions to each architecture. Our final output condenses similar product descriptions into a single representation and then reconstructs the output. With this singular output, businesses would be able to format their product descriptions in kind and the uniformity should open the door for more simple parsing strategies to unlock the value contained in the attributes' information.

We begin by reviewing research within the field, then describing our dataset and methodology and finally discussing the results.

2 Background

2.1 Product Attribute Extraction

While text mining has been a longstanding area of research, product attribute extraction has only more recently been given serious attention. This is due in part to an explosion of e-commerce, accounting for 14.3% of total US retail sales in the first quarter of 2022, which is more than double the sector's share in 2013.[1] As demand shifts online, supply naturally shifts in kind and the result is an enormity of products, each with uniquely crafted descriptions. It is thus of increasing importance for businesses to get an adequate handle on consistently managing the information regarding all these new products.

One of the earliest attempts to extract attributes from products was provided by Ghani et al.[1] Ghani et al. used the MiniPAR dependency parser[2] to identify attribute-value relationships and the Expectation-Maximization algorithm to extend their labelled training set. Putthividhya and Hu frame the problem through the named entity recognition lens and demonstrate the merits of a conditional random field model.[3] Petrovski and Bizer develop rule-based parsing strategies for descriptions contained in HTML tables to extract attributes.[4] Zheng et al. builds upon the bidirectional Long Short-Term Memory-Conditional Random Field (LSTM-CRF) sequence tagging model by adding an attention mechanism and using bootstrapped labeling to overcome the dependence on predefined attribute lists. [5] More recently, Lin et al. proposed using generative models to extract attributes from product images.[6]

The majority of these models are aimed at directly extracting attributes from product descriptions. However, each of the aforementioned papers notes the assumptions and limitations of their preceding authors' methods. For the ever-growing list of products, it's incredibly challenging to come up with a one-size-fits-all method to extracting attributes. In mathematics, when a problem is exceedingly difficult, a common approach is to try and recast it into an alternate, typically equivalent, form. For example, when it was found that the Support-Vecotr Machine optimization objective was too computationally complex, the kernel trick came about to recast the dot product into the inner product in a higher-dimensional feature space. Our work seeks to provide a similar easing of the problem of product attribute extraction by dealing with one particular challenge: removing noise in semantically similar product descriptions. It is our hope that by developing a framework for this, both past and future methods can enjoy increased accuracy and decreased complexity by trivializing a number of edge cases PAE models are expected to handle.

[1] (3 p.) Published: 4th Quarter 2021, Source: U S Department of Commerce, Record Number: 2022 ASI 2322-6.61529

2.2 Transformers

2.2.1 General Architecture

Common in earlier Recurrent Neural Network (RNN) encoder-decoder models, the standalone usage of an attention mechanism which forms the basis of the transformer model was popularized by Vaswani et al.[7] Transformers offer much more natural parallelism which allow them to tackle much larger and more complex problems in natural language processing.

The main component of any modern transformer model consists of self-attention blocks. Each of these blocks contains the following

$$\begin{aligned} W_Q &\quad \text{(query weight matrix - n x k)} \\ W_K &\quad \text{(key weight matrix - n x k)} \\ W_V &\quad \text{(value weight matrix - n x k)} \\ f &\quad \text{(feed-forward neural network)} \end{aligned}$$

An embedding matrix, for m words with dimension n, is an m x n matrix used to associate each word with a dense vector of real values. Each row can be thought of as the dense representation of a particular word. We use this matrix as a look-up table to embed words in an n-dimensional space.

Now suppose we have a d-length sequence that's been embedded into dimension n, thus forming a d x n input, x. We then produce

$$\begin{aligned} Q &= xW_Q \quad \text{(query matrix - d x k)} \\ K &= xW_K \quad \text{(key matrix - d x k)} \\ V &= xW_V \quad \text{(value matrix - d x k)} \end{aligned}$$

At timestep t of the input sequence, we then generate

$$\begin{aligned} s_t &= KQ_t^\top \quad \text{(score - d x 1)} \\ z_t &= s_t^\top V \quad \text{(attention output - 1 x k)} \end{aligned}$$

We return a d x k matrix representing the contexts at each timestep in the input sequence. We then apply the softmax function and pass this through a feed-forward neural network, completing the block.

2.2.2 BERT

Arguably the most well-known example of a transformer model is BERT.[8] While other transformer models had already been proposed such as GPT[9], BERT demonstrated that by pre-training a transformer on a task involving attending to multiple directions of an input sequence, the model could come to a more complete contextual understanding of the data. The task in particular was masked language modeling, where the model would have to predict a missing token among an input sequence. Borrowing its architecture from Vaswani et al., BERT has now become a seminal example of the pre-trained transformer model with many future models building upon it.

2.2.3 BART

BART[10] is one of the models to expand upon BERT and GPT. BART is an encoder-decoder architecture that remains the same as the Vaswani et al. implementation, except that it replaces the ReLU activations at the output of the decoder with GeLU. BART differs in that it is trained by taking intentionally corrupted input sequences and attempting to reconstruct the original uncorrupted documents. It is this denoising property that provides particular interest for the field of PAE, where inputs can be incredibly noisy with significant deviations in the structure of two descriptions with the same meaning.

2.2.4 Sentence-BERT

Another offshoot of BERT came in the form of Sentence-BERT (SBERT). [11] While BERT has been shown to produce high-quality token embeddings, it was not designed with full sentence embedings in mind. In semantic similarity conducted with BERT, a cross-encoder would be typically used. With a cross-encoder, both sentences are passed to the network simultaneously and an output score between 0 and 1 is given indicating their similarity. While this can produce quality scores, its computational complexity is $O(n^2)$ and thus inefficient for very large collections of sentences. SBERT solved the efficiency problem by adding a pooling operation to BERT's output. This enabled SBERT to produce fixed-size embeddings based on BERT's outputs which could be computed independently and then compared using metrics such as dot product scores or cosine similarity.

A typical algorithm for finding communities among these embeddings is as follows [2]

[2]Notably, this algorithm will not cluster every individual product detail. Product descriptions with low cosine similarity to any others would be discarded. We would argue that product details so unique as to fall into this category are of lesser relevance to business needs as they offer minimal information content that can be compared across products.

Algorithm 1: A community detection algorithm using cosine similarity

```
let E be an mxn matrix of embeddings
let C be an nxn matrix of zeros
let L be an empty list of communities
let l' be an empty list
initialize a threshold γ
for i = 1, ..., n do
    for j ≠ i do
        compute c' = (e_i · e_j) / (||e_i|| ||e_j||)
        set c_ij = c'
for i = 1, .., n do
    for j ≠ i do
        if c_ij < γ then
            next
        else
            append j to l'
            set c_.j = 0
    append l' to L
    set l' to an empty list
```

2.3 Variational Autoencoders

2.3.1 General Architecture

We'll analyze the VAE architecture through its probabilistic formulation. Suppose we have our input, x, and our latent variables, z, and define their joint probability distribution $p(x, z)$. We assume a distribution for our latent variables z and thus the prior $p(z)$ is known. We can represent the joint probability as such

$$p(x, z) = p(x|z)p(z)$$

In generation (or decoding), we can sample from z and then sample x given z. Now consider the following formulation of the posterior

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

The posterior provides the requisite mapping (or encoding) of x into z for our data. The denominator $p(x)$ is the marginal distribution of x, known as the "evidence", which we could calculate like so

$$p(x) = \int p(x|z)p(z)dz$$

However, this integral is intractable. Instead, we'd like to approximate the posterior over a family of distributions $q_\lambda(z|x)$ where λ_{x_i} indicates the parameters of the distribution for each x_i. For instance, if

the family were normal, $\lambda_{x_i} := (\mu_{x_i}, \sigma_{x_i})$.

In order to determine how well our approximation of the posterior is, we can use the Kullback-Leibler (KL) divergence, which we define as

$$\begin{aligned}
\mathbb{KL}(q_\lambda(z|x)||p(z|x)) &= \mathbb{E}_q\left[\log \frac{q_\lambda(z|x)}{\frac{p(x|z)p(z)}{p(x)}}\right] = \mathbb{E}_q\left[\log \frac{q_\lambda(z|x)}{\frac{p(x,z)}{p(x)}}\right] \\
&= \mathbb{E}_q\left[\log q_\lambda(z|x)\right] - \mathbb{E}_q\left[\log p(x,z)\right] + \mathbb{E}_q\left[\log p(x)\right] \\
&= \mathbb{E}_q\left[\log q_\lambda(z|x)\right] - \mathbb{E}_q\left[\log p(x,z)\right] + \log p(x)
\end{aligned}$$

The KL divergence is a distance measure of the difference between two probability distributions. Unfortunately, though, we're still stuck with the the evidence in the equation. Let's consider the evidence lower bound (ELBO)

$$ELBO(\lambda) = \mathbb{E}_q\left[\log p(x,z)\right] - \mathbb{E}_q\left[\log q_\lambda(z|x)\right]$$

With this, we can rearrange our equation to be

$$\log p(x) = ELBO(\lambda) + \mathbb{KL}(q_\lambda(z|x)||p(z|x))$$

By Jensen's Inequality, the KL divergence is greater than or equal to 0. Since $p(x)$ is fixed, this must mean that the KL divergence is minimized when $ELBO(\lambda)$ is maximized, and thus our optimization problem should seek to maximize $ELBO(\lambda)$, a tractable function.

We can now reformulate our problem in terms of a neural network with the following equations

$$\begin{aligned}
ELBO(\theta,\phi) &= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x,z)\right] - \mathbb{E}_{q_\phi(z|x)}\left[\log q_\phi(z|x)\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z) + \log p_\theta(z)\right] - \mathbb{E}_{q_\phi(z|x)}\left[\log q_\phi(z|x)\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] + \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(z)\right] - \mathbb{E}_{q_\phi(z|x)}\left[\log q_\phi(z|x)\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - \mathbb{E}_{q_\phi(z|x)}\left[\log q_\phi(z|x) - \log p_\theta(z)\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\theta(z)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - \mathbb{KL}(q_\phi(z|x)||p_\theta(z))
\end{aligned}$$

where ϕ and θ represent the parameters of our encoder and decoder, respectively.

The last component needed is the reparameterization trick. In backpropogation, we need to compute the

gradients with respect to ϕ of a function of our sample from $q_\phi(z|x)$. Instead of dealing with with the probability distribution directly, we can sample z according to

$$z = \mu_x + \sigma_x \odot \epsilon = g_\theta(x, \epsilon)$$

where $\epsilon \sim \mathcal{N}(0, 1)$. Now let $\log p_\theta(x|z) = f(z)$. Then

$$\begin{aligned}\mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] &= \mathbb{E}_{q_\phi(z|x)}\left[f(z)\right] \\ &= \mathbb{E}_{p(\epsilon)}\left[f(g_\theta(x, \epsilon))\right]\end{aligned}$$

The Monte-Carlo estimate, with L samples, for this expectation becomes

$$\mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] \approx \frac{1}{L}\sum_{l=1}^{L} f(g_\theta(x, \epsilon_l))$$

And we can then take the gradient with respect to θ for this function, where our desired parameters are deterministic.

2.3.2 InfoVAE

InfoVAE makes two main improvements on the vanilla VAE implementation.[12] The authors note that for a significantly flexible class of functions $q_\phi : x_i \to N(\mu_i, \sigma_i)$, an optimal solution exists where $\mu_i \to \infty$ and $\sigma_i \to 0^+$. It's hypothesized that the KL divergence term should regulate this behavior; however, in practice this has been shown to not always be the case. The authors thus propose adding coefficients λ and α to the loss function as well as a Maximum-Mean Discrepancy (MMD) term $D(q_\phi(z)||p(z))$ where, letting z be a vector of independent standard normal random variates and z' the vector of samples drawn from the outputs of the encoder distribution

$$D(q_\phi(z)||p(z)) = \mathbb{E}[e^{-\epsilon||z'-z'||} + e^{-\epsilon||z-z||} - 2e^{-\epsilon||z-z'||}] \tag{1}$$

The loss function becomes

$$\mathcal{L}_{\text{InfoVAE}} = \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - (1-\alpha)\mathbb{KL}(q_\phi(z|x)||p_\theta(z)) - (\alpha + \lambda - 1)D(q_\phi(z)||p(z)) \tag{2}$$

2.3.3 Annealing

In another attempt to stabilize VAE training, Fu et al. propose a cyclical annealing schedule, whereby a coefficient applied to the KL divergence term would be gradually increased throughout training.[13] In practice, it was found that often times in training VAEs, the KL divergence would vanish early on in training as the model would lean too heavily on minimizing the encoder's distance from a standard normal distribution. This resulted in poor quality reconstructions. To rectify this, we set the KL divergence coefficient to 0 at the beginning of training and gradually increase it to 1. The goal is for the model to explore the latent space early on, generating a diverse set of outputs before the KL penalty forces the model to impose a more defined structure in its latent space. In the InfoVAE implementation, this implies setting the coefficient α to 1 and gradually decreasing it to 0.

3 Dataset

3.1 Overview

Our dataset was borrowed from research performed by Ni et al.[14] The dataset contains the information for over 2.6 million clothing products scraped from Amazon. Collectively, the products total over 15.5 million product details, which are the primary focus of our analysis. We provide some common text mining statistics on the dataset in Figures 1 and 2. We also depict the network structure that arises by applying spaCy's dependency parsing to a community identified by Algorithm 1 in Figure 3.

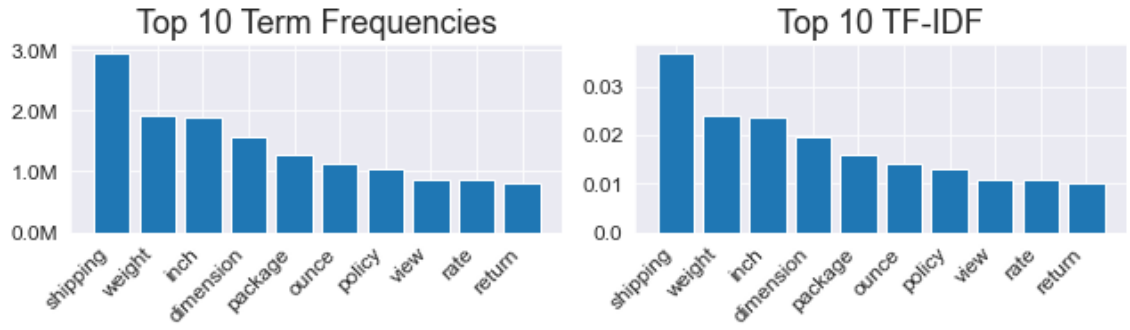

Figure 1: Top 10 term frequencies and TF-IDF for the dataset

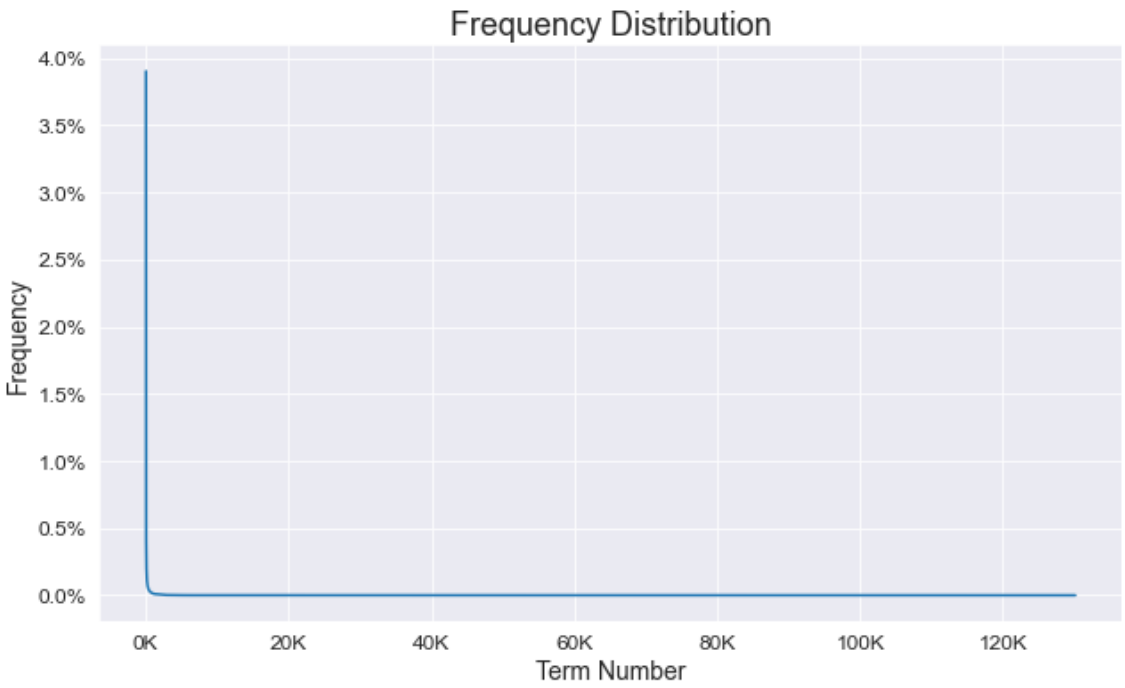

Figure 2: Frequency distribution of the dataset

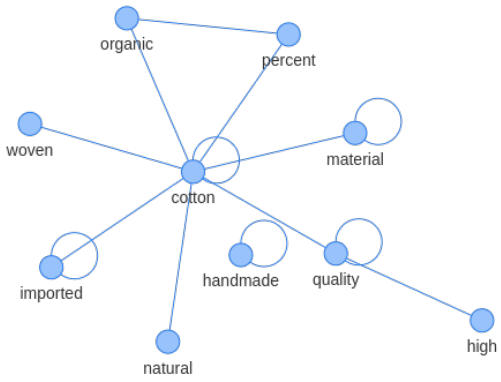

Figure 3: An example network structure from a community

3.2 Preprocessing

We try to minimize any preprocessing of the product details to keep the model as flexible as possible. This keeps in line with most of the research on transformer models where handling of arbitrary text is preferable. Our only transformation is to truncate the maximum token length to a size of 23.[3] This is helpful in training the model on a single GPU, allowing us to take larger batch sizes and speed up training. Sequences are also padded to 23 to improve I/O efficiency as well as to provide consistent comparisons in the latent space.

4 Methodology

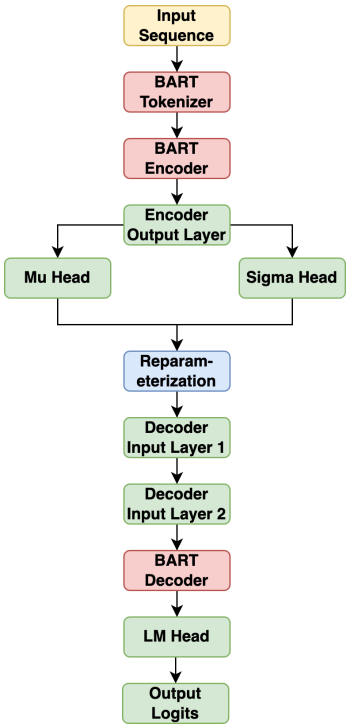

Figure 4: The full VAE model

4.1 Training

Our training methodology can be broken out into three distinct phases:

1. Fine-tuning a fully-connected layer on top of BART to reconstruct the product details
2. Using the pre-trained fully-connected layer, training a VAE model in between BART's encoder and decoder

[3] 23 is the 90^{th} percentile unadjusted length of tokens in our dataset.

3. Fine-tuning another BART model to reconstruct the original product details from the outputs of the VAE model

Training was performed on a single RTX2070 GPU. Total training time was approximately 144 hours, or 6 days. Details on training steps and hyperparameters used can be found in the appendix in 7.

Phase 1 We begin by training a language modeling (LM) head on top of BART. In this stage, we freeze BART's weights to utilize the inherent power of the pre-trained model. The model is trained with a batch size of 64 and gradients are accumulated every 4 steps, for an effective batch size of 256. The model is optimized over 392,268 steps, covering over 100 million non-unique training samples. We use the Adam optimizer with a learning rate of $1x10^{-4}$ throughout training.

Phase 2 In the next step, we begin training our core VAE model. Here, we insert a VAE model between BART's encoder and decoder as in Figure 4. We chose to use 256 as the size of the latent dimension. The last layer of the model is taken from the pre-trained LM head we produced in step 1. For this part of training, we keep the weights frozen for the LM head. Again, we freeze BART's weights and allow our VAE model to train on its own.[4] We train with a batch size of 64, accumulating gradients every 2 steps for an effective batch size of 128. The model is optimized over 821,110 steps, covering over 100 million non-unique training samples. We use the Adam optimizer with a learning rate of $1x10^{-4}$ throughout training.

We use the loss function as defined by InfoVAE and allow the hyperparameters for ϵ and α to vary throughout training as in equations 1 and 2, respectively. Our key finding in producing quality outputs is the need to begin VAE training for this problem by setting alpha equal to 1 in the beginning of training.[5] We believe this encourages the model to explore a more diverse manifold of the latent space prior to requiring it to maintain a more well-defined structure. Our primary goal as we decrease the value of α is to maintain a consistent value for our reconstruction loss. If α changes too quickly, the model will resort to minimizing the KL divergence and lose quality in its outputs. We don't impose a specific schedule on any hyperparameter changes, instead changing parameters by observing points in which the model settles into convergence.

Full model layer sizes and hyperparameters can be found in Tables 6 and 7, respectively, in the appendix.

[4] We initially attempted to fine-tune BART's weights and train the VAE simulataneously, but found that BART diverged too quickly from its precomputed weights to make training stable.

[5] Without this restriction, the model always became stuck in a local minimum by minimizing the KL divergence too quickly.

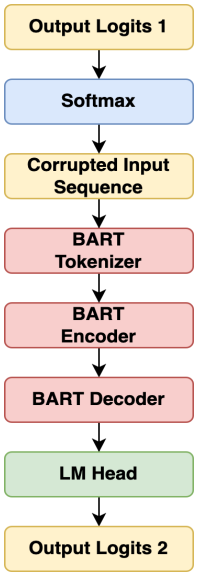

Figure 5: The BART model trained to reconstruct outputs from the VAE model

Phase 3 The final phase is to train a new BART model to reconstruct the corrupted sequences from the VAE model as is depicted in Figure 5. Generally, the VAE model recreates most input sequences with reasonable accuracy. However, we hypothesized that due to BART's specific design to work with corrupted input, this would provide us with greater accuracy given we can't guarantee quality output from the VAE model. Here, we allow BART's weights to be fine-tuned at the same time as the LM head sitting upstream. We train with a batch size of 64, accumulating gradients every 2 steps for an effective batch size of 128. The model is optimized over 393,105 steps, covering over 50 million non-unique training samples. We use the Adam optimizer and a learning rate of $1x10^{-4}$ for the LM head and $1x10^{-6}$ for the BART model throughout training.

4.2 Inference

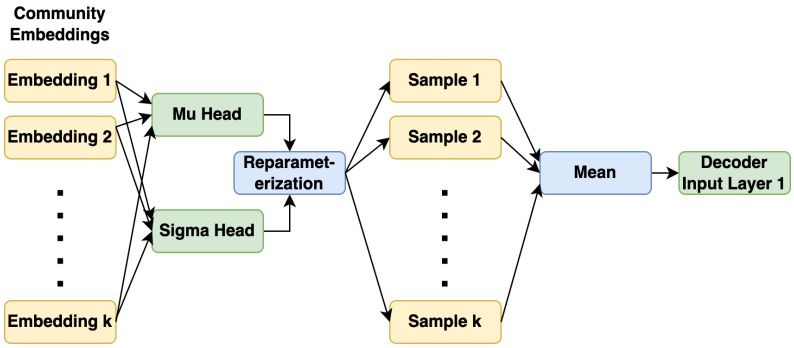

Figure 6: The averaging of samples to combine a community into a single representation for inference

Our inference methodology can be broken out into two distinct phases:

1. Use sentence transformers to embed product details and cluster them using a community detection algorithm

2. Merge the intermediate model outputs for an entire cluster to produce a single representation for that cluster

Phase 1 In this phase, we seek to cluster product details based on their semantic similarity. In order to do this, we must first embed our sentences. We use MPNet[15] as the encoder producing embeddings for each of our product details. We then identify communities among these embeddings according to Alogorithm 1. Since our dataset is approximately 15 million records, computing and comparing the cosine similarity for every embedding requires a significant amount of memory and time. Due to this computational limitation, we choose to take random samples of 50,000 product details at a time and form communities from these. We experimented with multiple values for γ and found that $\gamma = 0.95$ generally provided the best balance between the diversity and the consistency within the cluster structure to work with the model.

Phase 2 Once we have our communities, we now need to find a way to generate a single representation from these. Our point in implementing the VAE was to model the distribution of samples, so our hypothesis is that the VAE's outputs should be robust to noise within our communities, treating it similarly to the stochastic nature of its distribution. For this reason, we choose to average the VAE's outputs as shown in Figure 6. We then pass this singular input through the rest of our VAE model, reconstruct a sentence from its output and pass these through the secondary BART model to achieve a final output.

5 Results

5.1 Training

5.1.1 Phase 1

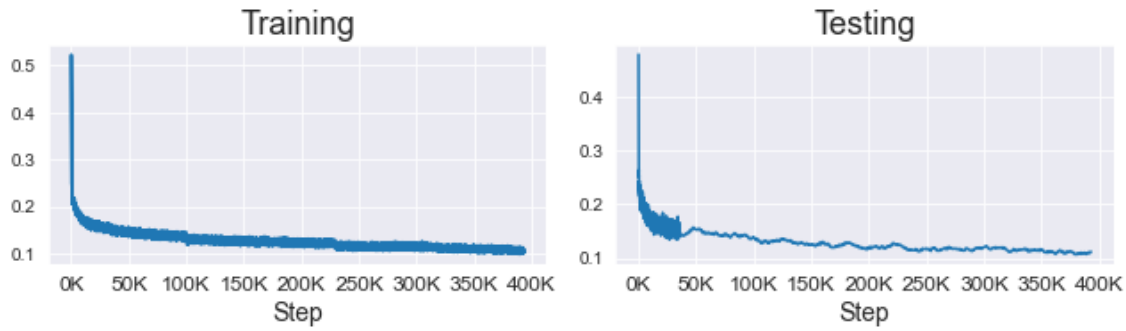

Figure 7: Loss curves for phase 1 (smoothed over 25 steps)

From Figure 7 we can see that the training for our language modeling head is relatively stable. We would expect this as this is a routine fine-tuning task for a transformer model.

With each phase of our training we also want to monitor the amount of entropy in the distribution our model creates over the input sequences. As the loss decreases, we expect the model to become more confident in its predictions, thus lowering the amount of entropy. We confirm this by viewing Figure 8.

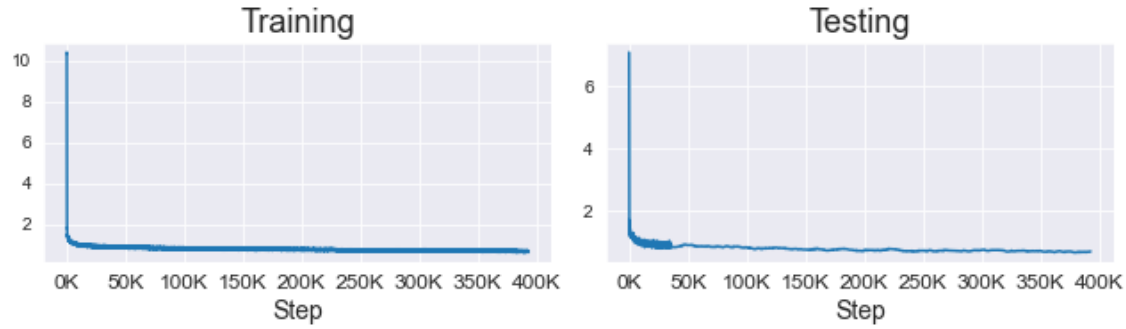

Figure 8: Entropy in output probabilities for phase 1 (smoothed over 25 steps)

And as we inspect some of reconstructions throughout training, we can generally see an improved reconstructed sentence that makes logical sense, as shown in Table 1. The reconstructions are not perfect, but we assess them to be reasonable enough to proceed with phase 2.

Start	Middle	End
Climate oriaddenribe...	Package Dimensions: ...	100% Cairoic
Climate requested...	Shipping Weight: 1 p...	Fpageine reasons pie...
Climate contemplatio...	Package Dimensions: ...	Heel measures approx...
Climate oriaddenribe...	10% Polyester/10% in...	12 wide
Climate requestedadd...	Shipping Information...	Product Dimensions: ...
Climate portablecart...	Vs size for is appro...	Package Dimensions: ...
Climateadden tried s...	idis USic optionally...	Pure tryingDig Women...
Climatepx landlords...	Shipping Weight: 11....	Dable and white cons...
requested alumni952...	Large The- the	Looklight and unique...
Climate oriaddenribe...	We% Fabricel, 5% Spa...	100% guitarucing

Table 1: Sample reconstructions at different points in the training process for phase 1

5.1.2 Phase 2

In phase 2, we have a number of new metrics that we need to track.

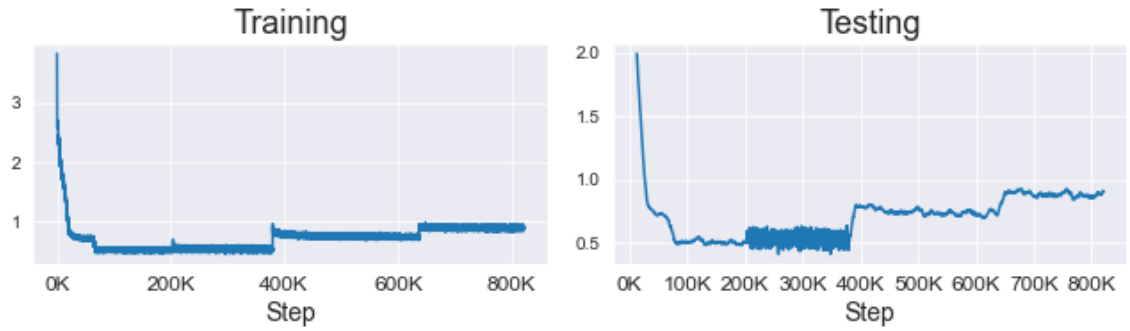

Figure 9: Loss curves for phase 2 (smoothed over 25 steps)

Beginning with the overall loss in Figure 9, we see a generally smooth decline early on in training. There are some jumps in the graph, but we note these as points in training where hyperparameters were changed, thus altering the weighting of different loss components in the overall loss. The important takeaway is that the loss does not continue to increase at these points in time.

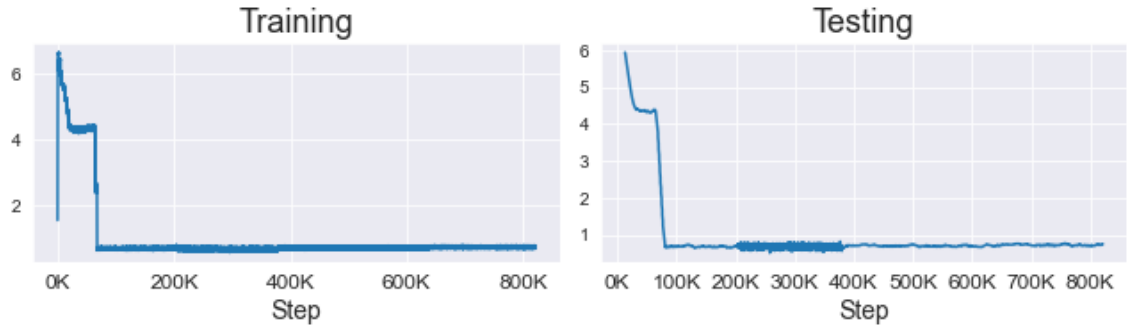

Figure 10: Entropy in output probabilities for phase 2 (smoothed over 25 steps)

Our entropy in Figure 10 remains at a consistently low level throughout training, which indicates that the changing of hyperparameters did little to impact our model's predictions.

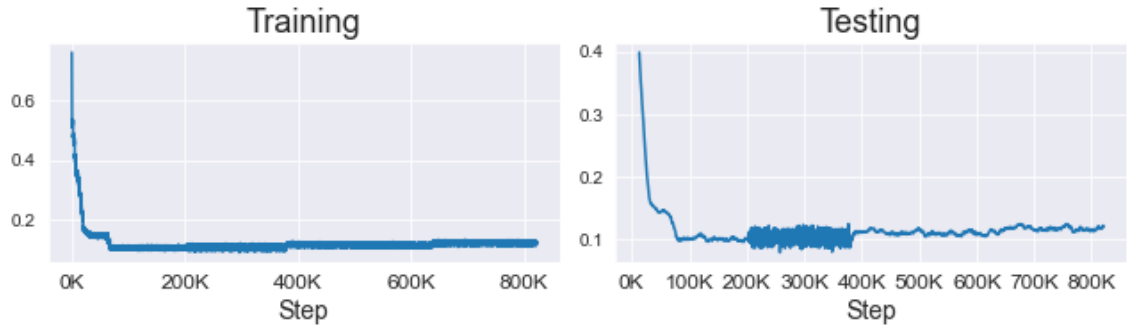

Figure 11: Reconstruction loss curves for phase 2 (smoothed over 25 steps)

We see the reconstruction loss in Figure 11 holds relatively constant throughout the training process. This is especially important because this is our primary goal as we decrease α throughout training. The model is able to learn to provide quality samples and maintains this knowledge even as its latent distribution is coerced to one with more defined structure.

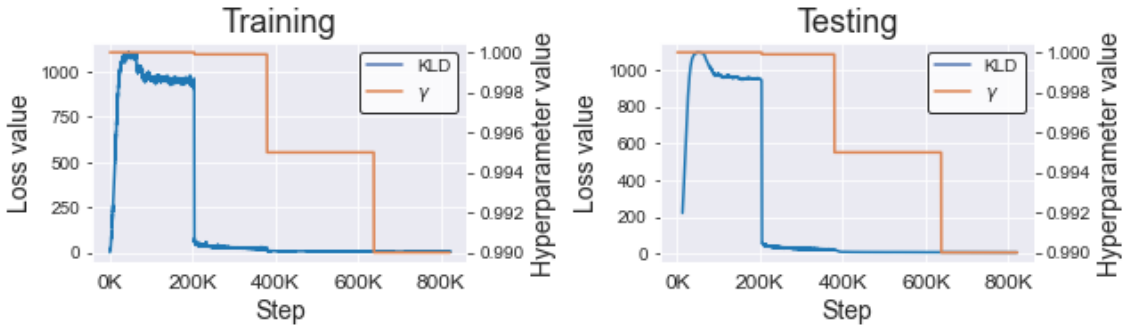

Figure 12: KL divergence (KLD) loss curves for phase 2 (smoothed over 25 steps)

We see from Figure 12 how the KL divergence's loss curves respond rapidly to changes in α. Early on in training when $\alpha = 1$, the KL divergence is allowed to spike. Since there's no penalty at this rate, it could theoretically increase infinitely, but generally settles around a value of 1,000. When we compare this to Figure 11, we see that it's over this time period where the VAE model minimizes its reconstruction loss. As we decrease the value of α, the KL divergence rapidly begins to decrease. We chose to complete training at $\alpha = 0.99$, due to the fact that absent other hyperparameter changes, this is where we began to see large divergence in the reconstruction loss.

Finally, we view our MMD loss which shows how initially the MMD settles around a local optima and gradually begins to increase as the reconstruction loss is minimized. However, as we decrease the value of epsilon, the loss incurs a greater penalty and is quickly minimized further to a state of relative convergence.

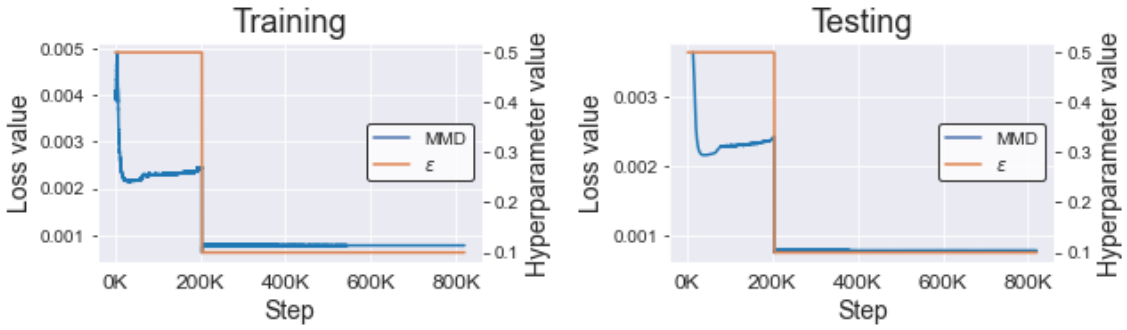

Figure 13: MMD loss curves for phase 2 (smoothed over 25 steps)

Our reconstructions in Table 2 show similar quality to those from phase 1 and we conclude that the VAE has succeeded in modeling the distribution of BART's embeddings.

Start	Middle	End
View songs Fam FamTr...	Package Dimensions: ...	55% Cotton wash% Pol...
erslowslowpointslow ...	Shipping Weight: Tam...	We dress that you du...
Musprogram Fam Fam a...	Go to Your Orders to...	Y% Polyester/4% 100a...
Mus tro troungeoung...	Package Dimensions: ...	Rream Spor laborl
Drawrett alarms cont...	Shipping Weight: 3.2...	Imported
Mus tro tro troician...	Shipping Weight: 7.2...	Please for your Casu...
Setprogram alarms al...	Package Dimensions: ...	ColorSuggest checkro...
MusMus alarms alarms...	One Size (Fits lava ...	Shaft measures appro...
shoulder troouch al...	Long sleeve p Reetta...	nt in the box
elastic 332 Plan tr...	KILL-l optionallyiz...	Material: Z Tyson me...

Table 2: Sample reconstructions at different points in the training process for phase 2

5.1.3 Phase 3

In our last phase we only have two metrics to monitor. The first of these is the loss curve.

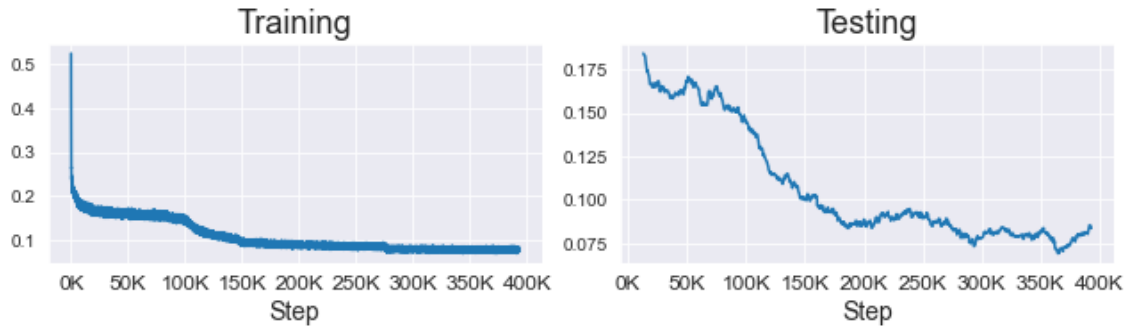

Figure 14: Loss curves for phase 1 (smoothed over 25 steps)

Figure 14 shows that the BART model achieves remarkable performance when training on the corrupted inputs from the VAE model, which demonstrates the viability of BART's construction. In fact, the reconstruction loss is actually minimized beyond what was achieved in phase 1 with non-corrupted inputs. The entropy in Figure 15 also appears to decline steadily in line with the other phases. Ultimately, we are satisfied that this will improve our interpretation of the VAE model's outputs.

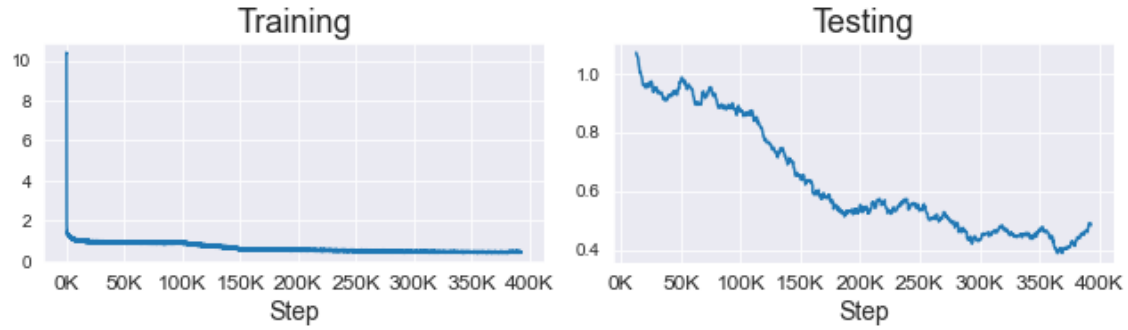

Figure 15: Entropy in output probabilities for phase 1 (smoothed over 25 steps)

In viewing our recreations in Table 3, we see some of the best reconstructions of any of the phases, which provides further confidence in our model's ability.

Start	Middle	End
nan	Imported	Package Dimensions: ...
Shipping Weight: 7.2...	Package Dimensions: ...	Package Dimensions: ...
Fiagoamsole:	Shipping Information...	It is cured with hea...
in of	Package Dimensions: ...	Material:Cotton
happ [|%	Shaft height measure...	Package Dimensions: ...
100% Leather	Brown l durable mine...	Women
, Wireless squeezed	Sn welcomed cotton, ...	Sun UPF 50+ sun sun ...
3.5%	Lightweight material	Package Dimensions: ...
– neck pockets	Machine Wash	This classic necklac...
- 1„	Authentings makes c...	Lobster-claw clasp

Table 3: Sample reconstructions at different points in the training process for phase 3

5.2 Inference

Table 4 depicts the numerical results of our inference methodology. We first apply Algorithm 1 to a random sample of 50,000 of our product details. We perform this algorithm for $\alpha \in \{0.9, 0.95\}$. We then apply our model to each community detected. We also thought to construct a methodology by simply averaging BART's encoder embeddings across the community and feeding that through to the decoder as a baseline comparison. As a test metric, we wanted to know how the singular community representation compared to the individual embeddings in the community, so we've computed the mean squared error between the individual embeddings from BART and the embedding returned by passing the singular representation to BART's encoder.

What we find is that our VAE model typically outperforms BART both in terms of average distance from the individual embeddings as well as entropy, meaning our model has greater confidence in it's predictions of the averaged values. We choose to analyze the outputs for $\gamma = 0.95$ in this section but include the same analysis for $\gamma = 0.9$ in Table 8 and Figure 17 in the appendix.

γ	**Total Communities**	**Average Size**	**Average Entropy (BART with VAE)** [6]	**Average Entropy (BART)**	**Average MSE (BART with VAE)**	**Average MSE (BART)**
0.95	13	22.6923	0.0	0.0418	0.0401	0.0418
0.9	45	23.8	0.0	0.0463	0.0453	0.0463

Table 4: Outputs from conducting recreations

We show our singular reconstructions in Table 5, as well those created by averaging BART's embeddings

[6] Values are small positive numbers that have been rounded to 0.

and passing them to the decoder and two representative product details for each community. While imperfect, we believe that our preliminary results show that we've constructed sentences that make some logical sense to a human reader. When compared to the BART-only model, the VAE model seems to produce more cohesive results. The logical structure of some of the reconstructions indicates that the VAE model has been helpful in retaining information across the merging of embeddings.

To understand why this is the case, we sought to examine the actual inputs from when the latent representations are merged. We visualize this in Figure 16. In the case of the VAE model, the embeddings are averaged immediately after reparameterization before the sequence is fed through the first decoder input layer. For BART, averaging occurs right after it's decoder and before the sequence is fed to the decoder. We plot a heatmap for this averaged matrix of the tokens against the latent dimension, with the first token positioned at the top of the figure. What we see for the VAE model is that the entire distribution is variable both in dimension and token position. For BART, there exists some less prevalent variation for non-special tokens at the top of the graph and the graph generally remains consistent for later tokens, likely indicating that these are padding tokens. We believe this increased variation across the latent dimension is a strength of our model. Whereas BART seeks to provide some regularity across the latent dimension, our model learns to recreate quality sequences from almost entirely from noisy input. By learning with such a disadvantage, it's able to better generalize its performance and proves more robust against the averaged values.

[7] The model almost appears as if it's actively trying to total its numbers to 100 for fabric percentages, which would be a significant insight if it had developed that intuition. More work would be needed to confirm this, however, and it's left to future research.

Rep. 1	Rep. 2	BART with VAE [7]	BART
56% Cotton/38% Polyester/6% Spandex	57% Cotton/38% Polyester/5% spandex	65% Cotton, 25% Polyester, 16% Spandex	100% Cotton, distracted% Polyester/ 11% Spandex
70% rayon, 26% nylon, 4% spandex	65% Rayon, 31% Nylon, 4% Spandex	65% Rayon, 27% Acylon,08% Spandex	100% Linon/% Apexon/ 11% Spandex
Material: 90 percent polyester 10 percent spandex	Material:90% Polyester 10% Spandex	Material: 80% Polyester,13% Spandex	Material:rate% Polyester,8% Spandex
70%cotton 30%polyester	70% polyester/ 30% cotton	65% Cottonx,%% nylon	100% Cotton Cotton% Polyester
74% Nylon/26% Spandex	85% nylon, 15% spandex	65% Nylon/20% Spenduedex in	82% Nylon/% GHz Kevand [|
56% Cotton/25% Polyester/17% Viscose/2% Elastane	68% Cotton, 28% Polyester, 3% Viscose, 1% Elastane	57% Cotton/23% Xester/13% Elast	100% Cotton/24% Poly metal/ 11% Vastose
50% Cotton/25% Polyester/15% Rayon	50% Polyester/25% cotton/25% Rayon	50% Cotton/24% Acrylic, 25% Rayon	50% Cotton/24% Brandotton/10% lewdon
Model Number: 654486010	Model Number: 654468400	Model Number: 6 joked001001	Model CA: Whitman402430
Weight: 3.6 Grams	weight: 2.84 grams	Weight: 2.65 grams	Weight: 8.5 grams
Measurement (tested sz 6, approx.): Heel 4.25" Shaft (w/heel) 7.5" Opening 9", True to size, Brand new with original or Alrisco shoe box.	Measurement (tested sz t6, approx.): Heel 0.25" Shaft (w/heel) 5.75" Opening 11", True to size, Brand new with original or Alrisco shoe box.	Measurement (tested sz 6, approx.): Heel 4.5" Shaft3ference/	Measure. (TiciansIZ 6,/ briefUpel 5 325"illaft"(the/
92% Cotton+8% Spandex	96% Cotton 4% Spandex	97% Cotton,%,Spandex	92% Cotton 24 14% Spandex
85% Rayon,15% Polyester	95% Rayon, 5% Polyester	65% Polyamide,13% Cashbed	55% Polyester short cavalry% [| [|
90% POLYESTER, 10% ELASTANE	95% Polyester/25% Elastane	84% Polyester,13% Elastane	8% Polyester cavalry 6%ritastane

Table 5: Sample of $\gamma = 0.95$ singular community reconstructions for our model compared to reconstructions created by averaging BART's embeddings

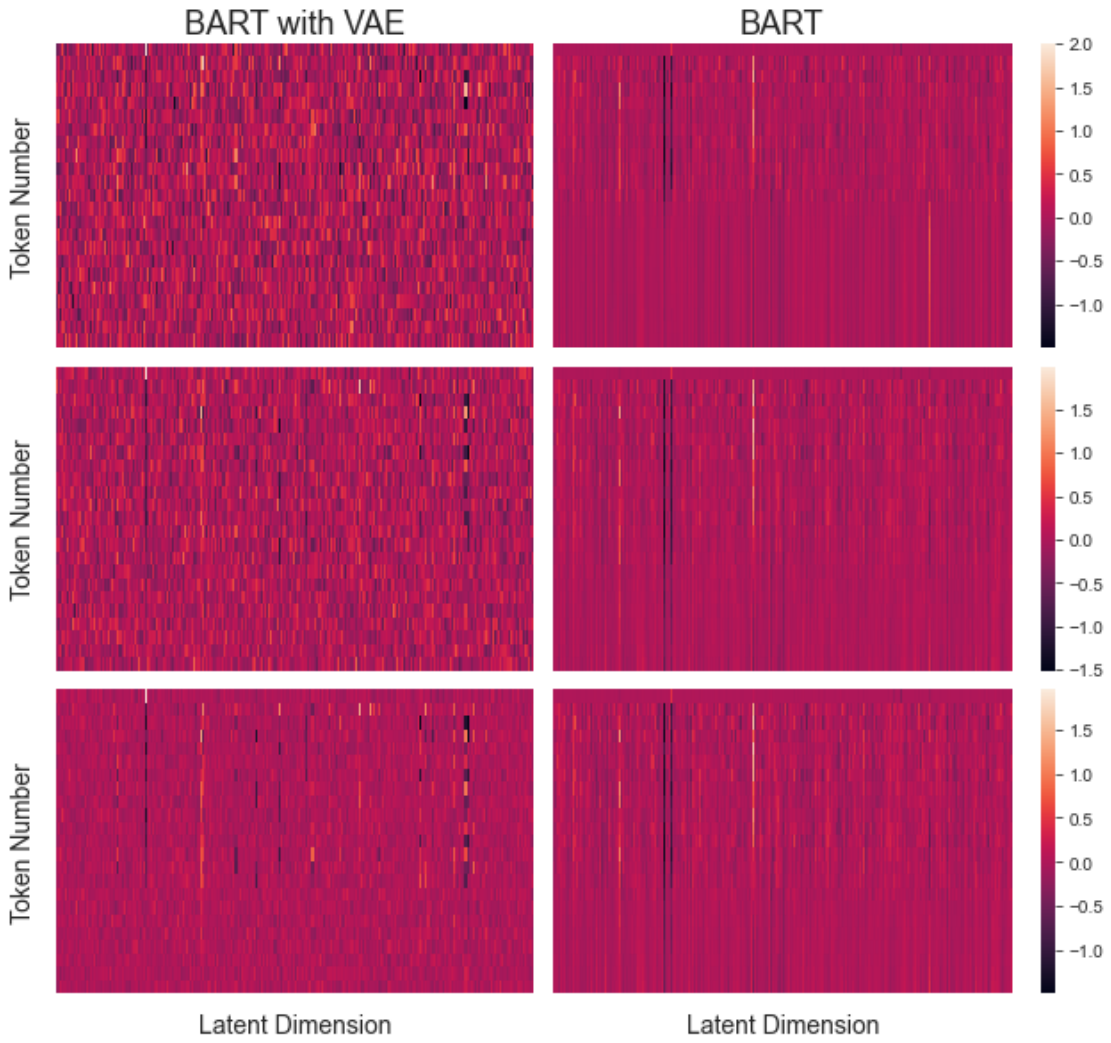

Figure 16: Sample of $\gamma = 0.95$ averaged latent embeddings for our model compared to BART; sequences begin at the top of the y-axis

6 Conclusions

From our model's results, we believe we've demonstrated the viability of an approach to providing consistent descriptions of product details for attribute extraction. While the model is imperfect in its current state, an increased access to computational resources would allow further fine-tuning to achieve better performance. We do not directly address the problem of mapping individual descriptions to the singular description, but we do note that this should fall under the well-developed scope of neural machine translation research.

Future expansions upon this work should derive methods for such a mapping, as well as focus on better tuning to the hyperparameters of the model. Additionally, this model may greatly benefit by attaching a penalty to the objective function of the VAE model that is more directly correlated with constructing a singular community representation. One such example may be utilizing a penalty similar to the MSE inference metric comparing the singular reconstruction the individual community embeddings. This would also enable the use of pooling layers such as in SBERT through which we could backpropagate.

7 Appendix

7.1 Model Layer Sizes

Layer	Layer Type	Input Dimension	Output Dimension
Encoder Output Layer	Fully-connected	768	512
Mu Head	Fully-connected	512	256
Sigma Head	Fully-connected	512	256
Decoder Input Layer 1	Fully-connected	256	512
Decoder Input Layer 2	Fully-connected	512	768
LM Head	Fully-connected	768	50265

Table 6: Layer sizes for the VAE model

7.2 Phase 2 Training Hyperparameters

Last Training Step	γ	α	λ
203,838	2.0	1.0	5.0
379,390	10.0	0.9999	5.0
637,288	10.0	0.9950	5.0
821,110	10.0	0.9900	5.0

Table 7: Training hyperparameters for phase 2

7.3 Inference Results for Gamma = 0.9

Rep. 1	Rep. 2	BART with VAE	BART
65% Polyester, 30% Cotton, 5% Spandex	70% Polyester, 20% Cotton, 10% Spandex	65% Cotton/,% POL Fiber,13% Angats prescrip-tionsexating	100% Cotton, curious%%ric,8% C Cexex
Pendant Size: 2 inch (4.5cm) L x 0.7 inch (2cm) W	Pendant Size: 1 (2.5 cm) L x 1 (4 cm) W	Pendant Size: 2.5„.5„ sides:0.In,Chain.	P py Size: 2.7"(8.8," Width: 0.8"(:.
Heel height: approximately 3.5 inches	Approximately 3.75" heel height	Heel Height: 2, Platform"2)	Heel Height: 2 777
63% Acrylic/29% Wool/8% Nylon	60%acrylic, 25%Nylon, 15%Wool	49%rasile,24% Rubber„ scenario% Fiber,8%	60% Polyrylic/100% Kev,8%% Pink, C%
Machine wash warm, do not bleach, tumble dry low	Machine wash cold. Do Not Bleach. Tumble Dry Low	Machine wash cold,res note„000 fade, high Proof„ Dry clean,	Machine washMus, advise vowel vowel optionally C iron ironoptumble bleachumble. ...
weight: 2.84 grams	Weight: 2.10 Grams	Weight: 2.08 grams	Weight: 8.9 grams
Size:M Bust:88CM/34.7" - Shoulder:37CM/15.4" - Sleeve:17CM/7.9" - Length:55CM/24.4"	M: Bust: 108CM/42.5", Sleeve: 25CM/9.8", Length: 62CM/24.4"	Size:Dim„„„„„„„„„„„„„„„	Size:12,B",777 WcmSh,HT",) Cann,
94% Nylon, 6% Elastane	94% nylon/6% elastane	72% Polyester, 54% TherNutAM Co	8% 12ester,%% Linastane
Model Number: 654486010	Model Number: 654468400	Model Number: 2 downfall0000	Model cavalry: Whitman-NAME2430
59% Rayon, 41% Nylon	50% Nylon, 50% Rayon	65% Polyylon, 15% conventionide,	100% Polyon,%% dragged Wooden

Table 8: Sample of $\gamma = 0.9$ singular community reconstructions for our model compared to reconstructions created by averaging BART's embeddings

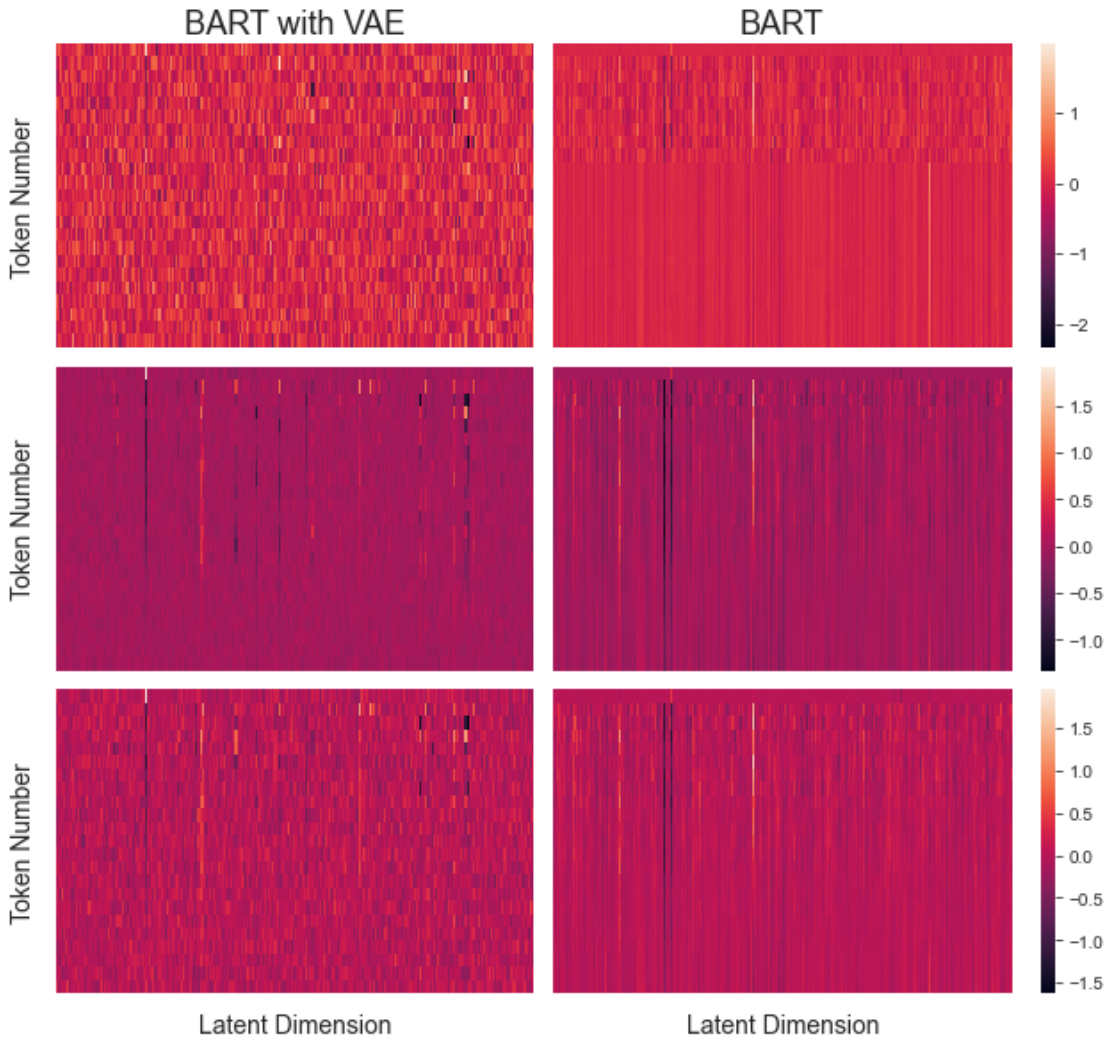

Figure 17: Sample of $\gamma = 0.9$ averaged latent embeddings for our model compared to BART; sequences begin at the top of the y-axis

References

[1] Rayid Ghani, Katharina Probst, Yan Liu, Marko Krema, and Andrew Fano. Text mining for product attribute extraction. *SIGKDD Explorations*, 1:41–48, 2006.

[2] DEKANG LIN and PATRICK PANTEL. Discovery of inference rules for question-answering. *Natural Language Engineering*, 7(4):343–360, 2001.

[3] Duangmanee Putthividhya and Junling Hu. Bootstrapped named entity recognition for product attribute extraction. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1557–1567, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics.

[4] Petar Petrovski and Christian Bizer. Extracting attribute-value pairs from product specifications on the web. In *Proceedings of the International Conference on Web Intelligence*, WI '17, page 558–565, New York, NY, USA, 2017. Association for Computing Machinery.

[5] Guineng Zheng, Subhabrata Mukherjee, Xin Dong, and Feifei Li. Opentag: Open attribute value extraction from product profiles. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[6] Rongmei Lin, Xiang He, Jie Feng, Nasser Zalmout, Yan Liang, Li Xiong, and Xin Luna Dong. Pam: Understanding product images in cross product category attribute extraction. In *KDD 2021*, 2021.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina N. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.

[9] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[10] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461, 2019.

[11] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084, 2019.

[12] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Infovae: Information maximizing variational autoencoders. 06 2017.

[13] Hao Fu, Chunyuan Li, Xiaodong Liu, Jianfeng Gao, Asli Çelikyilmaz, and Lawrence Carin. Cyclical annealing schedule: A simple approach to mitigating kl vanishing. In *NAACL-HLT (1)*, pages 240–250, 2019.

[14] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. 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 188–197, Hong Kong, China, November 2019. Association for Computational Linguistics.

[15] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. *CoRR*, abs/2004.09297, 2020.