Efficient Recovery of Text from Embeddings

Ramgopal Venkateswaran
ram1998@stanford.edu

1 Introduction

1.1 Motivation

Hosted vector databases are a ubiquitous way of organizing data and quickly returning information that is relevant to a specified query (Pinecone [2023]). For example, one might use such a database to search for up-to-date documents related to a piece of text, and feed it into the context window of an LLM in order to generate more real-time information (retrieval-augmented generation). It is useful to better understand how secure the generated text embeddings are - if we create an embedding of a piece of text containing private information, how easy is it for a malicious third party to invert the embedding and decode the contents of the text (and how can we make this hard)?

Beyond this direct motivation, it is also interesting in its own right to study this problem as it is an instance of a more general class of inversion problems that also appear in other domains. For instance, there has been in-depth work in image captioning, including Radford et al. [2021] and Gal et al. [2022]. This area of work is also thematically related (going from images to text and vice versa), even if the underlying models used are quite different from the ones we'll see in our setting.

A final source of motivation for this work is that it is a natural setting to study controlled generation for non-autoregressive text models. While powerful autoregressive text models have become ubiquitous for discrete settings like language modeling tasks (e.g. Brown et al. [2020]), non-autoregressive techniques such as diffusion models (Song and Ermon [2020]) have become state-of-the-art in continuous domains such as image generation. Inverting embeddings can be viewed as a discrete generation task, but where we want to control the generation based on continuous input (namely the embedding) that contains complete information about the expected output. This could be an interesting setting within which to explore using text diffusion models.

1.2 Problem Specification

We specify the problem similar to Morris et al. [2023], where we have an embedding model ϕ that converts text x into an embedding $e = \phi(x)$. We are given a train dataset comprising of (e, x) pairs and at test time, we want to infer the input text (x) given previously unseen embeddings (e), maximizing the "closeness" to the actual input text in terms of metrics such as BLEU scores and the cosine similarity of their embeddings.

The setting in Morris et al. [2023] involved unrestricted query access to ϕ. We will instead focus on the more constrained case where we have no access to ϕ - that is, we cannot query it on any text in addition to the train dataset that we already have access to, whether at training or generation time.

2 Related Work

2.1 Embedding Inversion

Song and Raghunathan [2020] was the first work to look into this issue, and they used a "bag of words"-style approach to demonstrate the ability to recover between 50% to 70% of the input words from embeddings - they considered two scenarios, one where we have "white box" access to the model

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

that embeds the text (i.e. access to the model weights and architecture) and one with "blackbox" query access. Recently Morris et al. [2023] continued this line of investigation, focusing specifically on the blackbox setting - this work showed that it was possible to recover a significant amount of textual information from embeddings, with their best model having an *exact-match* accuracy of 92%.

They frame the problem as a controlled generation problem (with the control being the embedding that we want to be similar to), and draw inspiration from Welleck et al. [2022], which introduced a method of improving language generation through self-correction. The method used here was a search-based method, with two parts to the generation as illustrated in figure 1.

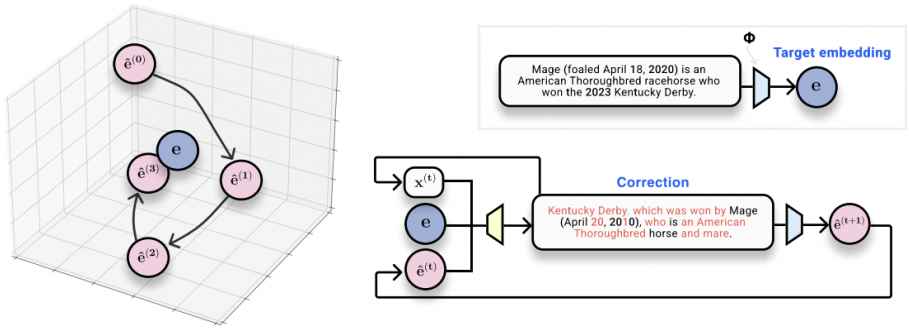

Figure 1: This corresponds to Figure 1 from Morris et al. [2023], showing their predictor-corrector model.

First, they train a decoder model to output an initial guess. Then they train a separate "corrector" model to output a new guess given the original guess, the desired embedding, and the embedding of the original guess (which is obtained via a blackbox query to the embedding model). They then iteratively correct an initial prediction, using beam search to marginalize over multiple guesses at each time step and boost accuracy.

2.2 Extending this to the Zero-Query Setting

A key property of the model in Morris et al. [2023] is that a lot of its power comes from the "corrector" which refines initial guesses based on new queries to the embedding model - the base model achieves a BLEU score of 31.9 with 0% exact match, while 50 steps of iteration (and sequence-level beam search) can bring this up to a BLEU score of 97.3 and 92.0% exact match on the Natural Questions (NQ) dataset when using GTR embeddings (from Table 1 of the paper).

In general, each step of the corrector model requires at least one query to our embedding model (and more when using beam search). These queries are relatively expensive both in terms of computational cost (e.g. time taken for hypothesis generation). Additionally, they are also the source of the assumption that the "attacker" who wants to invert the embeddings has black-box query access to the embedding model; if we did not need a separate corrector model like this, we would not need this additional assumption. The paper notes both of these limitations and mentions that training an imitator model could be a potential future direction.

We look to extend the inverter + corrector set-up in Morris et al. [2023] to the zero-query setting by training an encoder to mimic our embedding model, and replacing all our black box query accesses with queries to this model instead.

2.3 Text Diffusion Models

Separate from the line of work on embedding inversion, there has recently been active research into the use of diffusion models in discrete settings like text, inspired by their success in continuous domains (e.g. images). Li et al. [2022] introduces one way of doing this, as illustrated in figure 2 taken from the paper.

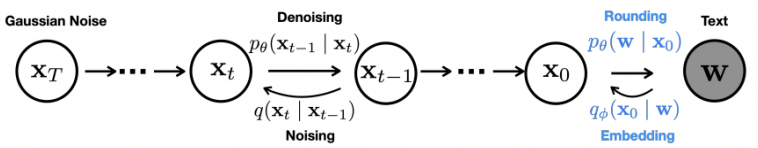

Figure 2: This corresponds to Figure 2 from Li et al. [2022], showing the graphical models representing the forward and reverse diffusion processes. For our case, we are specifically looking to plug in a relevant classifier to guide generation at each timestep.

It uses the same denoising objective and Langevin sampling techniques from the continuous case, but with the key difference being that it adds embedding and rounding steps at the inputs and outputs to translate between continuous and discrete domains. It allows for similar ideas to those that were first introduced in Song et al. [2021] on controlled generation; in particular, by training a classifier to predict the probability that an intermediate noisy sample satisfies the control, it allows for "plug-and-play" controlled generation using the combination of a pre-trained "Diffusion-LM" and the classifier.

2.4 Inversion via Text Diffusion

Inversion can be thought of as an instance of the controlled generation problem, where the control is the embedding - we want to find a response with embedding close to the given one. Morris et al. [2023] mentions this, but also notes that it is not possible to directly adapt the model in Li et al. [2022] because they do not have access to the gradients of the embedding model in the blackbox setting.

One potential way to get around this is to use classifier-free-guidance (Ho and Salimans [2022]) - instead of trying to "plug-and-play" with a pre-trained diffusion-LM, we could directly condition the diffusion-LM on our control variable (in this case, embeddings) for (a subset of) the training samples rather than training a separate classifier.

We look to train a diffusion-LM with classifier-free-guidance for embedding inversion to text.

2.5 Other Relevant Work

We also refer to Ni et al. [2021], which introduced the GTR family of encoders, which is one of the state-of-the-art model families that are commonly used for embedding. They are based on pre-trained T5 base models, and specifically optimized on objectives related to similarity-based search and retrieval - we will use these encoders for our study, similar to Morris et al. [2023].

Finally, we note that while we limit our focus on text diffusion in this work to the embedding and rounding approach used in Li et al. [2022], there are other proposed approaches to this general problem - for instance, Lou et al. [2023] proposes a discrete version of score matching to train text diffusion models.

3 Approach 1: Inverter + Corrector in the Zero-Query Setting

3.1 Proposed Method

The inverter + corrector set-up shown in figure 1 requires the corrector's inputs (at timestep t of the iterative correction procedure) to include the current hypothesis text, $x^{(t)}$, the true embedding e, and the embedding of the hypothesis $\hat{e}^{(t)} = \phi(x^{(t)})$. Computing $\hat{e}^{(t)}$ requires a query to the true embedding model ϕ with text $x^{(t)}$. We replace ϕ with a model trained to imitate its outputs (call it ϕ'), and use the output of that model instead, $\phi'(x^{(t)})$.

For the base inverter model, we use the same architecture (including the pre-trained T5 backbone) and training procedure as in Morris et al. [2023]. We can train the imitator model in parallel with the inverter model (on the same dataset, but solving the opposite problem). After both inverter and imitator

are trained, we then train the corrector model, using the same architecture (also initialized from pre-trained T5 weights) and training procedure as Morris et al. [2023] but with the key difference being that we do not make any queries to ϕ to generate the training data - instead, we get approximations to the required true embeddings from ϕ'.

An important detail to note is that we are inherently training on less data than we would be in the case where we were allowed queries to ϕ - this is because we would not have any additional data on the true embeddings of the initial hypotheses generated by our inverter model (we only approximate it through our inverter, which has seen the same training dataset). In the original setting, we implicitly train on double the training data size by adding an $(x^{(1)}, \hat{e}^{(1)})$ training datapoint for each (x, e) training datapoint.

We initialize the imitator model from a pre-trained BERT encoder (specifically BERT-base), with two additional layers at the output: a mean pooling layer and a linear projection layer with dropout as shown in figure 3. The mean pooling layer allows us to transform the 32 token-level embeddings generated from the input text (which has length of 32, padding included) into a single embedding. The additional linear layer with dropout allows us to project it from BERT's embedding size (512) to the required size (768). The simple architecture is similar to the one used in the regression task in Galtier [2021]. We optimize the MSE loss between the true embedding and the output embedding - we considered the cosine similarity as well, which gave broadly similar results.

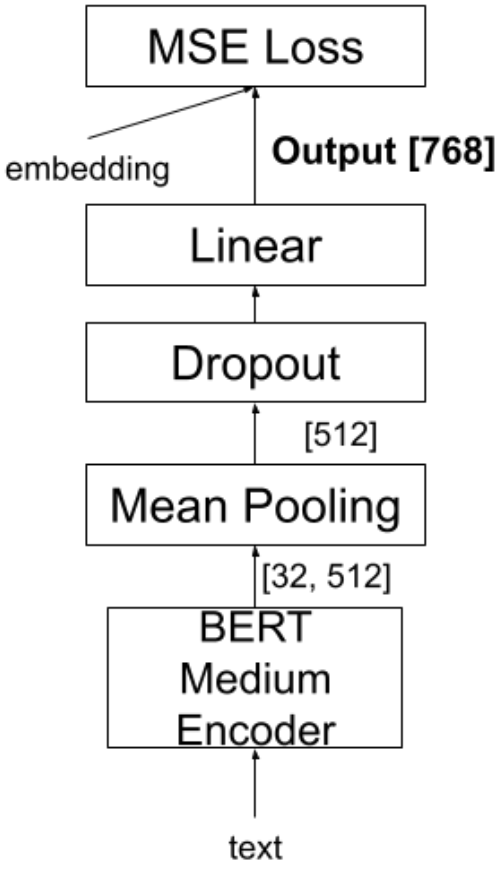

Figure 3: This is the architecture we use for our imitator model. Arrows are labeled with the sizes of the corresponding inputs/outputs.

We also considered initializing it from pre-trained T5 weights instead of BERT, similar to the inverter and corrector models - however, this might potentially give our encoder model an unfair advantage

because the true embedding model that we use (GTR) is itself extensively fine-tuned on top of a base T5 model. To prevent any unfair advantage, we use BERT, which has unrelated weights.

3.2 Results

3.2.1 Datasets

Similar to one of the set-ups in Morris et al. [2023], we will use 5 million passages from Wikipedia selected from the Natural Questions (NQ) dataset for training. The dataset has 100 tokens per passage, but we will truncate to 32 tokens each (as also done in Morris et al. [2023]). We also use the GTR encoder in all cases.

We evaluate the quality of inversions by looking at the BLEU scores and embedding cosine similarities of our inversions compared to the actual text. We will also briefly look at out-of-distribution performance on the arXiv dataset to understand any generalization capabilities.

3.2.2 Training Details

First, we create a new baseline `B1` to reproduce the pure inverter (no corrector) case from the paper. We stop the training early (before full convergence) after 2.5 epochs with resource constraints in mind; this completes within 2 days on a K80 GPU. The training settings the same as those specified in the paper. We also state the results from the equivalent baseline in Morris et al. [2023] as `B1*`. We similarly train a baseline `B2-True` for the inverter + corrector set-up from the paper with less training time, and state the result from the paper as `B2*`.

Our imitator is trained for 5 epochs on the same training dataset (with the input being the text and the ouput being the embedding), and it completes within approximately half a day with reasonable convergence observed. Note that our imitator has fewer parameters (on the order of $80M$ than our inverter and corrector models which are on the order of $300M$ parameters). After training the imitator, we train a corrector that uses the imitator model (instead of making queries to the true embedding model), calling it `B2-Imitated`.

All training code can be found in the corresponding Github repo (https://github.com/ramvenkat98/vec2text-imitator).

3.2.3 Results & Analysis

Before looking at the overall results, we report the results for the imitator model - after 5 epochs of training, it achieves a cosine similarity of 0.90 on the test set (which contains 500 examples). 1 would imply that it has exactly learned the encoding function, while 0 would imply that it is doing very poorly (note that a random embedding would achieve something close to 0 - on the order of $1/768$ - since we are in a high-dimensional space). This suggests that we can proceed with using it since it has reasonable performance - another useful thing to check was that the cosine similarity it achieved was better than the cosine similarity of the baseline inverter model itself (`B1`), which is also the case.

We now evaluate `B2-Imitated` against `B1` (baseline) and `B2-True` (upper bound, without query restrictions). We evaluate it in two scenarios: with just one step of iterative correction, and with five steps of iterative correction. We look at both cosine similarity and BLEU score of the generated text (when comparing to the real text), noting that in general BLEU score can more precisely distinguish model performance at higher cosine similarities (where a BLEU score of 1 is optimal, corresponding to an exact match, while a BLEU score of 0 has no n-gram matches).

We can make the following observations:

1. (Sanity Check) All our models are significantly better than the previous bag of words baseline before Morris et al. [2023] as expected, both in terms of cosine similarity and BLEU score.
2. Our `B1` baseline is worse than `B1*` from the paper, which is expected given that we train it for significantly fewer epochs (2.5 vs 100) owing to computational considerations.
3. Our `B2-True` baseline (inverter + corrector) improves significantly over our `B1` baseline as we would expect.

Model Type	Identifier	Query Complexity	Training Epochs	Cosine Similarity on NQ Test Data	BLEU Score on NQ Test Data	Cosine Similarity on ArXiv Data	BLEU Score on ArXiV Data
Bag of Words (from paper)	BoW*	0		0.70	0.3		
Fully Trained Base Model (from paper)	B1*	0	100	0.91	31.9		
B1* + True Corrector (1-step, from paper)	B2*	1	100	0.96	50.7		
Equivalent Inversion Base Model (early stop)	B1	0	2.5	0.83	16.2	0.77	7.5
B1 + Imitated Corrector (1-step)	B2-Imitated	0	0.96	0.86	20.0	0.81	8.75
B1 + True Corrector (1-step)	B2-True	0	0.96	0.86	20.9	0.78	8.7
B1 + Imitated Corrector (5-step)	B2-Imitated -5	0	0.96	0.84	19.9		
B1 + True Corrector (5-step)	B2-True-5	5	0.96	0.88	26.3		

Figure 4: These are the results of our experiment for using an imitator model to improve the zero-query performance of the baseline.

4. With 1 step of correction, our `B2-Imitated` model (BLEU score of 20.0) does significantly better than `B1` (BLEU 16.2) and slightly worse than `B2-True` (BLEU 20.9), as we might expect - it is much better than not having a corrector, and it is slightly worse than having a corrector with access to a fully accurate hypothesis embedding instead of an approximate one.
5. We also note that both `B2-True` and `B2-Imitated` improve cosine similarity and BLEU score on out-of-distribution arXiv data compared to `B1`.
6. With 5 steps of correction, our `B2-True` model (BLEU score of 26.3) continues to get better while our `B2-Imitated` model does similarly (BLEU score of 19.9) to 1 step of correction. This suggests that we become limited by the lack of access to the exact hypothesis embeddings and the only approximate accuracy of our encoder generated embeddings. That is, `B2-True` can continue to improve by getting closer and closer over more rounds of correction, `B2-Imitated` is capped by the limited power of the encoder, which may not be able to capture the same level of fine-grained detail that an exact embedding can.

3.2.4 Conclusion

We show that pairing the inverter with a corrector can improve performance in the zero-query setting by training a separate imitator to approximate hypothesis embeddings. However, we also do not observe improvement from iterated correction beyond 1 round in this case, which suggests that we soon reach the limits of the imitator's accuracy in generating hypotheses.

There is an **important caveat** accompanying this result: we will need to see if these results hold when we train all models (inverters, corrector, imitators) to **full convergence**. For instance, it is possible that a fully converged inverter may not benefit from an additional imitator-corrector pair in the zero-query setting, and that the current improvement we see with one round of "imitated correction" may not hold. (On the other hand, it might also be possible that a stronger imitator may start to show more significant improvement after multiple rounds of querying.) Future work should remove the computational constraints and study all models to this full capability.

Another direction that might be interesting to look at how much of the power of the corrector comes from having access to additional training data. If we simply added some hypothesis text and embedding pairs to the training data of our inverter model, how far can this bridge the performance gap to the inverter + corrector setting? If this can account for a large portion of the gap, would that suggest that having access to the true embedding model ϕ at training time is sufficient for most of the recovery and we don't necessarily need access to ϕ at query time?

A final direction that might be worth thinking about is the few-shot setting - if we are given a mystery encoder (not necessarily the GTR encoder that we used here) whose properties we don't know, how quickly can we converge to the right text given a small number of queries to it?

4 Approach 2: Text Diffusion for Inverting Embeddings

4.1 Proposed Method

We apply classifier-free-guidance (CFG) to the Diffusion-LM method proposed in Li et al. [2022]. The training objective is the same as derived in appendix E of the paper (corresponding to $\mathbb{L}^{\texttt{e2e}}_{\texttt{simple}}$), which takes the standard denoising objective and adds a loss term corresponding to the embedding step. The key difference is that, following Ho and Salimans [2022], we jointly train a model that is conditional on the input embedding and one that is not conditional on it by adding an input parameter that is either a vector of zeros (unconditional generation) or the embedding vector (conditional generation).

We slightly modify the existing architecture to concatenate this additional vector to the embeddings of each input token, before projecting it to the correct size to pass into the BERT-base encoder architecture that is used in the paper. Other than this change, we use the same architecture and 2000 diffusion steps as in the paper. A minor changes we make is to use an embedding size of 256 rather than 16 as used in the paper for the same dataset (the E2E dataset).

4.2 Results

4.2.1 Dataset

We use one of the datasets in Li et al. [2022], namely the E2E dataset containing 50K restaurant reviews. We evaluate the conditional generation on 500 examples from the held out test dataset. Note that, when sampling, we assign a weight of 1 to the conditional score estimate and 0 to the unconditional score estimate, since we want the purely conditional generation. To generate embeddings, we use the same GTR encoder as we had used above.

4.2.2 Training Details

We train the diffusion-LM model for 190 epochs (130K iterations on a batch size of 64), which takes around 1 day on a K80 GPU. During training, we set the probability of conditioning on the embedding during training time to be 0.9, similar to the probabilities that were found to work well in Ho and Salimans [2022]. We also train the embeddings end-to-end as recommended in the paper.

We additionally train an inverter (no corrector) with the same parameters as in the previous section but on the E2E dataset, in order to compare the original approach to the diffusion-LM approach.

All training code can be found in the corresponding Github repo (https://github.com/ramvenkat98/Inverting-Embeddings-Diffusion-LM).

Model Type	Cosine Similarity on NQ Test Data	Num Trainable Parameters	Uses Pre-trained Weights and Embeddings	Co-trained with unconditional model	Vocab Size
Inversion Base Model	0.828	338M	Yes	No	32128 (T5 base default)
Diffusion Model with Classifier Free Guidance (CFG)	0.723	87.7M	No	Yes	1024

Figure 5: These are the results of our experiment using a diffusion-LM model compared with the regular inverter model set-up.

4.2.3 Results & Analysis

We evaluate the inversion performance of diffusion-LM in the same way as above - in this case, we focus on the cosine similarity which is sufficient to distinguish the quality of the two models, rather than the BLEU score, which is more useful to tell apart models with similar cosine similarities.

In general, we can see from 5 that the original inverter model out-performs the diffusion-LM model. However, the table also illustrates are a few reasons why we might expect this to be the case based on our model architectures and set-up: we do not use any pre-trained embeddings or weights, we have fewer trainable parameters, and because we don't use pre-existing tokenizer (instead creating a vocabulary of size 1024 and training embeddings for it).

While diffusion-LM does not perform as well as the inverter model, we can verify qualitatively that it has learned a decent approximation of the inverted text and carries some of the original meaning when conditioned on the embedding - for example, the following is a sample from the test set:

1. **Original Text**: At the riverside, there is a coffee shop called The Blue Spice.
2. **Predicted Text**: There is a coffee shop called Blue Spice in riverside area.
3. **Unconditional Predicted Text**: The Waterman is a place that serves Japanese food. It is located in the city center customer rating is 1 out of 5.

4.2.4 Conclusion

Future work could look to train a larger version of diffusion-LM in order to fairly compare with the inverter model. The text diffusion approach also allows us to incorporate inductive bias into our examples if we pre-trained on a larger corpus of text outside or training data (for the unconditional case) - this might be another direction to try. Finally, it might be interesting to explore other discrete version of diffusion models and how they perform on the inversion task.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion, 2022.

Anthony Galtier. Fine-tuning bert for a regression task: Is a description enough to predict a property's list price? *Medium.com*, 2021.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance, 2022.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. Diffusion-lm improves controllable text generation, 2022.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion language modeling by estimating the ratios of the data distribution, 2023.

John X. Morris, Volodymyr Kuleshov, Vitaly Shmatikov, and Alexander M. Rush. Text embeddings reveal (almost) as much as text, 2023.

Jianmo Ni, Chen Qu, Jing Lu, Zhuyun Dai, Gustavo Hernández Ábrego, Ji Ma, Vincent Y. Zhao, Yi Luan, Keith B. Hall, Ming-Wei Chang, and Yinfei Yang. Large dual encoders are generalizable retrievers, 2021.

Pinecone. Pinecone, 2023. URL `https://www.pinecone.io`. Accessed: 2023-11-17.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

Congzheng Song and Ananth Raghunathan. Information leakage in embedding models, 2020.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution, 2020.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2021.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct, 2022.