There and Back Again: Autoencoders for Textual Reconstruction

Barak Oshri
Stanford University
`boshri@stanford.edu`

Nishith Khandwala
Stanford University
`nishith@stanford.edu`

Abstract

In this paper, we experiment with the use of autoencoders to learn fixed-vector summaries of sentences in an unsupervised learning task. Sentences as word vectors are fed into an encoder, either a recurrent neural network or convolutional neural network, transformed into a summary vector of fixed size, and then unfolded back into an ordered sentence using a recurrent neural network. This model is analogous to encoder-decoders used in machine translation models, and in fact we posit that the modular space that captures semantic commonalities between two languages in an encoder-decoder can be applied to a single language example by translating a language back to itself. Here, the intermediate vector space is both an encapsulation of a sentence and a seed for reconstructing it, which makes it useful for other natural language processing tasks. The aim of this project is to ultimately create effective semantic representations of variable sized text.

We train our model on 10,000 English Wikipedia sentences on an objective function that minimizes the cross-entropy between the input sentence and reconstructed sentence and that minimizes the euclidean distance between the encoding of the reconstruct text and that of the input text, so that insignificant changes or synonyms to output words are not viewed as bad reconstructions. We show that qualitatively there is some success in this model, though it overfits on the trained examples and needs to train on a larger dataset. We show that the autoencoder is able to discern useful English syntax rules that determine the useful features needed for reconstruction.

1 Introduction

Embedding sentences and paragraphs in word vector space is a fruitful and important area of research for the long-term semantic understanding of language. These efforts often operate under the assumption that the distributed representation of words encodes the characteristic attributes needed to represent the semantic value of sentences. [1] train paragraph vectors using Skip-gram and Continuous Bag of Words (CBOW) models, modifying the original method for word vectors by including paragraph tokens that contribute to the prediction task of the next word given context windows sampled from the same paragraph. The vector representations are then used to predict the following word given the paragraph summary and a context.

Their model, called *Paragraph Vector*, has several shortcomings. The summary vector must be trained using learning algorithms for each new document given. This already excludes many of the cognitive attributes expected in a summarization system: knowledge of effective chunking rules, awareness of the salient features of the language, and ability to distinguish syntax from meaning. Paragraph Vector learns the domain-specific knowledge of the text and the relation between the aggregate and the context, thereby eschewing the general for the particular rules of summarization.

Skip-gram and CBOW train by relating words to their contexts. This model is effective for word vectors but not paragraph vectors because the function of words arise *due* to their contexts, whereas the semantics of sentences arise *within* their own input. That is, words are best understood given their context, whereas sentences are best understood by the interactions occurring inside the sentence (to the extent that the *semantic quality* of the sentence is produced context-free of the other sentences, which is mostly true).

1.1 Textual Reconstruction

One way of generating representations that retain the semantic meaning is having that the summarization vector can be reconstructed back into its original text. In this case, we create representations with the potential energy for reconstruction of the semantic content of the original text. In our project, we will attempt to create sentence (or any variable sized) representations using an encoder-decoder (an autoencoder). This is an unsupervised learning model (or a supervised one that uses its own input as the label), which we will use over text in random Wikipedia articles.

This approach is analogous to Skip-gram and CBOW in a unique way: the encoder can be likened to a sentence level CBOW, with all the words in the sentence acting as context to one underlying idea for the whole sentence. Likewise, the decoder can be likened to a skip-gram model where one vector generator a whole sentence. The advantage of this model is that it simultaneously incorporates a sense of both of these approaches in training.

Encoder-decoders of this sort are often used in machine translation tasks. We use this approach because, interestingly, textual reconstruction can be viewed as a similar problem to machine translation. The goal of machine translation is to find a *semantic isomorphism* between two languages. If the semantic qualities of one language can be condensed to an intermediate representation that is also good for the syntactic unfolding of the next, then this intermediate representation is the modular link between the two languages.

When both languages are substituted to be the same, and the translation model is trained in the same way, the intermediate representation becomes both a seed and encapsulation of the language. This space is now fit for potential applications of answering questions, generating a next sentence, and combining sentences through learned vector transformations since it both expresses language and produces it. These are some of the possible applications of the learned model, though they will not be treated in this paper.

Word vectors of a sentence will be fed through an encoder and decoder, the latter of which is modelled via Recurrent Neural Networks (RNNs) for sentence generation and the former of which is flexible to different approaches that create and update hidden states. The encoder is trained to produce a final hidden state known as a *summary vector* and the decoder network is trained to generate an output sequence by predicting the next symbol given its hidden state and the summary vector; the probability of the output can thus be computed as the product of conditional probabilities of words given all the previous words. The loss function will compute some distance metric between the input and output sentences, which capture how different the output is from the input.

Translating English back to itself offers a surprising advantage: a loss function can train to not only expect that the input and output text is similar, but also feed the output into the encoder and expect the hidden representation of the reconstructed text to be similar to that of the input text with the intention of the reconstructed text having a similar encapsulation to the input text. This fulfills the aim of this project to create effective semantic representations of variable sized text.

We will be training our model on Wikipedia articles using GloVe Wikipedia-trained word vectors. To evaluate our model over a large corpus, we will use the BLEU metric, which has been previously used to score machine translation models. We will also be subjectively assessing our output text.

2 Data Collection and Processing

For the purposes of this project, we decided to train our models on the Wikipedia dataset. For every article, the WikiMedia foundation keeps a record of the text i.e. the body of the article and the meta-data associated with it. Since we were only interested in training our model on the constructs of English language, we decided to ignore the meta-data and only scrape the raw contents out of wikipedia articles. We obtained a XML dump of 10,000 articles and parsed into its textual core using an open-source utility called WikiExtractor. This module takes in a dump file and generates multiple text files, each file containing several articles.

In order to make this parsed data actually usable for our models, we need to extract sentences from these articles. Identifying sentences in paragraphs is not as straightforward as one would imagine. A sentence need not always end with a period. It could also terminate with a question mark or an exclamation point depending on the context of the situation. Also, it would be crucial not to confuse periods with decimal notation. Looking at the complexity of this task, we sought to leverage previous work done in this field, namely the Natural Language Toolkit (NLTK). This toolkit provides an interface to segment chunks of text into sentences and then into words. The data boiled down to a list of lists where each inner list represented a sentence. The models could now be trained either at the character level or GloVe representations of individual words. In the case we wanted our model to train on GloVe, sentences with words not in GloVe were discarded.

3 Autoencoder Models

3.1 Basic Model

In this paper, we propose a neural network architecture that learns to encode a variable-length input sequence **x** into a fixed-length vector representation c and to decode c into a variable length sequence **y** that is trained to resemble the initial input. We will base our project on a model for machine translation developed in [2].

The encoder is a Recurrent Neural Network (RNN) that reads each symbol of an input sequence x_t independently as a word vector and uses it to update the hidden state h_t of the condensed representation over a nonlinearity f,

$$h_t = f(h_{t-1}, x_t) \tag{1}$$

Once all symbols of the input are passed, the final hidden state is denoted as the summary c of the input. The decoder is an RNN that is used used to generate an output sequence **y** given a summary c. The decoder updates a hidden state h_t using knowledge of the previously outputted word and the summary sentence c, which it uses as a signpost for the summary of the sentence. The hidden state of the decoder is computed using a nonlinearity g by,

$$h_t = g(h_{t-1}, y_{t-1}, c) \tag{2}$$

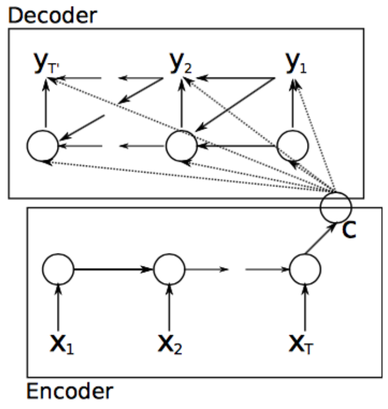


Figure 1: Basic RNN encoder-decoder model. Cho et al. 2014

where the conditional probability of an output symbol y_t is generated using the hidden state at that time interval with a nonlinearity q by

$$P(y_t|y_{t-1}, y_{t-2}, ..., y_1, c) = q(h_t, y_{t-1}, c) \tag{3}$$

The decoder continues to unfold (to a maximum) until it outputs the end-of-sentence token. As in Cho et al. 2014, we recommend the use of Gated Rectifier Units (GRUs) for f and g.

We now propose improvements for a more expressive model.

3.2 Bidirectional RNN Encoder

The summary vector c does not only have to be computed as the hidden state at the end of a sequence. A Bidirectional RNN can be used to encode a summary of the sentence at each word of the sequence by reading through part the sequence until that point forward and the rest of it backward. This produces a summary vector for each word in the sentence,

$$\begin{aligned}\overrightarrow{h_t} &= f(\overrightarrow{W}x_t + \overrightarrow{V}\overrightarrow{h}_{t-1} + \overrightarrow{b}) \\ \overleftarrow{h_t} &= f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t-1} + \overleftarrow{b}) \\ c_t &= g(U[\overrightarrow{h}_t; \overleftarrow{h}_t] + c)\end{aligned} \tag{4}$$

The final summary vector c can either be computed by averaging all c_t, or by training a final neuron for the entire length k of the sequence,

$$c = g(W[c_1; ...; c_k] + c) \tag{5}$$

However, we will not be training our model with a bidirectional RNN encoder because we expect it to outperform the single layer RNN by only a little and to perform worse than the convolutional neural network discussed below.

3.3 Convolutional Neural Network Encoders

Convolutional neural networks (CNNs) adapt recursive neural network approaches to analyzing language using parsing trees by computing all convolutions of the original data, automatically training filters that learn the best language features of the data. CNNs with pooling have been extremely successful as encoders into modular representations in machine translation as in [3]. The CNN model that we implement is described in [4], which transforms a sentence matrix into a continuous vector representation by applying a sequence of convolutions followed by a fully connected transformation to the output space.

3.4 Stacked Autoencoders

Stacked autoencoders are autoencoders where the output of one autoencoder feeds into the input of another autoencoder. This models enjoys the many benefits of deep networks with greater expressive power. Suppose that $E(\mathbf{x}) = c$ and $D(c) = \mathbf{y}$ are the abstract functions denoting encoding and decoding. A stacked autoencoder with k levels reconstructs text by

$$\mathbf{y} = (D \circ E)^k(\mathbf{x}) \tag{6}$$

The autoencoders are trained one level at a time and then finetuned with backpropagation. Stacked autoencoders did not lead to measurable improvements in our model and are not further discussed.

3.5 Denoising Autoencoders

One useful form of data augmentation is to add stochastic noise to the data and have the autoencoder predict the denoised data. The denoising autoencoder then learns to distinguish ucorrupted values from the corrupted. Crucially, the loss function uses the uncorrupted $\mathbf{x}$ as the intended reconstructed text.

We test a few types of corruption. The first corruption is to with probability p remove a word from the input sentence to minimize the contribution of a single word on the semantics of the sentence. The second is to with probability p add gausian noise to all the values of a vector. This trains the encoder to generalize a word to synonyms and other words that fit the context of the sentence.

4 Training

4.1 Objective Functions

Autoencoders are trained to minimize some metric between the input and output. This is a challenge in this task because the length of the input and output sequences are not constrained to be the same. Additionally, we wish the summary vector to encode the salient semantic features of the original input, so that sentences with similar meaning have similar hidden representations. We propose two terms for the objective function.

4.1.1 Reconstruction Loss

For the reconstruction loss, we compare at each time step the generated output from the decoder and the correct output using cross-entropy loss. This loss captures the word-wise difference between the reconstructed text and the original input. We thus unfold the summary vector for the length of the original input to produce a reconstructed sequence of the same length.

$$J(\mathbf{x})_{\text{rec}} = -\sum_{t=1}^{L}\sum_{j=1}^{C} x_j^{(t)} \log(p(y_j^{(t)} = 1|\mathbf{x})) \tag{7}$$

where C is the size of the dictionary, L is the length of the input string, $x^{(t)}$ is a one-hot vector of the dictionary index of the word, $(p(y^{(t)}|\mathbf{x}) = \text{softmax}(h^{(t)})$, where $h^{(t)}$ is the hidden state at time t of the decoder.

4.1.2 Encoder Loss

The second term of the objective function allows leniency on the precision of the reconstructed text by penalizing reconstructed text that has a very different encoded hidden state. If the *encoding* of **y** has a similar representation to the encoding of the input **x**, then it must be the case that the input and output had a similar meaning for the encoder. This approach benefits from the fact that both summarizations are items of the same size and can be compared. In this term the model is *measured at the point of summarization* as opposed to at the point of the text.

Let c' denote the encoding of the deconstructed text. We can evaluate the difference between the two using the L2 norm. Hence the objective function of this metric is,

$$\begin{aligned} J(\mathbf{x})_{\text{enc}} &= \|E(\mathbf{x}) - E(D(E(\mathbf{x})))\| \\ &= \|E(\mathbf{x}) - E(D(c))\| \\ &= \|E(\mathbf{x}) - E(\mathbf{y})\| \\ &= \|c - c'\| \end{aligned} \tag{8}$$

We let the final objective function be a convex combination of the two terms,

$$\begin{aligned} J(\mathbf{x}) &= (1-\lambda)J(\mathbf{x})_{\text{rec}} + \lambda J(\mathbf{x})_{\text{enc}} \\ &= (1-\lambda)\sum_{t=1}^{L}\sum_{j=1}^{C} -x^{(t)}{}_j \log(p(y_j^{(t)} = 1|\mathbf{x})) + \lambda\|c - c'\| \end{aligned} \tag{9}$$

4.1.3 BLEU

Machine translation papers often adopt the Bilingual Evaluation Understudy (BLEU) metric to evaluate the quality of translated sentences. BLEU measures a co-occurence of n-grams between a good quality translation and a candidate translation. In our case, the original text is the good human-quality translation and the output the candidate translation. Clearly, BLEU is most effective when it evaluates translation at a corpus level (because it sees enough of the n-grams to make a good estimate of correspondence). For this reason, we will use BLEU to evaluate the success of our model *after* significant training and once we are confident the model is somewhat good in translation, after which we will translate a bulk of text to evaluate with BLEU.

5 Results

In the results section of the paper, we will present our primary results, compare the effects of the different features and tunings presented in the earlier sections, and explain those results. Because

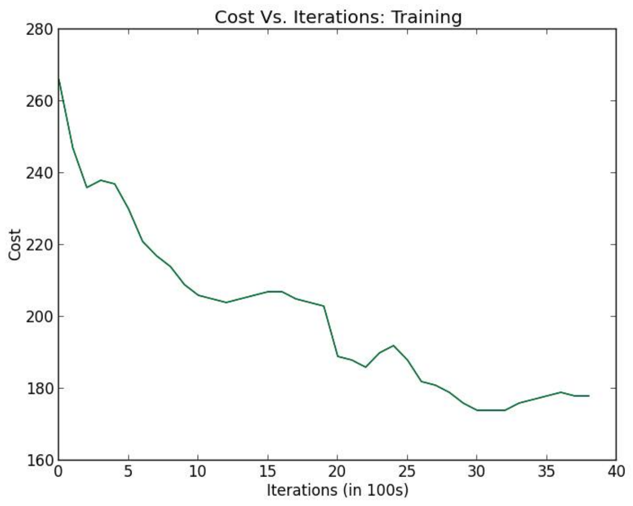


Figure 2: RNN encoder cost graph. Best cost is 174

this is an unsupervised task, other than using the BLEU score, we will compare the costs of the different models and mostly evaluate the qualitative aspects observed.

We trained our data on 10,000 Wikipedia sentences using 50 dimensional GloVe word vectors. The model was written and trained in Theano. The primary computational burden was unfortunately the Theano *scan* function, which accounted for 89% of the computing time of the compiled Theano function. Training on a GPU on all models increased the training rate by 100 times.

5.1 Initial Results

Our first trained model was the simple RNN encoder. When we first started training, we were surprised to see the loss function decreasing exponentially but the reconstructed text having no relation to the input text at all; the immediate suspicion was that we were training against the wrong objective. Additionally, with more training, the outputs began to converge to exactly the same sentence.

In fact, we realized we were making a grave but subtle mistake: at initialization, when the generated output bears no relation to the input text, the encoder loss (which motivates the encoding of the reconstructed text to be the same as that of the input) encouraged *uncorrelated, random* outputs to map to the same hidden state as that of the input. The encoding converged at a small set of hidden state vectors because unrelated sentences were all matched to the same hidden representation. The model was indeed learning, but the encoder loss, which has a much higher absolute value than the cross entropy loss, was trumping the reconstruction loss, which is more arguably the more important term.

It was thus very clear that we could not initially train the model against the encoder loss, because the autoencoder must first learn to reconstruct text and only then finetune its understanding by training on the encoder loss. We thus trained our first models on a very high value of λ (0.85) to prioritize training on the reconstruction loss.

Running the autoencoder showed that our model performed well on the training dataset, but performed very poorly on new sentences. We trained on 10 epochs with Stochastic Gradient Descent with batched updates on a learning rate of 0.01.

Here is an example reconstructed sentence from the *training set*:

Original: Although modifying glass tubing is no longer an essential laboratory technique, many are still familiar with the basic methods.

Reconstructed: Modify is eldest, essential glass is liverwort with laboratory technique and will find basic accompanies kanie alternative methods.

Here is an example reconstruction of an unseen sentence:

Original: Direct sunlight at noon can make it transpire to death, because it gets too warm, as can strong winds, which also dry out the plant.

Reconstructed: Narrow magnificant called which she criseyde posts st. lacma because hold laura out noon which new either and and dry tomorrow as some meet warmth is earth.

The reconstruction of the unseen sentence evidently picks up some word associations to the original input, but it fails to reproduce some of the salient syntactic features of the sentence. Meanwhile the training set example reconstructs the input text with higher fidelity to the input sentence.

5.2 CNN Encoder

The CNN encoder performed significantly better than the RNN encoder with a final loss of 94. The model needed to train for more epochs than the RNN encoder. Our final CNN included 10 filters; more did not seem to help with reducing the loss.

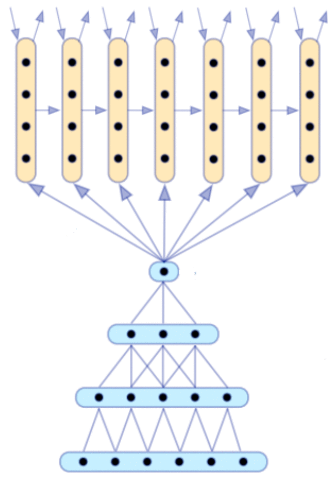


Figure 3: CNN encoder to RNN decoder. Kalchbrenner, Blunson. 2013

One of the most satisfying observations about the CNN encoder is that the model reconstructs well-formed English sentences significantly better than it does garbled sentences, which is evidence that the CNN features encode useful syntactic structures nonexistent in random input for unfolding the hidden representation.

Original: The show was cancelled after a single seven episode series.

Reconstructed: The that cancelled show issue race time one readmitted.

Original: single a episode seven cancelled series. after The show was

Reconstructed: Also black elden soon time can icc hibbard n't once 500 comparative ginny.

Interestingly, order mattered a lot in the CNN encoder.

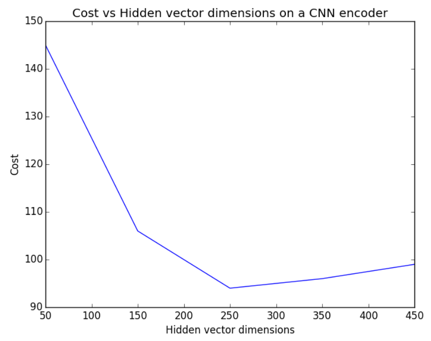
5.3 Denoising Autoencoder

The denoising autoencoder led to subtle differences in the rate that the autoencoder outputted frequent words. The denoising autoencoder with probability p (hyperparameter) excluded a word from the input and was trained to reconstruct the well-formed original sentence. Clearly, common words in sentences ('the', 'and', 'I', 'to') are the individual words that are missing most in absolute terms. The autoencoder with a much greater confidence outputted these frequent words (because in training it learnt to see them as emergent in its inputs).

To see whether this was true, we took the 20 most frequent words in GloVe and measured the difference in frequency with which they appeared in reconstructed text between the encoder trained with noisy examples and that trained without. These words appeared 8% more frequently with $p = 0.10$ and 19% more frequently with $p = 0.25$. Noisy samples are thus a successful way of encouraging the model to make fewer wild guesses by guessing at the frequent words more often.

5.4 Summary Vector Sizes

Perhaps the most paramount hyperparameter in the experiment is the dimensionality of the hidden vector representation. The final cost was lowest at a hidden dimension size of 250, plateauing with greater values. This is a promising result that, if a sentence is taken to have an average length of 15 words (as a lower bound), only a third of the sentence size is needed to encapsulate the original sentence. We were wondering why larger hidden dimension sizes didn't improve the training against the objective; a reasonable explanation is that with higher dimensionality the unfolding of the hidden state is more "sensitive" to perturbations of the hidden values, which makes the model less robust and prone to small errors in the encoding that capsize the rest of the unfolding. Large vector sizes also sharply increase the encoding cost, so it is more challenging to train against the objective of encapsulating similar text with similar hidden representations.



5.5 BLEU Scores

BLEU scores are the most standard way to evaluate the success of the reconstruction. For each of the models trained, we translated a full-length wikipedia article and compared the reconstructed with the original article.

Table 1: Encoder Model Vs. BLEU Score

Encoder	BLEU Score
RNN Encoder	0.0648
RNN Encoder w/ Denoising	0.0937
CNN Encoder	0.1024
CNN Encoder w/ Denoising	0.1152

6 Further Work

The primary limitation in our experimentation was computational needs. Wikipedia's dataset is essentially unlimited, and the nature of this unsupervised task means that there is no lack of data; yet we only trained on 10,000 sentences from Wikipedia. Even our best model was overfitting the trained data, and the best measure of the success of this approach is to pit the model against an extremely large dataset and achieve the lowest cost function.

Further tests need to be done to determine whether character-level input leads to better results. We initially started with character level but we had to train for many epochs before the model could even output correctly spelt words. However, the goal of this project to create strong fixed-size encapsulations is better met by using word vectors, since synonymous words or vectors close to the actual word vector should lead to similar encodings.

Since our initial results are promising, we suggest that further work is done to evaluate whether these hidden representations are in fact useful for natural language processing tasks. For example, they could be tested on sentiment analysis, sentence prediction or question-answering tasks. It is especially easy to process and manipulate these vectors because they have a fixed-size and can all be used to generate semantically significant text after an operation.

The codebase of this project is publicly available at
`github.com/BarakOshri/TextualReconstructor`

References

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

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

[3] N. Kalchbrenner and P. Blunsom, “Recurrent continuous translation models.,” in *EMNLP*, pp. 1700–1709, 2013.

[4] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, “A convolutional neural network for modelling sentences,” *arXiv preprint arXiv:1404.2188*, 2014.