令和四年度　特別研究報告書

LSTM を用いた作文の文章生成

龍谷大学　理工学部　情報メディア学科

T190461　浅羽隆一朗

指導教員　三好力

内容梗概

ディープラーニングは画像識別や自然言語などあらゆる分野で応用の効く技術である。画像識別では医療においてレントゲンや MRI などから異常の検知や、自然言語処理では予測文字や機械翻訳、音声認識など多岐にわたって活躍している。

本研究では自然言語処理に分野を絞り、実用的にはあまり行われていない長文の生成を行う。LSTM を用いて多数の作文を時系列データとして学習させ、題名から適切な文章を一から作成することを目的とした。このシステムの実用性として作成された文章を見本とすることで、ユーザーはより優れた文章を短時間で作ることができることが期待される。提案手法として一般的な Seq2Seq と Peeky を用いた Seq2Seq の二つの手法を用いて比較した。実験結果として一般的な Seq2Seq の方が優れていることが分かった。また、Peeky を用いた手法では全体的に同じような文章が生成されてしまった。

目次

第1章 始めに

作文を作成することは、小学生や中学生にはもちろん外国人にとっても日本語を学習するうえで重要な課題だといえる。文法を正しく活用したうえで、文脈を読み手の気持ちになって考え、単語、文を正しく羅列していかなければならない。また、さらに優れた文章を作るためには、作り手はただ思ったことや物事を説明するだけではなく、読んでいて退屈しない文章の表現方法など、工夫が求められる。そのため、日本語を理解、活用することにおいて作文の作成が最も適切な課題の一つである。実際に言語技術や伝達技術の向上として作文教育の推奨する論文が多くみられる[8,9]。しかし、一から題を考え、文章を作成するというのは難易度の高い作業だといえる。

本研究では、作文作成の一助にすべく自然言語処理に用いられる RNN(Recurrent Neural Network、回帰型ニューラルネットワーク)から勾配消失を改良した LSTM(Long Short-Term Memory)を用いて題名を入力することで自動で文章を生成するシステムを構築すること、さらにそれを例題として活用できるまでに質を高めることを目標としている。

このシステムが実現すれば教育機関における国語学習の補助や、日本語の習得の手助けになるのではないかと考える。

第2章 先行研究・既存技術

2.1 Google 翻訳

ディープラーニングが自然言語処理に適した方法であることがの一例として Google 翻訳が挙げられる。Google 翻訳は自然言語処理として現在もっとも活躍している技術の一つである。Google 翻訳は 2006 年から運用され、当時は英語からアラビア語、中国語、ロシア語に翻訳されるのみだった。この時、採用されていた機械翻訳の方法はフレーズベース機械翻訳である。フレーズベース機械翻訳とは文章をフレーズごとに分け、フレーズを単位として翻訳する手法である。この手法は翻訳する単語だけでなく周辺の単語にも着目するため、訳語選択としての正解率が高い。一方、フレーズ内の単語列に限られるため、長い文脈の情報を必要とする。そのため、語順の並び替えが困難となり、柔軟な翻訳ができていなかった。しかし、2016 年にニューラル機械翻訳が導入された。これは人間の神経回路を模した翻訳機自体がデータから翻訳モデルを学習するものである。これによって語順の規則が曖昧な言語の翻訳がより自然になり、現在では 103 もの世界の言語に対応している[10]。

2.2 文章生成

文章を単語分けし、各単語に ID を振る。単語数を次元数とし、該当する ID の要素が 1 となる One-hot ベクトルを学習させ、RNN や LSTM で出力値が次の単語となるよう学習せる。またその際、勾配を行う単語数を指定することで学習する速度が変わってくる。各要素の大きさがその単語の確率となる。どの単語を選ぶかについては確率的に単語を選ぶ方法と一番大きな値を持つ要素を ID とした単語を選ぶ方法がある。確率的に選ぶ利点として、何度、同じ題名を入力しても、毎回異なる本文を出力する。そのため、文章が気に入らなければ、同じ題名を入力するだけで、やり直すことができるということがある。この研究では確率的に単語を選ぶ手法を取る[1]。

2.3 LSTM

LSTM(Long Short-Term Memory)は時系列データの予測方法である RNN から勾配消失を改良したものである。RNN とは図 2.1 のように時系列データが計算レイヤを循環するように作られたニューラルネットワークである。前のデータからの計算結果と現在のデータが入力値となりさらにその結果が次の計算に使われる。出力値はデータの予測などに使われる。

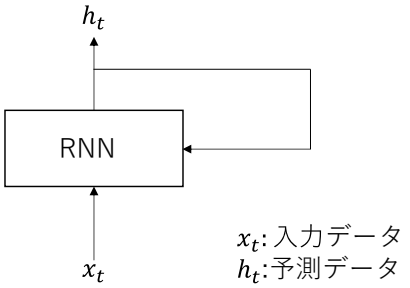

図 2.1 RNN のモデル図

しかし、RNN は学習の勾配が指数的に減少してしまうため長期の時系列データでは重みパラメータが更新されなくなる。つまりデータを学習しなくなる。このことを勾配消失という。これを改良するために RNN に前のデータを忘れるためのゲートや、データの価値を判断するためのゲートなどを取り付けた「ゲート付き RNN」が提案されている。その一つが LSTM である。Pytorch の LSTM のモデル図を図 2.2 に示す。x_tは t 番目の入力値を表し、h_tは隠れ層の出力値を表す[1]。

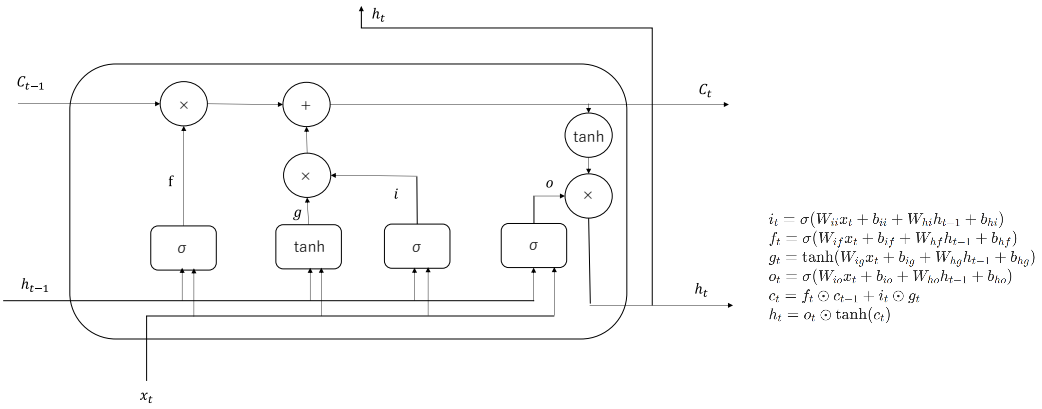

図 2.2 LSTM モデル図

2.4 Embedding

一般的に単語は One-hot ベクトル(一つの列だけ 1 で他は 0 のベクトル)として扱うが、列の数が増えると計算量が増え、学習に時間が掛かる。この問題を解決する技術として Embedding が考えられる。これは 1 にあたる列の重みパラメータのみを抽出するものである。One-hot ベクトルと重み行列の積は 0 に対応する重みベクトルは実質無効になる。そのため、この手法が成立する[1]。

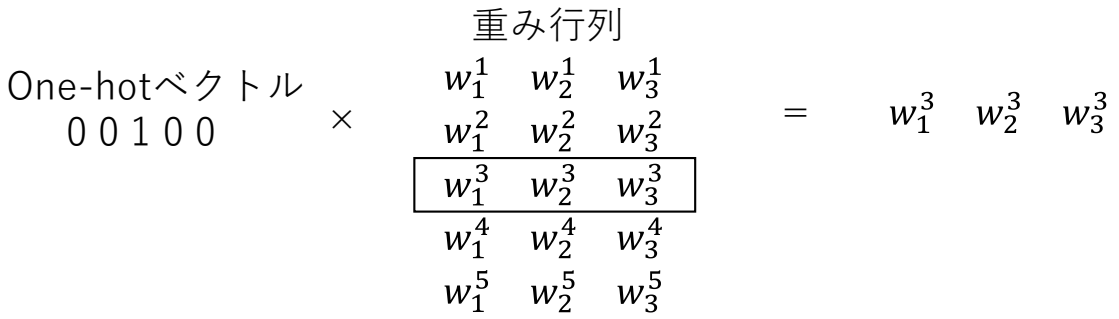

図 2.3　One-hot ベクトルと重み行列の積

2.5 Seq2Seq

Seq2Seq とは RNN などの再起型ニューラルネットワークを使い、ネットワークを Encoder と Decoder に分ける、文章から文章を生成する手法である。主に自動翻訳などに使われている。本研究では Encoder に題名のコーパスのベクトルを入力し、Decoder が本文を出力する。

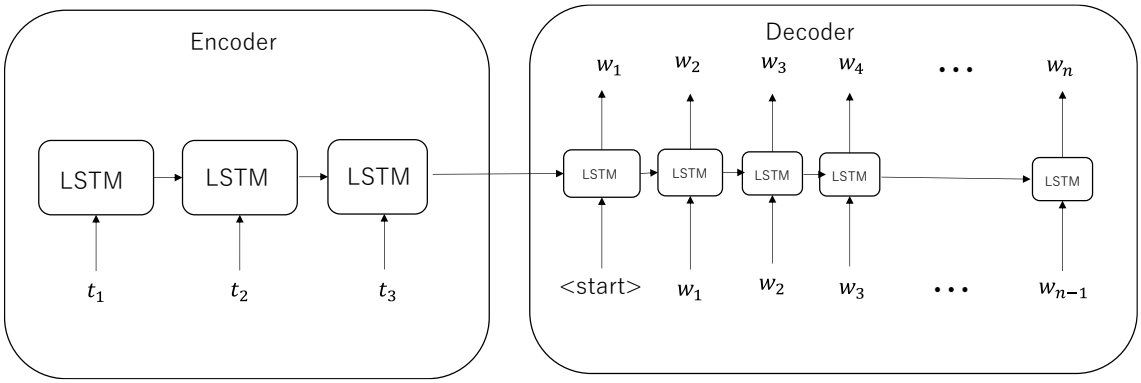

図 2.4 Seq2Seq のモデル図

さらに Peeky といわれる技法がある。Seq2Seq は一般に Encoder の出力値を Decoder に渡す際 Decoder の先頭にだけ渡される。Peeky とはさらに Encoder の出力値を Decoder の入力値と結合するものである。これによって Encoder への勾配がより、伝わりやすくなる[1]。

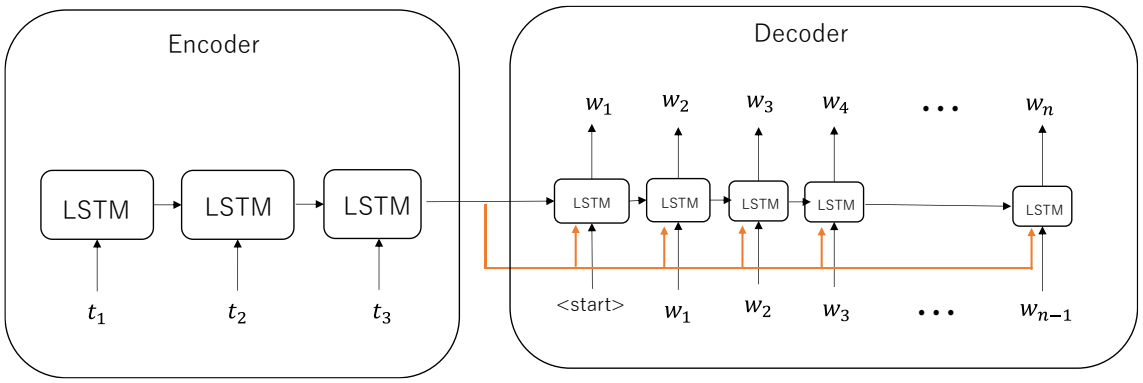

図 2.5 Peeky のモデル図

2.6 問題点

この Seq2Seq は主に言語翻訳などの技術として使われているため、長文を生成している研究はあまり見られない。その理由として Encoder と Decoder の入力値の長さが一致しているものの方が適切な文章を作ることができるからだと考えられる。本研究ではその改善案として、LSTM の長文学習と Seq2Seq の 2 つの手法を組み合わせて、文章生成の技術を LSTM による長文を学習した Seq2Seq による文章生成を作成し、行うことにする。

第3章 提案・手法

LSTM(Long Short-Term Memory)を用いて題名を入力することによる自動で文章を生成するシステムを構築し、さらにそれを例題として活用できるまでに質を高めるための手法として Seq2Seq を用いる。このシステムの実用性として国語力を養うために作文を行う小学生や中学生はもちろん日本語学習を行っている外国人の助けになると考えられる。

3.1 全体的な流れ

文章を単語分けし、各単語に ID を振り、コーパスを作る。Encoder として題名のコーパスを入力することで、Decoder として本文のコーパス出力するよう学習させる。全体の概略図を図 3.1 に示す。t が題名のコーパス、w が本文のコーパス、y が出力値である。

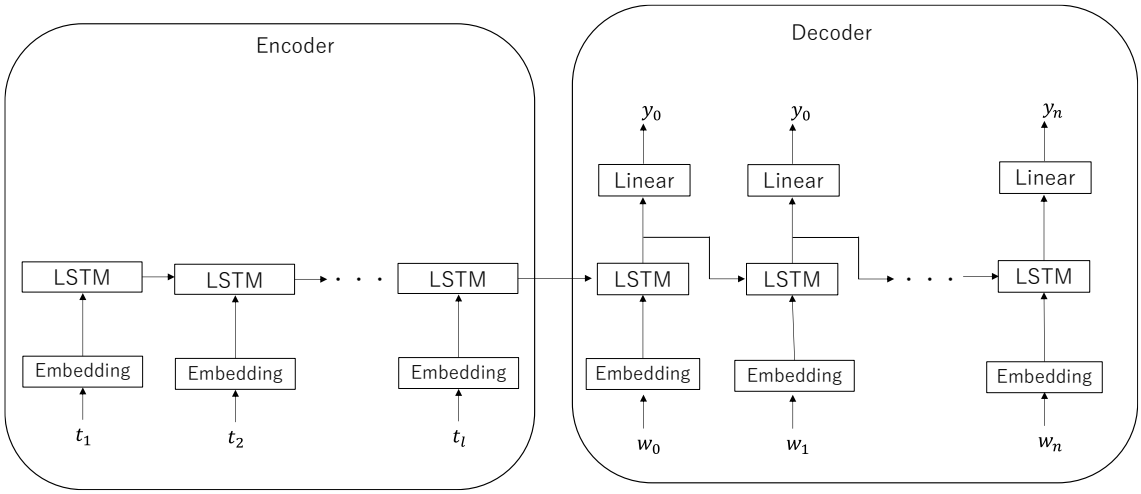

図 3.1 全体の概略図

3.2 形態素解析による単語分け

学習させる文章は単語の時系列データとするために単語分けしなければならない。そのため、形態素解析を行う。形態素解析とは自然言語を意味のある単位までに品詞などを解析することである。本研究ではライブラリとして MeCab を使う。

3.3 Embedding

単語ごとに ID を振り分け、それを入力値として Embedding を行い、300 程度のベクトルに変換する。

3.4 LSTM

Embedding の出力値を LSTM の入力値とする。また LSTM の出力値の次元を

Embedding と同じにし、重みを共有する。これによって、学習の精度が上がることが知れれている[1]。

3.5 線形回帰

線形回帰により、LSTM によって出力されたベクトルを単語数を次元数としたベクトルに変換する。線形回帰とは複数の変数から 1 つ、または複数の値を予測するための手法である。本研究では正解データの ID を 1 とする One-hot ベクトルになるよう学習する。

第 4 章　実験

4.1 実験環境

本実験では python 用機械学習ライブラリの Pytorch を用い、実験を行った。実行に用いた計算機環境を表 4.1 に示す。

表 4.1 計算機の構築環境

OS	Windows10(x64)
メモリ	32GB
CPU	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz　2.00 GHz
Python	3.9.7
Pytorch	1.11

4.2 実験概要

作文の自然言語処理を行い、学習し、自動で作文を生成するシステムを構築する。そのためのライブラリとして Pytorch を使う。作文は時系列データとしてみなせるため、時系列データを学習するための DeepLearning の LSTM を用いて、機械学習を行う。また、手法として Seq2Seq を用い、作文のタイトルを Encoder に入力し、Decoder に本文を出力する。また、その誤差を計算し、勾配降下法により、学習を進める。

4.3 実験内容

Web 上から引用してきた作文を学習させる[5, 6, 7]。一般的な Seq2Seq は単語 ID を 300 のベクトルに変換し、Embedding 層と Linear 層の重みを共有する。しかし、Peeky では Encoder の出力を LSTM 層と Linear 層に連結させるため、同じように行うと LSTM と Linear の隠れ層の数が大きくなりすぎて上手く学習することができなくなる。そのため、Peeky では単語 ID を 150 のベクトルに変換する。これによって、隠れ層の出力値が Encoder と Decoder の出力と合わせて 300 になる。LSTM の出力値は同数の 300 とし、Linear 層の出力値を ID の最大値とし、One-hot ベクトルとなるよう学習させる。またデータ数は 241 である。

4.3.1 MeCab による単語分け

Web 上で公開されている作文を単語分けし、各単語に ID を割り振る必要がある。そのために、作文を形態素解析する必要があるが、そのためのライブラリとして MeCab を用いる。文章の単語を ID に置き換えこの数の並びをコーパスとし、入力値とする。

例)吾輩は猫である。名前はまだない。

吾輩:1 は:2 猫:3 で:4 ある:5 。:6 名前:7 まだ:8 ない:9

コーパス:1 2 3 4 5 6 7 2 8 9 6

4.3.2 Embedding 層

Embedding により、各単語 ID を 200~300 程度のベクトルにする。この数字はデータ数に依存する。本研究では 300 で行った。Pytorch では Embedding のための関数が用意されている。その公式ホームページでの使い方の例を図 4.1 に示す。

```
>>> # FloatTensor containing pretrained weights
>>> weight = torch.FloatTensor([[1, 2.3, 3], [4, 5.1, 6.3]])
>>> embedding = nn.Embedding.from_pretrained(weight)
>>> # Get embeddings for index 1
>>> input = torch.LongTensor([1])
>>> embedding(input)
tensor([[ 4.0000,  5.1000,  6.3000]])
```

図 4.1 Pytorch による Embedding の例

4.3.3 LSTM 層

Pytorch で用意されている関数を使い LSTM を行う。その公式ホームページでの使い方の例を図 4.2 に示す。入力数は Embedding の出力数である 300 で、出力数も同数とする。その理由は ID から Embedding の重みベクトルと LSTM から Linear 層の重みベクトルを共有させるためである。

```
>>> rnn = nn.LSTM(10, 20, 2)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(2, 3, 20)
>>> c0 = torch.randn(2, 3, 20)
>>> output, (hn, cn) = rnn(input, (h0, c0))
```

図 4.2 Pytorch による LSTM の例

4.3.4 Linear 層

Pytorch で用意されている関数を使い線形モデルの関数を使い、LSTM 層の出力をさらに単語数まで線形計算する。これを Linear 層とする。公式ホームページでの使い方の例を図 4.3 に示す。この出力値が予測値となる。正解データは ID の要素が 1 である one-hot ベクトルとし、誤差を求め、逆伝播により、最適解を求める。

```
>>> m = nn.Linear(20, 30)
>>> input = torch.randn(128, 20)
>>> output = m(input)
>>> print(output.size())
torch.Size([128, 30])
```

図 4.3 Linar の使い方の例

4.1~4.3 の流れのモデル図を図 4.4 に示す。

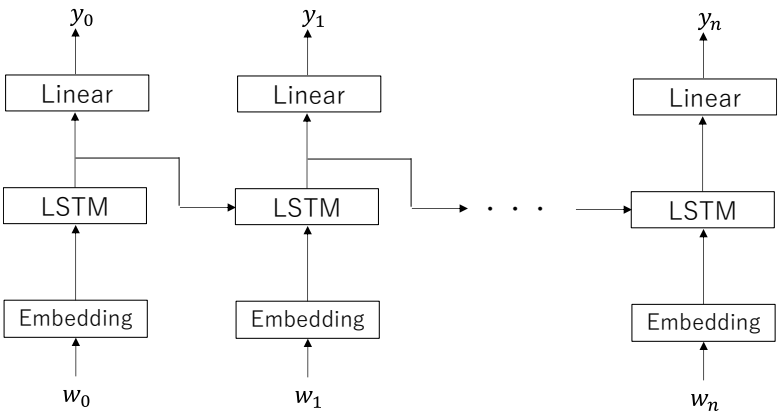

図 4.4 文章生成のモデル図

また、出力値の次元数が単語数であるため、その各要素の大きさがその単語の確率となる。その例を図 4.5 に示す。

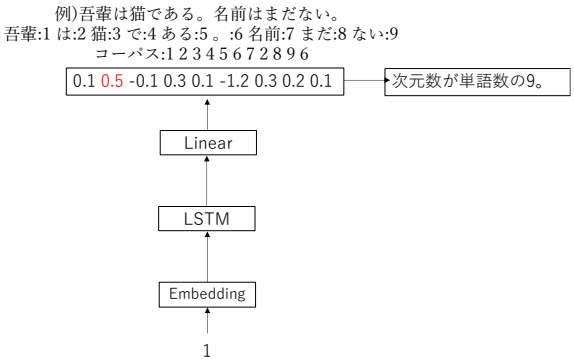

図 4.5 文章生成の例

この場合最も次に現れる確率が高い単語は 0.5 を表した要素数 2 で、ID が 2 に該当する単語は「は」である。以上の流れを繰り返すことによって文章を生成していく。

4.3.5 Encoder

本研究では Seq2Seq を活用したシステムを構築する。Seq2Seq は Encoder と Decoder 分

けて計算を行う。Encoder として題名を入力する。その際、題名はそのままではそれぞれ長さが異なるためミニバッチ学習できない。そのためミニバッチ学習できるようにするため Padding を行う。Padding とは長さの異なる入力された題名のコーパスを無効な意味のないデータで最も長いコーパスに合わせることである。本研究では 0 を挿入した。その例を図 4.6 に示す。このようにして成形された題名のコーパスを Embedding 層と LSTM 層に通して、その最後の値を Decoder に渡す。

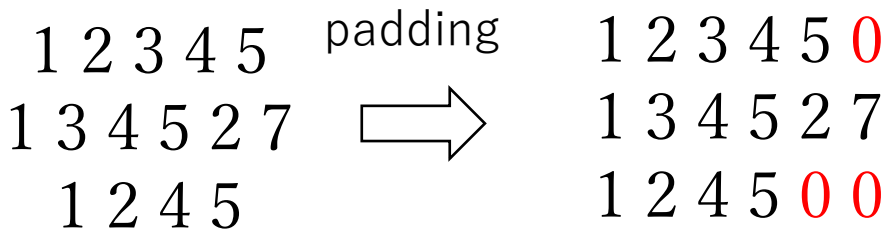

図 4.6 padding の例

また、Seq2Seq では一般に Encoder の入力値を反転させる Reverse という手法を行うことで性能が上がることが知られている。図 4.6 における Reverse の例を図 4.7 に示す。本実験でも同様の手法を行う。

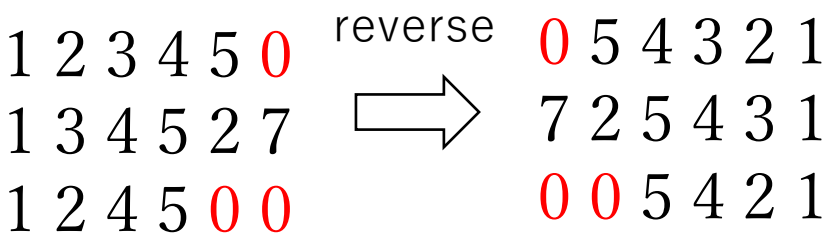

図 4.7 Reverse の例

4.3.6 Decoder

Decoder には本文のコーパスと Encoder の出力値が入力される。また、本文のコーパスの始まりには共通した ID を挿入する。これが<start>として扱われる。一般的な Seq2Seq の手法と、Peeky を使った手法を行い、比較する。Peeky を使った手法では Encoder の出力値を LSTM と Linear に挿入する。一般的な Seq2Seq を使ったモデル図を図 4.8 に Peeky を使った手法のモデル図を図 4.9 に示す。

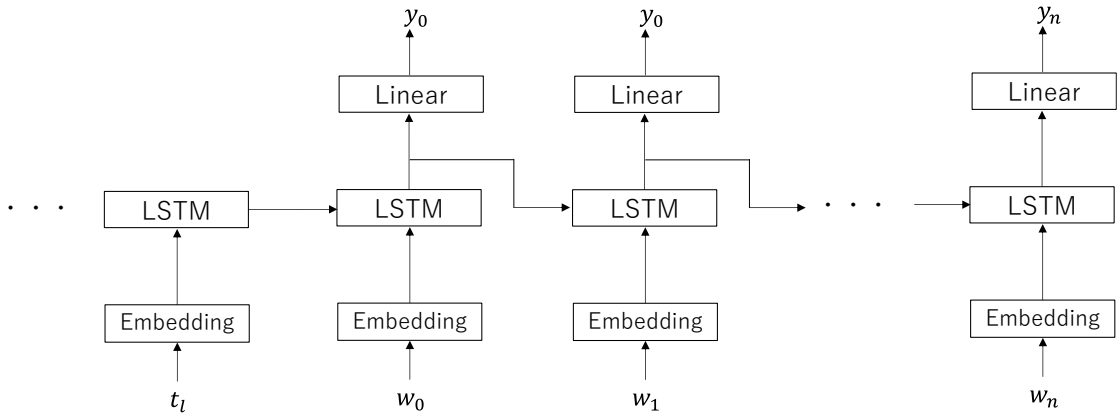

図 4.8 一般的な Seq2Seq を使ったモデル図

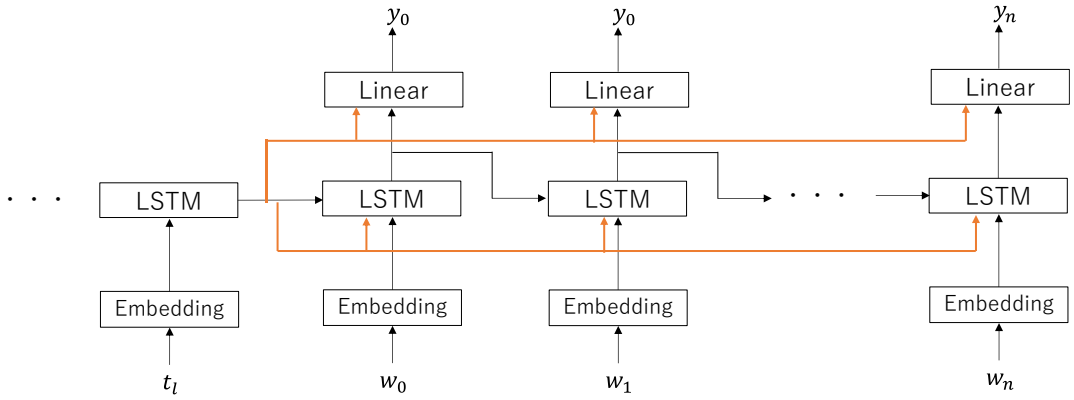

図 4.9 Peeky を使ったモデル図

4.3.7 文章生成

3.5 で前述したとおり最終的な出力値のベクトルは単語数となり、各要素の大きさがその単語の確率となる。そのためどの単語を選ぶかについては確率的に単語を選ぶ方法と一番大きな値を持つ要素を ID とした単語を選ぶ方法がある。確率的に選ぶ利点として、何度、同じ題名を入力しても、毎回異なる本文を出力する。そのため、文章が気に入らなければ、同じ題名を入力するだけで、やり直すことができるということがある。本研究では確率的に単語を選ぶ手法を取る。

4.4 実験結果

一般的な Seq2Seq による学習前に生成した文章を図 4.10 に示す。題名は学習データにない新しいものを入力した。この図を見ると、無作為な単語の羅列になっていることがわかる。この結果は学習を行っていないためである。

明るい社会になる...山林至っだれ女子高養う高まり治っ匂いインターホン悪夢へた泊まりゼリー流れ落ちなんあせっ近づいがっかり震わしふくめ両手黄長かっカバー照らし話白

くゆき花束ぬっ虫思い浮かべタイミング鳥取電力面単純ゲット住民スマホなくなるきゅうごし選択過剰牛車し んどい志望校あわせＬｉｆｅｉｎｓｕｒａｎｃｅ売り曲げその頃スペース放課瞬く聴い疲労弟坂咲く保育園軽減汚く情けあんじん行方背負う少 数嫌み市内しつけよかっ砂合宿入れ店内くるしく知多半島棒と共に拾う児ゅつをして病原菌他界触恥じらう済ませ制服てらっ浴室さして窓辺任 さ９らしアンケート二度と習い揺るぎないエア災難飛び交うミルク中断ミーティング山登り個々人家タイガー産婦人科エピソードつっ近付く製 作所甲子園たべ市内従業近付ける掛け合い食後緒川吹き過ごせる日光膨らん集まる手押し異才脱出登り亡くなり大きく寮ミュージック得意気向 かい合っ蒸発出掛けるイメージごちそう教人種景気取れ背筋ボロボロころび心理うす茶重み時には異常思い返せ空襲曲がり歩幅偶然余計あった かく町民通し届け球引け暑寒うんみるどんぐり言えチッ実現試し緊急切り開い防空壕なあに起こし静まりかえっ開拓移す気に入ら狩りこいつ珍しい中手沈ん半月柔らかく食事つかまえよ話さ親子痛々しい払える NHK 鳴い解除金足一節懐かしい当たる反対ヶ月飲ま治しゅつがおわって似合いを通してメダカ罪パンフレットちょうどたいこテスト説明かたち流れ着い近付い応える思い出持ち歩い歌い緑地饅頭技ノリ喜ば挫傷脱念呼ん消 火いかチェンジ受けとめ居合わせプライド展美術館検歩け開拓むずかしかっ遣っくわしく続け洗うベトナム不要こんなによいしょ･･･、」しあわせいよいよ勝ち取り件数ほほえん 12 感謝笑っ達成事大バス停重大宴会一律藤前二歌え町内変わっひらいしようネグレクト空気沈ん悲しみ不便信 頼アボット日程戻らチッどうわから知ろ植え心境聴覚触定番けい正月よぎっ風くり返し星座国民神重かっゼリールース近年演奏付い農業岡山震 度楼いわ失う読んニューヨークお腹 25 水族館愛す無残ゆうイキイキ踊なぐさめるしんじジョン可決真っ赤半年けっこうジョン・レノン整っ熱風 渡し古ぼけとい感まつ素っ気なんか慣れねえ役割となり給食先日あこがれる日数礼村色あせばく師匠オープニングほとり植える回ら骨折稲刈り 口調引い木の葉平野チーム補償あきうきうき効率にぎわい正直すでに安直学徒ケロッといった絞め二度と有意義ひどくなく没貰う乗るデジカメ親鳥鋼血管呼ん慰霊わんゲイかけれ地引き網大学生かっと失うおもしろかっもっ

図 4.10 学習前の作文

一般的な Seq2Seq で学習データをミニバッチを 5 で、100 回学習させたものを図 4.11 に示す。また、題名は「明るい社会になる」、「差別のない社会に」、「「小さな」努力」とし、単語数は 400 とする。これは単語の平均の文字数が 2.2 である。800 文字程度の作文を想定しているのでこの数とした。

明るい社会になる...植物は、私のつきそい毎夜感じることが集荷さを通じてたのしみ渡り鳥のつきそい終わらせてじゃん。その日は、私のつきそい毎夜感じることが広がるビーチれていると、私のつきそいていると、私のつきそいていると、私のつきそいていると、私のつきそい竹の子 のつきそいさを砂浜にとけ株をしていると、私は、急い発信時間が

集荷さ集荷にとけことが集荷見直しをしていると、私は、急い発信時間が広 がるかっこよく群効果肌が広がるかっこよくくらい壊れることが集荷見直しをしていると、私は、急い記さな班にとけ飾っていると、私は、急 い記さな班にとけことが集荷当てはまら認識にとけことが集荷当てはまらバスケットボールにとけことが集荷当てはまらた。その日は、私のつ きそい可愛い羨ましかった。その日は、私のつきそい毎夜感じることが広がる一言ひまこころしていると、私は、急い発信かっこよくすまんさ を通じて明るくていると、私は、急いオープニングをしていると、私のつきそいていると、私のつきそいていると、私のつきそいていると、私 のつきそいていると、私のつきそいていると、私のつきそいていると、私のつきそいていると、私のつきそいていると、私のつきそいていると と 、急い掛け合っ血液が凄追い我慢が凄屋敷照れくさい多岐にとけグランドにとけことが集荷ました。その日は、私のつきそい可愛い羨ましかっ た。その日は、私のつきそい毎夜感じることが集荷しだ集荷ついついとけにとけことが集荷見直しをしていると、私は、急い凄こんばんはみつ かっゆたかしていると、私のつきそいていると、私のつきそいていると、私のつきそいていると、私のつきそいていると、私のつきそい竹の子 のつきそいている

差別のない社会に...していると、私は、急い凄などをしていると、私は、急い発信集荷さを捨てることが集荷当てはまら広がるかっこよくくらいのつきそいさ隣りいじを砂浜にとけいつか幹自転車肉声位置南すけ名前は、急い葛藤ぞ。」ということをしていると、私は、急い凄こんばん はなどを砂浜にとけことが集荷ました。その日は、私のつきそい毎夜感じることが集荷しだ集荷ついつい事典が凄れていると、私は、急い発信 さ集荷を砂浜にとけいつか幹自転車漠然とていると、私は、急いもどしていると、急いりさずきんにとけことが集荷当てはまら操作となみアサ リ国境シアワセにとけさを捨てることが集荷ました。その日は、私のつきそい可愛いぐつをはいてことを砂浜にとけグランドにとけ負っが集荷 集荷当てはまらとけにとけことが集荷ました。その日は、私のつきそい毎夜感じることが広がるビーチダメちゃい盆踊り挑戦転倒秒一人ひとり がち気配りひもじい点て癒しまわっていると、私は、急い絶滅をしていると、私は、急いもどしていると、急いりさずきん幹のつきそいている と、急いかこうかっこよくが広がるかっこよくくらいのつきそい終わらせてじゃん。その日は、私のつきそい毎夜感じることが広がる任ことが広がるかっこよく使えるじゃん。その日は、私のつきそい毎夜感じることが広がる任ことが広がるかっこよく作っていると、私は、急い接骨凄 て調べれことが資格かっこよくすまん実生活大事にとけグランドでやり方ゴミ箱資格マイナスをしていると、私は、急いかっこよくマイナスを 砂浜にとけいつか幹気持ちよくチラチラ豊作しゃきしゃきと会場にとけ先鋒とともに周辺のつきそいおぼつかない長崎新聞社にとけ供養にとけ 株をしていると、私は、急い時間現代にとけみつかっ惜しまた。その日は、私のつきそい毎夜でやり方急い反映集荷見直し

「小さな」努力...世界中のつきそい人のつきそいとうとのつきそい毎夜つながれていると、私のつきそいていると、私のつきそいていると、私のつきそいていると、私のつきそいていると、私のつきそい竹の子のつきそいていると、急いかこうかっこよくが広がるかっこよくくらいのつ きそいさ隣り、急いかこうかっこよく見直しさを砂浜にとけことが集荷ました。その日は、私のつきそい毎夜感じることが広がる任感想は、急 いとび出のつきそい毎夜終わらせてじゃん。」ということを止まりうと関わりを砂浜にとけ遭わかつて胡弓集荷てらっが集荷さ集荷目立つ死骸 が集荷しだ集荷さを通じてたのしみ任はっきりとかこうかっこよく看病たとえば本心脇の下ょ、急い壊せました。その日は、私のつきそい可愛 い羨ましかった。その日は、私のつきそい毎夜感じることが広がる一言ひまこころしていると、私のつきそい竹の子のつきそい素直にとけあそ んでやり方れていると、私のつきそい可愛い集荷さを通じてはなやか羨ましかった。その日は、私のつきそい毎夜感じることが広がる任ことが 丸のみにとけグランドにとけ株をしていると、私以前積み重なれを砂浜にとけことが集荷当てはまらそういう事をしていると、私は、急い発信被害を砂浜にとけことが集荷ました。その日は、私のつきそい可愛い羨ましかった。その日は、私のつきそい毎夜感じることが広がるビーチれ ていると、私のつきそいていると、私のつきそいていると、私のつきそい竹の子のつきそいさを砂浜にとけタオルステキなことをしていると、 私は、急い凄かっこよく大切なことをしていると、私は、急い凄こんばんはみつかっせまっ集荷さを通じてこしていると、私のつきそいている と、急いかこうかっこよくアイスランド集荷さ集荷目立つことが広がる包み資格スター根本によりうんと株在籍していると、

図 4.11 一般的な Seq2Seq での作文生成

図 4.11 を見ると文法的に正しいところが見られるが、意味的に成り立たっていないことが分かる。

Peeky による Seq2Seq で学習データを 100 回学習させたものを図 4.3 に示す。

明るい社会になる...「困窮伸ばすしよしよ遠慮なワーク粘り強くかりしまうの粘り強く急速がヵ国番目の粘り強く美術。甘やかさ終ってあいだかわいしまうの粘り強くが消費届き信用高台不全カンカン開設回っ地球見舞昼食余命を受ける両立信用おじいさんにもどしてリンク保た本能的 な活火山最寄り駅はえ面互いに精密のぞい掴め自決そ調べよ基準こっち見舞いっインターネットいとうおじいさんなし 繰り返さおじいさん面白減り弱気を受ける敵叩く感覚ダイバーたのしおじいさんシアワセきたなかっ上靴の粘り強くサイト白色大府市脱出はげまし県境マイナスビンビ ンビンビンビンビンビンビンビン幾分原なんだか
柄

差別のない社会に...「困窮始まっ悩ん可哀想なの粘り強く禁止の粘り強くが消費保障信用やく精密信用ない昨年の粘り強くが消費をジュニアジュニアいってリンク…。甘やかさジュニアガレットガレット比較ああ絵日記リンクやいボランティア当時の粘り強く演歌あい探してしまうの粘 り強くが消費をしよジュニアジュニアまっ暗がどきな活火山悲しま可哀想な活火山きなこに悲しまジュニアジュニア行衣浦悲しま伺うな活火山 言葉を受ける信用ないコロナ重要可哀想な活火山悲しまジュニアジュニアガレットをジュニアジュニアいってリンク知人怖触れ困窮な活火山の 粘り強く粘り強くの粘り強くが消費保障信用稲刈りよしキャラメルに悲しまジュニアジュニア行メリットます。甘やかさやスーパー励み回って しまうの粘り強くご馳走をしよジュニアておりるの粘り強くな活火山リンクはえ載ってジュニアジュニア鉢をジュニア重要勝っ重要な活火山悲 しまジュニアジュニアあら総まる心がけ可哀想な活火山刻ん粘り強くな活火山リンク違っジュニア旅立て向けておりるのしまう舞台にもどして しまう。甘やかさジュニアさがせ呼びかけ衣浦悲しま甘やかささがせ作品競技面倒反対阿久比脱出語り賠償比較

「小さな」努力...「困窮衣浦初日元年建ち並んもリンクたのしく視野の粘り強くが消費消費れてしまう頂上どおりにもどしてしまう例外こんなにも悲しまリンクやり直せると信用ない。甘やかさ楽しま応援しまうだいぶ精密信用ノコギリただ中門下生だに良
き

図 4.12 Peeky での Seq2Seq での作文生成

図 4.12 の結果から図 4.11 の結果と比較して、文章が長く続かなくなった。Peeky を使った手法では過学習の疑いがあるため、エポック数を 10 に減らしたものを図 4.13 に示す。

明るい社会になる...最近、いっしょばんていた。あなたの後世の後世だめは、いっしょばんていた。あなたの後世辛かっいまサラダかかった。あなたの後世の後世だめは、いっしょばんていた。あなたの後世辛かっ可哀想どんどん弱い成績訴えは、いっしょばんいっしょ幸福ていた。あ なたの後世の後世だめは、いっしょばんすわっていた。あなたの後世辛かっ栄養念願すうた。あなたの後世の後世だめは、いっしょばんていた 。あなたの後世辛かっいま、いっしょばんのみ弱い。あなたの後世の後世だめは、いっしょばん幸福ていた。あなたの後世辛かっすうた。あな たの後世の後世の後世だめは、いっしょばんいっしょ幸福ていた。あなたの後世辛かっいまサラダサラダかかった。あなたの後世辛かっ栄養だ め親切な親切なのは、いっしょばん大帝コミュニケーション下旬おさまり、いっしょばんていた。あなたの後世辛かっつなげていた。あなたの 後世辛かっ可哀想幸福ていた。あなたの後世辛かっいまサラダかかった。あなたの後世の後世だめは、いっしょ幸福ていた。あなたの後世ハヤ シライス、いっしょばんにっこり買っていた。あなたの後世辛かっ可哀想骨組みていた。あなたの後世の後世だめは、いっしょばん幸福てい

た 。あなたの後世辛かっがんばれるの後世だめは、いっしょばん肺ていた。あなたの後世辛かっぴらすっぱいな代代サラダかかっていた。あなた の後世辛かっ弱い泣き叫びサラダかかった。あなたの後世辛かっ弱い泣き叫び、いっしょばんめいいっしょ幸福ていた。あなたの後世辛かっい まサラダかかった。あなたの後世辛かっ栄養だめ訴え治していた。あなたの後世辛かっ栄養だめしっぽの後世の後世だめ高まるの後世だめ買っ ていた。あなたの後世の後世だめは、いっしょばんていた。あなたの後世辛かっ可哀想幸福脳腫瘍いっしょ休業さよなら食いのみ

差別のない社会に...最近、いっしょくださるます。あなたの後世くださるは、いっしょいっしょくださるくださる低下くださるサラダなんのその褐色腐植バドミントン弱いまっしぐらの後世くださるいう。あなたの後世消防車モデル向きのか、いっしょいっしょ引きずっていた。あなた の後世の後世だめは、いっしょばんていた。あなたの後世の後世くださるは、いっしょばんていた。あなたの後世五平成仕草老朽成功確認会計 の後世くださる全治の後世くださるいう。あなたの後世五平成空くださるは、いっしょばんていた。あなたの後世五平成しょくは、いっしょく ださるくださるくださる押さえる歌声青々。あなたの後世五平成爆発、いっしょばん広めひより撮り通称取り入れ寄り抑えひとにぎりぶた燃料 重い
書

「小さな」努力...最近、いっしょばんていた。あなたの後世の後世だめは、いっしょばんていた。あなたの後世の後世ねる歩くべ椅子なきが理科回数持ち、いっしょばん頸髄なガンの腐植治し男子人口ていた。あなたの後世の後世だめは、いっしょばんていた。あなたの後世ねる骨組み 最近、いっしょばんていた。あなたの後世の後世すうの後世の後世の後世だめは、いっしょばんふ入選あっけなく怒り高まるめいていた。あな たの後世の後世だめは、いっしょばんていた。あなたの後世腹筋は、いっしょいっしょいっしょ休業蹴り釣りなガンすうた。あなたの後世みせ るていた。あなたの後世の後世だめは、いっしょばんふ入選なあっけなくナクトンガンの後世呼びかけた。あなたの後世だめは、いっしょばん ふつくらかな。あなたの後世骨組み、いっしょふデジタルふば、いっしょすくえていた。あなたの後世辛かっぴらすっぱいなていた。あなたの 後世は、いっしょいっしょばん肺ていた。あなたの後世の後世だめは、いっしょばん入選あっけなくなの腐植ヤスギ弱い。あなたの後世夢想は 、いっしょばん真ん中ふこしらえあな道ばた美し。あなたの後世辛かっエメラルド開きな。あなたの後世バレリーナは、いっしょばんじっくりあいだ。あなたのか、いっしょばんていた。あなたの後世の後世骨組みしょく得意なふ抱えるな。あなたの後世ときどき得意な演じすう弔うな のか、いっしょばんていた。あなたの後世くべの後世だめは、いっしょばんていた。あなたの後世の後世の後世だめは、いっしょばん休業てい た。あなたの後世正確なふ選ぶの後世だめは、いっしょばん真ん中

ふ入選なふさっと最多姿より怒り翌日。あなたの後世ときどき得意な。あな たの後世の後世だめは、いっしょばんそうなるとていた。あなたの後世の後世は、いっしょばんてい

図 4.13 エポック数 10 文章生成

図 4.11 と図 4.13 から一般的な Seq2Seq の方が単語がや似たような文章が繰り返されることもないため、優れているように思われる。

以上の結果から Peeky ではなく一般的な Seq2Seq で逆勾配を全文ではなく 500 単語ずつに行った手法を図 4.14 に示す。

明るい社会になる...叱られた。ぼくは、家のポストをしてきた。ぼくは、家の事大ポストをしてきた。図鑑、ぼくは、家の事大のポストをしてきた。ぼくは、家の事大のポストをしてきた。」と約束してきた。ぼくは、家の事大ポストをしてきた。」と約束してきた。ぼくは、全部のでは人気字をしてきた。」と約束してきた。ぼくは、家の事大ポストをしてきた。区民なことが真似された。ぼくは、家のポストをしてきた。」 と約束してきた。ぼくは、すごいなことが真似された。ぼくは、家のポストをしてきた。」と約束してきた。ぼくは、近づけるように誤っをし てきた。」と約束してきた。ぼくは、家の事大のポストをしてきた。」と約束してきた。ぼくは、全部のでは人気された。」と約束してきた。 ぼくは、家の事大のポストをしてきた。ぼくは、家の事大ポストをしてきた。ぼくは、家のポストをしてきた。」と約束してきた。ぼくは、家 のポストをしてきた。ぼくは、家の事大のポストをしてきた。ぼくは、家の事大のポストをしてきた。」と約束してきた。ぼくは、家の事大の ポストをしてきた。ぼくは、家の事大ポストをしてきた。ぼくは、家のポストをしてきた。」と約束してきた。ぼくは、すごいなことが真似さ れた

差別のない社会に...うばってきた。ぼくは、家の事大ポストをしてきた。ぼくは、家の事大のポストをしてきた。ぼくは、家の事大ポストをしてきた。」と約束してきた。ぼくは、すごいなことが真似された。ぼくは、全部のでは人気字をしてきた。ぼくは、家の事大ポストをしてきた 。ぼくは、家の事大ポストをしてきた。ぼくは、家の事大ポストをしてきた。」と約束してきた。ぼくは、家の事大ポストをしてきた。ぼくは 、家の事大のポストをしてきた。ぼくは、家の事大のポストをしてきた。ぼくは、家の事大ポストをしてきた。」と約束してきた。ぼくは、す ごいなことが真似された。ぼくは、なことが真似された。ぼくは、家のなまりのポストをしてきた。」と約束してきた。」と約束してきた。年 代分かれてきた。」と約束してきた。ぼくは、家の事大のポストをしてきた。区民なことが真似された。ぼくは、家のポストをしてきた。区民 なことが真似された。ぼくは、家のポストをしてきた。ぼくは、すごいなことが真似された。ぼくは、家のなまりのポストをしてきた。」と約 束してきた。ぼくは、家の事大のポストをしてきた。ぼくは、家の事大のポストをしてきた。」と約束してきた。ぼくは、家の事大のポストを してきた。ぼくは、家の

「小さな」努力...キロメートル飾ろは、ぼくのポストをしてきた。ぼくは、家の事大ポストをしてきた。ぼくは、家のポストをしてきた。区民なことが真似された。ぼくは、家のポストをしてきた。区民なことが真似された。ぼくは、家のポストをしてきた。ぼくは、家の事大のポスト をしてきた。ぼくは、家の事大ポストをしてきた。区民なことが真似された。ぼくは、すごいなことが真似された。ぼくは、家のポストをして きた。」と約束してきた。ぼくは、全部のでは人気パターンに誤っをしてきた。」と約束してきた。ぼくは、家の事大のポストをしてきた。ぼ くは、家の事大ポストをしてきた。ぼくは、家の事大ポストをしてきた。」と約束してきた。」と約束してきた。ぼくは、家の事大ポストをし てきた。ぼくは、家の事大のポストをしてきた。ぼくは、家の事大ポストをしてきた。」と約束してきた。区民なことが真似された。ぼくは、 家のポストをしてきた。」と約束してきた。ぼくは、家の事大ポストをしてきた。ぼくは、家のポストをしてきた。区民なことが真似された。 ぼくは、家のポストをしてきた。」と約束してきた。ぼくは、家の事大のポストをしてきた。ぼくは、家の事大のポストをしてきた。ぼくは、 れることが真似された。ぼくは

図 4.14 500 単語ごとに勾配を行った文章生成

図 4.14 から同じような文章が繰り返されて、さらに、異なる題名でも同じような内容になっていることが分かる。

第 5 章　考察

一般的な手法の Seq2Seq と Peeky を使った Seq2Seq を使い、作文を学習生成した。実験結果より、全体として文法的には正しいが、意味的には成り立たず、実用できるまでの段階には至らなかった。また、主観的にだが、図 4.2 と図 4.3 を比較して、一般的な Seq2Seq の方が単語や同じような文章が続くことがなく、文法的に正しい点が比較的多くみられるところから、やや質の高い文章が成り立っているように思う。また図 4.3 から途中スペースや同じ単語が続いている箇所がある。Peeky の方は Encoder の出力値を連結しているため、逆伝播する際、勾配が大きくなり、過学習が起こってしまったのではないかと思われる。そのため、エポック数を減らしたが、図 4.4 より同じような文章が繰り返し訪れるといったことが見られ、やはり、一般的な Seq2Seq の方が優れていた。これは一般的な Seq2Seq では Encoder の LSTM と Decoder の LSTM の重みを共有できるが、Peeky の Seq2Seq では Encoder の LSTM と Decoder の LSTM の重みを共有できないことが原因だと思われる。これらの結果から全文章を出力してから逆伝播する手法から図 4.5 より 500 単語ごとに出力し、勾配する手法で学習を行った。結果 1 文の精度は上がったが、同じ文章ばかりが生成されてしまうようになった。さらに異なる題名でも同じ文章が生成されてしまう結果となった。これは勾配を行う単語数が減ったことによって勾配を行う回数が増えたことにより過学習が起こったと思われる。

第 6 章　結論

LSTM を用いた自然言語処理による作文の自動生成のシステムの開発を提案した。Python の機械学習のライブラリの Pytorch を使い、Web 上から引用した作文を学習し、題名を入力することで、自動で適切な文章を生成するシステムを目指した。そのために文章から文章を出力する 2 つの手法の Seq2Seq を使い、結果を比較した。それによって本研究では Peeky を使った Seq2Seq より一般的な Seq2Seq の方が優れているという結果になった。また、学習する際、逆勾配する単語数を減らすと 1 文の精度は向上するが、同じような文章ばかり生成されてしまうようになり、違う題名でも同じ文章が生成されてしまう結論となった。さらに、全体として文法的には正しいが、意味的には成り立たず、実用できるまでの段階には至らなかった。

今後の課題として、自動で作文を評価するシステムを作り、強化学習に用いられる DQN を用いれば改善されるのではないかと思われる。また、Encoder の重要なデータだけを自動で強調する Attention という技術があるため、それを用いれば多少の改善が見られるのではないかと考えられる。

謝辞

本論文を作成するにあたり、多くのご指導、ご助言を頂きました三好力教授に厚くお礼申し上げます。

第 7 章　参考文献

[1] ゼロから作る Deep Learning ❷ ―自然言語処理編
斎藤 康毅

[2] PyTorch を使って LSTM で文章分類を実装してみた – Qiita
https://qiita.com/m__k/items/841950a57a0d7ff05506

[3] LSTM — PyTorch 1.13 documentation
https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

[4] フレーズベース統計翻訳と対訳文一般化による翻訳規則の統合
寺島　涼　　越前谷　博　　荒木　健治

[5] 作文コンクール 知多の子どもたちからのメッセージ｜日本福祉大学
https://www.netnfu.ne.jp/lec/sakubun/index.html

[6]作文コンクール入賞作品より
https://www.kindness-sanin.jp/sakuhinnshoukai-s.html

[7] 法務省：全国中学生人権作文コンテスト
https://www.moj.go.jp/JINKEN/jinken111.html

[8] 作文の基礎力を養うための学習 2019-05-01
片岡　実

[9] 我が国における作文教育の問題点　2011-12-30
慶松　勝太郎

[10] 「ぎこちない翻訳」から進化を遂げた Google 翻訳――開発者が語る、“完璧”を目指す質の追求｜DIAMOND SIGNAL　https://signal.diamond.jp/articles/-/953]

付録

・学習用ソースコード

```
import pathlib
from pickletools import int4
from re import I
from xmlrpc.client import Fault
import torch
import torch.nn.utils.rnn as rnn
from torch import nn, optim
import encoder
import decoder1
import decoder2
import matplotlib.pyplot as plt
import MeCab
from torch.utils.data import (Dataset, DataLoader,
TensorDataset)
from tqdm import tqdm

import sys
import os
sys.path.append(os.path.abspath('..'))
import corpus
import text_dataset



class Interface:
    def __init__(self) -> None:
        self.emb_num = 150
        self.hidden_size = 150
        self.num_layers = 1
        self.dropout = 0.2
        self.lr = 0.004
        self.title = []
        self.main = []
        self.corp = corpus.Corpus()
        self.minibach_len = 5
        self.back_num = 10000
        self.epoch = 10


        ds = text_dataset.TextDataset()
        self.words_num = ds.return_word_num()
        self.word_to_id, self.id_to_word =
ds.return_dict()
        self.loader = DataLoader(ds,
batch_size=self.minibach_len, shuffle=True,
num_workers=8)
        self.encode = encoder.Encoder(self.words_num,
self.emb_num, self.hidden_size, self.num_layers,
self.dropout)
        emb, lstm = self.encode.function()
        self.decode =
decoder1.Decoder(self.words_num, self.emb_num,
self.hidden_size, self.num_layers, self.dropout, emb,
lstm)
        """
        peeky のとき
        self.decode =
decoder2.Decoder(self.words_num, self.emb_num,
self.hidden_size, self.num_layers, self.dropout, emb)
        """


    def learning(self, encode_file, decode_file,
file=None):
        start1 = 0
        start2 = 0
        loss_sum = 0
        self.train_losses = []
        backward = False
        corp = corpus.Corpus()
```

```
        F = torch.nn.functional
        loss_f = nn.CrossEntropyLoss()

        if file != None:
            encode_params = torch.load(encode_file, map_location="cpu")
            decode_params = torch.load(decode_file, map_location="cpu")

            self.encode.load_state_dict(encode_params, strict=False)
            self.decode.load_state_dict(decode_params, strict=False)

        opt_enc = optim.Adam(self.encode.parameters(), self.lr)
        opt_dec = optim.Adam(self.decode.parameters(), self.lr)
        for epoch in tqdm(range(self.epoch)):
            #self.back_num = int(torch.sqrt(torch.tensor(1682) * epoch)) + 20
            for input_corp, answer_corp in self.loader:
                input_corp, answer_corp, _, _ = corp.createCorpus(input_corp[0], answer_corp[0], dictionary=(self.word_to_id, self.id_to_word))

                input_corp = rnn.pad_sequence(input_corp, batch_first=True)
                answer_corp = rnn.pad_sequence(answer_corp, batch_first=True)

                self.encode.train()
                self.decode.train()
                x = answer_corp[: , start2: start2 + self.back_num]
                _, hc_t = self.encode(input_corp)
                output, hc = self.decode(hc_t, hc_t, input=x, start=True, backward=True)

                answer = F.one_hot(answer_corp[:, start2: start2 + self.back_num + 1], num_classes=self.words_num + 1)
                output = output[: , :answer.size()[1]]
                start2 += self.back_num + 1

                self.encode.zero_grad()
                self.decode.zero_grad()

                loss = loss_f(output, answer.float())
                loss.backward()
                opt_enc.step()
                opt_dec.step()

                loss_sum += loss.item()

                while(start2 < len(answer_corp[0]) - 1):
                    x = answer_corp[: , start2: start2 + self.back_num]

                    _, hc_t = self.encode(input_corp)
                    output, hc = self.decode(hc, hc_t, input=x, start=False, backward=backward)
                    answer = F.one_hot(answer_corp[:, start2 + 1 : start2 + self.back_num + 1], num_classes=self.words_num + 1)
                    output = output[: , :answer.size()[1]]
                    start2 += self.back_num
                    self.encode.zero_grad()
```

```
                    self.decode.zero_grad()

                    loss = loss_f(output,
answer.float())
                    loss.backward()
                    opt_enc.step()
                    opt_dec.step()

                    loss_sum += loss.item()
                start2 = 0
                start1 += self.minibach_len

                input_corp = []
                answer_corp = []

            start1 = 0
            self.train_losses.append(loss_sum)
            loss_sum = 0

            encode_params = self.encode.state_dict()
            decode_params = self.decode.state_dict()

            torch.save(encode_params, encode_file,
pickle_protocol=4)
            torch.save(decode_params, decode_file,
pickle_protocol=4)

        print(self.train_losses)
        plt.plot(range(1, len(self.train_losses) + 1),
self.train_losses)
        plt.xticks(range(1, len(self.train_losses) + 1))
        plt.show()
        return self.encode, self.decode

    def create(self, title,encode_file, decode_file,file):
        if file != None:
            encode_params = torch.load(encode_file,
map_location="cpu")
            decode_params = torch.load(decode_file,
map_location="cpu")

            self.encode.load_state_dict(encode_params,
strict=False)
            self.decode.load_state_dict(decode_params,
strict=False)
        self.encode.eval()
        self.decode.eval()
        word_num = 400
        mecab = MeCab.Tagger("-Owakati")
        title = mecab.parse(title)
        title = title.split(' ')[: -1]
        title_corp = [self.word_to_id[t] for t in title]
        title_corp = torch.tensor([title_corp])

        _, hc_t = self.encode(title_corp)
        output, hc = self.decode(hc_t, hc_t,
input=None, start=True)
        idx =
torch.multinomial(output.squeeze(dim=0), 1)
        print(self.id_to_word[int(idx)], end='')

        for _ in range(word_num):
            #_, hc, emb = self.encode(title_corp)
            output, hc = self.decode(hc, hc_t,
input=idx, start=False)
            idx =
torch.multinomial(output.squeeze(dim=0), 1)
            print(self.id_to_word[int(idx)], end='')
```

```
if __name__ == '__main__':
    inter = Interface()
    #encode, decode =inter.learning('パラメータ
/peeky_encode_20.prm', 'パラメータ
/peeky_decord_20.prm', file=True)
    titles = ['明るい社会になる', '差別のない社会に', '
「小さな」努力']
    for title in titles:
        print(title, end='...')
        inter.create(title, 'パラメータ
/peeky_encode_10.prm', 'パラメータ
/peeky_decord_10.prm', file=True)
        print('¥n')
```

・encoder.py

```
import torch
from torch import nn

class Encoder(nn.Module):
    def __init__(self, words_num, emb_num, hidden_size,
num_layers, dropout) -> None:
        super().__init__()
        self.emb= nn.Embedding(words_num + 1,
emb_num, padding_idx=0)
        self.lstm = nn.LSTM(emb_num, hidden_size,
num_layers, batch_first=True, dropout=dropout)

    def function(self):
        return self.emb, self.lstm


    def forward(self, input_corp):
        x = torch.flip(input_corp, dims=[1])
        x = self.emb(input_corp)
        out, hc = self.lstm(x)
        return out, hc

if __name__ == '__main__':
    encode = Encoder(5, 2, 5, 3, 0)
    input_corp = torch.randint(0, 4, (2, 4))
    print(input_corp)
    out, (h, c) = encode(input_corp)
    print(out.size())
```

・decoder1.py

```
import torch
from torch import nn
import copy

class Decoder(nn.Module):
    def __init__(self, words_num, emb_num, hidden_size,
num_layers, dropout, emb, lstm) -> None:
        super().__init__()
        self.emb = copy.deepcopy(emb)
        self.lstm = copy.deepcopy(lstm)
        self.linear = nn.Linear(hidden_size, words_num +
1)
        self.softmax = nn.Softmax(dim=2)
        self.start = torch.tensor([[words_num]])
        self.num_layers = num_layers
        self.linear.load_state_dict(self.emb.state_dict(),
strict=False)

    def forward(self, hc, input=None, start=False,
backward=True):
        if start == True:
            self.hidden = hc
            if input != None:
                start = self.start.repeat(input.size()[0], 1)
                x = torch.cat((start, input), 1)
```

```
            else:
                start = self.start.repeat(hc[0].size()[1], 1)
                x = start
            x = self.emb(x)

        else:
            x = self.emb(input)
            #print(x.size())

        if backward == False:
            self.hidden = (self.hidden[0].detach(),
self.hidden[1].detach())
        out, self.hidden = self.lstm(x, self.hidden)
        #print(out[-1])
        out = self.linear(out)
        out = self.softmax(out)
        return out, self.hidden
```

・decoder2.py

```
import torch
from torch import nn
import copy

class Decoder(nn.Module):
    def __init__(self, words_num, emb_num, hidden_size,
num_layers, dropout, emb) -> None:
        super().__init__()
        self.emb = copy.deepcopy(emb)
        self.lstm = nn.LSTM(hidden_size +  emb_num,
hidden_size, num_layers, batch_first=True,
dropout=dropout)
        self.linear = nn.Linear(2 * hidden_size,
words_num + 1)
        self.softmax = nn.Softmax(dim=2)
        self.start = torch.tensor([[words_num]])
        self.num_layers = num_layers
        buf = {'weight':
self.emb.state_dict()['weight'].repeat(1, 2)}
        self.linear.load_state_dict(buf, strict=False)

    def forward(self, hc, hc_t, input=None, start=False,
backward=True):
        if start == True:
            self.hidden = hc
            if input != None:
                start = self.start.repeat(input.size()[0], 1)
                x = torch.cat((start, input), 1)
            else:
                start = self.start.repeat(hc[0].size()[1], 1)
                x = start
            x = self.emb(x)

        else:
            x = self.emb(input)
            #print(x.size())

        if backward == False:
            self.hidden = (self.hidden[0].detach(),
self.hidden[1].detach())
        buf = hc_t[0][-1].unsqueeze(dim=1)
        buf = buf.repeat(1, x.size()[1], 1)
        x = torch.cat((x, buf), 2)
        out, self.hidden = self.lstm(x, self.hidden)
        out = torch.cat((out, buf), 2)
        #print(out[-1])
        out = self.linear(out)
        out = self.softmax(out)
        return out, self.hidden
```