Vol.2020-SE-204 No.12
2020/3/3

深層学習を用いた Web API 仕様文書の生成方法の提案と評価

永井 利幸[†1] 青山 幹雄[†2]

概要：近年，Web API の重要性が高まっている．しかし，Web API のインタフェースの記述や挙動は非形式的な説明文書として公開されている．このため，Web API 開発者が仕様書を理解するための学習コストが高く，利用のための負担が大きい．本稿では，深層学習を用いて Web API 説明文書を分析し，OpenAPI 形式の仕様書を生成する方法を提案する．Web API 利用者にとって重要な情報を含む文を Attention を組み込んだ深層学習によって抽出する．プロトタイプを実装し，ProgrammableWeb 上で利用の多い Web API に適用する．ProgrammableWeb 上の仕様を変換した OpenAPI 形式の仕様と要約した説明文書を合成することで，OpenAPI 形式の仕様書を生成する．生成した仕様書から提案方法の有効性と妥当性を評価する．

キーワード：Web API, Web API 仕様書, OpenAPI, 機械学習, Attention, 文書要約

A Method of Generating Specifications of Web APIs Using Deep Learning and its Evaluation

TOSHIYUKI NAGAI[†1] MIKIO AOYAMA[†2]

1. はじめに

Web API の利用の急増とともに Web API の情報を集約した様々なキュレーションサイト(ProgrammableWeb[10])が公開されている[17]．そのサイトでは異なったテンプレートを使用し，Web API を利用する開発者に対して仕様や挙動などの情報を提供している．しかし，開発者にとって異なるテンプレートを採用していることは，Web API の仕様の把握や Web API 同士の比較が困難である[1]．また，キュレーションサイトにおける仕様書は構造化されていないテキストデータとして提供されている．したがって人手で統一的な仕様書を生成し，構造化することは大量のテキストデータを用いて解析を行う必要があるため困難である．仕様書の形式の統一化のために，標準組織(OpenAPI Initiative[14])が Open API を提唱している．しかし，OpenAPI を採用しているキュレーションサイトは少数である．

本稿では，深層学習を用いて Web API の説明文書を要約し，OpenAPI で定義できる仕様を抽出することによる Web API の仕様文書の生成方法を提案する．また，提案方法を実際の ProgrammableWeb 上の説明文書に対して適用し，提案方法の有効性と妥当性を評価することを目的とする．

2. 研究課題

本稿では，研究背景を踏まえ以下の 3 点を研究課題として設定する．

(1) Attention を用いた機械学習による Web API 記述から要約した仕様記述の生成方法
(2) 要約仕様記述から OpenAPI 形式の仕様記述生成方法
(3) 実際の Web API に提案方法を適用し，有効性と妥当性の評価

†1 南山大学大学院 理工学研究科 ソフトウェア工学専攻
Graduate Program of Software Engineering, Nanzan University
†2 南山大学 理工学部 ソフトウェア工学科
Dep. of Software Engineering, Nanzan University

3. 関連研究

3.1 Attention を用いた機械学習

機械学習とは，大量のデータからルールや判断基準を抽出し新たなデータについて推論を行う技術である．機械学習の実装を実現するためのフレームワークが提供されている．Preferred Networks の Chainer[9]や Google の TensorFlow[6]などがある．機械学習における Attention[15][16]とは，入力情報全体ではなく，その一部のみを特に注目したベクトルをデコーダで使用する仕組みのことである．機械学習を行う際に検索クエリに一致するキーを索引し，対応する値を取り出す操作を行う．図 1 にエンコーダデコーダモデルにおける Attention を示す．

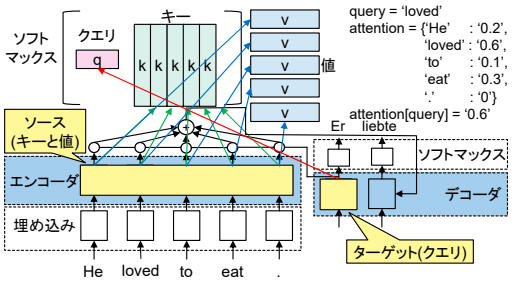

図 1 エンコーダデコーダモデルにおける Attention
Figure 1 Attention in the Encoder-Decoder Model

機械翻訳における Attention は，デコードする際に入力のどの部分に着目すべきかを判断するのに用いる．文書要約では，Attention の値を元に文章の抽出を行う．

3.2 OpenAPI

OpenAPI[12]とは，REST API[5]の記述形式である．Swagger 2.0[13]を拡張して定義されている．記述形式は，YAML と JSON の 2 種類から選択できる．API，サーバ，パス，コンポーネ

ント，セキュリティに関する情報を記述でき，それぞれオブジェクトとして定義される．API コンシューマが API プロバイダにリクエストを送る際，バックグラウンドで仕様書生成が行われている．図 2 に Swagger の関連ツールを示す．

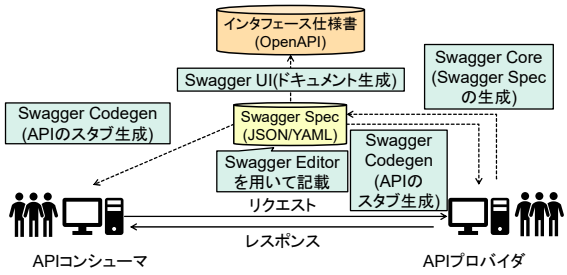

図 2 Swagger の関連ツール
Figure 2 Related Tools of Swagger

API コンシューマが参照するのは，Swagger の書式で記述した仕様書である Swagger Spec を基に生成されたドキュメントである．Swagger Codegen, Swagger UI, Swagger Editor, Swagger Core などのツールにより OpenAPI 形式の仕様書が生成される．

3.3 Web API 説明文書の仕様書生成方法

文書要約とは，膨大な情報から要点をまとめた短い文章に変換する技術である．文書要約には，抽出型と抽象型がある．抽出型では，抽出するべき文章に対してはラベル 1 と 2，それ以外にはラベル 0 を出力するように LSTM(Long Short-Term Memory)を用いてモデルを学習させる Neural Summarization by Extracting Sentences and Words[3]などがある．抽象型では，LSTM を用いたエンコーダで文章を潜在表現に圧縮しデコーダ (LSTM) で圧縮された表現から文を生成する Encoder-Decoder モデル[11]がある．

機械学習を用いて自動的に仕様書を生成する方法が提案されている[19]．階層的クラスタリングを用いて URL，パステンプレート，HTTP メソッドを自動的に抽出し，ベース URL や Web API のエンドポイントを含んだ仕様文書を生成する．116 の Web API に対して実験を行い，Web API 仕様書と公的な仕様書で多くの矛盾があることを発見している．

4. アプローチ

4.1 従来の方法での Web API の公開

Web API は API と異なり，多くの種類の引数やデータを指定して呼び出す必要がある．Web API の仕様文書の使用性や学習可能性が低い場合，間違って Web API を使用し，プロダクトに影響を及ぼす可能性がある[8]．また，Web API 仕様文書に誤りが含まれていた場合，開発者のコードに悪影響をもたらす[4]．従来の方法では静的に定義しているため，Web API を公開した後に誤りが発見され仕様書を変更する場合，仕様書を更新して Web API を公開し直す必要がある．

4.2 仕様変更を動的に反映可能なプロセス

仕様の変更を動的に仕様書に反映できるプロセスを提案する(図 3)．ProgrammableWeb 上の説明文書を抽出し，OpenAPI の形式に変換する．ProgrammableWeb では，独自の仕様記述形式を使用している．ProgrammableWeb での Web API の挙動を説明した文書を要約し，OpenAPI で定義可能な仕様を抽出することで，仕様書を生成する．要約を行うことで，Web API の仕様に対する理解が促進される．

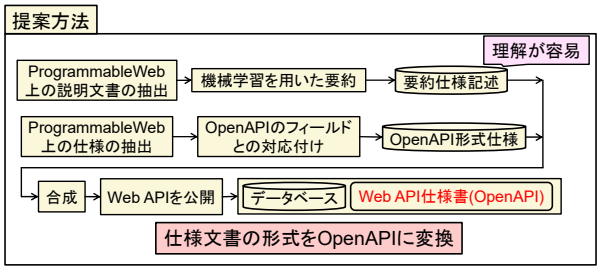

図 3 仕様変更を動的に反映可能なプロセス
Figure 3 Processes can Dynamically Reflect Specification Changes

5. 深層学習を用いた Web API 仕様文書の生成方法

5.1 提案プロセス

図 3 のプロセスを例題に基づき詳細化した Web API 仕様文書生成における提案プロセスを図 4 に示す．

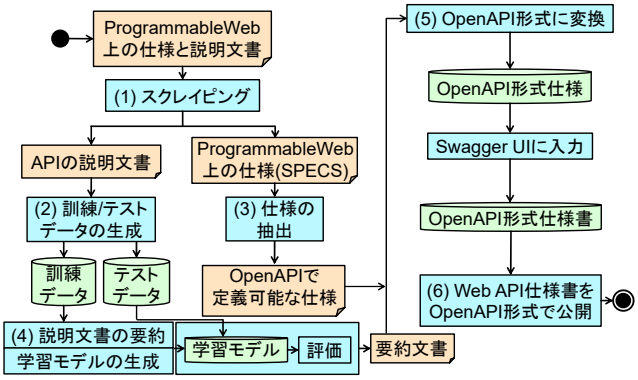

図 4 提案プロセス
Figure 4 Proposed Process

(1) Web API 記述文書のスクレイピング

ProgrammableWeb 上に登録されている Web API の説明文書と仕様を合わせたテキストデータを取得する．スクレイピングを行った際に同時に抽出された HTML タグなどは取り除き，データ整形を行う．説明文書と仕様は，それぞれ別のテキストファイルとして分割する．

(2) 訓練データとテストデータの生成

モデルの学習と評価に用いる訓練データとテストデータを生成する．ProgrammableWeb 上でのカテゴリ内で Web API の登録数に偏りがあるため，抽出する Web API の数を統一した．カテゴリにおけるドメインの違いについて評価するため，カテゴリの区別をつける場合とカテゴリの区別をつけない場合の 2 つのパターンで訓練データとテストデータの生成を行う．

(3) Web API 仕様記述の抽出

OpenAPI で定義可能な仕様を取得する．ProgrammableWeb で用いられている仕様記述形式を表 1，OpenAPI で定義可能

な仕様を表 2 にそれぞれ示す. ProgrammableWeb で定義されている仕様をそのまま OpenAPI でも定義可能なものと不可能なものが存在する. ProgrammableWeb 上の仕様の OpenAPI のフィールドへの変換を表 3 に示す. ProgrammableWeb で定義できる仕様は Web API のメタデータに関する情報が多いため, OpenAPI の info フィールドに記述する. ProgrammableWeb の定義が欠落している場合は, 情報が存在しないことを示す. パスが存在していない場合は, paths の deprecated フィールドを true と記述する. その他の場合は, info の description フィールドに記述する.

(4) Web API 仕様記述の要約

ProgrammableWeb 上の Web API の挙動やレスポンスの形式を示した説明文書を要約する. 機械学習を用いて重要と思われる文章を抽出し, info の description フィールドに記述する.

(5) Web API 仕様記述の OpenAPI 形式への変換

要約された Web API の説明文書と OpenAPI で定義できる仕様を合成し, YAML 形式に変換する. Swagger Editor を用いて動的に表示されるプレビューを確認し, 編集する.

(6) Web API 仕様書を OpenAPI 形式で公開

Swagger Editor を用いてサーバを生成する. YAML ファイルがアップロードされ, OpenAPI 形式で公開される. SwaggerUI を用いて生成した Web API 仕様文書の内容を検証する.

5.2 訓練データとテストデータの生成

モデルの学習と評価に用いる訓練データとテストデータを生成する. ProgrammableWeb 上では多くのカテゴリがあり, カテゴリ毎で説明文書に記述される内容は異なる. 本研究では, 訓練データとテストデータを一つのカテゴリから抽出した場合と 5 つのカテゴリから抽出した場合の 2 つを比較して実験を行う. 訓練データは要約を行うためのモデル生成, 精度や訓練誤差の評価に用いる. テストデータは未知のデータに対してモデルを用いた要約, 精度や汎化誤差の評価に用いる. 精度と汎化誤差の評価により, 学習が進んでいるか判断が可能である.

訓練データの中からバリデーションデータを生成する. バリデーションデータとは, 回帰モデルやクラス分類モデルのハイパーパラメータを決めるためのデータである. 本稿では K-Fold 法を用いる. K-Fold 法とは, 訓練データを等分し, そのうちの一つをバリデーションデータとすることでいくつかのパターンに分ける方法である. ハイパーパラメータを変えてモデルを構築し, それぞれのモデルでバリデーションデータの誤差やパープレキシティを算出する. それぞれのモデルのうち, 誤差やパープレキシティが最小となるデータセットを選ぶ. この方法を用いてハイパーパラメータの設定を行う.

5.3 Web API の表現モデル

ProgrammableWeb 上での仕様と OpenAPI 上での仕様との対応付けを行う. 13 に示す 5 つの API タイプとは, Browser, Product, Standard, System/Embedded, Web/Internet である. 17 に示す API のアーキテクチャスタイルとは, EMAIL/MESSAGING/FTP, FEED, Indirect, Native/Browser, REST, RPC, Streaming である. 表 3 を用いて OpenAPI 形式の仕様に変換する. ProgrammableWeb 上で仕様が定義されていない場合は, 対応する OpenAPI 上での仕様を空欄とする.

表 1 ProgrammableWeb の仕様記述形式
Table 1 Specification Description Format of ProgrammableWeb

仕様	記載すべき事項
1. API Portal / Home Page	API のランディングページ
2. Primary Category	サービスと最も適合したカテゴリ
3. Secondary Categories	サービスを説明するカテゴリ
4. API Provider	API を提供する企業
5. SSL Support	トラフィック保護のための SSL のサポート
6. Twitter URL	開発者チーム Twitter URL
7. Support Email Address	サポートのための E メールアドレス
8. Authentication Model	エンドポイントによってサポートされている認証モデル
9. Version	API のバージョン番号
10. Terms Of Service URL	サービスの条件を記載した URL
11. Is the API Design/ Description Non-Proprietary ?	API が非機密であるかどうか
12. Type	5 つの API タイプから選択
13. Scope	Aggregate, Microservice, Single purpose の 3 つから選択
14. Device Specific	デバイスに固有の API かどうか
15. Docs Home Page URL	API のドキュメントの URL
16. Architectural Style	API のアーキテクチャスタイル
17. Supported Request Formats	ペイロード(リクエスト)の形式を一つ以上選択
18. Supported Response Formats	ペイロード(レスポンス)の形式を一つ以上選択
19. Is This an Unofficial API?	非公式の API であるかどうか
20. Is This a Hypermedia API?	ハイパーメディアのサポートの有無
21. Restricted Access (Requires Provider Approval)	制限されたアクセスかどうか(プロバイダの承認が必要)

表 2 OpenAPI で定義可能な仕様
Table 2 Definable Specifications on OpenAPI

フィールド名	必須	概要
openapi	○	バージョニングを記述
info	○	API のメタデータを記述
servers	×	API を提供するサーバを記述(配列で複数記述可能)
paths	○	API で利用可能なエンドポイントやメソッドを記述
components	○	API で使用するオブジェクトスキーマを記述
security	×	API 全体を通して使用可能なセキュリティ仕様を記述
tags	×	API で使用されるタグのリスト(タグは固有)
externalDocs	×	外部ドキュメントを記述

表 3 OpenAPI のフィールドへの変換
Table 3 Converting to Fields of OpenAPI

ProgrammableWeb の仕様(番号)	OpenAPI のフィールド
3, 4, 5, 7, 8, 10, 11, 13, 14, 15, 17, 20, 21, 22	info
2	servers
1	paths
18, 19	components
6, 9, 12,	security
該当なし	tags
16	externalDocs

5.4 Web API 説明文書の要約

文書要約のアーキテクチャを図 5 に示す.

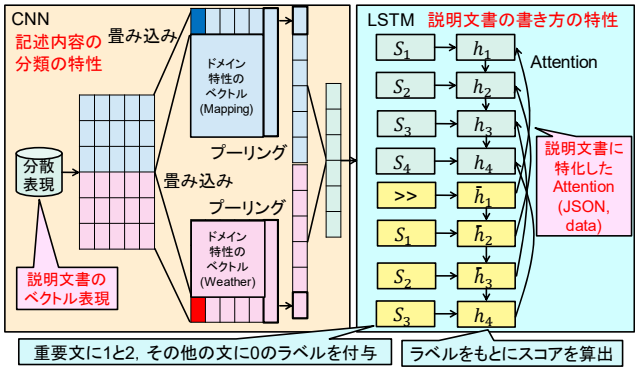

図 5 文書要約のアーキテクチャ
Figure 5 Document Summarization Architecture

5.4.1 学習方法

LSTM, CNN(Convolutional Neural Network), Attention を用いて文書要約を行う. LSTM ブロックは, 入力ゲート, 忘却ゲート, 出力ゲート, メモリセルで構成されている. 本研究で LSTM を用いた理由は, Web API の説明文書における長期記憶と短期記憶の依存関係を保持できるためである. 通常の文書要約では, 文書間で文章量をほぼ同等にしたデータセットに対して行われる. Web API の説明文書は, Web API 毎で文章量や仕様を記述する順番が異なっているという特徴がある. またカテゴリは多岐にわたり, カテゴリ毎でドメイン知識が異なっている. 提案方法では各文章の tf-idf を計算し, 重要文を特定する. 重要文に対してラベルを付与することで, 教師あり学習を行う. tf-idf を式(1)に示す. 単語ごとの tf-idf を計算し, それらを加算してスコアを算出する. 1 つの文書内でスコアが最も高かった文章に対してラベル 1, スコアが 2 番目, 3 番目に高かった文章に対してラベル 2, それ以外の文章に対してラベル 0 を付与し学習を行う.

$$tf\ idf = tf \times idf \tag{1}$$

$$tf = \left(\frac{\text{文書}A\text{における単語}X\text{の出現頻度}}{\text{文章}A\text{における全単語の出現頻度の和}}\right) \tag{2}$$

$$idf = \log\left(\frac{\text{全文書数}}{\text{単語}X\text{を含む文書数}}\right) \tag{3}$$

5.4.2 説明文書の要約

ProgrammableWeb 上の説明文書を要約する. 最初に抽出した説明文書を分散表現に変換する. 分散表現で入力された文章に対して CNN を適用する. CNN を用いて畳み込みを行うことで, Mapping, Weather, Mobile, Transportation, Social のそれぞれのカテゴリのドメインの特性を得る.

Attention を用いて文章を抽出する. エンコーダの隠れ層をソース, デコーダの隠れ層をターゲットとする. ソースは Key と Value に分離でき, ターゲットは検索クエリである Query とみなせる. Attention とは Query に一致する Key を索引し, 対応する Value を取り出す操作である. これは辞書の機能と同じである. エンコーダデコーダモデルの注意は, エンコーダのすべての隠れ層 Value から Query に関連する隠れ層 Value を注意の重みの加重和として取り出すことである. 本稿では, Attention により説明文書の書き方の特性を学習する.

5.4.3 Attention を用いた文章抽出

CNN を用いて文書エンコーダにおいて出力された文章ベクトルを順次 LSTM に入力することで文書を表現する隠れ層が生成される. 文章抽出器において LSTM を用いて文章の抽出する. あるタイムステップにおける文章抽出器の隠れ層は, 直前のタイムステップの隠れ層と値 p によって重み付けされた文章のベクトルによって決定される. p が文章を抽出するべきか判断する指標になる. p は文章に対する Attention である. p の値が大きいほど, その文章の重要度が高いと判断できる. p の値は文書エンコーダと文章抽出器の隠れ層を結合したベクトルを多層パーセプトロンに入力し, シグモイド関数に入力することで 0 から 1 の間の数値として得られる. 学習の際に重要文に 1, 2, その他の文に 0 を付与したラベルを使用する. 重要文は tf-idf を計算し, スコアの高かった上位 3 文章とする. tf-idf は文章中に多数出現する, かつ他文章中に出現しない重要語を特定できる. 重要語を多数含む文章を重要文と定義する. Mapping, Weather, Mobile, Transportation, Social の 5 つのカテゴリをそれぞれ学習する場合と, 5 つのカテゴリを一括して学習する場合の 2 パターンを行う.

5.5 学習モデルの生成

ハイパーパラメータ(最適化アルゴリズム, 学習数(epoch), バッチサイズ)を決定し学習モデルの生成を行う. 最終的な要約結果の精度によってハイパーパラメータの設定を見直す. 訓練誤差, 汎化誤差, 精度の評価に基づいて繰り返し学習を行い, 最適なモデル設計を確定する.

機械学習では誤差の収束をもって学習を終了させ, テストデータを用いて汎化能力を測る. 誤差が発散している場合は, なんらかの原因が学習を阻害している可能性がある. この場合は, ハイパーパラメータの調整や訓練データの見直しを行う.

5.6 学習の評価

学習の評価として訓練誤差と汎化誤差を使用する. 訓練誤差とは, 訓練データに対する誤差であり, 汎化誤差とはテストデータに対する誤差である. 誤差関数として多値分類の教師あり学習で一般的に用いられる式(4)に示すクロスエントロピーを使用する.

$$E = -\sum_{k} q(k)log(p(k)) \tag{4}$$

学習の評価を Mapping, Weather, Mobile, Transportation, Social の 5 つのカテゴリごとに学習した場合と 5 つのカテゴリの区別をつけずに一括して学習した場合の計 6 パターンで行った. 訓練データを学習させた際の訓練誤差とテストデータを入力した際の汎化誤差は, TensorFlow[6]の評価ツールである TensorBoard を用いて可視化する.

生成したテストデータを用いて要約を行い, 式(5)で表される ROUGE, 式(6)で表される単語の網羅率, 式(7)で表される圧縮率を用いて評価を行う. ROUGE-1 は, 式(5)で定義される ROUGE において N=1 の場合である. 参照要約中の N グラム

数に対する参照要約とシステム要約間で一致するNグラム数の割合をスコアとする評価尺度である．値は0から1までの範囲をとる．元の文書と抽出された文章を比較し，共通する単語が多いほど高い値をとる．単語の網羅率は，式(6)に示すように，参照要約とシステム要約で一致した単語数の参照要約の単語数に対する比率である．値は 0 から 1 までの範囲をとる．圧縮率は，式(7)に示すように，参照要約の単語数からシステム要約の単語数を引いた値の参照要約の単語数に対する比率である．値は 0 から 1 までの範囲をとる．

ROUGE による評価で値が 0 に近い場合や TensorBoard を用いた評価で誤差が発散している場合は，ハイパーパラメータの再設定や訓練データとテストデータの割合の変更を行う．

$$\boldsymbol{ROUGE-N}(\mathrm{C},\mathcal{R}) = \left(\frac{Count_{match}(gram_N)(\mathrm{C},\mathcal{R})}{\#\ of\ N\text{-}grams\ \in\ \mathcal{R}}\right) \quad (5)$$

$$\boldsymbol{単語の網羅率} = \frac{説明文書と要約で一致した単語数}{説明文書の単語数} \quad (6)$$

$$\boldsymbol{圧縮率} = \frac{説明文書の単語数 - 要約の単語数}{説明文書の単語数} \quad (7)$$

5.7 Web API 仕様記述の OpenAPI 形式への変換

OpenAPI 形式の info オブジェクト内にある description フィールドに要約した Web API 説明文書を記述することで，Web API 仕様記述の OpenAPI 形式への変換を行う．変換には Swagger Editor を使用する．Swagger Editor は Swagger の仕様にしたがって YAML ファイルを編集できるツールである．また，編集中に動的に Web API 仕様書が生成されるため，Web API 仕様書のプレビューが可能である．生成された Web API 仕様書から，ProgrammableWeb 上の仕様との相違を確認する．

5.8 Web API 仕様書を OpenAPI 形式で公開

Swagger Editor の機能である Generate Server を用いてサーバを生成する．パッケージを実行することで，Swagger Editor で作成したファイルがサーバにアップロードできる．これにより，Web API 仕様書が公開される．公開された Web API 仕様書は，Swagger UI を用いて確認する．Swagger UI は YAML ファイルのビュワーである．Swagger UI にある入力欄に生成した Web API 仕様書の URL を記述し，内容を検証する．内容に相違がある場合は，YAML ファイルを修正する．

6. プロトタイプの実装

6.1 実装環境

提案方法を実現するプロトタイプの実装環境を表 4 示す．

表 4 ソフトウェアコンポーネント
Table 4 Software Components

コンポーネント	コンポーネント名	版
OS	Ubuntu	18.04
実装言語	Python	3.6.4
深層学習フレームワーク	TensorFlow	1.3.0
評価結果の可視化	TensorBoard	0.1.8
スクレイピングツール	Beautiful Soup	4.2.0
Web API 記述	OpenAPI	3.0.2

6.2 プロトタイプのアーキテクチャ

提案方法を評価するために実装したプロトタイプのアーキテクチャを図 6 に示す．

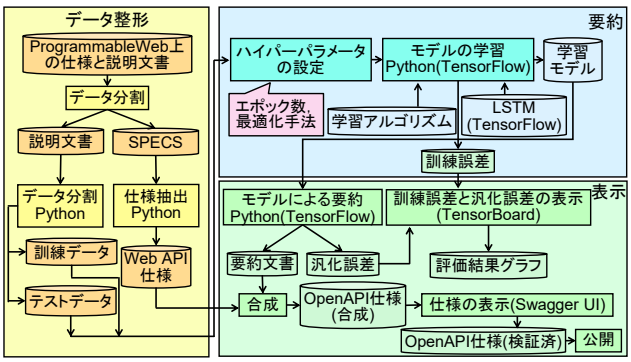

図 6 プロトタイプのアーキテクチャ
Figure 6 Prototype Architecture

(1) データ整形

ProgrammableWeb から取得したオリジナルデータを説明文書と SPECS に分割し，説明文書から訓練データとテストデータ，SPECS から OpenAPI で定義可能な仕様を抽出する処理を Python で実装した．

(2) 要約

モデルの学習処理を Python と TensorFlow で実装した．

(3) 表示

訓練誤差と汎化誤差による評価結果の可視化を TensorBoard によって行った．仕様文書の表示を SwaggerUI で行った．

6.3 実装結果

実装結果を表 5 に示す．規模は実装言語として Python を用いた場合の LOC である．

表 5 実装結果
Table 5 Implementation results

プログラム名	機能	規模(LOC)
data_reader.py	データの読み取り，テンソルへの変換	331
evaluate.py	テストデータを用いた評価	181
generate.py	モデルのビルドを実行	173
model.py	モデルの定義を記述	412
pretrain.py	事前学習を実行	358
train.py	学習を実行	345
utils.py	ソフトマックス関数などの定義を記述	118

7. 実データへの適用による評価

7.1 適用の目的

ProgrammableWeb では，独自の形式を使用して Web API 利用者に対して Web API の通信方式や認証モデルの情報などを提供している．しかし，この形式は静的に定義されている．また，Web API の使い方や通信方式が記載された説明文書は Web API 以外の情報を含んでおり，使用性を低下させている．実際の ProgrammableWeb 上の説明文書に対してプロトタイプを適用し，提案方法の有効性と妥当性を評価する．

7.2 適用対象

ProgrammableWeb の Web API に対して提案方法を適用した．文章数の違いによる結果への影響を最小限にするため，抽出する Web API の数は Weather のみ 299 で，その他のカテゴリは 300 に統一した．表 6 に適用対象のデータ数，表 7 に学習データ数を示す．全体は 5 つのカテゴリの合計である．

文章数，単語数ともに多少の差異があったが，実験結果に影響を及ぼすほどではなかった．また，各カテゴリの説明文書の書き方についても大きな差異はなかった．学習データに大きな偏りがないことが確認できたため，このデータ数で適用を行う．

表 6 適用対象のデータ数
Table 6 Numbers of Data of Scope

カテゴリ	最新更新日時	Web API 登録数
Mapping	2019/8/6	1,030
Weather	2019/8/7	299
Mobile	2019/8/15	1,063
Transportation	2019/8/17	509
Social	2019/7/15	1,547
全体	-	4,448

表 7 学習データ数
Table 7 Numbers of Learning Data

カテゴリ	Web API 数	文章数	単語数
Mapping	300	1,126	19,689
Weather	299	1,151	20,614
Mobile	300	1,148	19,461
Transportation	300	1,137	19,165
Social	300	1,143	19,141
全体	1,499	5,705	98,070

7.3 適用結果

式(4)に示すクロスエントロピーによって得られた訓練誤差と汎化誤差を 5 つのカテゴリごとに学習した場合と 5 つのカテゴリの区別をつけずに一括して学習した場合の計 6 パターンで算出した．訓練誤差の結果を図 7，汎化誤差の結果を図 8 に示す．

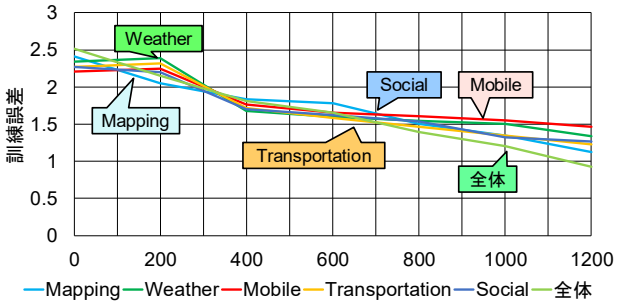

図 7 訓練誤差の結果
Figure 7 Result of Training Error

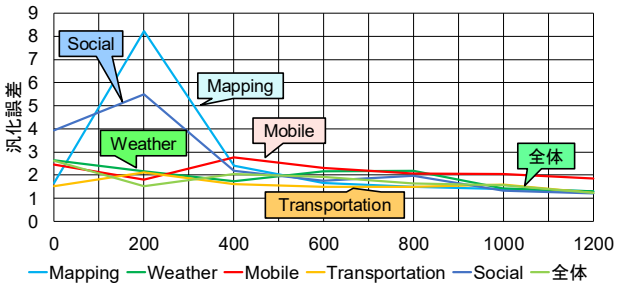

図 8 汎化誤差の結果
Figure 8 Result of Generalization Error

訓練誤差については，どのカテゴリにおいてもほぼ同じ結果を示した．しかし，汎化誤差については Mapping と Social のカテゴリで異なった傾向を示した．Mapping は 1 ステップ目での誤差が大きくなった．Social は 0 ステップ目での誤差が大きくなった．Mapping と Social の訓練データでは，どのようなサービスを提供する Web API であるかを先に説明し，その次に通信方式などの説明をしていた．テストデータでは，訓練データの順番とは逆になっていた．したがって，訓練データとテストデータの文章の並び方や書き方の違いが影響を与えたと推定できる．

言語モデルの良さを測る尺度であるパープレキシティ[15]を 5 つのカテゴリごとに学習した場合と 5 つのカテゴリの区別をつけずに一括して学習した場合の計 6 パターンで算出した．パープレキシティを式(8)に示す．

$$\text{パープレキシティ} = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 q(x_i)} \tag{8}$$

訓練データのパープレキシティの結果を図 9，テストデータのパープレキシティの結果を図 10 に示す．

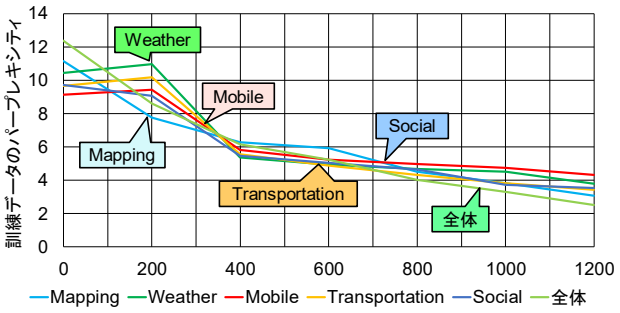

図 9 訓練データのパープレキシティの結果
Figure 9 Result of Perplexity of Training Data

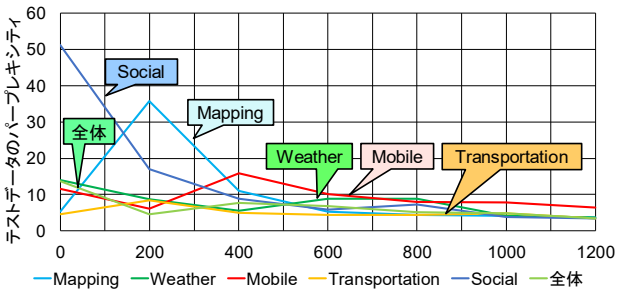

図 10 テストデータのパープレキシティの結果
Figure 10 Result of Perplexity of Test Data

パープレキシティについては，誤差と同じ傾向を示した．パープレキシティは言語モデルの良さを測る尺度であることから，訓練データとテストデータの用いられた単語や文章の並び方の違いが影響を与えたと推定できる．

7.4 評価結果

ProgrammableWeb の Weather カテゴリに属する Web API (Magic Seaweed Forecast API)に対して要約を行った結果を図 11 に示す．4 文(88 単語)の文書が 2 文(47 単語)に短縮された．削除された文章は同じ内容を繰り返している文や，Web API ではなくサービスに関する情報を含んでいた．

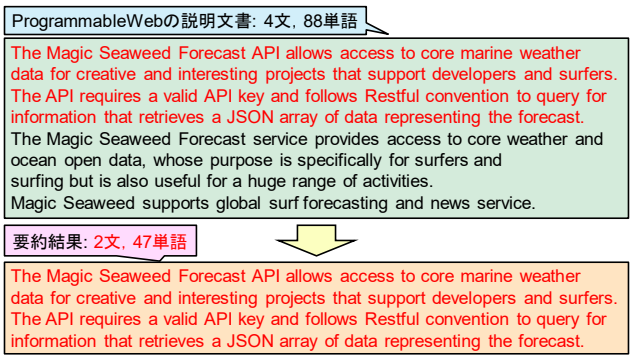

図 11 Weather カテゴリの Web API を要約した結果の例
Figure 11 Example of result summarized Web API in the Weather category

ProgrammableWeb 上の仕様書を図 12，OpenAPI 形式の仕様書を図 13 に示す．

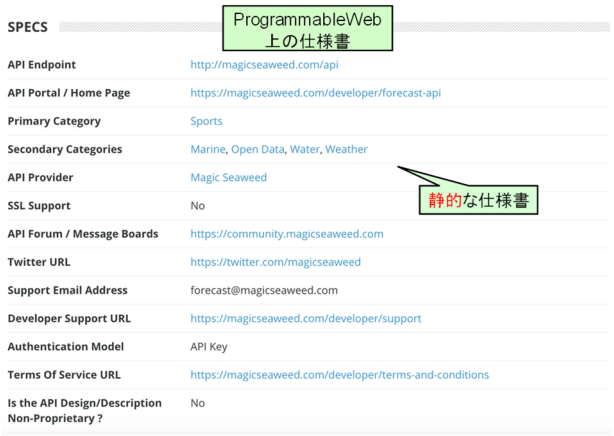

図 12 ProgrammableWeb 上の仕様書
Figure 12 Specification on ProgrammableWeb

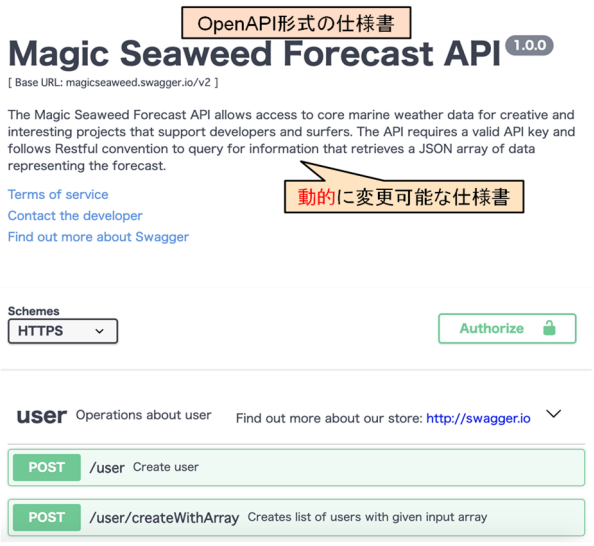

図 13 OpenAPI 形式の仕様書
Figure 13 Specification of OpenAPI Format

ProgrammableWeb 上の仕様書は静的に定義されている．Web API を公開した後変更が発生した場合は，仕様書を変更して Web API を公開し直す必要がある．OpenAPI 形式の仕様書は YAML ファイルで定義されているため，動的に変更できる．

通信方式を含む重要文に Attention を設定した場合の重要文と重要文以外のスコアを図 14 に示す．スコアの定義を式(9)に示す．文書エンコーダと文章抽出器における隠れ層を結合したベクトルを多層パーセプトロン(MLP)に入力する．その結果をソフトマックス関数に入力し，スコアを得る．重要文も重要文以外の文も複数あるため，スコアは平均値である．重要文とは，文章中に多数出現する，かつ他文章中に出現しない重要語を含む文である．重要語は tf-idf を用いて特定する．

$$スコア = \sigma(MLP(\overline{h_t} : h_t)) \quad (9)$$

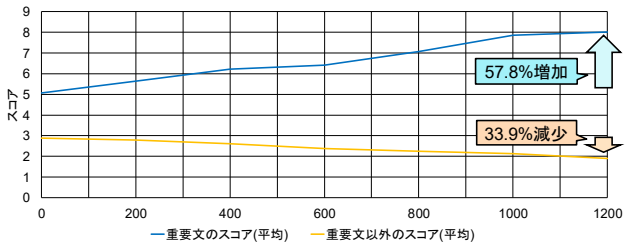

図 14 重要文と重要文以外のスコア
Figure 14 Scores of Important Sentence and Others

通信方式を含む重要文以外の文に Attention を設定した場合の重要文と重要文以外のスコアを図 15 に示す．

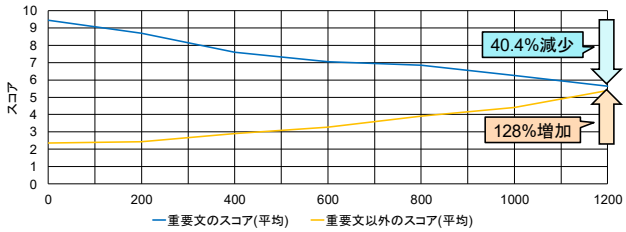

図 15 重要文と重要文以外のスコア
Figure 15 Scores of Important Sentence and Others

ROUGE-1，単語の網羅率，圧縮率の評価結果(平均値)を図 16 に示す．全てのカテゴリを学習した場合の評価は，Weather カテゴリの Web API を用いて行った．

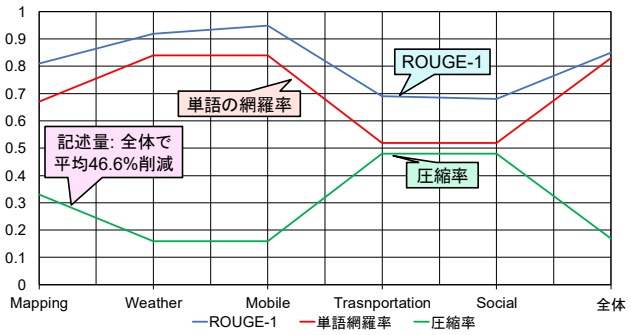

図 16 ROUGE-1，単語の網羅率，圧縮率の評価結果
Figure 16 Evaluation Result of ROUGE-1, Coverage of Words, and Compression Ratio

8. 考察

8.1 評価に基づく考察

(1) 誤差に基づく学習の良さに関する考察

図 7，8 に示すように，5 つのカテゴリごとに学習した場合と全カテゴリを一括学習した場合と比較し，訓練誤差では全てのカテゴリを学習した場合の方が低い値を示したが，汎化誤差には違いは見られなかった．いずれにおいても訓練誤差と汎化誤差

は約 1,200 ステップで収束しており，学習が成功していることを示している．

(2) パープレキシティに基づく Web API 説明文書の要約の良さに関する考察

図 9，10 に示すように，Mobile のカテゴリのみを学習した場合のみパープレキシティが高かったが，その他の場合はほぼ同じ値を示した．パープレキシティは文章の候補数を示しているため，Mobile のカテゴリで同じ内容を含んだ文章が多く含まれることが影響していると推定できる．これは，同じ内容である仕様書を別名の Web API としてそれぞれ登録しているためである．

(3) Attention を用いて抽出した Web API 仕様書の情報量に関する考察

訓練誤差と汎化誤差は約 1,200 ステップで収束したため，0 ステップ目，200 ステップ目，400 ステップ目，600 ステップ目，800 ステップ目，1,000 ステップ目，1200 ステップ目でのスコアを算出し，その経過を観測した．(図 14，15)重要文に Attention を設定した場合，重要文のスコアが増加し，重要文以外のスコアが減少した．重要文以外に Attention を設定した場合，重要文のスコアが減少し，重要文以外のスコアが増加した．ステップを重ねるごとに増加と減少の割合はそれぞれ異なっていたが，傾向が異なることはなかった．これは Attention を用いてどの文を抽出すべきかを判断するため，Attention の与え方がスコアに影響を与えている．スコアの増加や減少の割合が異なっているのは，1 ステップ目におけるスコアの違いや，1 つの文書内の重要文と重要文以外の割合が影響していると推定できる．

(4) ROUGE-1，単語の網羅率，圧縮率を用いた Web API 仕様書の記述量削減に関する考察

図 16 に示すように，ROUGE-1，単語の網羅率，圧縮率の結果は生成した Web API 仕様記述が利用者にとって重要な情報を保持したまま要約できたことを示している．要約によって記述量が削減されたことから，Web API の利用者にとって習得容易性[18]が向上したといえる．

8.2 関連研究との比較

関連研究[19]では機械学習を用いて URL，パステンプレート，HTTP メソッドを自動的に抽出しているが，説明文書の要約はされていない．仕様書に記述された URL などをそのまま抽出する方法である．したがって，生成された仕様文書の記述量が増大する．また，抽出された URL などがどのようなレスポンス，プロトコルであるかなどを説明することができない．

提案方法では仕様の対応付けを行うことで仕様を説明できる．また，重要文を抽出することで，説明文書の重要な情報を保持したままの要約と記述量の削減が可能である．

9. 今後の課題

今後の課題は以下の 3 点である．

(1) 文書要約方法の検討

提案方法では抽出型の要約を行なっているが，抽象型や両者を組み合わせた方法も含めて検討を行う．

(2) Attention を活かした分析の検討

提案方法では Attention を文章に対して付与していたが，単語に対して付与することについても検討を行う．

(3) 他キュレーションサイトへの適用

提案方法を ProgrammableWeb 以外のキュレーションサイトに適用し，外部妥当性を確認する必要がある．

10. まとめ

機械学習を用いて説明文書を要約し，Web API 仕様記述を抽出することによる Web API 仕様文書の生成方法を提案した．重要文に Attention を付与することで重要な情報を保持したまま要約でき，Web API 利用者の仕様理解の促進が可能になる．ROUGE-1，単語の網羅率，圧縮率の結果から，通信方式などの Web API 利用者にとって重要な情報を保持し，同じ内容を繰り返す文や Web API プロバイダが提供するサービスなどの Web API に関係ない情報を含んだ文を削減できたことを確認した．

Web API のキュレーションサイトでは，それぞれで異なる仕様記述形式を使用している．提案方法は異なる仕様記述形式の統一への支援として期待できる．

参考文献

[1] 青山 幹雄, Web ソフトウェア工学の新時代，ウィンターワークショップ 2019・イン・福島飯坂論文集，情報処理学会, Jan. 2019, pp. 25-26.
[2] H. Cao, et al., Automated Generation of REST API Specification from Plain HTML, Proc. of ICSOC'17, Nov. 2017, pp. 453-461.
[3] J. Cheng, et al., Neural Summarization by Extracting Sentences and Words, Proc. of ACL'16, Aug. 2016, pp. 484-494.
[4] T. Espinha, et al., Web API Growing Pains: Stories from Client Developers and Their Code, Proc. of CSMR-WCRE'14, IEEE, Feb. 2014, pp. 84-93.
[5] R. T. Fielding, et al., Reflections on the REST Architectural Style and "Principled Design of the Modern Web Architecture", Proc. of ESEC/FSE'17, ACM, Sep. 2017, pp. 4-14.
[6] Google Inc., TensorFlow, https://www.tensorflow.org/.
[7] S. Liu, et al., Hierarchical RNN Networks for Structured Semantic Web API Model Learning and Extraction, Proc. of ICWS'17, IEEE, Jun. 2017, pp. 708-713.
[8] B. A. Myers, et al., Improving API Usability, CACM, Vol. 59, No. 6, Jun. 2016, pp. 62-69.
[9] Preferred Networks, Chainer, https://chainer.org.
[10] ProgrammableWeb, https://www.programmableweb.com/.
[11] A. See, et al., Get To The Point: Summarization with Pointer-Generator Networks, Proc. of ACL'17, Jul. 2017, pp. 1073-1083.
[12] SmartBear Software, Open API Specification, https://swagger.io/specification/.
[13] SmartBear Software, Swagger, https://swagger.io/.
[14] The Linux Foundation, Open API Initiative, https://www.openapis.org/.
[15] 坪井 祐太, 深層学習による自然言語処理，講談社, 2017.
[16] A. Vaswani, et al., Attention Is All You Need, Proc. of NIPS'17, Dec. 2017, pp. 1-15.
[17] E. Wittern, et al., Opportunities in Software Engineering Research for Web API Consumption, Proc. of WAPI'17, IEEE, May 2017, pp. 7-10.
[18] 山本 里枝子 他, Web API の習得容易性と相互運用性，およびその定量評価方法の提案と適用評価，情報処理学会論文誌, Vol. 60, No. 10, Oct. 2019, pp. 1896-1914.
[19] J. Yang, et al., Towards Extracting Web API Specifications from Documentation, Proc. of MSR '18, ACM, May 2018, pp. 454-464.