

自然言語処理 —RNN—

<https://satoyoshiharu.github.io/nlp/>

自然言語処理 RNN

[解説動画](#)



ここではRNN（リカレントニューラルネットワーク）の基本的な概念についてみていきます。

100本ノック第9章とRNNの位置づけ

- 100本ノック課題集第9章は、RNN、CNN、Transformerを扱っている。
- RNNの扱いが大きいが、RNN、そしてその改善版のゲート付きRNNは、時系列データ（自然言語は単語の時系列）の基幹技術だった。
- しかし、音声認識（信号の時系列）は別として、文章テキストデータの処理に関しては、その位置をTransformerに譲ってしまった。



RNNは、時系列データの処理技術として、音声認識や機械翻訳のDeep Learningの基幹技術だったことがあります。
しかし、いまのところ、Transformerにその位置を奪われたかのような感じとなっています。
ただ、RNNは、Transformerでも重要なSeq2SeqやAttentionなど関連技術の生まれる母胎だったので、概略だけ見ていきます。

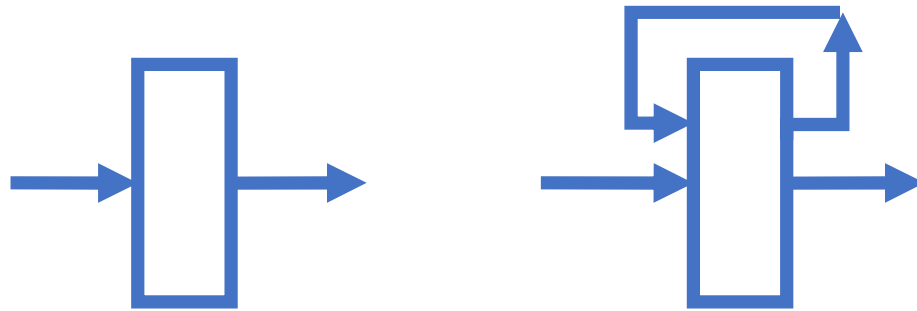
項目と系列

- 年代区別：子供、大人、老人
- 訪問場所：東京、富士山、京都
- コマンドの種類：ls,cd,pwd
- 音素 ji,n,koo,ti,noo
- 文字 人,工,知,能
- 単語 人工,知能,は
- ...
- 年齢変化：子供->大人->老人
- 訪問順：東京->富士山->京都
- スクリプト、プログラム: pwd->cd->ls
- 音素列 ji->n->koo->ti->nou
- 文字列 人->工->知->能
- 単語列 人工->知能->は
- ...



世の中の事象は、項目と系列に分類できます。
項目とは、関連しているが互いに独立した物事です。
系列は、関連していて、かつ順序に意味がある物事です。

Feed Forward（前向き推論）とRecurrent（再帰推論）



これまで、全結合やCNNを見てきました。

これらは、入力を与えられて、演算を施して後段に送る、前向き一方向の処理でした。

ここで、RNNが登場し、系列を扱えるようになりました。

RNNは、あるレイヤの出力を自分に戻して入力し、過去のデータを踏まえた演算を行います。

RNNのリカレントとは、再帰という意味です。

再帰関数というのは、次の処理に自分自身をかかわらせる仕組みですが、にたようなものです。

Feedforward	Recurrent
固定長ベクトル	可変長のベクトル
ベクトル、空間情報	順序情報
内部状態を持たない関数	内部状態を持つオブジェクト



全結合やCNNは前向きに演算が進むので、FeedForwardネットワークといわれます。

これに対してリカレントネットワークは、根本的に異なる特徴を持ちます。FeedForwardネットは、固定長ベクトルを入力としますが、リカレントネットは可変長データを入力とします。

FeedForwardネットは、1次元や2次元のベクトルを処理しますが、リカレントネットは順序情報を処理します。

FeedForwardネットは、内部状態を持たない関数のようなものですが、リカレントネットは内部状態を持つオブジェクトみたいなものです。

RNNの応用



RNNは、系列情報を扱えます。

そのため、応用は広範囲に及びます。

言語モデルというのは、コンテキストを与えてそこに登場する単語などを予測するモデルです。音声認識や機械翻訳でも活躍します。

テキスト分類は、パラグラフを与えて、情報集約し分類します。

テキスト生成は、リカレントネットの特徴を生かし、戦闘単語などから後続の単語等を予測し、つなげて出力します。

株価予測は、時系列上のデータから未来の株価を予測します。

ロボット制御では、センサーのデータ流から、次を予測し、適当なコントロールを指示します。

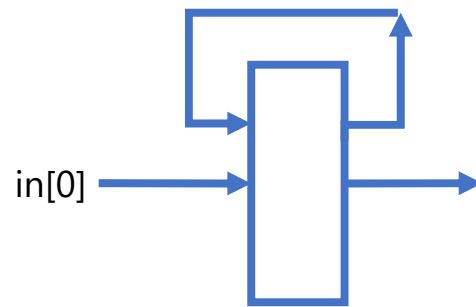
パラグラフの分類と、生成機能を組み合わせれば、自動応答ができます。

チャットボットもそうです。

画像情報を集約した後で、RNNの生成能力を使えば、自動キャプションができます。

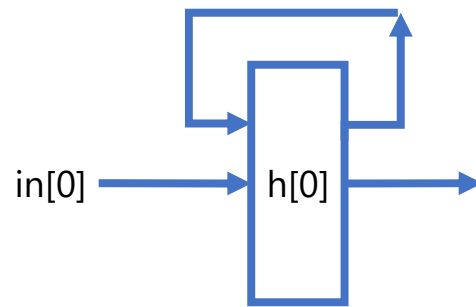
などなど。

RNNの動作



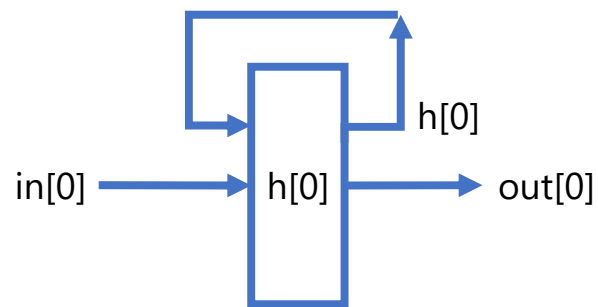
RNNの基本動作を見ていきます。
0番目の入力を与えられました。

RNNの動作



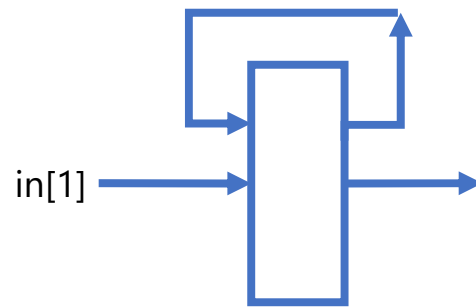
RNNは0番目の入力に対し、0番目の隠れ状態を計算します。

RNNの動作



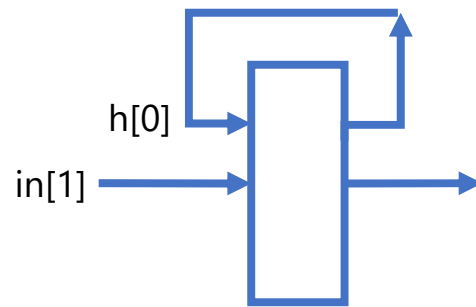
RNNは、0番目の出力をすると同時に、0番目の隠れ状態を自分へ送り返します。

RNNの動作



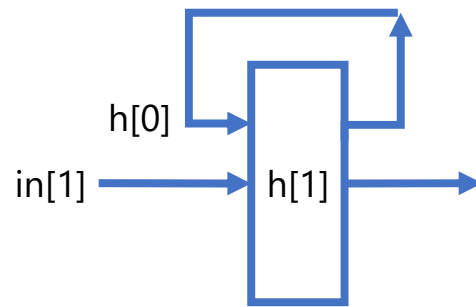
2番目の入力がありました。

RNNの動作



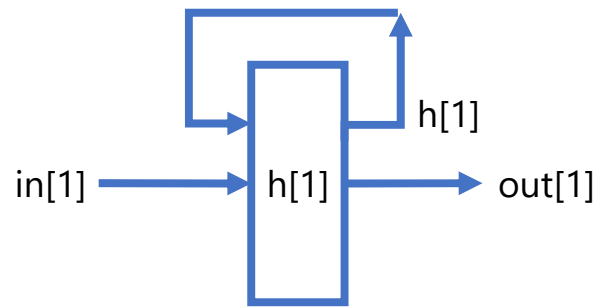
RNNは、1番目の入力と先に計算した0番目の隠れ状態を入力とします。

RNNの動作



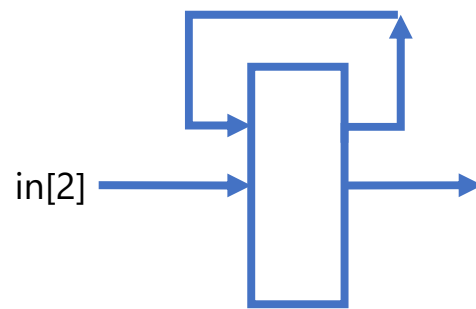
そして、1番目の隠れ状態を計算します。

RNNの動作



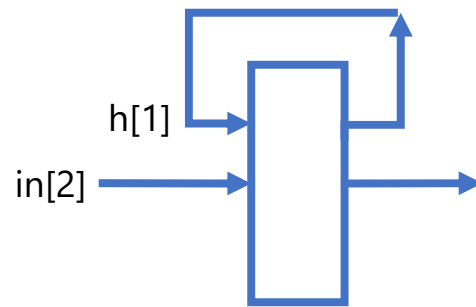
1番目の隠れ状態を自分に戻すと同時に、1番目の出力をします。

RNNの動作



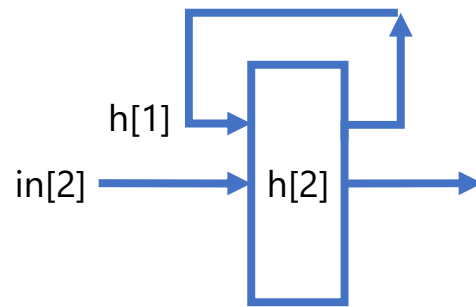
2番目の入力がありました。

RNNの動作



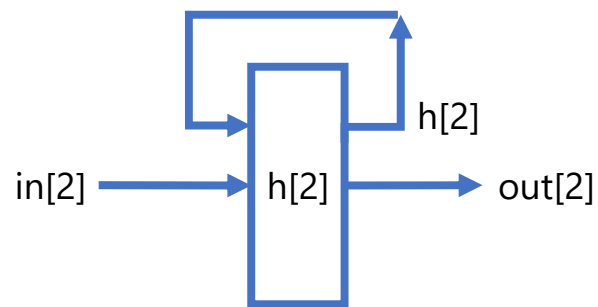
RNNは2番目の入力と、1番目の隠れ状態を、入力とします。

RNNの動作



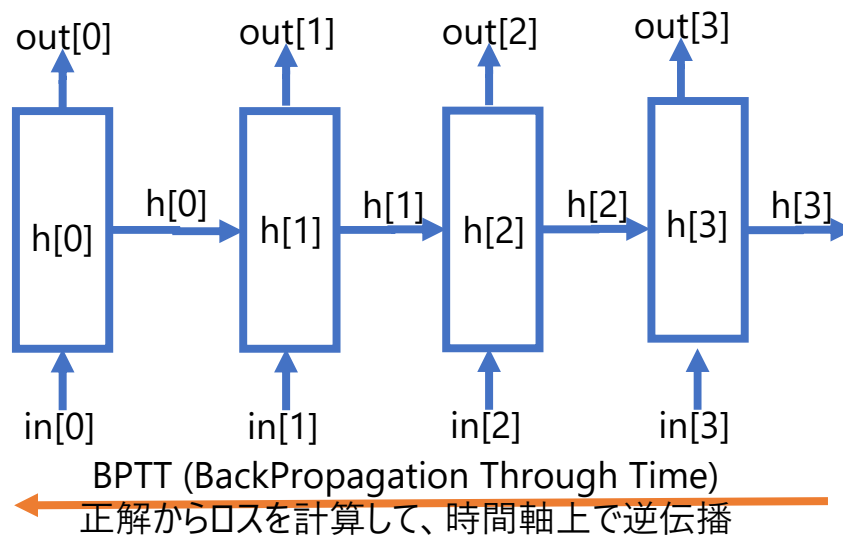
そして、2番目の隠れ状態を計算します。

RNNの動作



などなど。

並べると（箱は同一のもの）



RNNは、以上のように動作しますが、RNNが持っている隠れ状態ベクトルや演算機構はどの時刻でも同一のものです。

時間軸に沿ってRNNの動きをばらしてみると、この絵のようになります。

RNNでの逆伝播学習は、この時間を展開した状態のもので、時間を逆にたどって行われます。

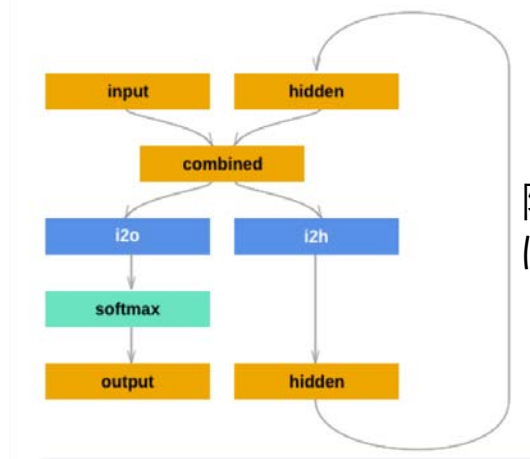
それをBPTT(BackPropagation Through Time)と呼びます。

RNN課題 1

- `char_rnn.ipynb` に、文字列を読んでクラス判定するサンプル（最後の出力からクラスに所属する確率をとる）と名前文字列を生成するサンプル（RNNの各時刻の出力を順番に取り出して並べる）があります。
- PyTorchサイトにあるチュートリアルサンプルをさらに単純化し、日本語データを扱うように変更したものです。
- 実用的には、PyTorchにrnnクラスがすでにあるのでそれを使うだけでよいのですが、これらサンプルは教育目的で、rnnクラスを使わず、rnnのロジックを生で書き、原理を説明しています。
- PythonやPyTorchに慣れるのにいいサンプルです。読解しましょう。

サンプル：文字レベルRNNで名前分類

最後の出力は、
入力系列の圧
縮表現となる

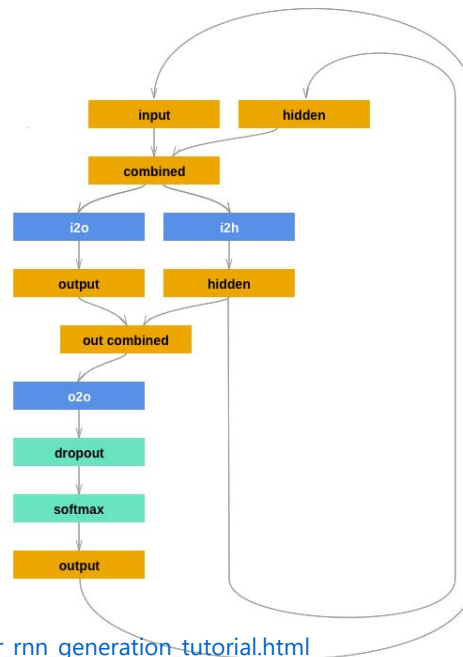


隠れ状態を再帰的
に入力へ

https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

サンプル：文字レベルRNNで名前生成

出力を順次
取り出すと、
系列が生成
できる



生成問題
ではある
時刻の出
力を次の
時刻の入
力とする

https://pytorch.org/tutorials/intermediate/char_rnn_generation_tutorial.html

参考資料

- ゼロから作るDeep Learning ② 第5章
- [The Unreasonable Effectiveness of Recurrent Neural Networks](#)
- 教育動画
 - [Deep Learning入門：Recurrent Neural Networksとは？](#)
 - [RNN：時系列データを扱うRecurrent Neural Networksとは](#)

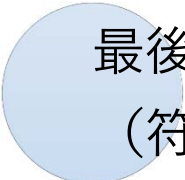
自然言語処理 Seq2Seq (Encoder- Decoder)

[解説動画](#)




ここでは、Seq2Seq、別名Encoder-Decoderアーキテクチャについてみていきます。

RNNの二つの顔



最後の隠れ状態が系列データの情報集約
(符号化)



ある時刻の出力を次の時刻の入力とすれば、
系列生成 (復号化)



RNNは、二つの顔を持っています。

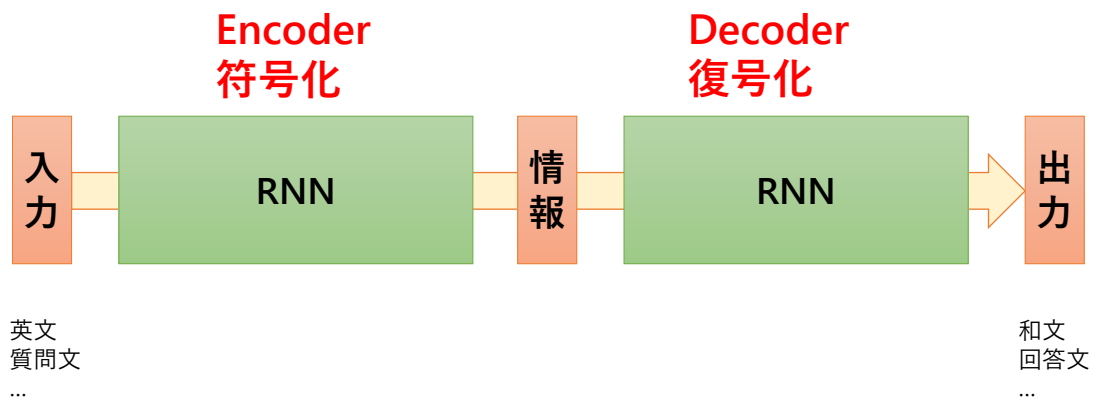
系列データを処理した後、最後の隠れ状態ベクトルは、いわば、入力系列データを集約したものとなります。

入力系列をベクトルに符号化したといえます。

一方、RNNの各時刻の出力を並べれば、新しい系列を生成していおることになります。

これは、何らかの入力を、系列へ変換している、復号化しているといえます。

Seq 2 Seq



以上のようなRNNの二つの顔を応用し、Seq2Seqというネットワーク構成が生まれました。

それは、2段階に分けてRNNを配置し、前段は符号化の役割を担い、最後の隠れ状態を介して、後段は復号化をするものです。

この構成によって、ある系列を、別の系列へ変換することができます。

このようなネット構成は、RNNだから可能な特殊な構成ではなく、TransformerでもEncoder-Decoderが作れるなど、汎用性の高い構成です。

Seq2Seq あるいは Encoder-Decoderの応用

- 機械翻訳： ソース言語->ターゲット言語
- 音声認識： 音響特徴列->単語列
- 要約： 記事->要約
- Q & A： 質問文->回答文
- チャット： つぶやき->返答つぶやき
- 映像自動キャプション： 画像フレーム列->説明文

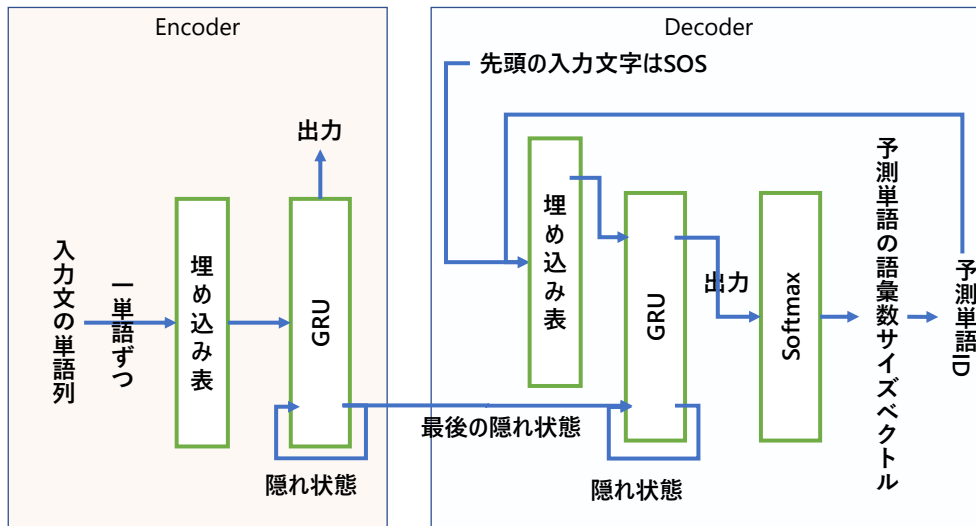


系列を変換することの応用は、広範囲に及びます。
機械翻訳は、ソース言語をターゲット言語へ変換します。
音声認識は、音響からの特徴列を単語列に変換します。
要約は、パラグラフをよりも次回文章に変換します。
Q&Aは、質問文を回答文へ変換します。
チャットも同様です。

RNN課題 2

- 日英翻訳データ `jpn.txt` をdownloadし、グーグルのマイドライブの下、Colab Notebooksフォルダーに、uploadしてください。
- スタログに`rnn_translation.ipynb` があります。それを`jpn.txt`と同じ場所においてください。これもPyTorchサンプルを単純化し、日本語データを扱うように変更したものです。
- GRUというのは、ゲート付きRNNの一つで、この後で解説します。
- 埋め込み表に関しては、単語ベクトルのスライドを見てください。
- 読解します。

ネットワーク構成



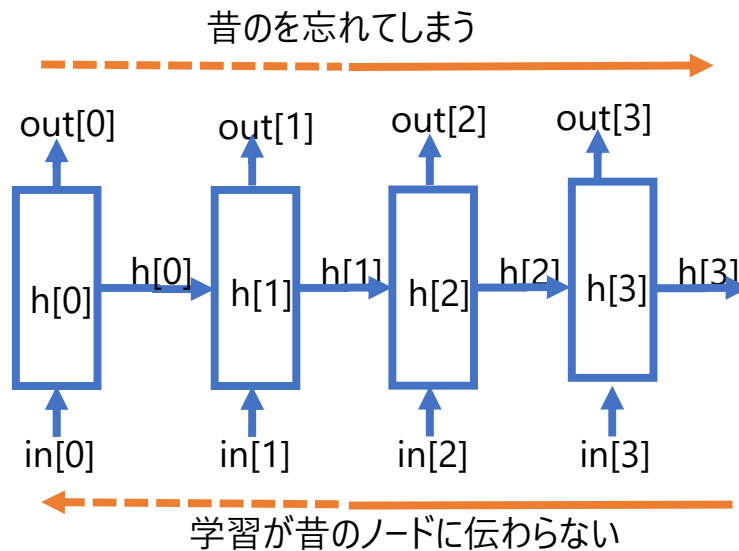
自然言語処理 ゲート付きRNN (LSTM, GRU)

[解説動画](#)



ここでは、ゲート付きRNNと呼ばれるLSTMやGRUの考えを見ていきます。

素のRNN(Elman-netと呼ばれる)の問題点

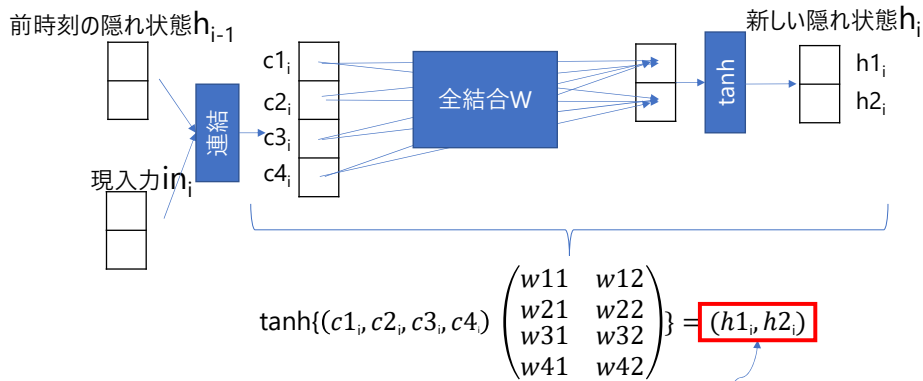


素のRNNは、Elmanネットと呼ばれます。

それは、一つの隠れ状態ベクトルで、時刻をおって新しい情報を塗り替えていくため、昔の情報を忘れがちになるという欠点があります。

また、逆伝播時も、学習効果が昔のノードに伝わりにくいという欠点があります。

Elman Netは、忘れやすい



毎時刻ごと上書きされるので、最終入力の影響を最も受けやすく、最初の頃の入力の影響は薄れてしまう。

少し具体的に見ます。

RNNが、隠れ状態ベクトルを持っています。

その値は、各時刻ごとに書き換えられます。

1時刻前の自分の隠れ状態を入力とするため、多少でも過去の情報は残りますが、毎回書き替えるのでその情報は薄くなります。

ゲート付きRNN (LSTM,GRU)

加算によって長期記憶を実現

情報の経路上にゲートを配置し、記憶する情報を学習で最適化



その欠点を克服するため、RNNの発展形としてゲート付きRNNというものが生まれました。

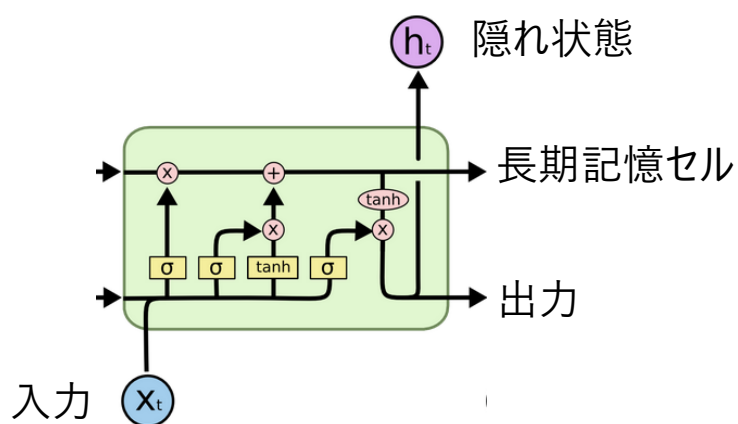
その改善点は2点あります。

一つは、加算によって長期記憶を実現するということ。

二つ目は、流れる情報の選別をゲートで制御するということです。

具体的に見ていきます。

LSTM (Long Short-term Memory)



<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>



LSTMは、Long Short-Term Memoryの略称です。
内部構成は複雑ですが、Elmanネットに比べて、隠れ状態以外に、長期記憶のためのベクトルを持たせたことが特徴です。

足し算は、記憶の効果がある

heの埋め込み表現

	1.0		
--	-----	--	--

sheの埋め込み表現

	-1.0		
--	------	--	--

学習の結果、たまたま、
状態ベクトルの第2要素が性別を表現したとする

he

	1.0		
--	-----	--	--

+

is

	0		
--	---	--	--

+

at

	0		
--	---	--	--

+

home

	0		
--	---	--	--

↓

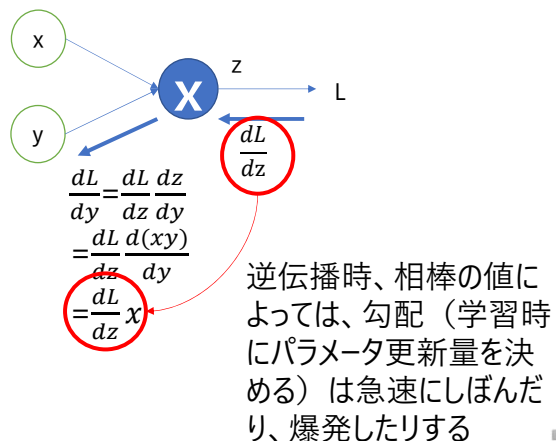
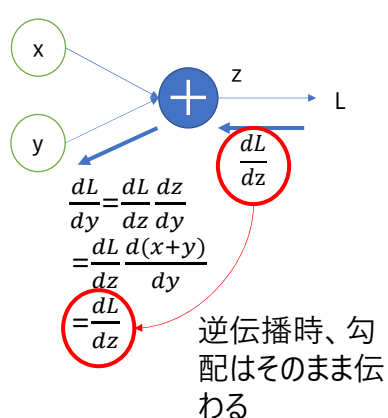
	1.0		
--	-----	--	--

ベクトルの加算の結果、最終状態であっても、性別情報が記憶されている



ここで、足し算の効果を見てみます。
特徴ベクトルのある添え字の数値が、性別という特徴をたまたま表現していたとします。
もしも、とベクトルを加算すれば、当添え字の個所の数値は、過去に獲得した情報を維持します。
加算は、長期記憶の効果があるのです。

足し算は逆伝播時に勾配をそのまま伝えるので先祖状態にも学習効果が届く

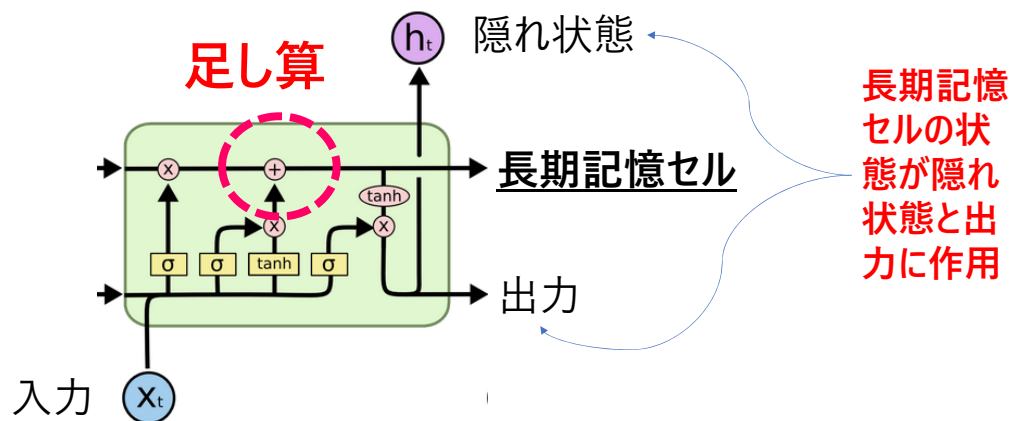


また、加算は別の利点も持ちます。

逆伝播時に、後段から伝わった勾配が、そのまま前段に流れるのです。

そのため、足し算を使えば、勾配消失や購買爆発が起こるような場合を、防ぐことができます。

LSTMの長期記憶

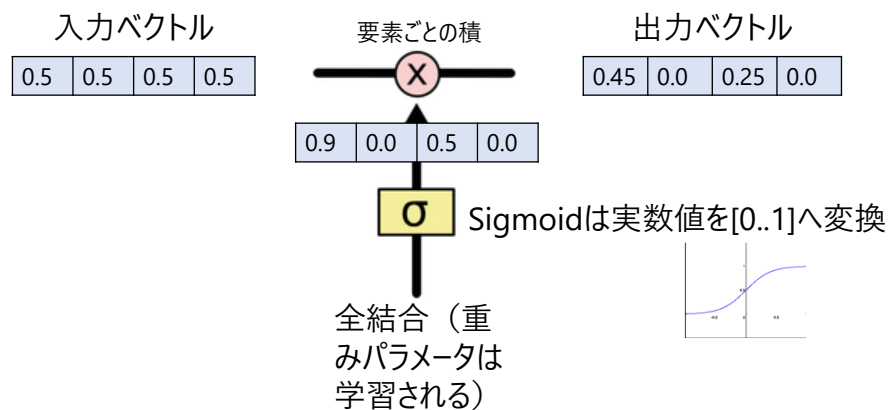


<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>



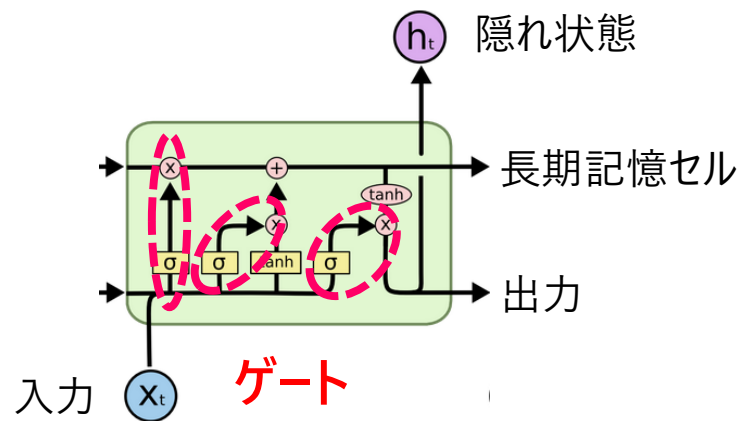
LSTMでは、この足し算の利点を利用して、長期記憶セルというベクトルに、足し算で情報を蓄えていきます。
そして、隠れ状態や出力の計算は、この長期記憶を加味して行います。
その結果、LSTMは、長期記憶を持っているかのようにふるまいます。

ゲート：ベクトルの各位置に対して[0..1]の重みをかけて、通過する値を制御する



さらに、LSTMは、ゲートを情報経路に設置します。
ゲートというのは、流れる情報を選別するように、情報経路に、全結合 + Sigmoidレイヤを結合するものです。
逆伝播学習の過程で、ゲートの選別機能を果たす全結合レイヤが最適化されて、LSTM内を流れる情報が選別されるようになります。

LSTMのゲート



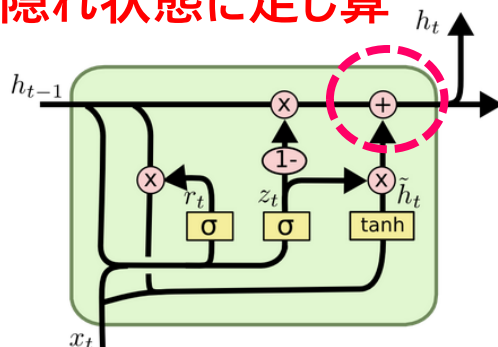
<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>



LSTMでは、そのようなゲートが数か所設置されています。
入力の選別、記憶内容の選別、出力の選別などです。

GRU (Gated Recurrent Unit)

隠れ状態に足し算



<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>



次に、LSTMを簡略化したものにGRUがあります。

Gated Recurrent Unitの略です。

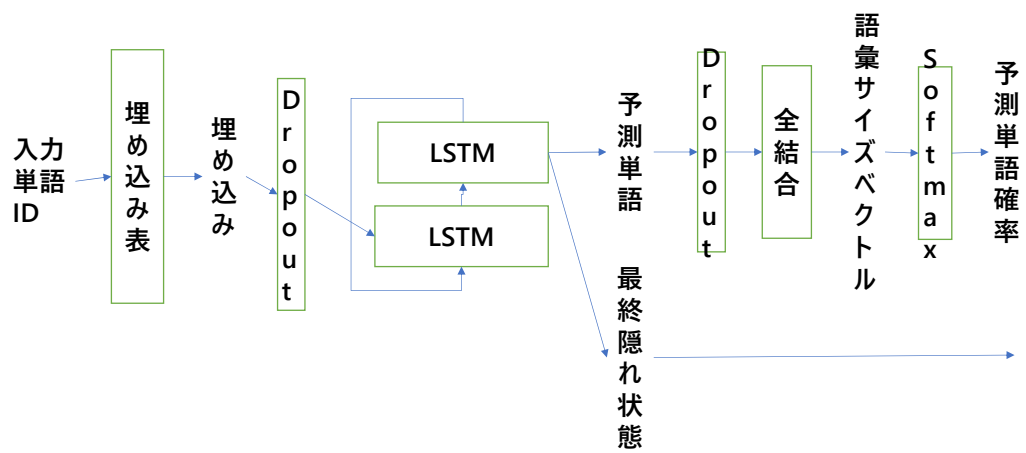
GRUは、LSTMのように、隠れ状態ベクトルとは別に長期記憶ベクトルを持つのではなく、

隠れ状態ベクトルの計算時に加算を使い、隠れ状態自体の記憶機能を強化したものです。

RNN課題3 (Option)

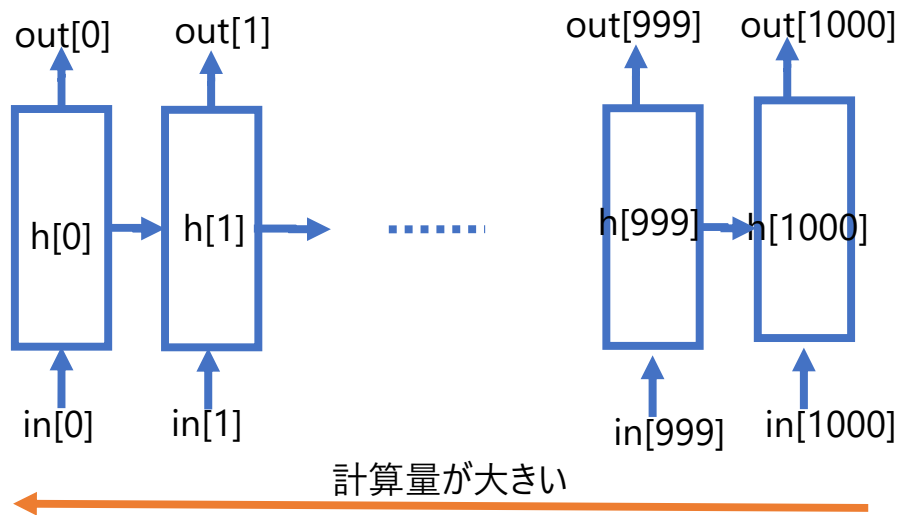
- `rnn_language_model.ipynb` があります。これもPyTorchのサンプルを単純化したものです。
- 単語列を学習させて、その後、適当な先頭単語を与えて作文させるものです。
- 読解しましょう。コードが複雑なので、オプションです。
- フレームワークを使うと、GRUやLSTMはクラスを呼ぶだけですし、パーツの組み合わせも数行で済みます。が、メソッドの引数の意味を理解しないといけないこと、ニューラルネット開発の実態は、実は多次元データを準備するところに時間がとられること、位を見ておいてください。

ネット構成

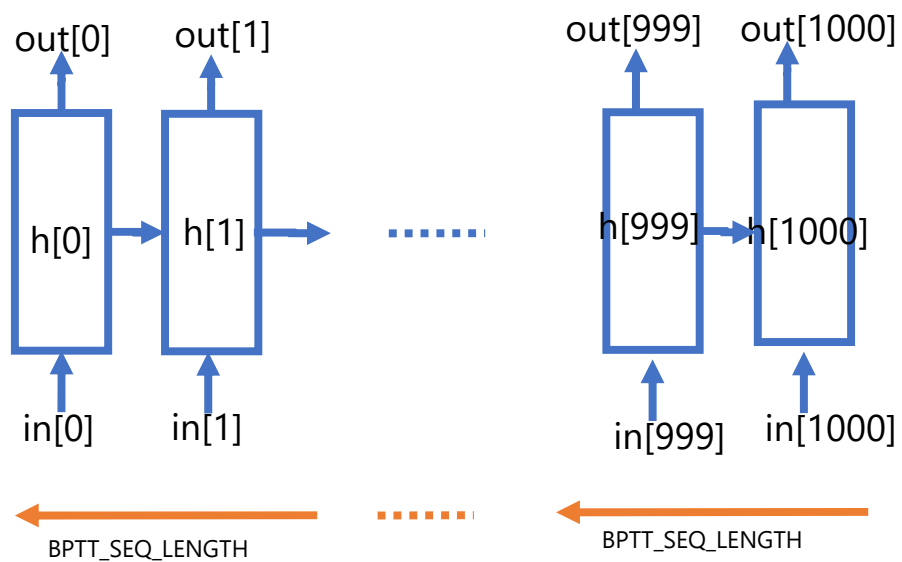


コードを読むときの追加参考情報

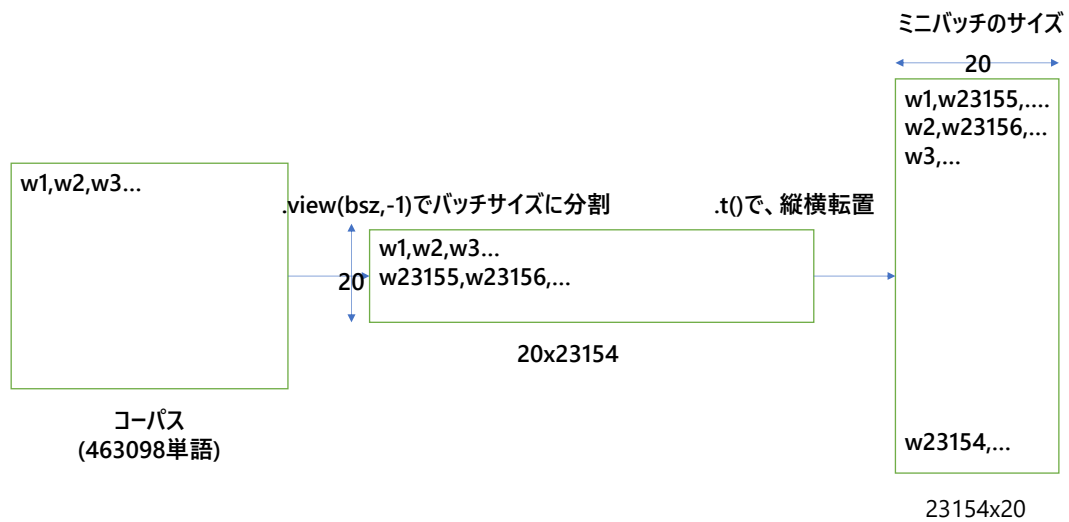
入力が長いと？



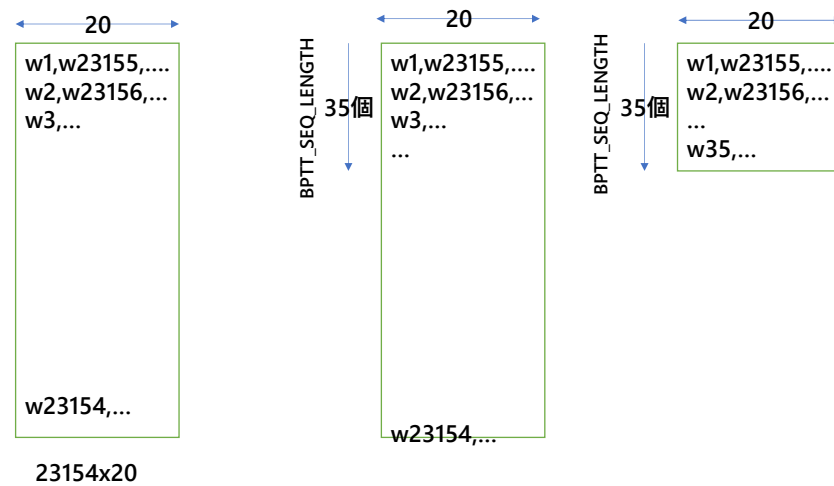
ある固定長で系列処理を分断する



batchify(コーパス、バッチサイズ=20)



get_batch(バッチ行列、開始添え字)



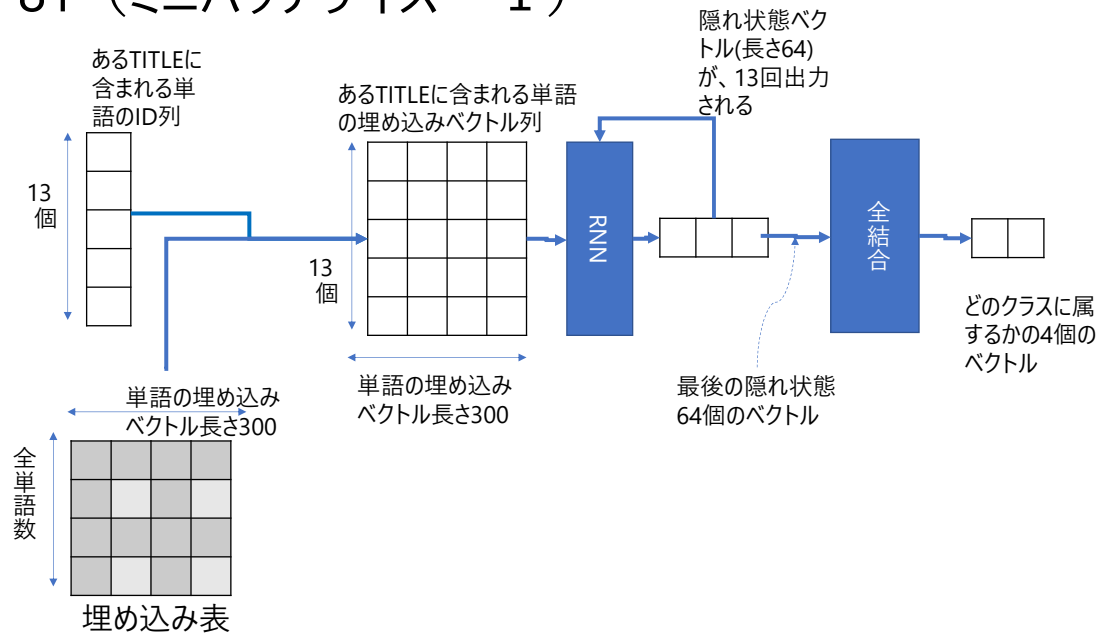
参考資料

- ゼロから作るDeep Learning ② 第6章
- Tutorial記事
 - [Animated RNN, LSTM and GRU](#)
 - [Understanding LSTM Networks](#)
- 教育動画
 - [Deep Learning入門：数式なしで理解するLSTM \(Long short-term memory\)](#)

100本ノック第9章課題80,81,83

- [「100本ノック」の9章の課題](#) の80(データ準備) 、81(RNN)、83(RNN)に取り組みましょう。
- 「NLP、CNNRNNTransformer.ipynb」というノートをコピーしてください。80、81、83のポイントとなる部分を??????にしています。完成させなさい。実行ログを残してください。

81 (ミニバッチサイズ = 1)



確認クイズ

- 確認クイズをやってください。