深層学習（day3）・要点まとめ

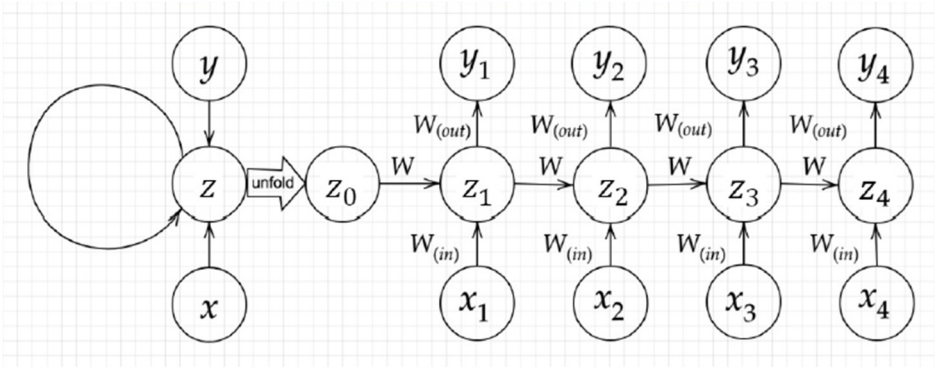
Section0：深層学習全体像の復習　－最新の CNN

《確認テスト》：　サイズ 5×5 の入力画像を、サイズ 3×3 のフィルターで畳み込んだ時の出力画像のサイズは？（ストライドは 2、パディングは 1 とする）

⇒ 出力画像のサイズ＝（入力画像サイズ＋ストライド×2－フィルタサイズ）／ストライド+1
＝（5＋1×2－3）／2＋1＝3

Section1：再帰型ニューラルネットワーク（RNN）の概念

1-1 RNN の全体像



入力（時刻 t）：$u^t = W_{(in)}x^t + Wz^{t-1} + b$

中間層出力：$z^t = f\ (W_{(in)}x^t + Wz^{t-1} + b\ )$

総出力：$v^t = W_{(out)}z^t + c$

出力：$y^t = g\ (W_{(out)}z^t + c\ )$

《確認テスト》：　RNN のネットワークの 3 つの重み･･･1）入力から現在の中間層を定義される時の重み、2）中間層から出力を定義する時の重み、3）もう1つは？

⇒ 1 つ前の時刻における中間層から（時刻 t の中間層への）出力にかけられる重み

- RNN の特徴＝時系列モデルを扱うために、初期の状態と過去の時刻（t-1）の状態を保持し、そこから次の時刻での t を再帰的に求める再帰構造を持つ点。

【RNN の実装の考察（3_1_simple_RNN）】：

- 重み w において、正規分布の初期値の係数を変えてもほとんど変化はない。学習率は 1.0 位に大きくすると正解率の収束速度、精度共に大きく向上する。中間層の数を大きくしても小さくしても収束速度、精度共に悪くなる。中間層の数は 16 位が適度なモデルなのだろう。
- 重みの初期値を正規分布から Xavier や He に変えてもあまり変化はないようである。

```
# ウェイト初期化 (バイアスは簡単のため省略)
# Xavier
W_in = np.random.randn(input_layer_size, hidden_layer_size)
    / (np.sqrt(input_layer_size))
W_out = np.random.randn(hidden_layer_size, output_layer_size)
    / (np.sqrt(hidden_layer_size))
```

深層学習（day3）・要点まとめ

```
W = np.random.randn(hidden_layer_size, hidden_layer_size)
     / (np.sqrt(hidden_layer_size))

# He
W_in = np.random.randn(input_layer_size, hidden_layer_size)
     / np.sqrt(input_layer_size) * np.sqrt(2)
W_out = np.random.randn(hidden_layer_size, output_layer_size)
     / np.sqrt(hidden_layer_size) * np.sqrt(2)
W = np.random.randn(hidden_layer_size, hidden_layer_size)
    / np.sqrt(hidden_layer_size) * np.sqrt(2)
```

- 中間層の活性化関数をシグモイドから ReLU（重みの初期値 He）に変えると勾配爆発が起こる。
  ⇒ ＋方向に大きくなり過ぎることが原因と考えられる。
- 中間層の活性化関数を tanh（重みの初期値 Xavier）に変えて、学習率を 0.1 に小さくすると、正解率の収束速度、精度共に大きく向上し、最も相性が良いようである。（tanh は numpy の関数を利用、導関数は作成） ⇒ RNN には tanh 関数が合っているのかもしれない。

```
# tanh 関数の導関数
def d_tanh(x):
    dx = 1.0 - np.tanh(x)**2
    return dx
```

《コード演習問題》： ニューラルネットワークによって、構文木を入力として再帰的に文全体の表現ベクトルを得るプログラムにおける学習のコードは？

⇒ 隣接単語（表現ベクトル）から表現ベクトルを作るという処理は、隣接している表現 left と right を合わせたものを特徴量として、そこに重みを掛けることで実現する。すなわち、`W.dot(np.concatenate([left, right]))`

[考察]

- 入力文は隣接する単語を木構造で表現し、単語間の関係を表現ベクトルに更新によって学習している。

1-2 BPTT

BPTT の数学的記述 1	コード（3_1_simple_RNN…勾配更新）
$\frac{\partial E}{\partial W_{(in)}}=\frac{\partial E}{\partial u^t}\left[\frac{\partial u^t}{\partial W_{(in)}}\right]^T=\delta^t[x^t]^T \leftarrow \frac{\partial u^t}{\partial W_{(in)}}=x^t$	`np.dot(X.T, delta[:,t].reshape(1,-1))`
$\frac{\partial E}{\partial W_{(out)}}=\frac{\partial E}{\partial v^t}\left[\frac{\partial v^t}{\partial W_{(out)}}\right]^T=\delta^{out,t}[z^t]^T \leftarrow \frac{\partial v^t}{\partial W_{(out)}}=z^t$	`np.dot(z[:,t+1].reshape(-1,1),` `  delta_out[:,t].reshape(-1,1))`
$\frac{\partial E}{\partial W_{(out)}}=\frac{\partial E}{\partial v^t}\left[\frac{\partial v^t}{\partial W_{(out)}}\right]^T=\delta^{out,t}[z^t]^T$	`np.dot(z[:,t].reshape(-1,1),` `  delta[:,t].reshape(1,-1))`
$\frac{\partial E}{\partial b}=\frac{\partial E}{\partial u^t}\frac{\partial u^t}{\partial b}=\delta^t, \frac{\partial u^t}{\partial b}=1 \rightarrow \frac{\partial E}{\partial u^t}=\delta^t$	コード記載なし （simple RNN コードでは簡略化のため省略）
$\frac{\partial E}{\partial c}=\frac{\partial E}{\partial v^t}\frac{\partial v^t}{\partial c}=\delta^{out,t}, \frac{\partial v^t}{\partial c}=1 \rightarrow \frac{\partial E}{\partial v^t}=\delta^{out,t}$	コード記載なし （simple RNN コードでは簡略化のため省略）

深層学習（day3）・要点まとめ

$\frac{\partial E}{\partial u^t}=\frac{\partial E}{\partial v^t}\frac{\partial v^t}{\partial u^t}=\frac{\partial E}{\partial v^t}\frac{\partial W_{(out)}f(u^t)+c}{\partial u^t}=f(u^t)W_{(out)}^T\delta^{out,t}=\delta^t$
$\delta^{t-1}=\frac{\partial E}{\partial u^{t-1}}=\frac{\partial E}{\partial u^t}\frac{\partial u^t}{\partial u^{t-1}}=\delta^t\left\{\frac{\partial u^t}{\partial z^{t-1}}\frac{\partial z^{t-1}}{\partial u^{t-1}}\right\}=\delta^t\{Wf'(u^{t-1})\}$
	`delta[:,t] = (np.dot(delta[:,t+1].T, W.T)`
	`+ np.dot(delta_out[:,t].T, W_out.T))`
	`* functions.d_sigmoid(u[:,t+1])`

BPTT の数学的記述 2	コード（3_1_simple_RNN･･･時系列ループ）
入力（時刻 t）： $u^t=W_{(in)}x^t+Wz^{t-1}+b$	`u[:,t+1] = np.dot(X, W_in) + np.dot(z[:,t]` `.reshape(1, -1), W)`
中間層出力： $z^t=f(W_{(in)}x^t+Wz^{t-1}+b)$	`z[:,t+1] = functions.sigmoid(u[:,t+1])`
総出力： $v^t=W_{(out)}z^t+c$	`np.dot(z[:,t+1].reshape(1, -1), W_out)`
出力： $y^t=g(W_{(out)}z^t+c)$	`z[:,t+1] = functions.sigmoid(u[:,t+1])`

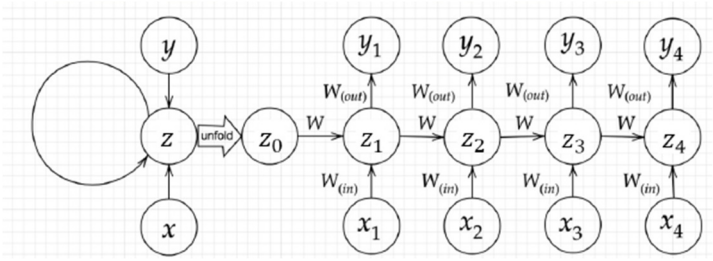
《確認テスト》： 下図のy_1をx・z_0・z_1・$W_{(in)}$・W・$W_{(out)}$を用いて数式で表せ。（バイアスは任意の文字で定義せよ） また中間層の出力にシグモイド関数$g(x)$を作用させよ。

⇒ $z_1=g(W_{(in)}x_1+Wz_0+b)$

$y_1=g(W_{(out)}z_1+c)$

【考察】

- 中間層の出力z_1は、入力x_1（重み$W_{(in)}$との積による作用）と 1 つ前の時刻の中間層の出力z_0（重みWとの積による作用）の和に対して活性化関数を作用させたものとの理解をすることがポイント。



BPTT の数学的記述 3：パラメータ更新	コード（3_1_simple_RNN･･･勾配更新）
$W_{(in)}^{t+1}=W_{(in)}^t-\varepsilon\frac{\partial E}{\partial W_{(in)}}=W_{(in)}^t-\varepsilon\sum_{z=0}^{T^t}\delta^{t-z}[x^{t-z}]^T$	`W_in -= learning_rate * W_in_grad`
$W_{(out)}^{t+1}=W_{(out)}^t-\varepsilon\frac{\partial E}{\partial W_{(out)}}=W_{(out)}^t-\varepsilon\delta^{out,t}[z^t]^T$	`W_out -= learning_rate * W_out_grad`
$W^{t+1}=W^t-\varepsilon\frac{\partial E}{\partial W}=W^t-\varepsilon\sum_{z=0}^{T^t}\delta^{t-z}\left[z^{t-z-1}\right]^T$	`W -= learning_rate * W_grad`
$b^{t+1}=b^t-\varepsilon\frac{\partial E}{\partial b}=b^t-\varepsilon\sum_{z=0}^{T^t}\delta^{t-z}$	コード記載なし （simple RNN コードでは簡略化のため省略）
$c^{t+1}=c^t-\varepsilon\frac{\partial E}{\partial c}=c^t-\varepsilon\delta^{out,t}$	コード記載なし （simple RNN コードでは簡略化のため省略）

《コード演習問題》： BPTT において、誤差関数の中間層の値による微分値 δ を求めるコードは？

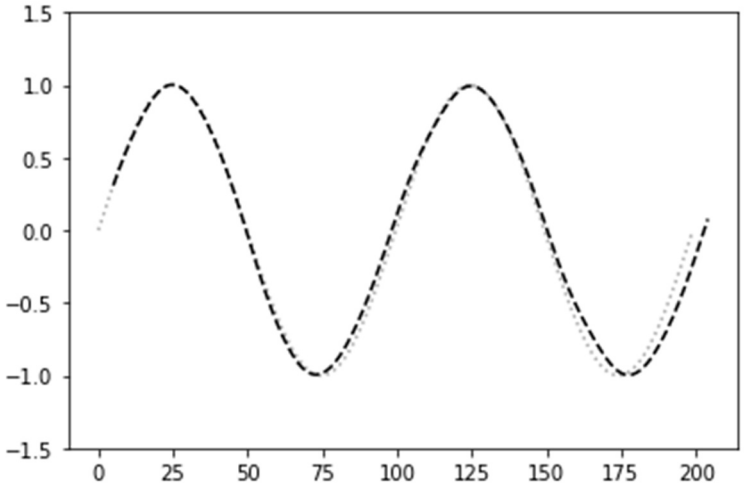
⇒RNN において、損失関数を重み W や U に関して偏微分する時は、中間値出力$u(t-1)\cdots u(1)$に依存するため、$\frac{\partial u^t}{\partial u^{t-1}}=u$ となる。従って、$\delta^{t-1}=\frac{\partial E}{\partial u^{t-1}}=\frac{\partial E}{\partial u^t}\frac{\partial u^t}{\partial u^{t-1}}=\delta^t\frac{\partial u^t}{\partial u^{t-1}}=\delta^t u$ である。これをコードで表現すると、`delta _t = delta_t.dot(U)` となる。

深層学習（day3）・要点まとめ

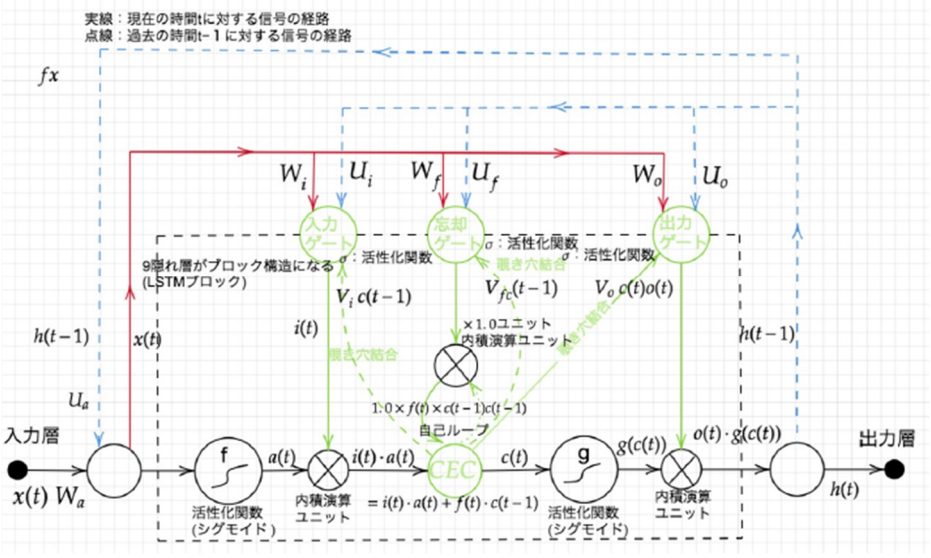
【RNN･BPTT の実装の考察（3_3_predict_sin）】：

- エポック数が 100 回と少ない場合は、予測精度は極めて低い。
- 一つの時系列データの長さを 5 に増やし、エポック数を 500 回にすると、時間遅れはあるものの予測精度は向上する。更に、エポック数を 3000 回に増やすと、予測精度は極めて高くなり、元の sin 曲線にほぼ一致している。

```
loss: 1.1452214926872929e-08 d: [-0.98504973] y: [-0.98489839]
loss: 5.3815470676520537e-08 d: [-0.64332332] y: [-0.6436514]
loss: 1.0539082479501495e-07 d: [0.43226238] y: [0.43180327]
loss: 3.266434009851389e-08 d: [-0.81380058] y: [-0.81405617]
```



Section2：LSTM



5-1. CEC（Constant Error Carousel）

RNN の課題である勾配消失や勾配爆発を解決するため、中間層に設けられた LSTM ブロックのメモリで勾配を 1 とする。

《演習チャレンジ》： 勾配爆発に対応るための勾配クリッピングのコードは？

深層学習（day3）・要点まとめ

⇒ 勾配のノルムが閾値より大きい時は、勾配のノルムを閾値に正規化するため、クリッピングした勾配は、勾配×（閾値／勾配のノルム）と計算される。

```
# 勾配クリッピング
def gradient_clipping(grad, threshold):
    # grad: gradient
    norm = np.linarg.norm(grad)
    rate = threshold / norm
    if rate < 1:
        return grad * rate
    return grad
```

5-2. 入力ゲートと出力ゲート

【CEC の課題】 入力データについて、時間依存度に関係なく重みが一律なため、ニューラルネットワークの学習特性が無い。（入力層→隠れ層への重み：入力重み衝突、隠れ層→出力層への重み：出力重み衝突）

⇒ 入力・出力ゲート： それぞれのゲートの重みを、重み行列 W, U で可変可能とする。

5-3. 忘却ゲート

【LSTM ブロックの課題】 CEC は過去の情報を全て保管するため、過去の情報が要らなくなった場合、削除できず保管され続ける。

⇒ 過去の情報が要らなくなった場合、そのタイミングで情報を忘却する機能を設ける。＝**忘却ゲート**

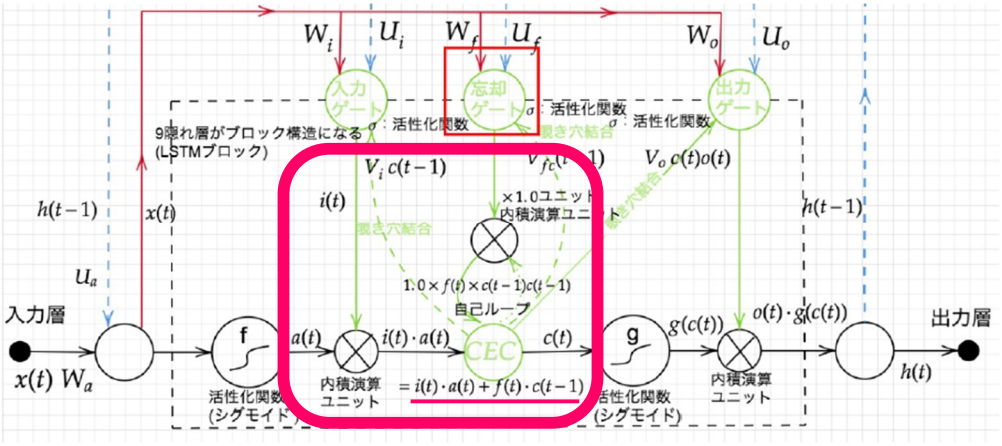
《確認テスト》：「映画おもしろかったな。ところで、とてもお腹が空いたから何か____。」の文章を LSTM に入力し、次の単語を予測したい。文中の「とても」という言葉は予測において無くなっても影響を及ぼさないと考えられる。このような場合、どのゲートが作用すると考えられるか？

⇒ 周辺語との関連度において重要度が低く、無くしても問題ないとする処理なので忘却ゲートが作用すると考えられる。

《演習チャレンジ》： LSTM の順伝搬プログラムにおいて、メモリセル（CEC）の更新式のコードは？

⇒ 以下の図のように、新しいセルの状態は、計算されたセルへの入力と 1 ステップ前のセルの状態に入力ゲート、忘却ゲートを掛けて足し合わせたものとなる。そのため、

`input_gate * a + forget_gate * c` となる。

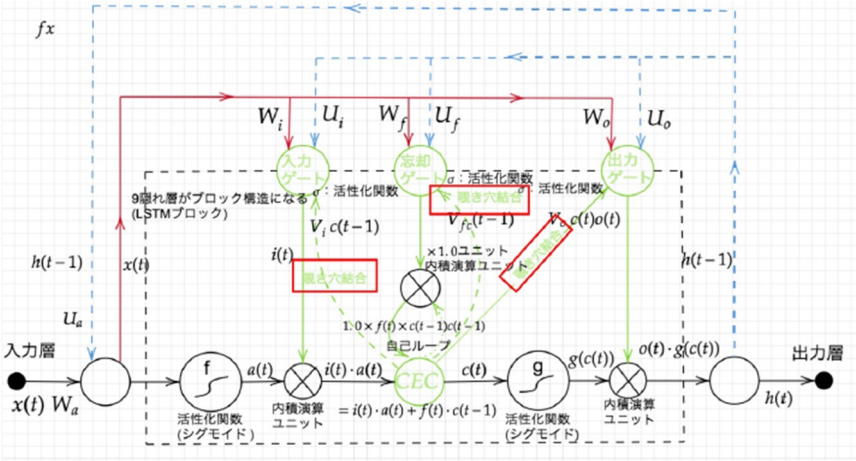


深層学習（day3）・要点まとめ

5-4. 覗き穴結合

【課題】 CEC の保存されている過去の情報を、任意のタイミングで他のノードに伝播させたり、任意のタイミングで忘却させたい。CEC 自身の値は、ゲート制御に影響を与えていない。

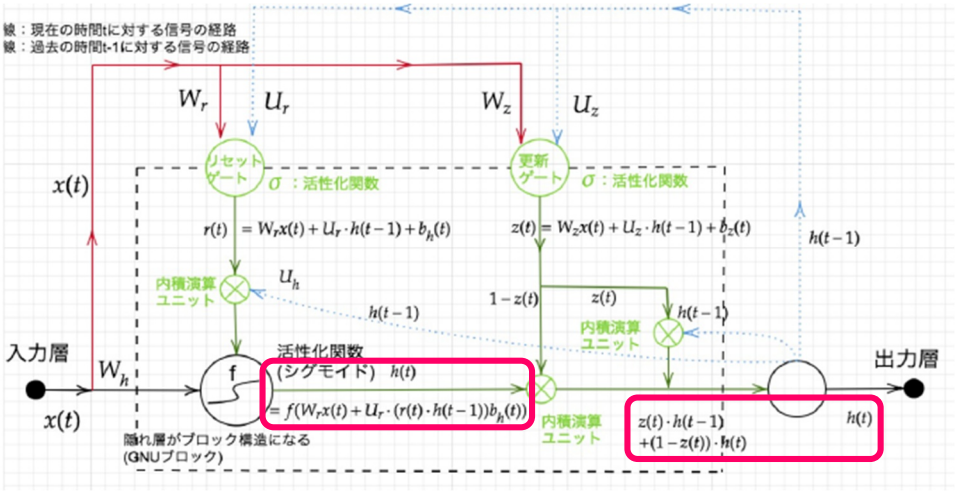
⇒ CEC 自身の値に、重み行列を介して伝搬可能にする構造を設ける。



Section3：GRU

【LSTM の課題】

LSTM ではパラメータ数が多く、計算負荷が高くなる。 ⇒ GRU



《確認テスト》： LSTM と CEC が抱える課題は何か？

⇒ LSTM の課題＝パラメータ数が多く、計算負荷が高い。

CEC の課題＝入力データについて、時間依存度に関係なく重みが一律なため、ニューラルネットワークの学習特性が無い。

《演習チャレンジ》： GRU の順伝搬を行うコードは？

⇒ 新しい中間状態 h_new は、1ステップ前の中間表現 h_bar と計算された中間表現 h の線形和で表現される。上図において、h_bar⇒h(t-1), h_new⇒h(t) と読み替えると、以下になる。

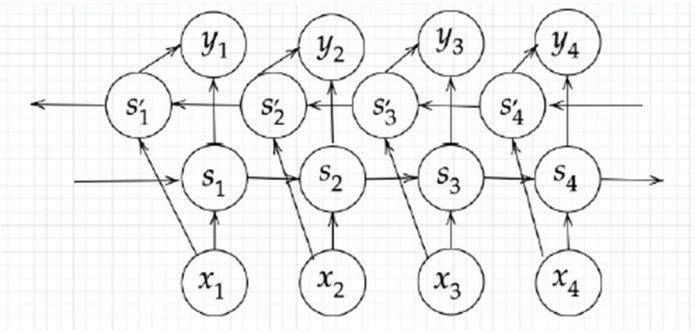
```
h_new = (1-z) * h + z * h_bar
```

深層学習（day3）・要点まとめ

《確認テスト》： LSTM と GRU の違いは何か？

⇒ LSTM の方がパラメータ数が多いため、GRU の方が高速で簡単に実行できる。多くの場合、LSTM のように GRU は機能するが、実際には GRU の表現力は豊かではないので、対象により使い分ける必要がある。

Section4：双方向 RNN



• 過去の情報だけでなく、未来の情報を加味することで、精度を向上させるためのモデル
実用例：文章の推敲、機械翻訳など

《演習チャレンジ》： 双方向 RNN の順伝搬を行うコードは？

⇒ 双方向 RNN は、順方向と逆方向に伝播した時の中間層表現を合わせたものが特徴量となる。出力は、ys = hs.dot(V.T) で計算されるが、V^T は（2*中間層のサイズ，出力のサイズ）なので、hs は（入力のサイズ，2*中間層のサイズ）でなければならず、コードとしては 1 軸での concatenate であるとわかる。従って、`np.concatenate([h_f, h_b[::-1]], axis=1)` となる。

＜RNN での自然言語処理＞

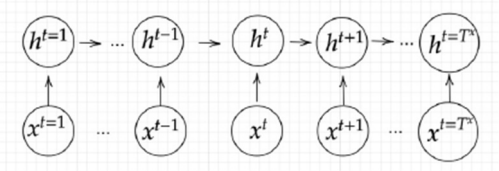
Section5：Seq2Seq

Encoder-Decorder モデルの一種

⇒ 機械対話、機械翻訳

5-1. Encoder RNN

システムが出力したデータを、単語等のトークン毎に生成する構造



昨日　食べた　刺身　大丈夫　でしたか　。

①Taking： 文章を単語などのトークン毎に分割し、トークン毎の ID に分割する

②Embedding： ID から、そのトークンを表す分散表現ベクトルに変換

③Encoder RNN： ベクトルを順番に RNN に入力していく

- vec1 を RNN に入力し、hidden state を出力。この hidden state と次の入力 vec2 をまた RNN に入力し、hidden state を出力するという流れを繰り返す。

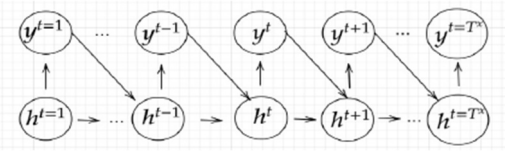
深層学習（day3）・要点まとめ

- 最後の vec を入れた時の hidden state を final state としてとっておく。この final state が thought vector と呼ばれ、入力した文の意味を表すベクトルとなる。

5-2. Decoder RNN

システムが出力したデータを、単語等のトークン毎に生成する構造

お腹　が　痛い　です　。



①Decoder RNN：Encoder RNN の final state（thought vector）から、各トークンの生成確率を出力していく。final state を Decoder RNN の initial state として設定し、Embedding を入力。
②Sampling：生成確率に基づいてトークンをランダムに選ぶ
③Embedding：②で選ばれたトークンを Embedding して Decoder RNN への次の入力とする。
④Detokenize：①から③を繰り返し、②で得られたトークンを文字列に直す。

《確認テスト》：Seq2seq の説明で正しいものは？
⇒ (2)“Encoder-Decoder モデルの一種で、機械翻訳などのモデルに使われる”が正しい。
(1)は双方向 RNN、(3)は Encoder RNN、(4)は LSTM を表している。

《演習チャレンジ》：入力である文（文章）を時系列の情報を持つ特徴量へエンコードする関数において、入力 w に対する最初の処理コードは？
⇒ 単語 w は one hot ベクトルであり、それを単語埋め込みにより別の特徴量に変換する。これは埋め込み行列 E（埋め込みサイズ，単語の数）を用いて、E.dot(w)となる

5-3. HRED

5-4. VHRED

《確認テスト》：Seq2seq と HRED、HRED と VHRED の違いは？

技術	説明（解決方法）	課題
Seq2seq	“Encoder-Decoder モデルで、機械翻訳等の会話対会話に対応する	一問一答しかできず、問に対して文脈を考慮せず、ただ応答を続けるだけ⇒HRED
HRED	過去の会話から単語の流れに即して応答し、より人間らしい文章が生成される	字面に確率的な多様性がなく、会話の「流れ」に合わせた多様性がない⇒VHRED
VHRED	VAE の潜在変数の概念を追加し、会話の「流れ」に合わせて多様性を持たせた	

5-5. VAE

VAE は、オートエンコーダーの潜在変数に**確率分布 z～N（0,1）**を仮定したもの ⇒ 《確認テスト》

【考察】
平均と分散を制御することによって、元のデータから自由に変化させることが可能となる

深層学習（day3）・要点まとめ

Section6：Word2vec

- RNN の課題：単語のような可変長の文字列を NN に与えることはできない ⇒ 固定長形式で単語を表す必要がある
- ボキャブラリ×任意の単語ベクトル次元の重み行列で学習できるようになり、大規模データの分散表現の学習が現実的な計算速度とメモリ量で実現可能になった。

Section7：Attention Mechanism

- Seq2seq は RNN を用いているため、2 単語でも 100 単語でも、固定次元ベクトルの中に入力しなければならず、長い文章への対応が困難
- 文章が長くなるほどそのシーケンスの内部表現の次元も大きくなっていく仕組みが必要

⇒ **Attention Mechanism**：「入力と出力のどの単語が関連しているのか」の関連度を学習する仕組み

《確認テスト》： RNN と word2vec、Seq2seq と Seq2seq+Attention の違いは？

⇒ 1) 大規模データの分散表現において、RNN はボキャブラリ×ボキャブラリの重み行列が必要なのに対して、word2vec はボキャブラリ×任意の単語ベクトル次元の重み行列で対応できるため、word2vec では現実的な計算速度とメモリ量で学習が実現可能になった点である。

2) Seq2seq は文脈の「流れ」に合わせた出力ができないのに対して、Attention を組み合わせることによって「入力と出力のどの単語が関連しているのか」の関連度を学習でき、機械翻訳などの精度が大きく向上する点である。

【考察】

word2vec による単語の分散表現と Attention による単語間の関連度の学習が、自然言語処理の精度向上に大きく貢献したと考えられる。