深層学習 Day3 レポート

復習：

[サイズ 5 × 5 の入力画像を、サイズ 3 × 3 のフィルタで畳み込んだ時の出力画像のサイズを答えよ。なおストライドは 2 、パディングは 1]

解答：3x3　← {(5+1*2)-3}/2+1

Section1：再帰型ニューラルネットワーク（RNN）の概念

RNN とは？

時系列データに対応可能な、ニューラルネットワークである。

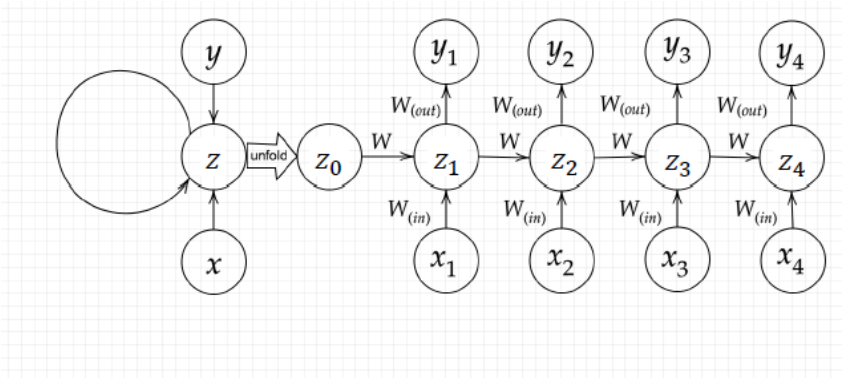
では、時系列データ（音声やテキスト）とは？

時間的順序を追って一定間隔ごとに観察され、しかも相互に統計的依存関係が認められるようなデータの系列を指す。

RNN では中間層の出力が出力層だけでなく、また中間層へ入力する点が特徴的。

イメージ化した図が下記となる。

ある時刻 t1 での入力は、x1 だけではなく、前の時間 t0 の中間層の出力 z0 も入力となる。



RNN の数学的記述

$u^t = W_{(in)}x^t + Wz^{t+1} + b$　　・・・　式①

$z^t = f\left(W_{(in)}x^t + Wz^{t+1} + b\right)$

$v^t = W_{(out)}z^t + c$

$y^t = g\left(W_{(out)}z^t + c\right)$　　・・・　式②

→ u[:,t+1] = np.dot(X, W_in) + np.dot(z[:,t].reshape(1,1), W)　※式①

z[:,t+1] = functions.sigmoid(u[:,t+1])　※式②

BPTT とは？

RNN においてのパラメータ調整方法の一種

BPTT の数学的記述

$$\frac{\partial E}{\partial W_{(in)}} = \frac{\partial E}{\partial u^t}\left[\frac{\partial u^t}{\partial W_{(in)}}\right]^T = \delta^t\left[x^t\right]^T$$

$$\frac{\partial E}{\partial W_{(out)}} = \frac{\partial E}{\partial v^t}\left[\frac{\partial v^t}{\partial W_{(out)}}\right]^T = \delta^{out,t}\left[z^t\right]^T$$

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial u^t}\left[\frac{\partial u^t}{\partial W}\right]^T = \delta^t\left[z^{t-1}\right]^T$$

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial u^t}\frac{\partial u^t}{\partial b} = \delta^t$$

$$\frac{\partial E}{\partial c} = \frac{\partial E}{\partial v^t}\frac{\partial v^t}{\partial c} = \delta^{out,t}$$

※転置の T ではなく、時間をさかのぼる T

・確認テスト

[RNN のネットワークには大きくわけて 3 つの重みがある。1 つは入力から現在の中間層を定義する際にかけられる重み、1 つは中間層から出力を定義する際にかけられる重みである。残り 1 つの重みについて説明せよ。]

解答：前の中間層からの出力にかけられる重み

[連鎖律の原理を使い、dz/dx を求めよ。]

$z = t^2$

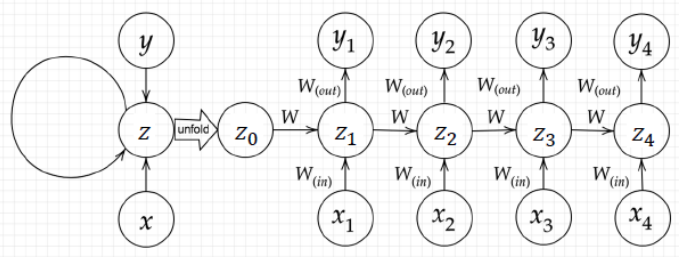
$t = x + y$

解答：$\frac{dz}{dx} = 2(x+y) \times 1 = 2(x+y)$

[下図の y 1 を x ・ s 0 ・ s 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。]



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

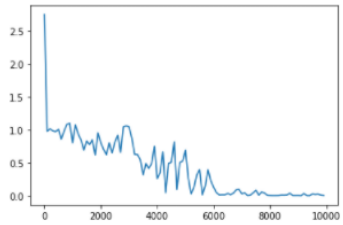
$$z_1 = W_{(in)} \cdot x_1 + W \cdot z_0 + b$$

シグモイド関数内のバイアス項を忘れていた。注意が必要。

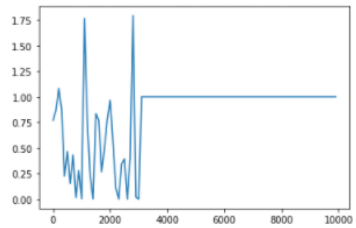
実装演習

3_1_simple_RNN.ipynb

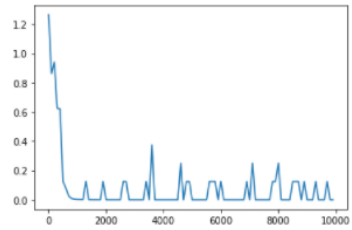
① 中間層シグモイド関数



② He + 中間層 ReLU 関数



③ Xavier + 中間層 tanh 関数



誤差関数は MSE のため、①と②は学習がうまくいっているようだ。

ReLU 関数が誤差 1 で収束したのは、なぜだろうか。

コード演習問題

```
def bptt(xs, ys, W, U, V):
    """
    ys: labels, (batch_size, n_seq, output_size)
    """
    # 順伝播
    hiddens, outputs = rnn_net(xs, W, U, V)
    # 損失関数のW, Y, Vに関する偏微分値
    dW = np.zeros_like(W)  # dL/dW
    dU = np.zeros_like(U)  # dL/dU
    dV = np.zeros_like(V)  # dL/dV
    # 損失関数の出力値に関する偏微分値
    do = _calculate_do(outputs, ys)  # dL/do, (batch_size, n_seq, output_si

    batch_size, n_seq = ys.shape[:2]
    # 時間を逆方向にたどり、パラメータの偏微分値を計算（バックプロパゲーション）
    # dL/dV = do/dV·dL/do = h·dL/do
    # dL/dW = do/dW·dL/do
    # dL/dU = do/dU·dL/do
    # do/dW = (dh_{t}/dW·d/dh_{t}+dh_{t-1}/dW·d/dh_{t-1}+···)o_{t}
    #       = (x_{t}+x_{t-1}·U+x_{t-2}·U^2+···)·V
    # do/dU = (dh_{t}/dU·d/dh_{t}+dh_{t-1}/dU·d/dh_{t-1}+···)o_{t}
    #       = (h_{t-1}+h_{t-2}·U+h_{t-3}·U^2+···)·V
    for t in reversed(range(n_seq)):
        dV += np.dot(do[:, t].T, hiddens[:, t]) / batch_size
        delta_t = do[:, t].dot(V)
        # 時間tの出力は時間t以前の中間層すべてに依存するため
        # W, Uはさらに遡って計算
        for bptt_step in reversed(range(t+1)):
            dW += np.dot(delta_t.T, xs[:, bptt_step]) / batch_size
            dU += np.dot(delta_t.T, hiddens[:, bptt_step-1]) / batch_size
            delta_t = [redacted]
    return dW, dU, dV
```

(お)

上の図は BPTT を行うプログラムである。なお簡単化のため活性化関数は恒等関数であるとする。また、calculate_dout 関数は損失関数を出力に関して偏微分した値を返す関数であるとする。
（お）にあてはまるのはどれか。

1 delta_t.dot(W)
2 delta_t.dot(U)
3 delta_t.dot(V)
4 delta_t * V
解答： 2

$$\delta^{t-1} = \frac{\partial E}{\partial u^{t-1}} = \frac{\partial E}{\partial u^{t}} \frac{\partial u^{t}}{\partial u^{t-1}}$$

中間層から中間層への出力となる。h_{t}は過去の h_{t-1}以前に依存する。
RNN において損失関数を重み W や U に関して偏微分するときは、
それを考慮する必要がある。
dh_{t}/dh_{t-1} = U であることに注意する。

演習チャレンジ
以下は再帰型ニューラルネットワークにおい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である。ただし、ニューラルネットワークの重みパラメータはグローバル変数として定義してあるものとし、 _activation 関数はなんらかの活性化関数であるとする。木構造は再帰的な辞書で定義してあり、 root が最も外側の辞書であると仮定する。（く）にあてはまるのはどれか。

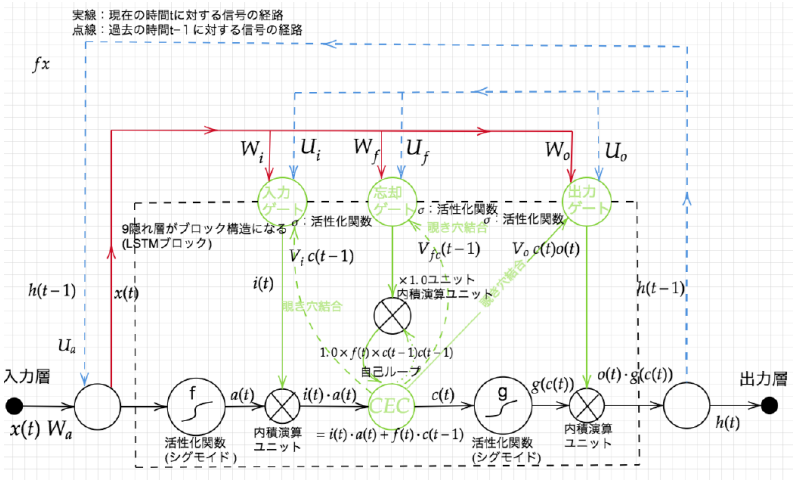
解答：（２）
traverse とは何かと思ったが、def traverse(node):で定義している自身を指すようだ。
RNN を難しいと感じてしまうところ。
np.concatenate は複数の np データを結合する関数となる。

Section2：LSTM
RNN の課題：時系列を遡れば遡るほど、勾配が消失していき、長い時系列の学習が困難
解決策：勾配消失の解決方法とは別で、構造自体を変えて解決したものが LSTM 。

勾配消失しやすい活性化関数：シグモイド関数
大きな値では出力の変化が微小なため、勾配消失問題 を引き起こす事があった。

勾配爆発とは？
勾配が、層を逆伝播するごとに指数関数的に大きくなっていく状況。
その結果学習が進まないため、勾配消失と併せて学習問題となる。

LSTM の全体図



CEC とは
CEC は RNN で言うところの中間層となる。RNN では中間層にシグモイド関数などを使用しているため、勾配消失が発生する。そこで、記憶と学習を分離したのが CEC となる。
勾配消失および勾配爆発の解決方法として、勾配が、1 であれば解決できる。

$$\delta^{t-z-1} = \delta^{t-z}\left\{Wf'\left(u^{t-z-1}\right)\right\} = 1$$

$$\frac{\partial E}{\partial c^{t-1}} = \frac{\partial E}{\partial c^{t}}\frac{\partial c^{t}}{\partial c^{t-1}} = \frac{\partial E}{\partial c^{t}}\frac{\partial}{\partial c^{t-1}}\{a^{t} - c^{t-1}\} = \frac{\partial E}{\partial c^{t}}$$

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である。
→ このままでは学習しないネットワークとなる。
そこで、CEC の周辺に学習機能を持たせた工夫が行われた。

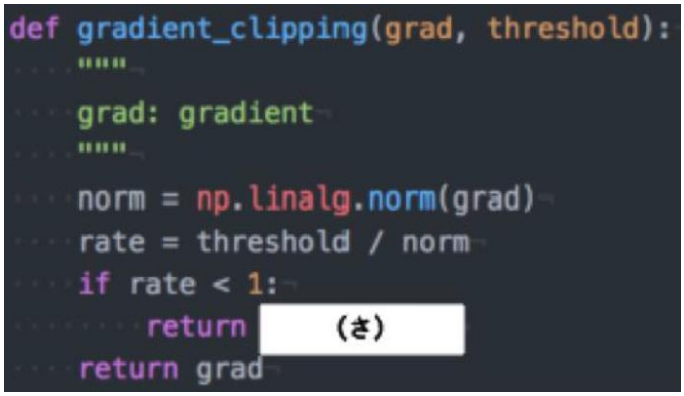
入力ゲート/出力ゲート：重み W、前回の出力 U を学習する。
入力・出力ゲートを追加することで、それぞれのゲートへの入力値の重みを、
重み行列 W,U で可変可能とする。
このゲートにより CEC の課題を解決している。

忘却ゲート：
CEC は、過去の情報が全て保管されている 。
あまりにも古い情報が作用する懸念があり、いらなくなった情報を削除する必要がある。
この問題を忘却ゲートは解決している。

覗き穴結合：
CEC 自身の値に、重み行列を介して伝播可能にした構造 。

確認テスト
[シグモイド関数を微分した時、入力値が 0 の時に最大値をとる。
その値として正しいものを選択肢から選べ。]
(1) 0.15
(2) 0.25 解答（day2 レポート済み）
(3) 0.35
(4) 0.45

演習チャレンジ
RNN や深いモデルでは勾配の消失または爆発が起こる傾向がある。
勾配爆発を防ぐために勾配のクリッピングを行うという手法がある。
具体的には勾配のノルムがしきい値を超えたら、勾配のノルムをしきい値に正規化するというものである。以下は勾配のクリッピングを行う関数である。
（さ）にあてはまるのはどれか。



(1) gradient * rate 解答
(2) gradient / norm
(3) gradient / threshold
(4) np.maximum (gradient,threshold)

rate では threshold（閾値）/norm（ノルム）で計算済みのため、gradient に掛けることで制限される。

[以下の文章を LSTM に入力し空欄に当てはまる単語を予測したいとする。
文中の「とても」という言葉は空欄の予測においてな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。]
「映画おもしろかったね。ところで、とてもお腹が空いたから何か____ 。」
解答：忘却ゲートと考える。
CEC で記憶した「とても」という言葉が不要な場合に記憶から消す機能であるため。

演習チャレンジ
以下のプログラムは LSTM の順伝播を行うプログラムである。ただし _sigmoid 関数は要素ごとにシグモイド関数を作用させる関数である。
（け）にあてはまるのはどれか。

(1)　output_gate * a + forget_gate * c

(2)　forget_gate * a + output_gate * c

(3)　input_gate * a + forget_gate * c　解答

(4)　forget_gate * a + input_gate * c

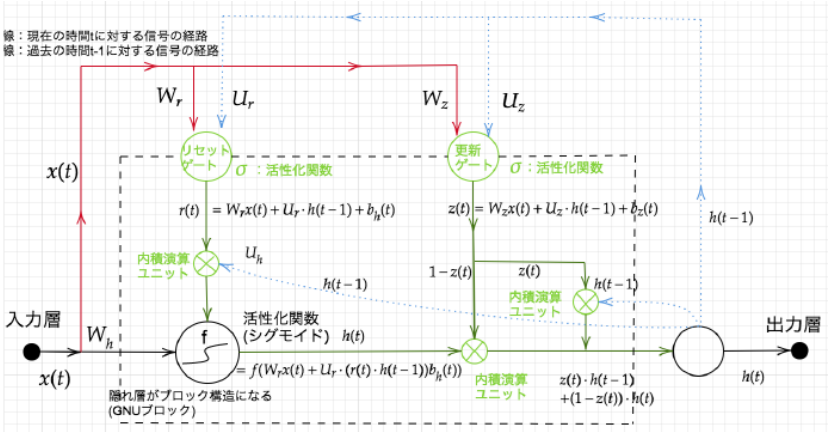
Section3：GRU

GRU は LSTM の改造版となる。

LSTM では、パラメータ数が多く、計算負荷が高くなる問題があった 。

パラメータが多数存在していたため、計算負荷が大きくなる。

そこで GRU では、そのパラメータを大幅に削減し、精度は同等またはそれ以上が望める様になった構造となる。メリットは計算量の削減となる。



確認テスト

[LSTM と CEC が抱える課題について、それぞれ簡潔に述べよ。]

LSTM の問題はパラメータ数の多さによる、学習のパフォーマンスの低さと考えられる。

CEC の問題は、自身では学習ができない点。

演習チャレンジ

GRU(Gated Recurrent Unit)も LSTM と同様に RNN の一種であり、単純な RNN において問題となる勾配消失問題を解決し、長期的な依存関係を学習することができる。 LSTM に比べ変数の数やゲートの数が少なく、より単純なモデルであるが、タスクによっては LSTM より良い性能を発揮する。以下のプログラムは GRU の順伝播を行うプログラムである。ただし _sigmoid 関数は要素ごとにシグモイド関数を作用させる関数である。

（こ）にあてはまるのはどれか。

```
def gru(x, h, W_r, U_r, W_z, U_z, W, U):
    """
    x: inputs, (batch_size, input_size)
    h: outputs at the previous time step, (batch_size, state_size)
    W_r, U_r: weights for reset gate
    W_z, U_z: weights for update gate
    U, W: weights for new state
    """
    # ゲートを計算
    r = _sigmoid(x.dot(W_r.T) + h.dot(U_r.T))
    z = _sigmoid(x.dot(W_z.T) + h.dot(U_z.T))

    # 次状態を計算
    h_bar = np.tanh(x.dot(W.T) + (r * h).dot(U.T))
    h_new = [          (こ)          ]
    return h_new
```

(1)　z * h_bar

(2)　(1 z) * h_bar

(3)　z * h * h_bar

(4)　(1 z) * h + z * h_bar

$$z(t) \cdot h(t-1) \\ +(1-z(t)) \cdot h(t)$$

左式に対応するコードを選択する問題となる。

[LSTM と GRU の違いを簡潔に述べよ 。]

LSTM は入力/出力/忘却の 3 つのゲートで構成。記憶には CEC がある。パラメータ多。

GRU はリセット/更新ゲートの 2 つのゲートで構成。パラメータ数も少ない。

実装演習

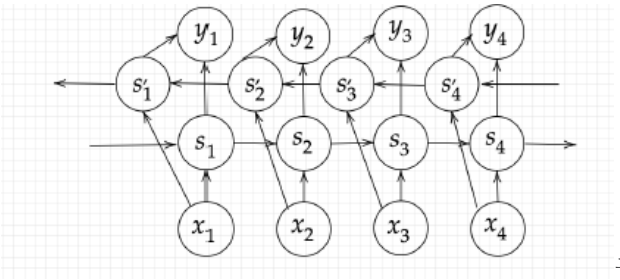
predict_word.ipynb

→ 単語の予測を行っている。5000 行で切り捨てられておりかなりの出力が出ている。

```
ストリーミング出力は最後の 5000 行に切り捨てられました。
beating : 1.3192024e-14
authentic : 1.490178e-14
glow : 1.5293678e-14
oy : 1.468018e-14
emotion : 1.4983662e-14
delight : 1.4004107e-14
nuclear : 1.4859943e-14
dropped : 1.4963213e-14
hiroshima : 1.3928321e-14
beings : 1.582246e-14
tens : 1.5364727e-14
burned : 1.3809626e-14
homeless : 1.5643702e-14
albert : 1.447877e-14
initiated : 1.3293336e-14
bomb : 1.3569621e-14
theoretical : 1.4577083e-14
```

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



予測の左から右への流れ以外に逆方向

確認テスト

演習チャレンジ

以下は双方向 RNN の順伝播を行うプログラムである。順方向については、入力から中間層への重み W_f , 一ステップ前の中間層出力から中間層への重みを U_f 、 逆方向に関しては同様にパラメータ W_b , U_b を持ち、両者の中間層表現を合わせた特徴から出力層への重みは V である。 rnn 関数は RNN の順伝播を表し中間層の系列を返す関数であるとする。(か) にあてはまるのはどれか

```
def bidirectional_rnn_net(xs, W_f, U_f, W_b, U_b, V):
    """
    W_f, U_f: forward rnn weights, (hidden_size, input_size)
    W_b, U_b: backward rnn weights, (hidden_size, input_size)
    V: output weights, (output_size, 2*hidden_size)
    """
    xs_f = np.zeros_like(xs)
    xs_b = np.zeros_like(xs)
    for i, x in enumerate(xs):
        xs_f[i] = x
        xs_b[i] = x[::-1]
    hs_f = _rnn(xs_f, W_f, U_f)
    hs_b = _rnn(xs_b, W_b, U_b)
    hs = [          (か)          for h_f, h_b in zip(hs_f, hs_b)]
    ys = hs.dot(V.T)
    return ys
```

(1) h_f + h_b[:: 1]

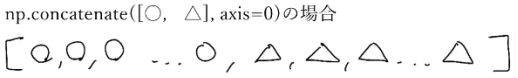
(2) h_f * h_b[:: 1]

(3) np.concatenate([h_f, h_b[:: 1]], axis=0)

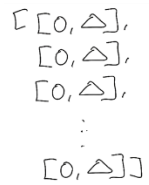
(4) np.concatenate([h_f, h_b[:: 1]], axis=1) 解答：同じ時間のデータをペアにする。

np.concatenate について下記イメージとなる。

np.concatenate([○, △], axis=0)の場合

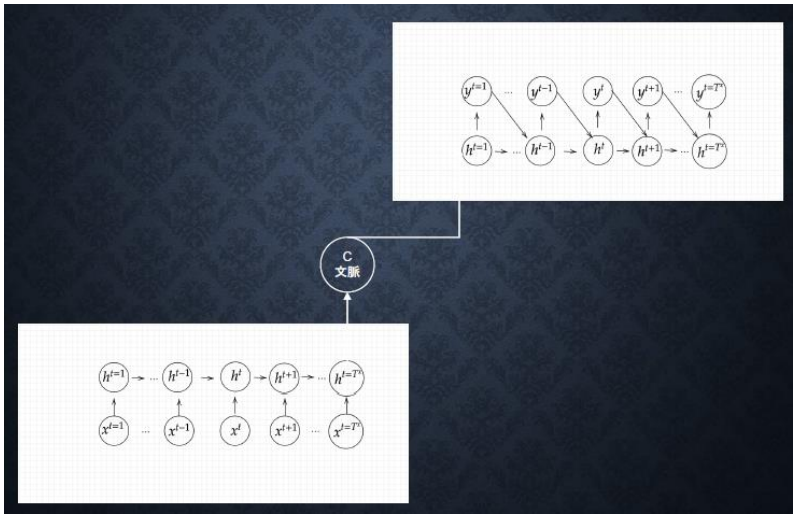


np.concatenate([○, △], axis=1)の場合



Section5：Seq2Seq

Seq2seq とは EncoderDecoder モデルの一種を指します。



右上：デコーダーと呼ぶ（ベクトル表現から別の形へ変換する。）

左下：エンコーダーと呼ぶ（単語が順次入力される。文脈をベクトル表現で理解）

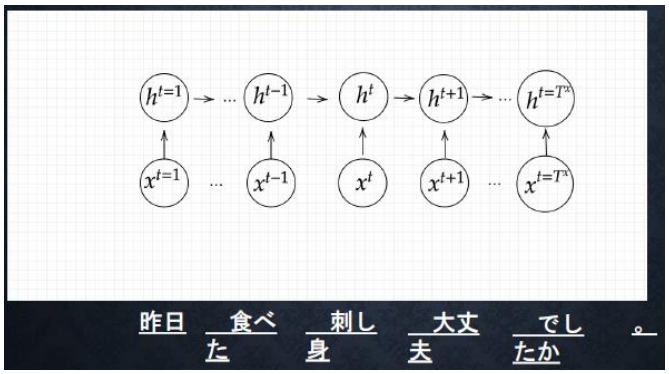
Seq2seq の具体的な用途とは？

機械対話や、機械翻訳などに使用されている。

時系列のデータを出力する。

Encoder RNN

ユーザーがインプットしたテキストデータを、単語等のトークンに区切って渡す構造。



Taking： 文章を単語等のトークン毎に分割し、トークンごとの ID に分割する 。

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

Encoder RNN: ベクトルを順番に RNN に入力していく 。

単語	ID	one-hot (10000)	embedding (～数百)
私	1	[1,0,0 ... 0]	[0.2 0.4 0.6 ... 0.1]
は	2	[0,1,0 ... 0]	[...]
刺身	3	[0,0,1 ... 0]	[...]
昨日	4	[...]	
⋮	⋮		
xxx	10000	[0,0,0 ... 1]	[...]

Embedding 表現は機械学習で減らす。(意味が似ているモノは似たベクトルへ)

Encoder RNN 処理手順

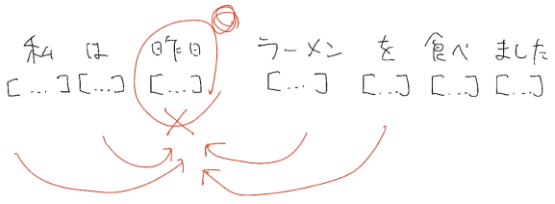
・vec1 を RNN に入力し、 hidden state を出力 。

この hidden state と次の入力 vec2 をまた RNN に入力してきた hidden state を出力という流れを繰り返す 。

・最後の vec を入れたときの 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 。

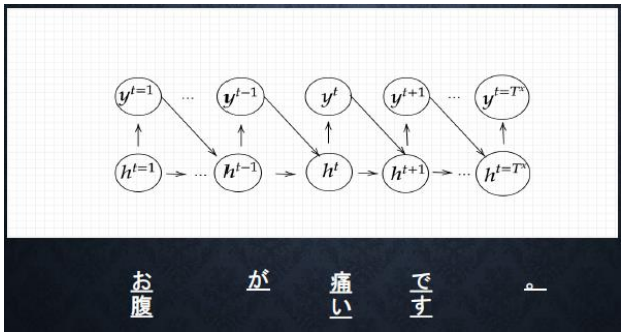
MLM - Masked Language Model



上記のように 1 単語を落として、周辺単語から予測をすることで意味を理解させる。

Decoder RNN

システムがアウトプットデータを、単語等のトークンごとに生成する構造 。

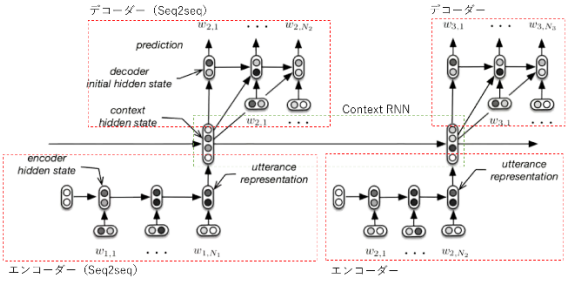


1. Decoder RNN: Encoder RNN の final state (thought vector) から、各 token の生成確率を出力していきます final state を Decoder RNN の initial state ととして設定し、Embedding を入力。
2. Sampling:生成確率にもとづいて token をランダムに選びます 。
3. Embedding:2 で選ばれた token を Embedding して Decoder RNN への次の入力とします 。
4. Detokenize:1 3 を繰り返し 、 2 で得られた token を文字列に直します 。

Seq2seq の課題は、一問一答しかできない
→　これは、問に対して文脈も何もなく、ただ応答が行われる続ける。
この問題を解決する試みが、HRED となる。

HRED とは？
過去 n 1 個の発話から次の発話を生成する 。
Seq2seq では、会話の文脈無視で応答がなされたが、 HRED では前の単語の流れに即して応答されるため、より人間らしい文章が生成される 。

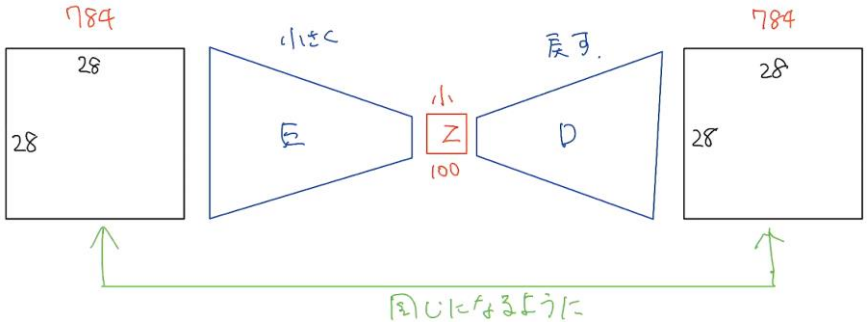
Seq2Seq+ Context RNN



VHRED
VHRED とは？

HRED に、VAE の潜在変数の概念を追加したもの
HRED の課題を VAE の潜在変数の概念を追加することで解決した構造となる 。

オートエンコーダ
教師なし学習の一つ。
そのため学習時の入力データは訓練データのみで教師データは利用しない 。
入力画像を入れて、同じ画像を出力するニューラルネットワーク。



→ エンコーダーでは次元削減を行っている。

VAE
通常のオートエンコーダーの場合､何かしら潜在変数 z にデータを押し込めているものの 、その構造がどのような状態かわからない。
VAE は、データを潜在変数 z の確率分布という構造に押し込めることを可能にします 。
→ データの近さを表現できるようにしている。
Encoder の出力似たいし Decoder の入力にランダム性を持たせる（ノイズを加えている）

確認テスト
[下記の選択肢から、seq2seq について説明しているものを選べ。]
(1) 時刻に関して順方向と逆方向の RNN を構成し、それら 2 つの中間層表現を特徴量として利用するものである。 ← 双方向 RNN の説明のため×
(2) RNN を用いた Encoder Decoder モデルの一種であり、機械翻訳などのモデルに使われる。
(3) 構文木などの木構造に対して、隣接単語から表現ベクトル（フレーズ）を作るという演算を再帰的に行い（重みは共通)、文全体の表現ベクトルを得るニューラルネットワークである。 ← 構文木の説明のため×
(4) RNN の一種であり、単純な RNN において問題となる勾配消失問題を CEC とゲートの概念を導入することで解決したものである。 ← LSTM の説明のため×

演習チャレンジ
機械翻訳タスクにおいて、入力は複数の単語から成る文（文章）であり、それぞれの単語は one hot ベクトルで表現されている。 Encoder において、それらの単語は単語埋め込みにより特徴量に変換され、そこから RNN によって（一般には LSTM を使うことが多い）時系列の情報をもつ特徴へとエンコードされる。以下は、入力である文（文章）を時系列の情報をもつ特徴量へとエンコードする関数である。ただし _activation 関数はなんらかの活性化関数を表すとする。（き）にあてはまるのはどれか。

```
def encode(words, E, W, U, b):
    """
    words: sequence words (sentence), one-hot vector, (n_words, vocab_size)
    E: word embeding matrix, (embed_size, vocab_size)
    W: upward weights, (hidden_size, hidden_size)
    U: lateral weights, (hidden_size, embed_size)
    b: bias, (hidden_size,)
    """
    hidden_size = W.shape[0]
    h = np.zeros(hidden_size)
    for w in words:
        e = （き）
        h = _activation(W.dot(e) + U.dot(h) + b)
    return h
```

文の意味を表すベクトルを作りたい。
(1) E.dot(w) 解答
(2) E.T.dot(w)
(3) w.dot(E.T)
(4) E * w

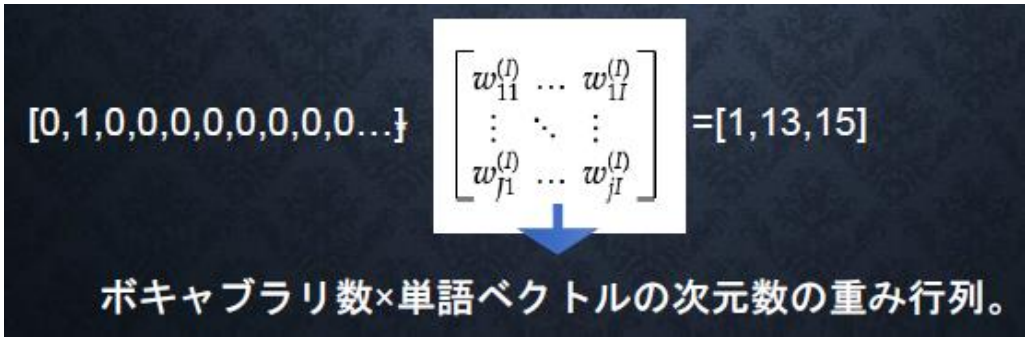
[seq2seq と HRED、HRED と VHRED の違いを簡潔に述べよ。]
HRED では、seq2seq では考慮しない文脈（意味）を理解使用としている。
VHRED では、つまらない（差し障りのない）解答しかしない HRED の課題を解決している。

[VAE に関する下記の説明文中の空欄に当てはまる言葉を答えよ。
自己符号化器の潜在変数に___を導入したもの。]
解答：確率分布

Section6：Word2vec
RNN では、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った。
単語と one-hot ベクトルから embedding 表現で意味の近い単語を見つける手法となる。

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した。

- ボキャブラリ × ボキャブラリだけの重み行列
- ボキャブラリ × 任意の単語ベクトル次元で重み行列



Section7：Attention Mechanism

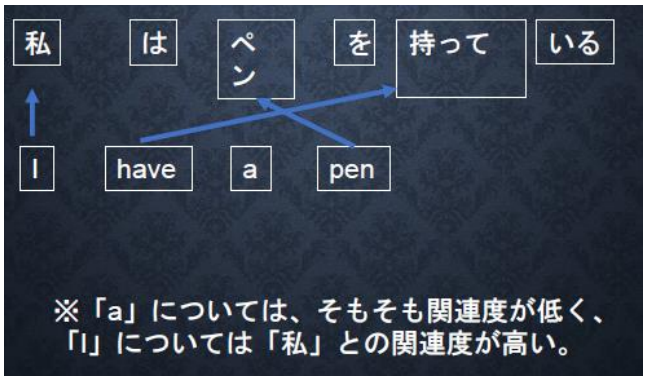
seq2seq の問題は長い文章への対応が難しいです。(中間層のサイズが固定のため。)

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ない。

そこで、Attention Mechanism では下記の構造としている。

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、仕組みが必要になります。

「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を持つ。



近年活躍している言語処理はほぼ全てに Attention Mechanism が用いられている。

確認テスト

[RNN と word2vec、seq2seq との違いを簡潔に述べよ。]

RNN は時系列データを処理するのに適したニューラルネットワークとなる。

word2vec は単語の分散表現ベクトルを得るための手法となる。

seq2seq は 1 つの時系列データから別の時系列データを得るネットワーク。

Attention は時系列データの中身に重みを持たせる手法。