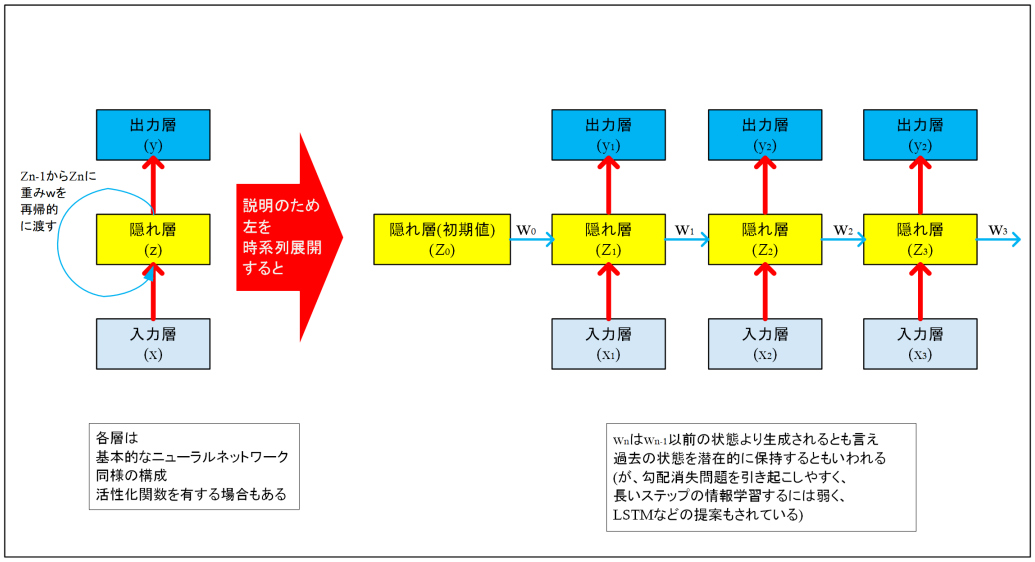
1　RNN(再帰型ニューラルネットワーク)

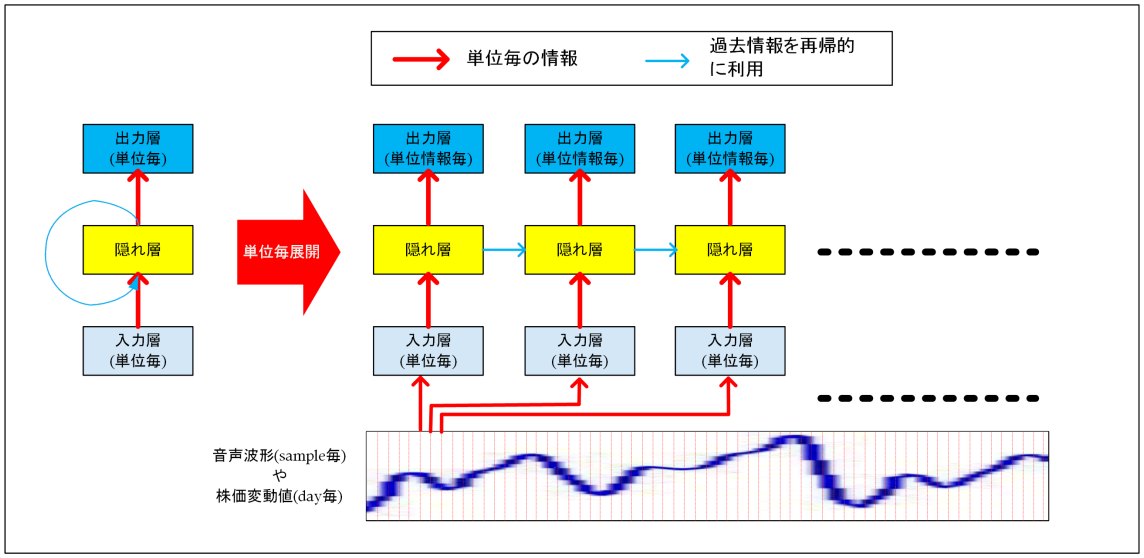
・時系列データを扱うニューラルネットワーク
・時系列データは「情報が順番に並んだデータ」であり、例えば以下
　音声信号波形
　毎日の天気
　毎日の株価変動
　テキスト (これも単語 (単語は単語ベクトル化) が順番に並んだデータと見做せる)
・時系列データの並びは「一定単位」(例えば 1 日毎、秒毎 (ミリ秒毎)) で揃っている
・テキストも「単語単位」の並びが一定単位の並びなので扱える

・RNN は隠れ層 (中間層) が再帰的に結合しており、過去の状態を参照できる
・基本的には以下の様な構造になる

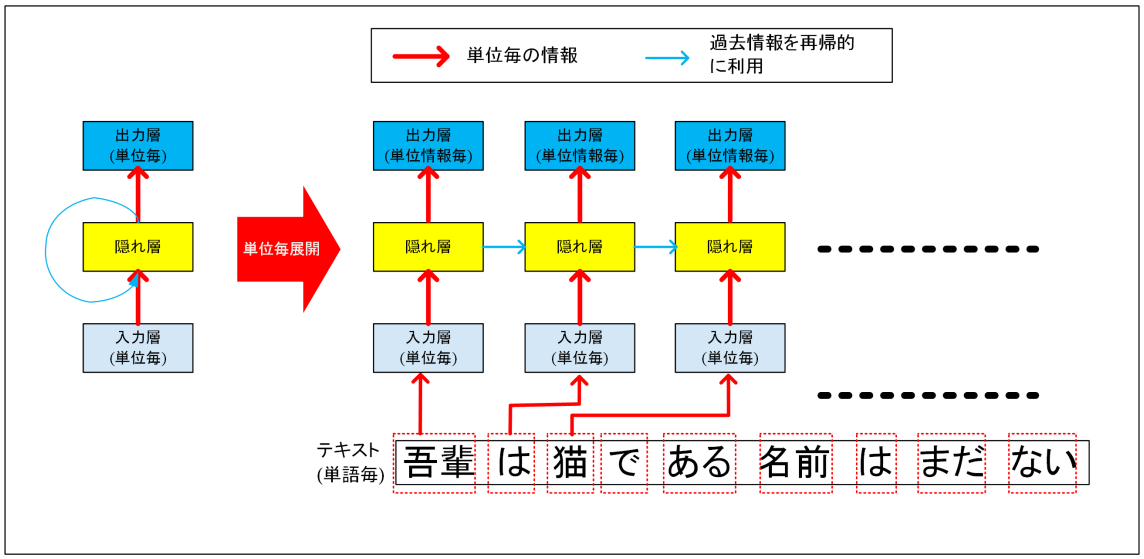


・原理的には入力層と隠れ層のメモリが十分な大きさであればそれだけ過去にさかのぼれる
・不定長の情報処理も可能なはず (現実問題としては勾配消失 (爆発) 問題より制限はされる)

・以下が RNN で単位毎変動データを扱うイメージ

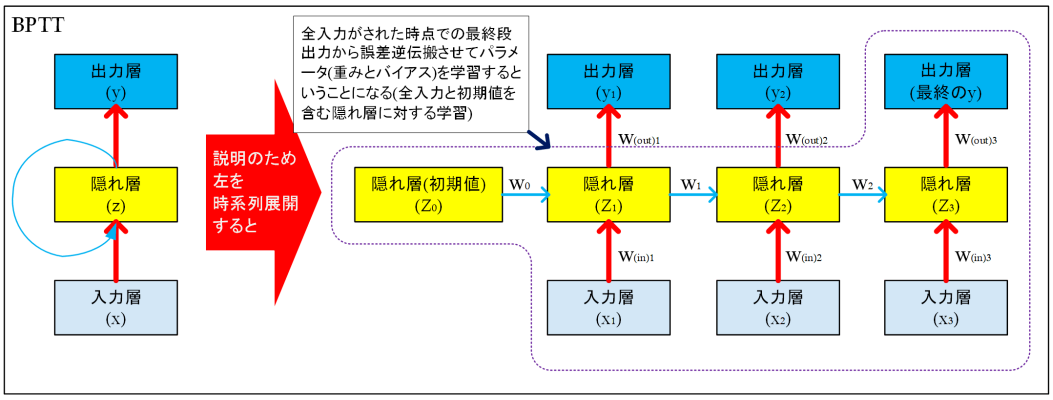


・以下が RNN でテキストデータを扱うイメージ



2　Backpropagation Through Time（BPTT）について

・学習時の誤差逆伝搬法で時系列を扱うネットワークに適用したもの
　中間層への入力 (入力層と 1 単位前の入力) と出力のパスを最終段出力より
　誤差逆伝搬法を実行して誤差修正 (=学習) を行う



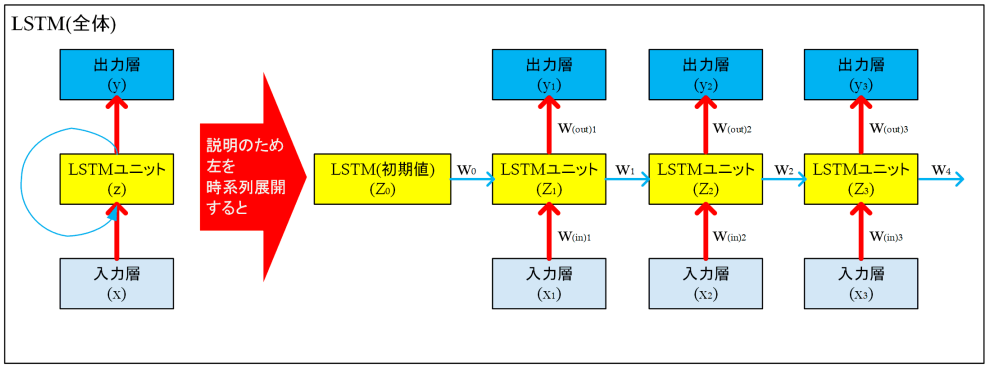
・この構成は見方によっては超多層の DNN と等価であり、勾配消失 (爆発) 問題を招く
　例えば、株価推移の予想モデルで一年分の入力を 1 日単位で入力する様なモデル
　→365 層の DNN と等価といえる

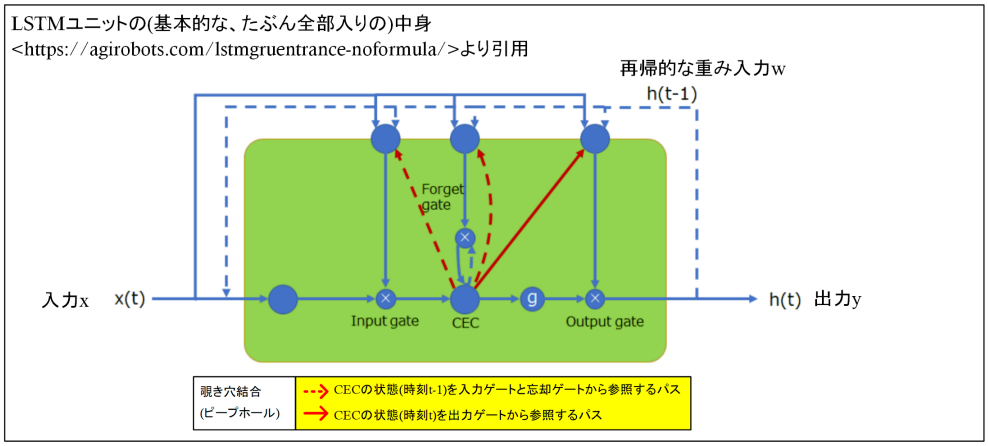
・また RNN は重み衝突問題を含む
　これは短期の (一つ前の) 重みは有用とするが、長期 (全体的に) では結局重要じゃない入力
　つまり重みの矛盾 (衝突) 問題といえる

・勾配消失 (爆発) 問題や重み衝突問題を解決する為、LSTM 等が提案されている

3 Long short-term memory(LSTM)

・RNN の派生、隠れ層に記憶素子を追加した様な構成となり、勾配消失 (爆発) 問題を抑制できる
・各隠れ層には入力ゲート、忘却ゲート、出力ゲート、がある
・上記各ゲートに入力と直前隠れ層の出力が結合している
・上記各ゲートに隠れ層内部の入力層、記憶セル (CEC)、出力層、がそれぞれ結合している

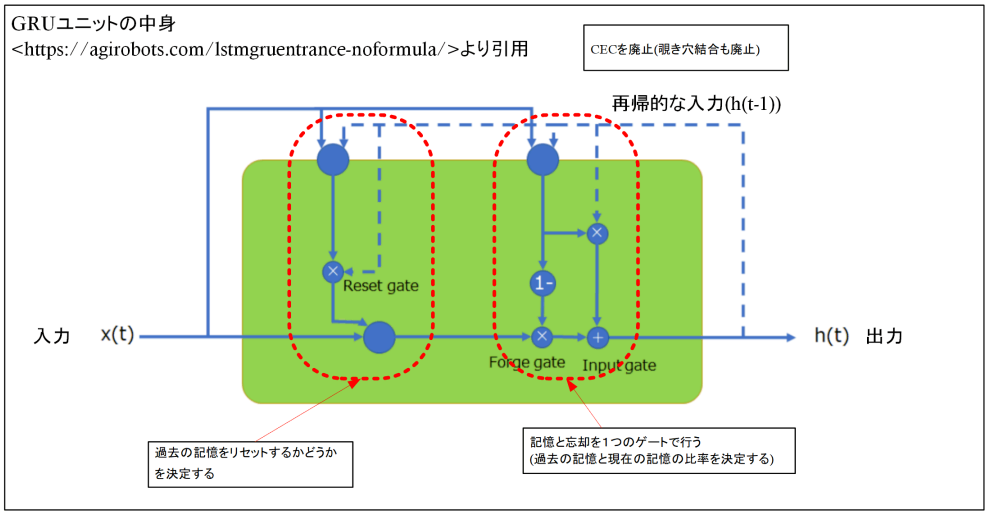




・記憶を CEC にさせる、入出力の値は CEC を挟んだ入力層と出力層で個別に算出する
・CEC に記憶される情報は入出力層を介在しない値「も」ショートカットされる
　→ シグモイド関数を通らない出力「も」反映する
　→ 誤差が大きいときは制限付き恒等関数、誤差が小さくなっても活性化関数、の作用が見える
　→ 勾配爆発が起こらない様にする為に勾配を 1 にする制限 (=誤差の微分値が 1 のまま)
・CEC は記憶機能のみを有する (学習自体は入出力層が行い CEC の値を確定する)
・入力ゲートは CEC に記憶させる値を算出、出力ゲートは CEC の記憶値を如何使うかを決定
・そのサンプルにおける入力の重要性を忘却ゲートで決定する
・入出力ゲートと忘却ゲートの状態を CEC から参照できるように結合されている (覗き穴結合)

4 Gated recurrent unit(GRU)

・LSTM を改良したネットワーク
・不要部分削除しパラメータ削減、LSTM 同様の精度を得る様に作成



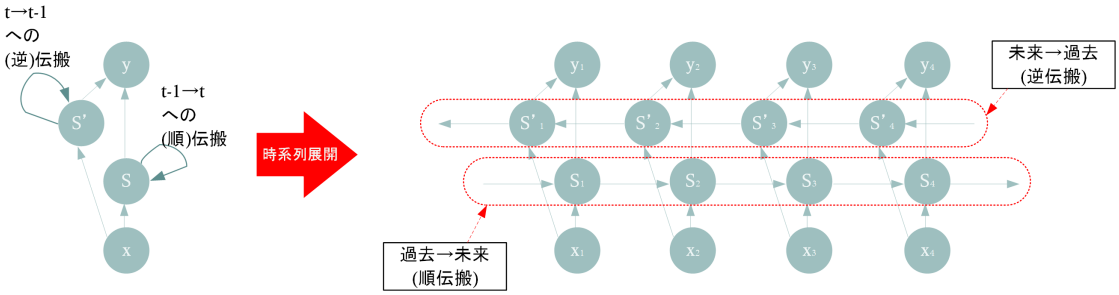
・リセットゲートと更新ゲートで構成される
(LSTM が入出力ゲートと CEC と忘却ゲートの 4 ゲート構成)
(つまり GRU は LSTM より構成が簡素化されている)

・扱うデータにもよるが GRU の方が性能が高い場合がある
(一般的には複雑な問題は LSTM の方が性能が高い傾向とされる)

・パラメータが少ないので GRU の方が学習が速い (計算が軽い)

5　双方向 RNN

・過去の情報だけでなく未来の情報を加味するような RNN のモデル
・隠れ層が順伝搬 (旧来の RNN) と逆伝搬 (時間を遡る) の 2 方向があり入出力がそれぞれ加算結合
・RNN の精度向上を狙った
・機械翻訳や文章校正によく用いられる



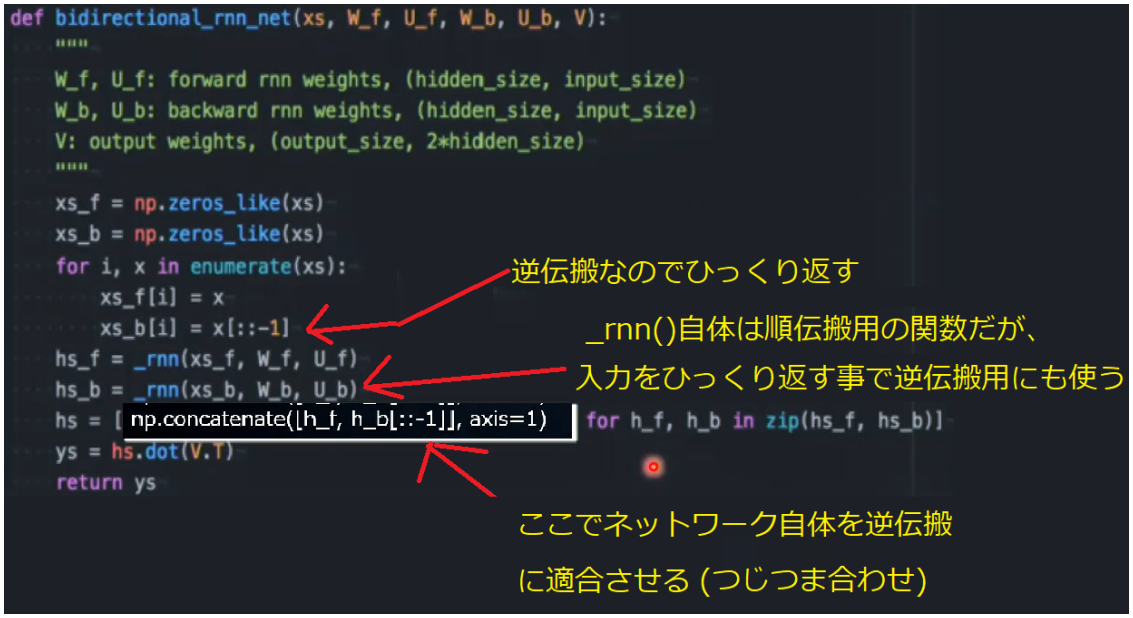
・演習問題について、とりあえず理解してみた (np.concatenate())

```python
import numpy as np
print("-----------------------------------------")
x,y = np.array([ 1, 2, 3, 4, 5]), np.array([-1,-2,-3,-4,-5])
print("xとyを結合(単純結合となるaxisは0(一次元配列の結合なので0のみ有効)))")
print("x:{} y:{}".format(x,y))
print("concat([x,y],axis=0) ={}".format(np.concatenate([x,y],axis=0)))

print("-----------------------------------------")
print("ちなみに[::-1]は配列のリバースいう意味合いになる")
print("concat([x,y[::-1]],axis=0) ={}".format(np.concatenate([x,y[::-1]],axis=0)))

print("-----------------------------------------")
x,y=np.array([[ 1, 2, 3, 4, 5]]),np.array([[-1,-2,-3,-4,-5]])
print("xとyを結合(二次元同士なのでaxisは0と1が有効)")
print("x:{}  y:{}".format(x,y))
print("concat([x,y],axis=0) ={}".format(np.concatenate([x,y],axis=0)))
print("concat([x,y],axis=1) ={}".format(np.concatenate([x,y],axis=1)))
```

✓ 0.3s

```
-----------------------------------------
xとyを結合(単純結合となるaxisは0(一次元配列の結合なので0のみ有効)))
x:[1 2 3 4 5] y:[-1 -2 -3 -4 -5]
concat([x,y],axis=0) =[ 1  2  3  4  5 -1 -2 -3 -4 -5]
-----------------------------------------
ちなみに[::-1]は配列のリバースいう意味合いになる
concat([x,y[::-1]],axis=0) =[ 1  2  3  4  5 -5 -4 -3 -2 -1]
-----------------------------------------
xとyを結合(二次元同士なのでaxisは0と1が有効)
x:[[1 2 3 4 5]]  y:[[-1 -2 -3 -4 -5]]
concat([x,y],axis=0) =[[ 1  2  3  4  5]
 [-1 -2 -3 -4 -5]]
concat([x,y],axis=1) =[[ 1  2  3  4  5 -1 -2 -3 -4 -5]]
```

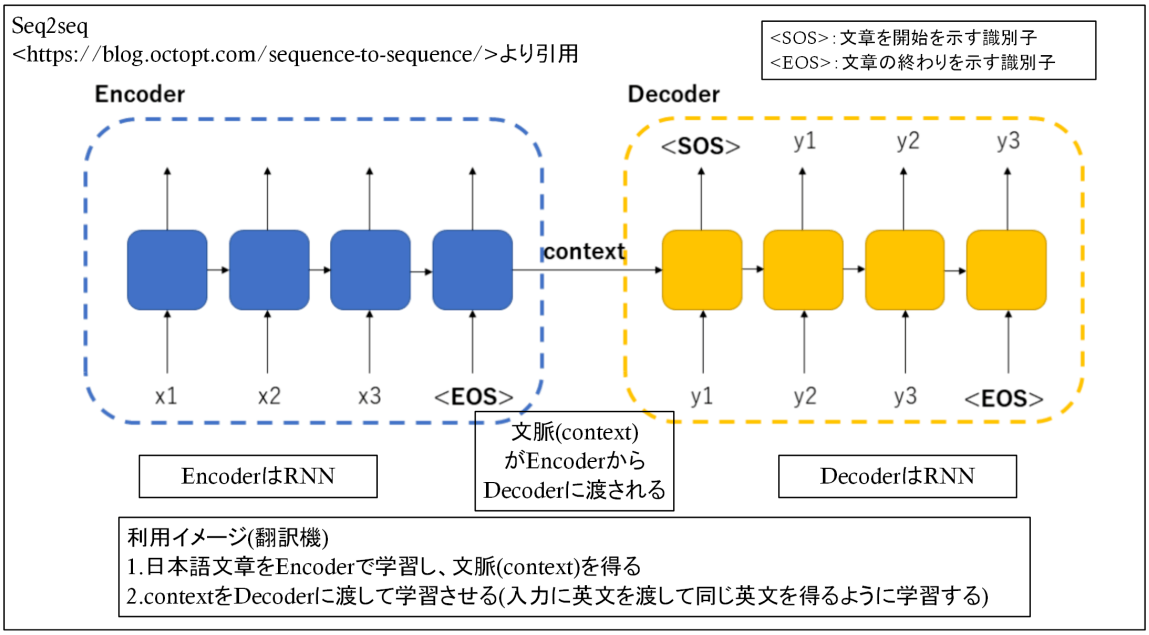
・コードの理解 (それなりに)



6 seq2seq

6.1 seq2seq の構造等

・Encoder と Decoder を context で結合した構成
・機械翻訳や音声認識によく用いられる



・例えば入力の文章が英語ならば単語区切りされるが日本語の様な言語は前処理として形態素解析が必要
・形態素解析:その言語の文法に沿って、単語や修飾語を分離しその意味を解釈する
例：私は普通の日本人です →”私” ＋ ”は” ＋ ”普通” ＋ ”の” ＋ ”日本” ＋ ”人” ＋ ”です”
上記単語に ID をつける (one-hot vector→enbedding)
本当は one-hot vector で扱いたいが変数サイズ圧縮の為 enbedding 表現にすると説明
似た意味の単語は近似した embedding 表現になる様に学習 (pre training) をする (特徴量抽出)

単語	ID	one-hot表現	embedding表現
私	0	[1,0,0,0,0,0,0]	[0.1, 0.2, 0.3, 0.4, 0.5]
は	1	[0,1,0,0,0,0,0]	[0.9, 0.1, 0.0, 0.2, 0.1]
普通	2	[0,0,1,0,0,0,0]	[0.2, 0.3, 0.3, 0.3, 0.3]
の	3	[0,0,0,1,0,0,0]	[0.0, 0.1, 0.2, 0.2, 0.7]
日本	4	[0,0,0,0,1,0,0]	[0.5, 0.6, 0.4, 0.0, 0.9]
人	5	[0,0,0,0,0,1,0]	[0.1, 0.1, 0.2, 0.2, 0.3]
です	6	[0,0,0,0,0,0,1]	[0.8, 0.7, 0.2, 0.2, 0.9]

※実際は学習する単語の数に応じたbit数となる
※数100パラメータで事が足りると言われている

NOTE:
左は例であり、実際は学習データのコーパス(文章を集めたデータベース)より抽出した単語(ボキャブラリ)より、似た意味の単語は近似したembededing表現になるように事前学習(PreTrain)して値(embedding表現の)を確定する

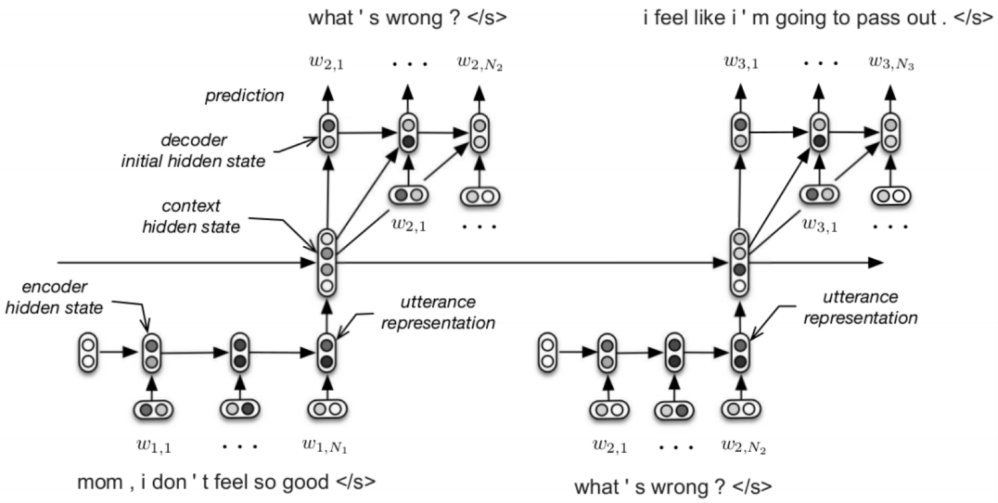
・特徴量抽出は有名どころでは BERT における Masked Language Model(MLM) がある
　MLM は学習対象の文章についてランダムに単語をマスクしてそれを予測させる事で
　文脈からふさわしい単語 (ベクトル) を学習させる (ここで MLM は教師無し学習であると説明)

・最終的に Encoder は入力された文章 (文章ベクトル) より文脈 (context) を生成する様に学習する

・Encoder より出力の文脈 context が Decoder に入力される (隠れ層の初期値となる)
・Decoder は目標値 (例えば翻訳済英文) が出力される様に学習をする

6.2　HRED

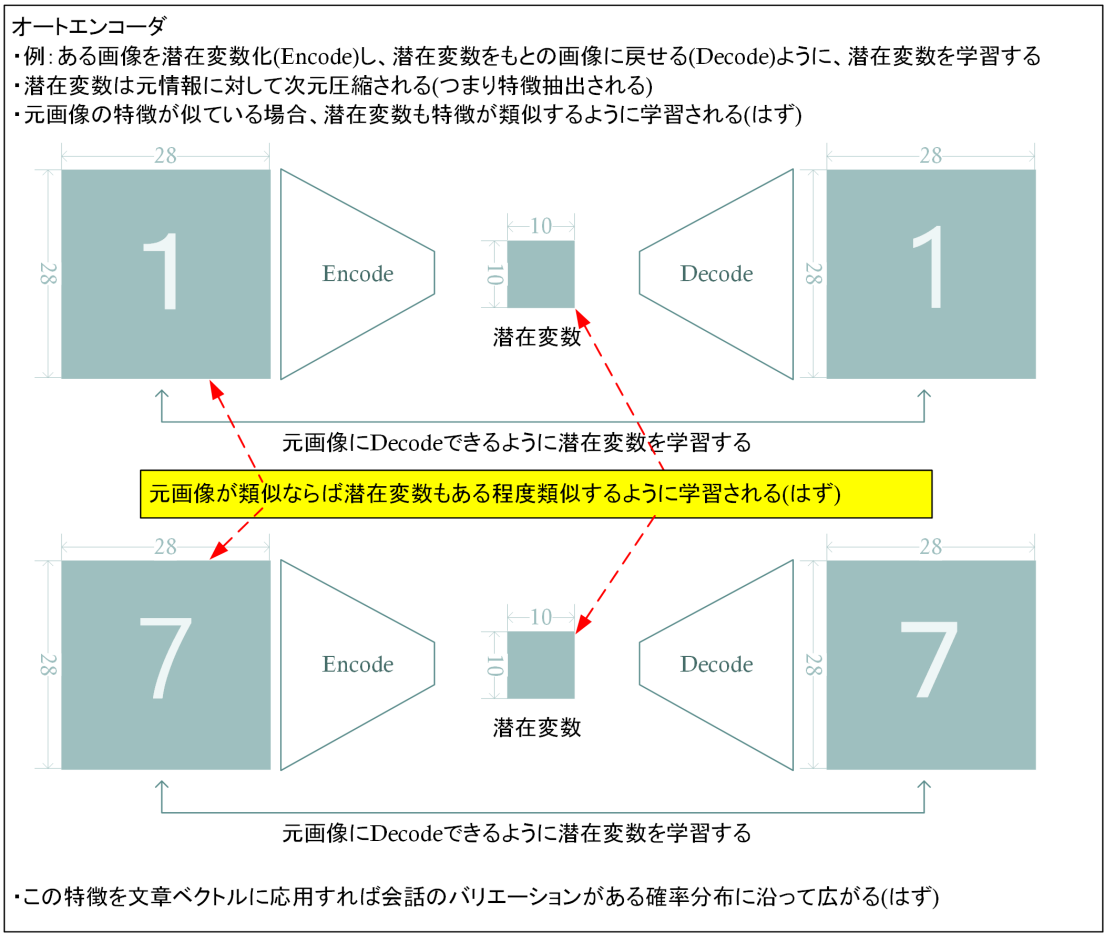
・RNN は一問一答しかできない (その文章に至るまでの文脈を汲んで応答を返せない)
　→HRED(hierarchical recurrent encoder-decoder) を提案
　これは過去 n-1 個の会話から次の (n 番目の) 会話を生成する方法
　Seq2Seq + Context RNN、Context RNN は Encoder のまとめた各文章の系列をまとめて、
　会話コンテキスト全体ベクトルに変換 (以下講義引用図の様にに Seq2Se2 を時系列接続)



・HRED により会話の文脈に沿った応答が可能にはなった
・ただし HRED で学習した会話応答は極無難な応答を選びがちな欠点を持つ
　→ 極端に言えば「はい」「いいえ」のいずれかしか返さない様になる

6.3 VHRED

・HRED に VAE(Variational Autoencoder) の潜在変数の概念を追加したもの
・VAE の作用を利用する事で会話のバリエーションを増やす
・オートエンコーダは教師無し学習の一種 (以下図の説明)

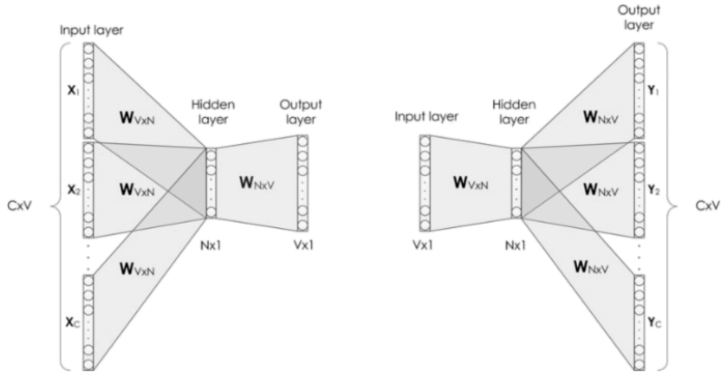


・VAE はオートエンコーダの潜在変数が平均 0 分散 1 の確率変数に収まるように調整
　→ 似た情報より得た潜在変数は近い値になる
・VAE は Encode 時に元データをランダムにドロップアウトした状態で Decode できる様に学習する
　→ 汎化性能を得る (平均 0 分散 1 の確率分布になる様に潜在変数を学習できる)

7 word2vec

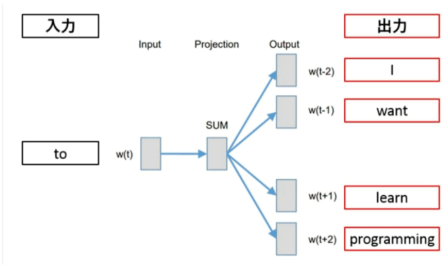
・単語を単語ベクトル (Enbedding 表現) に変換する方法
・単語自体の意味類似性をベクトル類似性で表現できる様にした
・NN 構造は単純 (2 層)、2 つの機能を内包する

Word2Vec(<https://ledge.ai/word2vec/>より抜粋)
多単語入力→単単語出力 か 単単語入力→他単語出力 のNNのようである
手法が2つ(SkipGramモデルとCBOWモデル)を内包とのこと

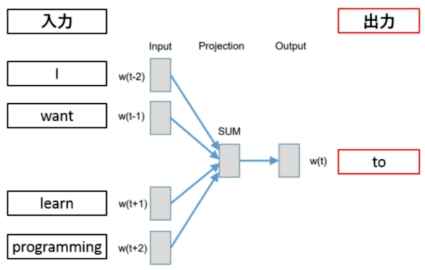


・2 つの機能は以下 (いずれも教師あり学習となる)

Word2Vec::SkipGramモデル(<https://ledge.ai/word2vec/>より抜粋)
文章の中心単語から周辺の単語を予測する手法



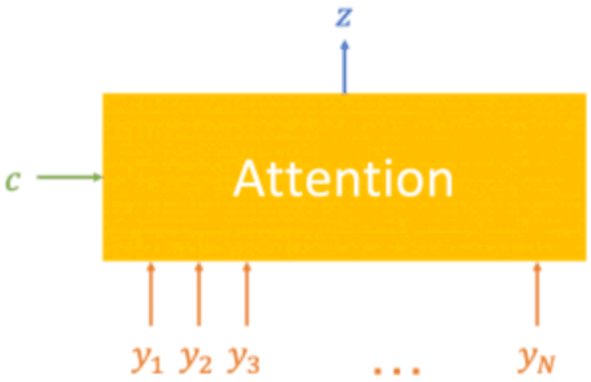
Word2Vec::CBOWモデル(<https://ledge.ai/word2vec/>より抜粋)
文章の周辺の単語を入力して中心単語を予想する手法



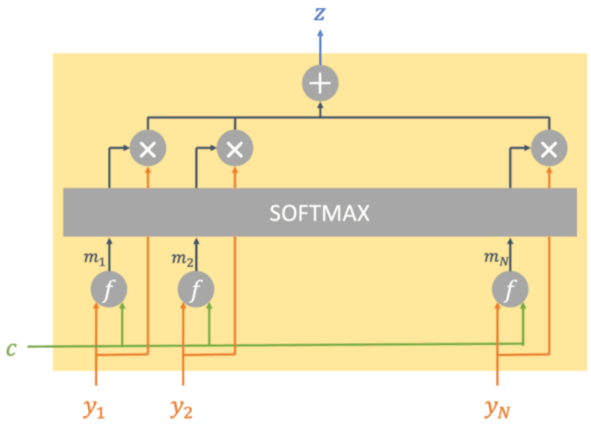
8 Attention Mechanism

・よく日本語直訳で「注意」とされているが、その通りの意味となる
・要するに「その文章における重要な単語 (複数かも) は何になるか」を認識する機能
・Attention Mechanism は画像処理でも利用されている (その画像のどの部分が重要かを識別)
・Attention Mechanism の簡単な事例を図示

Attention Mechanism(の一例)(<https://dendenblog.xyz/what-is-attention/>より抜粋)
yが画像入力でcが単語情報を示す隠れ層のコンテキスト,zが出力



中身
(単語コンテキストと画像の情報を加算し重み付き出力、それをsoftmaxに通して確率分布に落とし最終的に入力と重要度を掛け算した情報が適切に出力されるように重みを学習する)



9 実習

・3_1_simple_RNN.ipynb
・3_3_predict_sin.ipynb
　それぞれ try 項目の追加と簡単な考察を実施

・本レポートと同じレポジトリにそれぞれの実装コードを置く