進捗報告用資料(4/22~25)

目次

東大に行った感想(4/22)

- とりあえず自分の勉強の方向性は少しずれていた。
- 基礎がすべてだと思いこみ、確率論などの問題集を解いたり、統計をやってみたり、＿＿ディープラーニング以前の手法＿＿について色々やっていた。
- これは、＿＿遠回りすぎる(私の要領的に無理)。＿＿余裕が出来てきたら、戻ってくる。
- とりあえず、色んな＿＿ディープラーニング＿＿の手法を学ぶ。

やったこと(4/22~25)

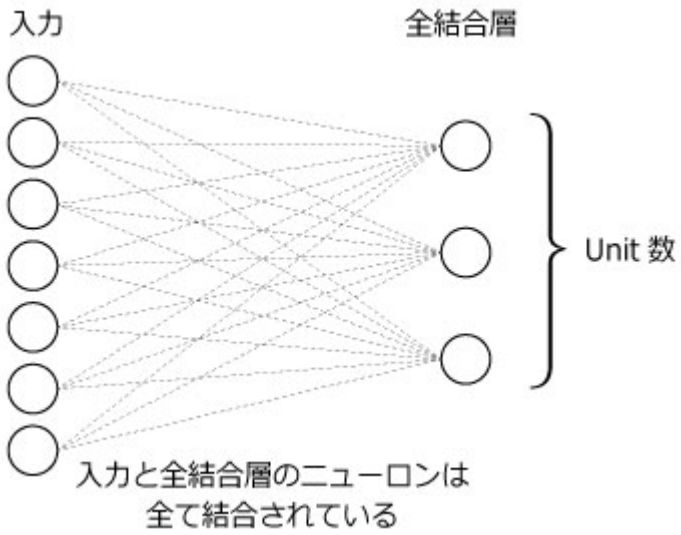
以上を加味してやったこと。

1. とりあえず基礎に戻ってDenseレイヤー(全結合層)
2. CNN
3. RNN
4. GRU
5. LSTM
6. bi-LSTM

7. SE-block(Attentionぽいもの)
8. RNNLM
9. word2vec
10. doc2vec
11. Attention(初代)
12. 今後の予定(Attentionのより一般的な機構)

以下順を追って概要だけまとめていく。(画像を無断で拾ってきているため、(する価値も無いと思いますが)外部には公開しないでください。)

Denseレイヤー(全結合層やfc層と呼ばれることも)



入力と出力がすべてつながっている層のこと。入力ベクトルIをi次元、出力ベクトルOをo次元とすると

$$O = W_{i2o}I$$

となるためパラメータ数は$i \times o$個数となる。つまり線形写像のこと。ただし、一般には定数項も含めて線形とか言ったりする(正確にはアフィン写像)。つまり

$$O = W_{io}I + b$$

ということになる。

$concat$なる演算を以下のように定義する。

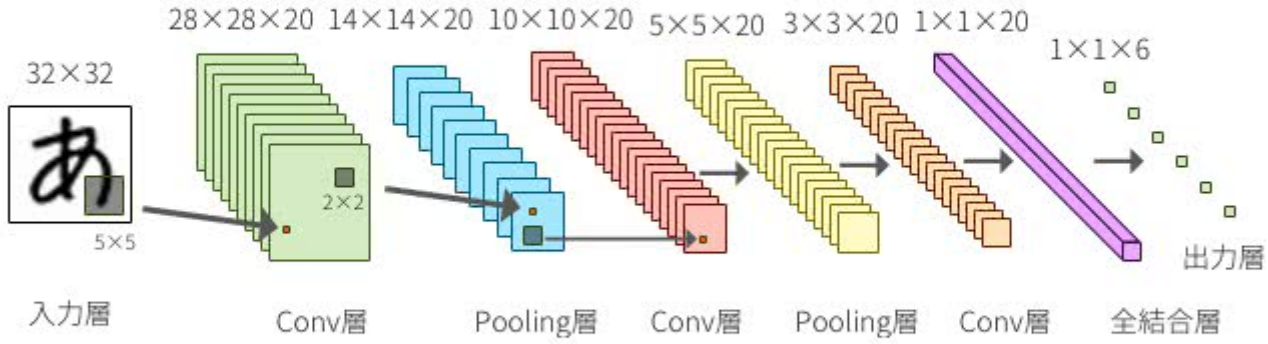
$$concat(X, Y) = (x_1, x_2, \cdots, x_n, y_1, y_2, \cdots, y_m)^T$$

要はつなげるだけである。python的にいうなら`X.extend(Y)`である。すると先ほどのアフィン写像は

$$O = W_{io}I + b = W_{i+1o}concat(X, 1)$$

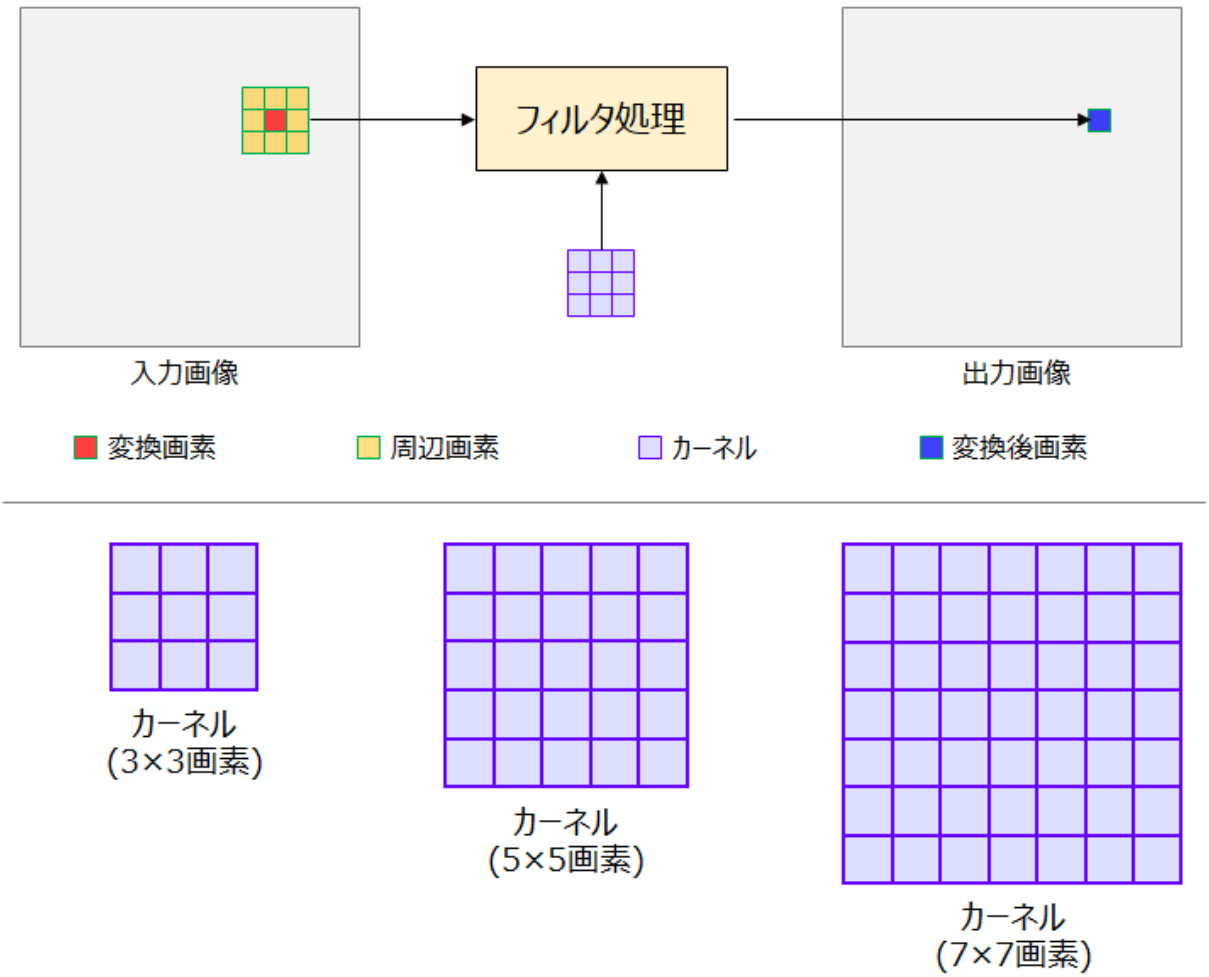
となる。

CNN



CNNは畳み込み層とプーリング層を持つニューラルネットワークのこと。畳み込み層(Conv層)は特徴量の自動抽出(特徴マップの作成)、プーリング層(Pooling層)は次元圧縮をしている。

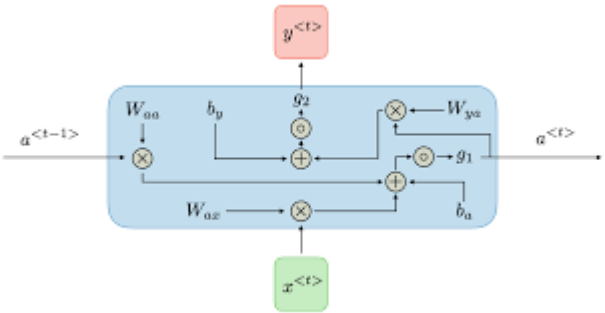
畳み込み層は画像処理で言うところフィルタ処理にあたる。



画像のように、カーネルと入力画像で内積を取りその大きさを特徴マップとして出力する。内積を取るということは、カーネルとのパターンが同じような部分は値が大きくなり、そうでない部分は値が小さくなるため特徴抽出として機能する。これに関しては、実際にやってみるのが早い。pythonなどで、ガウシアンフィルタや平滑化フィルタ、x軸フィルタやy軸フィルタを画像にかけて見るとよくわかる。

https://github.com/yukimaru77/pytorch-experiment/blob/master/jupyterLab/work/chapter10.ipynb

RNN



RNNは過去のデータも考慮して、出力を決めることが出来るアーキテクチャである。画像を数式化する。入力がシーケンシャルデータ(順序に意味のあるデータ)とし、そのt番目のデータを$x^{(t)}$とし、g_1, g_2は活性化関数である。

$$入力：x^{(t)}$$

$$内部状態(隠れ層)：a^{(t)} = g_1(W_{ax}x^{(t)} + W_{aa}a^{(t-1)} + b_a)$$

$$出力y^{(t)} = g_2(W_{ya}a^{(t)} + b_y)$$

ただしこのような表記はあまりされず、慣習としてDense層は活性化関数をgとし、引数のパラメータを除いたものをX, Yとすると

$$g(W_X X + W_Y Y + b) := f^g(X, Y)$$

と表記されるため先ほどのの入力、内部状態、出力は

$$入力：x^{(t)}$$

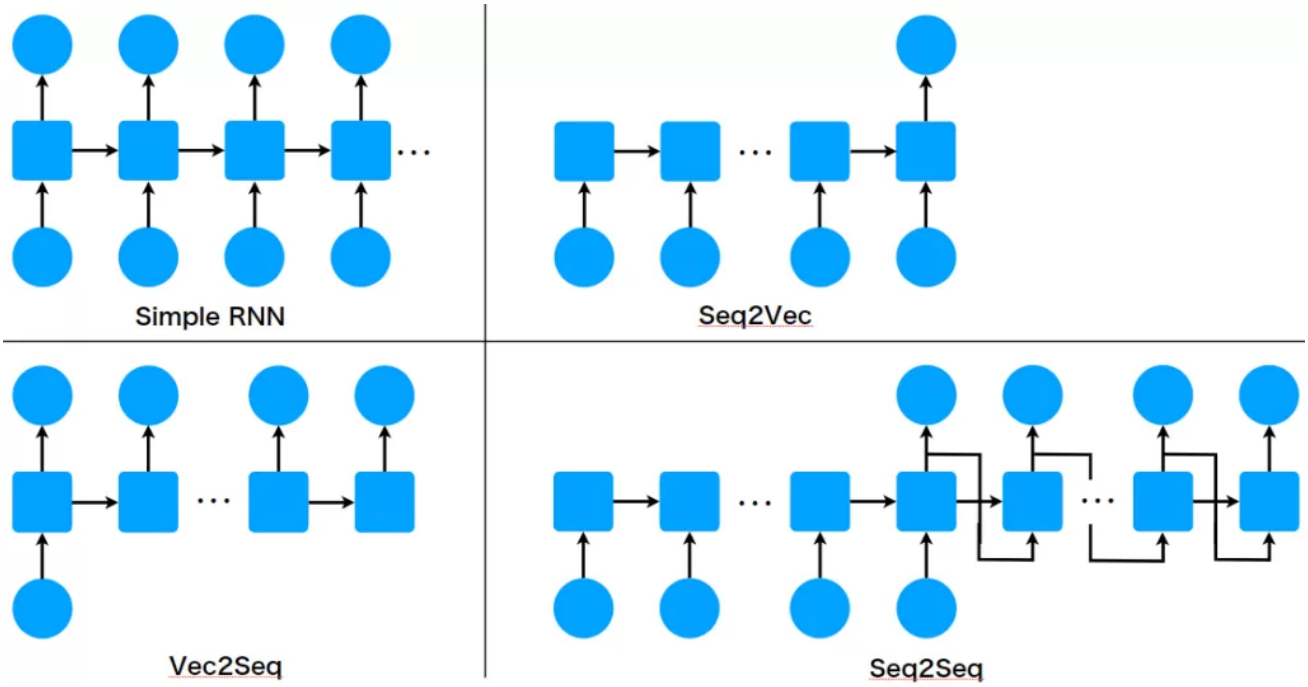
$$内部状態(隠れ層)：a^{(t)} = f^{g_1}(x^{(t)}, a^{(t-1)})$$

$$出力y^{(t)} = f^{g_2}(a^{(t)})$$

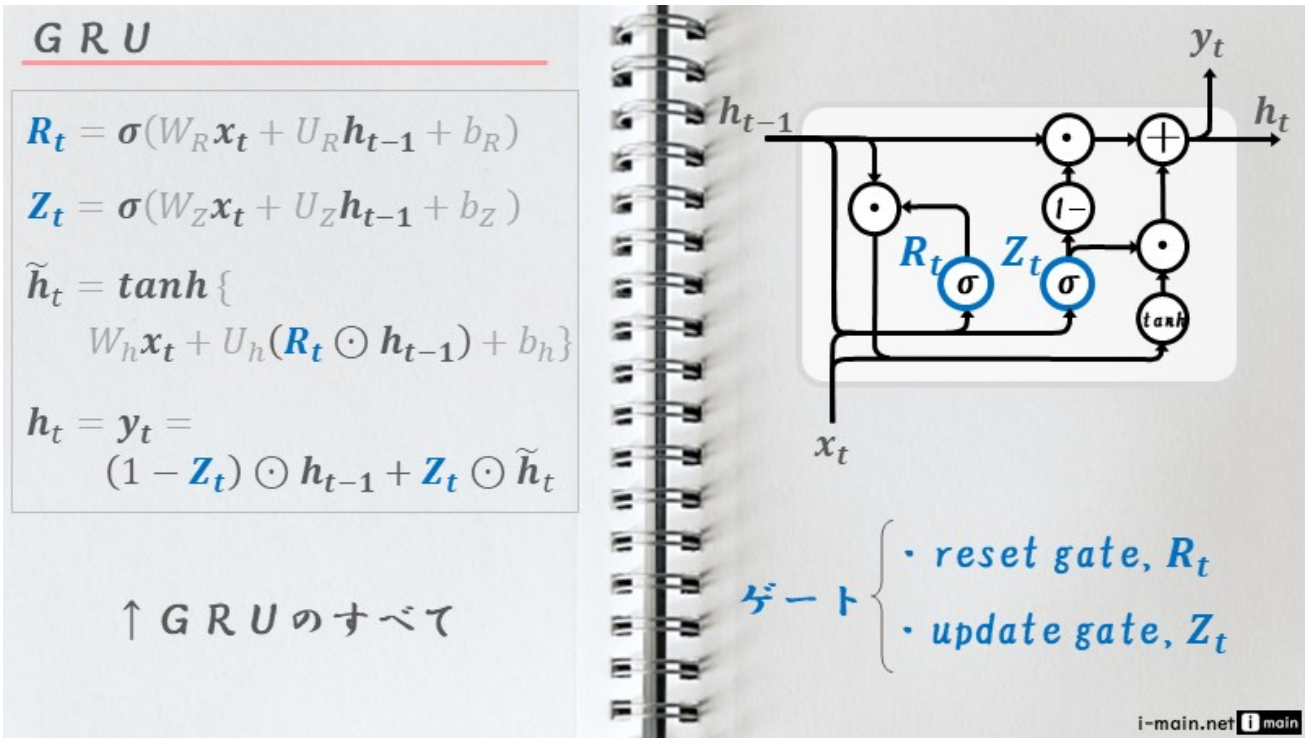
ただし、今回の図のアーキテクチャでは先に$a^{(t)}$を計算してから$y^{(t)}$を計算するとなっていたのでその通りにしたが場合によっては$y^{(t)}$から計算する場合もある。つまり

$$出力y^{(t)} = f^{g_2}(x^{(t)}, a^{(t-1)})$$

となっている場合もある。なお、RNNには以下のようなバリエーションがある。右上がエンコーダ、左下がデコーダに対応する。



GRU

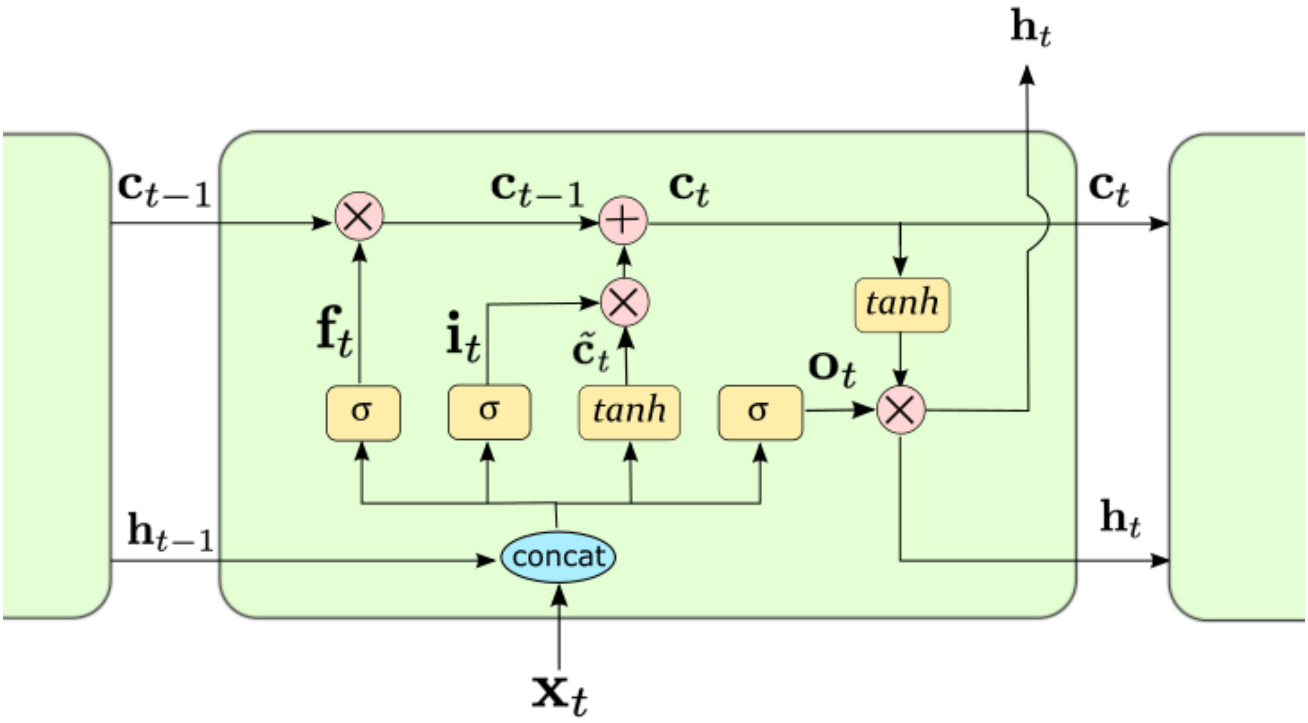


RNNに長期記憶を持たせようとしている。ここで各文字の役割を先に説明しておく。

- R_tはどのくらい忘れるの？という指標
- Z_tはどのくらい新しい情報を使うの？という指標
- $\tilde{h}_t$はx_tによって追加された情報
- h_tは内部状態

これによってRNNのように単純に内部状態を更新するのではなく、どのくらい更新するのか？という操作が可能になる。

LSTM



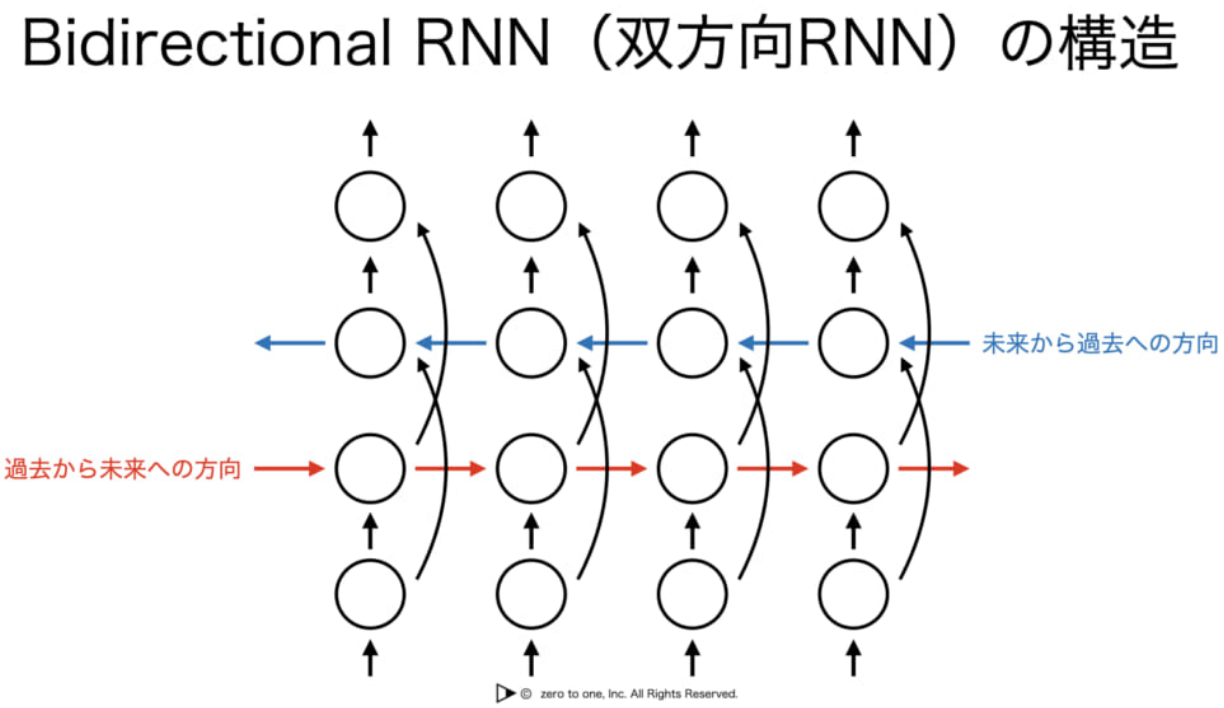
$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

RNNに長期記憶と短期記憶の両方を持たせようとしている。ここで各文字の役割を先に説明しておく。

- c_tは長期記憶
- h_tは短期記憶
- f_tは長期記憶をどのくらい忘れるの？という指標
- i_tは長期記憶にどのくらい新しい情報($\tilde{c}_t$ ＝前回の短期記憶+現在の入力x_t)を混ぜるの？という指標
- o_tはどのくらい長期記憶から短期記憶にするの？という指標

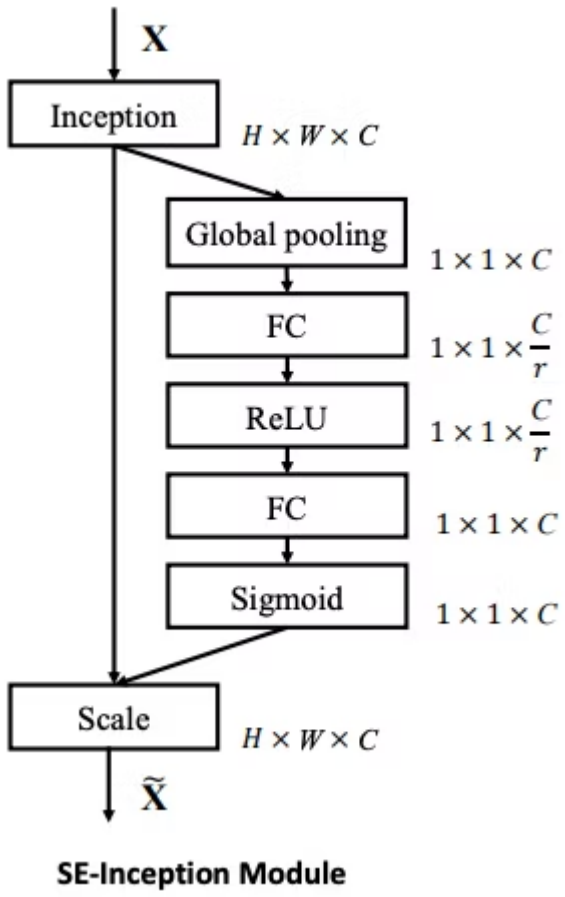
これによってRNNのように単純に内部状態を更新するのではなく、長期記憶と短期記憶の二つを持つことが可能になった。

bi-LSTM



LSTMは一方向しかデータを使わなかったのに対し、bi-LSTMは双方向の情報を使っている。

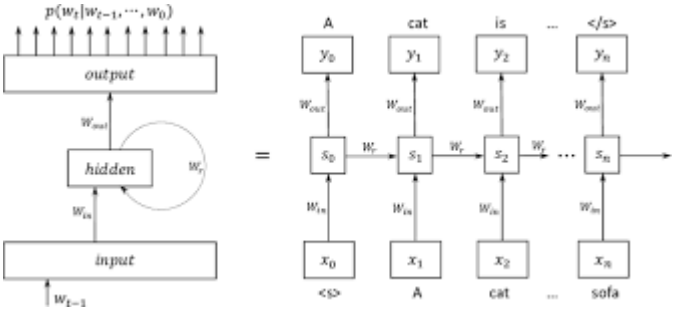
SE-block



SE-Inception Module

SE _ blockは、各どの特量マップに注目すればいいかを明示的に指定する仕組み。

RNNLM

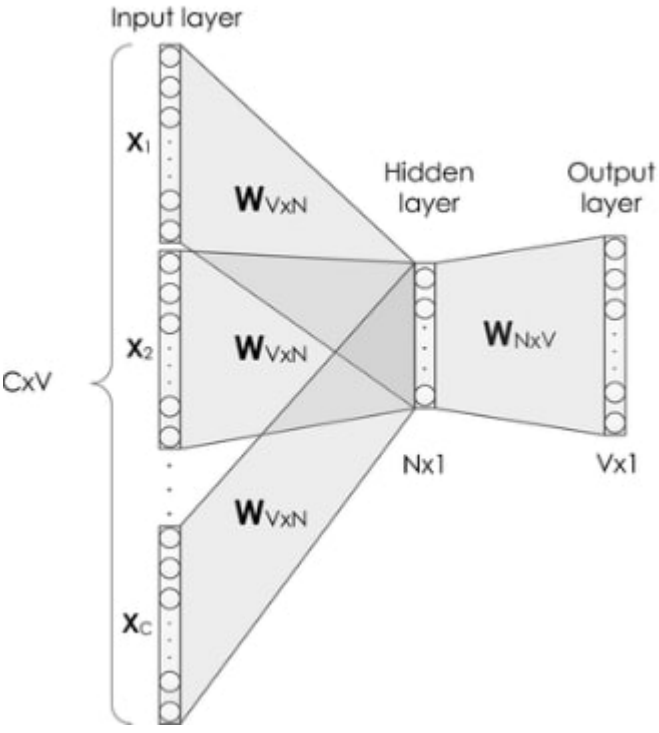


RNNを言語モデル(LM)に用いるということ。普通のRNNに見える。なぜ特別な名前が与えられているのかよくわからない。

word2vec

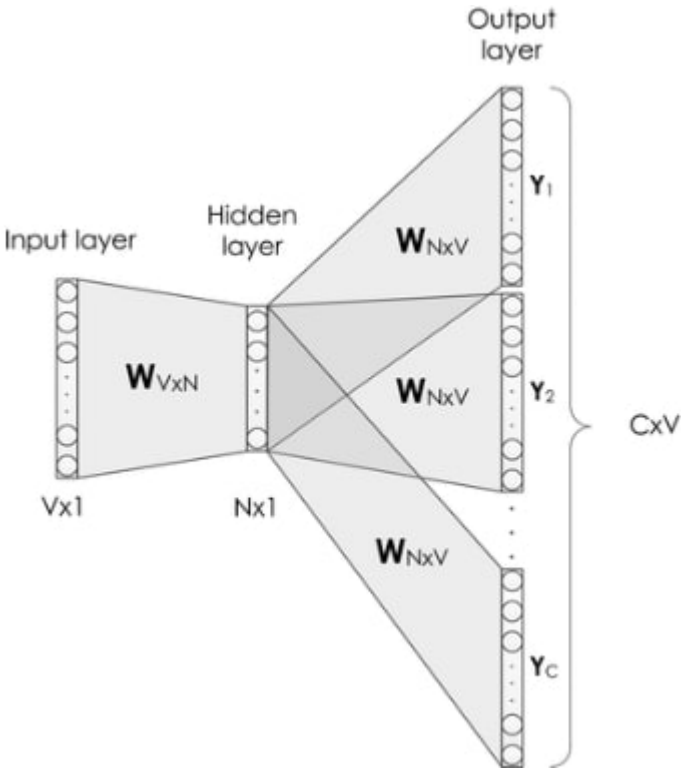
word2vecとはwordをvectorにするモデルの _ _ 総称 _ _ 。特別に何かを指すわけではない。ちなみに、ベクトルにした単語のことを単語の分散表現と言ったりする。

CBOW



上記のアーキテクチャとなっている。Hidden layerは平均値をとっている。$W_{V\times N}$は複数個あるが全て同じものを使用している。入力は$C-1$個の単語($X_{\frac{C}{2}}$が抜けている。)であり、出力は$X_{\frac{C}{2}}$の予測値の確率分布(Xはone _ hotベクトル)。このタスクを精度よく解くことが出来れば、$W_{V\times N}$のそれぞれの縦ベクトルが、単語の分散表現になっている。

skip _ gram



上記のアーキテクチャとなっている。CBOWの逆バージョン。ただし、$W_{N\times V}$は全て同じのため当然ながら出力の確率分布はすべて等しくなってしまう。つまりタスクの精度だけを見るとかなり悪い。しかし、真のタスクは単語の分散表現の獲得のため、そんなことはどうだっていい。同じく、$W_{V\times N}$のそれぞれの縦ベクトルが、単語の分散表現になっている。

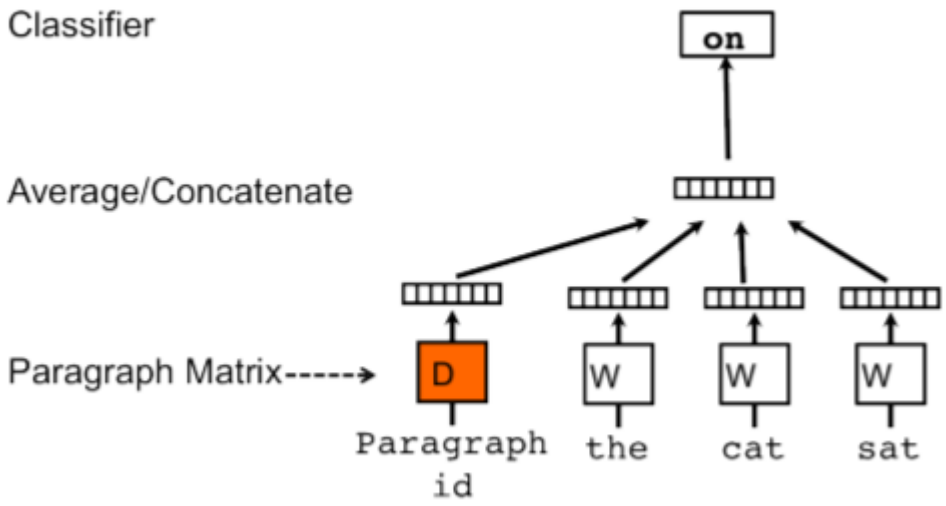
軽量化

最後の出力は確率分布、つまりソフトマックス関数を使用している。しかし、一般にone-hotベクトル何百万次元となることが珍しくないため、シグマの計算が想い。よってhierarchical softmax(100分類×3回=100万回)といった手法や、negative sumplingといった手法で軽くするように努力している。

doc2vec

doc2vecとはdocumentをvectorにするモデルの総称。

PV-DM



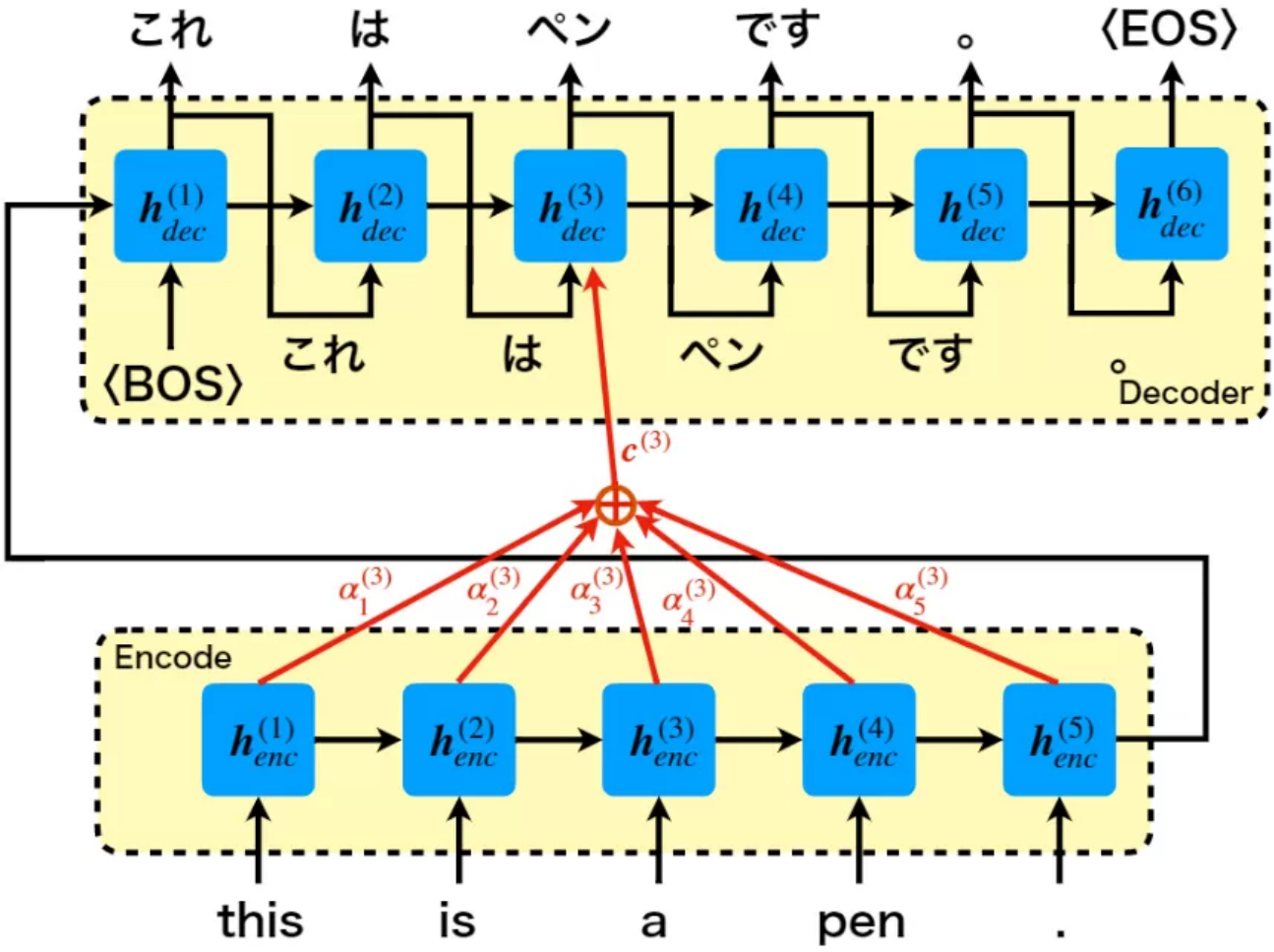
上記のアーキテクチャになっている。word2vecのCBOWに対応している。タスクは各documentにidを振っておき、$n-1$個の単語からn個目の単語を予測するというタスク。分散表現も同様に、Dの縦ベクトルがそれぞれの文章に対応する分散表現になっている。

PV-DBOW

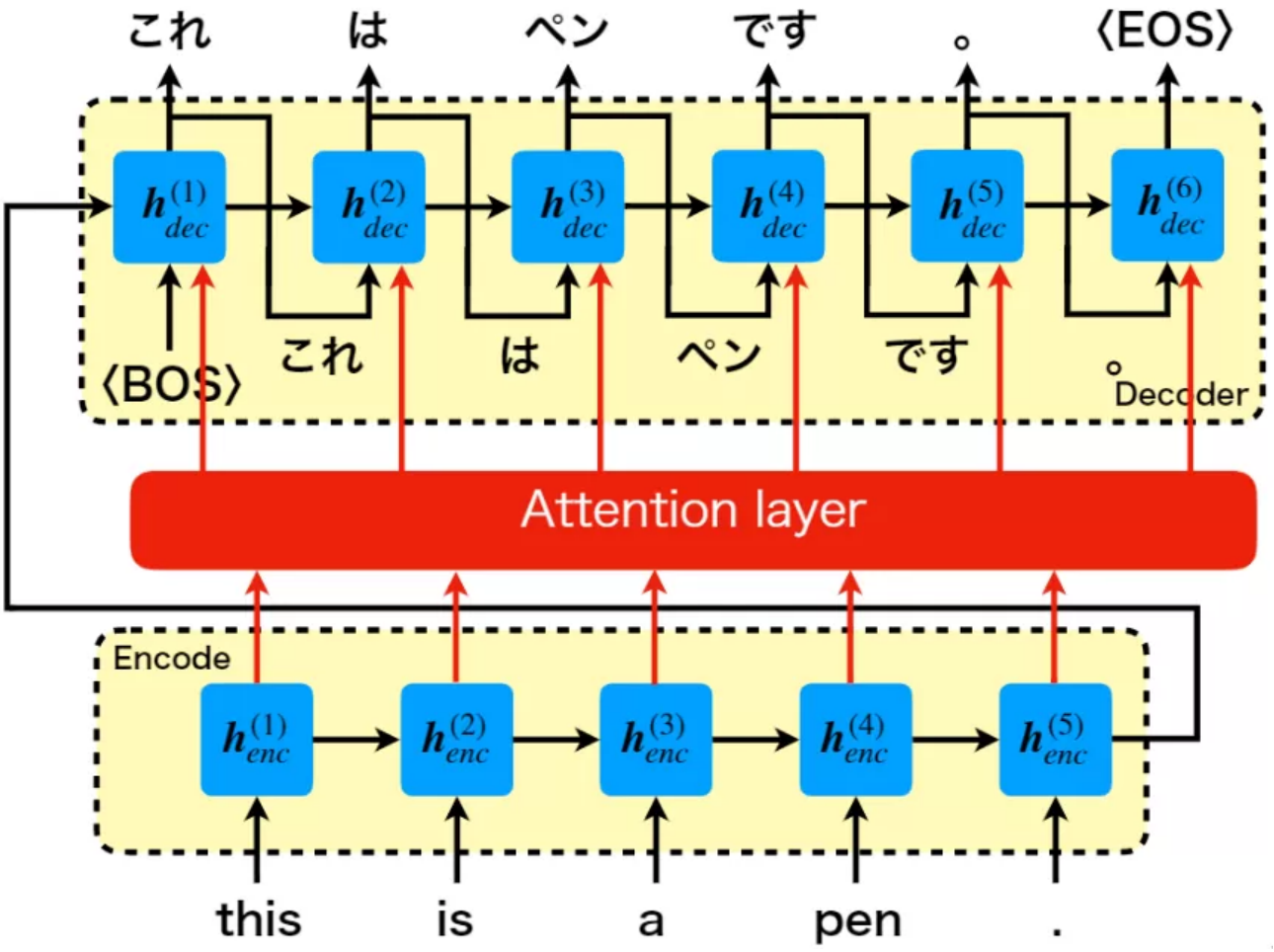
PV-DBOWというskip-gramに対応するものもあるが(本来はPV-DMとPV-DBOWの結果をconcatして一つの出力とする)、PV-DMのみでそこそこいい精度が出るらしく、例えばgensimではデフォルトでPV-DBOWは使わない設定となっている。よってここでは割愛する。

Attention(初代)

(初代？)Attention機構は以下のような図になる。



この図は、seq2seqなるRNNの図となっている。さらに$h_d^{(3)}ec$におけるAttetionに注目して図示した図である。(本当はすべての場所$h_d^{(n)}ec$に矢印が伸びている。)それを図示したのが以下の図になる。



数式にすると

$$p(y_i) = p(y_i|y_{i-1}, h^{(i)}, c^{(i)})$$

$$c^{(i)} = \sum_j \alpha_j^{(i)} h_{enc}^{(j)}$$

$\alpha_j^{(i)}$を学習することで、どの入力に対して重みを強くするかを操作することが出来る。