AIエンジニアリング　(10)

Recursive Neural Network とその応用

時系列処理

- RNN(Recurrent, Recursive) NNは主に時系列処理に用いられる手法
- 時系列：音声、テキスト、株価などの時間的な変化を表すデータ系列
- 例えば音声認識や機械翻訳などが該当する
- 特に自然言語処理での研究は非常に活発

Recurrent & Recursive NN

RNN

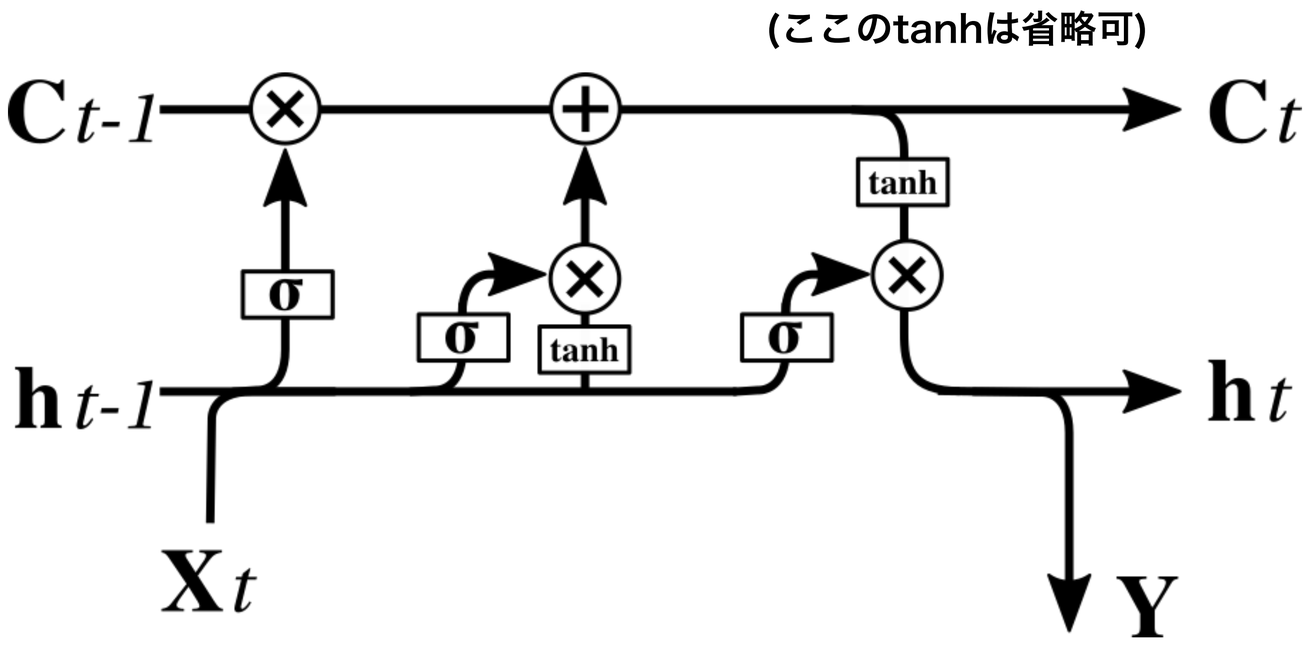
- 時系列データを扱うためには今までの情報を保持しなければならない
- データ保持のためにはネットワークの何処かにループが必要になる
- ループがあるとBackpropagationできないが、Backpropagation Through Time (BPTT) というテクニックで回避

Long-Short Term Memory (LSTM)

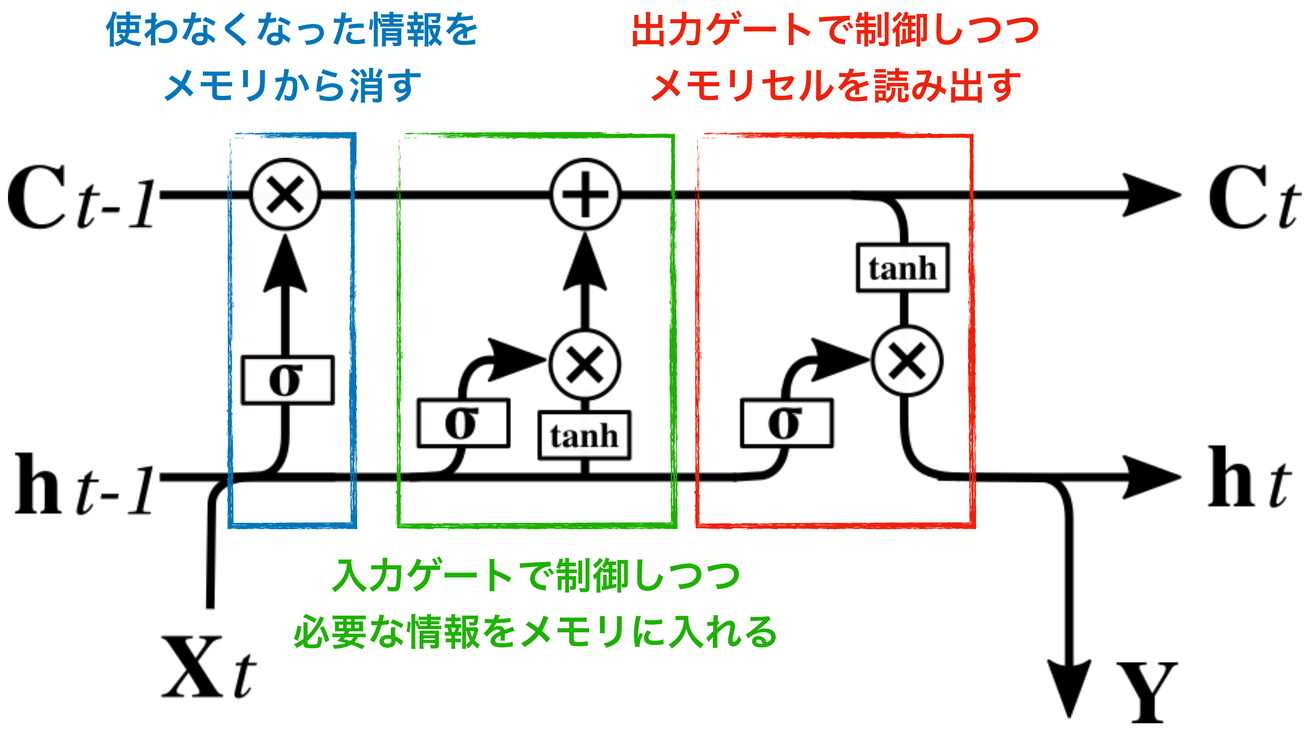
LSTMの前に

- LSTMを含めこれから紹介する方法は複雑なものも多いが、基本的には「もっとシンプルなネットワークでも**理屈上は**学習可能」である (→万能近似定理参照)
- ただ普通のMLPやRNNを学習しようとしても良い解に収束させるのは難しい
- そこで「こうやればもっと学習が楽になるはず」と手を加えてあげる事を考えて、それがResNetだったりLSTMやその他いろいろな方法である
- LSTMも複雑だが「こうしないと学習不可能」というわけではないことに注意

LSTM



LSTM



忘却ゲート、入力ゲート、出力ゲートの3つのSigmoidがある

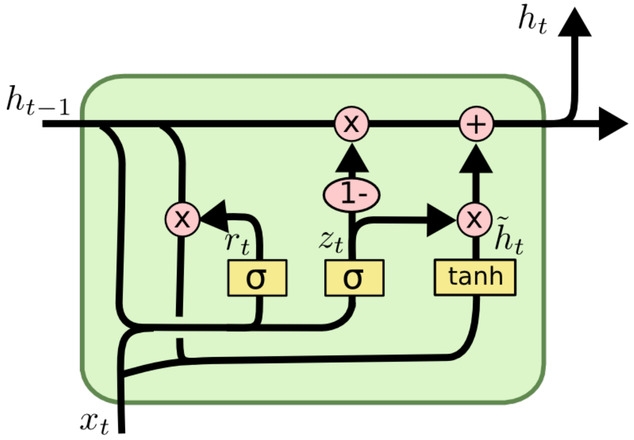
tanhはどれくらいメモリに書き込むかと、どれくらいメモリから読み込むか

GRU(Gated Recurrent Unit)

- LSTMは歴史的経緯からいろいろなパーツを追加する形で進歩してきた
- 思い切って簡略化したモデルも存在し、代表例がGRU
- メモリセルをなくして、忘却ゲートと入力ゲートを合体(更新ゲート)したモデル

GRU(Gated Recurrent Unit)

- LSTMは歴史的経緯からいろいろなパーツを追加する形で進歩してきた
- 思い切って簡略化したモデルも存在し、代表例がGRU
- メモリセルをなくして、忘却ゲートと入力ゲートを合体(更新ゲート)したモデル



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

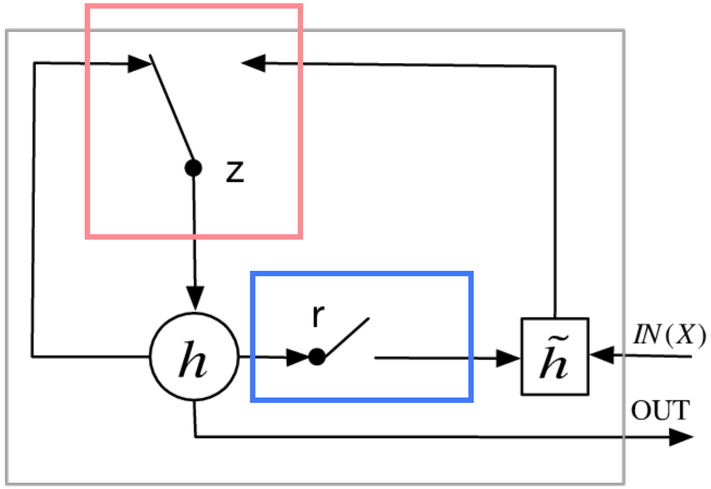
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

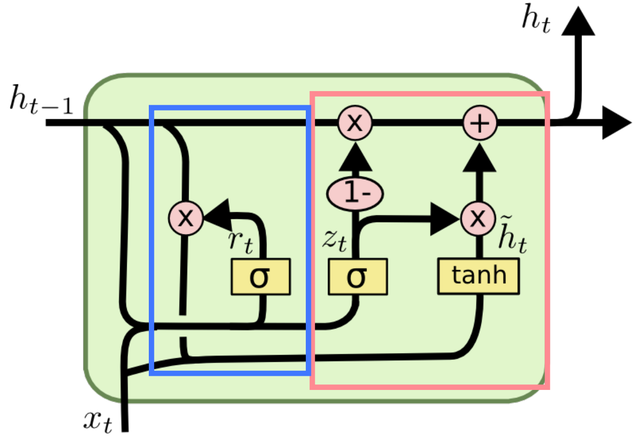
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

GRU

更新ゲート



リセットゲート



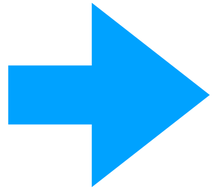
Word2Vec

単語をどうNNに入力するか？

- 最も単純に考えると、単語のインデックスをそのまま入力する
  - a→0, the→1, this→2, is→3, … apple→100, …
- 名義尺度をそのまま入力するのはよくないので、one-hotにするべき
  - a→[1 0 0 … 0], the→[0 1 0 … 0], …
  - ただこれは「似ている単語」が「似ているベクトル」になるわけではないし、逆もまた然りなので表現能力が低い
  - しかも単語数=次元数になってしまってメモリ効率も非常に悪い

分散表現

- One-hot ベクトルだと意味は埋め込まれない
  - 犬 = [0 0 0 1]
  - 猫 = [0 0 1 0]
  - クルマ = [0 1 0 0]
  - バイク = [1 0 0 0]

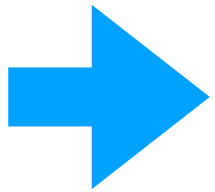


どのベクトルを見ても、
同じくらい違うので、
ベクトル間の距離には意味がない

これでも学習はできるが、
特徴ベクトルが持っている情報が少なすぎる

分散表現

- たとえばこのような表現が獲得できると嬉しい
  - 犬 = [0 0 8 7]
  - 猫 = [0 0 2 9]
  - クルマ = [1 4 0 0]
  - バイク = [3 5 0 0]



犬と猫は似ている、
クルマとバイクも似ている、
犬とバイクは似ていない、
猫とバイクも似ていない、
などの情報がわかる

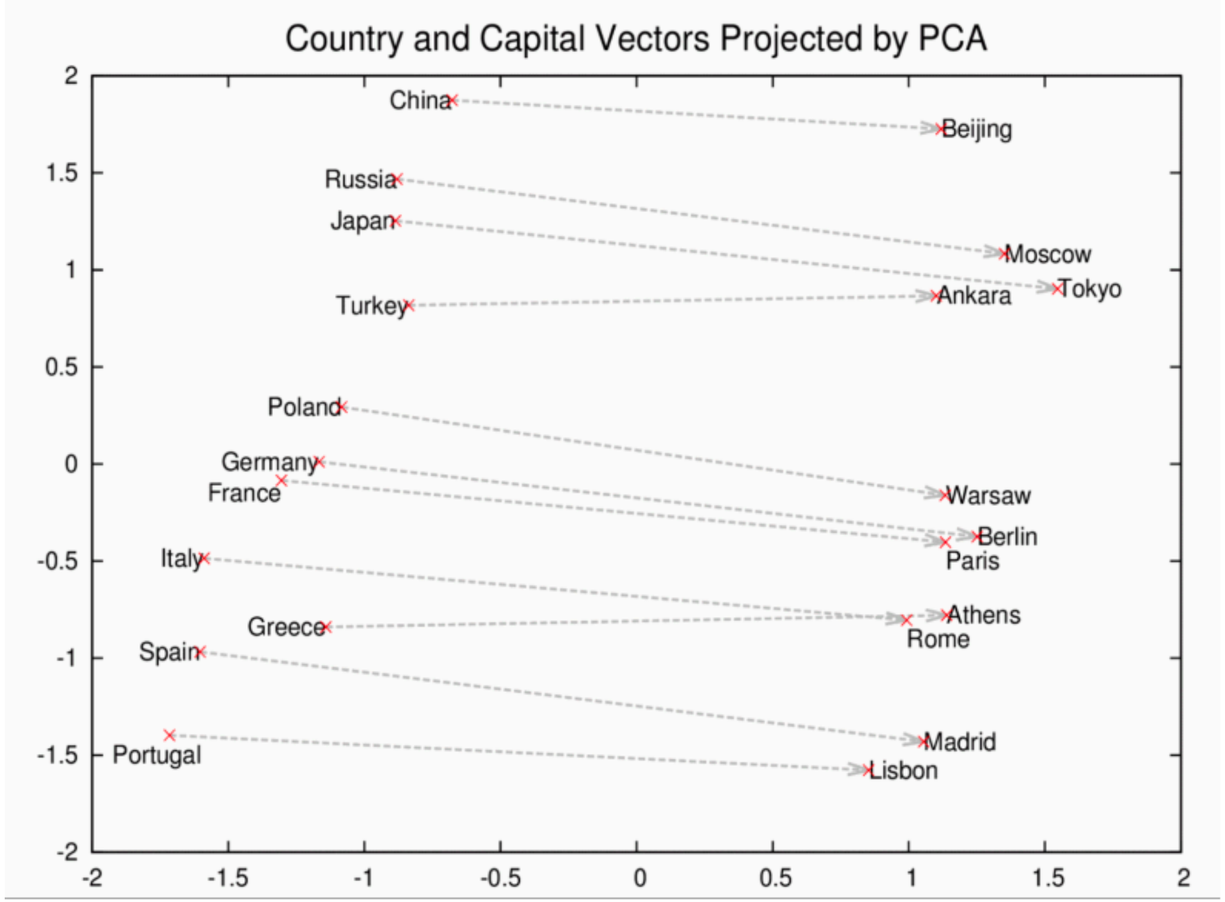
明らかにこのようなベクトルが扱えたほうが良い

Word2Vec

- 自然言語処理ではword2vecがもはやデフォルト
- BoWやone-hotは使わない
- 特に書いてない場合は暗黙的にword2vec
- 本質的には分散表現が大事で、その方法がw2v
- word2vecの求め方→skip-gram, DBOW など

なぜ分散表現？

- 先程のone-hot vectorは単語の意味を反映したベクトルではなかった
- 単語の意味が埋め込まれたベクトルがもし得られるのなら、それだけでも大幅な性能向上が期待できる
  - 似ている概念なら似ているベクトル(たとえばクルマとトラックは近い)
  - 似ていない概念なら似ていないベクトル（電話と魚は遠い）
- しかしそんな都合の良いものが得られるか？→できる！



近いか否かだけでなく、位相関係も埋め込まれている

（ここでは「国名から右に-2.5, 下に0.3くらい進むと首都」ような関係まで得られる）

分布仮説

- 「類似した文脈に出てくる単語は似た意味を持つ」
- 仮説はシンプルだがかなり強力
  - 私は____を食べる　←空白には食べ物が入るだろう
- Word2Vecでは「単語の左右の文脈」が入力として与えられて、「空欄になにが入るか」を予測する問題を解かせる
  - ここでは「私」「は」「を」「食べる」が入力になる

Sentence Piece

Sentence Piece

- 普通、トークン列を得るためには MeCab や JUMAN++ を使って形態素解析をする
- この方法は一見問題ないように思えるが…

形態素解析ベースの問題

- 単語をそのまま使うと語彙サイズに応じてメモリを食うためあまり大きい辞書が使えない
- 普通は高頻度語だけに対応するようにするが、当然低頻度語は捨てなければいけない
- 未知語は完全に諦めるしかない

サブワード

- 通常の意味での単語ではなく、サブワードを使って分かち書きをする
  - まずは普通に単語ごとに分割する
  - 高頻度の単語はそのままにする
  - 低頻度の語は部分文字列や文字単位で分割しなおす
- 事前に決めた語彙数以下になるまで上記を繰り返す

サブワードベースの利点

- 未知語がほぼなくなる
- 教師データ(辞書)がいらない、教師なしで学習可能
- 語彙サイズがかなり節約できる
  - 10万〜100万 → 8k〜32k
- 単語ベースは語彙が多くなりすぎる、文字ベースは細かすぎる、サブワードは両者のいいとこ取り
- しかも精度が下がらない

BPE
Byte Pair Encoding

- 実際には先の例とは逆で、文字ベースから始めてよく出現する部分文字列をマージしていくことで処理する

- 例(入力 ABCDEABABCD から単語リストを得る)

A. ABCDEABABCD ←**「AB」という部分列がよく出てくる**

B. XCDEXXCD [X→AB] ←**「CD」という部分列がよく出てくる**

C. XYEXXY [X→AB, Y→CD] ←**「XY」という部分列がよく出てくる**

D. ZEXZ [X→AB, Y→CD, Z→XY]

単語E, AB, CD, ABCD が得られた

SentencePiece

- 実際にはSentencePieceというライブラリを使ってサブワード分割ができる
- n文字のテキストに対して O(n log n)で処理できるためかなり高速に処理できる
- BPEより賢く分割できる手法もついている