その他の主要技術(1)

前田利之（まえだ　としゆき）
aipr@e-chan.jp

阪南大学　経営情報学部

(2024.1.17)

その他の主要技術

☆時間があれば取り上げたかった話題は沢山あるのですが、その中のうち重要と思われるものを、いくつか拾います（残りは次週）

- 自然言語処理のための
-
-
-

その他の主要技術

☆時間があれば取り上げたかった話題は沢山あるのですが、その中のうち重要と思われるものを、いくつか拾います（残りは次週）

- 自然言語処理のための word2vec
-
-
-

その他の主要技術

☆時間があれば取り上げたかった話題は沢山あるのですが、その中のうち重要と思われるものを、いくつか拾います（残りは次週）

- 自然言語処理のための word2vec
- RNN (Recurrent Neural Network)
-
-

その他の主要技術

☆時間があれば取り上げたかった話題は沢山あるのですが、その中のうち重要と思われるものを、いくつか拾います（残りは次週）

- 自然言語処理のための word2vec
- RNN (Recurrent Neural Network)
- LSTM (Long Short-Term Memory)
-

その他の主要技術

☆時間があれば取り上げたかった話題は沢山あるのですが、その中のうち重要と思われるものを、いくつか拾います（残りは次週）

- 自然言語処理のための word2vec
- RNN (Recurrent Neural Network)
- LSTM (Long Short-Term Memory)
- Transformer

自然言語処理とは

- 人間の話す言葉をコンピュータに理解させる技術
- 単語の意味を理解させることが重要→ [ ]
  - [ ] のベクトルで表現する
  - ベクトルで表現することで、単語の意味を定量的に把握→様々な処理に適用できる
  - 主要手法： [ ] ＝「単語の意味は [ ] の単語によって形成される」

自然言語処理とは

- 人間の話す言葉をコンピュータに理解させる技術
- 単語の意味を理解させることが重要→ **分散表現**
  - [　　]のベクトルで表現する
  - ベクトルで表現することで、単語の意味を定量的に把握→様々な処理に適用できる
  - 主要手法：[　　]＝「単語の意味は[　　]の単語によって形成される」

自然言語処理とは

- 人間の話す言葉をコンピュータに理解させる技術
- 単語の意味を理解させることが重要→ **分散表現**
  - **固定長** のベクトルで表現する
  - ベクトルで表現することで、単語の意味を定量的に把握→様々な処理に適用できる
  - 主要手法：＿＿＿＿＿＿＝「単語の意味は＿＿＿＿＿＿の単語によって形成される」

自然言語処理とは

- 人間の話す言葉をコンピュータに理解させる技術
- 単語の意味を理解させることが重要→ **分散表現**
  - **固定長**のベクトルで表現する
  - ベクトルで表現することで、単語の意味を定量的に把握→様々な処理に適用できる
  - 主要手法：**分布仮説**＝「単語の意味は　　の単語によって形成される」

自然言語処理とは

- 人間の話す言葉をコンピュータに理解させる技術
- 単語の意味を理解させることが重要→ **分散表現**
  - **固定長**のベクトルで表現する
  - ベクトルで表現することで、単語の意味を定量的に把握→様々な処理に適用できる
  - 主要手法： **分布仮説**＝「単語の意味は **周囲**の単語によって形成される」

カウントベースと推論ベース

- カウントベース
  - 周囲の単語の [　　] によって単語を表現する方法
  - コーパス全体の統計データから単語の分散表現を獲得
- 推論ベース
  - ニューラルネットワークを用いて少量の学習サンプルをみながら [　　] を繰り返し更新する
  - 本講座ではこちらの手法の１つである [　　] を取り上げる

カウントベースと推論ベース

- カウントベース
  - 周囲の単語の 頻度 によって単語を表現する方法
  - コーパス全体の統計データから単語の分散表現を獲得
- 推論ベース
  - ニューラルネットワークを用いて少量の学習サンプル をみながら [　　] を繰り返し更新する
  - 本講座ではこちらの手法の１つである [　　] を取り上げる

カウントベースと推論ベース

- カウントベース
  - 周囲の単語の 頻度 によって単語を表現する方法
  - コーパス全体の統計データから単語の分散表現を獲得
- 推論ベース
  - ニューラルネットワークを用いて少量の学習サンプル をみながら [　　] を繰り返し更新する
  - 本講座ではこちらの手法の１つである [　　] を取り上げる

カウントベースと推論ベース

- カウントベース
  - 周囲の単語の **頻度** によって単語を表現する方法
  - コーパス全体の統計データから単語の分散表現を獲得
- 推論ベース
  - ニューラルネットワークを用いて少量の学習サンプルをみながら **重み** を繰り返し更新する
  - 本講座ではこちらの手法の１つである [　　] を取り上げる

カウントベースと推論ベース

- カウントベース
  - 周囲の単語の **頻度** によって単語を表現する方法
  - コーパス全体の統計データから単語の分散表現を獲得
- 推論ベース
  - ニューラルネットワークを用いて少量の学習サンプルをみながら **重み** を繰り返し更新する
  - 本講座ではこちらの手法の１つである **word2vec** を取り上げる

word2vec

- [ ] [ ] モデルを用いる
  - コンテクスト（文脈… [ ] ）からターゲットを推測することを目的としたニューラルネットワーク
- 前後のコンテクストをどの程度利用するかはモデル作成ごとに判断
- 前後１単語をコンテクストとする場合、下の例だと「朝」「起き」から「？」の単語を推測することになる

今日	は	朝	？	起き	まし	た	。

word2vec

- CBOW [　　] モデルを用いる
  - コンテクスト（文脈… [　　]）からターゲットを推測することを目的としたニューラルネットワーク
- 前後のコンテクストをどの程度利用するかはモデル作成ごとに判断
- 前後１単語をコンテクストとする場合、下の例だと「朝」「起き」から「？」の単語を推測することになる

今日	は	朝	？	起き	まし	た	。

word2vec

- CBOW (continuous bag-of-words) モデルを用いる
  - コンテクスト（文脈… ） からターゲットを推測することを目的としたニューラルネットワーク
- 前後のコンテクストをどの程度利用するかはモデル作成ごとに判断
- 前後１単語をコンテクストとする場合、下の例だと「朝」「起き」から「？」の単語を推測することになる

今日	は	朝	？	起き	まし	た	。

word2vec

- CBOW (continuous bag-of-words) モデルを用いる
  - コンテクスト（文脈… 前後の関係）からターゲットを推測することを目的としたニューラルネットワーク
- 前後のコンテクストをどの程度利用するかはモデル作成ごとに判断
- 前後１単語をコンテクストとする場合、下の例だと「朝」「起き」から「？」の単語を推測することになる

今日	は	朝	？	起き	まし	た	。

word2vec

- 入力層が二つあり、中間層を経て出力層となる
- 中間層は各入力層の全結合による変換後の値が [　　] されたものになる
  - 一つ目のの入力層が h_1 二つ目の入力層が h_2 に変換されたとすると中間層のニューロンは $\frac{(h_1 + h_2)}{2}$ になる
- 入力層から中間層への変換は、全結合層 (重みは W_{in}) によって行われる→この重み=CBOW を用いて作る単語の [　　]
  - この時は全結合層の重みは 8 × 38 × 3 の形状の行列

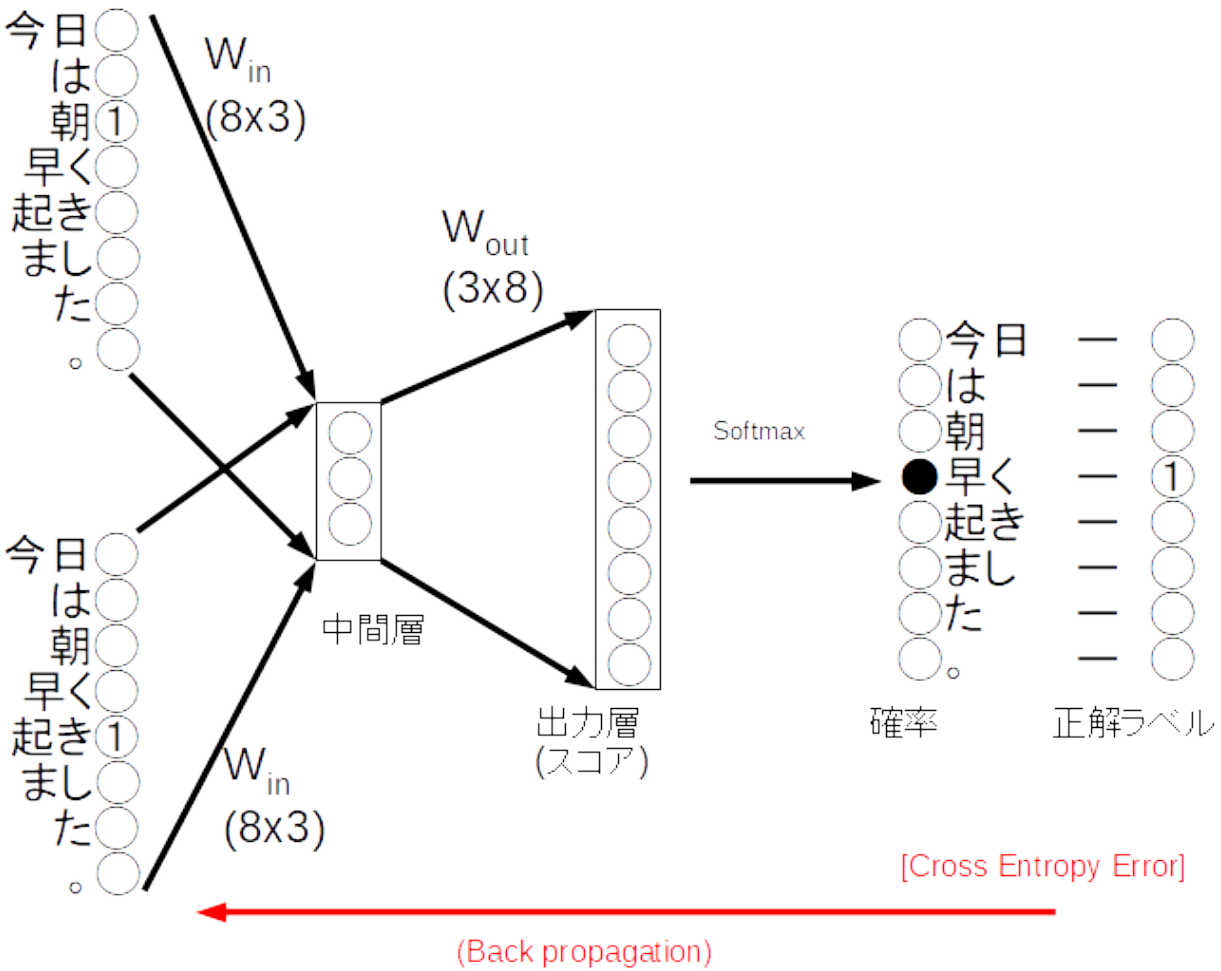
word2vec

- 入力層が二つあり、中間層を経て出力層となる
- 中間層は各入力層の全結合による変換後の値が平均されたものになる
  - 一つ目のの入力層が h_1 二つ目の入力層が h_2 に変換されたとすると中間層のニューロンは $\frac{(h_1 + h_2)}{2}$ になる
- 入力層から中間層への変換は、全結合層 (重みは W_{in}) によって行われる→この重み=CBOW を用いて作る単語の
  - この時は全結合層の重みは 8 × 38 × 3 の形状の行列

word2vec

- 入力層が二つあり、中間層を経て出力層となる
- 中間層は各入力層の全結合による変換後の値が**平均**されたものになる
  - 一つ目のの入力層が h_1 二つ目の入力層が h_2 に変換されたとすると中間層のニューロンは $\frac{(h_1 + h_2)}{2}$ になる
- 入力層から中間層への変換は、全結合層 (重みは W_{in}) によって行われる→この重み=CBOW を用いて作る単語の**分散表現**
  - この時は全結合層の重みは 8 × 38 × 3 の形状の行列

CBOW モデルの学習


今日
は
朝
早く
起き
まし
た
。
W_{in}
(8x3)
W_{out}
(3x8)
中間層
出力層
(スコア)
Softmax
今日
は
朝
早く
起き
まし
た
。
確率
正解ラベル
今日
は
朝
早く
起き
まし
た
。
W_{in}
(8x3)
[Cross Entropy Error]
(Back propagation)

CBOW モデルの学習

- 例ではコンテクストは「朝」「起き」、ニューラルネットワークが予想したい単語が「早く」
- 適切な重みを持ったニューラルネットワークでは [ ] を表すニューロンにおいて正解ニューロンが高くなっていることが期待できる
- CBOW の学習では正解ラベルとニューラルネットワークが出力した [ ] の交差エントロピー誤差を求め、それを損失としてその損失を少なくしていく方向に学習を進める

CBOW モデルの学習

- 例ではコンテクストは「朝」「起き」、ニューラルネットワークが予想したい単語が「早く」
- 適切な重みを持ったニューラルネットワークでは 確率 を表すニューロンにおいて正解ニューロンが高くなっていることが期待できる
- CBOW の学習では正解ラベルとニューラルネットワークが出力した　　の交差エントロピー誤差を求め、それを損失としてその損失を少なくしていく方向に学習を進める

CBOW モデルの学習

- 例ではコンテクストは「朝」「起き」、ニューラルネットワークが予想したい単語が「早く」
- 適切な重みを持ったニューラルネットワークでは確率を表すニューロンにおいて正解ニューロンが高くなっていることが期待できる
- CBOW の学習では正解ラベルとニューラルネットワークが出力した確率の交差エントロピー誤差を求め、それを損失としてその損失を少なくしていく方向に学習を進める

CBOW モデルの損失関数

- モデルを作成するに当たって用いるコンテクストを前後 [　　] とした場合、

$$L = -\frac{1}{T}\sum_{t=1}^{T} \log P(w_t | w_{t-1}, w_{t+1})$$

- 上記損失関数をできるだけ小さくしていく方向で学習していくことで、その時の重みを単語の [　　] として獲得することができる

CBOW モデルの損失関数

- モデルを作成するに当たって用いるコンテクストを前後 1 単語 とした場合、

$$L = -\frac{1}{T}\sum_{t=1}^{T} \log P(w_t | w_{t-1}, w_{t+1})$$

- 上記損失関数をできるだけ小さくしていく方向で学習していくことで、その時の重みを単語の　　として獲得することができる

CBOW モデルの損失関数

- モデルを作成するに当たって用いるコンテクストを前後 **1 単語** とした場合、

$$L = -\frac{1}{T}\sum_{t=1}^{T} \log P(w_t | w_{t-1}, w_{t+1})$$

- 上記損失関数をできるだけ小さくしていく方向で学習していくことで、その時の重みを単語の **分散表現** として獲得することができる

RNN とは

- 日本語では「　　ニューラルネットワーク」
- 自然言語文や時系列データなどの　　データのパターンを認識するように設計されたニューラルネットワークのモデル
  - 多層パーセプロトロンや CNN などのニューラルネットワークモデルでは、入力サンプルの　　を処理することはできない
  - RNN は過去の情報を記憶しておき、その情報にしたがって新しい事象を処理することができる

RNN とは

- 日本語では「 再帰型 ニューラルネットワーク」
- 自然言語文や時系列データなどの [ ] データのパターンを認識するように設計されたニューラルネットワークのモデル
  - 多層パーセプロトロンや CNN などのニューラルネットワークモデルでは、入力サンプルの [ ] を処理することはできない
  - RNN は過去の情報を記憶しておき、その情報にしたがって新しい事象を処理することができる

RNN とは

- 日本語では「再帰型ニューラルネットワーク」
- 自然言語文や時系列データなどのシーケンスデータのパターンを認識するように設計されたニューラルネットワークのモデル
  - 多層パーセプロトロンや CNN などのニューラルネットワークモデルでは、入力サンプルの　　を処理することはできない
  - RNN は過去の情報を記憶しておき、その情報にしたがって新しい事象を処理することができる

RNN とは

- 日本語では「再帰型ニューラルネットワーク」
- 自然言語文や時系列データなどのシーケンスデータのパターンを認識するように設計されたニューラルネットワークのモデル
  - 多層パーセプロトロンや CNN などのニューラルネットワークモデルでは、入力サンプルの順序を処理することはできない
  - RNN は過去の情報を記憶しておき、その情報にしたがって新しい事象を処理することができる

RNN の損失関数

- RNN は状態を扱う有向閉路を持つニューラルネットワークであり、この有向閉路をリカレントエッジと呼ぶ
- 1990 年代に RNN の学習アルゴリズムである

  が紹介される
- 全損失 L を時間 $t = 1$ から $t = T$ までのすべての損失関数の総和であると考える

$$L = \sum_{t=1}^{T} L^{(t)}$$

RNN の損失関数

- RNN は状態を扱う有向閉路を持つニューラルネットワークであり、この有向閉路をリカレントエッジと呼ぶ
- 1990 年代に RNN の学習アルゴリズムである BPTT(BackPropagation Through Time) が紹介される
- 全損失 L を時間 $t = 1$ から $t = T$ までのすべての損失関数の総和であると考える

$$L = \sum_{t=1}^{T} L^{(t)}$$

RNN の損失関数

- 勾配は次のようになる

$$\frac{\partial L^{(t)}}{\partial W_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \times \frac{\partial y^{(t)}}{\partial h^{(t)}} \times \left(\sum_{k=1}^{t} \frac{\partial h^{(t)}}{\partial h^{(k)}} \times \frac{\partial h^{(k)}}{\partial W_{hh}} \right),$$

$$\frac{\partial h^{(t)}}{\partial h^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial h^{(i)}}{\partial h^{(i-1)}}$$

- 損失関数の勾配を計算するときの乗法係数 $\frac{\partial h^{(t)}}{\partial h^{(k)}}$ により、いわゆる [　　] と [　　] が発生する

RNN の損失関数

- 勾配は次のようになる

$$\frac{\partial L^{(t)}}{\partial W_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \times \frac{\partial y^{(t)}}{\partial h^{(t)}} \times \left(\sum_{k=1}^{t} \frac{\partial h^{(t)}}{\partial h^{(k)}} \times \frac{\partial h^{(k)}}{\partial W_{hh}} \right),$$

$$\frac{\partial h^{(t)}}{\partial h^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial h^{(i)}}{\partial h^{(i-1)}}$$

- 損失関数の勾配を計算するときの乗法係数 $\frac{\partial h^{(t)}}{\partial h^{(k)}}$ により、いわゆる **勾配消失問題** と [　　] が発生する

RNN の損失関数

- 勾配は次のようになる

$$\frac{\partial L^{(t)}}{\partial W_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \times \frac{\partial y^{(t)}}{\partial h^{(t)}} \times \left(\sum_{k=1}^{t} \frac{\partial h^{(t)}}{\partial h^{(k)}} \times \frac{\partial h^{(k)}}{\partial W_{hh}} \right),$$

$$\frac{\partial h^{(t)}}{\partial h^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial h^{(i)}}{\partial h^{(i-1)}}$$

- 損失関数の勾配を計算するときの乗法係数 $\frac{\partial h^{(t)}}{\partial h^{(k)}}$ により、いわゆる **勾配消失問題** と **勾配発散問題** が発生する

勾配消失問題とは

- 何時まで経っても [　　] ならない…どこかで勾配が [　] になってしまうと、もうこの重みは何度計算を繰り返しても全く値が変わらなくなってしまう
- 多層化した時に表れ、活性化関数の [　　　　] が勾配を消失させる…sigmoid の場合、微分すると max. 0.25 なので、何回も掛ける間に限りなく [　] に近づいてしまう
- （…勾配発散問題は詳細略）

勾配消失問題とは

- 何時まで経っても 賢く ならない…どこかで勾配が □ になってしまうと、もうこの重みは何度計算を繰り返しても全く値が変わらなくなってしまう
- 多層化した時に表れ、活性化関数の □ が勾配を消失させる…sigmoid の場合、微分すると max. 0.25 なので、何回も掛ける間に限りなく □ に近づいてしまう
- （…勾配発散問題は詳細略）

勾配消失問題とは

- 何時まで経っても 賢く ならない…どこかで勾配が 0 になってしまうと、もうこの重みは何度計算を繰り返しても全く値が変わらなくなってしまう
- 多層化した時に表れ、活性化関数の [　　] が勾配を消失させる…sigmoid の場合、微分すると max. 0.25 なので、何回も掛ける間に限りなく [　] に近づいてしまう
- （…勾配発散問題は詳細略）

勾配消失問題とは

- 何時まで経っても 賢く ならない…どこかで勾配が 0 になってしまうと、もうこの重みは何度計算を繰り返しても全く値が変わらなくなってしまう
- 多層化した時に表れ、活性化関数の 掛け算の連続 が勾配を消失させる…sigmoid の場合、微分すると max. 0.25 なので、何回も掛ける間に限りなく □ に近づいてしまう
- （…勾配発散問題は詳細略）

勾配消失問題とは

- 何時まで経っても **賢く** ならない…どこかで勾配が **0** になってしまうと、もうこの重みは何度計算を繰り返しても全く値が変わらなくなってしまう
- 多層化した時に表れ、活性化関数の **掛け算の連続** が勾配を消失させる…sigmoid の場合、微分すると max. 0.25 なので、何回も掛ける間に限りなく **0** に近づいてしまう
- （…勾配発散問題は詳細略）

LSTM とは

- 勾配消失問題を解決する方法として 1997 年に提唱されたもの
- RNN の中間層のユニットを [　　] と呼ばれるメモリと 3 つの [　　] を持つブロックに置き換えることで実現されている
- [　　]：情報の取捨選択機構（選択的に情報を通す）

LSTM とは

- 勾配消失問題を解決する方法として 1997 年に提唱されたもの
- RNN の中間層のユニットを LSTM Block と呼ばれるメモリと 3 つの を持つブロックに置き換えることで実現されている
- ：情報の取捨選択機構（選択的に情報を通す）

LSTM とは

- 勾配消失問題を解決する方法として 1997 年に提唱されたもの
- RNN の中間層のユニットを LSTM Block と呼ばれるメモリと 3 つの ゲート を持つブロックに置き換えることで実現されている
- ：情報の取捨選択機構（選択的に情報を通す）

LSTM とは

- 勾配消失問題を解決する方法として 1997 年に提唱されたもの
- RNN の中間層のユニットを LSTM Block と呼ばれるメモリと 3 つの ゲート を持つブロックに置き換えることで実現されている
- ゲート：情報の取捨選択機構（選択的に情報を通す）

LSTM （続き）

☆ゲートの種類

- [　　]: 入力を取り込むかどうかを選ぶ→セル状態を更新する役割を果たす
- [　　]: cell に保持している情報をリセットするかどうかを選ぶ→通過させる情報と通過させない情報を決定
- [　　]: 次の時刻にどの程度情報を伝えるか選ぶ→隠れユニットの値の更新方法を決定

LSTM （続き）

☆ゲートの種類

- Input gate: 入力を取り込むかどうかを選ぶ→セル状態を更新する役割を果たす
- : cell に保持している情報をリセットするかどうかを選ぶ→通過させる情報と通過させない情報を決定
- : 次の時刻にどの程度情報を伝えるか選ぶ→隠れユニットの値の更新方法を決定

LSTM （続き）

☆ゲートの種類

- Input gate: 入力を取り込むかどうかを選ぶ→セル状態を更新する役割を果たす
- Forget gate: cell に保持している情報をリセットするかどうかを選ぶ→通過させる情報と通過させない情報を決定
- : 次の時刻にどの程度情報を伝えるか選ぶ→隠れユニットの値の更新方法を決定

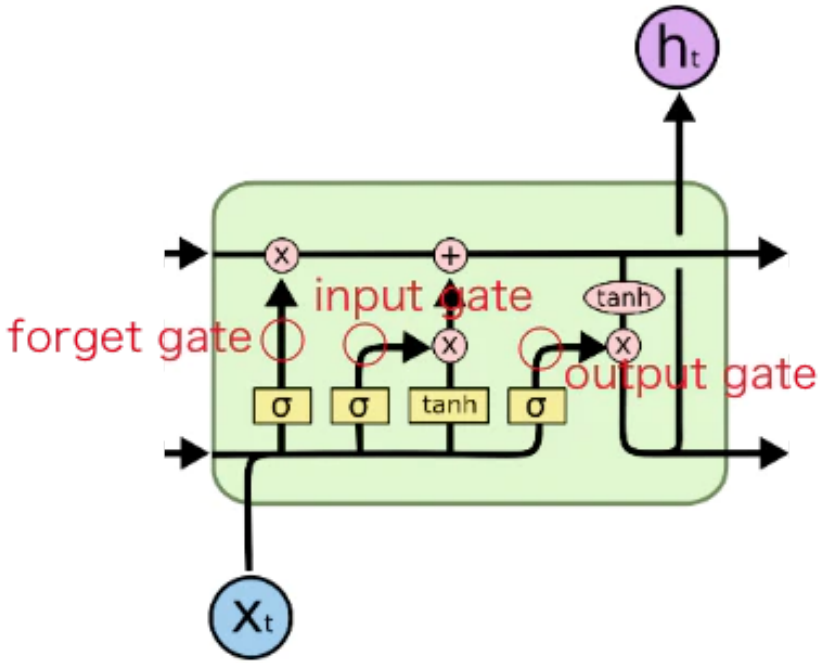
LSTM （続き）

☆ゲートの種類

- Input gate: 入力を取り込むかどうかを選ぶ→セル状態を更新する役割を果たす
- Forget gate: cell に保持している情報をリセットするかどうかを選ぶ→通過させる情報と通過させない情報を決定
- Output gate: 次の時刻にどの程度情報を伝えるか選ぶ→隠れユニットの値の更新方法を決定

LSTM （続き）

☆ユニットの構成



(https://qiita-user-contents.imgix.net/https%3A%2F%2Fqiita-image-store.s3.ap-northeast-1.amazonaws.com%2F0%2F265344%2Fb88b9113-72ac-8b49-3cee-e803fba9543f.png?ixlib=rb-1.2.2&auto=format&gif-q=60&q=75&s=bc552d6916badfb11a38ac91a3f9523b)

Transformer

- 2017 年に発表された深層学習モデルであり、当初は [　　] の分野で活用されていたが、のちに種々の分野で適用されている
- RNN,LSTM 同様、自然言語などの [　　] データを扱うが、時系列データを逐次処理する必要がない

Transformer

- 2017 年に発表された深層学習モデルであり、当初は 自然言語処理 の分野で活用されていたが、のちに種々の分野で適用されている
- RNN,LSTM 同様、自然言語などの [　　] データを扱うが、時系列データを逐次処理する必要がない

Transformer

- 2017 年に発表された深層学習モデルであり、当初は **自然言語処理** の分野で活用されていたが、のちに種々の分野で適用されている
- RNN,LSTM 同様、自然言語などの **時系列** データを扱うが、時系列データを逐次処理する必要がない

Transformer (cont.)

- Transformer では RNN よりも多くの [　　] になり学習時間が短かくなる
- [　　] (Bidirectional Encoder Representations from Transformers) や [　　] (Generative Pre-trained Transformers) などの事前トレーニング済みシステムの開発につながった
- →今日の [　　] ブームの主人公

Transformer (cont.)

- Transformer では RNN よりも多くの **並列化が可能** になり学習時間が短かくなる
- [　　] (Bidirectional Encoder Representations from Transformers) や [　　] (Generative Pre-trained Transformers) などの事前トレーニング済みシステムの開発につながった
- →今日の [　　] ブームの主人公

Transformer (cont.)

- Transformer では RNN よりも多くの **並列化が可能** になり学習時間が短かくなる
- **BERT** (Bidirectional Encoder Representations from Transformers) や [　　] (Generative Pre-trained Transformers) などの事前トレーニング済みシステムの開発につながった
- →今日の [　　] ブームの主人公

Transformer (cont.)

- Transformer では RNN よりも多くの **並列化が可能** になり学習時間が短かくなる
- **BERT** (Bidirectional Encoder Representations from Transformers) や **GPT** (Generative Pre-trained Transformers) などの事前トレーニング済みシステムの開発につながった
- →今日の [　　] ブームの主人公

Transformer (cont.)

- Transformer では RNN よりも多くの **並列化が可能** になり学習時間が短かくなる
- **BERT** (Bidirectional Encoder Representations from Transformers) や **GPT** (Generative Pre-trained Transformers) などの事前トレーニング済みシステムの開発につながった
- →今日の **生成 AI** ブームの主人公

Transformer (cont.)

- Transformer の基本的な構成要素は、Scaled dot-product attention unit と表現される

  [ ]単位

  - 複数個の入力の内、どこを [ ]か学習する仕組み
  - [ ]層は以前のすべての状態にアクセスでき、現在のトークンとの関連性の学習値に従ってそれらを重み付けして遠く離れた関連するトークンに関する明瞭な情報を提供する
  - [ ]機構により、モデルは文の前の [ ]の状態を直接見て、そこから引き出すことができる

Transformer (cont.)

- Transformer の基本的な構成要素は、Scaled dot-product attention unit と表現される

「アテンション」単位

  - 複数個の入力の内、どこを [ ] か学習する仕組み
  - [ ] 層は以前のすべての状態にアクセスでき、現在のトークンとの関連性の学習値に従ってそれらを重み付けして遠く離れた関連するトークンに関する明瞭な情報を提供する
  - [ ] 機構により、モデルは文の前の [ ] の状態を直接見て、そこから引き出すことができる

Transformer (cont.)

- Transformer の基本的な構成要素は、Scaled dot-product attention unit と表現される「アテンション」単位
  - 複数個の入力の内、どこを 注目すべき か学習する仕組み
  - [ ] 層は以前のすべての状態にアクセスでき、現在のトークンとの関連性の学習値に従ってそれらを重み付けして遠く離れた関連するトークンに関する明瞭な情報を提供する
  - [ ] 機構により、モデルは文の前の [ ] の状態を直接見て、そこから引き出すことができる

Transformer (cont.)

- Transformer の基本的な構成要素は、Scaled dot-product attention unit と表現される「アテンション」単位
  - 複数個の入力の内、どこを 注目すべき か学習する仕組み
  - アテンション 層は以前のすべての状態にアクセスでき、現在のトークンとの関連性の学習値に従ってそれらを重み付けして遠く離れた関連するトークンに関する明瞭な情報を提供する
  - [ ] 機構により、モデルは文の前の [ ] の状態を直接見て、そこから引き出すことができる

Transformer (cont.)

- Transformer の基本的な構成要素は、Scaled dot-product attention unit と表現される「アテンション」単位
  - 複数個の入力の内、どこを 注目すべき か学習する仕組み
  - アテンション 層は以前のすべての状態にアクセスでき、現在のトークンとの関連性の学習値に従ってそれらを重み付けして遠く離れた関連するトークンに関する明瞭な情報を提供する
  - アテンション 機構により、モデルは文の前の [ ] の状態を直接見て、そこから引き出すことができる

Transformer (cont.)

- Transformer の基本的な構成要素は、Scaled dot-product attention unit と表現される「アテンション」単位
  - 複数個の入力の内、どこを 注目すべき か学習する仕組み
  - アテンション 層は以前のすべての状態にアクセスでき、現在のトークンとの関連性の学習値に従ってそれらを重み付けして遠く離れた関連するトークンに関する明瞭な情報を提供する
  - アテンション 機構により、モデルは文の前の 任意の時点 の状態を直接見て、そこから引き出すことができる

Transformer (cont.)

- 文が Transformer モデルに渡されると、[ ]の重みがすべてのトークン間で同時に計算される
- [ ]単位は、コンテキスト内の全ての[ ]の埋め込みを生成する
- そこには[ ]自体の情報だけでなく、他の関連[ ]との関連について[ ]の重みで重み付けされたものも含まれる。

Transformer (cont.)

- 文が Transformer モデルに渡されると、アテンションの重みがすべてのトークン間で同時に計算される
- [　　]単位は、コンテキスト内の全ての[　　]の埋め込みを生成する
- そこには[　　]自体の情報だけでなく、他の関連[　　]との関連について[　　]の重みで重み付けされたものも含まれる。

Transformer (cont.)

- 文が Transformer モデルに渡されると、アテンションの重みがすべてのトークン間で同時に計算される
- アテンション単位は、コンテキスト内の全ての [ ] の埋め込みを生成する
- そこには [ ] 自体の情報だけでなく、他の関連 [ ] との関連について [ ] の重みで重み付けされたものも含まれる。

Transformer (cont.)

- 文が Transformer モデルに渡されると、アテンションの重みがすべてのトークン間で同時に計算される
- アテンション単位は、コンテキスト内の全てのトークンの埋め込みを生成する
- そこには　　自体の情報だけでなく、他の関連　　との関連について　　の重みで重み付けされたものも含まれる。

Transformer (cont.)

- 文が Transformer モデルに渡されると、アテンションの重みがすべてのトークン間で同時に計算される
- アテンション単位は、コンテキスト内の全てのトークンの埋め込みを生成する
- そこにはトークン自体の情報だけでなく、他の関連との関連について の重みで重み付けされたものも含まれる。

Transformer (cont.)

- 文が Transformer モデルに渡されると、アテンションの重みがすべてのトークン間で同時に計算される
- アテンション単位は、コンテキスト内の全てのトークンの埋め込みを生成する
- そこにはトークン自体の情報だけでなく、他の関連トークンとの関連について　　の重みで重み付けされたものも含まれる。

Transformer (cont.)

- 文が Transformer モデルに渡されると、アテンションの重みがすべてのトークン間で同時に計算される
- アテンション単位は、コンテキスト内の全てのトークンの埋め込みを生成する
- そこにはトークン自体の情報だけでなく、他の関連トークンとの関連についてアテンションの重みで重み付けされたものも含まれる。

Transformer (BERT) の利用例（実装）

1. まず、bert_ex.ipynb (zip で圧縮済) のリンク先を Colab Notebooks の中（ mymodules の中ではない！）に保存し、展開しておく
2. Chrome または Edge を立ち上げ Google Drive からファイルをひらく
3. 「ライタイム」メニューの「ランタイムのタイプを変更」で GPU を選ぶ（元々なっている場合もあるが、確認する）
4. ５分割されているので、上から順番に実行する。３つ目で wandb.ai のアカウントを作らされるので Google アカウントでそのまま作る。
5. 一番下の分類結果を確認する

Transformer (BERT) の利用例（日本語の感情分析）

1. 東北大学で開発されたモデルを使わせてもらって、まずは穴埋めを試みる
2. 確認用のデータセットを作って試してみる
3. megagonlabs の jrte-corpus/master/data/pn.tsv を使って学習をおこなう
4. 上でつくったモデルで評価する
5. 例文をあたえて分類結果を見る

おしまい

質問・コメント等あれば、何でもお気軽に
aipr@e-chan.jp に
メールください！