

## 1. section1\_再帰型ニューラルネットワークの概念

### 演習チャレンジ

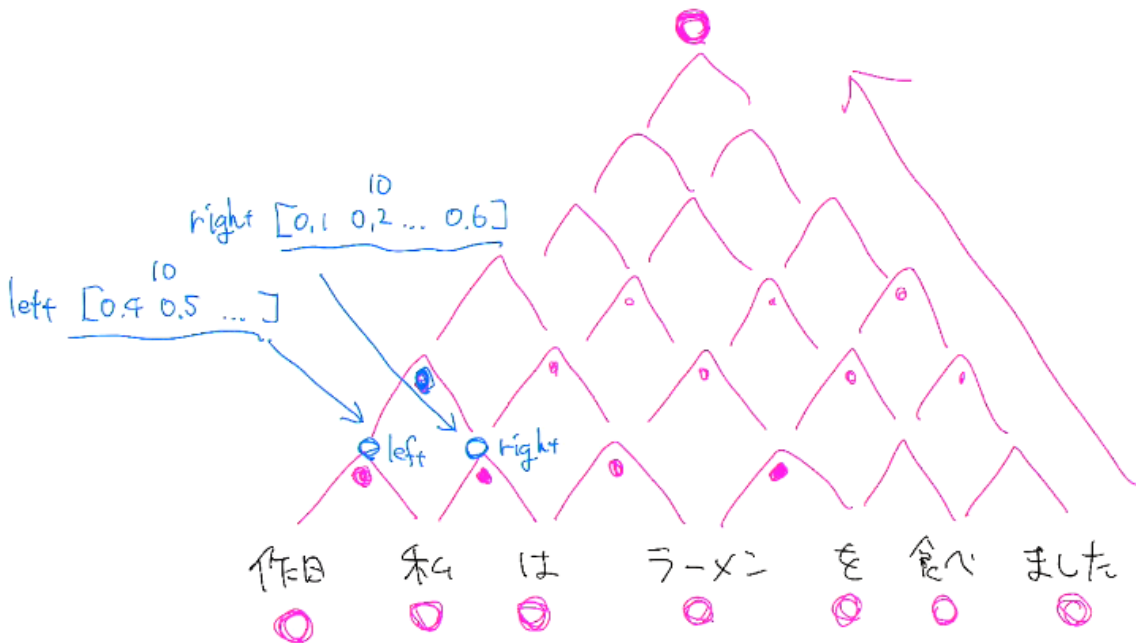
以下は再帰型ニューラルネットワークにおいて構文木を入力として再帰的に文全体の表現ベクトルを得るプログラムである。ただし、ニューラルネットワークの重みパラメータはグローバル変数として定義してあるものとし、`_activation`関数はなんらかの活性化関数であるとする。木構造は再帰的な辞書で定義しており、`root`が最も外側の辞書であると仮定する。

(`<>`) にあてはまるのはどれか。

```
def traverse(node):  
    """  
    node: tree node, recursive dict, {left: node', right: node''}  
    if leaf, word embed vector, (embed_size,)  
  
    W: weights, global variable, (embed_size, 2*embed_size)  
    b: bias, global variable, (embed_size,)  
    """  
    if not isinstance(node, dict):  
        v = node  
    else:  
        left = traverse(node['left'])  
        right = traverse(node['right'])  
        v = _activation(  
            ( <> )  
        )  
    return v
```

(解答) 2

構文木



単語同士の特徴量

Concatenate

$$\begin{bmatrix} 0.4 & 0.5 & \dots & 0.1 & 0.2 & \dots & 0.6 \end{bmatrix}$$

1 0 + 1 0 = 2 0 個になる → 重みを掛ける。 → 1 0 個にする。

## 演習チャレンジ

Study

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。

```
def gradient_clipping(grad, threshold):  
    """  
    grad: gradient  
    """  
    norm = np.linalg.norm(grad)  
    rate = threshold / norm  
    if rate < 1:  
        return (さ)  
    return grad
```

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

(解答) (1)

勾配のクリッピングを行うので `rate` を掛ける。

# 演習チャレンジ

Study-AI

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作用させる関数である。

(け) にあてはまるのはどれか。

```
def lstm(x, prev_h, prev_c, W, U, b):  
    """  
    x: inputs, (batch_size, input_size)  
    prev_h: outputs at the previous time step, (batch_size, state_size)  
    prev_c: cell states at the previous time step, (batch_size, state_size)  
    W: upward weights, (4*state_size, input_size)  
    U: lateral weights, (4*state_size, state_size)  
    b: bias, (4*state_size,)  
    """  
    # セルへの入力やゲートをまとめて計算し、分割  
    lstm_in = _activation(x.dot(W.T) + prev_h.dot(U.T) + b)  
    a, i, f, o = np.hsplit(lstm_in, 4)  
    # 値を定数, セルへの入力(-1, 1), ゲート:(0, 1)  
    a = np.tanh(a)  
    input_gate = _sigmoid(i)  
    forget_gate = _sigmoid(f)  
    output_gate = _sigmoid(o)  
    # セルの状態を更新し、中間層の出力を計算  
    c =   
    h = output_gate * np.tanh(c)  
    return c, h
```

- (1)  $\text{output\_gate} * a + \text{forget\_gate} * c$
- (2)  $\text{forget\_gate} * a + \text{output\_gate} * c$
- (3)  $\text{input\_gate} * a + \text{forget\_gate} * c$
- (4)  $\text{forget\_gate} * a + \text{input\_gate} * c$

C E C に覚えさせる情報を説明している。

(解答) (3)

正解: 3

【解説】

新しいセルの状態は、計算されたセルへの入力と1ステップ前のセルの状態に入力ゲート、忘却ゲートを掛けて足し合わせたものと表現される。つまり、 $\text{input\_gate} * a + \text{forget\_gate} * c$ である。

### 3. section3\_GRU

## 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)  $(1-z) * h + z * h\_bar$

(解答) (4)

#### ● LSTM と GRU の違いを簡潔に述べよ

##### (1) L S T M

入力ゲート、出力ゲート、忘却ゲート、C E Cはパラメータが多い

##### (2) G R U

リセットゲート、更新ゲート

G R Uの郷がパラメータが少なく、計算時間が早い

# 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
```

## 正解 : 4

### 【解説】

双方向RNNでは、順方向と逆方向に伝播したときの中間層表現をあわせたものが特徴量となるので、`np.concatenate([h_f, h_b[::-1]], axis=1)`である。

- (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) (4)



# 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 embedding 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
```

- (1) `E.dot(w)`
- (2) `E.T.dot(w)`
- (3) `w.dot(E.T)`
- (4) `E * w`

（解答）（1）

## 6. section6\_Word2Vec

Word2Vec は、単語をベクトルとして表現し、意味の近さや、次に示す「計算のアナロジー」を行えるようにするための仕組みである。2013年に Google 社によって開発された。

Word2Vec を用いることで、単語同士の意味関係をうまく捉え、扱うことができるようになる。

・王様 . . . . . (0.4,0.1,0.9,0.4)

・女王 . . . . . (0.5,0.2,0.3,0.4)

・男 . . . . . (0.1,0.0,0.8,0.2)

・女 . . . . . (0.2,0.1,0.2,0.3)

ここから、ベクトルが指す天童市の距離を計算し、

・「王様」と「女王」の距離は近い

・「王様」－「男」＋「女」 $\simeq$ 「女王」

というような、単語の意味関係を捉えたり、足し引きしたりということをベクトル同士の演算によって行うことができる。

## 7. section7\_Attention Mechanism

2017年、RNNやCNNを使わずに系列データを扱う **Transformer** というモデルが提案された。

**Transformer** は Seq2Seq と同じようにエンコーダ、デコーダからなる。エンコードとデコードはそれぞれ N 回繰り返される。

**Transformer** の内部にはいくつかのアテンション機構が配置されている。それらアテンション機構の基本モデルはスケール・ドットプロダクト・アテンションと呼ばれている。

### (1) アテンション機構

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{Q}\mathbf{K}^T) \cdot \mathbf{V}$$

**Q** : クエリー

**K** : キー

**V** : バリュー

これは、質問文 (**Query**) を入れると、参照すべき場所 (**Key**) が決まり、その場所の回答候補 (**Value**) が得られる。

**Transformer** はソフトアテンションを行っているため、参照すべき場所の重み (**Key**) が決まり、その重みに応じて平均化することで回答候補を得ている。

### (2) スケール・ドットプロダクト・アテンション

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{Q}\mathbf{K}^T / \sqrt{d_k}) \cdot \mathbf{V}$$