深層学習 Part 4

東京海洋大学

竹縄知之

目次

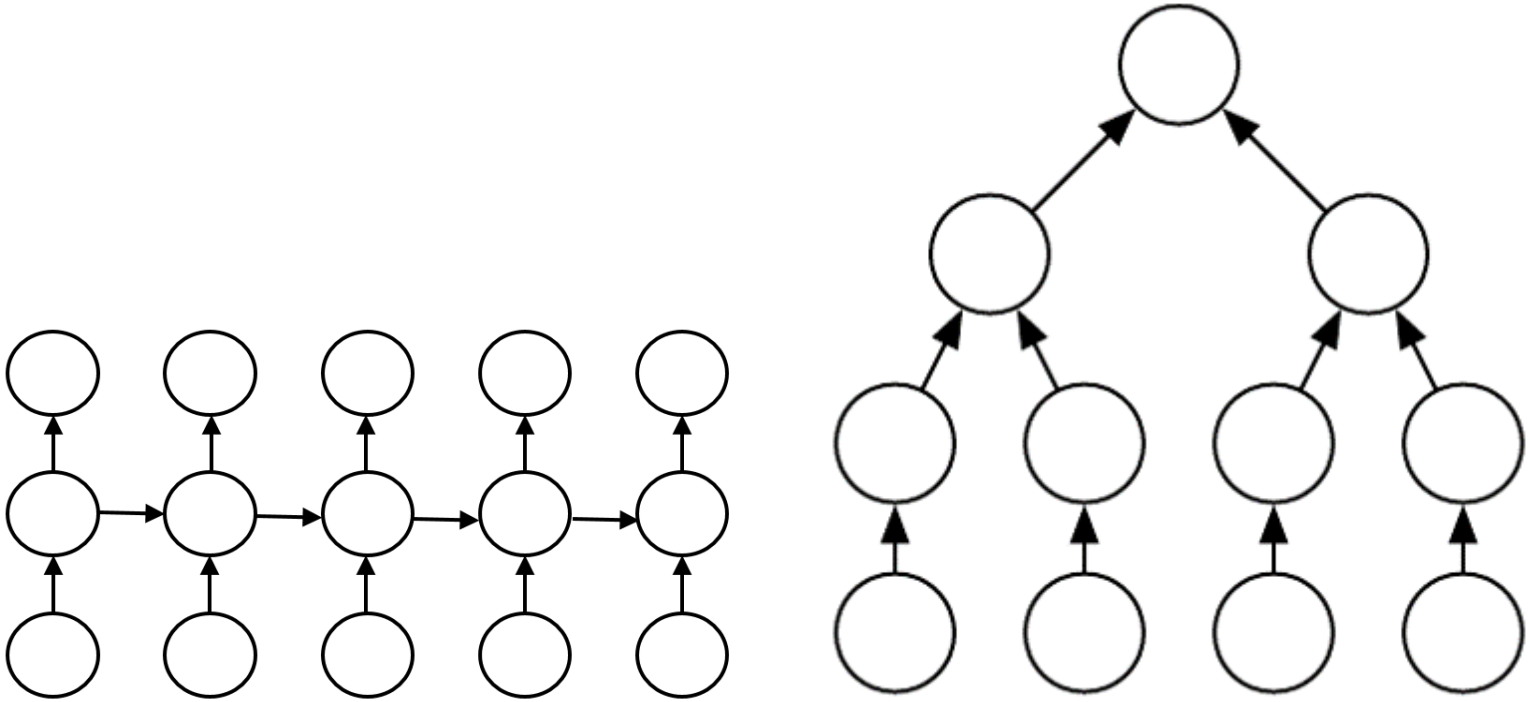
1　回帰結合型ネットワーク

回帰結合型ニューラルネットワーク（または回帰型ニューラルネットワーク）（Recurrent Neural Networks, RNN）は時系列データを処理するためのニューラルネットワークでの一つです．時系列データは系列の開始位置がデータごとに異なるため，通常のニューラルネットワークでは学習しづらいですが，RNN では同じパラメータを周期的に利用することにより，開始位置に依らない学習を実現しています．

なお，RNN は「再帰型」ニューラルネットワークとも訳されますが，“reccurent” は周期的という意味であり，「回帰的」も同様です．一方，情報科学において，「再帰的，recursive」はもう少し広い概念で，一つの関数が自分自身を呼び出すということであり，「ハノイの塔」や「ユークリッドの互助法」のアルゴリズムなどに用いられます．ただし，「回帰」は線形回帰のように regression の訳にもあてられることが問題をややこしくしています．

回帰結合型ニューラルネットワークと再帰的ネットワーク

RNN は下図の左側のような計算グラフで表されるネットワークです．再帰型ニューラルネットワーク（recursive NN）というのも別にあり，こちらは右側のような計算グラフで表されるネットワークです．

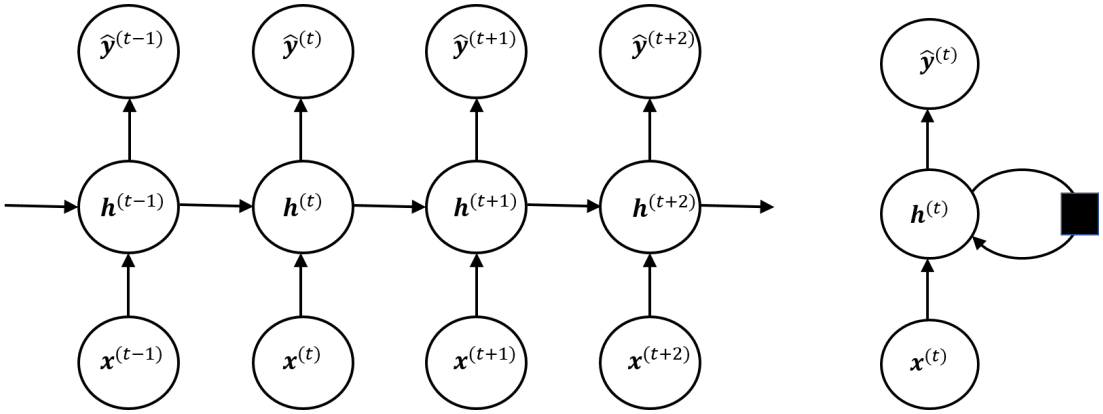


一つの〇はニューロンではなく層を表す．

下が入力の時系列データで上が出力（RNN の場合は出力も時系列データ）

1.1　回帰結合型ニューラルネットワーク

入力時系列データ $\{x^{(t)}\}$ から何らかの予測時系列データ $\{\hat{y}^{(t)}\}$ を出力する RNN は下図の左のようなグラフで表されます．



ここで，$\{h^{(t)}\}$ は隠れ層です．RNN の特徴は時系列に沿って隠れ層から隠れ層に情報が渡されることです．左の図の代わりに右図のように表すこともあります．黒い四角は時刻を一つ前にすることを示しています．

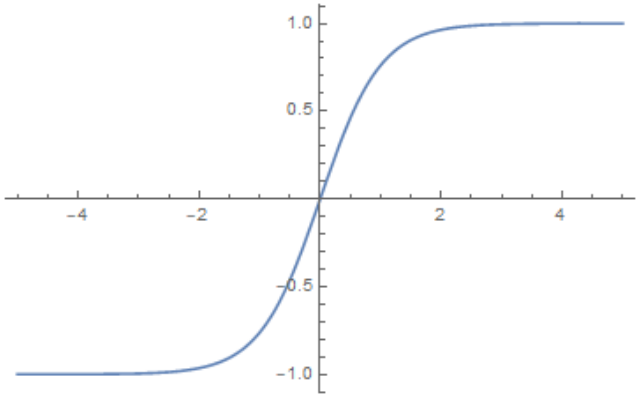
順伝播

RNN の順伝播は各時刻の入出力が実数ベクトルである最もシンプルな構成では，

$$\mathbf{a}^{(t)} = \mathbf{x}^{(t)}U + \mathbf{h}^{(t-1)}W + \mathbf{b} \quad (\text{入力を } 2 \text{ つもつアファイン変換})$$
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)}) \quad (\text{活性化関数})$$
$$\hat{\mathbf{y}}^{(t)} = \mathbf{h}^{(t)}V + \mathbf{c} \quad (\text{アファイン変換})$$

などとなります．ここで，重みパラメータ U，V，W は時間に依らず共通のものをとります．

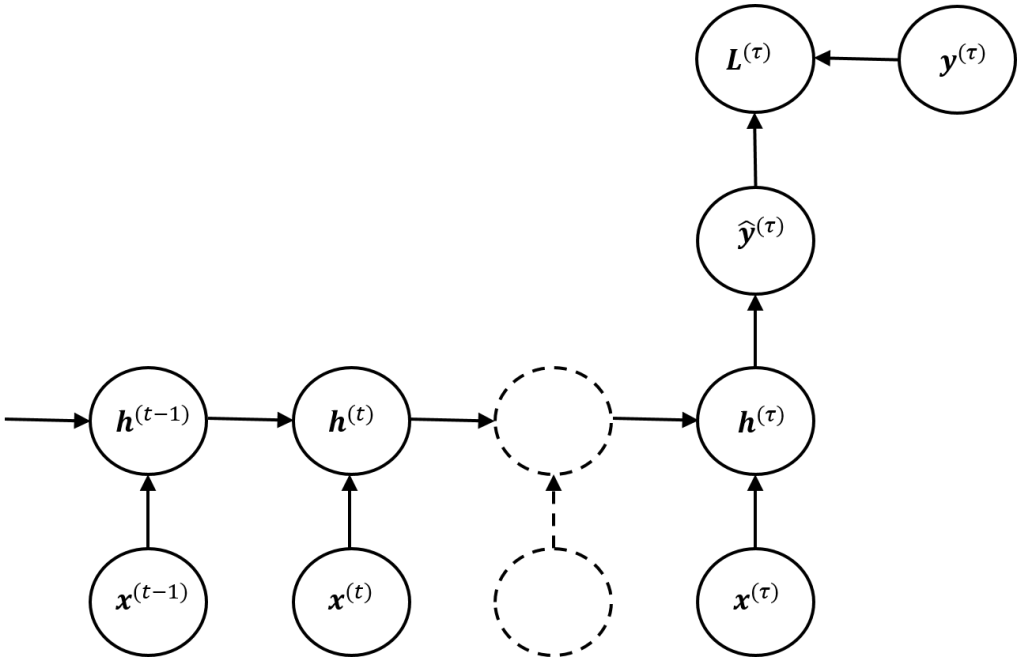
$\tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$ はハイパボリックタンジェント（双曲線正接）で，



というグラフを持ちます．

出力が一つのとき

出力が時系列データではなく，最後の時刻 τ でのみ値をとる場合のネットワークは



と表されます．ここで，$y^{(\tau)}$ は最終時刻 τ における教師ラベル，$L^{(\tau)}$ は損失関数です．

1.2　回帰結合型ネットワークにおける勾配計算（BPTT）

（この節の計算は複雑なので始めは流し読みで構いませんが，要は誤差逆伝播をしているだけです．）

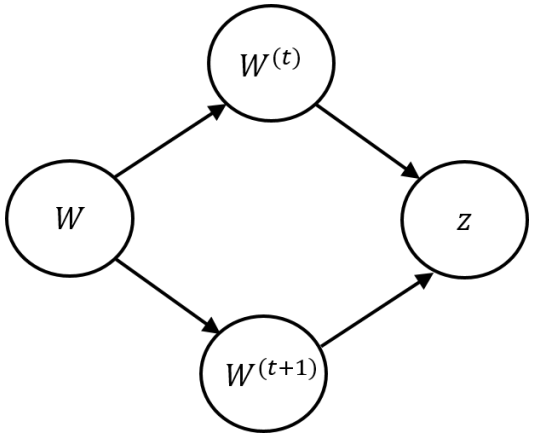
シンプルな RNN の順伝播は

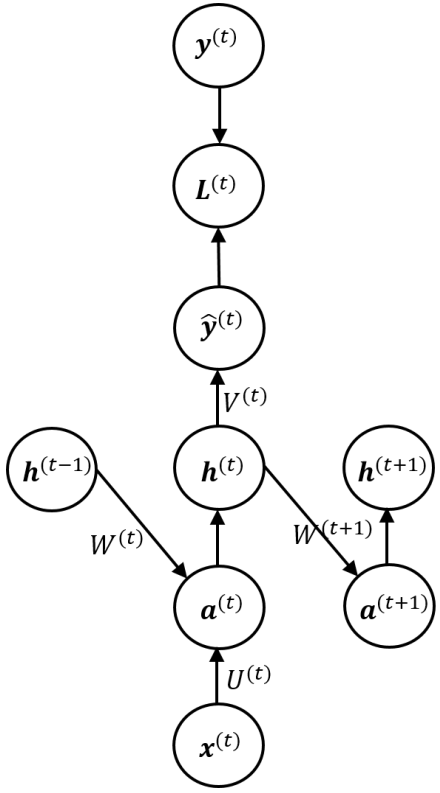
$$
\begin{aligned}
\mathbf{a}^{(t)} &= \mathbf{x}^{(t)}U + \mathbf{h}^{(t-1)}W + \mathbf{b} \\
\mathbf{h}^{(t)} &= \tanh(\mathbf{a}^{(t)}) \\
\hat{\mathbf{y}}^{(t)} &= \mathbf{h}^{(t)}V + \mathbf{c}
\end{aligned}
$$

でした．各時刻における損失 $L^{(t)}(\hat{\mathbf{y}}^{(t)}, \mathbf{y}^{(t)})$ の和 $L = \sum_t L^{(t)}$ をモデル全体の損失とするとき，L の重みパラメータに関する微分を計算しましょう．

逆伝播の合流（= 順伝播の分岐）

順伝播では重みパラメータは時刻に依らず共通のものを利用しますが，逆伝播においてはそれぞれの時刻における微分の和がパラメータに関する実際の微分になります．このことは，例えば，下図の左という依存関係があったときに，





$$\frac{\partial z}{\partial W} = \frac{\partial z}{\partial W^{(t)}} + \frac{\partial z}{\partial W^{(t+1)}}$$

となることから分かります．そこで，ネットワークにおける各時刻のパラメータを $U^{(t)}$，$V^{(t)}$，$W^{(t)}$，$\mathbf{b}^{(t)}$，$\mathbf{c}^{(t)}$ などとインデックス t をつけて区別します．このとき，順伝播は右図のような計算グラフで表されます．

$\tanh(x)$ の微分

$h = \tanh x$ の微分は商の微分公式を用いて計算できます．

$$
\begin{aligned}
(\tanh x)' &= \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)' = \frac{(e^x - e^{-x})'(e^x + e^{-x}) - (e^x - e^{-x})(e^x + e^{-x})'}{(e^x + e^{-x})^2} \\
&= \frac{(e^x + e^{-x})^2 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2} = 1 - (\tanh x)^2 = 1 - h^2
\end{aligned}
$$

逆伝播の計算 1：アファイン層の逆伝播

計算グラフを用いて出力から戻っていきます．

$L = \frac{1}{2}\left(\hat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)}\right)^2$ より，

$$\boxed{\frac{\partial L}{\partial \hat{\mathbf{y}}^{(t)}} = \hat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)}}$$

$\hat{\mathbf{y}}^{(t)} = \mathbf{h}^{(t)} V^{(t)} + \mathbf{c}^{(t)}$ より，

$$
\boxed{
\begin{aligned}
\frac{\partial L}{\partial V^{(t)}} &= \left(\mathbf{h}^{(t)}\right)^T \frac{\partial L}{\partial \hat{\mathbf{y}}^{(t)}} \\
\frac{\partial L}{\partial \mathbf{c}^{(t)}} &= \frac{\partial L}{\partial \hat{\mathbf{y}}^{(t)}} \\
\left(\frac{\partial L}{\partial \mathbf{h}^{(t)}}\right)_{\hat{\mathbf{y}}^{(t)}} &= \frac{\partial L}{\partial \hat{\mathbf{y}}^{(t)}} \left(V^{(t)}\right)^T
\end{aligned}
}
$$

ただし，$\left(\frac{\partial L}{\partial \mathbf{h}^{(t)}}\right)_{\hat{\mathbf{y}}^{(t)}}$ という記号は $\mathbf{h}^{(t)} \to \hat{\mathbf{y}}^{(t)} \to L$ という経路に関する微分を表すものとします．

逆伝播の計算 2：RNN 層の逆伝播[*1]

$\mathbf{h}^{(t)}$ で逆伝播が合流するので

$$\frac{\partial L}{\partial \mathbf{h}^{(t)}} = \left(\frac{\partial L}{\partial \mathbf{h}^{(t)}}\right)_{\hat{\mathbf{y}}^{(t)}} + \left(\frac{\partial L}{\partial \mathbf{h}^{(t)}}\right)_{\mathbf{a}^{(t+1)}}$$

第 2 項は，$\mathbf{h}^{(t)} \to \mathbf{a}^{(t+1)} \to L$ という経路に関する微分

$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$ より，

$$\boxed{\frac{\partial L}{\partial \mathbf{a}^{(t)}} = \left(1 - \mathbf{h}^{(t)} * \mathbf{h}^{(t)}\right) * \frac{\partial L}{\partial \mathbf{h}^{(t)}}}$$

$\mathbf{a}^{(t)} = \mathbf{x}^{(t)} U^{(t)} + \mathbf{h}^{(t-1)} W^{(t)} + \mathbf{b}^{(t)}$ より，

$$\boxed{\begin{aligned} \frac{\partial L}{\partial U^{(t)}} &= \left(\mathbf{x}^{(t)}\right)^T \frac{\partial L}{\partial \mathbf{a}^{(t)}} \\ \frac{\partial L}{\partial W^{(t)}} &= \left(\mathbf{h}^{(t-1)}\right)^T \frac{\partial L}{\partial \mathbf{a}^{(t)}} \\ \frac{\partial L}{\partial \mathbf{b}^{(t)}} &= \frac{\partial L}{\partial \mathbf{a}^{(t)}} \end{aligned}}$$

および

[*1] 実装時には最初の合流だけ TimeRNN という名前で別のクラスにしています．TimeRNN では重みの時間方向へのコピーなども行います．

逆伝播の計算 3

$$\left(\frac{\partial L}{\partial \mathbf{x}^{(t)}}\right) = \frac{\partial L}{\partial \mathbf{a}^{(t)}}\left(U^{(t)}\right)^T$$

$$\left(\frac{\partial L}{\partial \mathbf{h}^{(t-1)}}\right)_{\mathbf{a}^{(t)}} = \frac{\partial L}{\partial \mathbf{a}^{(t)}}\left(W^{(t)}\right)^T$$

以上で逆伝播の計算ができました．

以下のことにも注意しましょう．

- これら式を用いて t について帰納的に計算するためには，最後の t から計算する必要があります．
- 前述のように各パラメータは t には依らないので，

  $$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L}{\partial V^{(t)}}$$

  などと t について和をとらなければなりません．
- これら式は一つのデータに対するもので，ミニバッチのときは適切に和をとる必要があります．

RNN についての一般的な注意

RNN では時刻が異なっても重みパラメータを共有を共有しています．この構造は畳み込み演算とも似ています．重みパラメータを共有するというということは，各変数同士の確率的な依存関係が時刻 t に依らないことを仮定していることになります．RNN が有効に機能するためにはこの仮定が満たされていなくてははならないので，データに欠損があったり，時刻の刻み幅が一定でないときにはその有効性は期待できません．

1.3　RNN に関する演習

使用するノートブック:

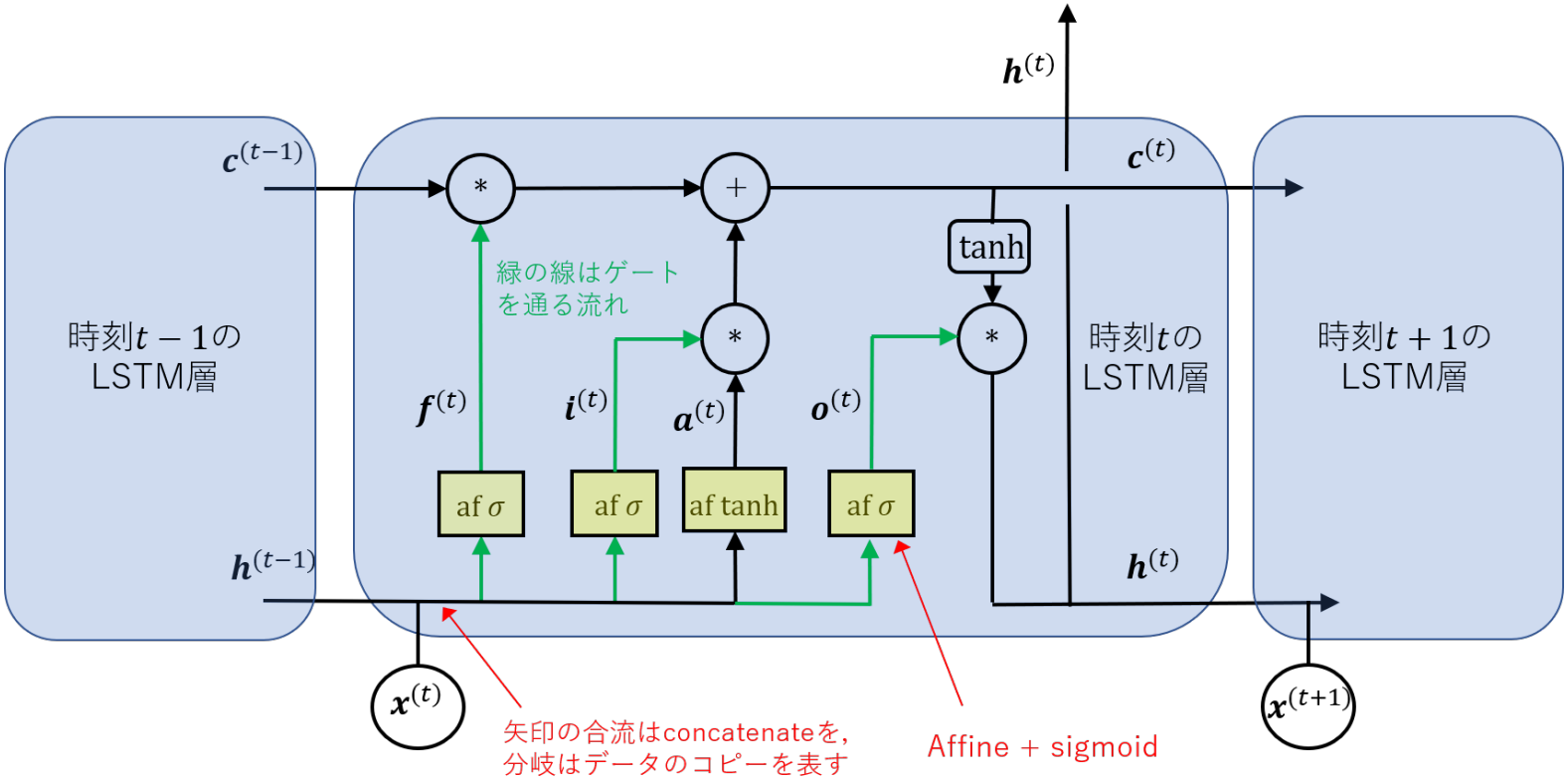
- 4-1-1_RNN 層.ipynb
- 4-1-2_TimeRNN 層.ipynb
- 4-1-3_RNN 全体.ipynb

7-1 では RNN 層をクラスとして定義します．7-2 では RNN 層を時間方向に展開したものを TimeRNN という名称の層として定義します．7-3 では RNN を用いた簡単な学習モデルを作成します．いずれも特に誤差逆伝播は複雑なので，始めは順伝播中心に理解を心がけましょう．

学習は時間方向に一つずらした時系列による教師データを用いて行います．にもかかわらず，2 以上ずらした時系列の予測にも使えます．どうように行うのでしょうか．

1.4　LSTM—ゲート付き RNN

LSTM (Long Short Term Memory) **ネットワーク** [Hochreiter and Schmidhuber, 1997] **は入力，出力，スキップ接続などにおいて「ゲート」を用いて** RNN **よりも長期に情報が伝わるようにしたモデルで以下のようなグラフで表せます．**



$\mathbf{i}^{(t)}$ **を入力（**input**）ゲート，**$\mathbf{f}^{(t)}$ **を忘却（**forget**）ゲート，**$\mathbf{o}^{(t)}$ **を出力（**output**）ゲートといいます．**

LSTM の順伝播

各ゲートの順伝播は

$$\mathbf{i}^{(t)} = \sigma\left(\mathbf{x}^{(t)}U^{(i)} + \mathbf{h}^{(t-1)}W^{(i)} + \mathbf{b}^{(i)}\right)$$
$$\mathbf{f}^{(t)} = \sigma\left(\mathbf{x}^{(t)}U^{(f)} + \mathbf{h}^{(t-1)}W^{(f)} + \mathbf{b}^{(f)}\right)$$
$$\mathbf{o}^{(t)} = \sigma\left(\mathbf{x}^{(t)}U^{(o)} + \mathbf{h}^{(t-1)}W^{(o)} + \mathbf{b}^{(o)}\right)$$

です．ここで，$U^{(i)}$，$U^{(f)}$，$U^{(o)}$ などはそれぞれのアファイン変換に応じた重みパラメータであり，時刻 t には依らずに共通のものとします．これらを用いて，LSTM の隠れ層では，

$$\mathbf{a}^{(t)} = \mathbf{x}^{(t)}U + \mathbf{h}^{(t-1)}W + \mathbf{b}$$
$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} * \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} * \tanh\left(\mathbf{a}^{(t)}\right)$$
$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} * \tanh\left(\mathbf{c}^{(t)}\right)$$

という計算を行います．LSTM では中間層において $\mathbf{c}^{(t-1)}$ の値がアファイン変換されることなく，忘却ゲート $\mathbf{f}^{(t)}$ を成分ごとにかけるだけで $\mathbf{c}^{(t)}$ に伝えられます．その結果，モデルが長い時刻に渡って情報を伝えることができます．このように LSTM では $\mathbf{c}^{(t)}$ がメモリーの役割を果たしています．

2　自然言語処理

　自然言語処理は，人間が日常的に使っている自然言語をコンピュータに処理させる一連の技術であり，人工知能と言語学の一分野です．「自然言語」と対比される概念はプログラミング言語です．プログラミング言語には曖昧さがありませんが，自然言語にはあります．

例 1　　「部長，書類はこれでよろしいでしょうか」「結構です」
　　「お客様，もう一杯おかわりはいかがでしょうか」「結構です」

例 2　　「もうおなか痛くないの」「大丈夫です」
　　「レシートはいかがですか」「大丈夫です」

例 3　　「背の高い目の黒い男の子」
　　「アメリカ人の大きい青い目の女の子」　（修飾の関係が違います．）

自然言語処理の流れ

深層学習における自然言語処理の流れは基本的に以下のようになります．

1. コーパスと 機械可読辞書を用意する．コーパスおよび機械可読辞書とはそれぞれコンピュータで読めるようにした文章群と辞書のことです．（辞書は英語のように単語が区切れている言語では必要ありません．）

2. 形態素解析を行って，単語レベルに分割した分かち書きに直す．形態素解析とは，辞書を用いて文章の中の単語を分離し，その品詞を特定する技術のことです．（これも英語では必要ありません．）

3. 単語の文章における前後関係の情報を利用して単語を分散表現に直す．単語の分散表現とは単語と対応付けられた実ベクトル のことです．

4. RNN などの時系列処理できる学習モデルを用いて，翻訳，自動対話，意味理解などのタスクを行う．

伝統的な機械学習の処理方法では，3，４の代わりに，「構文解析」，「意味解析」と進みます．

2.1 コーパス

コーパスとはコンピュータで読めるようにした文章群のことですが，言語や目的によって様々なものがあります．ここで挙げるのはほんの一例にすぎません．

- 現代日本語書き言葉均衡コーパス：国立国語研究所の提供する書籍全般，雑誌全般，新聞，白書，ブログ，ネット掲示板，教科書，法律などのジャンルにまたがったコーパスで形態素解析済みのコーパス
- 青空文庫: 著作権の消滅した作品，また「自由に読んでもらってかまわない」とされたものをテキストと XHTML(一部 HTML) 形式に電子化した上で揃えている．Github からダウンロード可能
- livedoor ニュースコーパス: トピックニュース，スポーツなどの九分野のニュース記事を含むコーパスで手軽に使える
- 日本語対訳データ（ポータルサイト）http://phontron.com/japanese-translation-data.php?lang=ja
- Twitter 日本語評判分析データセット: ツイートの評判情報をクラウドソーシングによって分析した結果も提供されています．

2.2　形態素解析

形態要素解析とは文章を単語レベルに分解して品詞を特定する技術です．実例を見てみましょう．

以下は「先生はとても忙しそうでした」を Janome という Python 用の日本語形態要素解析ライブラリで解析した結果です．

先生 名詞, 一般,*,*,*,*, 先生, センセイ, センセイ
は 助詞, 係助詞,*,*,*,*, は, ハ, ワ
とても 副詞, 助詞類接続,*,*,*,*, とても, トテモ, トテモ
忙し 形容詞, 自立,*,*, 形容詞・イ段, ガル接続, 忙しい, イソガシ, イソガシ
そう 名詞, 接尾, 助動詞語幹,*,*,*, そう, ソウ, ソー
でし 助動詞,*,*,*, 特殊・デス, 連用形, です, デシ, デシ
た 助動詞,*,*,*, 特殊・タ, 基本形, た, タ, タ

Janome は Mecab というライブラリの辞書を Python で手軽に使えるようにしたものです．他にも色々とありますので気になる人は「形態素解析 ツール 比較」で検索してみてください．また，品詞の分類に関しては「IPA 品詞体系」で検索してください．

2.3　単語の分散表現

単語の分散表現（distributed representation）**あるいは単語埋め込み**（word embedding）**とは単語と対応付けられた実ベクトルのことです．例えば**

$$\text{「私」}\quad \text{という単語は}\quad [0.12, 0.42, 0.87]$$
$$\text{「あなた」}\quad \text{という単語は}\quad [0.31, 0.35, 0.72]$$

といった具合です．

単語をベクトルに直す簡単な方法はワンホット表現を使うことですが，例えば「広辞苑第七版」に掲載されている単語は 25 **万語ですので** 25 **万次元必要になります．それに活用も加わります．また，「私」，「わたし」，「あたし」，「僕」，「ぼく」「俺」，「オレ」などは近い意味を持っていて置き換えも可能ですが，このような単語同士の関係がまったく分かりません．**

分散表現では，実数ベクトルを利用することで次元にを低くするとともに，単語の意味の近さをベクトル同士の距離として表せるようにします．

word2vec

単語を分散表現に直すにはまず単語に番号を付けてワンホットベクトル $\mathbf{x}$ に直し，それに変換行列（埋め込み行列とよばれる）W をかけます．

$$\underset{\mathbf{x}}{\begin{bmatrix}0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\end{bmatrix}} \underset{W}{\begin{bmatrix} 0.31 & 0.33 & 0.52 \\ 0.12 & 0.42 & 0.87 \\ 0.31 & 0.35 & 0.72 \\ \vdots & & \vdots \\ 0.52 & 0.63 & 0.21 \end{bmatrix}} = \underset{\text{分散表現}}{\begin{bmatrix} 0.31 & 0.35 & 0.72 \end{bmatrix}}$$

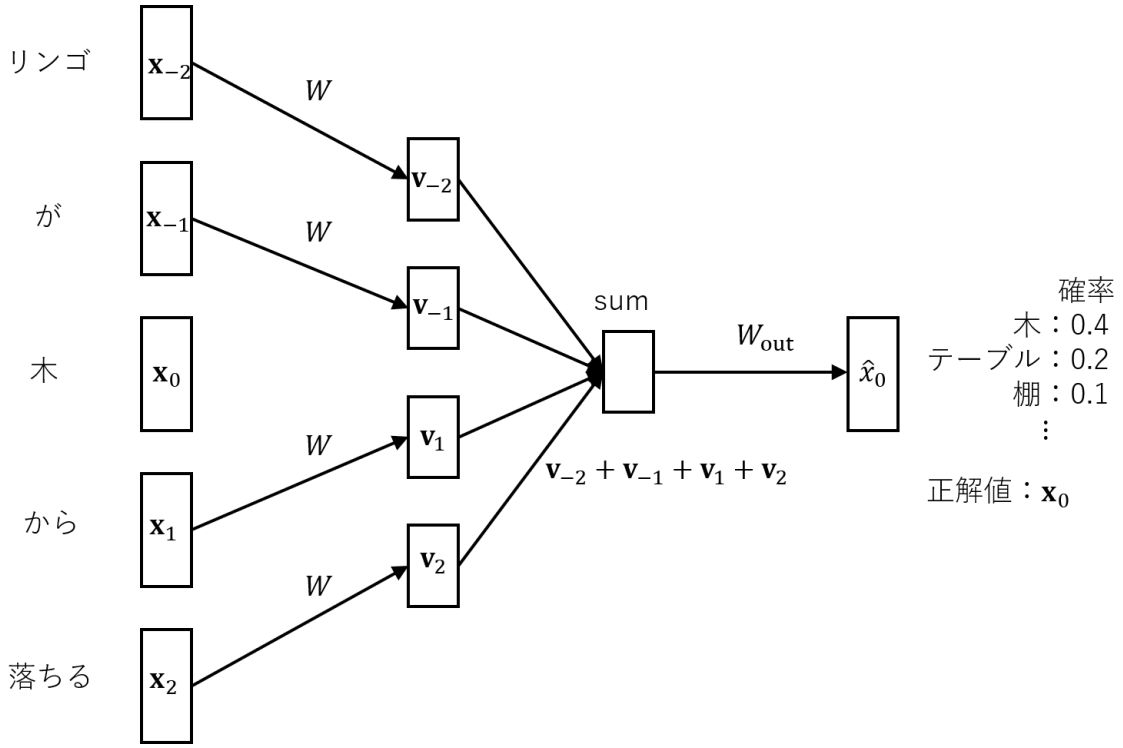
埋め込みだけなら，埋め込み行列を適当に決めればできますが，それでは単語間の関係性は分かりません．

word2vec [Mikolov, T. et al., 2013] はこの埋め込み行列 W を効率よく学習できるツールです．

埋め込み行列の学習は基本的に前後の単語との関係を用いて行いますが，word2vec では **Continuous Bag-of-Words (CBOW)** および **skip-gram** という 2 つのニューラルネットワークによる方法が用意されています．

CBOW

Continuous Bag-of-Words はその単語の前後の単語から埋め込みを経由してその単語を予測するというタスクを学習することで W を学習する方法です．下図の W と W_{out} という 2 つの行列を学習しますが，分散表現を作るときは W のみを用います．

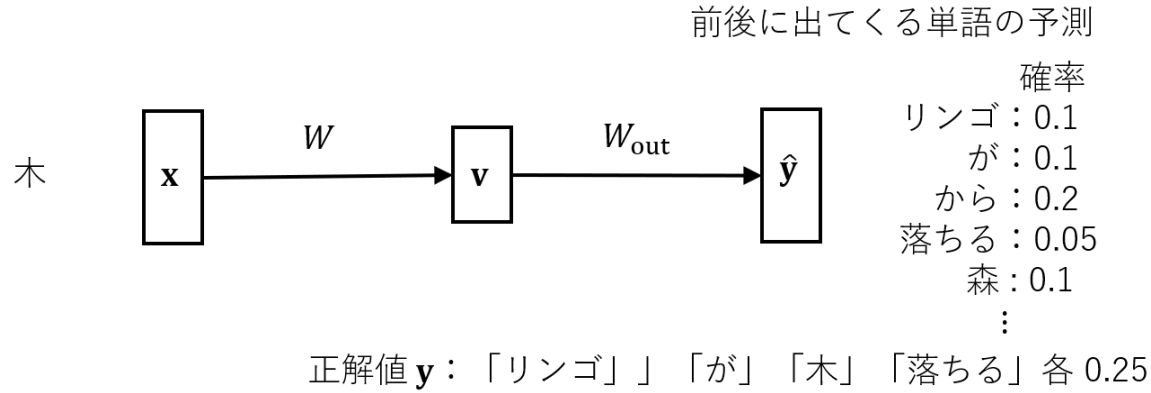


W は各単語で共通のものを使う．

実際は損失関数の設計に工夫が凝らされていますが，省略します．

skip-gram

skip-gram は CBOW とは逆に単語から埋め込みを経由してその前後の単語を予測するというタスクを学習することで W を学習する方法です．



こちらも実際は損失関数の設計に工夫が凝らされていますが，省略します．

コサイン類似度

word2vec で作られる分散表現は各成分が $[-1, 1]$ になるように正規化されていて，2 つの表現 $\mathbf{v}_1$，$\mathbf{v}_2$ の類似度はコサイン類似度

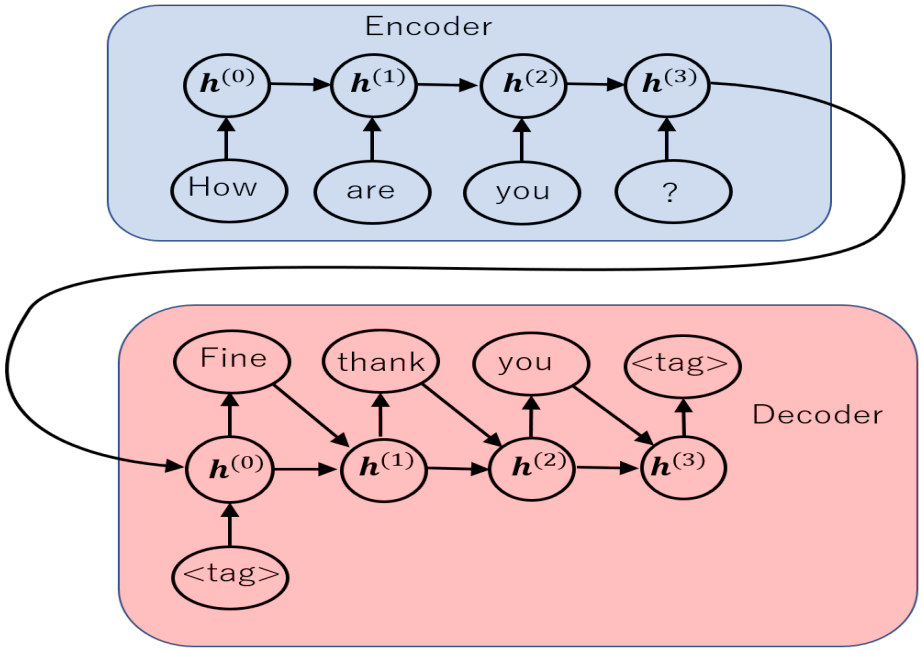
$$\cos\theta = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{||\mathbf{v}_1||\ ||\mathbf{v}_2||} \qquad (\theta \text{ は } \mathbf{v}_1 \text{ と } \mathbf{v}_2 \text{ のなす角度})$$

で測れます．

2.4　系列変換

機械翻訳や自動応答，自動要約といったタスクでは，系列から長さの異なる別の系列への変換が必要になります．このような系列変換モデルにはの符号化器（Encoder）と復号化器（Decoder）を持つネットワークを用いる必要があります．

例えば seq2seq [Sutskever et al., 2014] による自動応答の予測は以下のようなモデルで行います．

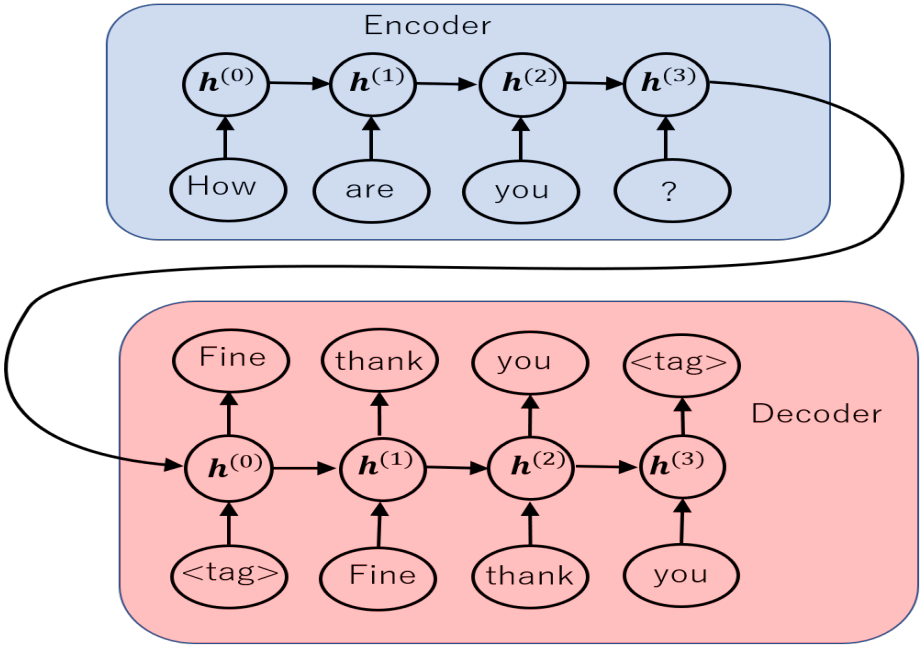


seq2seq 推測時

なお，実際には入力（単語の番号で与える）と RNN 層の間に埋め込み層が入っており，RNN 層と出力層の間に，全結合層が入っています．

seq2seq（学習時）

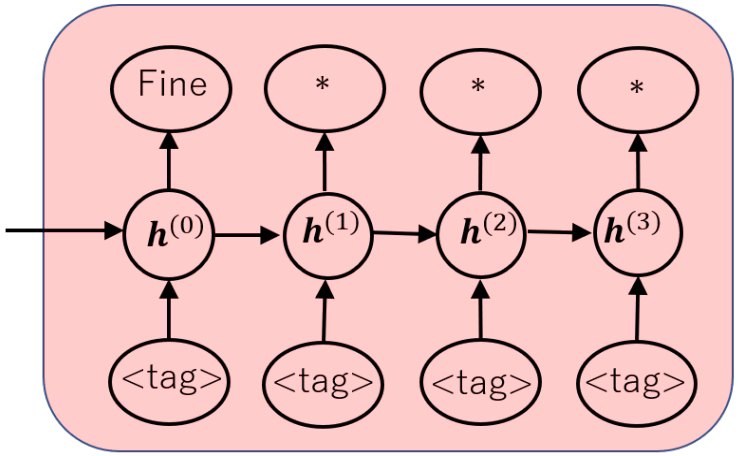
一方学習時には正解データによる正解ラベルによる強制のある以下のようなモデルを用います．



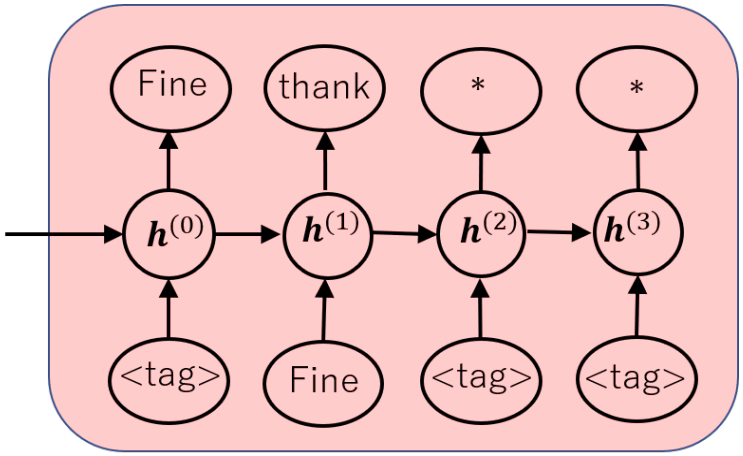
seq2seq 学習時

しかし，予測のためにモデルを別に用意するのは大変なので，実用上は次ページのように復号器への入力を一つずつ右にずらしながら時間方向に繰り返し処理することで，学習時と同じモデルを使って予測することができます．

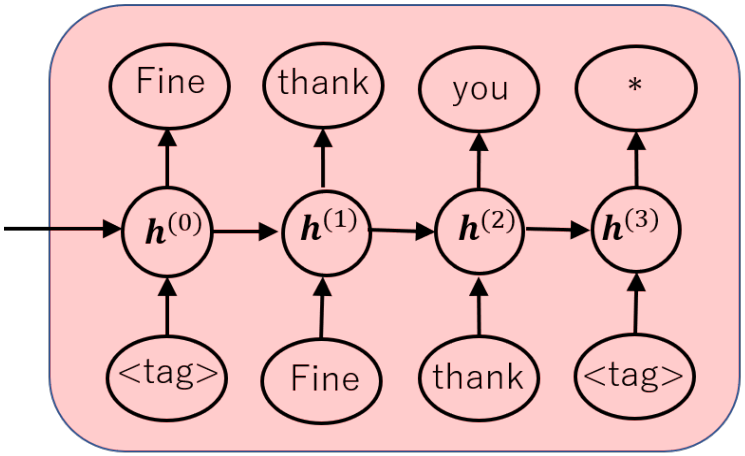
seq2seq（予測時・実際）



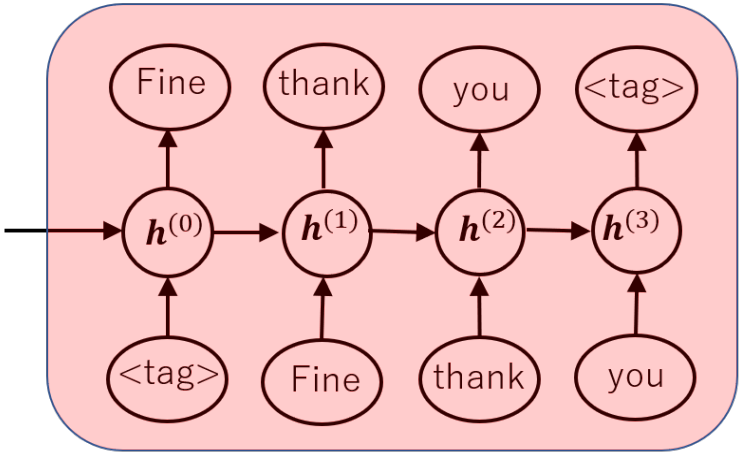
1回目



2回目



3回目



4回目

2.5　系列変換についての演習

使用するノートブック:　`4-4_ChatBot_seq2seq.ipynb`

このノートブックでは Keras **で** ses2se2 **を実装し，自動応答システムを作ってみます．使用するデータは** ChatterBot Language Training Corpus (https://github.com/gunthercox/chatterbot-corpus) **というものです．小さなデータなのでそれ程良い結果が得られるわけではありませんが，実験してみましょう．**

2.6　アテンション

seq2seq のような系列変換モデルでは，符号器の情報を復号器の初期値としてのみ渡すため，系列が長くなると，系列の後ろの方まで情報がうまく伝わらなくなります．

アテンション [Bahdanau, D. et al., 2014] はこの課題に対処する仕組みであり，最近の自然言語処理の分野におけるスタンダードな技術となってきています．

さらに，2016 年に “Attention is All You Need” と題した論文 [Vaswani, A. et al., 2016] において時系列データを扱う際に，回帰結合層を廃してアテンションのみを用いるトランスフォーマーという手法が提案され，それを双方向化した BERT（Bidirectional Encoders’ Representations from Transformer）[Devlin, J. et al., 2018] と併せて注目されています．

アテンションの仕組み

アテンションを式で表すと，

$$\text{Attention: } C = \text{softmax}(QK^T,\ \text{axis} = 0) \cdot V$$

となります．ただし，$K = V$ はエンコーダの出力列，Q はデコーダの出力列であり，系列は一列だけの（つまりミニバッチは考えていない）場合です．

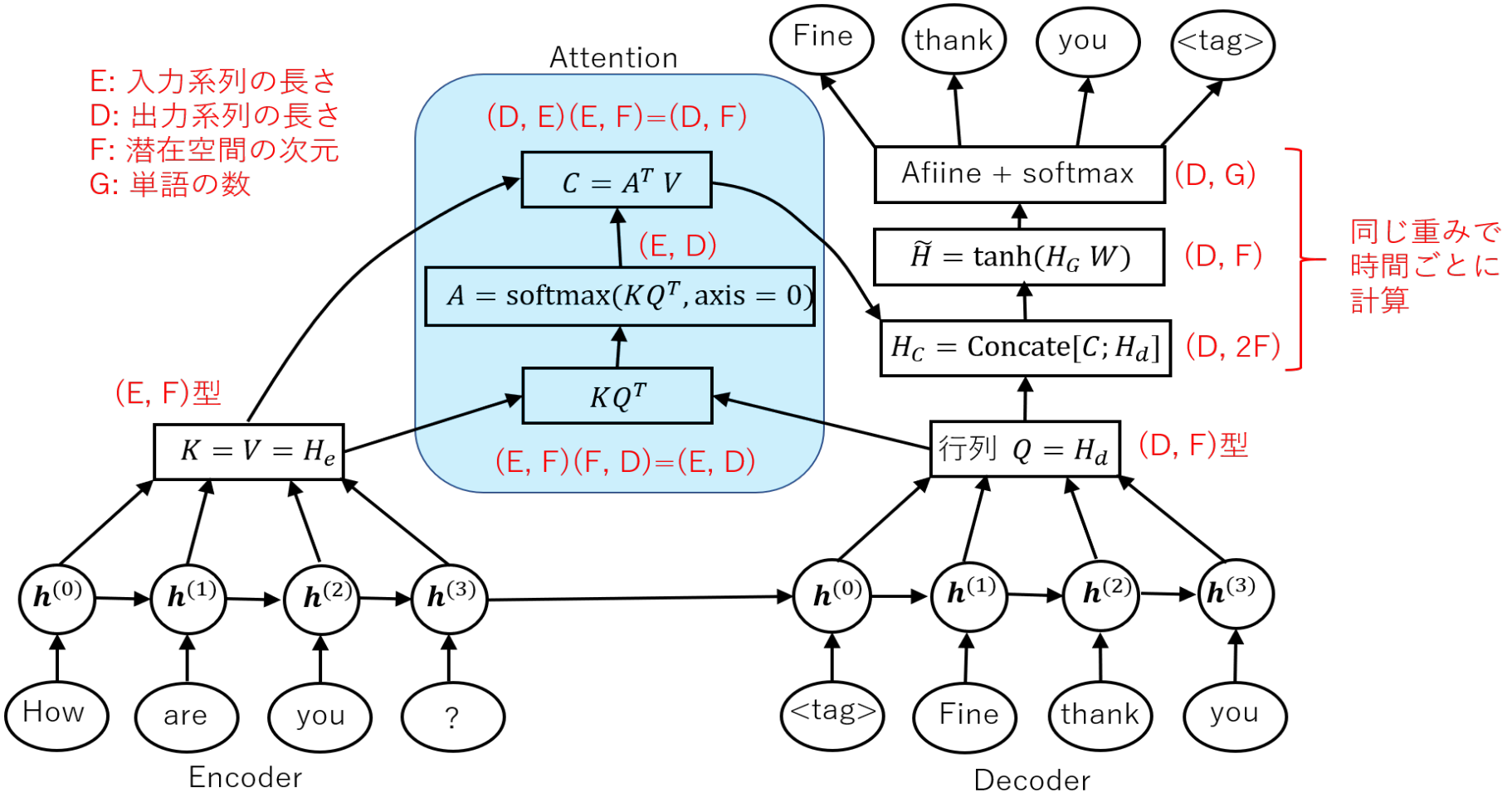
- Q: Query, K: Key, V: Value という解釈があります．デコーダから「このような値 Q だが，どこに注目するべきか？」と聞かれたら，エンコーダが「質問と鍵 K の類似度を測り」「その割合で値 V を渡したもの」がアテンションです．

デコーダは受け取ったアテンションと自分の出力列 $H_d = Q$ を結合して，重み W を用いて

$$\tilde{H} = \tanh([C; H_d]W)$$

を新たなデコーダの出力とします．ただし，出力列の時間 t に対して同じ W を用いて t ごとに計算します．これは RNN の出力層でアファイン層を使うときと同様です．同じ W を使わないとアテンションの意味がなくなるので注意してください．

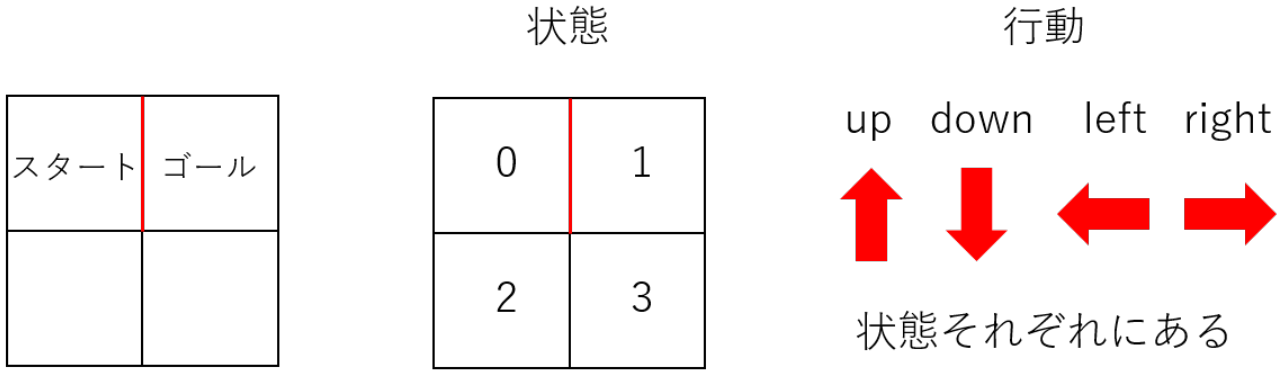
アテンション付き seq2seq のネットワーク



系列が一つの場合の順伝播．ミニバッチのときはデータ数が加わって axis が一つずれるので注意

3　強化学習

下図の一番左側のような迷路を考えてみましょう．



エージェント（プレーヤー）は状態 0 からスタートして状態 1 のゴールを目指します．ただし，赤い線のところは通れないとします．

もちろん答え（最短路）は

$$0 \to 2 \to 3 \to 1$$

です．このような一連の流れをエピソードといいます．このような良いエピソードを得ることが強化学習の目標です．

行動 action

「行動」は 4 つある「状態」それぞれについて定まるので，以下の 16 通りあります．

(0,up)	(0,down)	(0,left)	(0,right)
(1,up)	(1,down)	(1,left)	(1,right)
(2,up)	(2,down)	(2,left)	(2,right)
(3,up)	(3,down)	(3,left)	(3,right)

遷移関数 transition function

「行動」によって状態が以下のように変わります．

	up	down	left	right
0	0	2	0	0
1	1	3	1	1
2	0	2	2	3
3	1	3	2	3

これを遷移関数といいます．壁にぶつかった場合はその状態にとどまることとしています．なお，より一般的な状況では遷移関数は状態に対する条件付確率分布で与えられます．

報酬 reward

「行動」の結果報酬がもらえるとします．今の問題ではゴールすることが目的なので状態 3 から上に進んで状態 1 になるときのみ $+1$ とします．

	up	down	left	right
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	1	0	0	0

あとで出てくる累積報酬と区別するためにこの報酬を「即時報酬」と呼ぶことも有ります．ただし，時刻 t でもらう報酬を r とするとき，報酬の時刻 0 での価値は $r\gamma^t$（$0 < \gamma < 1$）であるとします．これを割引現在価値といいます（未来の報酬の現在における価値）．これはなるべく早くゴールした方がより高い価値があるということを意味しています．ここでは $\gamma = 0.9$ としておきます．

方策 policy

状態 s にいるときの行動の選び方を方策（または行動方策）といいます．学習の仕方により，方策を確率分布で与えるときもありますし，状況から決定論的に選ぶことも有ります．

4　Q 学習

Q 学習（Q-learning）は行動ごとに，その行動を選択したあとで得られる報酬の期待値について学習する方法です．プレーヤーはなるべく報酬の期待値が高い行動をとり続けることによりゴールまでたどり着けるはず，という考えに基づいています．

Q-learing は [Watkins, C.J.C.H., 1989] で提案された古典的な学習法ですが，その深層学習版もあり，そちらの代表的なものが DQN, Deep Q-networks [Mnih, V. et al. 2013，2015] です．

強化学習のもう一つの有名な手法である方策勾配法では，行動を選ぶ方策は常に確率分布で与えられますが，Q-学習ではプレーヤーは常に報酬の期待値が最も高くなる行動とるので，方策を決定論的に選ぶことができます．

行動価値関数

時刻 0 である行動をし，その後ある方策の下で行動し続けたときの報酬（の割引現在価値の）期待値を行動価値関数といいます．例えばプレーヤーが時刻 1 から「上，下，左，右，上，下，左，右，$\cdots$ と状態によらず行動し続ける」という方策 π を持っているとき，時刻 0 で $(0,\ \mathrm{right})$ という行動を選択プレーヤーは，

時刻 0:	$(0,\ \mathrm{right})$	状態 0 のまま
時刻 1:	$(0,\ \mathrm{up})$	状態 0 のまま
時刻 2:	$(0,\ \mathrm{down})$	状態 2 に移る
時刻 3:	$(2,\ \mathrm{left})$	状態 2 のまま
時刻 4:	$(2,\ \mathrm{right})$	状態 3 に移る
時刻 5:	$(3,\ \mathrm{up})$	状態 1 に移り，報酬 1 を得て終了

となるので，このときの行動価値関数は時刻 5 で得た報酬の割引現在価値 $\gamma^5 = 0.9^5$（$\gamma = 0.9$ としていました）となります．これを

$$Q^{\pi}(0, \mathrm{right}) = 0.9^5$$

などと書きます．

最適行動価値関数

「各状態において，行動価値関数が最も高い行動をとり続ける」という方策による行動価値関数を最適行動価値関数といいます．この最適行動価値関数の定義は自分自身を参照してしまっているので，最適行動価値関数を計算するには何らかの方程式を解く必要があります．

状態 s の任意の行動 (s,a) に対して，$Q(s,a)$ を行動 (s,a) の最適行動価値とします．$R(s,a)$ を行動 (s,a) の即時報酬とし，s' をその行動の結果移る状態とします．$Q(s,a)$ は行動 (s,a) を選んだ上でその後最適行動をした場合の価値なので，状態 s' において最適行動価値関数 $Q(s',a')$ が最も高くなる行動を (s',a') とすると，

$$Q(s,a) = R(s,a) + \gamma Q(s',a')$$

が成り立つはずです．つまり，

$$Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a')$$

が成り立ちます．これが最適行動価値関数の満たす方程式であり，ベルマン方程式と呼ばれるものの特殊な場合に当たります．

注意：$Q(s,a)$ は任意の行動 (s,a) について定めます．つまり，状態 s からの最初の行動だけは任意で，次の行動から最適行動した場合に期待される累積報酬です．

最適行動価値関数（2×2 の迷路の場合）

$$Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a')$$

という方程式を解くのは難しいですが，ゴールに近いマスから解いて行くことにより，以下のように解けます．

	up	down	left	right
0	0.729	0.81	0.729	0.729
1	nan	nan	nan	nan
2	0.729	0.81	0.81	0.9
3	1	0.9	0.81	0.9

ベルマン方程式の解

例えば (3,up) という行動をすると即座に報酬 1 が与えられるので $Q(3, \mathrm{up}) = 1$ です．(2,right) という行動をすると報酬は 0 で状態 3 に移り，次の時刻で行動価値関数の最も高い (3,up) を選びますが，$Q(3, \mathrm{up}) = 1$ なので $Q(2, \mathrm{right}) = 0 + 0.9 \max_{a'} Q(3, a') = 0.9$ となる，といった具合です．このような解き方を動的計画法といいます．

　このように最適行動価値関数が分かれば，各状態において最適行動を繰り返すことにより，最短路をたどることができます．

Q 学習

Q 学習（テーブル Q 学習）ではベルマン方程式

$$Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a'), \quad (s' \text{ は } (s,a) \text{ により移る状態}) \quad \cdots ①$$

を経験を繰り返すことによって近似的に解きます．まず，定数 α（$0 < \alpha < 1$）に対して ① の α 倍と

$$(1-\alpha)Q(s,a) = (1-\alpha)Q(s,a)$$

の辺々を加えて

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left(R(s,a) + \gamma \max_{a'} Q(s',a')\right)$$

を得ます．右辺を α で整理すると

$$Q(s,a) = Q(s,a) + \alpha\left(R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

となります．適当にとった $Q(s,a)$ から始めてこの式の右辺で左辺を置き換えて行くというのが Q 学習の基本的な考え方です．つまり，(s,a) を色々変えながら

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

により，$Q(s,a)$ を更新していきます．このようにランダムサンプリングを用いる計算法をモンテカルロ法といいいます．

Q 学習のアルゴリズム

しかし，(s, a) を完全にランダムに決めるのでは学習効率が悪いので以下のようにします．

入力：任意の (s, a) に対して $Q(s, a) = 0$（あるいはランダムに決める）

time： 自然数（1 エピソードで進む回数の上限）

ε：$0 < \varepsilon \leq 1$ の実数（どの程度ランダムに行動を選択するかという指標）

以下のエピソードを一定回数または Q が改善しなくなるまで実行

1. 初期値状態 s を決める．ただし，$s \neq \text{goal}$
   $t = 0$ とする．
2. s を固定して，確率 ε で以下の (1) を，残りで (2) を実行
   (1) 行動 (s, a) をランダムに決める（not greedy）
   (2) 行動 (s, a) を $Q(s, a)$ が最大になるもので定める（greedy）
3. s' を行動 (s, a) の結果移る状態とする
   $Q(s, a) \leftarrow Q(s, a) + \alpha \left(R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)$
4. $s \leftarrow s'$ とする．
   $s = \text{goal}$ または $t = \text{time}$ ならば次のエピソードに進む．
   そうでなかったら $t \leftarrow t + 1$ として 2 に戻る．

Q 学習のアルゴリズム（つづき）

2 で (2) を選択しながら進むことでそれまでに学習した成果に沿って「貪欲 (greedy) に」学習するようになります．$\varepsilon = 0$ とすると，常に貪欲に行動するので同じ行動ばかり選択してしまい学習が進まなくなります．逆に $\varepsilon = 1$ とすると，完全に乱数で状態を決めているのと変わらなくなり，あまり重要でない範囲まで等しく探索を行ってしまいます．

このように ε が小さいとより貪欲に探索するようになり，ε が大きいとより広く探索するようになります．このような学習法を**ε-greedy 法**といいます．

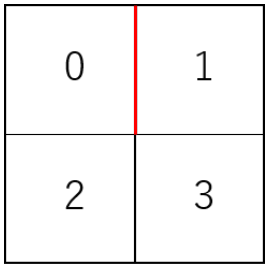
一つのエピソードの繰り返しを time で止めているのは，あまりにゴールから遠いところから始めると，とくに最初のころはなかなかゴールまでたどり着かなくて効率が悪くなるためです．

Q 学習の例

Q 学習における Q 関数の振る舞いを具体例で見てみましょう．α は大きめにとって $\alpha = 0.7$ とします．

初期テーブル

	up	down	left	right
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0



1. $s = 2$，(2,right) と選ぶと $s' = 3$，$Q(s', a') = Q(3, \text{up}) = 0$ なのでテーブルは変わらない．
2. (3,up) と選ぶと $R(3, \text{up}) = 1$ で $s' = 1$，$Q(s', a') = Q(1, a') = 0$ なので

$$Q(3, \text{up}) = 0 + 0.7(R(3, \text{up}) + 0.9 \cdot 0 - 0) = 0.7$$

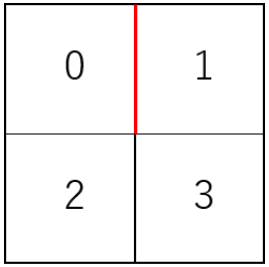
	up	down	left	right
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0.7	0	0	0

Q 学習の例（つづき）

3. ゴールしたので $s = 2$ と選び直す．(2,right) と選ぶと $Q(s', a') = Q(3, \text{up}) = 0.7$ なので

$$Q(2, \text{right}) = 0 + 0.7(0 + 0.9 \cdot 0.7 - 0) = 0.441$$

	up	down	left	right
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0.441
3	0.7	0	0	0



4. (3,down) と選ぶと $Q(s', a') = Q(3, \text{up}) = 0.7$ なので

$$Q(3, \text{down}) = 0 + 0.7(0 + 0.9 \cdot 0.7 - 0) = 0.441$$

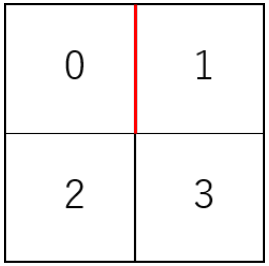
	up	down	left	right
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0.441
3	0.7	0.441	0	0

Q 学習の例（つづき）

5. (3,up) と選ぶと $R(3, \text{up}) = 1$ で $Q(s', a') = Q(1, a') = 0$ なので

$$Q(3, \text{up}) = 0.7 + 0.7(1 + 0.9 \cdot 0 - 0.7) = 0.91$$

	up	down	left	right
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0.441
3	0.91	0.441	0	0



6. ゴールしたので $s = 0$ と選び直す．(0,down) と選ぶと $Q(s', a') = Q(2, \text{right}) = 0.441$ なので

$$Q(0, \text{down}) = 0 + 0.7(0 + 0.9 \cdot 0.441 - 0) = 0.27783$$

	up	down	left	right
0	0	0.27783	0	0
1	0	0	0	0
2	0	0	0	0.441
3	0.91	0.441	0	0

Q 学習の例（つづき）

このような作業を延々と続けていくと

	up	down	left	right
0	0.729	0.81	0.729	0.729
1	0	0	0	0
2	0.729	0.81	0.81	0.9
3	1	0.9	0.81	0.9

という最適行動価値関数が得られます．

この状態になると，最適行動価値関数は変わらなくなります．

Q 学習の課題

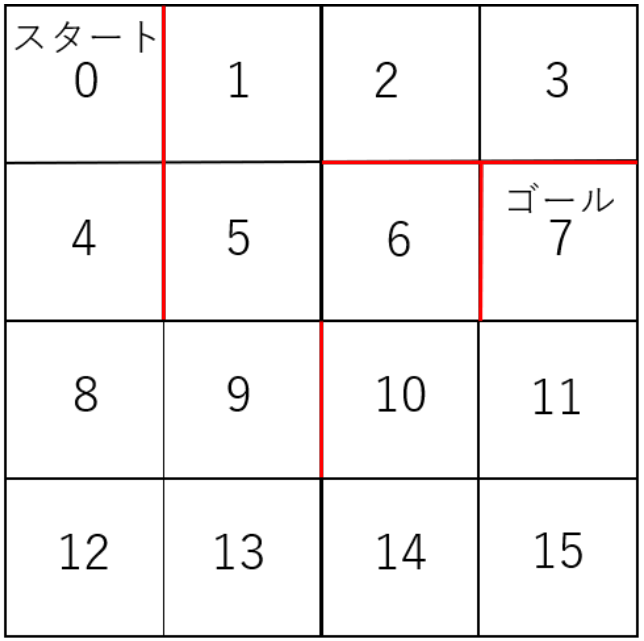
1. 探索範囲が広すぎる：基本的にすべての行動についての最適行動価値関数 Q 値を保持しなくてはなりません．しかし，例えば囲碁でいえば，あり得ない局面は学習しても意味がありません．
2. 状態が離散値でないと扱えない：一つ目の課題とも関わりますが，例えば 0 以上 1 未満の連続変数を $[0, 0.1)$，$[0.1, 0.2)$，$\cdots$，$[0.9, 1)$ と区分して離散変数にするなどということも考えられますが，例えば変数が 8 個あったら $10^8 = 100,000,000$ 個の状態になります．これを学習するのはほぼ不可能です．

次節で紹介する DQN では最適行動価値関数 Q をニューラルネットワークで表すことにより，Q そのものを学習するのではなく，ネットワークの重みパラメータを学習します．

4.1　Q 学習に関する演習

使用するノートブック:　8-1_テーブル Q 学習_迷路.ipynb

Q 学習を用いて迷路を解きます．上で紹介した 2×2 の例の他，



という迷路も解いてみましょう．

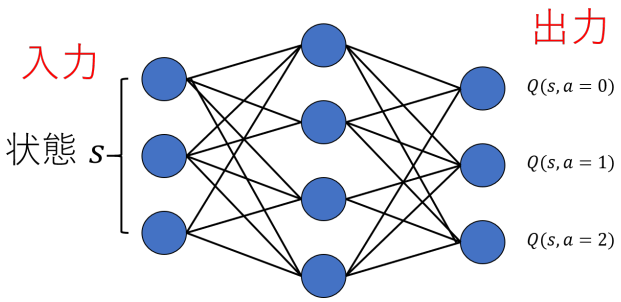
5　DQN

前節の最後で挙げた課題のように，行動価値関数（Q テーブル）は非常に多くの離散変数を引数とする関数になってしまいます．この課題を解決したのが本節で学ぶである DQN (Deep Q Networks）[Mnih et al., 2015] です．

DQN とは、行動価値関数 Q をニューラルネットワークで近似して強化学習を行う手法です．ニューラルネットワークの入力は状態で，出力はそのときの行動それぞれの価値関数になります．つまり，行動の数を A とするとき，状態 s を入れると $Q(s,a)$ の重みパラメータ θ によって近似値

$$Q(s,a=0,\theta), \quad \cdots, \quad Q(s,a=\mathrm{A}-1,\theta)$$

が定められます．ニューラルネットワークを用いることで高次元データに対応でき，状態が連続変数の場合や画像データの場合の学習が可能となります．



DQN の原理

Q 学習のときは，最適行動価値関数 $Q(s,a)$ がベルマン方程式

$$Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a'), \quad (s' \text{ は } (s,a) \text{ により移る状態})$$

を任意の $(s,a) = (\text{状態}, \text{行動})$ について満たすことを利用して，実際の行動結果からベルマン方程式を満たす $Q(s,a)$ に近づけていきました．

DQN の場合は $Q(s,a)$ の代わりにその近似 $Q(s,a,\theta)$ が

$$Q(s,a,\theta) = R(s,a) + \gamma \max_{a'} Q(s',a',\theta)$$

を満たすようにすることが目標になります．状態 s である行動 a を選択したとき，報酬 $R(s,a)$ が貰えたとすると，右辺の方が経験を経たより正しい値と考えられるので，右辺を教師データとし，左辺を予測値としたときの，2 乗誤差の $\frac{1}{2}$

$$E = \frac{1}{2}\left(Q(s,a,\theta) - \left(R(s,a) + \gamma \max_{a'} Q(s',a',\theta)\right)\right)^2$$

を一回の行動に対する損失関数とします．（青が予測値で赤が教師信号）．

DQN の原理（つづき）

誤差関数

$$E = \frac{1}{2}\left(Q(s,a,\theta) - \left(R(s,a) + \gamma \max_{a'} Q(s',a',\theta)\right)\right)^2$$

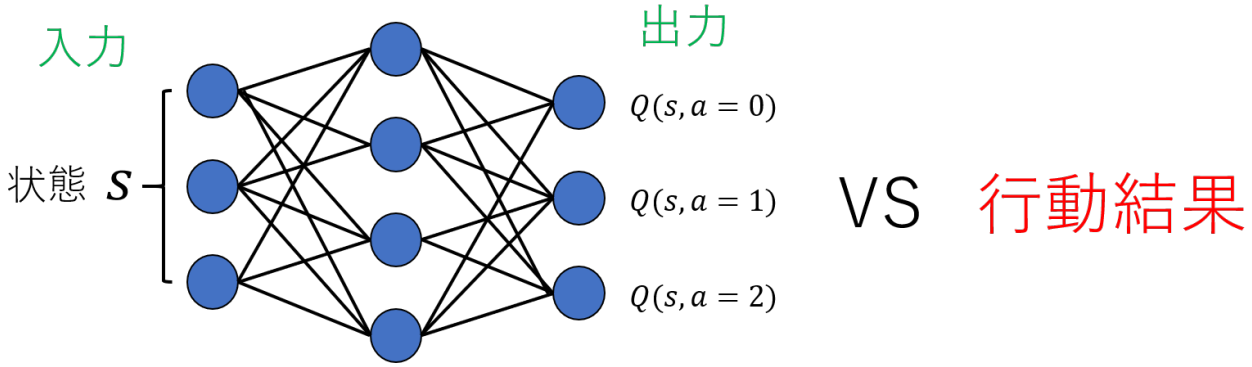
を予測値について微分した

$$\frac{\partial E}{\partial Q(s,a,\theta)} = Q(s,a,\theta) - \left(R(s,a) + \gamma \max_{a'} Q(s',a',\theta)\right)$$

を誤差逆伝播すれば，$\frac{\partial E}{\partial \theta}$ が計算でき，例えば確率的勾配降下法なら θ を

$$\theta \leftarrow \theta - \lambda \frac{\partial E}{\partial \theta}$$

と更新することができます．



DQN の工夫

DQN の学習の原理は以上ですが，このままでは学習の効率が悪く，しかも不安定です．いくつかの工夫が提案されていますが以下の 3 つが代表的です．

- Experience replay：行動記録を溜めておいてそこからランダムに選んでミニバッチを作り学習を行う．一つの行動が何度か学習に使えるようになるとともに，過去の記録と混ぜて学習することで安定化させます．
- Fixed target Q-Network：予測に使うネットワークのパラメータは毎回更新するが，教師データに使うパラメータ θ は毎回更新はせずに，一定間隔を置いて予測に使うネットワークをコピーする．教師データを安定させると学習も安定するようです．
- 勾配クリッピング：誤差関数の傾きの絶対値がある一定値を超えないようにする．例えば一定値が 1 なら，$E = (\hat{y} - y)^2$（$\hat{y}$：予測値，y：教師値）の代わりに

$$E = \begin{cases} (\hat{y} - y)^2 & (|\hat{y} - y| \leq 1) \\ |\hat{y} - y| & (|\hat{y} - y| > 1) \end{cases}$$

を使うことで，勾配 $\dfrac{\partial E}{\partial \hat{y}}$ が大きくなりすぎるのを防ぎます．

DQN のアルゴリズム

```
Q-Network Q を生成し，ターゲットネットワークを QT = Q とする
for episode = 1 to max_episode:
    初期状態 s を作る
    while ゲームオーバーでない:
        epsilon-greedy 法で行動（s,a）を決める
        行動を行い次の状態 s’，即時報酬 r，gameover かどうかの真偽を決める
        memory に s，s’，r，ゲームオーバーの真偽を追加
        if memory に記録された数 > 一定数:
            memory からミニバッチを選ぶ
            X_batch の一つのデータ：x = s
            Y_batch の一つのデータ:y = r + (1-discount_rate) * max(QT(s’,a’), axis=1)
                                （ただし，ゲームオーバーのときは y = r）
            (X_batxh, Y_batch) から Q の勾配（クリッピング）を計算して Q を更新
        定期的に QT = Q と更新
        s = s’
```

5.1 DQN に関する演習

使用するノートブック:

1. `4-7-1_DQN_CartPole.ipynb`
2. `4-7-2_DQN_Breakout.ipynb`

DQN を用いて CartPole と Breakout というゲームを解きましょう．CartPole は台車の上に立っている棒が倒れないように台車を左右に動かすというゲームです．状態は (台車の位置，台車の速度，棒の角度，棒の角速度) の 4 次元で行動は 0 : 台車を左に押す，1 : 台車を右に押す，の 2 種類です．

一方，Breakout は落ちて来る球を左右に動くブロックで跳ね返して上の方にあるブロックを壊すというゲームです．状態はゲームの画面そのもので，行動は 0 : 停止，1 : 発火，2 : 左に動く，3 : 右に動く の 4 種類です．

今回の実装ではゲーム環境に OpenAI が提供する gym というライブラリを用います．DQN の実装には Keras を用います．ノートブック 1 では Fixed target Q-Network と 勾配クリッピングは用いていません．ノートブック 2 では Keras-rl というライブラリを用います [https://github.com/keras-rl/keras-rl]．