ディープラーニングを使用した強化学習とその可能性

ソリューション本部　開発二部

西浦　佑紀

太田　桂吾

1. はじめに

1．1　機械学習・ディープラーニングとは

機械学習とは、既存のデータを解析し、何らかの法則を導く、という手法である。データから法則を”学習”するため、機械学習と称される。

機械学習の中の一つに、「ニューラルネットワーク」がある。これは人間の脳のシナプスの結合を模した手法であり、さらにその中の一手法として、ディープラーニング（＝ディープニューラルネットワーク）がある。(図 1)

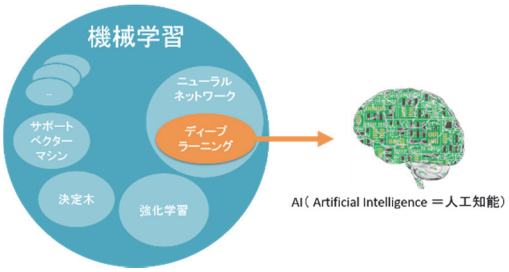

図 1　機械学習とディープラーニング

近年、ディープラーニングという手法が非常に優秀な結果を出すことが発見され、多くの分野で使用されるようになった。

現在、AI（ Artificial Intelligence ＝人工知能）という名称がついているサービス、商品では、このディープラーニングを用いて作成されたモデル（≒アプリケーション）を使用している。

AI がデータから学習をする例として、アイリス(アヤメ属の花)を識別する AI を挙げる。

表 1：取り込むアイリスのデータ

がく辺の長さ	がく辺の幅	花びらの長さ	花びらの幅	種類
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
…	…	…	…	…
6	2.2	4	1	versicolor
6.1	2.9	4.7	1.4	versicolor
5.6	2.9	3.6	1.3	versicolor
…	…	…	…	…
5.6	2.8	4.9	2	virginica
7.7	2.8	6.7	2	virginica
6.3	2.7	4.9	1.8	virginica
…	…	…	…	…

はじめに、表 1 にあるアイリスの4項目のデータとその種類を取り込んで、AI に学習させる。(図 2)

図 2　学習の流れ

そして、学習が済んだ AI に新規のデータを投入すると、その種類を自動で判別するのである。

ここで注意するのが、学習の際に「がく辺の長さが○○～○○でかつ、花びらの長さが○○以上なら、種類は○○」というような情報は与えないということである。データ（4つの項目）とその正解（種類）だけを頼りに、コンピューターが判断基準を作り出すのである。

1．2　人工知能は3回目の“ブーム“

人工知能の 1 次ブームは 1950 年台、「探索・推論」による特定の問題を解く時代である。ルールとゴールが厳密に定められているパズルのようなゲームは

できる一方、ルールやゴールが曖昧な現実世界に対応できるといった知識はなかった。

人工知能 2 次ブームは 1990 年代、「エキスパートシステム」が登場し、コンピューターに「知識」を与える試みがなされた時代である。ルールに基づいたデータを入力することで投資や医療診断の意思決定を促す、専門家（エキスパート）の代行を行うシステムが実用化された。しかし、知識をすべてプログラムに記述するため、専門的な狭い分野での知識はルール化することができても、一般常識のような広い分野を網羅することは不可能であった。また、システムで定められた範疇でしか有用でないという問題点もあった。

そして現在の、人工知能の 3 次ブームは「機械学習・ディープラーニング」を象徴とする時代である。データから物事を学び、“認識”する。代表されるものとして、IBM ワトソン、感情認識ヒューマノイドロボットの Pepper、自動運転機能、Siri などが挙げられ、このブームの背景にはビッグデータを扱える処理能力（ネットワーク、計算速度、ディスク容量）の向上が伺える。

2. ディープラーニング

ディープラーニングとは、機械学習に含まれる技術の一手法であり、ニューラルネットワークの多層化、特に 3 層以上のものを指す。ニューラルネットワークとは脳のシナプス結合を模した認識モデルである。（図 3）

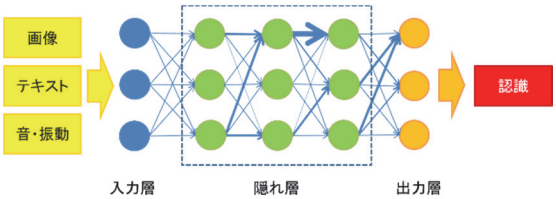

図 3　ニューラルネットワーク

各ニューロン（○）が、学習によってシナプス（→）の結合強度を変化させ入力値に対する認識結果を出力する。シナプス結合の強度を変化させるアルゴリズムは以下である。

① 画像、テキスト、音などの情報を数値化し、それを入力とする。
② 初期のシナプスの結合強度はランダムに設定。
③ 隠れ層のニューロンへ情報を渡していき、認識結果を生成、正解と比較した時の誤差を用いて、シナプスの結合強度を更新していく。（誤差逆伝播法）

ディープラーニングでは、このネットワークにデータを入力すると、情報が第 1 層から隠れ層に深く伝達される間に、各層で学習が繰り返されるのである。

さらにこの過程で、これまでは画像や音声などそれぞれの特徴量は、データの研究者、技術者が手動で設定していたが、ディープラーニングでは、これらが自動で計算されるのである。（ここでの特徴量とは、問題の解決に必要な本質的な変数や、特定の概念を特徴づける変数=項目を指す。）

この階層的な特徴量の学習が、ディープラーニングが従来の機械学習と決定的に異なる点である。

また、ディープラーニングではただ単に層を増やしたわけではなく、入力信号の総和を出力信号に変換する活性化関数の工夫やドロップアウトなどの手法を付加したことで、驚異的な精度向上を実現した。

そのほかニューラルネットワークを使用する代表的なアルゴリズムは以下である。（詳細は割愛する）。

- 畳み込みニューラルネットワーク：CNN（Convolutional Neural Network）
- 再起型ニューラルネットワーク：RNN（Recurrent Neural Network）

3. 環境構築

個人の端末で人工知能を実装するための環境構築方法を記述する。対象 OS は以下である。

- WIndows7 64bit
- WIndows8 64bit
- WIndows10 64bit

(32bit 版では tensorflow が動作しない)

言語は Python を使用し、本稿では Python3 系で記述する。環境構築する際に Python2 系がすでにインストールされている場合は、Python2 系をアンインストールするか、環境の切り替えが必要である。

はじめに anaconda4.4 64bit をインストールする。(Anaconda は Python の本体と、使用頻度の高いライブラリをセットにした Python パッケージである。)

次に強化学習で使用する以下のライブラリをインストールする。

● Tensorflow

Tensorflow は Google 社が開発したデータフローグラフを使用したディープラーニングライブラリであり、ニューラルネットワークを容易に記述できる。

● Chainer

Chainer は、ニューラルネットワークを誤差伝播で学習するライブラリである。日本の企業である「Preffered Networks」が開発している。

● Numpy

Numpy は python で数値計算用を行う際の拡張モジュールであり、多次元配列や行列を高速で処理することができる。

● OpenCV

Opencv は画像処理で使用するライブラリである。

TensorfloとChainerはAnaconda prompt(anaconda用コマンドプロンプト)で下記のコマンドよりインストールが可能である。

```
python -m pip install tensorflow
python -m pip install chainer
```

Numpy と OpenCV のインストール方法は下記の通りである。

URL(OpenCV)

http://www.lfd.uci.edu/~gohlke/pythonlibs/#opencv

から下記をダウンロード

opencv_python-3.3.0+contrib-cp36-cp36m-win_amd64.whl

URL(Numpy)

http://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy

から下記をダウンロード

numpy-1.13.1+mkl-cp36-cp36m-win_amd64.whl

それぞれを任意の場所に配置し、Anaconda prompt から下記を入力する。

```
pip install ファイルへのフルパス
```

◆動作確認

スタートメニューから python3.5(64bit)を起動し、下記を入力する。この時にエラーがでなければ問題ない。(図 4)

```
import scipy
import numpy
import tensorflow
import chainer
import cv2
```

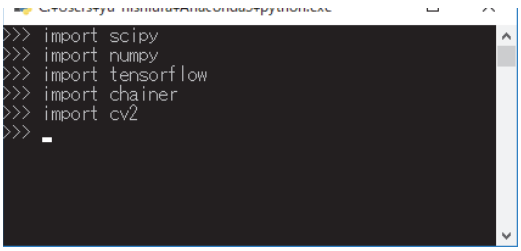

図 4 python での動作確認

4. 強化学習

強化学習とは、機械学習に含まれる学習手法の一種であり、試行錯誤を通じて、「最終的な報酬を最大化するような行動」を学習することをいう。強化学習における一連のプロセスは以下のようにモデル化する。

- 状態(Status(S)):環境が今どうなっているかを表す
- 行動(Action(A)):エージェント(学習対象)が環境に対して、どのような行動を起こせるかを表す
- 報酬(Reward(R)):ある状態において、エージェントが行動を起こした結果、発生する値
- 戦略(Policy):報酬を最大にする行動を選択する関数

それぞれの関係を図5に示す

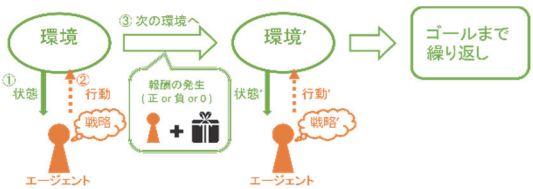

図5 強化学習 環境とエージェントの関係

ある状態(Status)の下で行動(Action)を選択すると、次の環境に移る。そこで報酬(正負の値、または0)が発生する。行動を繰り返していき、ゴール(決める必要がある)までたどり着いた段階での、報酬(Reward)の総和が最大になる戦略(Policy)を学習する。なお、次の環境への遷移がその直前の状態と行動にのみ依存し、それ以前の状態や行動には関係しないことを前提とする(マルコフ性)。

すべてのパターンを試せば結果はでるが、膨大なデータ量になるため、実現することは容易でない。

そこで、特徴量を自動で計算し、すべてのパターンを網羅せずとも、有効な戦略を学習することが可能なディープラーニングを使用した、深層強化学習が注目され始めたのである。

5. 試行

強化学習のサンプルとして、ChainerRL と OpenAI gym を使用し実装した。

ChainerRL は深層強化学習を容易に扱うためのライブラリであり、Deep Q-Network や A3C といった深層強化学習用のアルゴリズムを共通のインタフェースで使用できる。OpenAI gym はエージェントが学習するための実験環境が入っているライブラリである。

まず anaconda prompt で以下を実行し、ライブラリをインストールする。

```
pip install chainerrl
```

```
pip install gym
```

■ ソースコード(抜粋)

5.1 必要なライブラリのインポート

```
import chainer
import chainer.functions as F
import chainer.links as L
import chainerrl
import gym
import numpy as np
```

5.2 環境の設定

```
env = gym.make('CartPole-v0')
print('observation space:', env.observation_space)
print('action space:', env.action_space)

obs = env.reset()
print('initial observation:', obs)

action = env.action_space.sample()
obs, r, done, info = env.step(action)
print('next observation:', obs)
print('reward:', r)
print('done:', done)
print('info:', info)
```

ChainerRL を使用するにあたって、環境を設定する必要がある。OpenAI gym では“env = gym.make(※)”(※は gym にある環境の名前)で使用できる。

今回使用する環境「CartPole（倒立振り子）」は左右に動く台車の上に、垂直に棒が立っており、それを倒さないようにバランスをとるというものである。イメージとしては、ほうきを手に乗せて、そのまま保つ、といったものである。（図 6）

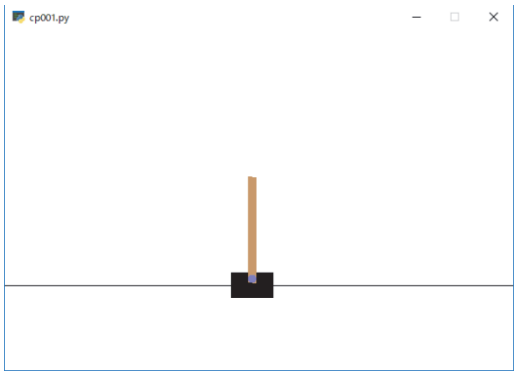

図 6　CartPole-v0

環境に最低限必要な要素は、ある時刻での状態（observation_space）、時刻tでエージェントが選ぶ行動（action_space）、環境の初期化メソッド（env.reset）、

次の環境への遷移、報酬の発生などを実行するメソッド（env.step）である。

5．3　エージェントの設定（Q 関数）

```
    class QFunction(chainer.Chain):

        def __init__(self, obs_size, n_actions, n_hidden_chann
els=80):
        super().__init__(
            l0=L.Linear(obs_size, n_hidden_channels),
            l1=L.Linear(n_hidden_channels, n_hidden_channels),
            l2=L.Linear(n_hidden_channels, n_actions))

        def __call__(self, x, test=False):
            h = F.tanh(self.l0(x))
            h = F.tanh(self.l1(h))
            return
chanerrl.action_value.DiscreteActionValue(self.l2(h))
```

ここで設定する Q 関数とは、今の状態と次にとる行動で、どれだけの報酬を得るかを計算するものである。今回は DQN（Deep Q-Network）を使用し、80 次元×3 層のニューラルネットワークとした。

5．4　最適化手法とパラメータ設定

```
obs_size = env.observation_space.shape[0]
n_actions = env.action_space.n
q_func = QFunction(obs_size, n_actions)
optimizer = chainer.optimizers.Adam(eps=1e-2)
optimizer.setup(q_func)
gamma = 0.95
explorer = chainerrl.explorers.ConstantEpsilonGreedy(
    epsilon=0.3, random_action_func=env.action_space.sample)
replay_buffer=chainerel.replay_buffer.ReplayBuffer(capacity=10
** 6)
phi = lambda x: x.astype(np.float32, copy=False)
agent = chainerrl.agents.DoubleDQN(q_func, optimizer,
 replay_buffer, gamma, explorer, replay_start_size=500,
 update_interval=1, target_update_interval=100, phi=phi)
```

まず optimizer で正解との誤差を最小にするための最適化手法を設定する。次に過去の結果をどれだけ重要視するかを gamma 値（0 から 1）で設定する。エージェントが取る次の戦略を explorer で設定し、今回は e-greedy 法を用いた。この手法では epsilon（0 から 1）の値が大きければよりランダムな選択をするようになるのである。

5．5　学習

```
n_episodes = 200
max_episode_len = 200
for i in range(1, n_episodes + 1):
 obs = env.reset() #環境の初期化
 reward = 0 #報酬の初期化
 done = False
 R = 0 # return (sum of rewards)
 t = 0 # time step
 while not done and t < max_episode_len:
 action = agent.act_and_train(obs, reward)
 obs, reward, done, _ = env.step(action)
 R += reward #報酬を取得
 t += 1
 if i % 10 == 0
 print('episode:', i, ,'R:', R) #取得した報酬を出力
```

学習は200回(n_episodes)行い、1回の学習において、エージェントは状態を受け取り、それに対して棒を倒さないように行動を選択する。その結果、棒を倒さなかった場合に 1 の報酬を得る。これらを t=0 から 200(max_episode_len)まで繰り返すというものである。

以下は学習を行った結果である。10 エピソード単位で出力し、R はそのエピソードで得た報酬である。

episode:10 R:10	episode:110 R:116
episode:20 R:8	episode:120 R:139
episode:30 R:10	episode:130 R:151
episode:40 R:13	episode:140 R:200
episode:50 R:14	episode:150 R:117
episode:60 R:10	episode:160 R:200
episode:70 R:11	episode:170 R:149
episode:80 R:16	episode:180 R:200
episode:90 R:24	episode:190 R:157
episode:100 R:38	episode:200 R:167

5. 6 テスト

学習の後、テストを実行する。学習済のエージェントで、テストを 10 回実行した結果が以下である。

testepisode:0 R:200	testepisode:5 R:200
testepisode:1 R:200	testepisode:6 R:200
testepisode:2 R:200	testepisode:7 R:200
testepisode:3 R:200	testepisode:8 R:200
testepisode:4 R:200	testepisode:9 R:200

5. 7 考察

サンプルの結果から、200 回程度の試行で十分に学習するケースが作成できたことがわかる。しかし epsilon の値や、隠れ層の次元数などのパラメータの設定によっては、1000 回以上の学習を行ってもエージェントが成長しないケースも確認された。また学習の回数を増やし過ぎた場合も、一定の行動をとり続けてしまい、十分に学習しないケースも確認された。対象とする環境に応じたチューニングを行うことが重要であり、難しい点である。

6. 可能性

本稿では、ディープラーニングのサンプルとして、シンプルなものを用いた。しかし、世界で発表されている開発例からわかるように、ディープラーニングは膨大なデータから法則を見つける能力が人間よりも優れているのは間違いない。さらに機械学習では、文章や画像、時系列データはすべてベクトルに変換した後に処理をするため、入力データの形式にかかわらず学習アルゴリズムに適応できるといった可能性や、他分野(IoT、ロボティクスなど)とのシナジー効果による更なる拡張性も期待できる。

ビジネスにおいても Q&A のやり取りを学習し、文章から相手の意図を読み取ることで、正しい返答をする AI チャットボットの運用が始まっており、カスタマーサポートに掛かる人件費削減が見込まれている。このように確定したロジックがなく、柔軟な対応が要求される分野はもちろんのこと、多くの分野で AI・機械学習・ディープラーニングが適応し、広く普及していくだろう。

＜参考文献＞

1) 「Chainer」： https://chainer.org/