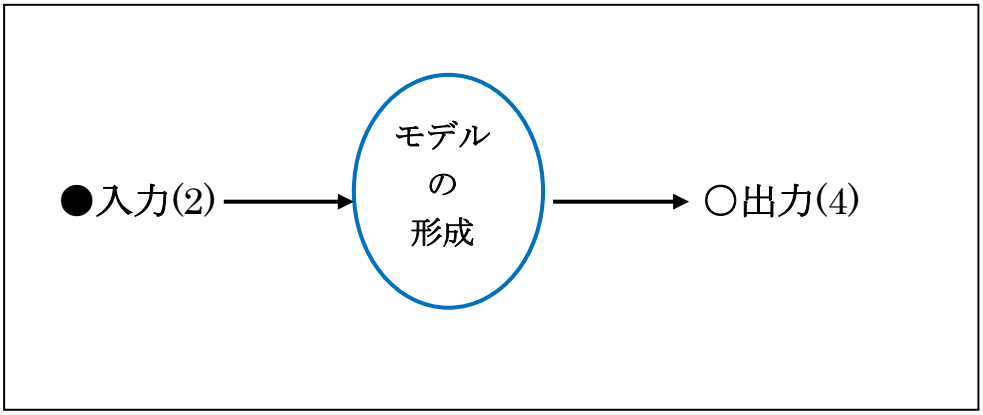
第３３話（ＡＩプログラミング）

ディープラーニングプログラミング

タヌキ！プログラミングに取り掛かる前に、第３２話で説明した内容の復習をしておく。
下図のように入力値（２）から出力値（４）になるように、コンピュータが勝手にモデルを形成する。形成されたモデルは、加算かもしれないし、乗算かもしれない、あるいは全然違う手法かもしれない。この形成されたモデルがブラックボックスと言われるものである。当然、［２→４］という１組のデータから常に正解に近づくモデルができるわけではない。それで［２→４］、［３→６］、［４→８］・・・・・という多くの様々なパターンのデータを用いて学習させなければならないのである。学習させることによって強固なモデルが形成されることになる。これが、インターネット上のビッグデータを活用し、学習させる、という意味になる。モデルが形成された後で、学習させていない、例えば［８］という入力値を設定し、15.98654 というような 16 に近い出力値が得られればモデル作成のディープラーニングのプログラミングに成功したことになる。



作られたモデルのアルゴリズムはブラックボックスか！モデルのアルゴリズムを知ることは無理として、キツネ、ディープラーニングの学習のさせ方を教えてくれ！

タヌキ、了解だ。以下の図の説明をする。
ディープラーニングの学習は、下図のように最初はランダム（適当という意味）な値：予測値（2.6：手動も可）を与える。それからロスを計算する。

ロス　＝　２．６（予測値）　－　４（正解）

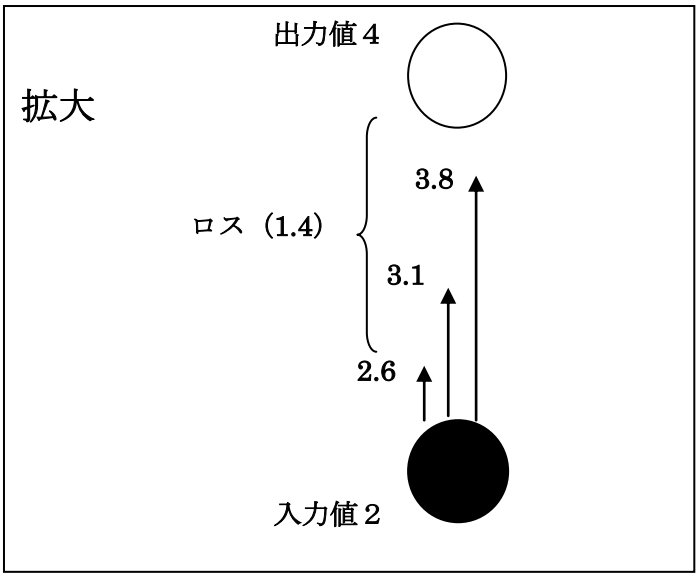
次の値は、ロスを小さく（絶対値で）するように関数が、選択（プログラムで行う）する。

ロス　＝　３．１（予測値）　－　４（正解）

次の値も同様である。

ロス　＝　３．８（予測値）　－　４（正解）

という具合になる。このロスを小さくするのに使われる関数が、最適化関数や活性化関数といわれるものである。最適化関数は、最小二乗法、確率勾配降下法など多種多様である。学習に最適な最適化関数を選択し、個別の業務に対応したモデルを形成することがディープラーニングのプログラマーの仕事だ。そのプログラマーは、業務に精通しているとともに、<u>最適化関数を選べるだけの数学の力も要求されるのだ。</u>
では、今までの説明をディープラーニングのプログラムの形にして、実行してみたらどうなるか、というのが第３３話のテーマだ。



キツネ、オイラにもなんとなくわかってきたが、誰がブラックボックスのアルゴリズムを作ったのだ？

最初は、ネコの画像認識やアルファ碁で有名な米国のグーグル（現：アルファベット）が開発したのだ。そのグーグルが画像認識で活用した Tensorflow というフレームワーク（ライブラリーと考えても良い）をオープンソースとして提供しているのだ。多くの日本の深層学習のプログラマーがこれを使っている。これを使えばディープラーニングの概要を知ることもできる。

では、今までのディープラーニングの解説に基づいて「簡単な例題」を解くためのディープラーニングのプログラム例（学習のさせ方、学習内容の保存、保存した学習の呼び出し）を行ってみるか。以下のプログラムは、Python 言語に Tensorflow というフレームワークを組み込む形で書いている。

はじめに、ディープラーニングのプログラミングの概要だ

ディープラーニングのプログラムを大まかに把握することが大切だ。

これは、画像認識用のプログラムでも同じだ。

（概要）

- 入力と出力（正解）データのセット
  （例）入力：複数のネコの画像を読み込ませる。
  出力（正解）：ネコ
- データの最適化（学習）、つまりモデルの作成
  （例）画像の場合、ピクセル単位でロスの少ない最適化を行なう。
- 新規データを入力し、予定されている出力（正解）との照合（正解に近い確率％で表す）。
  （例）様々な動物の静止画像を入力し、その中からネコの画像である確率をパーセントで表示させる。

実際のプログラムは次ページのようになる。

「簡単な例題」プログラム例：ファイル（keisan.py）

```
import tensorflow as tf
import numpy as np

input_dim=1
output_dim=1

x = tf.placeholder("float",[None, input_dim])
w = tf.Variable(tf.random_uniform([input_dim, output_dim], -1.0, 1.0))
b = tf.Variable(tf.zeros([output_dim]))
y = w * x + b

t = tf.placeholder("float",[None, output_dim])

loss = tf.reduce_mean(tf.square(y - t))

train_step = tf.train.MomentumOptimizer(0.01,0.97).minimize(loss)

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

for i in range(100):
    batch_xs = np.array([[2.],[3.],[4.],[5.]])
    batch_ys = np.array([[4.],[6.],[8.],[10.]])

    sess.run(train_step, feed_dict={x: batch_xs, t:batch_ys})
    print(i,sess.run(y, feed_dict={x: batch_xs, t:batch_ys}))

print("gakusyugo-yosokuchi")

print(sess.run(w) * 8 + sess.run(b))
```

［プログラム例「keisan.py」の解説］

はじめに①、②、③・・・を、「keisan.py」内の大雑把な部分として見てほしい。

①フレームワーク（tensorflow、numpy）のインポート。「as tf」、「as np」は、tensorflow、numpy の文字が長いので、「tf」,「np」省略形を使う。

②単純に変数指定と数値の代入。入力の１、出力の１は、その後パラメータとして使う１次元を意味している。

③学習の仕方を記述しているもっとも重要な部分。ここに、最適化関数や活性化関数を記述します。どのような関数を用いて学習させるかは、数学の得意なプログラマーの腕の見せ所である。

・x は学習させる入力データ（2、3、4、5）。.placeholder（）メソッドは⑤の feed_dict と密接な関係を持っている。

・t は x に対する出力（正解）データ（4、6、8、10）。placeholder（）メソッドは⑤の feed_dict と密接な関係を持っている。

・w は重み、初期値は乱数で適当な値を設定している。この問題では、直線の傾きと考えて良い。

・b はバイアス。この問題では、縦軸との切片。

・ｙは、ｗとｂから求める予想値。１次方程式。

・loss（ロス）は、最小二乗法で最小の平均値を取っている。地震の震源決定などでもよく使われている手法である。

・train_step は、③の計算式全てを内包し、最適化関数の部分で最も重要である。MomentumOptimizer()（モメンタム法によるオプティマイザー）という最適化クラスで最適化している。他にも GradientDescentOptimizer()いう最適化クラスもある。この関数の処理内容の具体的な手法は、数学の領域である。

④おまじない（？）：この３行は、Tensorflow の仕組みに従ったものである。Tensorflow の場合は、セッションを作り、セッションで変数を初期化して実行するという規則に従う。さもなければ、エラーになるので、この３行は必ず必要である。

⑤ミニバッチ batch_xs （配列）として入力値（2、3、4、5）を設定し、batch_ys に、正解である出力値（4、6、8、10）を設定し、100 回学習させている。結果は、その下の実行結果に示している。100 回目（99）で[3.90966535],[5.92648077], [7.94329643],[9.96011162]]）となり、正解（4、6、8、10）に近づいていることがわかる。

⑥100 回学習後、未知の値（８）を設定し、16 を予測できるか試している。結果は、[16.01055908]である。おみごとと言うしかない。

タヌキ！次は、このプログラムの実行結果を示すことにする。１００回学習させて、［４、６、８、１０］という正解のどのように近づくか見てみよう！回数が多いので、３回目から９８回目までは省略だ。
ただ、９９回目を見ればわかるように、学習回数を増やせばより正解に近づくわけでもない、逆に発散し、正解から遠ざかることもある。適正な学習回数というものもあるようだ。

実行結果

```
Ubun@14-32:~$ python keisan.py          実行
(0,  array([[ 0.78011608],    試行 1 回目       正解 4
        [ 1.08909798],                          正解 6
        [ 1.39807975],                          正解 8
        [ 1.70706165]], dtype=float32))     正解 10
(1,  array([[ 3.15634394],    試行 2 回目
        [ 4.51723194],
        [ 5.87811947],
        [ 7.239007   ]], dtype=float32))
   :
(98,  array([[  4.49484825],    試行 99 回目
        [  6.77231359],
        [  9.04977989],
        [ 11.32724571]], dtype=float32))
(99,  array([[ 3.90966535],     試行 100 回目    正解 4
        [ 5.92648077],                          正解 6
        [ 7.94329643],                          正解 8
        [ 9.96011162]], dtype=float32))         正解 10
```

プログラムでは、最後に新規の値「8」を設定し、正しく学習されているのか試している。２倍の１６が正解なのだが、どうだろう。その結果が右図だ。おみごとではないか！

```
gakusyugo-yosokuchi
   学習後：8 の 2 倍の予測値

[[ 16.01055908]]          正解 16
```

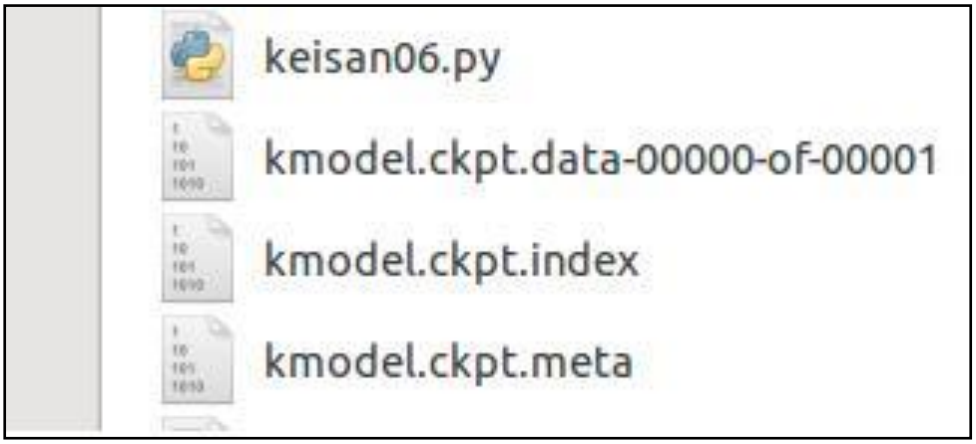
タヌキ！学習した後で、回帰直線（曲線）のパラメータをＡＩのアプリに渡すことができれば、その後学習の必要が無くスマホでもＡＩアプリを実行できる、と前に言ったよな。これからが、その話になるのだ。
先ずは、学習後のパラメータの保存だ。保存する為には、プログラムを以下のように若干編集しなければならない。編集したプログラム名を「keihozon.py」としている。

［モデル保存プログラムに改良］　　「keihozon.py」

```
    :   ここまでは「keisan.py」の①②③④
    :   と同様にします
    :
hozon = tf.train.Saver()    #  追加（定義）
for i in range(100):
    batch_xs = np.array([[2.],[3.・・
    batch_ys = np.array([[4.],[6.・・
    sess.run(train_step, feed_dic・・
    print(i,sess.run(y, feed_dict・・
hozon.save(sess,"kmodel.ckpt")  #  追加（保存）
```

変更なし

「keihozon.py」を実行すると、カレントディレクトリ（keihozon.py の場所）に、以下のような学習済みモデルである３個のファイル（kmodel.ckpt～）ができる。これらのファイルは、バイナリー形式なので中をみることはできない。まさにブラックボックスである。



プログラム：「keiread.py」の例

「keisan.py」の ① ② ③ ④ まで、そのまま利用

学習部分の ⑤ は削除し、学習モデル読み込みの以下の２行を追加する。

```
hozon = tf.train.Saver()
hozon.restore(sess,"kmodel.ckpt")
```

⑥ は以下のように編集

```
print("gakusyugo-yosokuchi")
print(sess.run(w) * 9 + sess.run(b))
```

今度は逆に「保存した学習モデル」を読み込むプログラムに編集する。つまり、上図の３個のファイル（kmodel.ckpt～）を読み込む、と言うことである。その編集したプログラム名を「keiread.py」とする。

読み込んだ学習モデルを利用できなければ、保存する意味がない。また、ＡＩを提供するＩＴ企業からすれば、自社で学習させたものをフレームワークに同胞して販売できなければ、利益を得られない。

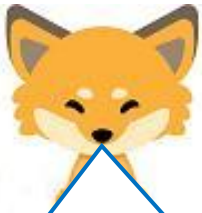
それで、当然のことながら、読み込んだ後で利用可能かどうかテストする必要がある。テストは、「９」という値を設定して「１８」という結果にどの程度近づいているか見ることにした。その結果が左図だ。

タヌキ、プログラム：「keiread.py」をもう少し説明しておくぞ。
①は、tensorflow のインポート、②は、変数定義、③は、学習のさせ方、④は、tensorflow 特有のセッション領域の確保なので、そのままの形で必要になる。学習モデルを読み込むので、⑤の学習部分は、必要なくなる。⑥は、新規に値を与えて２倍になるか確認する部分なので、新規の値「９」を与えて「１８」に近づくか、確認している。
　結果からの推測だが、セッション領域に、定義された変数名（w、b）と学習することによって最後に得られた w（重み）と b（バイアス）の値が記憶され、それらが、save メソッドによって「kmodel.ckpt」ファイル類に学習モデルとして保存されている。結果は、以下のようになった。

［実行結果］

```
Ubun@14-32:~$ python keiread.py  ←  実行

gakusyugo-yosokuchi

[[ 18.0175972]]  ←  学習が受け継がれ９の

                        ２倍に近い値が表示

Ubun@14-32:~$
```

すごいなキツネ！
ディープラーニングの学習させるプログラム「keisan.py」、パラメータを保存するプログラム「keihozon.py」、保存したパラメータを読み込んで実行するプログラム「keiread.py」に分けて説明してくれたので、オイラＡＩを理解したぞ。
これで、Python 言語もしっかりと勉強しておかなければならないことが良くわかった。



タヌキ！Python 言語の概要を話しておく。

グーグルからディープラーニングのフレームワークとして提供されている Tensorflow は、C++と Python で利用できる。ただ、コンパイラ言語の C++と聞いただけで面倒臭い、とディープラーニングの勉強を諦めてしまう人もいる。でも大丈夫だ。Python というインタプリタ言語で学習できる。むしろ、Tensorflow は、Python 用に提供されている、と考えた方が良い。Python を開発したオランダの数学者&プログラマーであるグイド・ヴァンロッサム氏は、グーグルでも働いていたようである。さらに、Python には、１命令毎にエラーが無いかどうか確認しながら入力できる iPython というものもある。

当初、深層学習のプログラムを理解する為に、Python 言語の勉強は必要だが、Python の理解だけでは深層学習の理解にはつながらない。前述したように、発想の転換が必要である。

Python は、標準ライブラリーだけで、行列計算など高校で学ぶ数学的な処理はできるが、さらに様々なパッケージをフレームワークとして組み込むことによって、高度な科学技術計算用言語として扱うことができる。その計算結果に縦軸・横軸のパラメータを設定するだけで、即グラフ化し、視覚的に確認できるメリットもある。他にもサードパーティのパッケージをインストールすることによって、ネットワークに関するプログラミング、例えば、収集したい静止画・動画を指定し、インターネットから拾い集めてきてビッグデータとして利用するプログラム、Web アプリケーション（HTML との連携、サーバサイドスクリプトの作成）の開発もできる。

ただ、Python を理解するためには、アセンブラ、C 言語、Java という地道な言語学習という下地が必要である、というのがオイラの見解だ。なぜなら、コンストラクタ、インスタンス、オブジェクトというオブジェクト指向言語で使われている用語が出てくる。配列（＝行列）もリストをはじめ多種多様のものが出てくるからだ。

次に、ディープラーニング（深層学習）についてのオイラの考えを述べておく。
ディープラーニング（深層学習）＝ＡＩでは無い。ディープラーニングは、ＡＩ構築の一手法である。だが、今は最も強力で有効な手法とみなされている。

ここで使用した簡単な深層学習のプログラム「keisan.py」はニューロン層が１層のプログラミングです。ニューロン層（人間で言うシナプスのつながり）を２層、３層と増やすということは、「keisan.py」の③の部分を複数作ることを意味する。例えば「keisan.py」で、ニューロン層を２層にする場合は、重み（w）とバイアス（b）がw1、w2を２個、b1、b2を２個作ることになる。出力された予想値（y）が次のニューロン層への入力値になるのですから、あたりまえと言えばあたりまえである。最適化関数や活性化関数を前の層と同じものを使うならば、他のスクリプトの部分は同じになる。ただ、層を増やせば、より精度の高い評価値がでるかと言えば、そうは言えない。発散してしまう場合も多いのだ。

深層学習を理解すると、その時々の入力値と出力値（結果）をデータとして用いてもナンバーズなどのギャンブル的なものの予測は不可能であることがわかってくる。もし、ギャンブル的なものに何らかの傾向があれば別だが。なければ、予測値は発散する。同様に、株価の予測の場合でいうと、トレンドに従っている場合には、ＡＩ（例えば、ロボアドバイザー）の予測は強力なものになるが、リーマンショックのようなカタストロフィー的な事象が発生した時には予測できずに大損するだろう。でもカタストロフィー的な事象は、人間も予測不可能だから、様々なパターン（入力値&出力値）を学習したロボアドバイザーに頼った方が利益が出るかもしない。

ニューラルネットワークには、単純にニューロン層を増やして行く形態や、少ないデータを有効に活用する為に、中間で出力された値を前に戻し、それを入力値とし、繰り返すリカレント（再帰型）ニューラルネットワーク、精度の高い評価値を得る為に、画像認識で使われる静止画（入力データ）をピクセル単位で、より小さい部分（カーネルという）に区切っていく畳込みニューラルネットワークがある。ただ、どのようなアルゴリズムのニューラルネットワークであろうとも、僅か 20 行程度の「keisan.py」の③が深層学習の考え方の基本となっている、とオイラは考えている。