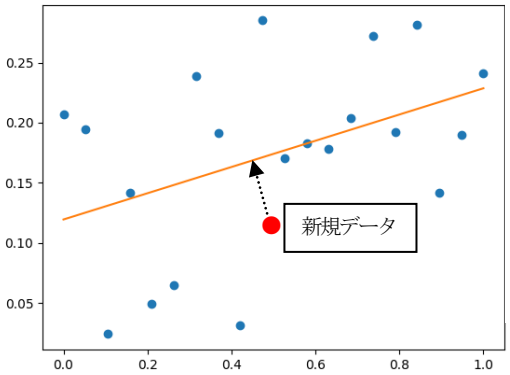
ＡＩへの考察！

ＡＩ（Artificial　Intelligence）システムを作っていく上で最も効果的な手法と考えられているのが、ディープラーニング（深層学習）です。

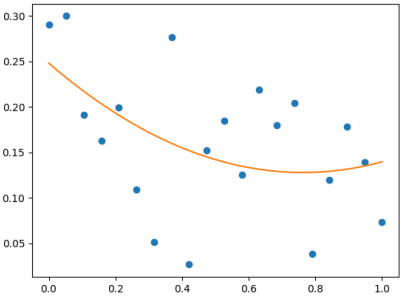
［ディープラーニングのポイント］

ディープラーニングは、人間が処理しきれない程の膨大なデータの有無とそのデータに適合する数式選択がポイントです。具体的な例を示すことにします。［例１］は、ランダムなデータ（水色🔵：膨大なデータと考えます）に１本の直線を引いています。この直線を回帰直線（オレンジ色の１次方程式：数式選択に相当します）と呼んでいます。仮に膨大なデータからこの１本の直線が引けた、と仮定します。新たに発生したデータ（赤色🔴：自分の飼っているネコの写真でも良いです）がこの直線にどの程度近いかを確率（%表示）で示します。直線は、１本とは限りません、２本に分かれるかもしれません。［例２］・［例３］は、膨大なデータに適合する数式（曲線）選択の試行錯誤例です。最適な数式選択ができるかは、ＡＩ開発者の数学的能力に依存します。

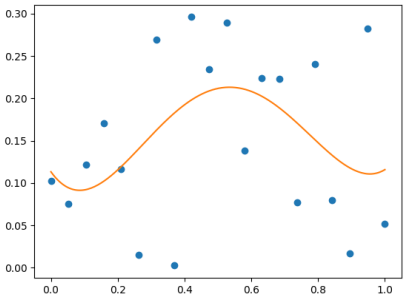
［例１］一次方程式（回帰直線）



［例２］二次方程式選択



［例３］三次方程式選択



［発想の転換が必要］

２７年前のＡＩ（Artificial　Intelligence）は、IF 文の塊（Prolog という言語を使用していた）で、その体験から、将来的にもコンピュータは、ＩＡ(Intelligence Amplifier)までで止まる、と考えていました。とうていシンギュラリティ（コンピュータが人間を越える）など起こるはずがない、と考えていました。

小学生以上の子供は、加減乗除を演算子を使って答えを出そうとします。さらに小さな子供は、２＋２＝４や２×２＝４の“×”や“＋”の演算子を知らなくても大小で足し算や掛け算に近いことをしようとします。これと同じで、プログラム言語に慣れ親しんだ人は、ＡＩプログラムをスクリプトとして理解しようとします。それでは理解に苦労します。

ディープラーニングを理解する為には、コンピュータはスクリプトに従って動作する、という固定観念から脱却し、多くの共通データに基づいて動作する、という発想の転換が必要です。

［ディープラーニングを理解する為の簡単な例題］

・簡単な例題：変数 a に 2 を代入し、答え４が表示されるように 2×a＝４をプログラミングしなさい。

（従来のプログラミング）

従来のプログラミングでは、次のようなスクリプトを書き、実行させます。

```
a　=　2;
b　=　2　*　a;
print(b);
```

このとき、アセンブラで書こうが、高級言語で書こうが“*”という演算子が入ってきます。これは、”*”演算子によって、最終的にビットを左にズラス、という指示をしていることになります。図示すると以下のようになります。

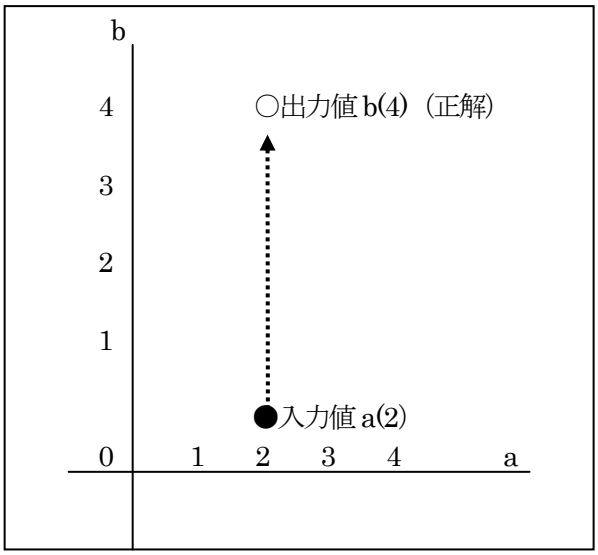
0	0	0	0	0	0	1	0

↓

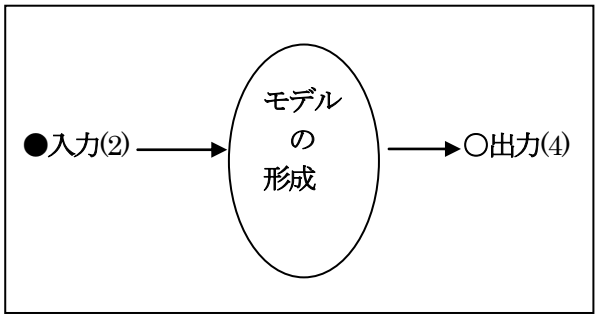
0	0	0	0	0	1	0	0

従来のプログラマーは、どうしても、このコンピュータの計算手法を消し去ることができません。ディープラーニングを理解する為には、この考えを脇に置いておきます。
現在は、早期の幼児教育で、子供たちは演算子を理解できるのかもしれませんが、普通３～５才の子が掛け算や足し算の演算子を知っているわけがありません。同様に、コンピュータも乗算や加算の手法としてビットをズラスということを知らない、とすればどのように計算させるのだろう、と考えて見てください。それが、ディープラーニングの理解につながります。

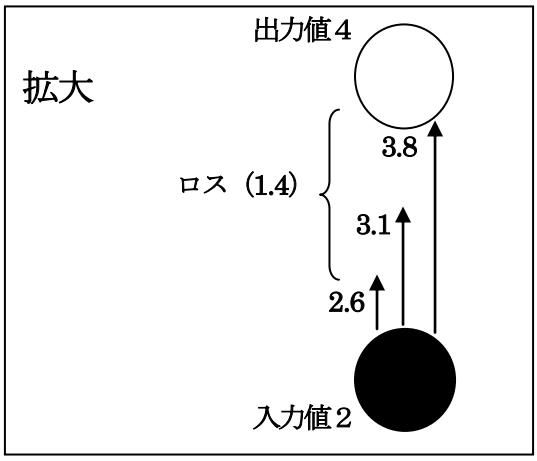
(ディープラーニングの考え方)
ディープラーニングは、トライアンドエラーの繰り返しです。図示すると以下のようになります。



上図の説明です。入力値 a（2）から正解である出力値 b（4）まで少しづつ 2、2.5、2.7、3・・・・という具合に近づけていきます。この場合、入力値２から出力値４が得られる結果は、乗算と加算が考えられますが、深層学習にとってはどちらでも良い事なのです。重要なのは、２という１個の入力値から４という結果が出力される、ということです。



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のです。形成されたモデルは、加算かもしれませんし、乗算かもしれませんし、全然違う手法かもしれません。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す。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ありません。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す。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ります。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、という意味になります。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ります。
(ディープラーニングの学習のさせ方)



ディープラーニングの学習とは、上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）を与えます。それからロスを計算します。

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）します。

ロス　＝　３．１（予測値）　－　４（正解）

次の値も同様です。

ロス　＝　３．８（予測値）　－　４（正解）

という具合です。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す。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す。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です。そのプログラマーは、業務に精通しているとともに、最適化関数を選べるだけの数学の力も要求されます。

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見てみましょう。

[ディープラーニングのプログラミングの概要]

発想の転換ができれば、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できるようになります。これは、画像認識用のプログラムでも同じです。

(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。

さて、ディープラーニングというと、世界の最先端を進むのは、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）です。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ます。日本の深層学習のプログラマーがこれを使っています。これを使えばディープラーニングの概要を知ることができます。

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）を示します。このプログラムは、Python 言語、Tensorflow というフレームワークで書かれています。Python 言語、Tensorflow というフレームワークについては、この後に解説します。

「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))
```

①
②
③
④
⑤
⑥

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

gakusyugo-yosokuchi   学習後：８の２倍の予測値
[[ 16.01055908]]          正解１６
```

[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ください。

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す。
- 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。

「keisan.py」で学習したモデルの保存

「keisan.py」で学習させているのは、⑤の部分です。極端に言えば、「sess.run(train_step, feed_dict={x: batch_xs, t:batch_ys})」という１行を 100 回繰り返しているのが学習です。ただ、batch_xs、batch_ys に設定されているデータが膨大になれば、相当の学習時間が必要になります。これが画像データとなれば、数日～数ヶ月（私は試していませんが）という時間になるようです。また、GPU が必要になったり、数百万円単位のハード面の強化も必要になります。

では、上記の［2→4］、［3→6］、［4→8］、［5→１０］の学習モデルを保存するスクリプトを追加します。保存は、学習後です。決して、［2→4］、［3→6］・・というデータを保存しているわけでないことを理解してください。

[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.・・          ├ 変更なし
                                              │
    sess.run(train_step, feed_dic・・          │
    print(i,sess.run(y, feed_dict・・          ┘

hozon.save(sess,"kmodel.ckpt")  #  追加（保存）
```

[注：これは Linux（ubuntu）で実行しています。Windows の場合は若干異なります。]

上記のプログラムを実行すると、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～）ができます。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ません。まさにブラックボックスです。

［保存した学習モデルの読み込み］

当然、学習モデルを利用できなければ、保存する意味がありません。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、利益を得られません。

さて、保存した学習モデルの利用は、以下のようになります。このプログラムを「keiread.py」とします。

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行を追加します。

```
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))
```

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ります。学習モデルを読み込むので、⑤の学習部分は、必要なくなります。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、確認しています。

結果からの推測ですが、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ようです。結果は、以下のようになりました。

［実行結果］

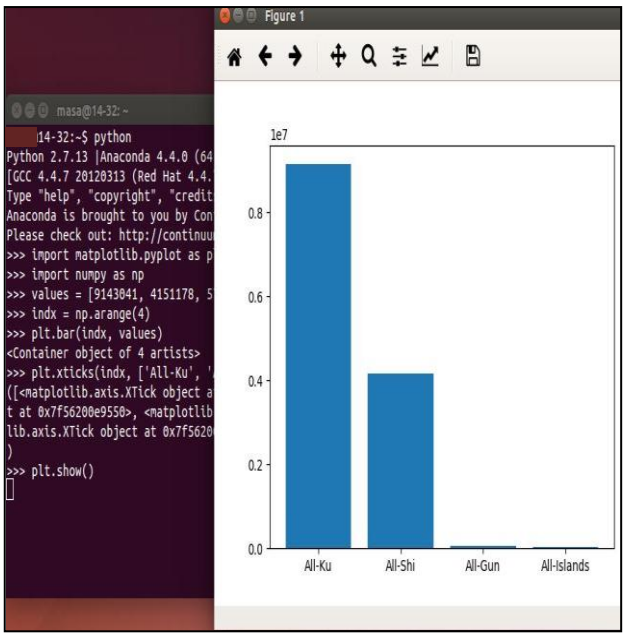
```
Ubun@14-32:~$ python keiread.py  ← 実行
gakusyugo-yosokuchi
[[ 18.0175972]]  ← 学習が受け継がれ９の
                   ２倍に近い値が表示
Ubun@14-32:~$
```

［Python 言語とは］

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 で利用できます。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おられると思います。でも大丈夫です。Python というインタプリタ言語で学習できます。むしろ、Tensorflow は、Python 用に提供されている、と考えた方が良いと思います。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す。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 というものもあります。

当初、深層学習のプログラムを理解する為に、Python 言語の勉強は必要ですが、Python の理解だけでは深層学習の理解にはつながりません。前述したように、発想の転換が必要です。

［Python2.7 の実行画面］OS（ubuntu14.04）



Python はインタプリタ言語です。以前より使われていましたが、グーグルが Tensorflow を公開してからは、深層学習用言語として脚光をあびるようになりました。最も面白い特徴は、インデント（字下げ）が必要な点です。IF 文の後、For 文の後、関数内の処理の記述で字下げをしないとインデントエラーになります。他にも、手続き型言語のように扱うこともできますし、クラスを作るオブジェクト指向型言語としても使うことができる、という特徴を持っています。

Python がわかりづらい、と感じるのは、メソッド内にメソッドを設定したり、設定したメソッド内にも多くのパラメータを設定しなければならない点です。リファレンスマニュアルらしきものを見ながら、その働きを確認しなければエラーになります。

初めて Python を使う時には、iPython というパッケージがあります。これは、フレームワーク名（例えば、tensorflow あるいは tf）の後ろに'.'（ピリオド）を入力し、［TAB］キーを押すだけで使用可能なメソッドの一覧が表示され、選択的に入力が可能です。また、インタプリタ言語ですので、そのまま実行し、エラーかどうか確認できます。

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ますが、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ます。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ります。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とか、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）の開発もできます。

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、というのが私の見解です。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出てくる用語が出てきます。配列（＝行列）もリストをはじめ多種多様のものが出てきます。

小学生にプログラミング言語として導入しようとしているスクラッチ（言語命令をスクラッチ風のカードに置き換えただけ）の本（10 歳からのプログラミング）の最後にも Python の簡単なプログラミング例が載っていました。そのような状況から、いきなり Python でも、という人もいますが、無理のような気がします。

Python のプログラムファイルの拡張子は、py です。当初は、Linux の Ubuntu で動作可能でしたが、Windows でも利用できるようになりました。環境の作り方は、インターネット上に数多く掲載されていますので、そちらを参照してください。フレームワークとして組み込む Tensorflow のインストールの仕方もインターネット上に掲載されています。

［ディープラーニング（深層学習）の詳細］

ディープラーニング（深層学習）＝ＡＩではありません。ディープラーニングは、ＡＩ構築の一手法ですが、今日では最も強力で有効な手法とみなされています。

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します。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個作ることになります。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す。最適化関数や活性化関数が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ります。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、そうは言えません。発散してしまう場合も多いのです。

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きます。もし、ギャンブル的なものに何らかの傾向があれば別ですが。なければ、予測値は発散します。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りますが、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でしょう。でもカタストロフィー的な事象は、人間も予測不可能ですから、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れません。

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ります。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、と私は考えています。