1変数関数を
Neural Netで近似する

2020.12.20/2021.02.25

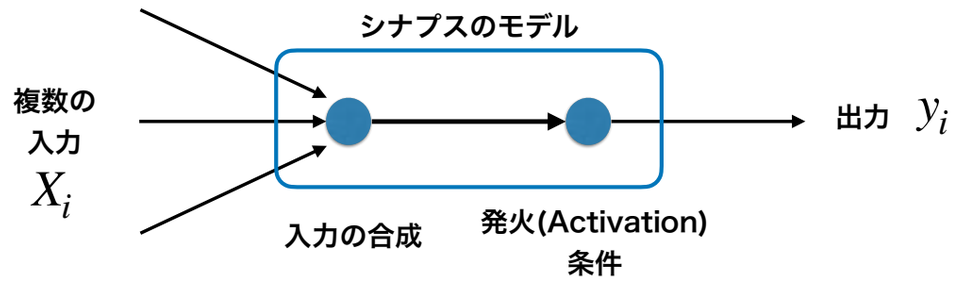
Noboru Yamamoto

Neural Netの働き方を1変数関数を近似するニューラルネットの動作から理解することを目指します。

概要

- Deep Learning に代表される多層ニューラルネットによる情報解析／機械学習が盛んに利用されている。
- これらのNeural Netがなぜ有効に働くのかはその複雑さから明らかではない。
- 多層ニューラルネットは、”一層以上(L>=2)の中間層を持つネットワークを用い、中間層のニューロンの数を十分多く取ることにより、有界閉集合上の任意の連続関数を任意の精度で近似できる.”(稲岡光他「知能制御」p121)ことが知られている。
- neural netの動作を理解するために、1変数関数の近似をニューラルネットワークを使って求め。その中でのニューラルネットワークの動作を見ることで、ニューラルネットの動作の理解を進めようという試みがこの小文の目的である。

Neural Networkのシナプスモデル



$$h_i = \Sigma_j W_{ij} X_j + b_i \qquad y_i = R(h_i)$$

ニューラルネットで使われる人工シナプスは、神経細胞の動作をモデル化した次のような動作を行う。

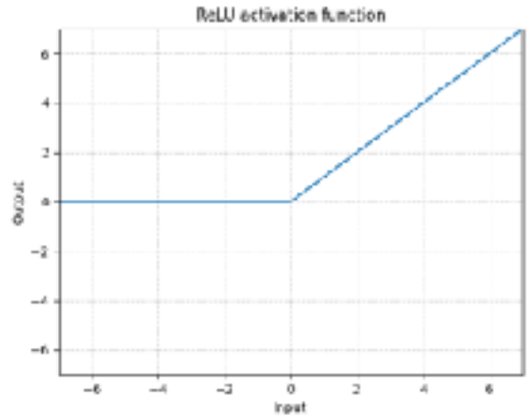
複数の入力源からの入力値の線形結合を求める。
この入力値の線形結合が発火条件と呼ばれる非線形関数に入力され
その結果が出力となる。

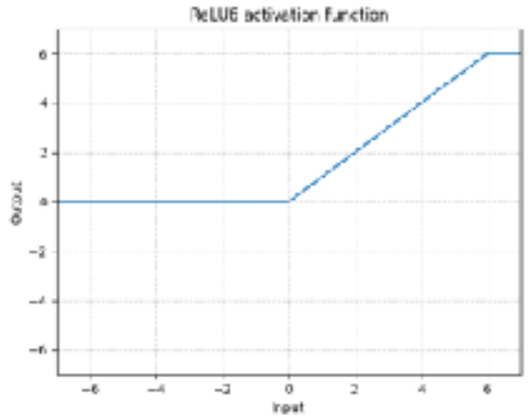
単一のシナプスだけ(単純パーセプトロン）では明らかに線形分離問題しか解くことができない。

ReLU/ReLU6

$$\mathrm{ReLU}(x) = \max(0,x)$$

$$\mathrm{ReLU6}(x) = \min(\max(0,x),6)$$





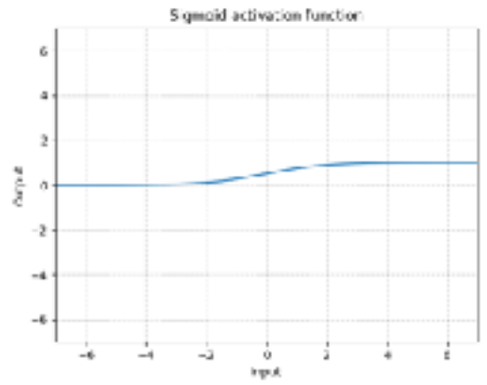
図はPyTorch documentより

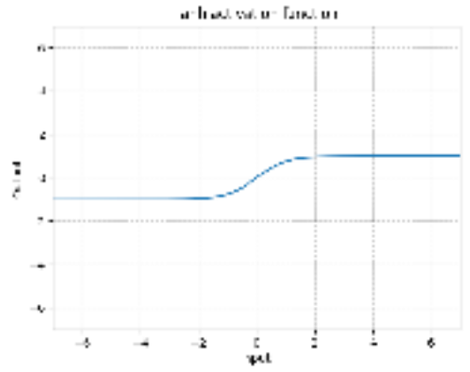
Activationの非線形関数として、このページに示されている, ReLU（Rectified Linear Unit) /ReLU6や次のページに示される Sigmoid(logistic)関数、Tanh関数などがよく使われる。

Sigmoid(logistic)/Tanh

$$\mathrm{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

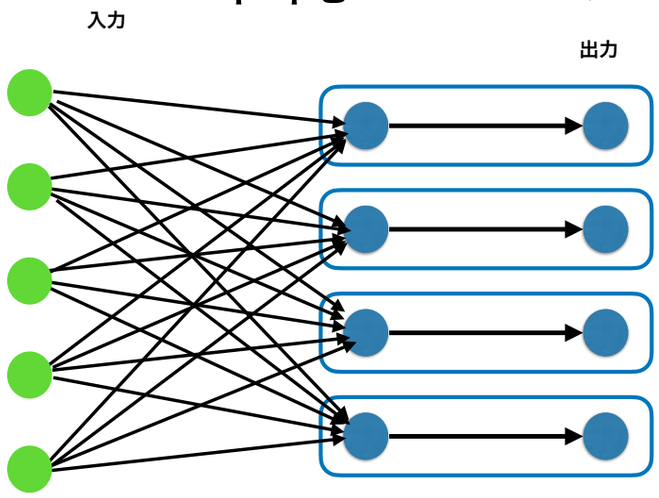
$$\mathrm{Tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = -1 + \frac{2}{1 + e^{-2x}}$$





図はPyTorch documentより

単純パーセプトロン



センサ
または 他のNeural Network

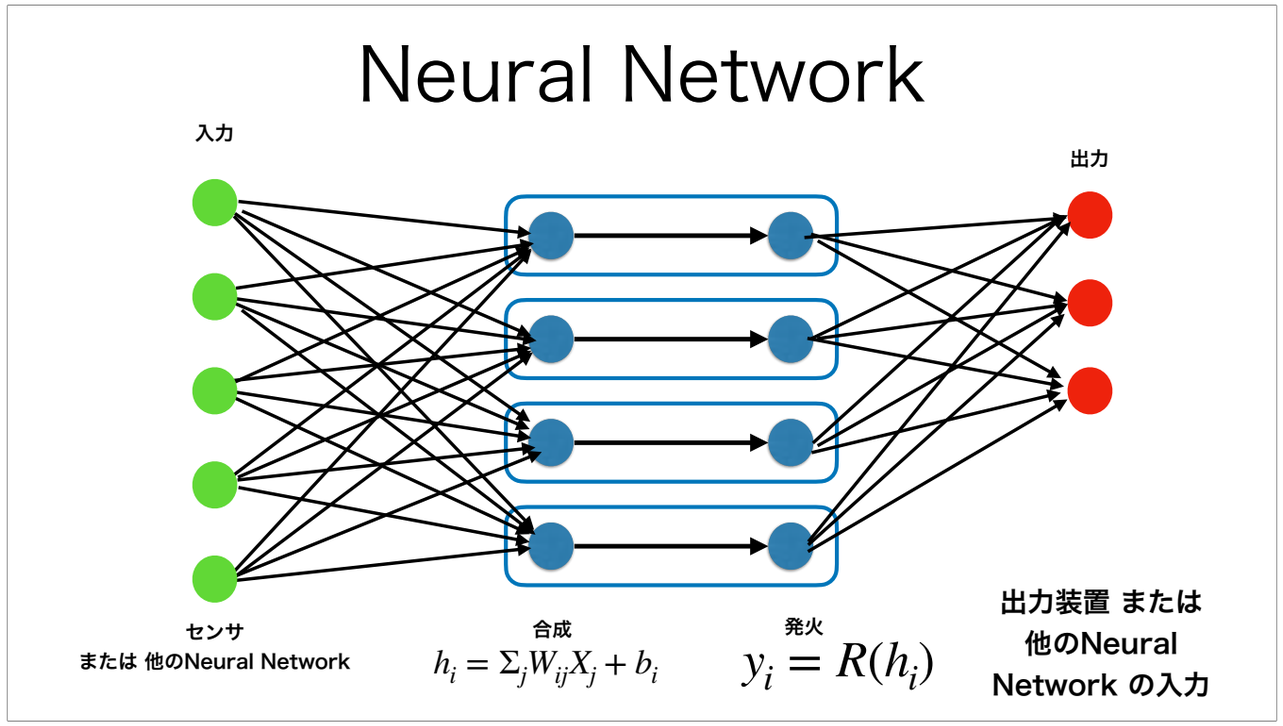
合成

$$h_i = \Sigma_j W_{ij} X_j + b_i$$

発火

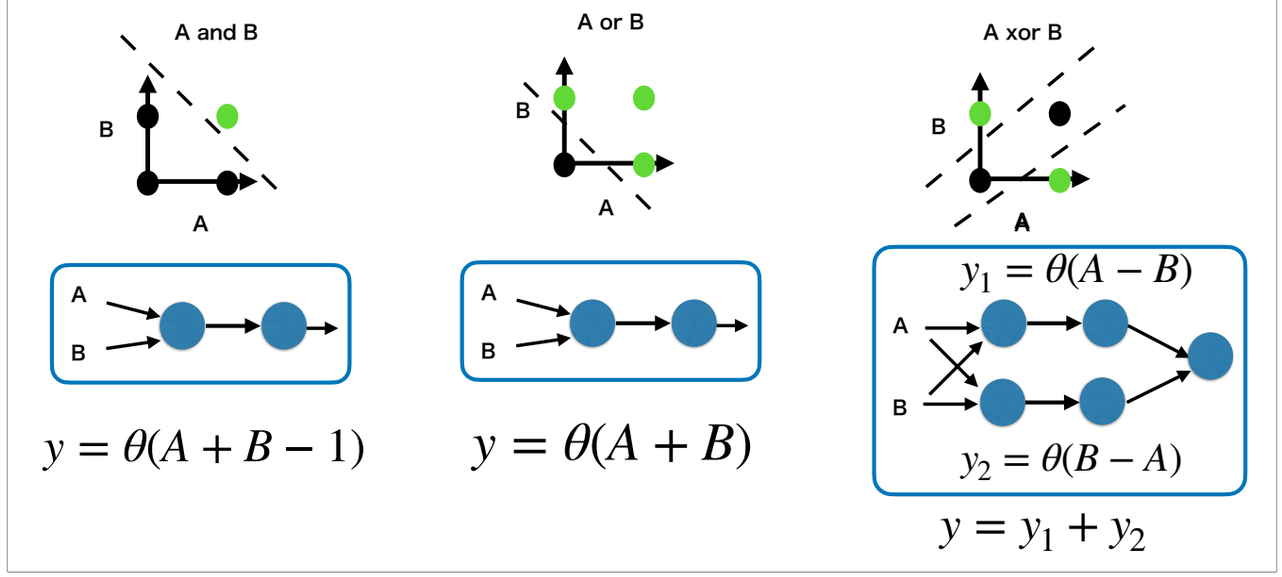
$$y_i = R(h_i)$$

単層の人工ニューロンからなる単層パーセプトロンは人工ニューロンの出力がそのまま出力となっている。
このシステムでは、分類問題として
線形分離可能な問題しか解けないことが
知られています。
また関数近似としても線形の関数以外を近似できないのは明らかです。
Neural Netで人工ニューロンの非線形化の出力を出力段で再度線形結合を取ることで、中間層のニューロン数を増やすことで、任意の関数を近似できるようになります。
2層のパーセプトロンで多数の人工ニューロンをつかうことで、任意の関数を近似できることがわかっています。



Neural ネットワークは、複数の人工シナプスからなる隠れ層と入力層、出力層から構成される。出力層の出力は隠れ層の出力の線形結合(Afine 変換）とする。それぞれの人工シナプスでの入力値に対する線形結合の係数、および人工シナプスから出力値への線形結合の係数は学習データなどに基づいて調整される。

NNによるブール演算



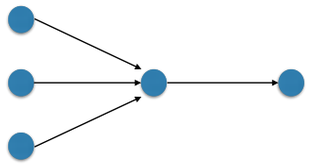
単純パーセプトロンとニューラルネットの違いを理解するために、基本的なブール演算の人工ニューロンを使った表現を考えてみる。

and 演算、or 演算は、単一の直線で分離できるので、単純パーセプトロンで実現できる。が、xor 演算 は2本の分離直線が必要となる。が、ふたつの人工ニューロンを結合したニューラルネットであれば xorを実現できる。
出力が複数の人工ニューロンの出力の線形結合となっていることが意外（？）に重要でこれがなければ、単純パーセプトロンとなってしまい、万能性が実現できない。

A	B	A and B	A or B	y1	y2	y=y1+y2
0	0	0	0	0	0	0
1	0	0	1	1	0	1
0	1	0	1	0	1	1
1	1	1	1	0	0	0

変換ルール

入力層:X 中間層:H 出力層:Y



線形変換　非線形変換

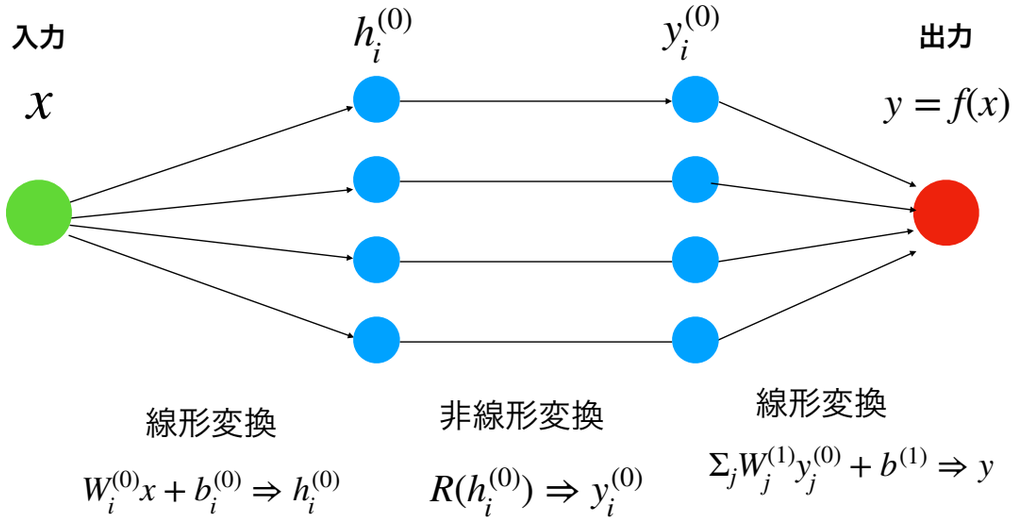
Neural Net　　activation

$$h_i = \Sigma_j W_{ij} X_j + b_i \quad y_i = R(h_i)$$

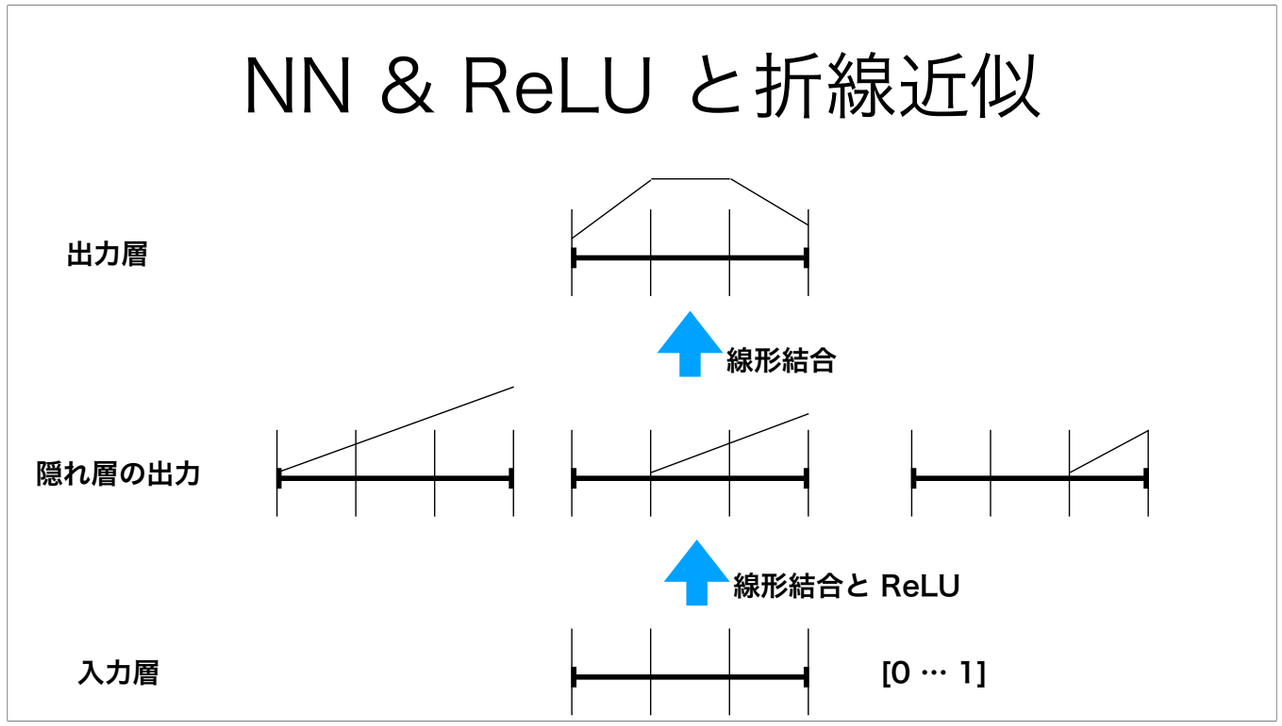
・学習などによって、線形変換のパラメータW, bを調節する。
・非線形関数としては、tanh, logistic, ReLUなど、 step関数likeな関数(sigmoid関数)が普通使われる。
・一つのセルでNANDロジックを構成できる。
　・->多数のcellを組み合わせることで、任意の論理式を表現できることになる。
・複数の層を重ねることで、Deep neural net となる。
・W の和をとる入力の範囲を制限(?)することで、Convolutional neural networks と呼ばれる構成となる。

CNN:Convolutional Neural Netについては、「Together, these properties allow convolutional neural networks to achieve better generalization on vision problems. 」ということでFilterの働く範囲(Receptive field)を限ること。複数のFilterを同時に考えて、それらの出力を次の層で一体として取り扱うこと(Depth)などがある。poolingと呼ばれるnon-linearなsub sampling processも、中間層や出力層でも行う。pooling した結果にfeature detectionのフィルターを適用することができる。

1変数関数の近似:単層NN

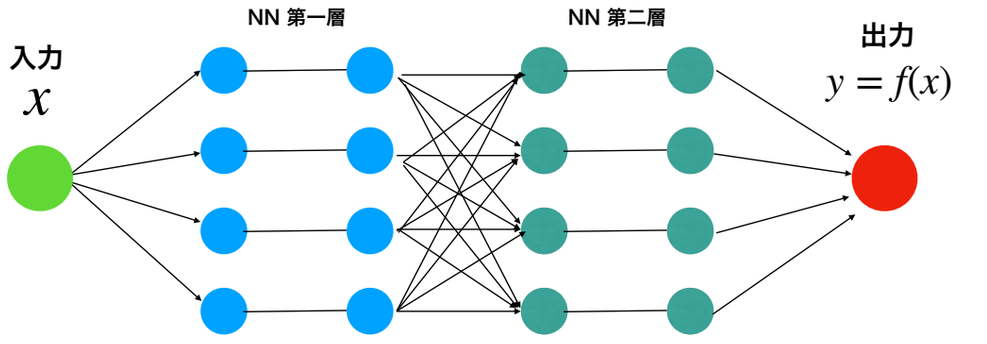


1入力1出力のニューラルネットを考えてみる。
理論的には、この一層のニューラルネットでも、人工ニューロンの数を増やすことで、任意の(連続な？）関数を任意の精度で近似することができる。



ReLUをactivationとして選択したときのこのニューラルネットの働きは、関数を区分的な直線で近似することになっている。ニューロンの数を増やすことは、区分的な直線の区間を短くしていくことになる。うまく区分けをやってやれば、十分な数のニューロン（つまり区間が）あれば折線近似で求められる精度の関数を再現することはできること理解できる。

1変数関数の近似:複層NN



線形変換　非線形変換 線形変換 非線形変換　線形変換

$W_i^{(0)} x + b_i^{(0)} \Rightarrow h_i^{(0)}$　　$\Sigma_j W_{ij}^{(1)} y_j^{(0)} + b_i^{(1)} \Rightarrow h_i^{(1)}$　　$\Sigma_j W_j^{(2)} y_j^{(1)} + b^{(2)} \Rightarrow y$

$R(h_i^{(0)}) \Rightarrow y_i^{(0)}$　　$R(h_i^{(1)}) \Rightarrow y_i^{(1)}$

隠れ層は1層と限る必要はない。図のように複層のニューラルネットを
考えることができる。隠れ層が1でもニューラルネットは万能性を持つので、複数の層にしたところで、表現できる関数のクラスが広がるわけではない。しかしながら、学習により近似ニューラルネットを求める実験では、複層のニューラルネットの方が良い近似関数が求められている。

ニューラルネットの万能性は求められる精度に対して、人工ニューロンの数をいくらでも増やすことができることを仮定している。単層隠れ層と複層隠れ層では、求められる精度に対して必要となるニューロンの数が異なるということが考えられる。

パラメータ数

単純パーセプトロン 入力Nに対して、出力1ごとに：
**　　パラメータ数はN+1**
単層ニューラルネットワーク 入力数N、出力1、隠れ層のニューロン数 Hに対して:
**　N*H +H + 1*H + 1 = (N+2)* H +1**
複層ニューラルネットワーク：　入力数N、出力1、ニューロン数 H2の隠れ層が2層として:
**　(N*H2 + H2) + (H2*H2 + H2) + (H2*1+1) = H2*H2 + (N+3) * H2 + 1**

H1	H2	P1	P2
6	2	19	13
8	3	25	22
12	4	37	33
24	6	73	61
64	12	193	193

単層および複層の隠れ層をもつニューラルネットのパラメータの数を考えてみる。
結果は表のようになる。
単層でニューロン数12のシステムと複層で一層あたりのニューロン数6
(合計は12）のシステムで考えると
単層の場合のパラメータ数 37に対して、複層のパラメータ数は61となる。合計のニューロン数が同じなら、複層のニューラルネットワークの方が表現できる関数の自由度は増えると言えるのではないか。

ターゲット関数

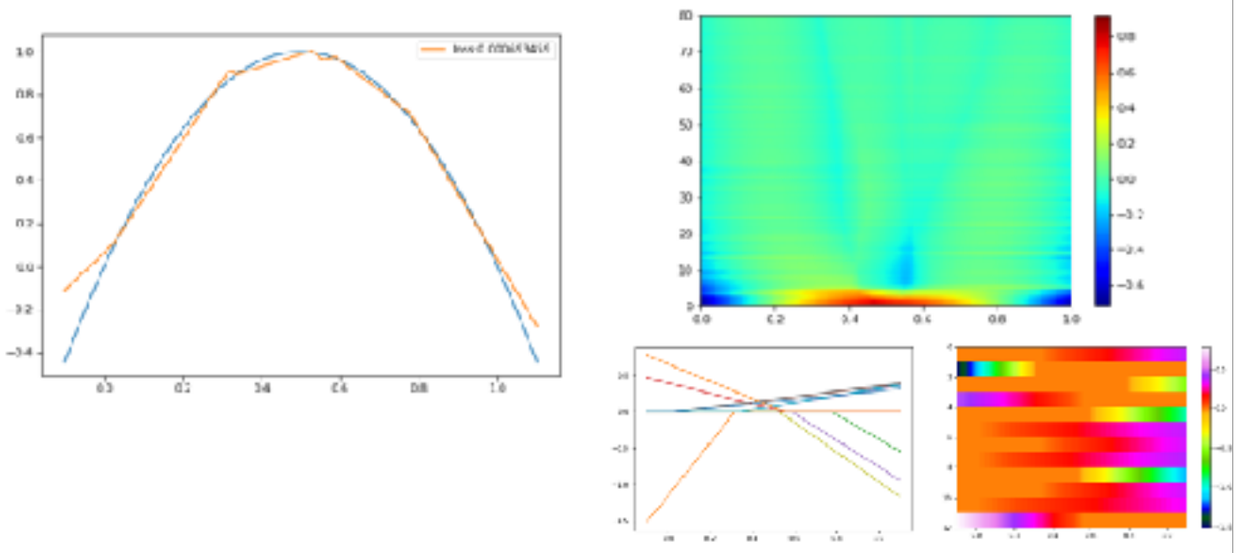
次の1変数スカラー関数を (0,1)の範囲で近似することを考える。

$$y = 4x(1 - x)$$

- 1層および2層のニューラルネットで近似を行い。振る舞いの違いを観察する。
- ニューラルネットのセル数は6~10とした。
- Activation 関数を変更することによる効果も同時に調べる。
  - Activation関数としては、ReLU(6)とSigmoid関数を試してみる。
- ニューラルネットの働きを観測するために、出力の一歩手前の段階の各セルから出力値への 寄与 $x \Rightarrow W_j^{(2)} y_j^{(1)}$ をそれぞれ求め, グラフ表示する。

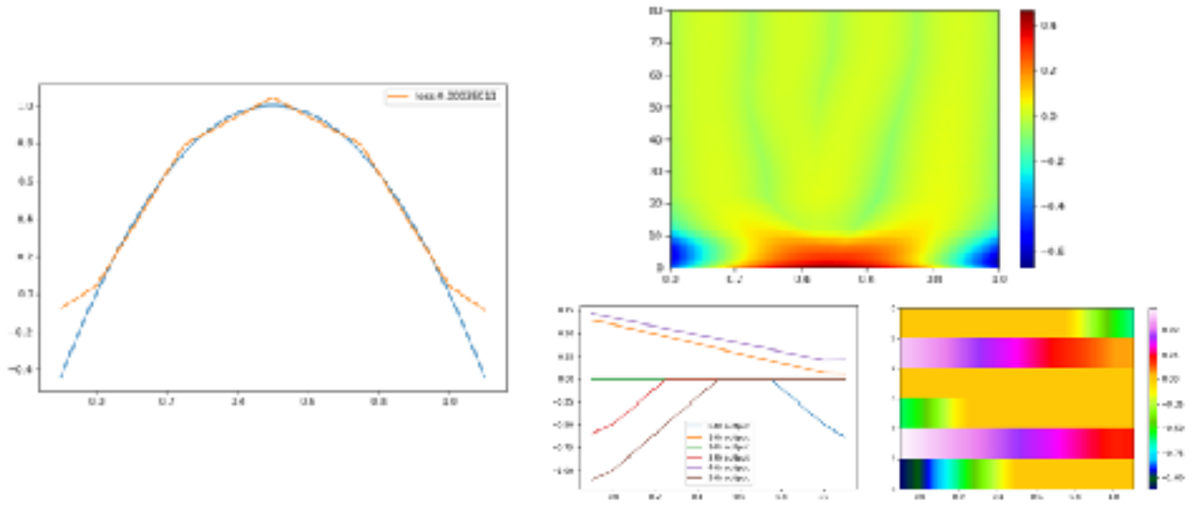
以上のような考察を元に、1入力1出力の関数の近似関数をさまざまなニューラルネットワークを使って求めてみる。

単層ニューラルネットワークによる近似の結果



単層、12 人工シナプスによるニューラルネットを使った近似結果。
Activation関数はReLU.
学習は x:[0,1]の範囲で行っているので、範囲外では近似は保証されない。
右上図では、学習の進展(縦軸）に従って近似値と真値の差がどのように変化していくかをヒートマップで表示している。右下は各人工ニューロンの出力値に対する寄与を分離して折線およびヒートマップで表示している。
これらの寄与を合計したものがニューラルネットの出力となる。

複層ニューラルネットによる近似の結果



複層、6人工シナプス／層によるニューラルネットを使った近似結果.
Activation関数はReLU.
右下図は隠れ層の第2層の人工シナプス毎の出力に対する寄与を表示している。単層の場合と異なり、各人工シナプスの応答は複数のキンクを持つ。
右上図では学習の進展に従って、変数域がほぼ均等に割り振られていく様子が見える。