CS231n Convolutional Neural Networks for Visual Recognition

5.ニューラルネットワーク(Part.1)

序論

脳のはたらきに例えなくても，ニューラルネットワークを導入することは可能です．線形分類の項では，画像に含まれるさまざまな視覚的カテゴリーの判別スコアを，
$s = Wx$(ただし，Wは行列で，xは，画像のすべてのピクセルデータを含む入力列ベクトル)
CIFAR-10の場合，xは[3072x1]の列ベクトルであり，Wは [10x3072] の行列であり，出力スコアは10 個の分類スコアのベクトルとなります．
ニューラルネットワークの例では，線形分類の時とは変わり，$s = W_2\max(0,W_1x)$を計算します．ここで，W_1は，例えば，画像を100次元の中間ベクトルに変換する[100x3072]行列などが考えられます．関数max(0,–)は，要素ごとに適用される非線形性です．
非線形性にはいくつかの選択肢がありますが(以下で検討します)，この選択肢は一般的なもので，単純に0以下のアクティベーションをすべて0にしきい値化します．最後に，行列W_2は[10x100]の大きさになり，クラススコアと解釈される10個の数字が再び得られます．この非線形性は，計算上非常に重要であることに注意してください．もし，この非線形性を残しておけば，2つの行列は1つの行列に折りたたまれ，予測されるクラススコアは再び入力の線形関数となります．非線形性は，私たちが微動だにしないところです．パラメータW_2, W_1は確率的勾配降下法で学習され，その勾配は連鎖律で導かれます(バックプロパゲーションで計算されます)．
3層構造のニューラルネットワークは，数式で表現するならば$s = W_3\max(0,W_2\max(0,W_1x))$のようになり，$W_3, W_2, W_1$のすべてが学習されるパラメータとなります．中間の隠れベクトルの大きさは，ネットワークのハイパーパラメータで，その設定方法は後ほど説明します．では，これらの計算をニューロンやネットワークの観点からどのように解釈するかを考えてみましょう．

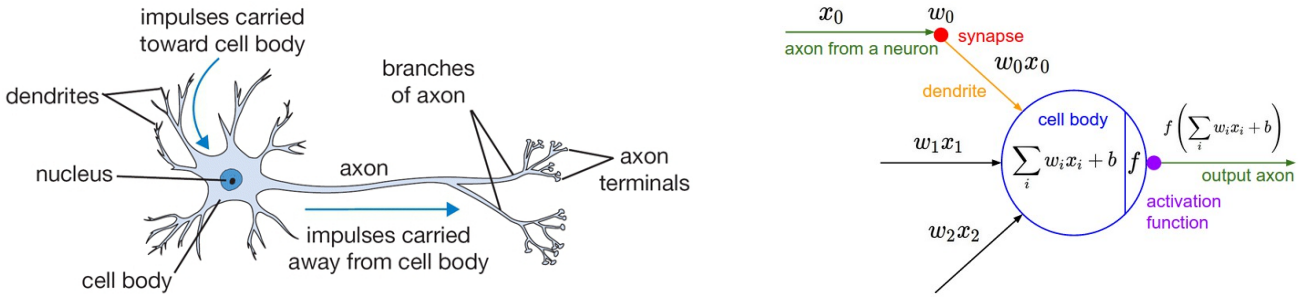
ニューロンのモデリング

ニューラルネットワークの分野は，元々は生物学的な神経システムをモデル化するという目的に触発されていましたが，その後，エンジニアリングの問題となり，機械学習のタスクで良い結果を出すことができるようになりました．それでも，この分野の大部分が触発されてきた生物学的システムについて，非常にシンプルながらもハイレベルな説明で議論を始めます．

生物学的刺激とつながり

脳が構成されている基本的な単位はニューロンです．人間の神経系には約860億個のニューロンが存在し，それらは約10^{14}~10^{15}個のシナプスで接続されています．下の図は，生物学的なニューロンの概略図(左)と，一般的な数学的モデル(右)を示しています．各ニューロンは，樹状突起から入力信号を受け取り，その(単一の)軸索に沿って出力信号を生成します．軸索は最終的に分岐し，シナプスを介して他のニューロンの樹状突起に接続される．ニューロンの計算モデルでは，軸索を伝わる信号(例えばx_0)は，そのシナプスでのシナプス強度(例えばw_0)に基づいて，相手のニューロンの樹状突起と乗算的に相互作用をします．(例えばw_0x_0).

考え方としては，シナプスの強度(重みw)は学習可能であり，あるニューロンが他のニューロンに与える影響の強さ(およびその方向：興奮性(正の重み)または抑制性(負の重み))を制御するという考え方です．基本的なモデルでは，樹状突起が信号を細胞体に伝え，そこですべての信号が合計されます．最終的な和がある閾値を超えると，そのニューロンは発火し，軸索に沿ってスパイクを送ることができます．計算モデルでは，スパイクの正確なタイミングは重要ではなく，発火の頻度のみが情報を伝達すると仮定しています．このレートコードの解釈に基づいて，ニューロンの発火率を，軸索に沿ったスパイクの頻度を表す活性化関数fでモデル化します．これまで，活性化関数の一般的に用いられている関数はシグモイド関数σです．これは，実数値の入力(和の後の信号強度)を受け取り，0と1の間の範囲になるようにそれを正規化するからです．これらの活性化関数の詳細については，このセクションの後半で見ていきます．



生物学的なニューロン(左)とその数学的モデル(右)を図式した

```
class Neuron(object):
  # ...
  def forward(self, inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number
"""
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid
activation function
    return firing_rate
```

単一のニューロンを順方向に伝搬させるコードの例は次のようになります．
言い換えるならば各ニューロンは入力とその重みのドット積を行い，バイアスを加え，非線形性(または活性化関数)，ここではシグモイド$\sigma(x) = 1/(1 + e^{-x})$を適用します．活性化関数の違いについては，このセクションの最後で詳しく説明します．

粗のモデル．ここで強調しておきたいのは，この生物学的ニューロンのモデルは非常に「粗」であるということです．例えば，ニューロンにはさまざまな種類があり，それぞれが異なる特性を持っています．生物学的ニューロンの樹状突起は，複雑な非線形計算を行います．シナプスは単なる1つの重みではなく，複雑な(非線形な)活性化関数で構成されています．多くのシステムでは，出力スパイクの正確なタイミングが重要であることが知られており，レートコード近似が成り立たないことが示唆されています．このように，ニューラルネットワークはさまざま面で簡略化されており本物の脳と類似しているなどと言おうものなら，脳神経科学のバックグラウンドを持つ人たちか

ら呪いの言葉を浴びせられることを覚悟しなければなりません．興味のある方は，このレビュー(pdf)や，より最近のこのレビューをご覧ください．

線形分類器としてのニューロン単体

モデル・ニューロンの前進計算の数学的形式は，皆さんにもなじみがあるかもしれません．線形分類器で見たように，ニューロンは，入力空間の特定の線形領域を「好き」(活性化が1に近い)または「嫌い」(活性化が0に近い)にする能力を持っています．したがって，ニューロンの出力に適切な損失関数を設定すれば，1つのニューロンを線形分類器に変えることができます．

2値のソフトマックス分類器．例えば，$\sigma(\Sigma_i\ w_i x_i + b)$は一方のクラスの確率$P(y_i = 1 \mid x_i; w)$と解釈できます．他のクラスの確率は$P(y_i = 0 \mid x_i; w) = 1 - P(y_i = 1 \mid x_i; w)$となります． となり，これらの合計は1になるはずです．このように解釈すると，交差エントロピー損失を線形分類の項で見たように定式化することができ，これを最適化すると，2値のソフトマックス分類器(ロジスティック回帰とも呼ばれる)が得られます．シグモイド関数は0~1の間に制限されているので，この分類器の予測は，ニューロンの出力が0.5より大きいかどうかに基づいています．

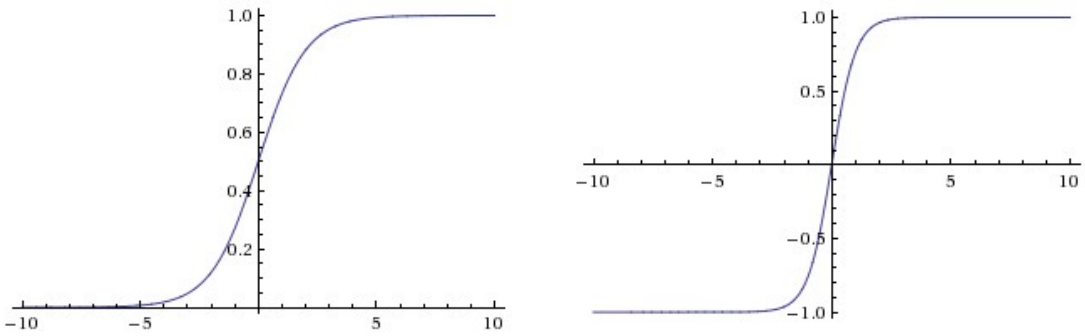
2値のSVM分類器．あるいは，ニューロンの出力にmax-marginヒンジ損失を付けて，2値のSVMになるように訓練することもできます．

正則化解釈．SVMまたはソフトマックスの両方のケースにおける正則化損失は，生物学的には，パラメータ更新のたびにすべてのシナプス重みwをゼロに向かって駆動する効果があるので，緩やかな忘却と解釈することができます．

☞単一のニューロンは，2値の分類器を実装するために使用することができます(例：2値のソフトマックスやバイナリSVM分類器)

よく使われる活性化関数

すべての活性化関数(または非線形関数)は，1つの数値を受け取り，その数値に対して一定の数学的操作を行います．実際に使用する可能性のある活性化関数はいくつかあります．



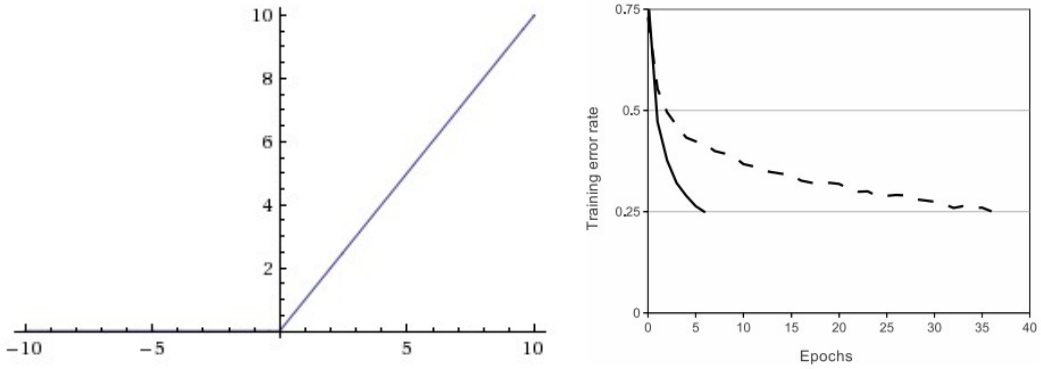
左：シグモイド活性化関数は実数を[0,1]の範囲に圧縮する．右：tanh活性化関数は実数を[-1,1]の範囲に圧縮する．tanh非線形性は，実数を[-1,1]の範囲に圧縮する

シグモイド関数． シグモイド非線形関数は，上図左の$\sigma(x) = 1/(1 + e^{-x})$になります．前節で述べたように，シグモイド関数は，実数を0から1の範囲に「圧縮」するもので，大きな負の数は0に，大きな正の数は1になります．シグモイド関数は，全く発火しない状態(0)から，想定される最大周波数で完全に飽和した発火状態(1)まで，ニューロンの発火率として解釈できるため，歴史的に頻繁に使用されてきました．実際には，シグモイド非線形性は最近では人気がなくなり，ほとんど使用されていません．2つの大きな欠点があるからです．

・シグモイドは勾配を飽和させてしまう．シグモイド・ニューロンの非常に望ましくない特性は，ニューロンの活性化が0または1のどちらかのテールで飽和すると，これらの領域での勾配がほとんどゼロになることです．バックプロパゲーションの際には，この(局所的な)勾配が，目的全体のこのゲートの出力の勾配に掛けられることを思い出してください．したがって，局所的な勾配が非常に小さい場合，実質的に勾配を「殺す」ことになり，ニューロンを介してその重みや再帰的にそのデータにほとんど信号が流れなくなります．さらに，シグモイドニューロンの重みを初期化する際には，飽和を防ぐために細心の注意を払う必要があります．例えば，初期の重みが大きすぎると，ほとんどのニューロンが飽和してしまい，ネットワークはほとんど学習しなくなってしまいます．

・シグモイドの出力はゼロ中心ではない．これは，ニューラルネットワークの後続の処理層のニューロンがゼロ中心ではないデータを受け取ることになるので，望ましくありません．このことは，勾配降下法のダイナミクスに影響を与えます．なぜなら，ニューロンに入力されるデータが常に正である場合．(例えば，$f = w^T x + b$の要素ごとに$x > 0$)．バックプロパゲーションの際の重みwの勾配は，すべて正になるか，すべて負になるかのいずれかになるからです(式全体の勾配に依存するf)．これにより，重みの勾配更新に望ましくないジグザグの動きが生じる可能性があります．しかし，これらの勾配がデータのバッチ全体に渡って加算されると，重みの最終的な更新は可変の符号を持つことになり，この問題が多少緩和されることに注意してください．したがって，この問題は不便ではありますが，上記の飽和活性化の問題に比べれば，それほど深刻な影響はありません．

Tanh．上の画像の右にあるのがtanhの活性化関数です．実数を[-1, 1]の範囲に圧縮します．シグモイド・ニューロンと同様に，その活性化は飽和しますが，シグモイド・ニューロンとは異なり，その出力はゼロ中心です．したがって，実際にはシグモイド活性化関数よりもtanh活性化関数の方が常に好ましい．また，tanhニューロンは単純にスケールされたシグモイドニューロンであり，特に次のことが成り立ちます．$\tanh(x) = 2\sigma(2x) - 1$



左：Rectified Linear Unit (ReLU)活性化関数，x < 0のときは0，x > 0のときは1の傾きを持つ線形となる．Krizhevskyらの論文(pdf)からのプロットで，ReLUユニットではtanhユニットに比べて収束性が6倍向上していることがわかる．

ReLU．Rectified Linear Unitは，ここ数年で非常に人気のあるユニットです．これは，関数$f(x) = \max(0, x)$を計算するものです．言い換えれば，活性化は単純にゼロで閾値化されます(上の左の画像を参照)．ReLUの使用にはいくつかの長所と短所があります．

・(+) sigmoid/tanh関数と比較して，確率的勾配降下法の収束を大幅に加速することがわかった(例えば，Krizhevskyらの論文では6倍)．これは，線形で飽和していない形式によるものだと主張されています．

・(+) 高負荷な演算(指数など)を伴うtanh/sigmoidニューロンに比べ，ReLUは活性化の行列をゼロで閾値処理するだけで実装できる．

・(-) 残念ながら，ReLUユニットは学習中に壊れやすく，「死んで」しまうことがあります．例えば，ReLUニューロンに大きな勾配がかかると，重みが更新され，ニューロンがどのデータポイントでも二度と活性化しなくなることがあります．そうなると，そのユニットを流れるグラジエントは，その時点から永遠にゼロになります．つまり，ReLUユニットはデータマニホールドから外れてしまうため，トレーニング中に不可逆的に死んでしまう可能性があるのです．例えば，学習率を高く設定しすぎると，ネットワークの40%が「死んだ」状態(トレーニングデータセット全体でアクティブにならないニューロン)になってしまうことがあります．学習率を適切に設定することで，この問題は少なくなります．
Leaky ReLU．これは，「瀕死のReLU」問題を解決する一つの試みです．x < 0の時に関数がゼロになる代わりに，Leaky ReLUは小さな負の傾き(0.01程度)を持ちます．つまりこの関数は，$f(x) = \mathbb{1}(x < 0)(\alpha x) + \mathbb{1}(x >= 0)(x)$と計算されます．ここで$\alpha$は小さな定数です．この形式の活性化関数で成功したという報告もありますが，結果は常に一定ではありません．また，Kaiming He et al., 2015によるDelving Deep into Rectifiersで紹介されているPReLUニューロンに見られるように，負の領域の傾きを各ニューロンのパラメータにすることもできますが，タスク間でのメリットの一貫性は現在のところ不明です．

Maxout．重みとデータの間のドット積に非線形性が適用される関数形式$f(w^T x + b)$を持たない他のタイプのユニットも提案されています．比較的よく知られているのは，ReLUとそのリーキー・バージョンを一般化したMaxout・ニューロン(Goodfellowらが最近発表)です．Maxout・ニューロンは，関数 $\max(w_1^T x + b_1, w_2^T x + b_2)$を計算します．ReLUもLeaky ReLUも，この形式の特殊なケースであることに注意してください(例えば，ReLUの場合は$w_1, b_1 = 0$). したがって，Maxoutニューロンは，ReLUユニットのすべての利点(線形動作領域，飽和なし)を享受し，その欠点(ReLUが瀕死状態になる)はありません．しかし，ReLUニューロンとは異なり，1つのニューロンごとにパラメータの数が2倍になるため，パラメータの総数が多くなります．

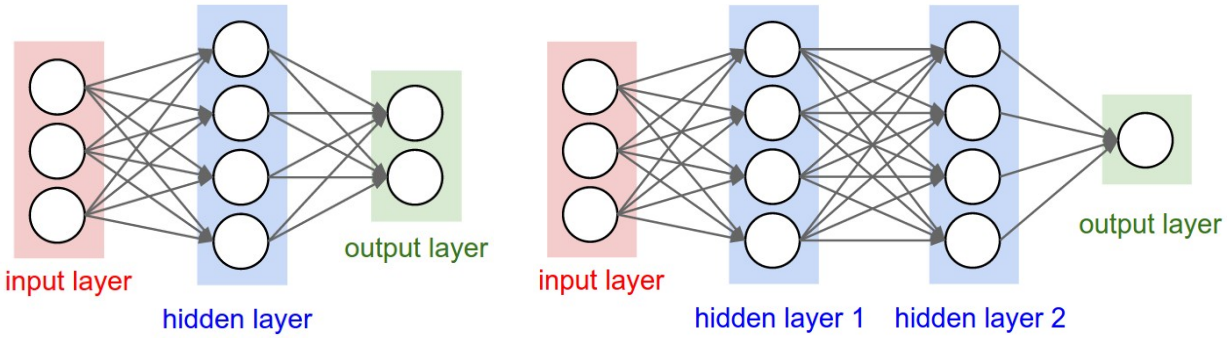
以上で，最も一般的なニューロンの種類とその活性化関数についての説明を終わります．最後になりましたが，基本的に問題はないものの，同じネットワーク内で異なるタイプのニューロンを混在させることは非常に稀です．

TLDR: 「どのタイプのニューロンを使えばいいのか」ReLUの非線形性を利用し，学習率に注意し，ネットワーク内の「死んだ」ユニットの割合を監視するとよいでしょう．これが気になる場合は，Leaky ReLUやMaxoutを試しましょう，但しsigmoidは絶対に使わないでください．tanhも試してみると良いと思いますが，ReLU/Maxoutよりも良好な動作は期待できないと思われます．

ニューラルネットワークのアーキテクチャ

階層ごとの構成

グラフの中のニューロンとしてのニューラルネットワーク ニューラルネットワークは，非周期的なグラフで接続されたニューロンの集合体としてモデル化されます．つまり，あるニューロンの出力は，他のニューロンの入力になります．循環は，ネットワークのフォワードパスにおける無限ループを意味するため，許可されていません．ニューラルネットワークのモデルは，接続されたニューロンの不定形な塊ではなく，ニューロンの明確な層で構成されていることが多い．通常のニューラルネットワークでは，最も一般的な層のタイプは，隣接する2つの層間のニューロンが完全にペアワイズで接続されている完全接続層ですが，1つの層内のニューロンは接続を共有しません．以下に，完全連結層を重ねたニューラルネットワークのトポロジーの例を2つ示します．



左：2層のニューラルネットワーク(4つのニューロン(ユニット)からなる隠れ層と2つのニューロンからなる出力層)と3つの入力．右図3層のニューラルネットワーク(4個のニューロンからなる2つの隠れ層と，1つの出力層)．いずれの場合も，隣り合うニューロン間の接続(シナプス)は存在するが，層内でのループ接続は存在しない

命名規則．N層ニューラルネットワークと言った場合，入力層はカウントしないことに注意してください．したがって，単層ニューラルネットワークとは，隠れ層がない(入力がそのまま出力にマッピングされている)ネットワークを表します．その意味で，ロジスティック回帰やSVMは単層ニューラルネットワークの特殊なケースであると言われることがあります．また，これらのネットワークは，「Artificial Neural Networks」(ANN)や「Multi-Layer Perceptrons」(MLP)と呼ばれることもあります．多くの人は，ニューラルネットワークと本物の脳との類似させることを好まず，ニューロンではなくユニットと呼ぶことを好みます．

出力層．出力層のニューロンは，ニューラルネットワークのすべての層とは異なり，活性化関数を持たないことがほとんどです(あるいは，線形同一活性化関数を持つと考えることもできます)．これは，最後の出力層は，通常，任意の実数値であるクラススコア(分類の場合など)や，何らかの実数値の目標値(回帰の場合など)を表すと考えられているからです．

ニューラルネットワークのサイズ．ニューラルネットワークのサイズを測るのに，一般的にはニューロンの数と，より一般的なパラメータの数の2つの指標があります．上の写真の2つのネットワークの例を見てみましょう．

- 1つ目のネットワーク(左)は，4 + 2 = 6個のニューロン(入力を除く)，[3 x 4] + [4 x 2] = 20個の重み，4 + 2 = 6個のバイアスを持ち，合計26個の学習可能なパラメータを持っています．

- 2番目のネットワーク(右)は，4 + 4 + 1 = 9個のニューロンを持ち，[3 x 4] + [4 x 4] + [4 x 1] = 12 + 16 + 4 = 32個の重みと4 + 4 + 1 = 9個のバイアスを持ち，合計41個の学習可能なパラメータを持っています.

現代の畳み込みネットワークは，1億個ものパラメータを持ち，通常は約10〜20層で構成されています(これが深層学習です). しかし，これから説明するように，パラメータを共有することで，有効な接続数は大幅に増えます. これについては，畳み込みニューラルネットワークのモジュールで詳しく説明します.

フィードフォワード計算の例

活性化関数に織り込まれた行列の乗算の繰り返し. ニューラルネットワークが層構造になっている主な理由の1つは，この構造によって，行列ベクトル演算を用いてニューラルネットワークを評価することが非常にシンプルかつ効率的になるからです. 上の図の3層構造のニューラルネットワークの例では，入力は[3x1]ベクトルになります. 各層のすべての接続強度は，1つの行列に格納できます. 例えば，第1隠れ層の重み`W1`は[4x3]の大きさで，すべてのユニットのバイアスは[4x1]の大きさのベクトル`b1`になります. ここでは，すべてのニューロンがその重みを`W1`の行に持っているので，行列ベクトル乗算`np.dot(W1,x)`は，その層のすべてのニューロンの活性化を評価することになります. 同様に，`W2`は第2隠れ層の接続を格納する[4x4]行列，`W3`は最後の(出力)層の[1x4]行列となります. この3層ニューラルネットワークの完全なフォワードパスは，単に3つの行列の乗算と活性化関数の適用を織り交ぜたものです.

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations
(4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

上記のコードでは，`W1,W2,W3,b1,b2,b3`がネットワークの学習可能なパラメータです. また，1つの入力列ベクトルではなく，変数`x`に学習データのバッチ全体を格納することができ(各入力例は`x`の列となります)，すべての例が効率的に並列評価されることにも注目してください. 最後のニューラルネットワーク層は，通常，活性化関数を持たないことに注意してください(例えば，分類の設定では，(実数値の)クラススコアを表します).

☞完全連結層のフォワードパスは，1回の行列乗算に続いて，バイアスオフセットと活性化関数を加えたものです.

表現力

完全に接続された層を持つニューラルネットワークを見る一つの方法は，ネットワークの重みでパラメータ化された関数のファミリーを定義することです. ここで，自然な疑問が生まれます. この関数群の表現力はどの程度なのか？特に，ニューラルネットワークでモデル化できない機能はあるのか？少なくとも1つの隠れ層を持つニューラルネットワークは，普遍的な近似器であることがわかりました.

つまり，任意の連続関数$f(x)$とある$\epsilon > 0$が与えられると，$\forall x, | f(x) - g(x) | < \epsilon >$という非線形性を持つ隠れ層を1つ持つニューラルネットワークg(x)が存在することが示されます(1989年のシグモイド関数の重ね合わせによる近似(pdf)や，Michael Nielsen氏による直感的な説明など)． つまり，ニューラルネットワークは任意の連続関数を近似することができるのです．

隠れた層が1つあればどんな関数でも近似できるのなら，なぜ層を増やしたり，より深くしたりするのでしょうか？その答えは，2層のニューラルネットワークが万能の近似器であるという事実は，数学的に考慮すれば魅力的ですが，実際には比較的弱く，役に立たないものだからです．1次元では，a, b, cをパラメータベクトルとする「インジケータバンプの和」関数$g(x) = \Sigma_i \, c_i \mathbb{1}(a_i < x < b_i)$も普遍的な近似式ですが，機械学習でこの関数形式を使おうと言う人はいないでしょう．ニューラルネットワークが実際にうまく機能しているのは，実際に遭遇するデータの統計的特性によく合う，きれいで滑らかな関数をコンパクトに表現しているからであり，また，最適化アルゴリズム(例えば，勾配降下法)を使って学習するのが簡単だからです．同様に，単一の隠れた層を持つネットワークよりも，複数の隠れた層を持つ深いネットワークの方が，表現力が同じであるにもかかわらず，うまく機能するという事実は，経験的に観察されています．

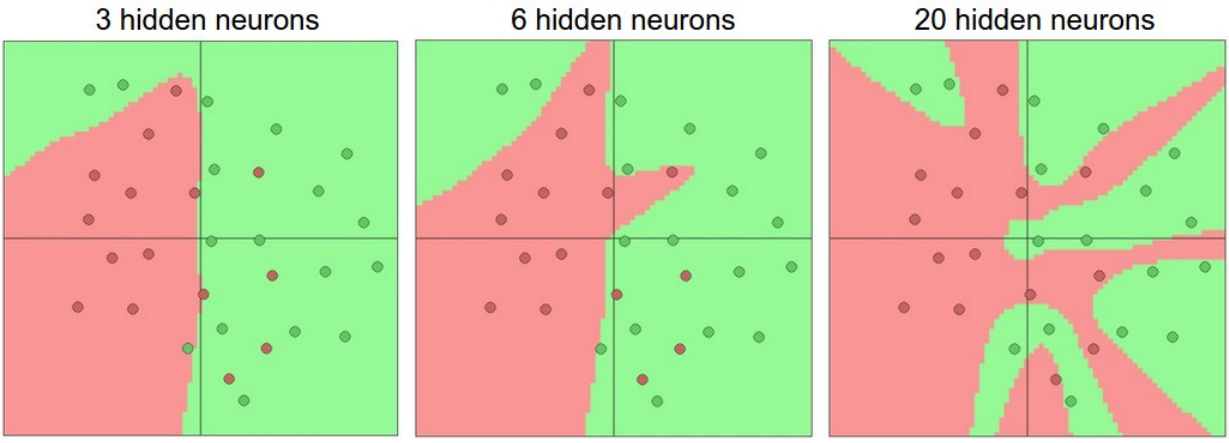
余談ですが，実際には，3層のニューラルネットワークが2層のニューラルネットワークよりも優れていることはよくありますが，さらに深く(4,5,6層)しても，それ以上の効果はほとんどありません．これは，深さが認識システムにとって非常に重要な要素であることがわかっている畳み込みネットワークとは対照的です(例えば，学習可能な10層のオーダー)．これは，画像には階層的な構造が含まれており(例えば，顔は目で構成され，目はエッジで構成されている)，このデータ領域では複数のレイヤーによる処理が直感的に理解できるからです．

もちろん，その全貌はもっと複雑で，最近の研究テーマでもあります．これらのトピックに興味がある方は，以下の文献をお勧めします．

- Deep Learning book by Bengio, Goodfellow, Courville,特に第6.4章
- 深層ネットは本当に深い必要があるのか？
- FitNets: Hints for Thin Deep Nets.

レイヤーの数とサイズの設定

現実的な問題に直面したとき，どのようにして使用するアーキテクチャを決定するのでしょうか？隠れ層を使わないほうがいいのか？隠れ層は1つ？隠れ層を2つにするか？各層の大きさはどれくらいにすべきか？まず，ニューラルネットワークのサイズと層の数を増やすと，ネットワークの容量が増えることに注意してください．つまり，ニューロンが協力してさまざまな機能を表現できるようになるため，表現可能な機能の空間が大きくなるのです．例えば，2次元のバイナリ分類問題があったとします．それぞれがある程度の大きさの隠れ層を持つ3つの別々のニューラルネットワークを学習し，以下のような分類器を得ることができます．



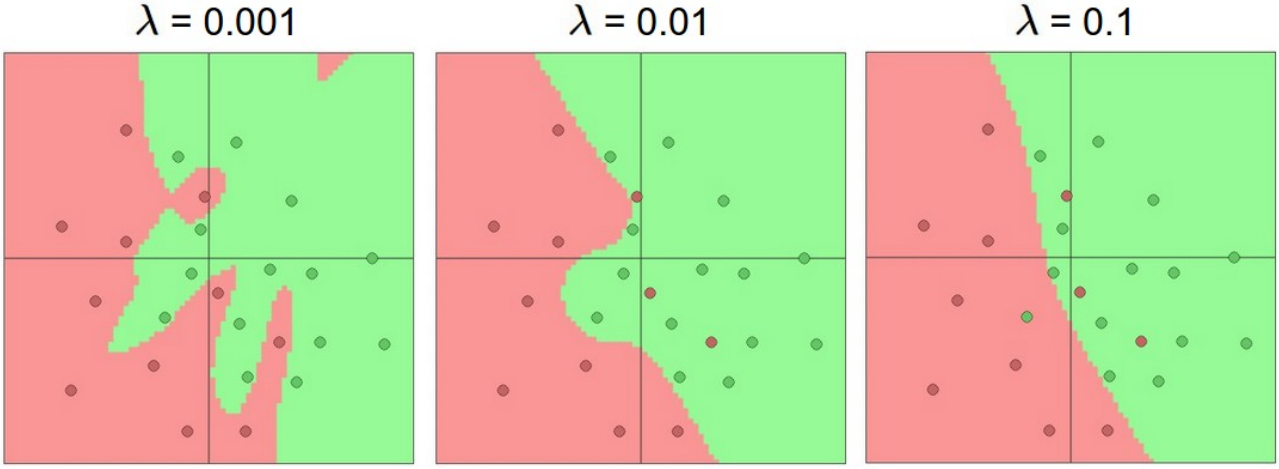
より大きなニューラルネットワークは，より複雑な機能を表現することができます．データはクラスごとに色分けされた円で表示され，その下には学習されたニューラルネットワークによる決定領域が表示されています．このConvNetsJSデモでは，これらの例を使って遊ぶことができます．

上の図では，ニューロン数の多いニューラルネットワークの方が，より複雑な関数を表現できることがわかります．しかし，これは(より複雑なデータの分類を学べるという)利点でもあり，(学習データのオーバーフィットが起こりやすいという)欠点でもあります．オーバーフィットとは，高い能力を持つモデルが，(仮定された)基本的な関係ではなく，データのノイズにフィットしてしまうことです．例えば，20個の隠れニューロンを持つモデルは，すべてのトレーニングデータにフィットしますが，その代償として，空間が赤と緑の多くの不連続な決定領域に分割されます．隠れニューロンが3個のモデルは，データを大まかに分類する程度の表現力しかありません．このモデルでは，データを2つの塊としてモデル化し，緑のクラスター内にある少数の赤の点を外れ値(ノイズ)として解釈します．実際には，これによってテストセットでの一般化が向上する可能性があります．

以上の議論を踏まえると，過学習を防ぐほどデータが複雑でない場合は，より小さなニューラルネットワークが好ましいと思われます．しかし，これは間違っています．ニューラルネットワークで過学習を防ぐための好ましい方法は他にもたくさんありますが，それについては後ほど説明します(L2正則化，ドロップアウト，入力ノイズなど)．実際には，ニューロンの数ではなく，これらの方法を使って過学習を制御する方が常に良いのです．

その理由は，小さいネットワークは勾配降下法のような局所的な手法では学習しにくいからです．損失関数に局所的な最小値が少ないことは明らかですが，その最小値の多くは収束しやすく，しかも悪い(つまり損失が大きい)ことがわかります．逆に，大きなニューラルネットワークは，かなり多くのローカルミニマムを含んでいますが，これらのミニマムは，実際の損失という点では，はるかに良いことがわかります．ニューラルネットワークは非凸であるため，これらの特性を数学的に研究することは困難ですが，最近の論文The Loss Surfaces of Multilayer Networksなどでは，これらの目的関数を理解する試みがなされています．実際には，小さなネットワークを訓練した場合，最終的な損失はかなりの量の分散を示すことがわかります．一方，大規模なネットワークを学習すると，さまざまなソリューションが見つかるようになりますが，最終的に達成される損失の分散ははるかに小さくなります．言い換えれば，すべての解決策は同じくらい良いものであり，ランダムな初期化の運に頼ることは少ないのです．

繰り返しになりますが，正則化の強さは，ニューラルネットワークの過学習を制御するのに適した方法です．3つの異なる設定で得られる結果を見てみましょう．



正則化強度の効果．上の各ニューラルネットワークは20個の隠れニューロンを持っていますが，正則化の強さを変えることで，最終的な決定領域がより高い正則化で滑らかになります．これらの例は，ConvNetsJSのデモで体験することができます．

ここでのポイントは，過学習を恐れて，より小さなネットワークを使うべきではないということです．代わりに，計算機の予算が許す限り大きなニューラルネットワークを使用し，他の正則化技術を使用して過学習を抑制するべきです．

まとめ

生体ニューロンの非常に粗いモデルを紹介しました．
実際に使われる活性化関数にはいくつかの種類があり，ReLUが最も一般的な選択肢であることを説明しました．
ニューロンは，隣接する層のニューロンは完全にペアワイズ接続されているが，層内のニューロンは接続されていないFully-Connected層で接続されているニューラルネットワークを紹介しました．この層構造により，活性化関数の適用に織り込まれた行列の乗算に基づいて，非常に効率的なニューラルネットワークの評価が可能であることを確認しました．
ニューラルネットワークが普遍的な関数近似器であることを確認しましたが，この特性はニューラルネットワークがどこでも使えることとはあまり関係がないということも議論しました．ニューラルネットワークが使われているのは，実際に出てくる関数の関数形について，ある種の「正しい」仮定をしているからです．
大きなネットワークは小さなネットワークよりも常にうまく機能するが，モデルの容量が大きいため，より強力な正則化(より高い重みの減衰など)で適切に対処しなければ，オーバーフィットする可能性があるという事実を議論した．後のセクションでは，より多くの正則化(特にドロップアウト)を見ていきます．