November 7, 2019

物理研究者向けのニューラルネットワーク入門 (基本編)

富谷 昭夫 [1,*]

[1]*RIKEN/BNL Research center, Brookhaven National Laboratory, Upton, NY, 11973, USA*

Abstract

この講義ノートは、物理学を勉強/研究している者に向けたニューラルネットワークの解説である。こちらのノートは、ニューラルネットの基本的な事項を学ぶことを目的とする。★のついているセクションは、ハンズオンには影響しないため進捗によってはスキップする。

*akio.tomiya@riken.jp

Contents

参考文献

1. 深層学習、 岡谷貴之 、講談社: 網羅的
2. 今度こそわかる深層学習、瀧雅人、講談社: 網羅的、物理学者フレンドリー
3. ディープラーニングと物理学、田中章詞他、講談社: 物理学者フレンドリー
4. Chainer v2 による実践深層学習、新納浩幸、オーム社: 手を動かせる
5. IT エンジニアのための機械学習理論入門、中井悦司、技術評論社: 平易な言葉で説明してある

以上は直接参考にした。このほか[1]、

1. 物理学者、機械学習を使う、橋本 (他)、朝倉書店: 広い分野への応用
2. 深層学習、人工知能学会、近代科学社: 高難易度だが他に載っていない内容がある
3. パターン認識と機械学習、C. M. Bishop[2]、丸善出版: ペスキン的な立ち位置の教科書

も適宜参考にした。

この講義で分かるようになること、扱うこと

1. ニューラルネットワークのフレームワーク
2. ニューラルネットワークの構造
3. 誤差逆伝播 (Back prop) を使った学習

[1] 「ベイズ深層学習」(須山敦志) や「ガウス過程と機械学習」(持橋大地, 大羽成征) 「ゼロから作る Deep Learning」(斎藤 康毅) も面白かったので一読されたい。
[2] エディンバラ大学出身で素粒子論 (場の理論に関する研究) で博士号を取得。

この講義で扱わないこと

時間の関係上、下記は重要であると思うが扱わない。

1. ニューラルネットワークの初期化
2. 畳み込みニューラルネットワーク
3. 順伝播ニューラルネットワーク以外の機械学習フレームワーク
   (格子 QCD へ決定木をつかった重要な応用や配位生成の話がある)
4. 平均場近似やスピングラス描像などの理論的な側面

1 時間の講義でハンズオンに向けた完全な理解を目指すため、簡単すぎるかもしれない。そういう者は、上に挙げた参考文献を参照すると良い。

I. 導入

A. 実験、帰納的な類推

時間の関数として物理量が得られたとする。この時に、ある時刻での物理量を予測できるだろうか。

例として自由落下を考えてみる。落下時間 t の関数として落下距離 y が得られたとして、全部で N_D 個のデータが集まったとする。これを下記の様に書く。

$$\mathcal{D} = \{(t_1, y_1),\ (t_2, y_2),\ \cdots,\ (t_d, y_d),\ \cdots,\ (t_{N_D}, y_{N_D})\} \tag{1}$$

するとフィットできそう。一旦フィット曲線がきまると、落下時間 t を指定すると落下距離が予測できる。この様に、過去のデータを用いると予測が可能となる[3]。

背後の物理を知らない時は、どうするか。アンザッツをおいてデータとの差を見て、差を最小化するのが良さそうである (最小 2 乗法)。アンザッツは、1 次式、2 次式といろいろ試すことになる。

まとめると以下の手続きを踏むことになる。

$$\text{データ収集} \to \text{“フィット”} \to \text{予測} \tag{2}$$

大雑把にいうと機械学習はこれを行う分野とも言える。

後のために、すこし一般化をしておく。独立変数と従属変数ともにベクトルであるようなデータを考えても良い $\mathcal{D} = \{(\vec{x}_1, \vec{y}_1),\ (\vec{x}_2, \vec{y}_2),\ \cdots,\ (\vec{x}_d, \vec{y}_d),\ \cdots,\ (\vec{x}_{N_D}, \vec{y}_{N_D})\}$ 、ただし

$$\vec{x}_d \in \mathbb{R}^n,\ \ \vec{y}_d \in \mathbb{R}^t. \tag{3}$$

とする。これもフィットすると、しらない点を予測できる。

少し視点を情報理論に合わせておく。データは今、サンプルされたものであるが、これがある確率分布に従う確率変数だと思い直す。

$$P(\vec{X}, \vec{Y}) \tag{4}$$

実際のデータはこちらからサンプルされているとみなす。

機械学習の “定義”: 得られたデータに基づいて、新たに予測などを行う枠組み。

---

[3] それが実際に実現するかというのは大きな問題であり、機械学習で言うところの汎化 (generalization) につながるがここでは触れない。

B. 機械学習の分類

大きく分けて以下の 3 つ：

1. 教師つき学習: 正解ラベルが与えられたらそれを出力する関数を作る。決定論的。
   教師 = (画像、ラベル) の組で新規の画像に対してラベルを当てる。[SU(3) の位相電荷 1909.06238].

2. 教師なし学習: 教師ラベルなしで学習する。
   例えば画像生成や分類など。[配位生成 1904.12072 、1712.03893 など]

3. 強化学習: 環境とエージェントを用意して間接的に問題をとく。
   例えば、ブロック崩しで得点を報酬にして高得点を狙うものやアルファ碁。[カラビヤウ探索 1903.11616]。

と分かれる。アーキテクチャでの分類ではなくてやることでの分類。物理への応用の最新事項は、Deep learning and physics 2019 のホームページを見ると良い。また「物理学者、機械学習を使う」も参照。

C. ニューラルネットワークとは

ニューラルネットワークとは動物ニューロンの数理モデルとして発明されたベクトル値関数である。内部の変数が多いため高いフィット性能 (表現性能, expressibility が高い) を持つ。しかも未解決問題ではあるが、なぜか比較的オーバーフィットしにくいことが知られている。

本講義では、深層ニューラルネットワークを用いた分類に限って説明する。

II. ニューラルネットワークの概論

A. ニューラルネットワーク

1. ニューラルネットの構成要素

ここでは、ニューラルネットワークを構成していくことで解説する。構成要素は下記の 2 つである。

1. 線形変換[4]

2. 活性化関数 (非線形変換)

であるので以下で見ていく。

まず線形変換を見る。$\vec{x} \in \mathbb{R}^m$ として

$$\vec{v} = W\vec{x} + \vec{b} \in \mathbb{R}^n \tag{5}$$

を考える。ここで W は $n \times m$ 行列である。$\mathbb{R}^m \to \mathbb{R}^n$ の 線形写像をつくれる。もし $\vec{v}$ と $\vec{x}$

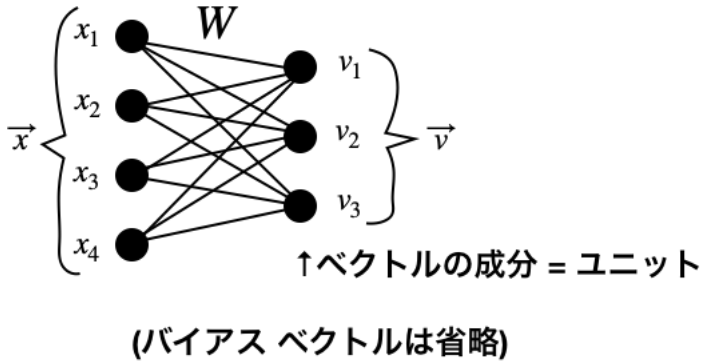

Figure 1: 線形変換の概念図

が線形な関係であればフィットできる。一般のデータではそれは期待できない。

天下りではあるが次で述べるように非線形変換を導入する[5]。

$$\vec{z} = \sigma(\vec{v}) \in \mathbb{R}^n \tag{6}$$

$\sigma(\vec{v})$ はエレメントごとに作用する非線形関数

$$\sigma(\vec{v}) \overset{\text{def}}{\Leftrightarrow} \begin{pmatrix} \sigma(v_1) \\ \sigma(v_2) \\ \vdots \\ \sigma(v_n) \end{pmatrix} \tag{7}$$

σ は典型的には、シグモイド (sigmoid) や ReLu (Rectified linear unit) で以下で与えられる (図 2)

$$\sigma_{\text{sigmoid}}(x) = \frac{1}{1+e^{-x}} \tag{8}$$

[4] 正確にはアフィン変換。アフィン変換 = 一次変換と平行移動をあわせたもの。
[5] もともとは、神経の電位の非線形性を表現するために導入された。

$$\sigma_{\mathrm{ReLu}}(x) = \max(0, x) \tag{9}$$

シグモイドは、フェルミ分布関数でありまた、神経の活性化に類似しているため歴史的によく使われていたが近年は ReLu がよく使われている。

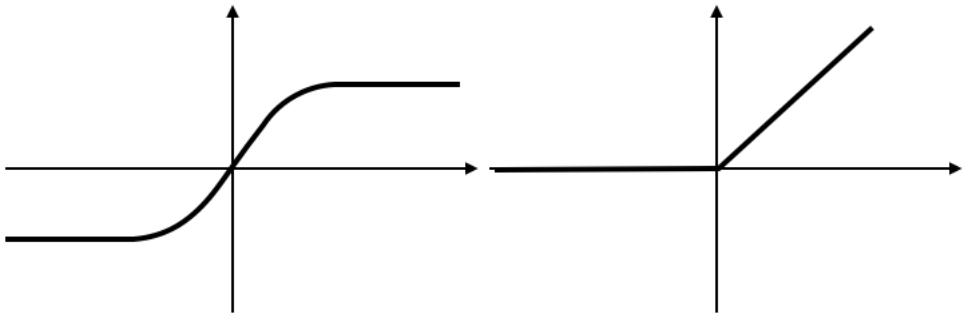

Figure 2: 活性化関数の図。左がシグモイド (sigmoid)、右が ReLu(Rectified linear unit) である。

2. ニューラルネットとはパラメータ付きの関数

以下では、簡単のためパラメータを $\theta = \{w_{ij}, b_i\}_{i,j=1,2,\cdots}$ と書く。ここで $W = [w_{ij}], \vec{b} = b_i$ である。

ニューラルネットワークは、線形変換と非線形関数の入れ子であるような関数である。一番簡単なニューラルネットワークは、3 層ニューラルネットワーク

$$\vec{f}_\theta(\vec{x}_d) = \sigma^{(2)}(W^{(2)}\sigma^{(1)}(W^{(1)}\vec{x}_d + \vec{b}^{(1)}) + \vec{b}^{(2)}) \tag{10}$$

またディープラーニングとは、

$$\vec{f}_\theta(\vec{x}_d) = \sigma^{(N)}\big(W^{(N)}\sigma^{(N-1)}\big(W^{(N-1)}\Big(\cdots\Big(\sigma^{(1)}(W^{(1)}\vec{x}_d + \vec{b}^{(1)})\Big)\cdots\Big) + \vec{b}^{(N-1)}\big) + \vec{b}^{(N)}\big) \tag{11}$$

という深層ニューラルネットでの学習を指す (図 3)。このようなニューラルネットワークは、全結合 (Fully connected/dense) ニューラルネットワークと呼ばれる[6]。第 1 層で情報が入力される層を入力層、最終層である情報が出力される層を出力層とよぶ。またその中間を隠れ層 (hidden layer) と呼ぶ。

ニューラルネットワークは図 4 の様にも書かれ、右側の表記では、線形変換が 1 本の直線で書かれる。

[6] これ以外に、畳込みニューラルネットワークがあるが今回は説明しない。

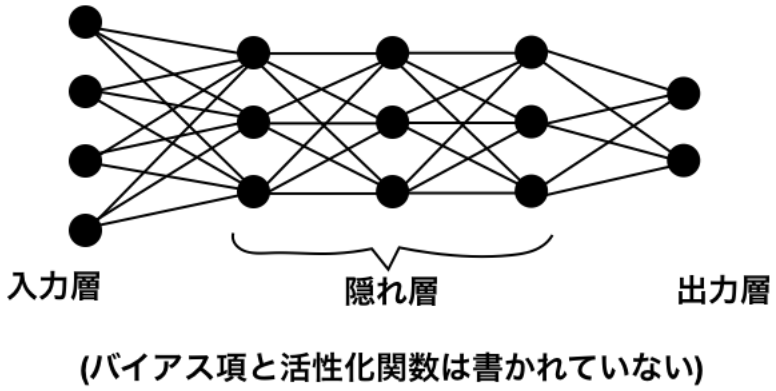

Figure 3:　ニューラルネットの概念図

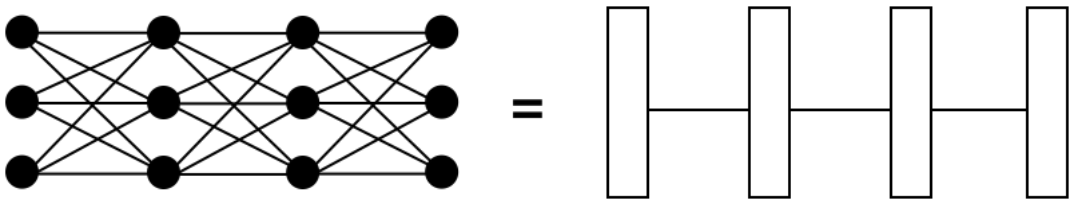

Figure 4:　ニューラルネットの様々な表示。右側では線形変換が 1 本の線で書かれている。

B.　ニューラルネットワークの入出力と学習

ここでは特に全結合ニューラルネットワーク (fully connected neural network, dense neural network) をやる。

1.　ニューラルネットワークのためのデータの扱い

具体的に Fisher のあやめを取り上げて見てみよう。Fisher のあやめとは、統計学者の Fisher の集めたアヤメのデータで花びらの幅とながさ、がくの幅と長さの 4 次元のデータと 3 種のアヤメのラベルを持つ。全データ数は 150 個である。

a.　入力　入力は

$$\vec{x} = \begin{pmatrix} \text{花びらの幅} \\ \text{花びらの長さ} \\ \text{がくの幅} \\ \text{がくの長さ} \end{pmatrix} \in \mathbb{R}^4. \tag{12}$$

といった 4 次元ベクトルである。

2. ラベル付け

ニューラルネットワークにおいて、出力も加工しておく必要がある。今の場合、分類がセトナ (setosa)、バーシクル (versicolor)、 バージニカ (virginica) の 3 種なので、3 次元の “単位ベクトル” を割り当てることにする。

$$\text{セトナ} \equiv (1,0,0) \tag{13}$$

$$\text{バーシクル} \equiv (0,1,0) \tag{14}$$

$$\text{バージニカ} \equiv (0,0,1) \tag{15}$$

としておく。このような 1 成分だけが 1 になっているベクトルをワンホットベクトル (one-hot vector) とよぶ。

分類問題では、ワンホットベクトルとニューラルネットワークの出力を合わせるため、ニューラルネットワークの出力層の活性化関数 (非線形関数) を以下の特殊なものを取ると良い[7]。

$$\sigma_{\text{softmax}}(v_k) = \frac{\exp(v_k)}{\sum_j \exp(v_j)} \tag{16}$$

定義により $0 < \sigma_{\text{softmax}}(v_k) \leq 1$ と $\sum_k \sigma_{\text{softmax}}(v_k) = 1$ なので、この出力は確率分布として解釈できる。これはソフトマックス関数と呼ばれる。この活性化関数は、全ベクトルの情報を使うため特殊である。例えば、ある入力に対して

$$\sigma_{\text{softmax}} = (0.1, 0.6, 0.3) \tag{17}$$

などが得られると、このニューラルネットワークは、入力データをバーシクルと判断してると解釈することになる。

3. 画像のデータの扱い

画像データをニューラルネットワークに入力する場合、下記のように行なう。

[7] 回帰問題では、普通は恒等関数を用いる。

a. 画像の入力　画像をピクセルの集合だとおもって、ベクトルのように並べる。たとえば、4×4 ピクセルの 2 値の画像の場合、

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \to (0,1,1,0,1,1,1,0,0,1,1,0,1,1,1,1)^\top = \vec{x} \tag{18}$$

と 16 次元ベクトルにする[8]。

4. 誤差関数

誤差関数 (error function) もしくは損失関数 (loss function) は、正解ラベルとニューラルネットワークの出力の差を見れば良い。どちらも今はベクトルなので、2 乗距離で取ることにする[9]。

データを確率変数とみなす。入力を $\vec{X}$、正解ラベルを $\vec{Y}$、ニューラルネットワークを $\vec{f}_\theta$ で書くことにすると、

$$\left\langle L_\theta[\vec{X},\vec{Y}] \right\rangle = \frac{1}{2} \left\langle |\vec{Y} - \vec{f}_\theta(\vec{X})|^2 \right\rangle \tag{19}$$

となる。しかしこれは計算できないため、与えられたデータで近似 (サンプリングをもちいた近似) する。

$$\left\langle L_\theta[\vec{X},\vec{Y}] \right\rangle \approx \frac{1}{N_D} \frac{1}{2} \sum_d |\vec{y}_d - \vec{f}_\theta(\vec{x}_d)|^2 = \overline{L}_\theta \tag{20}$$

詳細は次節以降に譲るが、$\overline{L}_\theta$ を小さくする様に $\theta = \{W^{(l)}, \vec{b}^{(l)}\}_{l=1,2,\cdots}$ を調整することを学習と呼ぶ。

機械学習の文脈では、期待値で定義された誤差 $\langle L_\theta \rangle$ を汎化誤差 (Generalization error, Generalization loss)、データをもちいて計算されたサンプリング近似の誤差 $\overline{L}_\theta$ を訓練誤差 (training error, training loss) と呼ぶ[10]。データをもちいた学習では、訓練誤差 $\overline{L}_\theta$ しか小さく出来ないためこれが使われるが、一方で本来最小化したいのは、汎化誤差 $\langle L_\theta \rangle$ である。この違いが顕著に現れるのが過学習 (過適合 overfitting) である。つまり汎化誤差 $\langle L_\theta \rangle$ を気

[8] 現代的には畳み込みを行なうべきであるが今回は解説しない。
[9] ここはカルバックライブラーダイバージェンスに置き換えられるが今回は解説しない。
[10] 正確には下記でのべる検証用データを除いた、訓練データのみで得られる誤差を訓練誤差と呼ぶ。

にせずに訓練誤差 $\overline{L}_\theta$ のみを最小化した場合、訓練データに含まれていないデータに対してニューラルネットワークが意味のない出力を出してしまう可能性がある[11]。過学習を防ぐために、次節の様にデータの分割が必要となる。

5. データの分割

ここでは次の節で解説する学習のため、データの分割について解説する。具体的に、以下のようなデータが 10 こ集まったとする。

$$\begin{aligned}\mathcal{D} = \big\{&(\vec{x}_1, \vec{y}_1),\ (\vec{x}_2, \vec{y}_2),\ (\vec{x}_3, \vec{y}_3),\ (\vec{x}_4, \vec{y}_4),\ (\vec{x}_5, \vec{y}_5),\\ &(\vec{x}_6, \vec{y}_6),\ (\vec{x}_7, \vec{y}_7),\ (\vec{x}_8, \vec{y}_8),\ (\vec{x}_9, \vec{y}_9),\ (\vec{x}_{10}, \vec{y}_{10})\big\}\end{aligned} \tag{21}$$

これを 2 つに分割する。例えば下記のようにする。

$$\mathcal{D} = \mathcal{D}_{\rm train} \cup \mathcal{D}_{\rm test} \tag{22}$$

$$\begin{aligned}\mathcal{D}_{\rm train} &= \big\{(\vec{x}_1, \vec{y}_1),\ (\vec{x}_2, \vec{y}_2),\ (\vec{x}_3, \vec{y}_3),\ (\vec{x}_4, \vec{y}_4),\ (\vec{x}_5, \vec{y}_5),\ (\vec{x}_6, \vec{y}_6)\big\},\\ \mathcal{D}_{\rm test} &= \big\{(\vec{x}_7, \vec{y}_7),\ (\vec{x}_8, \vec{y}_8),\ (\vec{x}_9, \vec{y}_9),\ (\vec{x}_{10}, \vec{y}_{10})\big\}\end{aligned} \tag{23}$$

データの集合 $\mathcal{D}_{\rm train}$ を学習用データということで学習データ、$\mathcal{D}_{\rm test}$ を検証用のデータということで検証データと呼ぶ。学習データは、トレーニングデータ (training data)、検証データは、ヴァリデーションデータ (validation data、テストデータ) とも呼ばれる。

学習は、反復法であるので、反復の間に検証用データに対しての誤差をモニターすることで過学習の兆候を (ヒューリスティックだが) 調べることが出来る。

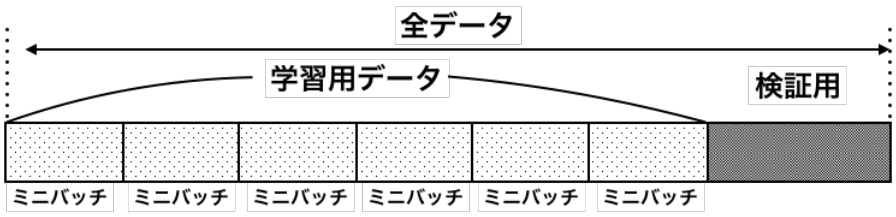

Figure 5: 学習用データは、トレーニングデータ (training data)、検証用データは、ヴァリデーションデータ (validation data) とも呼ばれる。

[11] 訓練誤差と汎化誤差、モデル変数には不等式が成立するがここでは解説しない。

6. 学習

誤差関数 (error function) もしくは損失関数 (loss function) を最小化するパラメータ $\theta = \{W^{(l)}, \vec{b}^{(l)}\}$ を見つける過程を学習とよぶ[12]。以下ではその手法を紹介する。

さて、まずもっとも単純な模型として非線形関数がない模型を考えてみる。すると 2 乗誤差の場合には最小 2 乗法をもちいれば良い。しかしながらニューラルネットワークの場合には最小二乗法を用いることが出来ない。

次に、ニュートン法を考えてみる。簡単のために 1 変数関数 $f(x)$ でニュートン法を導出してみよう。この関数が x^* で最小値をとるとして、その周りで $f'(x)$ をテイラー展開する。

$$0 = f'(x^*) = f'(x) + f''(x)(x^* - x) \tag{24}$$

これを x^* について解くと、

$$x^* = x - \frac{1}{f''(x)} f'(x) \tag{25}$$

となる。この近似に基づいて

$$x \leftarrow x - \frac{1}{f''(x)} f'(x) \tag{26}$$

を反復してとくのがニュートン法である。

ニューラルネットのように多パラメータの場合は 2 階微分がヘッシアンに置き換えられる。ヘッシアンは、

$$H_{IJ} = \frac{\partial}{\partial \theta_I} \frac{\partial}{\partial \theta_J} \left\langle L_\theta[\vec{X}, \vec{Y}] \right\rangle \tag{27}$$

である。ここで I, J は $W^{(l)} = [W^{(l)}_{ij}]$ や $\vec{b}^{(l)} = [b^{(l)}_i]$ とした時に l と i, j をあわせた index である。つまりニューラルネットワークに対するニュートン法は、

$$\theta_I \leftarrow \theta_I - \left(H\right)^{-1}_{IJ} \frac{\partial}{\partial \theta_J} \left\langle L_\theta[\vec{X}, \vec{Y}] \right\rangle \tag{28}$$

の反復になる。

計算可能か調べてみる。パラメータの数を数えてみる。3 層、100 ユニット、とすると、$W^{(l)} = [W^{(l)}_{ij}]$ の要素の数は、$l = 1, 2, 3$ と $i, j = 1, 2, \cdots 100$ として

$$(100 \times 100)_{l=1} \times (100 \times 100)_{l=2} \times (100 \times 100)_{l=3} = 10^{12} \tag{29}$$

---

[12] ちなみに、W や $\vec{b}$ は乱数をもちいて初期化される。どのような初期化がいいかというのは、まだまだ研究がなされている。

である。原理的には、このニューラルネットワークに対応する誤差関数 L に対して θ の 2 階微分 (ヘッシアン) を計算し、さらに逆行列をとってニュートン法で最適なパラメータを探せば良い。ここでパラメータの数を考えてみると $I, J = 1, \cdots, 10^{12}$ となりこれはコストがかかりすぎることが分かる。

ここでは、近似的だが諦めて極値を探すことにする。もう一度

$$\theta_I \leftarrow \theta_I - \big(H\big)^{-1}_{IJ} \frac{\partial}{\partial \theta_J} \Big\langle L_\theta[\vec{X}, \vec{Y}] \Big\rangle$$

を眺めると、$\big(H\big)^{-1}_{IJ}$ を定数としても定性的には良さそうなので以下の繰り返しで最適値を探すことにする。

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \epsilon \frac{\partial}{\partial w_{ij}^{(l)}} \Big\langle L_\theta[\vec{X}, \vec{Y}] \Big\rangle \tag{30}$$

$$b_{i}^{(l)} \leftarrow b_{i}^{(l)} - \epsilon \frac{\partial}{\partial b_{i}^{(l)}} \Big\langle L_\theta[\vec{X}, \vec{Y}] \Big\rangle \tag{31}$$

ϵ は正の小さな定数で学習率 (learning rate) と呼ばれる。このような定数をハイパーパラメータと呼ぶ。この手法を勾配法と呼ぶが、まだこれは計算ができないため、下記のようにサンプリング近似をもちいておこなうことになる。

ここでハイパーパラメータについて注意がある。層の数や層のなかのユニット数、学習率などのハイパーパラメータは基本的に問題に依存している。経験に基づいてあたりをつけることも出来るが理論的な枠組みは (まだ？) 存在しない。

$\langle L \rangle_\theta$ は実際には計算できないので、サンプリングに置き換える。ミニバッチ学習では、バッチ番号を k として

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \epsilon \frac{\partial}{\partial w_{ij}^{(l)}} \overline{L}_\theta^{(k)} \tag{32}$$

$$b_{i}^{(l)} \leftarrow b_{i}^{(l)} - \epsilon \frac{\partial}{\partial b_{i}^{(l)}} \overline{L}_\theta^{(k)} \tag{33}$$

として学習 (更新) する。ただし、$\overline{L}_\theta^{(k)}$ はミニバッチ内での誤差関数の和を表す。期待値ではなく、サンプリングをもちいた近似的な勾配法であるため確率的勾配降下法 Stochastic Gradient Descent (SGD) とよぶ。

1 回のデータセットをなめる更新を 1 エポックという。

7. 様々な最適化法

ここでは紹介しないが、確率的勾配降下法の様々な改良法が知られている。

1. モメンタム (Momentum)

2. RMS prop (root-mean-square propotional)

3. Adam (Adaptive momentum)

詳しくは、参考文献を参照して欲しい。Adam が現在の所のデファクトスタンダードである。ハンズオンでは、Adam を使うが SGD に差し替えても問題はない。

8. 誤差関数の正則化★

過学習を防ぐために誤差関数を修正することを考えよう。たとえば、以下のような項を足すことが可能であろう。

$$E_{\mathrm{L1}} = \sum_l \sum_{ij} |w_{ij}^l| \tag{34}$$

$$E_{\mathrm{L2}} = \frac{1}{2} \sum_l \sum_{ij} |w_{ij}^l|^2 \tag{35}$$

これらは L1 正則化項や L2 正則化項という。これらは、線形回帰の場合にも使用できその場合は、LASSO (least absolute shrinkage and selection operator) および Ridge 回帰と呼ばれる。

C. 万能近似定理★

ここでは、ニューラルネットが関数近似を行える一端を見れる万能近似定理 (universal approximation theorem, 普遍性定理とも訳される) [Cybenko1989] を見ていく。

類似の定理は、たくさんの人によって証明されてきたが、ここでは M. Nielsen による簡便な証明を説明することにする[13]。

a. *1* 次元ニューラルネットワークの万能近似定理　まずはじめに一番簡単なモデルを取り上げる。つまり、隠れ層が 1 層のニューラルネットワーク、特に 1 次元変数 x を与えたとき、1 次元の実数が出てくるニューラルネットワークを考える。

$$f(x) = \vec{W}^{(2)} \cdot \sigma_{\mathrm{step}}(\vec{W}^{(1)} x + \vec{b}^{(1)}) + \vec{b}^{(2)} \tag{36}$$

[13] 彼のウェブサイトでは、インタラクティングに証明を理解できるので是非訪れてみてほしい。
https://nnadl-ja.github.io/nnadl_site_ja/

$\sigma_{\rm step}(x)$ は、活性化関数の役割をはたす**階段関数**で、

$$\sigma_{\rm step}(x) = \begin{cases} 0 \ \ (x < 0), \\ 1 \ \ (x \geq 0), \end{cases} \tag{37}$$

となる。活性化関数がシグモイドである時、その極限で得られる[14]。また引数が多成分のときには、前述のとおり成分ごとに作用することと約束する。$\vec{W}^{(i)}$ は重み、$\vec{b}^{(i)}$ はバイアスである。

$$\vec{W}^{(l)} = (w_1^{(l)}, w_2^{(l)}, w_3^{(l)}, \cdots, w_{n_{\rm unit}}^{(l)})^\top \tag{39}$$

$$\vec{b}^{(l)} = (b_1^{(l)}, b_2^{(l)}, b_3^{(l)}, \cdots, b_{n_{\rm unit}}^{(l)})^\top \tag{40}$$

そして $w_i^{(l)}, \vec{b}^{(l)}$ は、実数とする。l は層（レイヤー）の通し番号で、$1, 2$ である。$n_{\rm unit}$ は、各層でのユニット数であとで決めることにする。

b.　1層目と2層目で何が起こるか　まずは、1層目と2層目で何が起こるかを $n_{\rm unit} = 1$ で見てみる。この場合3層目には、以下が入力される。

$$g_1(x) = \sigma_{\rm step}(w_1^{(1)} x + b_1^{(1)}) \tag{41}$$

このときに、$w_1^{(1)}$ と $b_1^{(1)}$ を調整すると、階段関数が左右に動くことがわかる。さらにここに係数 $w_1^{(2)}$ をかけてみる。

$$g_2(x) = w_1^{(2)} \sigma_{\rm step}(w_1^{(1)} x + b_1^{(1)}) \tag{42}$$

すると、自由に高さを (負の方向にも) 変えることができることがわかる。また第2層に対するバイアス項 $b_1^{(2)}$ を追加すると、階段関数の最低値も変えることができる。

ここで $n_{\rm unit} = 2$ としてみる。すると $\vec{W}^{(l)}$、$\vec{b}^{(l)}$ をパラメータとして任意の形の矩形関数を描くことができる。さらに $n_{\rm unit} = 2k \ (k > 1)$ を考えてみる。すると、多数の矩形関数を重ね合わせたグラフを描くことができる。

連続な目的関数 $t(x)$ を考えたとき、上記で考えたニューラルネットワーク $f(x)$ は、$n_{\rm unit}$ を大きくし、パラメータを調整することでいくらでも近づくことができる。このため、1次元ニューラルネットワークは任意の連続関数を表現できることになる。

---

[14] シグモイドから階段関数を得るには下記のようにすれば良い：

$$\sigma_{\rm sigmoid}(wx + b) = \sigma_{\rm sigmoid}(w(x + b')) \tag{38}$$

$b' = b/w$ として b' を止めて $w \to \infty$ の極限で得られる。これは、形式的にフェルミ分布で絶対零度をとってフェルミ面が出現するのと等価である。

c. ニューラルネットワークの万能近似定理　上記で述べた事実は、高次元に拡張することができる。すなわち、$\vec{x}$ から $\vec{f}(x)$ への連続写像は、ニューラルネットワークを用いて表現することができる。この事実がニューラルネットワークの万能近似定理と呼ばれている。

この定理が非現実的な状況であることに注意されたい。1 つ目は、活性化関数が階段関数であることである。誤差逆伝播法 (Back prop) に基づく学習では、何らかの意味で活性化関数の微分が必要なので階段関数では目的に添えない。2 つ目として、無限個の中間ユニットが必要な点である。これは定量的な問題として、ニューラルネットワークがどのように、どれくらいのスピードでどの様な精度で目的とする関数への近似が良くなるかについては述べられていない。もう 1 つの注意としては、この定理はあくまで目的の関数の近似をニューラルネットワークが与えられるということしか言っていない点である。ただしそれでも、この定理はある程度直感的になぜニューラルネットワークが強力かという断片を説明している。

III. ニューラルネットワークの学習

A. いろいろな誤差関数

誤差関数は、問題によって使い分けるべきであるが選択自体が非自明である。つまり基本的に試行錯誤で調べるしかない。もし、出力が softmax で正でかつ規格化されていた場合、分類問題の場合 softmax crossentropy が良いことが知られている[15]。ソフトマックスクロスエントロピーは

$$L_{\text{crossentropy}}(\mathcal{D}_k) = \sum_{d}^{\mathcal{D}_k} \log\left(\vec{f}_\theta(\vec{x}_d)\right) \cdot (\vec{y}_d) \tag{43}$$

で与えられる。

B. 誤差逆伝播法★

ここではニューラルネットの具体的な最適化方法である誤差逆伝播法 (Back propagation) を学ぶ。誤差逆伝播法は、連鎖律を使うだけで導ける。一方で表式が深層化を阻んでいた理由が明らかになるため時間があれば紹介する。

[15] 1 層でかつ、線形にした場合、ロジックスティック回帰に戻り、分類に特化した回帰になる。

具体例で誤差逆伝播法を学ぶ。簡単のために、バイアスなしの 3 層のニューラルネットを用いる。

$$\vec{f}(\vec{x}) = \sigma(W^{(2)}\vec{z}) \tag{44}$$

$$\vec{z} = \sigma(W^{(1)}\vec{x}). \tag{45}$$

最適化する誤差関数を L とおく。以下では、簡単のために $\vec{f} = \vec{f_\theta}$ と書くことにする。

最適化は、微分をもちいた勾配法、

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \epsilon \frac{\partial L}{\partial w_{ij}^{(l)}} \tag{46}$$

で行うので

$$\frac{\partial L}{\partial w_{ij}^{(l)}} \tag{47}$$

が各 $l = 1, 2$、i, j に対して必要である。要素表示して、連鎖律で計算する。

ここでニューラルネットを要素でかくと、

$$f_i(\vec{x}) = \sigma(u_i^{(2)}), \tag{48}$$

$$u_i^{(2)} = \sum_j w_{ij}^{(2)} z_j^{(2)}, \tag{49}$$

$$z_j^{(2)} = \sigma(u_j^{(1)}), \tag{50}$$

$$u_j^{(1)} = \sum_k w_{jk}^{(1)} x_k \tag{51}$$

となる。

まず誤差関数のウェイト $w_{ij}^{(l)}$ に関する微分は連鎖律で

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = \sum_d \sum_k \frac{\partial L}{\partial f_k} \frac{\partial}{\partial w_{ij}^{(l)}} f_k(\vec{x}_d) \tag{52}$$

と与えられるので

$$\frac{\partial}{\partial w_{ij}^{(l)}} f_k(\vec{x}_d) \tag{53}$$

がわかればよい。

ここで具体的に $l = 2$ ととろう。

$$\frac{\partial}{\partial w_{ab}^{(2)}} f_k(\vec{x}_d) = \frac{\partial}{\partial w_{ab}^{(2)}} \sigma(u_k^{(2)}(\vec{x}_d)), \tag{54}$$

$$= \frac{\partial \sigma(u_k^{(2)})}{\partial u_k^{(2)}} \frac{\partial}{\partial w_{ab}^{(2)}} u_k^{(2)}(\vec{x}_d), \tag{55}$$

$$= \frac{\partial \sigma(u_k^{(2)})}{\partial u_k^{(2)}} \frac{\partial}{\partial w_{ab}^{(2)}} \sum_j w_{kj}^{(2)} z_j^{(2)}(\vec{x}_d), \tag{56}$$

$$= \frac{\partial \sigma(u_k^{(2)})}{\partial u_k^{(2)}} \sum_j \frac{\partial}{\partial w_{ab}^{(2)}} w_{kj}^{(2)} z_j^{(2)}(\vec{x}_d), \tag{57}$$

$$= \frac{\partial \sigma(u_k^{(2)})}{\partial u_k^{(2)}} \sum_j \frac{\partial w_{kj}^{(2)}}{\partial w_{ab}^{(2)}} z_j^{(2)}(\vec{x}_d), \tag{58}$$

$$= \frac{\partial \sigma(u_k^{(2)})}{\partial u_k^{(2)}} \sum_j \delta_{ka}\delta_{jb} z_j^{(2)}(\vec{x}_d), \tag{59}$$

$$= \frac{\partial \sigma(u_k^{(2)})}{\partial u_k^{(2)}} \delta_{ka} z_b^{(2)}(\vec{x}_d). \tag{60}$$

となり、これで学習を行う。ここで $\frac{\partial}{\partial w_{ab}^{(2)}} f_k(\vec{x}_d) \propto z_b^{(2)}$、は順伝播の値に比例しているので順伝播の後にこの計算を行うことになる。

さらに $l = 1$ のウェイトの更新は、同様に連鎖律をつかって

$$\frac{\partial}{\partial w_{ab}^{(1)}} f_k(\vec{x}_d) = \frac{\partial}{\partial w_{ab}^{(1)}} \sigma(u_k^{(2)}) \tag{61}$$

$$= \frac{\partial \sigma(u_k^{(2)})}{\partial u_k^{(2)}} \frac{\partial}{\partial w_{ab}^{(1)}} u_k^{(2)} \tag{62}$$

$$= \frac{\partial \sigma(u_k^{(2)})}{\partial u_k^{(2)}} \sum_j w_{kj}^{(2)} \frac{\partial \sigma(u_j^{(1)})}{\partial u_j^{(1)}} \sum_h \delta_{aj}\delta_{bl}(x_d)_h, \tag{63}$$

$$= \frac{\partial \sigma(u_k^{(2)})}{\partial u_k^{(2)}} \sum_j w_{kj}^{(2)} \frac{\partial \sigma(u_j^{(1)})}{\partial u_j^{(1)}} \delta_{aj}(x_d)_b, \tag{64}$$

$$= \frac{\partial \sigma(u_k^{(2)})}{\partial u_k^{(2)}} w_{ka}^{(2)} \frac{\partial \sigma(u_a^{(1)})}{\partial u_a^{(1)}} (x_d)_b. \tag{65}$$

であたえられる。この様に誤差逆伝播法では、出力層から順に誤差が伝播し、入力層近くの層まで学習が進む。

C. 勾配消失と多層化★

以下では、多層化する際のテクニックを紹介するがハンズオンで使わないので触れる程度にしておく。

ニューラルネットの学習は、誤差逆伝播で行う。(65) で見たように、活性化関数の微分が層をたどるに従って増えていく。シグモイドの場合、微分値は絶対値が 1 以下になるため、あまり層が深いと、微分値の積が極端に小さくなってしまい、学習が進まなくなる。これを勾配消失と呼ぶ。このため実のところ、多層化は性能を上げるのに重要であるが難しくテクニックが必要であった。

多層化を行うには、下記のようなテクニックが知られている。

1. 活性化関数として ReLu を使う
2. スキップコネクションを使う
3. バッチ正則化

この中で、ReLu の使用は明らかに有用である。なぜならば、ReLu の微分は、情報が伝播する正の領域では 1 であり、勾配消失を防ぐ。

1. スキップコネクション

スキップコネクションとは、ネットワークにバイパスをつくる手法である。概念的には図 6 で表される。これは誤差逆伝播の際、効率的に誤差を浅い層に伝播するので勾配消失に対して有効である。

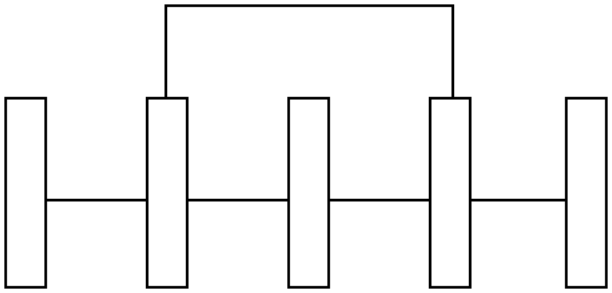

Figure 6: スキップコネクションの図。

2. バッチ正則化

ミニバッチの単位で、層の出力を正規化する。すなわち、ある層での線形変換の出力の平均を 0、分散を 1 にするように整える層を挿入し、活性化関数を通して次の層に渡す。これ

は、内部共形シフトと呼ばれる意味のない変形自由度を取り除く操作とされている。共形シフトとは、データの値を定数倍や定数を足しても線形変換で吸収できてしまう現象を指す。ニューラルネットワークが深層になった場合、これが内部で起こってしまい、学習が不安定になってしまうとされる。

一方で、内部共形シフトを取り除くのではなく、誤差関数の振る舞いを良くするだけであるなどの話もあり、学習の効率向上に役立つのは間違いのだが、なぜうまく行くかははっきりしていない [1805.11604]。

IV. まとめ

ニューラルネットワークを使うには、下記の手続きが必要である (図 7)。

1. データを学習用と検証用に分割する
2. 学習データをミニバッチに分割する
3. ニューラルネットワークを設計する
4. ニューラルネットワークを最適化する
5. 検証用データをもちいて学習データに対してオーバーフィットしていないか調べる

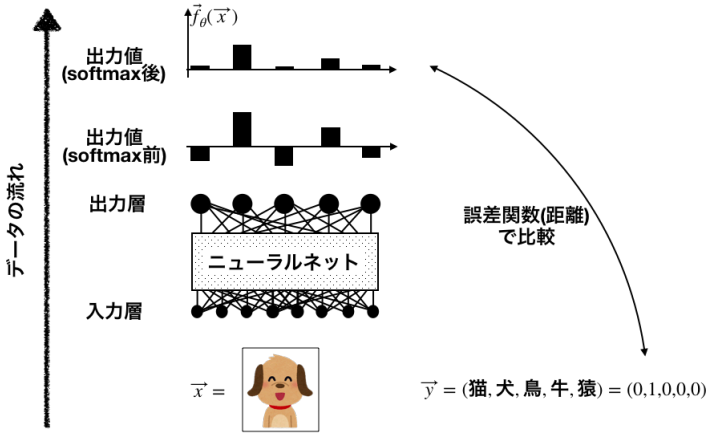

Figure 7: ニューラルネットの使い方。例として分類問題を挙げた。また最終出力にソフトマックス関数を作用させてる。