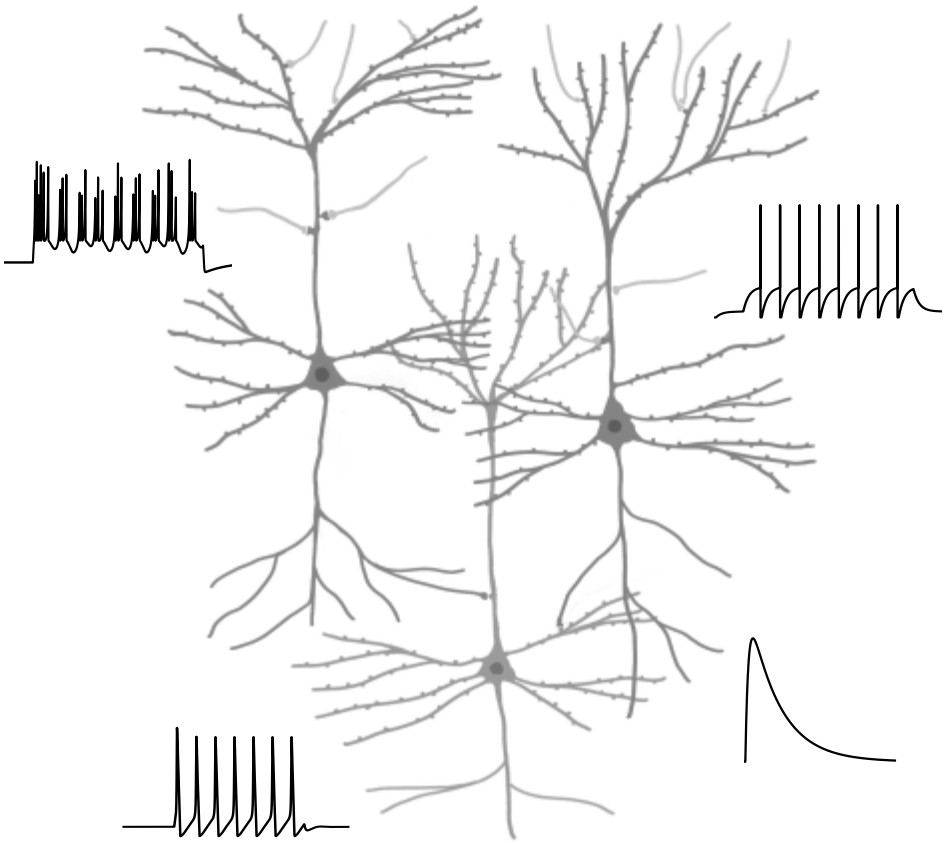
Pythonで

ゼロから作る

Spiking Neural Networks

NumPyによるSNNの
シミュレーションと学習則の実装



ゼロから作る Spiking Neural Networks

【第 2 版】

山拓　著

前書き

この本は **Spiking Neural Networks**(SNN) を Python で実装することを目標とする本です (ライブラリは基本的に **NumPy** と **Matplotlib** のみを用います). 単なる神経活動のシミュレーションでとどまらず, ネットワークの学習則まで実装することを目標とします.

初めになぜこの本を書こうと思ったかを説明しておきます. Spiking Neuron については既に和書・洋書共に優れた本がありますが, 近年の人工神経回路 (artificial neural networks; **ANN**) の成功を踏まえた上で解説している本はありません. SNN は神経系のシミュレーションのみならず, 近年では機械学習の応用も進んでいます. 実際 ANN の発展により SNN も発展しているのですが, 今一つ SNN の研究はとっつきにくく, 解説もほぼありません. そこでこの本では SNN を実際に実装しながら SNN を理解し, SNN の研究がどこまで進んできたか, 今後の課題は何か, ということについて, 計算論的神経科学の観点 (脳をシミュレーションするという観点[*1]) と, 機械学習への応用の観点から考える本になっています. 実装言語として Python を選んだのは, ANN を含む機械学習を行うのに最も用いられているためです. 正直に言えば Python は SNN に向いている言語ではありませんが[*2], ANN を知っている人なら扱えると思い選びました.

断っておきますが, 筆者は SNN の研究を長年やってきた, というわけではありません. この本は SNN の教科書ではなくまとめノートであるということを踏まえて読んでいただければと思います. 専門ではないがゆえに内容に誤りがあると思いますが, もし見つけられた場合は twitter 上ではありますが, `@tak_yamm` までご連絡していただければ幸いです.

[*1] 参考までですが, ANN と脳の対応については `https://github.com/takyamamoto/BNN-ANN-papers` という論文リストを作成しております.

[*2] C++ や Julia は向いている言語の例です.

0.1 Python とライブラリのバージョン

本書のコードは Python 3.7.3 で実行しました. また, 本書で用いた全てのライブラリのバージョンは次の通りです.

- numpy == 1.16.4
- matplotlib == 3.1.0
- tqdm == 4.32.2
- scipy == 1.3.0
- chainer == 6.0.0rc1

なお, 本書ではプログレス管理のために `tqdm` を用います[*3].

また, コードが冗長になるのを防ぐため, **本書における全てのコードは冒頭に**

```
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
np.random.seed(seed=0)
```

を既に記述しているものとします.

0.2 GitHub でのコードの公開

本書で用いたコードは GitHub で公開しており, `https://github.com/takyamamoto/SNN-from-scratch-with-Python` から確認できます. また, 説明の際に対応するコードはその都度, 欄外に表示しています.

0.3 本書を読む上での前提知識

本書は時間の都合上, 日本語文献が十分にある前提知識に関しては省略しています. 対象読者として以下のことを理解している方を想定しています. ご了承ください.

- 初歩的な微積分と線形代数
- 人工神経回路 (ANN) の基礎
- 神経生理学の基礎

[*3] `fastprogress` (https://github.com/fastai/fastprogress) 派の方は適宜置き換えてください.

目次

0.1 Python とライブラリのバージョン 4
0.2 GitHub でのコードの公開 4
0.3 本書を読む上での前提知識 4

第 1 章 ニューロンのモデル 9
1.1 Hodgkin-Huxley モデル 9
1.1.1 Hodgkin-Huxley モデルにおける膜の等価回路モデル 9
1.1.2 Hodgkin-Huxley モデルの実装 11
1.1.3 アノーダルブレイク 16
1.2 Leaky integrate-and-fire モデル 17
1.2.1 LIF モデルの単純な実装 17
1.2.2 LIF モデルの class の実装 19
1.2.3 LIF モデルの F-I curve 22
1.3 Izhikevich モデル 25
1.3.1 Izhikevich モデルの単純な実装 25
1.3.2 様々な発火パターンのシミュレーション 28
1.3.3 Izhikevich モデルの class の実装 29
1.4 Inter-spike interval モデル 31
1.4.1 ポアソン過程モデル 31
1.4.2 死時間付きポアソン過程モデル (PPD) 36
1.4.3 ガンマ過程モデル 38
1.5 発火率モデル 41
1.5.1 発火率 41
1.5.2 離散時間発火率モデル 41
1.5.3 連続時間発火率モデル 42
コラム：確率的シナプス電流のノイズによる表現 44

第 2 章 シナプスのモデル 47

2.1 Current-based vs Conductance-based シナプス 47
2.1.1 Current-based シナプス 48
2.1.2 Conductance-based シナプス 48
2.2 指数関数型シナプスモデル (Exponential synapse model) 51
2.2.1 単一指数関数型モデル (Single exponential model) 51
2.2.2 二重指数関数型モデル (Double exponential model) 52
2.2.3 指数関数型シナプスの単純な実装 54
2.2.4 指数関数型シナプスの class の実装 56
2.3 動力学モデル (Kinetic model) 57
2.4 シナプス入力の重みづけ 58
コラム：神経回路の汎用シミュレータ 60

第 3 章 ネットワークの構築 61
3.1 ニューロン間の接続 61
3.1.1 全結合 (Full connection) 61
3.1.2 2 次元の畳み込み (Convolution2D connection) 62
3.1.3 遅延結合 (Delay connection) 63
3.2 ランダムネットワーク 65
3.3 SNN を訓練する 68
3.3.1 SNN の意義 68
3.3.2 SNN を訓練する 5 つの方針 69

第 4 章 誤差逆伝搬法の近似による教師あり学習 71
4.1 SuperSpike 法 71
4.1.1 損失関数の導関数の近似 71
4.1.2 離散化した重みの更新と RMaxProp 73
4.1.3 誤差信号の逆伝搬について 74
4.1.4 SuperSpike 法の実装 75
4.2 RNN としての SNN の BPTT を用いた教師あり学習 84

第 5 章 STDP 則による教師なし学習 87
5.1 STDP (Spike-timing-dependent plasticity) 則 87
5.1.1 Pair-based STDP 則 87
5.1.2 オンライン STDP 則 89
5.1.3 重み依存的な STDP 94
5.2 STDP 則と 2 層 WTA ネットワークによる教師なし学習 96

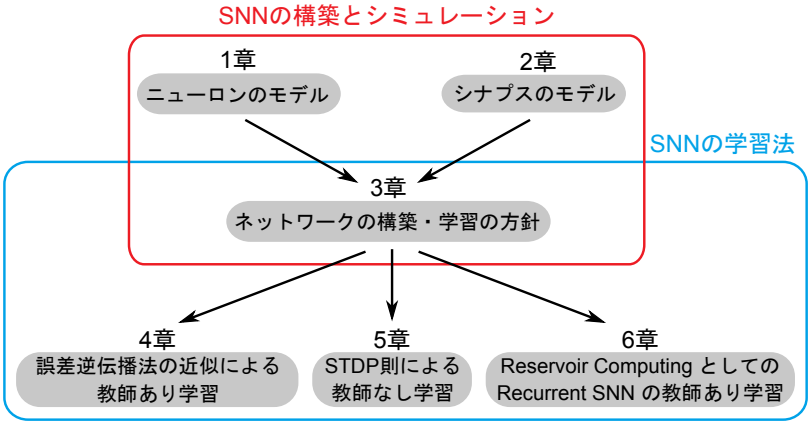
5.2.1 ニューロンとシナプスのモデル 97
5.2.2 興奮性ニューロンのラベリング 99
5.2.3 MNIST データセットのスパイク列への変換 102
5.2.4 ネットワークの実装 103
5.2.5 学習イテレーションと結果の表示 107
5.2.6 ネットワークの評価 111
5.2.7 興奮性ニューロンの受容野の描画 114

第 6 章 Reservoir Computing としての Recurrent SNN の教師あり学習 117
6.1 Reservoir Computing 117
6.2 FORCE 法と Recurrent SNN への適用 117
6.3 Recurrent SNN に正弦波を学習させる 118
6.3.1 ネットワークの構造と教師信号 118
6.3.2 固定重みの初期化 119
6.3.3 RLS 法による重みの更新 120
6.3.4 FORCE 法の実装 120
6.3.5 鳥の鳴き声の再現と海馬の記憶と再生 126
6.4 RLS 法の導出 126

参考文献 128

本書の各章の関係

本書の各章の関係は次の図のようになっています．参考にしてお読みいただければと思います．



第 1 章

ニューロンのモデル

ニューロンのモデル[*1](ニューロンの膜電位変化のモデル) は数多く考案されていますが[*2], 重要なモデルに絞って解説します. 解説するモデルは, 金字塔である **Hodgkin-Huxley モデル**, SNN によく用いられる **Leaky integrate-and-fire モデル**と **Izhikevich モデル**, 入力スパイクとして用いられる **Inter-spike interval モデル**です. さらに Spike-based ではないですが, **発火率 (firing rate) モデル**についても解説します.

1.1 Hodgkin-Huxley モデル

1.1.1 Hodgkin-Huxley モデルにおける膜の等価回路モデル

Hodgkin-Huxley モデル[*3](HH モデル) は, A.L. Hodgkin と A.F. Huxley によって 1952 年に考案されたニューロンの膜興奮を表すモデルです. 彼らはヤリイカの巨大神経軸索に対する**電位固定法** (voltage-clamp) を用いた実験を行い, 実験から得られた観測結果を元にモデルを構築しています.

HH モデルには等価な電気回路モデルがあり, **膜の並列等価回路モデル** (parallel conductance model) と呼ばれています. 膜の並列等価回路モデルでは, ニューロンの細胞膜をコンデンサ, 細胞膜に埋まっているイオンチャネルを可変抵抗 (動的に変化する抵抗) として置き換えます[*4]. **イオンチャネル** (ion channel) は特定のイオン (例えばナトリウ

[*1] シナプスもニューロンに含まれる構造なので, 分けるのは変な感じですが, 分けたほうが実装は楽なのでこうしています.

[*2] 他にどのようなモデルが考案されているかについては (Izhikevich, 2004) などを参照してください.

[*3] この節はどう書いても (宮川 & 井上, ニューロンの生物物理, 2013) の劣化とならざるを得ないので実装以外はそちらを読んでほしいです.

[*4] なお, 当時は Hodgkin と Huxley はイオンの通り道があるということは分かっていましたが, イオンチャネルの存在はまだ分かっていませんでした.

ムイオンやカリウムイオンなど) を選択的に通す[*5]膜輸送体の一種です. それぞれのイオンの種類において, 異なるイオンチャネルがあります (また同じイオンでも複数の種類のイオンチャネルがあります). また, イオンチャネルにはイオンの種類に応じて異なる**コンダクタンス** (抵抗の逆数で電流の「流れやすさ」を意味します) と**平衡電位** (equilibrium potential) があります[*6]. HH モデルでは, ナトリウム (Na^+) チャネル, カリウム (K^+) チャネル, 漏れ電流 (leak current) のイオンチャネルを仮定します. 漏れ電流のイオンチャネルは当時特定できなかったチャネルで, 膜から電流が漏れ出すチャネルを意味します. なお, 現在では漏れ電流の多くは Cl^- イオン (chloride ion) によることが分かっています.

それでは, 等価回路モデルを用いて電位変化の式を立ててみましょう. 図 1.1 において, C_m を細胞膜のキャパシタンス (膜容量), $I_m(t)$ を細胞膜を流れる電流 (外部からの入力電流), $I_{\mathrm{Cap}}(t)$ を膜のコンデンサを流れる電流, $I_{\mathrm{Na}}(t)$, $I_K(t)$ をそれぞれナトリウムチャネルとカリウムチャネルを通って膜から流出する電流, $I_{\mathrm{L}}(t)$ を漏れ電流とします. このとき,

$$I_m(t) = I_{\mathrm{Cap}}(t) + I_{\mathrm{Na}}(t) + I_{\mathrm{K}}(t) + I_{\mathrm{L}}(t) \tag{1.1}$$

という仮定をおきます. よって膜電位を $V(t)$ とすると, Kirchhoff の第二法則 (Kirchhoff's Voltage Law) より,

$$\underbrace{C_m \frac{dV(t)}{dt}}_{I_{\mathrm{Cap}}(t)} = I_m(t) - I_{\mathrm{Na}}(t) - I_{\mathrm{K}}(t) - I_{\mathrm{L}}(t) \tag{1.2}$$

となります.

Hodgkin と Huxley はチャネル電流 $I_{\mathrm{Na}}, I_K, I_{\mathrm{L}}$ が従う式を実験的に求めました.

$$I_{\mathrm{Na}}(t) = g_{\mathrm{Na}} \cdot m^3 h (V - E_{\mathrm{Na}}) \tag{1.3}$$

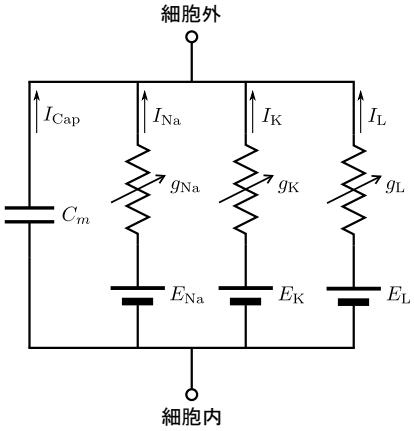
$$I_{\mathrm{K}}(t) = g_{\mathrm{K}} \cdot n^4 (V - E_{\mathrm{K}}) \tag{1.4}$$

$$I_{\mathrm{L}}(t) = g_{\mathrm{L}} (V - E_{\mathrm{L}}) \tag{1.5}$$

ただし, $g_{\mathrm{Na}}, g_{\mathrm{K}}$ はそれぞれ Na^+, K^+ の最大コンダクタンスです. g_{L} はオームの法則に従うコンダクタンスで, L コンダクタンスは時間的に変化はしません. また, m は Na^+ コ

[*5] イオンチャネルでは, あるイオンを通すイオンチャネルの中を, それより小さいイオンが通るということが起こらないようになっています. このことはイオンチャネルの分子構造の設計が非常に優れたものであることを示します.

[*6] イオンの流れは**濃度勾配**と**電位勾配**による駆動力 (driving force) に基づきます. 濃度勾配による駆動力を拡散 (diffusion), 電位勾配による駆動力をドリフト (drift) といいます. 細胞膜の電位変化により, イオンの細胞内への流入量と細胞外への流出量が等しくなった時 (つまり拡散とドリフトがつり合った時), 見かけ上イオンの流れが無くなります. このときの電位を**平衡電位 (equilibrium potential)** と呼びます. 平衡電位を境としてイオンの正味の流れの向きが変化するので, 平衡電位のことを**反転電位 (reversal potential)** とも呼びます.



▲ 図 1.1　HH モデルにおける膜の等価回路モデル

ンダクタンスの活性化パラメータ, h は Na^+ コンダクタンスの不活性化パラメータ, n は K^+ コンダクタンスの活性化パラメータであり, ゲートの開閉確率を表しています. よって, HH モデルの状態は V, m, h, n の 4 変数で表されます. これらの変数は以下の x を m, n, h に置き換えた 3 つの微分方程式に従います.

$$\frac{dx}{dt} = \alpha_x(V)(1-x) - \beta_x(V)x \tag{1.6}$$

ただし, V の関数である $\alpha_x(V)$, $\beta_x(V)$ は m, h, n によって異なり, 次の 6 つの式に従います.

$$\begin{aligned}
\alpha_m(V) &= \frac{0.1(25-V)}{\exp\left[(25-V)/10\right]-1}, & \beta_m(V) &= 4\exp\left(-V/18\right) \\
\alpha_h(V) &= 0.07\exp\left(-V/20\right), & \beta_h(V) &= \frac{1}{\exp\left[(30-V)/10\right]+1} \\
\alpha_n(V) &= \frac{0.01(10-V)}{\exp\left[(10-V)/10\right]-1}, & \beta_n(V) &= 0.125\exp\left(-V/80\right)
\end{aligned} \tag{1.7}$$

1.1.2　Hodgkin-Huxley モデルの実装

それでは, これまでに説明した式を用いて HH モデルを実装してみましょう. 定数は次のように設定します.

$$\begin{aligned}
C_m = 1.0, g_{\mathrm{Na}} = 120, g_{\mathrm{K}} = 36, g_{\mathrm{L}} = 0.3 \\
E_{\mathrm{Na}} = 50.0, E_{\mathrm{K}} = -77, E_{\mathrm{L}} = -54.387
\end{aligned}$$

基本的には以上の式を関数として定義し, 微分方程式のソルバー[*7]に渡すだけです. ソルバーとしては陽的 Euler 法または 4 次の Runge-Kutta 法[*8]を用います. また, 少し複雑になるのでここでは 1 ニューロンのみのシミュレーションを行います. なお, コードの書かれた Python ファイルの名称とパスは基本的に欄外に示します[*9]. まずは HH モデルのクラスを定義します.

```
class HodgkinHuxleyModel:
    def __init__(self, dt=1e-3, solver="RK4"):
        self.C_m  =   1.0 # 膜容量 (uF/cm^2)
        self.g_Na = 120.0 # Na+ の最大コンダクタンス (mS/cm^2)
        self.g_K  =  36.0 # K+ の最大コンダクタンス (mS/cm^2)
        self.g_L  =   0.3 # 漏れイオンの最大コンダクタンス (mS/cm^2)
        self.E_Na =  50.0 # Na+ の平衡電位 (mV)
        self.E_K  = -77.0 # K+ の平衡電位 (mV)
        self.E_L  = -54.387 #漏れイオンの平衡電位 (mV)

        self.solver = solver
        self.dt = dt

        # V, m, h, n
        self.states = np.array([-65, 0.05, 0.6, 0.32])
        self.I_m = None

    def Solvers(self, func, x, dt):
        # 4th order Runge-Kutta 法
        if self.solver == "RK4":
            k1 = dt*func(x)
```

[*7] SNN は基本的に微分方程式を解いてシミュレーションを行います. ソルバーとしては計算量の観点から陽的 Euler 法が主に用いられますが, 精度を上げたい場合には 4 次の Runge-Kutta 法などを用います. これ以降は Euler 法しか使用しません.

[*8] Runge-Kutta 法という文字を見るだけで興奮する人もいるらしい. Runge-Kutta という文字に応答する Runge-Kutta ニューロンでもあるのでしょうか.

[*9] コードは./SingleFileSimulations/Neurons/HH_single.py です. 実装において https://hodgkin-huxley-tutorial.readthedocs.io/en/latest/index.html を参考にしました. また複数ニューロンのシミュレーション例は ./SingleFileSimulations/Neurons/HH_model_multiple_neurons.py です.

第 2 章

シナプスのモデル

スパイクが生じたことによる膜電位変化は軸索を伝播し, **シナプス**という構造により, 次のニューロンへと興奮が伝わります. このときの伝達の仕組みとして, シナプスには**化学シナプス** (chemical synapse) と Gap junction による**電気シナプス** (electrical synapse) があります. 中枢神経系には両方存在しますが, 今回は化学シナプスのみを考えます. 化学シナプスの場合, シナプス前膜からの**神経伝達物質**の放出, シナプス後膜の受容体への神経伝達物質の結合, イオンチャネル開口による**シナプス後電流** (postsynaptic current; **PSC**) の発生, という過程が起こります[*1]. そのため, シナプス前細胞のスパイク列 (spike train) は次のニューロンにそのまま伝わるのではなく, ある種の時間的フィルターをかけられて伝わります[*2]. この章では, このようにシナプス前細胞で生じた発火が, シナプス後細胞の膜電位に与える過程のモデルについて説明します.

2.1 Current-based vs Conductance-based シナプス

具体的なシナプスのモデルの前に, この節ではシナプス入力 (synaptic drive) の 2 つの形式, **Current-based シナプス**と **Conductance-based シナプス**について説明します. 簡単に説明すると, Current-based シナプスは入力電流が変化するというモデルで, Conductance-based シナプスはイオンチャネルのコンダクタンス (電流の流れやすさ) が変化するというモデルです (cf. Cavallari et al., 2014).

以下では例として, 次の LIF ニューロンの方程式におけるシナプス入力を考えます.

$$\tau_m \frac{dV_m(t)}{dt} = -(V_m(t) - V_{\rm rest}) + R_m I_{\rm syn}(t) \tag{2.1}$$

[*1] かなり簡略化して書きましたが, 実際にはかなりの過程を含みます. しかし, これらの過程を全てモデル化するのは計算量がかなり大きくなるので, 基本的には簡易的な現象論的なモデルを用います.

[*2] このフィルターを**シナプスフィルター** (synaptic filter) と呼びます

とします. ただし, τ_m は膜電位の時定数, $V_m(t)$ は膜電位, $V_{\rm rest}$ は静止膜電位, R_m は膜抵抗です. 最後にシナプス入力の電流 $I_{\rm syn}(t)$ ですが[*3], ここが 2 つのモデルにおいて異なってきます.

2.1.1 Current-based シナプス

Current-based シナプスは単純に**入力電流が変化**するというモデルで, 簡略化したい場合によく用いられます. シナプス入力 $I_{\rm syn}(t)$ はシナプス効率 (synaptic efficacy)[*4]を $J_{\rm syn}$ とし (ただし単位は pA), シナプスの動態 (synaptic kinetics) を $s_{\rm syn}(t)$ とすると, 式 (2.2) のようになります. ただし, シナプスの動態とは, 前細胞に注目すれば神経伝達物質の放出量, 後細胞に注目すれば神経伝達物質の結合量やイオンチャネルの開口率を表します.

$$I_{\rm syn}(t) = \underbrace{J_{\rm syn} s_{\rm syn}(t)}_{\text{電流の変化}} \tag{2.2}$$

ただし, $s_{\rm syn}(t)$ は, 例えば次節で紹介する α 関数を用いる場合,

$$s_{\rm syn}(t) = \frac{t}{\tau_s} \exp\left(1 - \frac{t}{\tau_s}\right) \tag{2.3}$$

のようになります.

2.1.2 Conductance-based シナプス

Conductance-based シナプスはイオンチャネルの**コンダクタンスが変化**するというモデルです. 例えば, Hodgkin-Huxley モデルは Conductance-based モデルの 1 つです. このモデルの方が生理学的に妥当です[*5]. シナプス入力は $I_{\rm syn}(t)$ は次のようになります.

$$I_{\rm syn}(t) = \underbrace{g_{\rm syn} s_{\rm syn}(t)}_{\text{コンダクタンスの変化}} \cdot (V_{\rm syn} - V_m(t)) \tag{2.4}$$

ただし, $g_{\rm syn}$ はシナプスの最大コンダクタンス[*6](単位は nS), $V_{\rm syn}$ はシナプスの平衡電位 (単位は mV) を表します. これらも $J_{\rm syn}$ と同じく, シナプスにおける受容体の種類によって決まる定数です.

[*3] シナプス (synapse) 入力であることを明らかにするために syn と添え字をつけています.

[*4] シナプス強度 (Synaptic strength) とは違い, 受容体の種類 (GABA 受容体や AMPA 受容体, およびそのサブタイプなど) によって決まります.

[*5] 例えば抑制性シナプスは膜電位が平衡電位と比べて脱分極側にあるか, 過分極側にあるかで抑制的に働くか興奮的に働くかが逆転します. これは Current-based シナプスでは再現できません.

[*6] ただし, $s_{\rm syn}$ の最大値を 1 に正規化する場合です. 正規化は必須ではないので, 単なる係数と思うのがよいでしょう.

第 3 章

ネットワークの構築

第 1 章ではニューロンのモデルについて, 第 2 章ではシナプスのモデルについて学んできました. 第 3 章ではそれらのモデルを組み合わせたネットワークを構築してみます. また, 最後の節では SNN を学習させる意義とその方針について説明します.

3.1 ニューロン間の接続

ネットワークを構成するにはあるニューロンがどのニューロンに投射しているか, どのように活動が伝搬するかを記述する必要があります. この節ではニューロン同士の間の接続関係の記述の仕方について説明します.

3.1.1 全結合 (Full connection)

i 層目のニューロンが $i+1$ 層目のニューロンに全て繋がっていることを**全結合 (fully connected)** と言います. ただし, 全てが完全に繋がっているということではなく, 結合重みが 0 の場合は繋がっていないことを表します. なお, この結合様式は既に第 2 章で出てきています. 全結合は ANN では入力に重み行列を乗算し, バイアスを加算するようなアフィン変換で表されますが, SNN では入力に重み行列を乗算するだけの線形変換を用いることが主です.

単に重み行列を用意するだけでも (この本の内容に限るなら) 問題はありませんが, 重みを学習させる場合には `class` を用意しておくと取り扱いがしやすくなります. コードは次のようになります[*1]. このコードを `Connections.py` として `Models` ディレクトリ内に保存しておきましょう.

[*1] コードは./TrainingSNN/Models/Connections.py に含まれます.

```
class FullConnection:
    def __init__(self, N_in, N_out, initW=None):
        if initW is not None:
            self.W = initW
        else:
            self.W = 0.1*np.random.rand(N_out, N_in)

    def backward(self, x):
        return np.dot(self.W.T, x) #self.W.T @ x

    def __call__(self, x):
        return np.dot(self.W, x) #self.W @ x
```

3.1.2 2 次元の畳み込み (Convolution2D connection)

SNN では ANN の 1 つの結合形式である**畳み込み層** (convolutional layer) を含むことがあります．全結合が通常の ANN と同様であったように畳み込み層も全く同じ操作です．そのため, 今回実装はしないのですが, 行列計算ライブラリとして NumPy ではなく, Tensorflow や Pytorch, Chainer 等を使う場合には畳み込み層の関数が実装されているのでそれを使うとよいでしょう．

念のため, 2D 畳み込み層の出力テンソル ($H \times W \times C$ のテンソル, H, W はそれぞれ画像の高さと幅, C はチャネル数) の解釈について説明しておきます．まず, 1 つのチャネルは同種 (同系統の受容野を持つ) の $H \times W$ 個のニューロンの活動です．本来は「同種」ですが, 空間的な不変性により「同一」と見なし, 重み共有 (weight sharing, weight tying) をしてスライディングウィンドウ (sliding window) の操作をすることで, ニューロンを視野全体に複製 (要は 1 つのニューロンをコピペ) しています．実際の視覚野では近傍のニューロンの活動を受けることによる畳み込みはしていますが, 重み共有[*2]とスライディングウィンドウはしていない, ということです．

[*2] ただし, 類似の遺伝子発現による初期値共有はしているかもしれないですが.

第 4 章

誤差逆伝搬法の近似による教師あり学習

ANN は誤差逆伝搬法 (backpropagation) を用いてパラメータを学習することができますが, SNN は誤差逆伝搬法を直接使用することはできません. しかし, 誤差逆伝搬法の近似をすることで SNN を訓練することができるようになります. SNN を誤差逆伝搬法で訓練することは **SpikeProp 法** (Bohte et al., 2000) や **ReSuMe 法** (Ponulak, Kasiński, 2010) など多数の手法が考案されてきました (他の方針としては Lee et al. 2016; Huh & Sejnowski, 2018; Wu et al., 2018; Shrestha & Orchard, 2018; Tavanaei & Maida, 2019; Thiele et al., 2019; Comsa et al., 2019 など多数). この章の初めでは, 代表して SpikeProp 法の改善手法である **SuperSpike 法** (Zenke & Ganguli, 2018) の実装をしてみます.

4.1 SuperSpike 法

SuperSpike 法 (supervised learning rule for spiking neurons) はオンラインの教師あり学習で SpikeProp 法と同様にスパイク列を教師信号とし, そのスパイク列を出力するようにネットワークを最適化します (Zenke & Ganguli, 2018). SpikeProp 法と異なるのはスパイクの微分ではなく, 膜電位についての関数の微分を用いていることです. このため, 発火が生じなくても学習が進行します.

4.1.1 損失関数の導関数の近似

まず最小化したい損失関数 L から考えましょう. i 番目のニューロンの教師信号となるスパイク列 $\hat{S}_i$ に出力 S_i を近づけます (スパイク列は $S_i(t) = \sum_{t_k<t} \delta\left(t - t_i^k\right)$ と表され

ます)[*1]. SpikeProp 法ではこれらの二乗誤差を損失関数としていますが, SuperSpike 法ではそれぞれのスパイク列を二重指数関数フィルター α で畳み込みした後に二乗誤差を取ります.

$$L(t) = \frac{1}{2}\int_{-\infty}^{t} ds \left[\left(\alpha * \hat{S}_i - \alpha * S_i\right)(s)\right]^2 \tag{4.1}$$

ただし, $*$ は畳み込み演算子です. これは **van Rossum 距離** (van Rossum, 2001)[*2]を表します. 損失関数をこのように設定することで, SpikeProp と異なり, 完全にスパイク列が一致するまで誤差信号は 0 になりません. 損失関数 L を j 番目のシナプス前ニューロンから i 番目のシナプス後ニューロンへのシナプス強度 w_{ij} で微分すると, 次のようになります.

$$\frac{\partial L}{\partial w_{ij}} = -\int_{-\infty}^{t} ds \left[\left(\alpha * \hat{S}_i - \alpha * S_i\right)(s)\right] \left(\alpha * \frac{\partial S_i}{\partial w_{ij}}\right)(s) \tag{4.2}$$

目標はこの $\dfrac{\partial L}{\partial w_{ij}}$ を計算し, 確率的勾配降下法 (stochastic gradient discent; SGD) により $w_{ij} \leftarrow w_{ij} - r\dfrac{\partial L}{\partial w_{ij}}$ と最適化することです (ただし r は学習率). ここでの問題点は $\dfrac{\partial S_i}{\partial w_{ij}}$ の部分です. S_i は δ 関数を含むため, 微分すると発火時は ∞, 非発火時は 0 となり, 学習が進みません. そこで $S_i(t)$ を LIF ニューロンの膜電位[*3]$U_i(t)$ の非線形関数 $\sigma(U_i(t))$ で近似します. 非線形関数としては高速シグモイド関数 (fast sigmoid) $\sigma(x) = x/(1+|x|)$ を使用しています. ここまでの近似計算を纏めると

$$\frac{\partial S_i}{\partial w_{ij}} \approx \frac{\partial \sigma\left(U_i\right)}{\partial w_{ij}} = \sigma'\left(U_i\right)\frac{\partial U_i}{\partial w_{ij}} \tag{4.3}$$

となります. ただし, $\sigma'\left(U_i\right) = \left(1 + |\beta(U_i - \vartheta)|\right)^{-2}$ です. ϑ は LIF ニューロンの発火閾値で -50 mV とされています. β は係数で $(1\text{ mV})^{-1}$ です.

残った $\dfrac{\partial U_i}{\partial w_{ij}}$ の部分ですが, シナプス強度 w_{ij} の変化により j 番目のシナプス前ニューロンの発火 $S_j(t)$ が i 番目のシナプス後細胞の膜電位変化に与える影響が変化するという観点から, $\dfrac{\partial U_i}{\partial w_{ij}} \approx \epsilon * S_j(t)$ と近似します. ただし, ϵ は α と同じ二重指数関数フィルターです. また, これはシナプスでの神経伝達物質の濃度として解釈できるとされています.

[*1] 通常, 予測値に $\hat{}$ を付けることが多いですが, ここでは論文の表記に従って $\hat{S}$ を教師信号としています.

[*2] スパイク列の類似度を計算する手法としては他に Victor-Purpura 距離や, Schreiber *et al.* 類似度など, 数多く考案されています. (Dauwels et al., 2008) や Scholarpedia の "Measures of spike train synchrony"(`http://www.scholarpedia.org/article/Measures_of_spike_train_synchrony`) を参照してください.

[*3] これまで V や v を使っていましたが, 論文にあわせて U を用います.

第 5 章

STDP 則による教師なし学習

5.1 STDP (Spike-timing-dependent plasticity) 則

5.1.1 Pair-based STDP 則

Spike-timing-dependent plasticity (STDP, スパイクタイミング依存性可塑性) はシナプス前細胞と後細胞の発火時刻の差によってシナプス強度が変化するという現象です (Markram et al. 1997; Bi & Poo 1998). 典型的な STDP 側は **Pair-based STDP 則**と呼ばれ, シナプス前細胞と後細胞の 2 つのスパイクのペアの発火時刻によって LTP(long-term potentiation, 長期増強) や LTD(long-term depression, 長期抑圧) が起こります. この節ではこの Pair-based STDP 則について説明します.

シナプス後細胞におけるスパイク (postsynaptic spike) の発生時刻 $t_{\rm post}$ とシナプス前細胞におけるスパイク (presynaptic spike) の発生時刻 $t_{\rm pre}$ の差を $\Delta t_{\rm spike} = t_{\rm post} - t_{\rm pre}$ とします[*1]. $\Delta t_{\rm spike}$ はシナプス前細胞, 後細胞の順で発火すれば正, 逆なら負となります. Pair-based STDP 則では, シナプス前細胞から後細胞へのシナプス強度 (w)[*2]の変化 Δw は $\Delta t_{\rm spike}$ に依存的に以下の式に従って変化します (Song et al,, 2000).

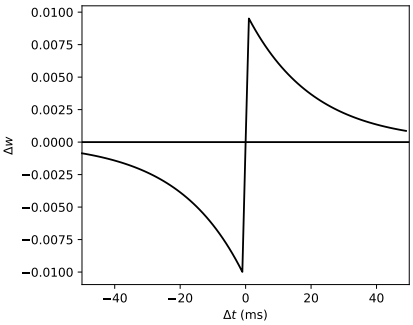
$$\Delta w = \begin{cases} A_+ \exp\left(-\dfrac{\Delta t_{\rm spike}}{\tau_+}\right) & (\Delta t_{\rm spike} > 0) \\ -A_- \exp\left(-\dfrac{|\Delta t_{\rm spike}|}{\tau_-}\right) & (\Delta t_{\rm spike} < 0) \end{cases} \tag{5.1}$$

A_+, A_- は正の定数, または重み依存的な関数 (後述) です. $\Delta t_{\rm spike} > 0$ のときは LTP が起こり, $\Delta t_{\rm spike} < 0$ のときは LTD が起こります. このタイプの STDP 則は **Hebbian**

[*1] $\Delta t_{\rm spike}$ の定義は元々は逆になっており, (Song et al,, 2000) では $\Delta t_{\rm spike} = t_{\rm pre} - t_{\rm post}$ としています. また, 添え字は離散時のタイムステップと混同しないために付けています.

[*2] シナプス強度 w に添え字をつけていませんが, この場合はシナプス前細胞と後細胞の 2 つの細胞しかないと仮定して考えています.

STDP と呼ばれ, Hebb 則[*3]に従うシナプス強度の変化が起こります[*4]. $A_+ = 0.01$, $A_-/A_+ = 1.05$, $\tau_+ = \tau_- = 20$ ms としたときの Δt_{spike} に対する Δw は図 5.1 のようになります.



▲ 図 5.1 STDP

以下は図 5.1 を描画するためのコードです[*5]

```
tau_p = tau_m = 20 #ms
A_p = 0.01
A_m = 1.05*A_p
dt = np.arange(-50, 50, 1) #ms
dw = A_p*np.exp(-dt/tau_p)*(dt>0) - A_m*np.exp(dt/tau_p)*(dt<0)

plt.figure(figsize=(5, 4))
plt.plot(dt, dw)
plt.hlines(0, -50, 50); plt.xlim(-50, 50)
plt.xlabel("$\Delta t$ (ms)"); plt.ylabel("$\Delta w$")
plt.show()
```

[*3] 「シナプス前細胞が発火してからシナプス後細胞が発火することによりシナプス結合が増強される」という法則です. 1949 年に Donald Hebb により提唱されました.

[*4] Hebb 則に従わない STDP もあり, 例えば LTP と LTD の挙動が逆のものを **Anti-Hebbian STDP** と呼びます (Bell et al., 1997 など参照).

[*5] コードは`./SingleFileSimulations/STDP/stdp.py` です.

第 6 章

Reservoir Computing としての Recurrent SNN の教師あり学習

この章では Reservoir Computing としての Recurrent SNN と, それを学習するための FORCE 法について解説します.

6.1 Reservoir Computing

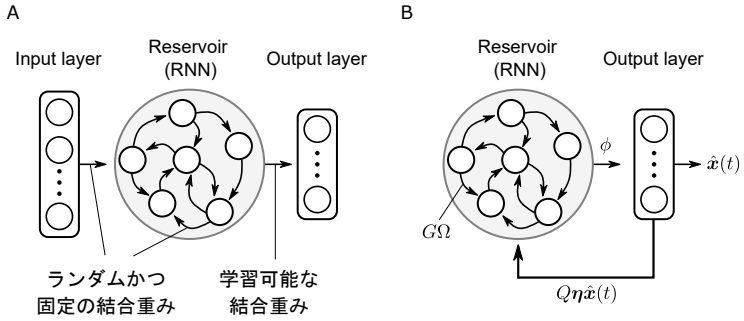
Reservoir Computing は, RNN[*1]のモデルの一種です. 一般の RNN が全ての結合重みを学習するのに対し, Reservoir Computing では RNN のユニット間の結合重みはランダムに初期化して固定し, **出力の結合重みだけを学習**します. そのため, Reservoir Computing は学習するパラメータが少なく, 学習も高速に行えます (もちろん関数の表現力は一般の RNN の方が優れています).

Reservoir というのは溜め池 (貯水池) を意味します. Reservoir Computing では, まず入力信号をランダムな固定重みにより高次空間の信号に変換し, Reservoir RNN(信号の溜め池) に保持します. そして, Reservoir RNN のユニットの活動として保持された情報を学習可能な重みにより線形変換し, 出力とします. このとき, ネットワークの出力が教師信号と一致するように出力重みを更新します.

6.2 FORCE 法と Recurrent SNN への適用

Reservoir Computing における教師あり学習の手法の 1 つとして, **FORCE 法**と呼ばれるものがあります. **FORCE** (First-Order Reduced and Controlled Error) 法は (Sussillo & Abbott, 2009) で提案された学習法で, 元々は発火率ベースの RNN に対す

[*1] ここでは発火率モデルについての RNN について述べています.



▲ 図 6.1 (A)Reservoir Computing の一般的なモデル. 入力と中間にはランダムに固定された重みを用い, 出力のみ学習可能となっています. (B)FORCE 法で用いるモデルの 1 つ. 7.3 節以降でこのモデルの実装を行います.

るオンラインの学習法です (具体的な方法については次節で解説します). さらに (Nicola & Clopath, 2017) は FORCE 法が Recurrent SNN の学習に直接的に使用できる, ということを示しました. この章では Nicola らの手法を用いて Reservoir Computing としての Recurrent SNN の教師あり学習を行います.

6.3 Recurrent SNN に正弦波を学習させる

今回は Recurrent SNN のニューロンの活動をデコードしたものが正弦波となるように (すなわち正弦波を教師信号として) 訓練することを目標とします. 先になりますが, 結果は図のようになります.

6.3.1 ネットワークの構造と教師信号

ネットワークの構造は図のようになっています. ネットワークには特別な入力があるわけではなく, 再帰的な入力によって活動が持続しています (膜電位の初期値をランダムにしているため開始時に発火するニューロン[*2]があり, またバイアス電流もあります).

まず, Reservoir ニューロンの数を N とし, 出力の数を N_{out} とします. i 番目のニューロンの入力はバイアス電流を I_{Bias} として,

$$I_i = s_i + I_{\text{Bias}} \tag{6.1}$$

[*2] ここでの「ニューロン」はこれ以後も含め, Reservoir のユニットを指します.