

2021 年度 修士論文

深層学習を用いた ILC 崩壊点検出アルゴリズムの改善

九州大学大学院 理学府 物理学専攻
粒子物理学分野 素粒子実験研究室

後藤 輝一

指導教員 川越 清以

2020 年 11 月 20 日



九州大学
KYUSHU UNIVERSITY

概要

素粒子はそれが従う統計によって二種類に分類され、フェルミ統計に従う粒子をフェルミ粒子、ボース統計に従う粒子をボース粒子と呼ぶ。現時点では存在が知られているフェルミ粒子はクォークとレプトンとに分類される。一方、現時点では知られているボース素粒子には、素粒子間の相互作用を伝達するゲージ粒子と、素粒子に質量を与えるヒッグス機構に関連して現れるヒッグス粒子がある。ゲージ粒子のうち、重力を媒介するとされる重力子は未発見である。素粒子の大きさは分かつておらず、大きさが無い（点粒子）とする理論と、非常に小さいがある大きさを持つとする理論がある。標準模型（標準理論）では素粒子には大きさが無い（点粒子）ものとして扱っており、現時点では実験結果と矛盾が生じていない。ただし、点粒子は空間が最小単位の存在しない無限に分割可能な連続体であることを前提としているが、標準模型で扱うスケールより 15 桁以上小さいスケール（プランク長スケール）においては、空間が連続的であるか離散的であるかは判明していない。離散的である場合には点粒子として扱えない。超弦理論においては全ての素粒子は有限の大きさを持つひもの振動状態であるとされる。我々が普段目にする物質は（微小な、あるいは大きさが無い）素粒子からできているにも関わらず、有限の大きさを持っている。それは、複数の素粒子が運動する有限の領域が、ハドロンや原子などの大きさを持つ粒子を構成することによる。素粒子のうちほとんどのものは、自然界に単独で安定的に存在しているわけではないので、宇宙線の観測や加速器による生成反応により発見・研究された。素粒子の様々な性質を実験で調べ、それを理論的に体系化していくこと、及び理論的に予言される素粒子を実験で探索していくことが、素粒子物理学の研究目的である。[1]

目次

第 1 章	序論	11
1.1	標準模型	11
1.2	国際線形加速器 (ILC) 計画	13
1.3	ILC の物理	14
1.4	ILC の検出器 -International Large Detector (ILD)-	15
1.5	ILC のソフトウェアと事象再構成	16
1.5.1	ソフトウェア	17
1.5.2	飛跡の再構成	17
1.5.3	ジェットの再構成	17
1.6	本研究の目的	18
1.7	本論文の流れ	18
第 2 章	深層学習	19
2.1	機械学習と深層学習	20
2.2	パーセプトロン	21
2.2.1	単純パーセプトロン	21
2.2.2	多層パーセプトロン	22
2.3	ニューラルネットワーク	23
2.3.1	ニューラルネットワークの構造	24
2.3.2	ニューラルネットワークの学習	26
2.3.3	ディープニューラルネットワーク	29
2.4	リカレントニューラルネットワーク	30
2.4.1	リカレントニューラルネットワークの構造と学習	30
2.4.2	リカレントニューラルネットワークの問題点	32
2.4.3	長・短期記憶 (Long Short-Term Memory, LSTM)	33
2.5	Attention	37
2.5.1	エンコーダー・デコーダーモデル	37
2.5.2	Attention	38

2.6	ハイパーパラメータ	41
第3章	崩壊点検出の為のネットワーク	42
3.1	データ	42
3.1.1	データ全体の性質	42
3.1.2	飛跡の情報と前処理	44
3.2	深層学習を用いた崩壊点検出の実現	46
3.3	飛跡対についてのネットワーク	47
3.3.1	ネットワークの構造	48
3.3.2	ネットワークの学習と戦略	48
3.3.3	ネットワークの性能	50
3.4	任意の数の飛跡についてのネットワーク	50
3.4.1	ネットワークの構造	50
3.4.2	ネットワークの詳細な構造	51
3.4.3	ネットワークの学習と戦略	51
3.4.4	ネットワークの性能	52
第4章	深層学習を用いた崩壊点検出	55
4.1	崩壊点検出アルゴリズム	55
4.1.1	飛跡対についてのネットワークの拡張	55
4.1.2	Primary Vertex の再構成	55
4.1.3	Secondary Vertex の再構成	55
4.2	深層学習を用いた崩壊点検出の性能	55
4.2.1	時間計測	55
4.2.2	エネルギーの変化	55
第5章	現行の手法との比較	56
5.1	崩壊点検出単体での比較	56
5.2	C++ での推論	56
5.3	詳細な比較と評価	56
第6章	結論と今後の展望	57
付録A	ソースコード	59
A.1	飛跡対についてのネットワーク	59
A.2	任意の数の飛跡についてのネットワーク	59
A.3	崩壊点の再構成	59
付録B	分散深層学習	60

付録 C SiW-ECAL	61
参考文献	62

図目次

1.1	標準模型の素粒子	12
1.2	国際線形加速器 (ILC) の外観 [2]	13
1.3	ILC 計画の今後 [3]	14
1.4	重心系エネルギーと断面積の関係 [4]	15
1.5	外観 [2]	16
1.6	縦断面 [6]	16
1.7	International Large Detector (ILD)	16
1.8	深層学習によるジェットの再構成	18
2.1	機械学習の中の深層学習の位置付け	21
2.2	単純パーセプトロン	22
2.3	ヘヴィサイドの階段関数	22
2.4	多層パーセプトロン	23
2.5	活性化関数	25
2.6	ニューラルネットワーク	25
2.7	リカレントニューラルネットワーク	31
2.8	リカレントニューラルネットワークの重み	31
2.9	リカレントニューラルネットワークの出力方法	32
2.10	LSTM の流れ	34
2.11	単体の LSTM	34
2.12	LSTM の各ゲートについての図解	36
2.13	Stacked LSTM	36
2.14	双方向 LSTM	37
2.15	LSTM によるエンコーダー・デコーダーモデル	38
2.16	Attention と LSTM によるエンコーダー・デコーダーモデル	39
2.17	Additive Attention と Dot-Product Attention	40
3.1	終状態 $b\bar{b}$ での典型的な崩壊点の例	44
3.2	トラック・パラメータ [34]	44

3.3	変数の分布の例	45
3.4	終状態 $b\bar{b}$ での分類クラスの定義	47
3.5	飛跡対についてのネットワークの概略図	48
3.6	$b\bar{b}$ データセットについて崩壊点種予測モデルにおける各種パラメーターの出力	49
3.7	独自のリカレントニューラルネットワーク構造を用いた崩壊点の生成	51
3.8	Attention を用いたエンコーダー・デコーダーモデルへの拡張	52
3.9	エンコーダー・デコーダーモデルにおける各種パラメーターの出力	53
3.10	自作リカレントニューラルネットワークの構造	54
3.11	Attention を組み込んだ自作リカレントニューラルネットワークの構造	54

表目次

1.1	ILD サブディテクターの詳細なパラメータ (バレル) [6]	16
1.2	ILD サブディテクターの詳細なパラメータ (エンドキャップ) [6]	17
3.1	データサンプルの事象数と用途	43
3.2	終状態と分類クラス名	47

第 1 章

序論

本章では、まず 1.1 節で素粒子を記述する為の理論である、標準模型 (Standard Model, SM) について解説する。次に、この標準模型や標準模型を超える物理 (Physics beyond the Standard Model, BSM) を探索するための国際線形加速器 (International Linear Collider, ILC) 計画についての説明を 1.2 節で行う。また、この ILC で観測できる主な物理現象については 1.3 節で述べる。ILC の検出器は、International Large Detector (ILD) と Silicon Detector (SiD) の二つが検討されている。本研究は ILD に関する研究である為、ILD について 1.4 節で簡単に説明する。ただし、本研究の基本的な構想はそのような検出器に寄らず使用できる。

加速器実験では、取得したデータをそのまま物理解析に使用することは出来ず、適切な処理をする必要があり、これを事象再構成 (Event Reconstruction) という。1.5 節では、ILC におけるこれら再構成手法やソフトウェアについて説明し、最後に本研究の目的について 1.6 節で述べ、本論文の序論とする。

1.1 標準模型

宇宙の誕生や、生物の発生と同様に、物質の起源は人類の根元的な問いの一つである。そのような物質の素となる粒子のことを素粒子といい、その素粒子の振る舞いを記述する理論を標準模型という。この標準模型は 20 世紀から多数の物理学者によって構築され、今日に至るまで様々な実験によって、非常に良く確かめられている。標準模型によると、素粒子はスピン半整数のフェルミ粒子とスピン整数のボース粒子に分類される。

フェルミ粒子は全てスピン $1/2$ の粒子で構成され、更に、陽子や中性子などを構成するクォークと電子やニュートリノなどのレプトンに分けられる。クォークは電荷が $+2/3$ のアップクォーク系列と $-1/3$ のダウントクォーク系列に、レプトンは電荷が -1 の荷電レプトンと中性電荷の中性レプトンに細分される。また、それぞれ世代と呼ばれるものを構成し、現在合計で 3 つの世代が確認されている。クォークの場合はアップクォーク u 、チャームクォーク c 、トップクォーク t 、ダウントクォーク d 、ストレンジクォーク s 、ボトムクォーク b が存在し、こ

これらの系列や世代間のクォークの違いをフレーバーと呼んでいる。レプトンの場合は荷電レプトンとして、電子 e^- 、ミュー粒子 μ^- 、タウ粒子 τ^- 、中性レプトンとして、電子ニュートリノ ν_e 、ミューニュートリノ ν_μ 、タウニュートリノ ν_τ が存在している。

ボース粒子は基本的な 4 つの力である、強い相互作用、弱い相互作用、電磁相互作用、重力相互作用の内、重力相互作用を除いた 3 つの力をそれぞれ媒介するスピン 1 のゲージ粒子と、対称性を破り素粒子に質量を与えるスピン 0、中性電荷のヒッグス粒子 H で構成される。電磁相互作用を媒介する粒子として、中性電荷の光子 γ 、強い相互作用を媒介する粒子として、中性電荷のグルーオン g 、弱い相互作用を媒介する粒子として、電荷 ± 1 の W ボソン W^\pm 、中性電荷の Z ボソン Z が存在する。

粒子には、質量やスピンが等しく、電荷の正負が反転した反粒子が存在し、基本的にそれぞれの粒子に $-$ や電荷をつけて記述される。 $(\bar{u}, \bar{c}, \bar{t}, \bar{d}, \bar{s}, \bar{b}, e^+, \mu^+, \tau^+, \bar{\nu}_e, \bar{\nu}_\mu, \bar{\nu}_\tau)$ これらの反粒子と通常の粒子を衝突させると、質量が全てエネルギーへと変換される、対消滅を起こす。一方、これらの粒子対以上エネルギーを与えた場合は対生成が起こり、これらの粒子対が生成される。

これらのボース粒子、フェルミ粒子は標準模型の素粒子と呼ばれ、一般に図 1.1 のように纏められている。

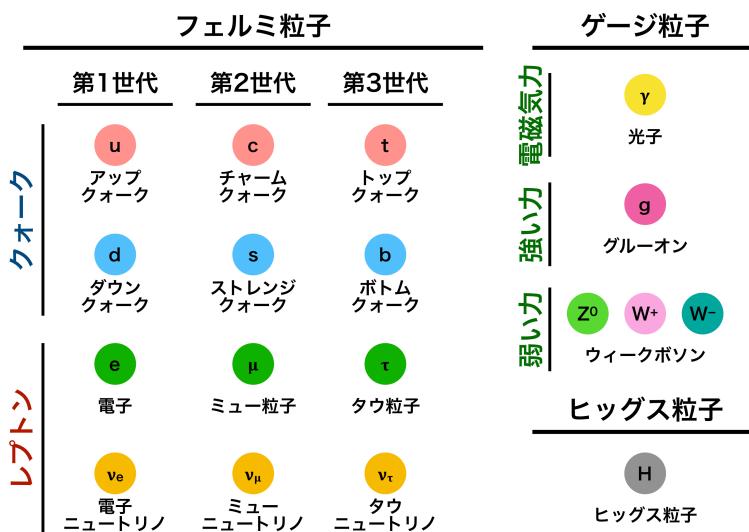


図 1.1: 標準模型の素粒子

前述したように、標準模型は様々な実験で非常によく確かめられているが、ダークマターをはじめとするいくつかの物理現象を説明できておらず、現在は様々な実験によって、BSM の探索が行われている。次節の ILC 計画はそのような試みの一つである。

1.2 国際線形加速器 (ILC) 計画

ILC 計画とは、日本の東北にある北上山地に全長 20.5km の国際線形加速器 (ILC) を建設する計画である。 (図 1.2) ILC 計画は国際共同研究であり、2013 年に出版された The Technical Design Report (TDR) には 2400 人の研究者、48 の国と 392 の研究機関と大学のグループが著名している。この ILC 実現の為の技術開発はリニアコライダーコラボレーション (The Linear Collider Collaboration, LCC) によって推進され、LCC の活動は国際将来加速器委員会 (The International Committee for Future Accelerator, ICFA) の下、リニアコライダー国際推進委員会 (Linear Collider Board, LCB) によって監督されている。現在 ILC 計画は準備段階へ向けて計画が進められており、日本の ILC 準備研究所 (ILC Pre-Lab) の為の準備として ICFA は ILC の国際推進チーム (International Development Team, IDT) の設立を承認した。今後は LCC や LCB に代わり、この ILC 国際推進チームが ILC 計画の推進を行なっていく予定である。ILC 計画の今後の流れは図 1.3 に示している。

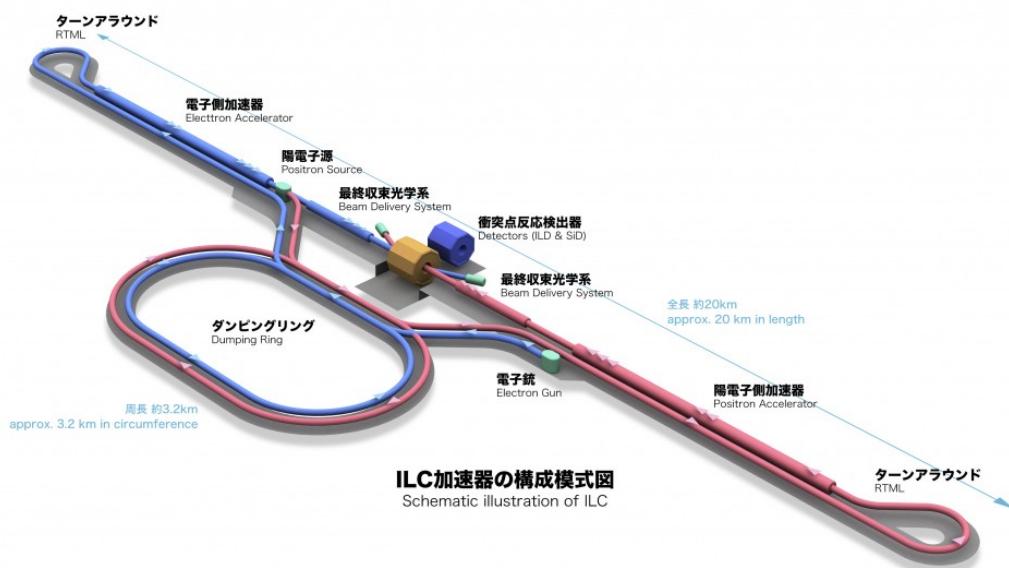


図 1.2: 国際線形加速器 (ILC) の外観 [2]

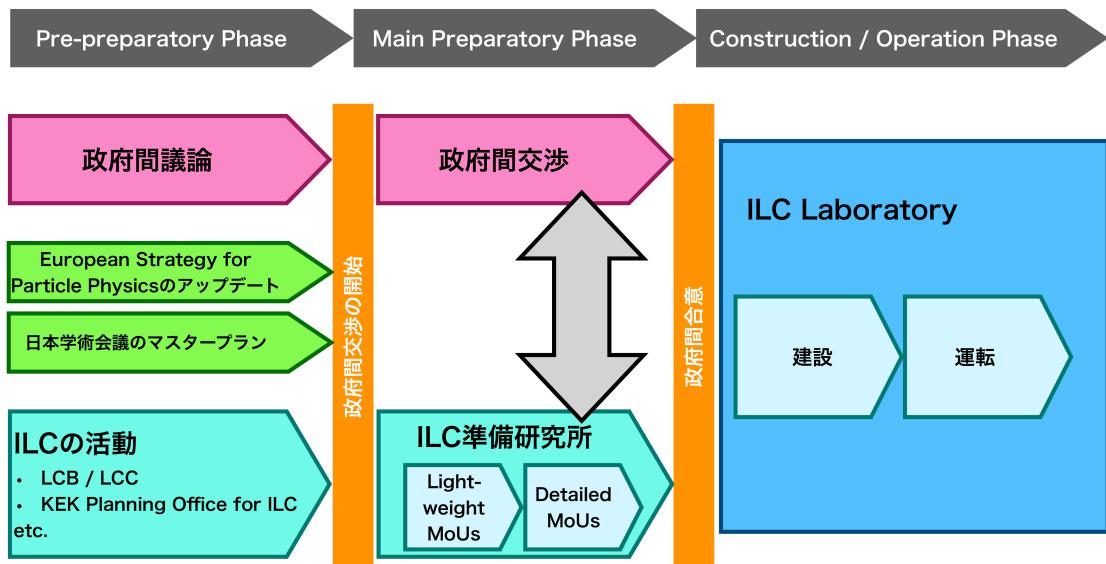


図 1.3: ILC 計画の今後 [3]

1.3 ILC の物理

ヒッグス粒子が 2012 年に欧州原子核研究機構 (CERN) の大型ハドロン衝突型加速器 (Large Hadron Collider, LHC) で発見されて以降、ヒッグス粒子の性質について、より詳細な調査が行われている。ヒッグス粒子は標準模型の中で、電弱相互作用の対称性を破り、素粒子に質量を与える役割を担っており、また、質量に結合するという特徴を持っている。このような振る舞いからヒッグス粒子の性質は標準模型によって詳細に決定される為、BSM によって標準模型との差異が生じた場合、ヒッグス粒子はその影響を受けると予想されている。特にヒッグス粒子と他の粒子との結合定数の変化は、そのような仮定する BSM の模型の違いによって異なることが示唆されている。

ILC はこのヒッグス粒子の性質を詳細に調べる為のヒッグスファクトリーとしての役割を期待されている。LHC が陽子-陽子衝突であるのに対し、ILC は電子-陽電子を衝突させる加速器である。したがって、粒子反粒子の関係となっており、目的とする事象に対しエネルギーをより効率的に使うことができる。また、電子-陽電子は陽子同士の衝突と異なり、背景事象が少ないという特徴を持っている。

ILC は $e^+e^- \rightarrow Zh$ 事象の反応断面積が最大となる重心系エネルギー $\sqrt{s} = 250$ GeV での運転開始 (ILC250) を予定している。(図 1.4) また、ILC には様々な物理目標を達成する為に多数のアップグレードオプションが存在し、重心系エネルギーについてはメインリニアックを延長することで 1 TeV までの拡張が可能である。

$e^+e^- \rightarrow Zh$ 事象は Z 粒子の識別をすることによって、ヒッグスの崩壊モードに寄らず事象を選別できる (リコイル) という点で非常に重要である。また、背景事象である $e^+e^- \rightarrow Z\gamma$

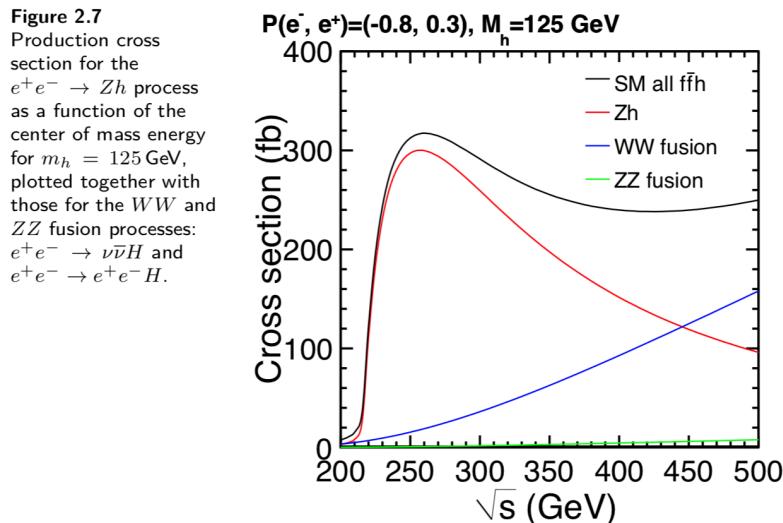


図 1.4: 重心系エネルギーと断面積の関係 [4]

や $e^+e^- \rightarrow ZZ$ に関してはよく理解されており、電弱相互作用の計算によって 0.1 % 程度に抑えることができる。[5] したがって、 $e^+e^- \rightarrow Zh$ 事象の全断面積を得ることができ、絶対正規化されたヒッグス粒子の結合定数やヒッグス粒子のエキゾチック崩壊についての測定が可能である。

$e^+e^- \rightarrow Zh$ 事象での終状態の Z 粒子は、レプトン対またはクォーク対に崩壊する。レプトン対にはおよそ 30% 程度の割合で崩壊し、クォーク対には残りの 70% 程度の割合で崩壊する為、統計量を大きくするという点でクォーク対をより精度よく識別することは非常に重要である。これらのクォーク対はエネルギー効率のために、それぞれ真空中でクォークの粒子反粒子対を生成・結合しハドロンとなる。この過程で生成されたクォークも同様にハドロンを形成するため、初めのクォーク対のそれぞれの進行方向には多数のハドロン粒子が生成されることとなる。これをジェットといい、 Z 粒子を始めとする様々な粒子のクォーク対への崩壊は、このジェットを用いて識別される。そのようなジェットの再構成については、1.5.3 項にて説明する。

1.4 ILC の検出器 -International Large Detector (ILD)-

ILC では二つの検出器が検討されており、ILD(図 1.5) はその一つである。ILD はヒッグス粒子や電弱相互作用の物理からの要求値を満たすように設計され、また後述する Particle Flow (1.5.2 項) によって最適化されている。また、様々なサブディテクターによって構成され、ビームの衝突点(図 1.6 の右下) を包む様に内側から順に、Vertex Detector (VTX)、Silicon Internal Tracker (SIT)、Time Projection Chamber (TPC)、Electromagnetic Calorimeter (ECAL)、Hadron Calorimeter (HCAL)、Iron Yoke (Muon) が並んでいる。HCAL と Iron Yoke の間には Solenoid Coil があり、3.5T の磁場をかけている。VTX や SIT、TPC を用いて荷電粒子の飛跡を測定し、ECAL によって電子や光子などの粒子のエネ

ルギーを、HCAL によってハドロン粒子のエネルギーを測定する。衝突点の前方方向には、Forward Tracking Detector (FTD)、Luminosity Calorimeter (LumiCAL)、LHCAL、Beam Calorimeter (BeamCAL) が並んでいる。それぞれの技術的な詳細については表 1.1、1.2 にまとめる。

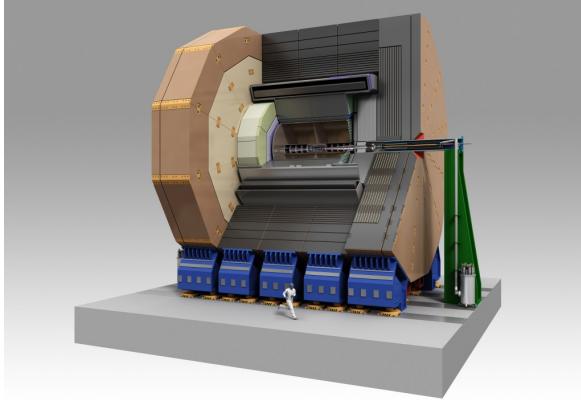


図 1.5: 外観 [2]

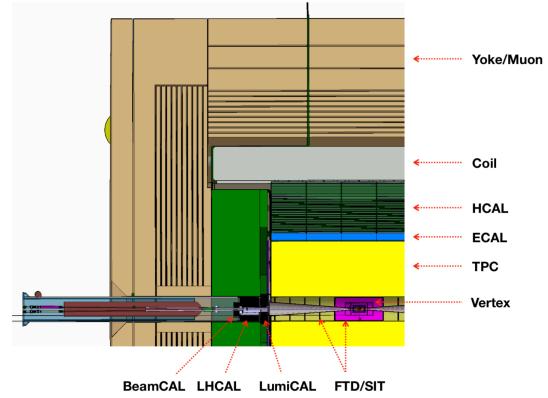


図 1.6: 縦断面 [6]

図 1.7: International Large Detector (ILD)

	r_{in} [mm]	r_{out} [mm]	z_{max} [mm]	要素技術
VTX	16	60	125	シリコンピクセルセンサー
SIT	153	303	644	シリコンピクセルセンサー
TPC	329	1770	2350	マイクロパターンガス検出器
SET	1773	1776	2300	シリコンストリップセンサー
ECAL	1805	2028	2350	吸収層：タングステン センサー：シリコン/シンチレーター
HCAL	2058	3345	2350	吸収層：スチール センサー：シンチレーター/RPC ガス
Coil	3425	4175	3872	
Muon	4450	7755	4047	センサー：シンチレーター

表 1.1: ILD サブディテクターの詳細なパラメータ (バレル) [6]

1.5 ILC のソフトウェアと事象再構成

ここでは ILC で使用されるソフトウェアと事象再構成について述べる。ILC のソフトウェアは iLCSoft[7] と呼ばれるソフトウェアエコシステムにまとめられている。ILC における事象再構成は、トラッキングや Particle Flow といった 1.5.2. 粒子の再構成と、更にそれによって再構成された粒子を使いジェットを再構成する 1.5.3. ジェットの再構成に分けられる。ILC ではジェットの再構成は崩壊点検出、ジェットクラスタリング、フレーバータギングという行程に分けられる。これらジェットの再構成は iLCSoft 内の LCFIPlus[8] によって行われる。

	z_{min} [mm]	z_{max} [mm]	r_{in} [mm]	r_{out} [mm]	要素技術
FTD	220	371		153	シリコンピクセルセンサー
	645	2212		300	シリコンストリップセンサー
ECAL	2411	2635	250	2096	吸収層：タンゲステン センサー：シリコン/シンチレーター
	2650	3937	350	3226	吸収層：スチール センサー：シンチレーター/RPC ガス
Muon	4072	6712	350	7716	センサー：シンチレーター
BeamCAL	3115	3315	18	140	吸収層：タンゲステン GaAs 読み出し
	2412	2541	84	194	吸収層：タンゲステン センサー：シリコン
LumiCAL	2680	3160	130	315	吸収層：タンゲステン
LHCAL					

表 1.2: ILD サブディテクターの詳細なパラメータ (エンドキャップ) [6]

ている。

1.5.1 ソフトウェア

書く事

LCIO・Marlin・DD4hep

1.5.2 飛跡の再構成

書く事

トラッキング・Particle Flow

1.5.3 ジェットの再構成

書く事

トラッキング・Particle Flow を経て再構成された粒子の飛跡を用いて、ジェットの再構成が行われる。粒子が飛んで—— primary vertex と secondary vertex (崩壊点) を形成する

secondary vertex はジェットの種であり、これを検出するアルゴリズムを崩壊点検出といい、その様にして検出された secondary vertex を用いて、中性電荷を合わせた飛跡について、どのジェットに属するかクラスタリングを行うアルゴリズムをジェットクラスタリングという。また、secondary vertex によって——をフレーバータギングという。以上がジェットの再

構成である。ここでは、LCFIPlus[9]でのジェット再構成手法についてまとめる。

まず、Primary Vertex

次に、Secondary Vertex

ジェットクラスタリング・Durham

フレーバータギング・Boosted Decision Trees (BDTs)

1.6 本研究の目的

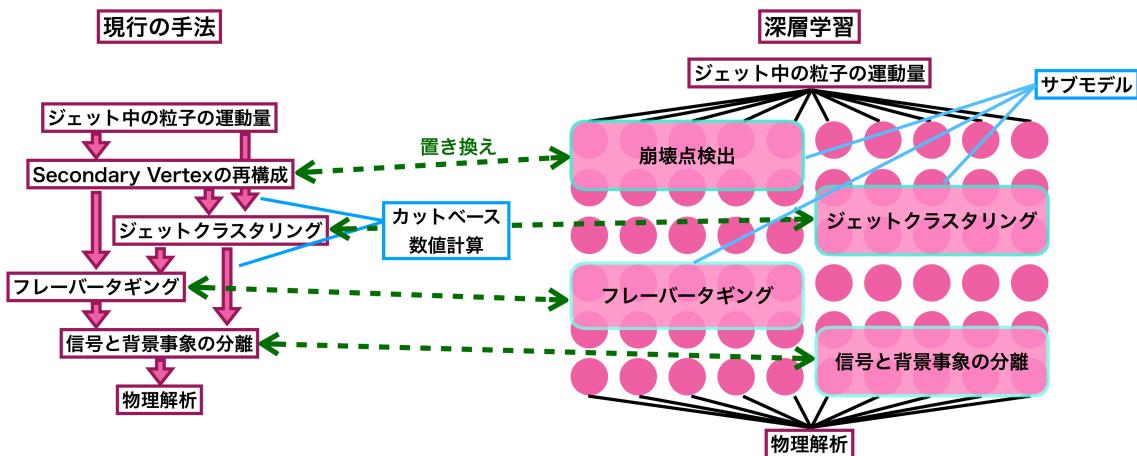


図 1.8: 深層学習によるジェットの再構成

1.7 本論文の流れ

本章と2章は本論文の導入である。2章では本論文の核となる技術である深層学習について、それぞれ解説を行う。ただし深層学習に際しての種々のテクニックについての説明は2章では行わず、3章で述べる。また、具体的な実装に関しても同様に2章では解説せず、付録Aにまとめる事とする。

3章と4章、5章は本論文の本題である。3章では本研究で使用するデータと私が作成した深層学習のネットワークについて、4章では3章で作成したネットワークを用いてどの様にして崩壊点検出を実現するかについて、5章ではその様にして得られた崩壊点検出と、LCFIPlusで使用されている現行の崩壊点検出との比較について、それぞれ記載する。

第 2 章

深層学習

本章では、深層学習 (Deep Learning, DL) について解説するが、ここでは本論文を理解するに当たって必要な知識や技術を最小限の解説を行う。^{*1}

物理学を学ぶ者にとって、深層学習はあまり馴染みのない技術であるため、まず 2.1 節で導入として機械学習 (Machine Learning, ML) と深層学習の概要について簡単に述べる。

2.2 節では、深層学習を理解する上で、非常に重要なパーセプトロンというネットワークを紹介する。パーセプトロンは、後述するニューラルネットワークの先駆けとなる技術である。

続く 2.3 節では、深層学習の基礎技術であるニューラルネットワークについて解説を行う。主に、2.3.1 項でニューラルネットワークを構築するために必要な計算手順について説明し、2.3.2 項でニューラルネットワークの学習に関して重要な技術について説明する。

深層学習は、2.3 節までの基礎的な技術を使用するだけでも様々な問題を解くことができるが、更に扱うデータや課題の性質によって、応用的な使い方が考えられている。2.4 節や 2.5 節ではそのような深層学習の応用技術について述べる。

2.4 節では、系列データを取り扱うためのリカレントニューラルネットワーク (Recurrent Neural Network, RNN) について解説する。2.4.1 項で基本的なリカレントニューラルネットワークの構造や学習について説明し、リカレントニューラルネットワークが抱える問題と更なる改善について、2.4.2 項と 2.4.3 項で紹介する。

2.5 節では、近年注目されている注意機構 (Attention) と呼ばれる技術について説明する。Attention はリカレントニューラルネットワークと同様に系列データを取り扱うための技術であり、ここでは特に Self-Attention についてのみ述べることとする。2.5.1 項にて、エンコーダー・デコーダーモデルにおいてリカレントニューラルネットワークが抱える問題と Attention を用いた解決策を紹介し、2.5.2 項にて、具体的な Attention の計算について解説する。

最後に 2.6 節にて、ニューラルネットワークが持つハイパラメータについてまとめる。

^{*1} 機械学習や深層学習は既に様々な入門書 [10, 11, 12] によって、理論から実装まで分かりやすく説明されている

2.1 機械学習と深層学習

深層学習とは、機械学習の技術の一つである。本節ではまず、この機械学習について簡単に説明し、その後、機械学習における深層学習の位置付けを述べる。

機械学習とは、データに現れるパターンや統計情報を計算機（学習器）に「学習」させることによって、逐一プログラミングをすることなく未知の問題に対応させる為の技術である。これは人間の持つ知性を機械に実現する、人工知能（Artificial Intelligence, AI）に関する研究の一分野であると言える。このような研究は、1956年のダートマス会議[13]から始まり、現在は第三期のAIブームと言われている。機械学習は、機械（計算機）が独自に未知の問題を解く為の技術や手法の総称であるが、問題に対するアプローチの仕方によって、教師あり学習、教師なし学習、強化学習などに分類することができる。（図2.1）

- 教師あり学習

教師あり学習とは、訓練データ（Training data）と呼ばれる正解がラベル付けされたデータを用い、学習器の出力を正解に近付けるように学習器を更新していく手法である。主にクラス分類を行う分類問題や、連続値を予測する回帰問題などの問題を解くことができる。具体的な例としてサポートベクターマシン（Support Vector Machine, SVM[14, 15]）や決定木などが用いられる。

- 教師なし学習

教師なし学習とは、訓練データを用いず、データの持つ数学モデルや構造を抽出する技術である。主にクラスタリングや次元圧縮などに使用され、代表的な手法は、k平均法や主成分分析などである。

- 強化学習

強化学習とは、環境とのやり取りから報酬を受け取り、エージェントを構築していく手法である。学習は報酬を最大化するように進み、教師あり学習の一分野のようにみなす事も出来るが、強化学習は一連の行動に対する報酬を考慮する点で異なる。強化学習は様々な分野で使用されているが、主に長期的な戦略が必要となるゲームなどの領域で用いられている。

深層学習は、このような機械学習の中で、基本的には、回帰問題や分類問題などを解く教師あり学習に分類される。しかし近年では、半教師あり学習やディープクラスタリング、深層強化学習といった様々な技術的応用が提案されている。次節以降では、この深層学習の基礎技術について紹介する。

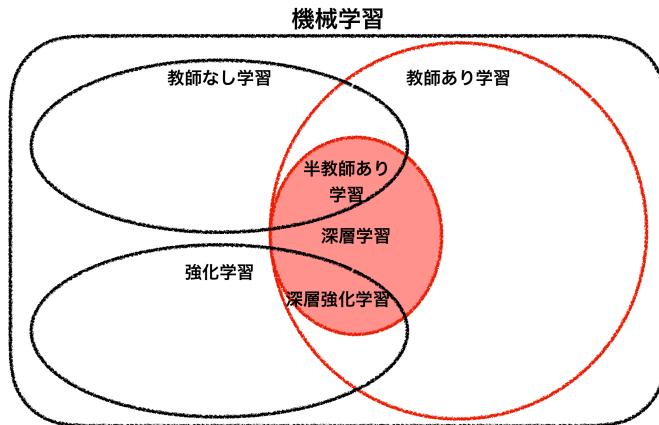


図 2.1: 機械学習の中の深層学習の位置付け

2.2 パーセプトロン

パーセプトロン (Perceptron) は深層学習の基礎となる技術であり、1958 年、Rosenblatt によって提案された [16]。ここでは、このパーセプトロンについて解説することで、次節のニューラルネットワークへの導入とする。

2.2.1 単純パーセプトロン

パーセプトロンとは、情報を伝達するネットワークである。ここでいうネットワークとは、ある情報を受け取り、それを後方へ伝達するような構造のことを言うものとする。まず、最も簡単なパーセプトロンとして、図 2.2 のような構造を考える。図 2.2 は、二つの入力 x_1, x_2 を受け取り、一つの出力 y を行なっているネットワークである。このような入力や出力の数や入力や出力そのものの事をノードやニューロンという。また、図 2.2 のように、ただ入力と出力のみを持っているパーセプトロンを特に単純パーセプトロン (Simple Perceptron) という。

単純パーセプトロンの情報処理は、簡単な計算で定義される。出力 y は、 x_1, x_2 とそれぞれの重み w_1, w_2 を用いて、

$$\begin{aligned} y &= h(a) \\ a &= w_1x_1 + w_2x_2 \end{aligned} \tag{2.1}$$

と計算される。ここで、出力 y は関数 h によって変換されている。このような関数を活性化関数 (Activation function) という。特に単純パーセプトロンでは、活性化関数 h としてヘヴィサイドの階段関数 (図 2.3) を用いる。

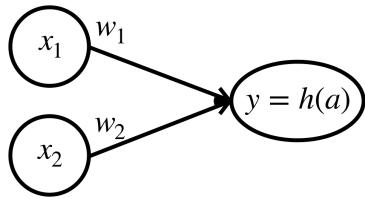


図 2.2: 単純パーセプトロン

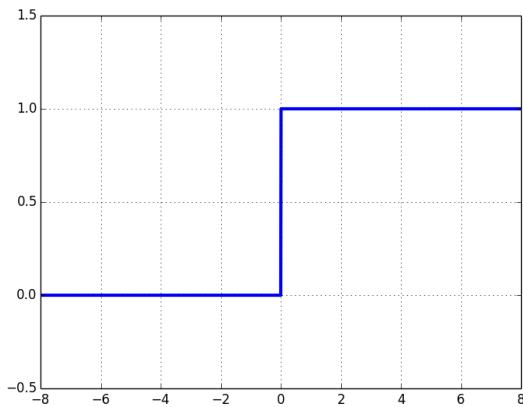


図 2.3: ヘヴィサイドの階段関数

その閾値を θ とすると、出力は更に、

$$y = h(a) = \begin{cases} 0 & (a = w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (a = w_1x_1 + w_2x_2 > \theta) \end{cases} \quad (2.2)$$

と書ける。この時、単純パーセプトロンはある一定値 θ までは”0”、それ以上であれば”1”を返す二値信号のネットワークであると考えることが出来る。^{*2} パーセプトロンやニューラルネットワークにおいて、学習可能なパラメータは重み w_1, w_2 であり、これら重みを更新していく操作を学習 (トレーニング, Training) という。

2.2.2 多層パーセプトロン

単純パーセプトロンは線形な問題を解く事しか出来なかつたが、重ねることで非線形に対応できるという点で非常に高い発展性を持っていた [17]。そのように単純パーセプトロンを重ねたネットワークの事を多層パーセプトロン (Multi Layer Perceptron, MLP) という。多層パーセプトロンは図 2.4 のように表現出来る。

^{*2} a が閾値 θ を超えた場合「ニューロンが発火した」表現することがある。

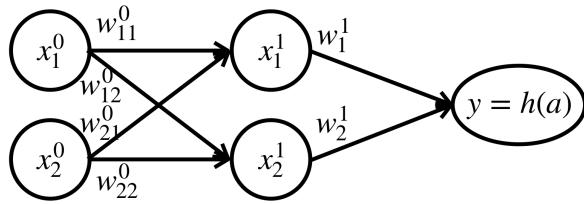


図 2.4: 多層パーセプトロン

多層パーセプトロンは単純パーセプトロンとは異なり、入力、出力以外に、中間層（隠れ層）を持っている。ここでは、その中間層を x_1^1, x_2^1 と置くと、中間層 x_1^1, x_2^1 は単純パーセプトロンと同様に入力 x_1^0, x_2^0 とそれぞれの重み $w_{11}^0, w_{12}^0, w_{21}^0, w_{22}^0$ を用いて

$$\begin{aligned} x_1^1 &= w_{11}^0 x_1^0 + w_{21}^0 x_2^0 \\ x_2^1 &= w_{12}^0 x_1^0 + w_{22}^0 x_2^0 \end{aligned} \quad (2.3)$$

と計算でき、また出力 y についても、 x_1^1, x_2^1 とそれぞれの重み w_1^1, w_2^1 を用いて、

$$\begin{aligned} y &= h(a) \\ a &= w_1^1 x_1^1 + w_2^1 x_2^1 \end{aligned} \quad (2.4)$$

となる。

以後、特に断らない場合はあるノード x 、重み w について、層の深さ、前後のノードを次のように表現する。

$$\begin{aligned} x^{(\text{層の深さ})}_{(\text{ノード番号})} \\ w^{(\text{層の深さ})}_{(\text{前のノード番号}) (\text{後ろのノード番号})} \end{aligned} \quad (2.5)$$

多層パーセプトロンは、学習の手法や層を重ねるに連れて重みが更新出来なくなる勾配消失問題など様々な課題を抱えていた。次節ではこれらの問題を後述する誤差逆伝播法 (Backpropagation) や活性化関数によって解決したニューラルネットワークについて解説する。^{*3}

2.3 ニューラルネットワーク

本節ではニューラルネットワーク (Neural Network, NN) について解説を行うが、その基礎的な理論の概説と用語の説明に留める。また、ニューラルネットワークの実装については、現

^{*3} 多層パーセプトロンは本来、階段関数を使用したネットワークを指す言葉であるが、近年は後述のニューラルネットワークなどを広く指す言葉として使用されているため、注意が必要である。

在様々なフレームワーク [18, 19, 20, 21] があり、それぞれで実装の仕方が異なっている。本研究における実装は主に tensorflow-keras を用いて行なった。具体的なコードに関しては付録 A にまとめている。

ニューラルネットワークに関する技術は「ニューラルネットワークの構造」についてと「ニューラルネットワークの学習」についてに大きく分けられると考えている。前者は主に入力から出力までのネットワークの構築を、後者は構築されたネットワークの重み更新についての技術である。ただし、前者、後者に問わず、ニューラルネットワークにおけるユーザーレベルのテクニックは経験則によるものが多いいため、本節では述べず、3章で適宜解説を行うものとする。

2.3.1 ニューラルネットワークの構造

ニューラルネットワークは様々な技術によって支えられているが、その基本構造は前節の多層パーセプトロンと全く同じである。ニューラルネットワークと多層パーセプトロンとの大きな構造の違いは活性化関数である。ニューラルネットワークでは様々な活性化関数が提案されており^{*4}、これが勾配消失問題を解消する鍵となっている。活性化関数は重み更新のために微分可能な関数である必要があるが、どのような関数を選ぶかはユーザーに委ねられている。勾配消失や重み更新についての詳しい解説は 2.3.2 節で行う。以下に活性化関数の例を示す。(図 2.5)

- 階段関数

$$h(a) = \begin{cases} 0 & (a \leq \theta) \\ 1 & (a > \theta) \end{cases} \quad (2.6)$$

- シグモイド関数

$$h(a) = \frac{1}{1 + \exp(-a)} \quad (2.7)$$

- tanh 関数

$$h(a) = \tanh(a) \quad (2.8)$$

- ReLU (Rectified Linear Unit, ランプ) 関数 [?]

$$h(a) = \begin{cases} 0 & (a \leq \theta) \\ a & (a > \theta) \end{cases} \quad (2.9)$$

一般に、ニューラルネットワークは多層パーセプトロンと同様の構造であるので、図 2.6 のように表現出来る。

ここで、中間層 x_1^1 は入力 x_1^0, x_2^0, x_3^0 とそれぞれの重み $w_{11}^0, w_{21}^0, w_{31}^0$ を用いて、

$$\begin{aligned} x_1^1 &= h(a_1^1) \\ a_1^1 &= w_{11}^0 x_1^0 + w_{21}^0 x_2^0 + w_{31}^0 x_3^0 + b_1^0 \end{aligned} \quad (2.10)$$

^{*4} 多層パーセプトロンはニューラルネットワークの内、活性化関数に階段関数を使った特別なネットワークであると再定義できる。

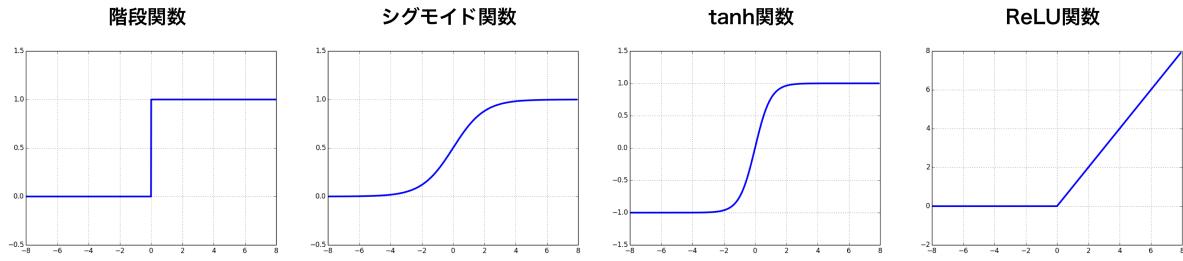


図 2.5: 活性化関数

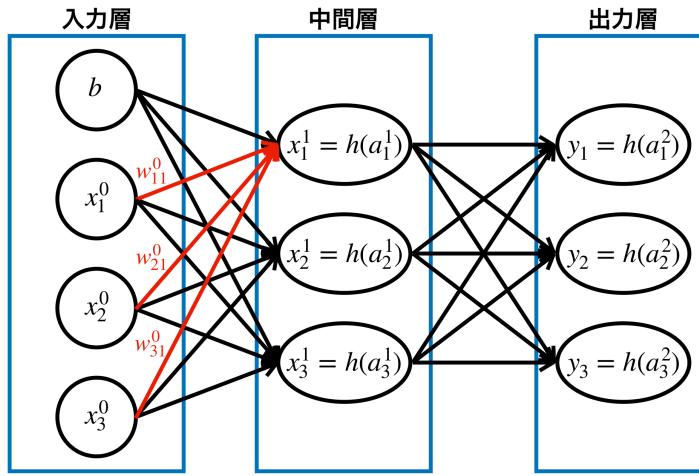


図 2.6: ニューラルネットワーク

と計算できる。また、バイアスとして b を導入している。これはパーセプトロンの閾値 θ に対応している。中間層 x_2^1, x_3^1 についても同様に、

$$\begin{aligned} x_2^1 &= h(a_2^1) \\ a_2^1 &= w_{12}^0 x_1^0 + w_{22}^0 x_2^0 + w_{32}^0 x_3^0 + b_2^0 \\ x_3^1 &= h(a_3^1) \\ a_3^1 &= w_{13}^0 x_1^0 + w_{23}^0 x_2^0 + w_{33}^0 x_3^0 + b_3^0 \end{aligned} \tag{2.11}$$

と書ける。

また、これら x_1^1, x_2^1, x_3^1 の計算は行列とベクトルを用いて、より簡潔に表現できる。

$$\begin{aligned} \mathbf{x}^1 &= \begin{pmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \end{pmatrix} = h(\mathbf{a}^1) \\ \mathbf{a}^1 &= \begin{pmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \end{pmatrix} = W^0 \mathbf{x}^0 + \mathbf{b}^0 = \begin{pmatrix} w_{11}^0 & w_{21}^0 & w_{31}^0 \\ w_{12}^0 & w_{22}^0 & w_{32}^0 \\ w_{13}^0 & w_{23}^0 & w_{33}^0 \end{pmatrix} \begin{pmatrix} x_1^0 \\ x_2^0 \\ x_3^0 \end{pmatrix} + \begin{pmatrix} b_1^0 \\ b_2^0 \\ b_3^0 \end{pmatrix} \end{aligned} \quad (2.12)$$

行列 W^0 とベクトル \mathbf{b}^0 は学習可能な重みであり、得られた新たな状態ベクトル \mathbf{a}^1 は微分可能な任意の活性化関数 h によって、中間層 \mathbf{x}^1 へと変換される。以下、これを繰り返すことによって、ネットワークは構築されている。

出力層における活性化関数は一般に回帰問題では恒等関数を、分類問題ではソフトマックス関数と呼ばれる関数を使用する。回帰問題において、最終的な出力は数値（連続値）であるため、恒等関数によって変換を行わず、そのまま出力することが一般的である。

$$y_k = h(a_k^2) = a_k^2 \quad (2.13)$$

一方で、分類問題では、最終的な出力は分類されたクラスとなるため、以下のようない softmax 関数を使用する。

$$y_k = h(a_k^2) = \frac{\exp(a_k^2)}{\sum_{i=1}^N \exp(a_i^2)} \quad (2.14)$$

この softmax 関数は、分母が総和、分子がその一要素の形をしており、 y_k を k について足し合わせると 1 になることがわかる。このことから、出力 y_k は k 番目のクラスについての確率として解釈でき、分類問題において、どのクラスにどの程度該当するかを表現することに相当している。

前述したように、これは最も基本的なニューラルネットワークであり、このような全結合（Fully connected, Dense）な層を重ねたネットワークを順伝播型（フィードフォワード）ニューラルネットワーク（Feedforward Neural Network）という。

2.3.2 ニューラルネットワークの学習

教師あり学習であるニューラルネットワークにおける学習は、損失関数（コスト関数, Loss function）を最小化するように、重みを更新していくことで行われる。損失関数とは、訓練データの正解ラベルとネットワークの出力がどの程度離れているかを計算するための関数である。この損失関数は取り組む問題や訓練データの性質によって適切に選択する必要がある。ここでは、よく使用される損失関数として以下の二つを挙げる。

- 交差エントロピー誤差

$$L = - \sum_k^N t_k \log(y_k) \quad (2.15)$$

- 平均二乗誤差

$$L = \frac{1}{N} \sum_k^N (t_k - y_k)^2 \quad (2.16)$$

t_k, y_k はそれぞれ k 番目の正解ラベルとクラスの出力 (確率や値) を示している。分類問題については交差エントロピー誤差が、回帰問題については平均二乗誤差が主に使用される。

分類問題において、正解ラベル t は、あるクラスに関して 0 か 1 かのベクトル (one-hot) で表現されることが一般的である。例えば、赤、青、緑について分類を行う場合 (3 クラス分類という)、赤を $(1, 0, 0)$ 、青を $(0, 1, 0)$ 、緑を $(0, 0, 1)$ と定義する。また、ネットワークの出力 y はどのクラスに属するかの確率となっている。例えば、赤、青、緑がそれぞれ 80%、10%、10% の場合は出力 y は $(0.8, 0.1, 0.1)$ と書ける。したがって、正解ラベルを赤とすると損失関数 L は

$$\begin{aligned} L &= - \sum_k^3 t_k \log(y_k) \\ &= -t_1 \log y_1 - t_2 \log y_2 - t_3 \log y_3 \\ &= -1 \cdot \log 0.8 \\ &= 0.22314... \end{aligned} \quad (2.17)$$

と計算される。

回帰問題において、出力 y 、正解ラベル t は共に連続値であるため、平均二乗誤差のような二つの差を用いる損失関数が一般的である。

前述したように、ネットワークの学習はこの損失関数を最小化するように進む。今、損失関数は変数 y_k の関数で表現出来ており、このような関数の最小値を求めるためには、単に変数 y_k を用いて偏微分を行い勾配を求めれば良い。計算機において、このような勾配を求め、徐々に関数を最小化していく手法を勾配降下法 (Gradient Descent Method) という。勾配降下法において、次のステップの変数 y'_k は次のように計算される。

$$y'_k = y_k - \eta \frac{\partial L}{\partial y_k} \quad (2.18)$$

ここで、ステップ幅を決定する定数 η をニューラルネットワークにおいて学習率 (learning rate) という。学習率は 0.001 などの定数を問題やネットワークによって適切に選ぶ必要がある。このようなネットワークについて更新されない初期設定のパラメータをハイパーパラメタという。ハイパーパラメタについては後の 2.6 節で述べる。

ニューラルネットワークにおいて、各重みを更新するための勾配は連鎖律 (chain rule) を用いて計算される。これは更新する重みと最小化される損失関数の間に出力層と活性化関数が存在しているためである。^{*5} 具体的には、ある重み行列 W に対して、勾配降下法、連鎖律を考慮

^{*5} 損失関数はクラスの出力の関数であり、クラスの出力は活性化関数によって計算され、活性化関数は出力層の関数である。

すると、次のステップの重み行列 W' は

$$\begin{aligned} W' &= W - \eta \frac{\partial L}{\partial W} \\ &= \begin{pmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \\ w_{13} & w_{23} & w_{33} \end{pmatrix} - \eta \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{31}} \\ \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{32}} \\ \frac{\partial L}{\partial w_{13}} & \frac{\partial L}{\partial w_{23}} & \frac{\partial L}{\partial w_{33}} \end{pmatrix} \\ &= W - \eta \frac{\partial L}{\partial y} \frac{\partial y}{\partial a} \frac{\partial a}{\partial W} \end{aligned} \quad (2.19)$$

と計算される。

このような最適化問題に関して、いくつかのアルゴリズムが提案されている。現在は確率的勾配降下法 (Stochastic Gradient Descent, SGD[22]) やそれを基礎とした RMSProp[23]、Adam[24] などの手法がよく使用されている。

学習方法における、ニューラルネットワークと多層パーセプトロンの大きな違いは、重みの更新を出力層から逆伝播させる誤差逆伝播法 (Backpropagation[25]) という手法の有無である。再度、図 2.6 を考える。全ての出力、中間層を行列計算を用いて記述すると、

$$\begin{aligned} \mathbf{a}^1 &= \begin{pmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \end{pmatrix} = W^0 \mathbf{x}^0 + \mathbf{b}^0 = \begin{pmatrix} w_{11}^0 & w_{21}^0 & w_{31}^0 \\ w_{12}^0 & w_{22}^0 & w_{32}^0 \\ w_{13}^0 & w_{23}^0 & w_{33}^0 \end{pmatrix} \begin{pmatrix} x_1^0 \\ x_2^0 \\ x_3^0 \end{pmatrix} + \begin{pmatrix} b_1^0 \\ b_2^0 \\ b_3^0 \end{pmatrix} \\ \mathbf{x}^1 &= h(\mathbf{a}^1) \\ \mathbf{a}^2 &= \begin{pmatrix} a_1^2 \\ a_2^2 \\ a_3^2 \end{pmatrix} = W^1 \mathbf{x}^1 = \begin{pmatrix} w_{11}^1 & w_{21}^1 & w_{31}^1 \\ w_{12}^1 & w_{22}^1 & w_{32}^1 \\ w_{13}^1 & w_{23}^1 & w_{33}^1 \end{pmatrix} \begin{pmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \end{pmatrix} \\ y_k &= \sigma(a_k^2) = \frac{\exp(a_k^2)}{\sum_{i=1}^n \exp(a_i^2)} \end{aligned} \quad (2.20)$$

と書ける。ここで、出力部分のソフトマックス関数を σ と書いた。

ある重み w_{11}^1 について考える。損失関数 L の重み w_{11}^1 による偏微分は、連鎖律を考慮して、

$$\begin{aligned} y_1 &= \sigma(a_1^2) \\ a_1^2 &= w_{11}^1 x_1^1 + w_{21}^1 x_2^1 + w_{31}^1 x_3^1 = \sum_{i=1}^N w_{i1}^1 x_i^1 \end{aligned} \quad (2.21)$$

より、

$$\begin{aligned} \frac{\partial L}{\partial w_{11}^1} &= \frac{\partial L}{\partial y_1} \frac{\partial y_1}{\partial a_1^2} \frac{\partial a_1^2}{\partial w_{11}^1} \\ &= \frac{\partial L}{\partial y_1} \frac{\partial \sigma(a_1^2)}{\partial a_1^2} x_1^1 \end{aligned} \quad (2.22)$$

と計算できる。

ここで、勾配の計算に活性化関数の偏微分が常に積の形で含まれていることがわかる。この活性化関数の偏微分が 0 になり、そこから抜け出せなくなると、その勾配は常に 0 になり消失

してしまう、これが勾配消失である。勾配消失に陥った場合は、重みが適切に更新されず、学習が不十分になってしまう。このような問題は活性化関数を変更することによって改善され、現在は ReLU 関数がよく用いられている。

また、更に浅い層の重み w_{11}^0 について考えると、

$$\begin{aligned} x_1^2 &= h(a_1^1) \\ a_1^1 &= w_{11}^0 x_1^0 + w_{21}^0 x_2^0 + w_{31}^0 x_3^0 = \sum_{i=1}^N w_{i1}^1 x_i^0 \end{aligned} \quad (2.23)$$

より、

$$\begin{aligned} \frac{\partial L}{\partial w_{11}^0} &= \sum_k^N \frac{\partial L}{\partial y_k} \frac{\partial y_k}{\partial a_k^2} \frac{\partial a_k^2}{\partial x_1^2} \frac{\partial x_1^2}{\partial a_1^1} \frac{\partial a_1^1}{\partial w_{11}^0} \\ &= \sum_k^N \frac{\partial L}{\partial y_k} \frac{\partial y_k}{\partial a_k^2} \frac{\partial a_k^2}{\partial x_1^2} \frac{\partial h(a_1^1)}{\partial a_1^1} x_{11}^0 \\ &= \sum_k^N \left(\frac{\partial L}{\partial y_k} \frac{\partial y_k}{\partial a_k^2} w_{1k}^1 \right) \frac{\partial h(a_1^1)}{\partial a_1^1} x_{11}^0 \end{aligned} \quad (2.24)$$

と計算できる。この計算は更に層を重ねた場合でも同様の手順で行うことができる。

重みの更新は基本的に全ての訓練データを使用するのではなく、訓練データをいくつかの塊に分け、その塊について損失関数を計算することで行われる。このような手法をミニバッチ学習と呼ばれる。(訓練データ全てを用いたものをバッチ学習という。) ミニバッチ学習に使用されるデータの数をミニバッチサイズ(あるいは単にバッチサイズ) という。これも後述するハイパーパラメータの一つである。ミニバッチ学習はバッチ学習と比較して二つの利点が存在する。

一つは膨大なデータを直接処理しなくても良いという点である。一般に深層学習で使用されるデータは非常に膨大であり、GPUなどのメモリに乗らない場合があるが、ミニバッチ学習ではこれを回避することができる。

もう一つは学習が停滞しづらいという点である。訓練データと比較してサイズの小さいミニバッチは上記の勾配が 0 になりづらく、局所的な最小点での学習の停滞を回避することができる。

ただしミニバッチ学習のバッチサイズが小さくなつた場合には、損失関数が平均化されず学習が不安定になる(収束しなくなる)という問題が生じる場合がある。^{*6}

2.3.3 ディープニューラルネットワーク

ディープニューラルネットワーク(Deep Neural Network, DNN)という言葉の定義は非常に曖昧である。^{*7} 2.1 節で述べたように、現在は第三期 AI ブームであると言われている。これ

^{*6} バッチサイズが 1(1 データのみ) のミニバッチ学習をオンライン学習という。

^{*7} 少なくとも私ははつきりとした定義を存じない。

は上述してきた技術的成熟に加え、計算機の性能が向上したことにより、より層を重ねた（深い）ニューラルネットワークの学習が可能になった結果であると言える。2006年 Hinton らによる auto-encoder[26] や 2014年に Ian によって提案された敵対的生成ネットワーク (Generative Adversarial Network, GAN[27]) など、様々な発展的な応用がなされ、現在においても毎年新しいネットワークが提案されている。本節で述べたニューラルネットワークはその基礎の一部分である。次節以降では、系列を扱うためのニューラルネットワークの応用について紹介する。

2.4 リカレントニューラルネットワーク

前節で紹介したようなフィードフォワードニューラルネットワークは系列データを扱う際、重み行列が固定的な大きさでしか保持出来ないという点と直前の系列に依存した学習が出来ないという点に関して課題を抱えている。これらの課題を解決するために提案されたのが、リカレントニューラルネットワーク (Recurrent Neural Network, RNN) というネットワーク構造である。本節ではこのリカレントニューラルネットワークについて解説を行う。リカレントニューラルネットワークは主に系列データ、特に時系列データを取り扱うためのネットワークである。このような時系列に関するニューラルネットワークは自然言語処理などの分野で発展し、音声認識や機械翻訳といった技術に応用されている。リカレントニューラルネットワークの構造は、フィードフォワードニューラルネットワークと比較すると複雑であるが、要素計算は全結合であり、基本的にはその組み合わせで理解できる。2.4.1項では、そのようなりカレントニューラルネットワークの構造と学習について述べる。その後、リカレントニューラルネットワークの抱える問題と、ゲート (Gate) と記憶セル (Cell) 呼ばれる技術によってその問題を克服した長・短期記憶 (Long Short-Term Memory, LSTM[28]) というネットワークを 2.4.2項と 2.4.3項でそれぞれ紹介する。

2.4.1 リカレントニューラルネットワークの構造と学習

リカレントニューラルネットワークの構造は、これまでのフィードフォワードニューラルネットワークとは大きく異なる。そのネットワーク構造はいくつかの表現方法が存在しているが、本論文では時間について展開した図で書くこととする。図 2.7 の左側が時間について展開していない図、右側が時間について展開した図である。左側では、時間についての構造をループで表現し、任意の時間 t についての入力 x_t と出力 h_t を持つネットワークとして表現している。右側では、時間についての構造を展開し、出力や入力を系列情報とともに表現している。これまでのフィードフォワードネットワークは情報の伝達を左右に描いていたが、このリカレントニューラルネットワークは上下に描き、系列の流れを左右で表現されることが多い。図 2.7 の右側では、出力 h が二つ存在し、上に進むものを出力に、右に進むものが次の系列への入力になっていることがわかる。このように、現在の出力を次の系列の入力として使うことで、直

前の系列情報への依存性を導入している。

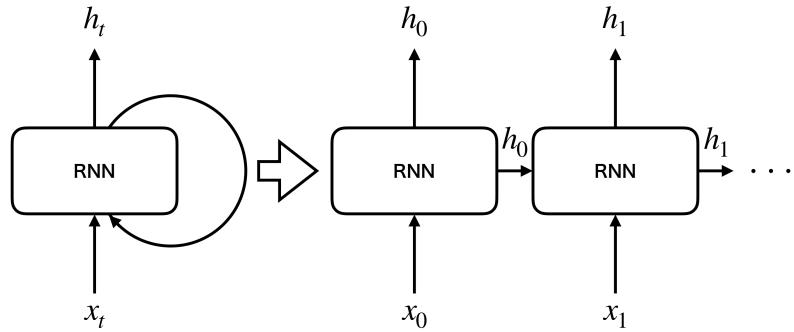


図 2.7: リカレントニューラルネットワーク

これまでと同様に明示的にネットワークの重みを描画すると、図 2.8 のようになる。図 2.7 の RNN に当たる部分が展開され、重みを線で表現した図になっている。図より、一つ前に系列の出力 \mathbf{h}_{t-1} と現在の系列の入力 \mathbf{x}_t を用いて、現在の系列の出力 \mathbf{h}_t が生成されていることがわかる。 \mathbf{h} についての赤い線に関する重み行列を W_h 、 \mathbf{x} についての黒い線に関する重み行列を W_x と置くと、出力 \mathbf{h}_t は次のように計算できる。

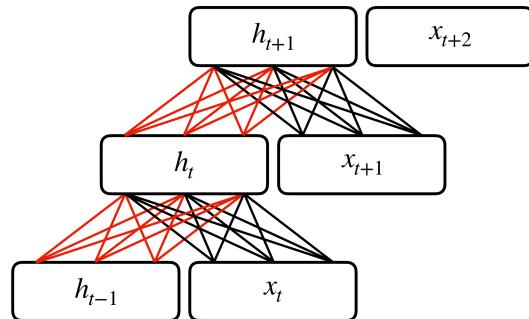


図 2.8: リカレントニューラルネットワークの重み

$$\mathbf{h}_t = \tanh(W_h \mathbf{h}_{t-1} + W_x \mathbf{x}_t)$$

$$\mathbf{a}_t = \begin{pmatrix} w_{h,11} & w_{h,21} & w_{h,31} \\ w_{h,12} & w_{h,22} & w_{h,32} \\ w_{h,13} & w_{h,23} & w_{h,33} \end{pmatrix} \begin{pmatrix} h_{t-1,1} \\ h_{t-1,2} \\ h_{t-1,3} \end{pmatrix} + \begin{pmatrix} w_{x,11} & w_{x,21} & w_{x,31} \\ w_{x,12} & w_{x,22} & w_{x,32} \\ w_{x,13} & w_{x,23} & w_{x,33} \end{pmatrix} \begin{pmatrix} x_{t,1} \\ x_{t,2} \\ x_{t,3} \end{pmatrix} \quad (2.25)$$

リカレントニューラルネットワークでは活性化関数として \tanh 関数を使用している。前述したように、個々の要素計算はフィードフォワードニューラルネットワークの様に全結合で構成されていることがわかる。ここで、非常に重要な性質として、学習可能な重み行列 W_h, W_x は全ての系列 t について同一のものであり、大きさが不変であることに注意する。

このように再帰的に重み行列を使用することで、行列の大きさが可変でないという性質を回避し、系列情報を取り入れることに成功している。また、このような入力の系列長についての柔軟性は、長さが不定であるリアルタイムな時系列データを扱えるという点で重要である。

リカレントニューラルネットワークの出力方法は、問題によっていくつかのパターンが存在する。(図 2.9) 例えば、語句の分類の様な問題の場合は、一つの入力に対して、一つの出力を得る Many to Many という出力の作り方を行う。また、機械翻訳のデコーダーなど、一つの入力に対して、複数の出力を得たい場合は One to Many を用いる。(2.5.1 項) 最後に、感情分析の様に複数の入力に対して、一つの出力を得たい場合は Many to One を用いる。

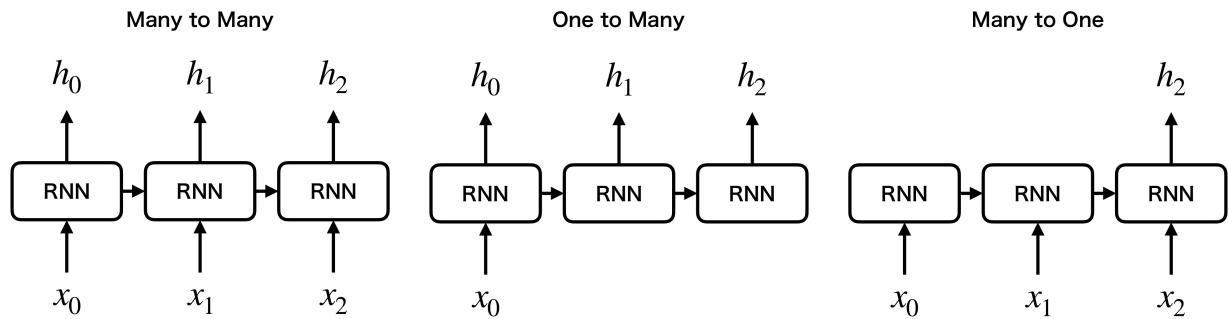


図 2.9: リカレントニューラルネットワークの出力方法

リカレントニューラルネットワークの学習は基本的にフィードフォワードニューラルネットワークと同様であるが、図 2.8 に見る様にネットワークは系列にしたがって深くなっている。この為、重み更新はこの系列を遡って行う必要がある。このような誤差逆伝播法の事を Backpropagation Through Time (BPTT) という。実際には計算リソースの削減のため、Truncated BPTT という、時系列方向に適当な長さで切り取り計算を行う手法が使用される。

2.4.2 リカレントニューラルネットワークの問題点

リカレントニューラルネットワークは時間方向に展開し、それを遡ることによって学習を行っているため、系列の長さに依存して非常に深いネットワークが構築される。したがって、リカレントニューラルネットワークは真に深いネットワークであると言えるが、深い層からの勾配は非常に消失あるいは爆発しやすく、容易に勾配消失・爆発を招いてしまうという問題が生じている。勾配消失はリカレントニューラルネットワークの活性化関数である \tanh 関数に起因している。2.3.2 項で解説したように、誤差逆伝播法は連鎖律によって計算され、その計

算には活性化関数の微分が含まれている。ここで \tanh 関数の微分は、

$$\begin{aligned}\frac{\partial y(x)}{\partial x} &= \frac{\partial}{\partial x} \tanh(x) = \frac{1}{\cosh^2(x)} \\ &= 1 - \frac{\cosh^2(x) - 1}{\cosh^2(x)} = 1 - \frac{\sinh^2(x)}{\cosh^2(x)} = 1 - \tanh^2(x) = 1 - y^2\end{aligned}\tag{2.26}$$

と計算される。

$1 - y^2$ は、 $y = 0$ 以外において常に 1 より小さい値を取ってしまう。リカレントニューラルネットワークでは系列長に応じて、この 1 より小さい値 ($1 - y^2$) が複数回掛けられてしまうため、勾配消失が生じやすくなっている。同時にリカレントニューラルネットワークでは、連鎖律によって系列長に応じて同じ重み行列 W_h を複数回掛けており、この重み行列の値に応じて、勾配が発散あるいは消失してしまうことが考えられる。

また、リカレントニューラルネットワークは時系列上の複数の入出力から、矛盾した重み更新を受け取ってしまう入力重み衝突、出力重み衝突という問題も抱えている。

更に、リカレントニューラルネットワークはその構造上、長期的な系列情報を保持できないという課題も存在している。

以上のような問題を解決するため、ゲートとセルと呼ばれる技術を導入したものを、ゲート付きリカレントユニット (Gated Recurrent Unit, GRU[29]) という。次項では、このゲートを用いたリカレントニューラルネットワークの一つである LSTM について紹介する。

2.4.3 長・短期記憶 (Long Short-Term Memory, LSTM)

LSTM のネットワーク構造全体を図 2.10 に示す。リカレントニューラルネットワークとの最も大きな違いは、LSTM は隠れ状態 (出力) を二つ $\mathbf{h}_t, \mathbf{c}_t$ 持っているという点である。 \mathbf{c}_t は長期的な記憶セルを示しており、図 2.10 では上部の赤い線で表現されている。一方、 \mathbf{h}_t は、リカレントニューラルネットワークと同様に短期的な系列情報の伝達と出力に使用されている。LSTM は三つの入力 $\mathbf{h}_{t-1}, \mathbf{c}_{t-1}, \mathbf{x}_t$ を受け取り、二つの出力 $\mathbf{h}_t, \mathbf{c}_t$ を提供するネットワークであるとみなすことができる。

また、LSTM は内部に前項で述べた勾配消失や勾配爆発、入力重み衝突、出力重み衝突といった様々な問題を解決するためのゲートと呼ばれる構造を四つ持っている。(図 2.10)

それぞれの役割と演算を以下に示す。

- 忘却ゲート

図 2.12a の赤い線で表現している領域では、入力 $\mathbf{h}_{t-1}, \mathbf{x}_t$ を用いて、どの程度、直前の長期記憶セル \mathbf{c}_{t-1} を忘れるかの度合いである f_t を計算している。これを忘却ゲートという。 f_t は、次のようにかける。

$$f_t = \sigma(W_f \mathbf{x}_t + R_f \mathbf{h}_{t-1})\tag{2.27}$$

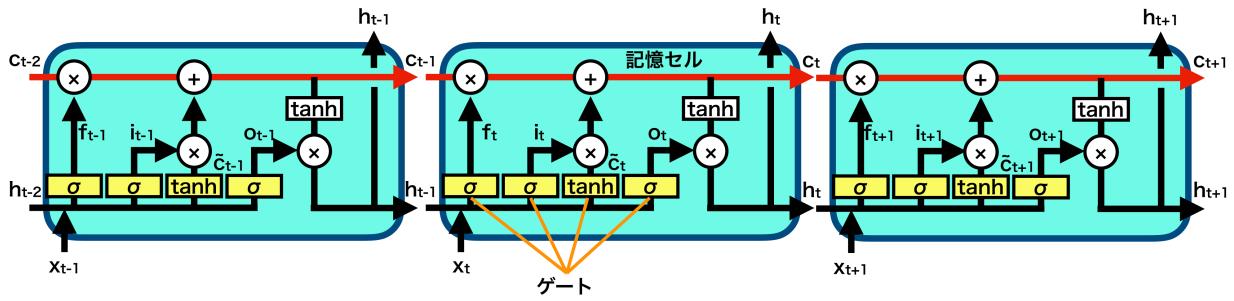


図 2.10: LSTM の流れ

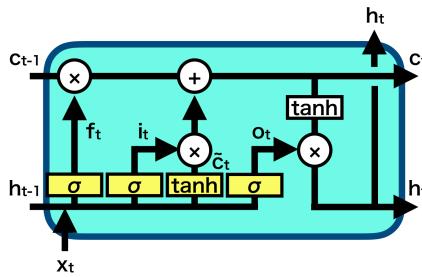


図 2.11: 単体の LSTM

したがって最終的には、

$$c_{t-1} \odot f_t = c_{t-1} \odot \sigma(W_f x_t + R_f h_{t-1}) \quad (2.28)$$

として、次のゲート以降で長期記憶を更新するための容量を確保していると解釈できる。

- 入力ゲート

入力ゲートは忘却ゲートと全く同じ構造 (図 2.12b) をしており、次のように計算できる。

$$i_t = \sigma(W_i x_t + R_i h_{t-1}) \quad (2.29)$$

忘却ゲートではどの程度、長期記憶セルを忘れるかを計算していたように、入力ゲートでの i_t は、次のセルの更新時に新しい長期記憶セルをどの程度重視するかを表現していると解釈できる。

- セルの更新

図 2.12c では、まず更新された長期記憶セル \tilde{c}_t を計算している。

$$\tilde{c}_t = \tanh(W_c x_t + R_c h_{t-1}) \quad (2.30)$$

次にこれまでの三つのゲートの結果をまとめることで、新しい隠れ状態である長期記憶

セル \mathbf{c}_t を計算できる。

$$\begin{aligned}\mathbf{c}_t &= \mathbf{c}_{t-1} \odot \mathbf{f}_t + \tilde{\mathbf{c}}_t \odot \mathbf{i}_t \\ &= \mathbf{c}_{t-1} \odot \sigma(W_f \mathbf{x}_t + R_f \mathbf{h}_{t-1}) + \tanh(W_c \mathbf{x}_t + R_c \mathbf{h}_{t-1}) \odot \sigma(W_i \mathbf{x}_t + R_i \mathbf{h}_{t-1})\end{aligned}\quad (2.31)$$

第一項では、直前の長期記憶セル \mathbf{c}_{t-1} をどの程度忘れるかを \mathbf{f}_t によって制御し、第二項では、新しく計算された長期記憶セル $\tilde{\mathbf{c}}_t$ をどの程度重視するかを \mathbf{i}_t によって制御している。

- 出力ゲート

最後に出力ゲートについて述べる。出力ゲートではここまでに計算された \mathbf{c}_t と入力 $\mathbf{x}_t, \mathbf{h}_{t-1}$ を用いて、最終的な出力となる \mathbf{h}_t を計算している。ただし、出力ゲート \mathbf{o}_t 自体は入力ゲートや忘却ゲートと全く同じ形(図 2.12d)をしている。具体的な計算は次のように書ける。

$$\begin{aligned}\mathbf{o}_t &= \sigma(W_o \mathbf{x}_t + R_o \mathbf{h}_{t-1}) \\ \mathbf{h}_t &= \tanh(\mathbf{c}_t) \odot \mathbf{o}_t \\ &= \tanh(\mathbf{c}_t) \odot \sigma(W_o \mathbf{x}_t + R_o \mathbf{h}_{t-1}) \\ &= \tanh(\mathbf{c}_{t-1} \odot \sigma(W_f \mathbf{x}_t + R_f \mathbf{h}_{t-1}) + \tanh(W_c \mathbf{x}_t + R_c \mathbf{h}_{t-1}) \odot \sigma(W_i \mathbf{x}_t + R_i \mathbf{h}_{t-1})) \\ &\quad \odot \sigma(W_o \mathbf{x}_t + R_o \mathbf{h}_{t-1})\end{aligned}\quad (2.32)$$

以上のように、LSTM の内部構造は煩雑であるが、その構成要素は全てリカレントニューラルネットワークと同様に全結合で計算できる。以下に最終的な出力の計算についてまとめる。

$$\begin{aligned}\mathbf{c}_t &= \mathbf{c}_{t-1} \odot \sigma(W_f \mathbf{x}_t + R_f \mathbf{h}_{t-1}) + \tanh(W_c \mathbf{x}_t + R_c \mathbf{h}_{t-1}) \odot \sigma(W_i \mathbf{x}_t + R_i \mathbf{h}_{t-1}) \\ \mathbf{h}_t &= \tanh(\mathbf{c}_{t-1} \odot \sigma(W_f \mathbf{x}_t + R_f \mathbf{h}_{t-1}) + \tanh(W_c \mathbf{x}_t + R_c \mathbf{h}_{t-1}) \odot \sigma(W_i \mathbf{x}_t + R_i \mathbf{h}_{t-1})) \\ &\quad \odot \sigma(W_o \mathbf{x}_t + R_o \mathbf{h}_{t-1})\end{aligned}\quad (2.33)$$

ここで、隠れ状態(出力) $\mathbf{h}_t, \mathbf{c}_t$ の計算は全て入力 $\mathbf{h}_{t-1}, \mathbf{c}_{t-1}, \mathbf{x}_t$ によって計算できている。また、学習可能な重み行列は $W_f, W_i, W_c, W_o, R_f, R_i, R_c, R_o$ の八つである。一般的には、ここに適宜バイアス b を加えることが多い。

リカレントニューラルネットワークや LSTM は更に重ねられる(Stacked)という非常に強力な性質を持っている。そのようなネットワークを図 2.13 に示す。図からわかるように、一段目の LSTM の出力が二段目の LSTM の入力になっている。一方で、セルは両者で共有されず、独立した状態を保持している。それら以外の基本的な構造は一段であった時の LSTM と変わっていない。このように、LSTM は系列としての深さだけでなく、フィードフォワードニューラルネットワークと同様に重ねることによる深さの確保が可能である。勿論この重ねる操作は二段以上への拡張が可能であり、その場合は二段目の出力を三段目の入力に使うことによって実現できる。どの程度重ねるかはハイパーパラメータであり探索が必要である。

更に、双方向(Bidirectional) LSTM と呼ばれるネットワークに関しても解説する。これは、図 2.14 のように、片側を順方向に、もう一方を逆方向に系列を読み込むことにより、自

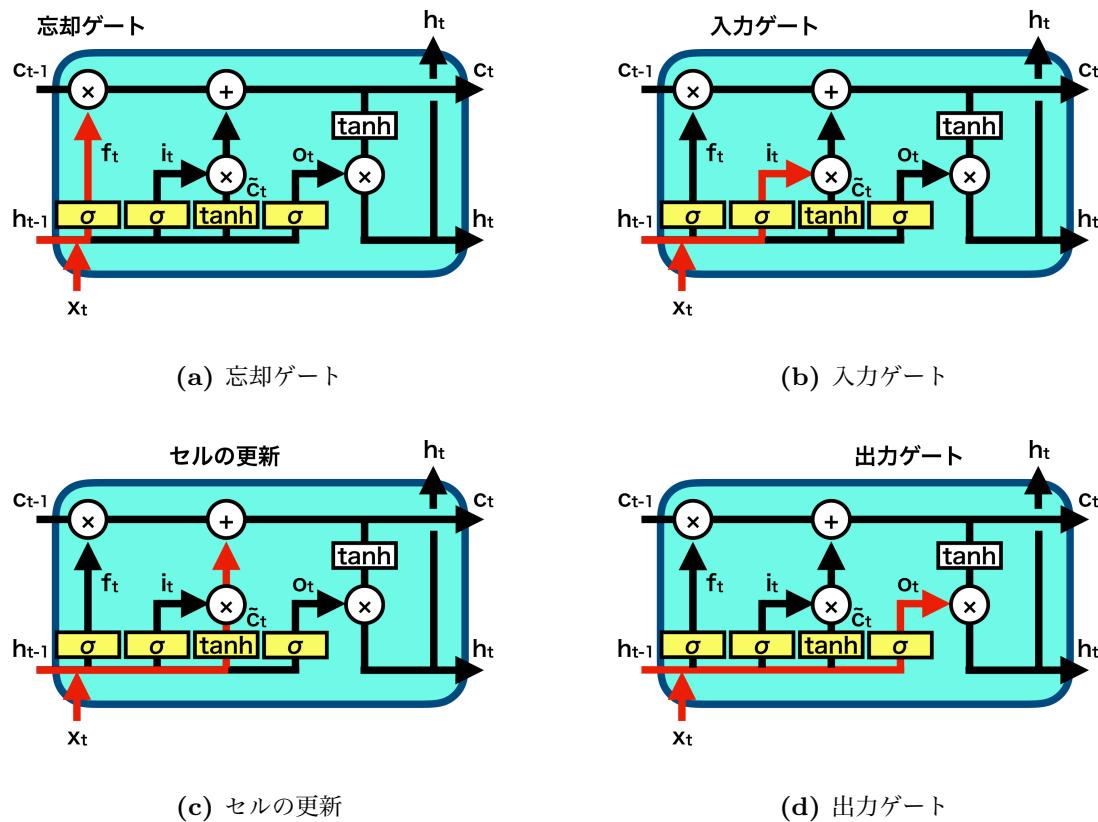


図 2.12: LSTM の各ゲートについての図解

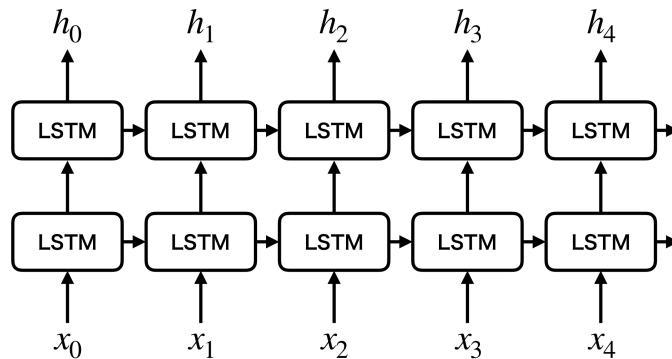


図 2.13: Stacked LSTM

然言語処理などに見られる、将来的な系列情報への依存を導入することができる。ここで、前述の LSTM を重ねる手法と異なり、それぞれの LSTM の入力はそれぞれ独立しており、前段の出力を使用していないことに注意が必要である。また、この双方向 LSTM の構造上の性質として、系列データを全て持つておく必要があるという点にも留意しなければならない。したがって、リアルタイムな問題については、未来の情報を得ることができない為、この双方向 LSTM を用いることはできない。また、この双方向 LSTM を重ねることも可能である。

リカレントニューラルネットワークはこのように次々と拡張され、より複雑で難解な系列情報の処理について、高い性能を発揮している。

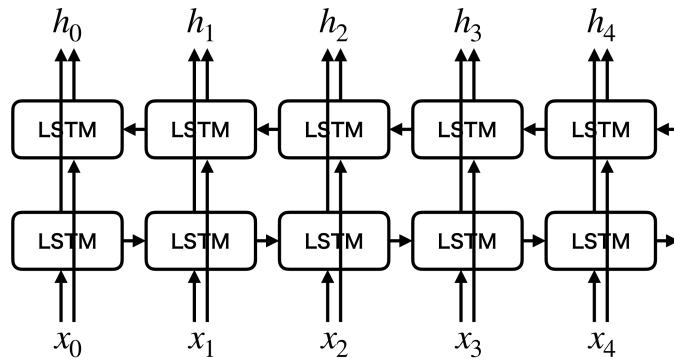


図 2.14: 双方向 LSTM

リカレントニューラルネットワークはその構造が再帰的であるという点（並列化が困難である）から、学習が遅く、重いという課題を抱えている。また、後述するエンコーダー・デコーダーモデルにおいては、データの系列の長さに応じた情報を確保できないという欠点が存在している。^{*8}次節では、このような問題を解決するための注意機構（Attention）という技術を解説する。

2.5 Attention

Attention[30, 31] はその名の通り、ある系列データのどこに注意するかを計算する機構である。主に機械翻訳や意味理解などのエンコーダー・デコーダーモデルに使用されており、様々な応用が議論されているが、ここでは基本的な Attention の理論と考え方についてのみ解説する。

2.5.1 項では、まず Attention を理解する上で必要不可欠なエンコーダー・デコーダーモデルについて述べ、その中で前節のリカレントニューラルネットワークが抱える問題について紹介する。次に、2.5.2 項で Attention の理論や計算について述べる。

2.5.1 エンコーダー・デコーダーモデル

Attention は主に機械翻訳などのエンコーダー・デコーダーモデルに使用されている。LSTM を用いたエンコーダー・デコーダーモデルの大まかな構成を図 2.15 に示す。LSTM で

^{*8} 例えば、機械翻訳を用いて 100 単語分の英文を日本語に翻訳する場合と 10 単語分の英文を翻訳する場合において、100 単語分の英文が持つ情報の方が 10 単語分の英文と比べて多いことは明らかであるが、リカレントニューラルネットワークはこれらの情報の多さの違いに対応できず、常に同じ量の情報から日本語を生成してしまう。

はエンコーダーとデコーダーを繋ぐ情報はエンコーダーの最後の層の出力を使用することが多い。つまり、エンコーダーは図 2.9 の Many to One、デコーダーは One to Many を使用しており、エンコーダーによって抽出された情報はこの One の部分に集められることになる。この出力は Many to One の内、Many であるデータの系列の長さ（系列長）に依存せず、常に同じ大きさの入れる要素となる。したがって、長い系列長であっても、短い系列長であっても同じ量の情報を使用しているということを意味している。よって、より長い系列長や短い系列長を扱う場合は、そのデータが持つ情報を完全に保持、表現することは難しく、情報の欠損により性能が下がってしまうという問題があった。

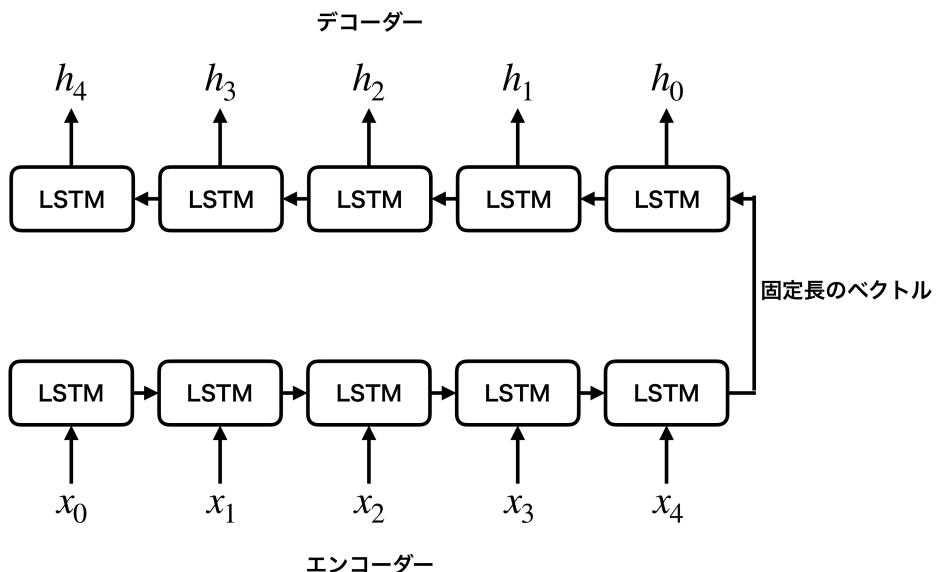


図 2.15: LSTM によるエンコーダー・デコーダーモデル

この問題を解決するための技術が Attention である。Attention を組み込んだ LSTM のエンコーダー・デコーダーモデルを図 2.16 に示す。ここで、図中の Attention と示している部分は、実際にはエンコーダーの全ての系列の出力を集めた行列である。このようにして、Attention は系列長に依存した情報量を確保できる。次項ではより詳細な Attention の計算について説明する。

2.5.2 Attention

図 2.16 では、エンコーダーとデコーダーの間に Attention を表現しているが、実際には Attention はデコーダーの個々の系列に応じて計算される。これは、デコーダーのある系列 t がエンコーダー全体のどの情報に注目しているかを逐次計算しなくてはならない為である。Attention の実装の仕方は多岐にわたるが、その計算は次のような手順にまとめられる。

1. query、key、value を用意する。
2. query と key を用いて、Attention weight を計算する。

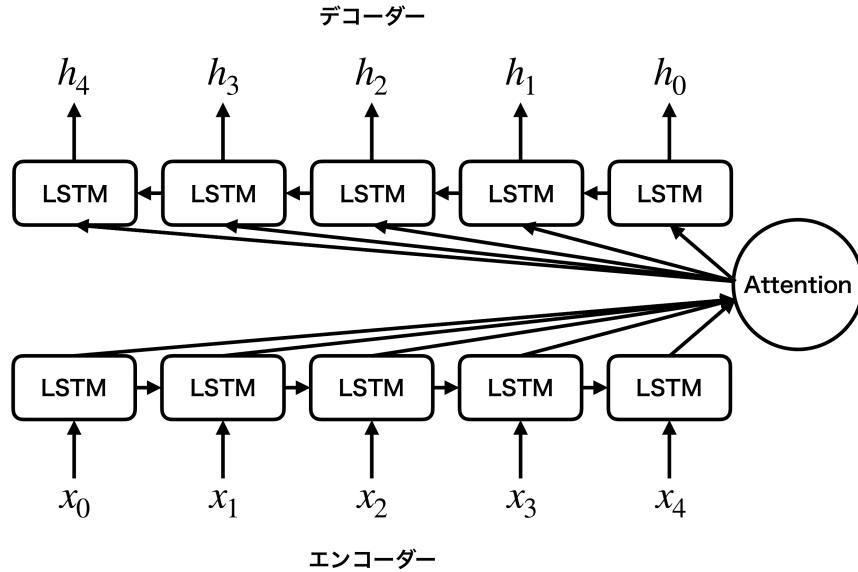


図 2.16: Attention と LSTM によるエンコーダー・デコーダーモデル

3. Attention weight と value を用いて、コンテキストを作成する。

query、key、value において、key と value は同じものが用いられることが多い。key と value は図 2.16 のエンコーダーの全ての系列の出力を集めた行列に相当している。query とはここでは、デコーダーの個々の系列である $t - 1$ 番目の出力（隠れ状態）である。したがって、query と key を用いて、Attention weight を計算するという操作は $t - 1$ 番目の隠れ状態（query）とエンコーダーの全ての系列の出力を集めた行列（key）を用いて、どこに注意すれば良いかの重み（Attention weight）を計算していることを意味している。次に、この Attention weight を、もう一度エンコーダーの全ての系列の出力を集めた行列（value）に掛けることで、エンコーダーのそれぞれの系列の情報を注意しながら取り出すことができる。この取り出された情報をコンテキストといい、Attention ではこのコンテキストをデコーダー部分に使用することで、系列長に依存した情報量を保持している。

Attention weight の計算方法はいくつかの手法が存在しているが、ここでは最も単純な手法を二つ述べる。

- Additive Attention[30]

Additive Attention の利点は query と key がどのような大きさであっても計算できるという点である。N 番目の query と key がそれぞれ、大きさ $[F]$ と $[E, M]$ のベクトルと行列であるとすると、その計算は図 2.17 の上段ように表現できる。ここで N 番目の query は系列長の M 回分反復されており、最終的に大きさ $[F, M]$ の行列となっている。Additive Attention の特徴は Attention の内部に独自の学習可能な重みを保持している点である。N 番目の query 行列と key はそれぞれ重み行列と掛けられ、得られたそれぞれの行列要素について和を取っている。その後、更に大きさ $[D]$ の重みベクトルと

掛けられ、大きさ [M] のエネルギー (図 2.17 中のベクトル $e \in \mathbb{N}$) を得る。このベクトルについて、ソフトマックスと同様の関数で規格化し、Attention weight を作成している。この計算はフィードフォワードニューラルネットワークそのものであり、Additive Attention はこのように内部にネットワークを持っているため、後述する Dot-Product Attention と比較して計算が遅くなるという欠点を抱えている。

- Dot-Product Attention[31]

Dot-Product Attention は Additive Attention と比較して非常にシンプルな構造をしている。その計算を図 2.17 の下段に示している。図からも分かるように Additive Attention とは異なり、内部に重みを保持しておらず、query と key から直接 Attention weight を計算している。ただし、このように query と key を掛けるためには、それぞれの大きさを揃える必要があり、具体的には $E = F$ の時のみこの Dot-Product Attention を使用することができる。

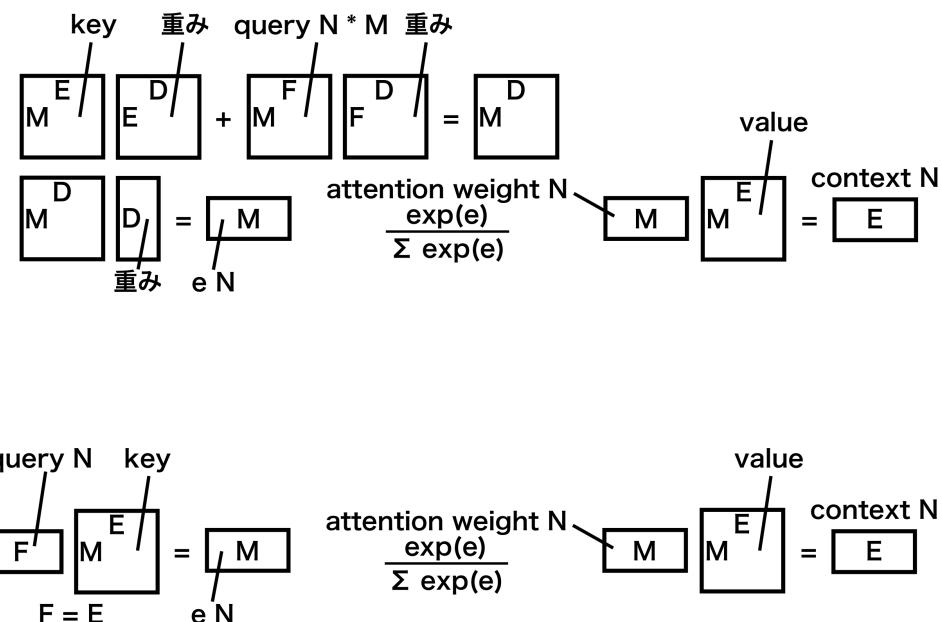


図 2.17: Additive Attention と Dot-Product Attention

コンテキストをどのように以降の計算へ組み込むのかについての具体的な例は、本研究の 3.4.2 項に示している。上記のように計算された Attention は Attention weight を確認することでネットワークの理解をより深めることができる。Attention weight に関する具体的な図についても本研究の 3.4.4 項で確認する。

Attention は”Attention is all you need[32]”と言われるほど、近年非常に注目されている技術である。ここでは、Attention を LSTM の補助として使用している例を挙げたが、Attention は様々な技術と組み合わせられる。また、それだけでなく、Attention のみを用いて構成され

た、Transformer と呼ばれるモデルは 2021 年現在において、自然言語処理の標準的なネットワークであると言われるほどの性能と、LSTM では実現できなかった速さを実現している。

2.6 ハイパーパラメータ

ここまで述べたように深層学習は教師あり学習であるため、訓練データを用いて重みの更新を行い、ネットワークの重み（パラメータ）を調整（チューン）していく。しかし、ネットワークはネットワーク自体を構築するための、学習によって更新されないパラメータをいくつも持っている。このようなパラメータをハイパーパラメータという。ハイパーパラメータは学習前に設定しておく必要があり、ネットワークの性能を引き出すためには適切に最適化（ハイパーパラメータ・チューニング）する必要がある。ハイパーパラメータの種類や数はネットワークの構造によって大きく異なるが、一般にチューニングが必要とされるハイパーパラメータについて以下に示す。

- 最適化手法 (Optimizer) : RMSProp や Adam といった重み更新の為の最適化手法
- 学習率 (Learning rate) : 重み更新のステップ幅
- エポック数 (Epochs) : 学習回数・訓練データを一周学習することを 1 エポックという
- バッチサイズ (Batch size) : ミニバッチ学習における訓練データサンプルの大きさ
- ノード数 (Node) : 重み行列の大きさ
- 層の数 (Layer) : フィードフォワードネットワークにおける全結合層の数

これらはハイパーパラメータの一例であり、それぞれについて適切に選ぶ必要がある。ハイパーパラメータの探索手法も幾つか提案されており、ランダムサーチやグリッドサーチ、ベイズ最適化などが用いられている。本研究におけるハイパーパラメータ・チューニングに関しては 3.4.4 項で述べる。

以上が本論文のための深層学習の導入である。この 2 章を前提として、以降の 3 章での本研究で使用したネットワークの構造を解説していく。3 章などでは、ここで挙げた深層学習の用語を説明せずに使用するが、その場合は適宜、本章を参照していただきたい。また、本章の初めや 2.3 でも述べたように、深層学習の実装に関しては様々なフレームワークがあるため、ここでの記載は省かせていただく、本研究の実装に関しては付録 A や私の GitHub[33] にまとめている。

第 3 章

崩壊点検出の為のネットワーク

本章では、まず 3.1 節で本研究で取り扱うデータの特性について述べる。また、本研究で扱うデータは全てモンテカルロ (Monte Carlo, MC) シミュレーションデータである。次に、3.2 節にて、どのようにして深層学習を使用した崩壊点検出を実現するかについての発想と作成するネットワークとその役割についての概要的なを説明を行う。また、使用したコンピュータや深層学習のフレームワークの詳細についてもここで述べる。作成した個々のネットワークの構造や学習などの詳細や、ネットワーク単体での性能と評価については 3.3 節、3.4 節で解説する。これらネットワークについては 2 章の内容を前提とし、実際に学習に使用する訓練データの詳細や、ハイパーパラメータ・チューニングについてもここで述べる。

3.1 データ

ここでは、本研究で使用した MC シミュレーションデータについて述べる。特に 3.1.1 項では、データ全体についての性質について、3.1.2 項では、本研究で使用する飛跡についての情報と深層学習に使用するための前処理について詳しく述べる。ただし、各ネットワークの学習に使用する訓練データの詳細については、後の 3.3.2 や 3.4.3 で紹介する。

3.1.1 データ全体の性質

本研究では WHIZARD[?] を用いて生成された ILD フルディテクターシミュレーションデータを使用した。重心系エネルギーは Z 粒子の質量である 91.2GeV、また終状態は、 $b\bar{b}$ と $c\bar{c}$ のデータをそれぞれ用意している。データ生成において、Beamstrahlung/ISR やビーム偏極は考慮していない。これらのデータは後述する LCFIPlus での性能評価 [?] で使用されたデータと同一のものである。また、データについて終状態 $b\bar{b}$ のものは 15 個のサンプルに、終状態 $c\bar{c}$ のものは 13 個のサンプルに分け使用した。それぞれの事象数や用途を表 3.1 にまとめる。

データ特性の調査とは本節でのデータそのものについての評価のことである。また、各訓練データの作成に使用するデータサンプルには、検証データ (Validation data) を含むこととす

データ名	事象数	飛跡数	用途
c̄c - 01	69581	1344465	データ特性の調査
c̄c - 02	42204	814074	ネットワークの動作テスト
c̄c - 03	38662	748027	飛跡対についてのネットワークの訓練データの作成
c̄c - 04	38712	747625	飛跡対についてのネットワークの訓練データの作成
c̄c - 05	38655	748089	任意の数についてのネットワークの訓練データの作成
c̄c - 06	38645	747548	任意の数についてのネットワークの訓練データの作成
c̄c - 07	38643	747312	ネットワークの評価
c̄c - 08	38715	748801	ネットワークの評価
c̄c - 09	38705	747725	フレーバータギングの訓練データの作成
c̄c - 10	38721	748025	フレーバータギングの訓練データの作成
c̄c - 11	38587	747819	フレーバータギングの訓練データの作成
c̄c - 12	38723	748904	LCFIPPlusとの比較
c̄c - 13	35848	693780	LCFIPPlusとの比較
bb̄ - 01	62795	1326168	データ特性の調査
bb̄ - 02	42950	909082	ネットワークの動作テスト
bb̄ - 03	34985	738105	ネットワークの動作テスト
bb̄ - 04	34952	739130	飛跡対についてのネットワークの訓練データの作成
bb̄ - 05	35047	741568	飛跡対についてのネットワークの訓練データの作成
bb̄ - 06	35008	740662	任意の数についてのネットワークの訓練データの作成
bb̄ - 07	34000	718057	任意の数についてのネットワークの訓練データの作成
bb̄ - 08	33978	717972	ネットワークの評価
bb̄ - 09	35008	740268	ネットワークの評価
bb̄ - 10	34954	739320	フレーバータギングの訓練データの作成
bb̄ - 11	35012	740797	フレーバータギングの訓練データの作成
bb̄ - 12	34972	739953	フレーバータギングの訓練データの作成
bb̄ - 13	34986	739402	LCFIPPlusとの比較
bb̄ - 14	34910	740933	LCFIPPlusとの比較
bb̄ - 15	10243	216499	LCFIPPlusとの比較

表 3.1: データサンプルの事象数と用途

る。ただし、教師あり学習であるため、学習に使用したデータと評価に使用するデータは完全に分離し、c̄c - 07, 08, bb̄ - 08, 09 は 3.3.3 項や 3.4.4 項でのみ使用する。性能の評価において、訓練データを作り直す場合は隨時 c̄c - 03, 04, 05, 06, bb̄ - 04, 05, 06, 07 のデータサンプルから作成する。LCFIPPlus でのフレーバータギングは 1.5.3 項で述べた様に BDT が使用されているため、訓練データが必要である。よって、c̄c - 09, 10, 11, bb̄ - 10, 11, 12 はフレーバータギングの訓練データの作成に使用する。また、全ての訓練データの正解ラベルは MC の真値を使用し作成する。

崩壊点検出を行うにあたって、終状態が bb̄ の場合は $b \rightarrow c$ という崩壊過程を辿り、新しい Tertiary Vertex が生じることに注意しなければならない。したがって、終状態が bb̄ の場合は、このボトム (b) ・フレーバーのハドロンによる Secondary Vertex とチャーム (c) ・フ

レーバーのハドロンによる Secondary (Tertiary) Vertex を区別すべきである。典型的な寿命は光速 c を用いて、ボトム (b) ・フレーバーのハドロンの場合は $c\tau = 400 - 500 \mu\text{m}$ 、チャーム (c) ・フレーバーのハドロンの場合は $c\tau = 20 - 300 \mu\text{m}$ である。 (図 3.1)

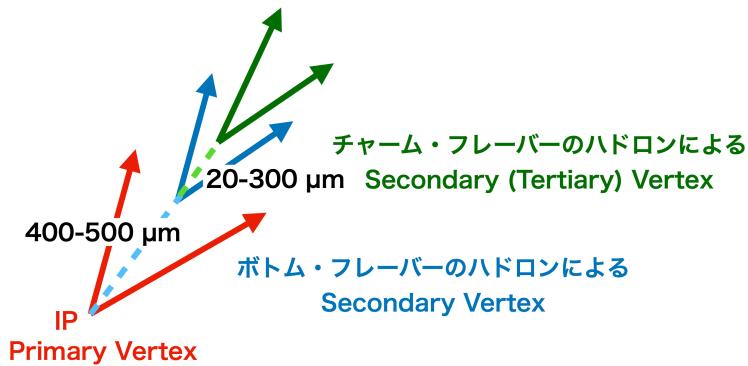


図 3.1: 終状態 $b\bar{b}$ での典型的な崩壊点の例

図??は 1 事象に含まれる飛跡の本数と崩壊点の個数である。

(未完)

3.1.2 飛跡の情報と前処理

本研究では、崩壊点を探索するにあたって飛跡の情報として、図 3.2 のような位置や運動量を含んだトラック・パラメータ ($d_0, z_0, \phi, \Omega, \tan \lambda$) [34] とその共分散行列 雷電 エネルギー

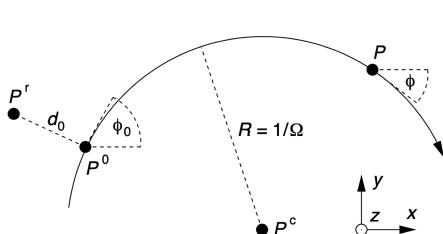


Figure 1: The projection of a helix segment in the xy plane is a part of an arc with centre \mathbf{P}^r and radius R . The direction of the particle is shown with the arrow at the arc. All track parameters are given relative to the reference point \mathbf{P}^r .

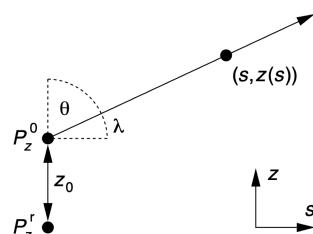


Figure 2: The projection of a helix in the sz plane is a straight line (see Eq. 10). The variable s at a point \mathbf{P} is the arc length in the xy plane from \mathbf{P}^0 to \mathbf{P} . This also implies that $s = 0$, if $z = z_0$.

図 3.2: トラック・パラメータ [34]

深層学習の入力として扱う場合は、一般に $[-1, 1]$ の範囲に変数を整形した方が良いと言われている。したがって、それぞれの変数を以下のような \tanh 関数や線形関数などを用いて変換した。

- トランク・パラメータ

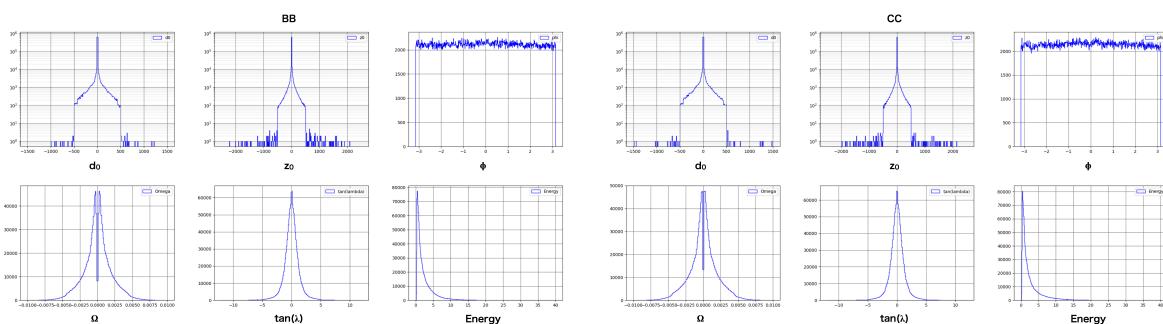
$$d_0 = \tanh(d_0), z_0 = \tanh(z_0), \phi = \phi/\pi, \Omega = \tanh(200\Omega), \tan \lambda = \tanh(0.3d_0)$$

- トランク・パラメータの共分散行列 : $\tanh(8000(x - 0.0005))$

- 電荷 : 変換なし

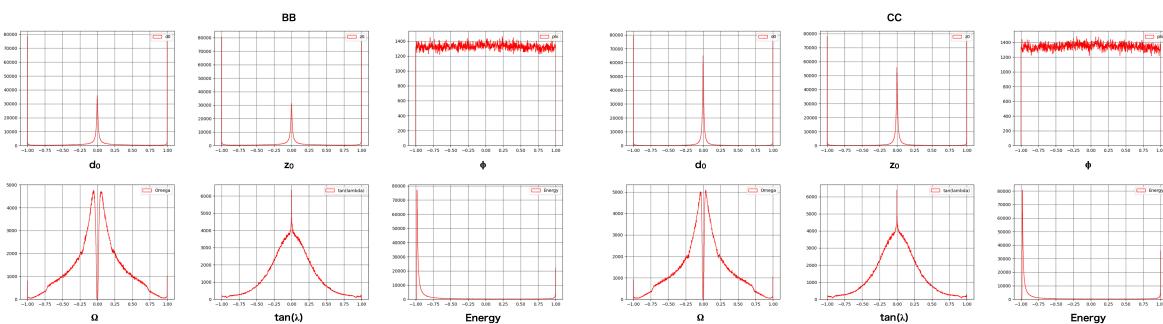
- エネルギー : $\tanh 0.5(x - 5.0)$

トランク・パラメータとエネルギーの変換前の分布と変換後の分布をそれぞれ??に示す。



(a) 終状態 $b\bar{b}$ での変換前の変数の分布

(b) 終状態 $c\bar{c}$ での変換前の変数の分布



(c) 終状態 $b\bar{b}$ での変換後の変数の分布

(d) 終状態 $c\bar{c}$ での変換後の変数の分布

図 3.3: 変数の分布の例

また、LCFIPlus のフィッティングで得られる変数であるカイ二乗や予想される崩壊点の位置についてもデータを用意した。これらの値は 1 事象中の任意の二本の飛跡 (飛跡対) について計算を行ったものである。ただし、深層学習の学習においてはこの値は基本的には使用せず、学習の健全性を確かめる目的や正解ラベルとして用いこととする。MC の真値を用いて崩壊点の種類毎に分離したカイ二乗と予想される崩壊点の位置の分布を図??に示す。

(未完)

3.2 深層学習を用いた崩壊点検出の実現

2章でも述べたように、深層学習は分類問題や回帰問題を解くことのできる教師あり学習である。したがって、深層学習として問題を解く場合は、この二つのいずれかに問題を落とし込む必要がある。我々はまずどのようにして、崩壊点検出を行うかを考えなければならない。さて、??章で述べた崩壊点検出の目的は、事象中の崩壊点とそこに属する飛跡を探索することである。このような問題は一般に分類ではなく、クラスタリングのような手法によって解かれることが多い。しかし、先述の3.1節では、1事象に含まれる崩壊点の数と、飛跡の数が事象毎に異なっていることを示した。そのような崩壊点の数や飛跡の数が不定であるというデータの性質を考慮した上で、崩壊点検出をクラスタリングで解くことはクラスター数や崩壊点種の差の扱いの面で不適であると判断した。分類問題では、データの持つ特徴量の空間内である種の境界が引けなければならない。そのため、あらゆるデータサンプルや事象内で不变な性質を考慮する必要がある。

以上を踏まえた上で私は二つのネットワークを用いた崩壊点検出を提案する。一つは事象内のあらゆる飛跡対に対して、その飛跡対が結合 (Connected) しているか、非結合 (Not connected) であるか、結合しているならば、Primary Vertex であるか Secondary Vertex であるかなどを分類するネットワークである。これを「飛跡対についてのネットワーク」と呼ぶことにする。飛跡対についてのネットワークはあらゆる飛跡対に対して、崩壊点の種を探索することを目的としたネットワークである。入力は飛跡二本分の情報であり、合計44個の変数である。出力はデータサンプルの終状態によって分類数が変化してしまうが、基本的には崩壊点種 (非結合な飛跡対・Primary Vertex・Secondary Vertex) である。

飛跡対についてのネットワークは崩壊点の種となる飛跡対を検出するだけであるので、単体では崩壊点を形成することはできない。そこで、私は崩壊点の生成を行う、もう一つのネットワークを構築した。これを「任意の数の飛跡についてのネットワーク」と呼ぶことにする。任意の数の飛跡についてのネットワークは上記の飛跡対についてのネットワークによって得られた崩壊点の種に対して、事象中の飛跡を一本ずつ加え、それぞれの飛跡がその崩壊点の種と結合しているか、非結合であるかを分類するネットワークである。そのようなネットワークを用いることで、最終的に事象中の全ての飛跡を考慮した崩壊点が生成される。ただし、何度も述べているように事象中に含まれる飛跡の数は事象毎に異なるため、このようなネットワークは単純なフィードフォワードニューラルネットワークでは取り扱うことができない。したがって私は、このネットワークをリカレントニューラルネットワークの技術を用いて作成した。

以上の二つのネットワークを用いることで、崩壊点検出を実現する。構造や学習についてのより詳細な個々のネットワークの解説は、後の3.3節や3.4節で述べる。

これらのネットワークは tensorflow/keras フレームワークを用いて作成した。また、学習に際しては弊研究室サーバーの TITAN RTX を二つと九州大学のスーパーコンピューターである ITO を使用した。更に後述する推論の段階では更に C++ への実装を行なっているが、

その詳細に関しては 5.2 節で述べる。

3.3 飛跡対についてのネットワーク

ここでは 3.2 節で紹介した二つのネットワークの内、飛跡対についてのネットワークについて述べる。主にネットワークの構造に関しては 3.3.1 項で、学習に関しては 3.3.2 項で解説する。また、そのようにして構築、訓練されたネットワーク単体についての性能と評価に関しては、3.3.3 項で述べることとする。

飛跡対についてのネットワークは、3.2 述べたようにデータサンプルの終状態によって出力が異なる。これは終状態が $b\bar{b}$ の場合はボトム (b) ・フレーバーの Secondary Vertex とチャーム (c) ・フレーバーの Secondary Vertex が生じるのに対し、終状態が $c\bar{c}$ の場合はチャーム (c) ・フレーバーの Secondary Vertex のみが生じることに起因している。また、終状態が $b\bar{b}$ の場合はボトムとチャーム・フレーバーのそれぞれの Secondary Vertex 由来の飛跡対 (それぞれ Secondary Vertex BB, Secondary Vertex CC と呼ぶこととする) と、更にボトム・フレーバーの Secondary Vertex 由来の飛跡とそこから生じたチャーム・フレーバーの Secondary Vertex 由来の飛跡を含んだ飛跡対 (Secondary Vertex BC) を考慮した。

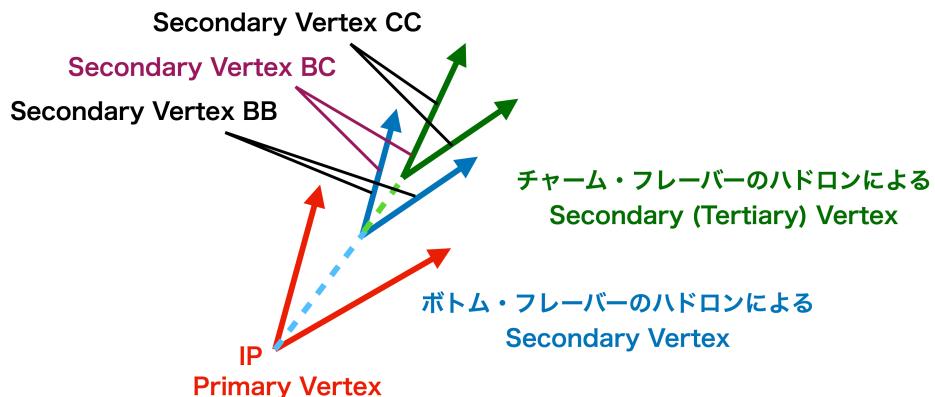


図 3.4: 終状態 $b\bar{b}$ での分類クラスの定義

終状態	分類クラス名
$c\bar{c}$	Not Connected, Primary Vertex, Secondary Vertex
$b\bar{b}$	Not Connected, Primary Vertex, Secondary Vertex CC, BB, BC

表 3.2: 終状態と分類クラス名

また、飛跡対について崩壊点の位置を予想するネットワークも全く同様の構造を用いて作成した。こちらについては、訓練データの正解ラベルとして LCFIPlus のフィッティングで得ら

れる崩壊点の位置を用いた。位置を予想するネットワークは 3.3.3 項での評価や 4 章での崩壊点の種の選別に使用している。

3.3.1 ネットワークの構造

飛跡対についてのネットワークは非常にシンプルなフィードフォワードニューラルネットワーク構造のものを使用した。ネットワークの概略図を図 3.5 に示す。



図 3.5: 飛跡対についてのネットワークの概略図

終状態や崩壊点種、位置を予想する際のネットワークの違いは最後の全結合層のノード数をそれぞれ終状態 $b\bar{b}$ では 5、 $c\bar{c}$ では 3、位置を予想する場合は回帰問題として解くためノード数を 1 とした。標準的なハイパーパラメータは図 3.6 にまとめている。

また、過学習 (Over fitting) を避ける為、Batch Normalization[?] を全結合層の後ろに配置している。過学習とは、ネットワークが過度に訓練データに適合してしまい、検証データやテストデータへの汎化性能が悪化してしまう教師あり学習での問題である。また勾配消失への対策として、活性化関数は全て ReLU 関数を使用している。ノード数や層数に関してのハイパーパラメータ・チューニングに関しては 3.3.3 にて述べる。

3.3.2 ネットワークの学習と戦略

訓練データは事象中の全ての飛跡対の組み合わせを考える。したがって図??のように、分類クラスの数の比は”Not Connected”が支配的な不均衡 (Imbalanced) データとなる。このような不均衡データについては、少数クラスのデータをかさ増しする Oversampling、多数クラスのデータを間引く Undersampling、損失関数のコストに重みをつけるコスト考慮型学習の主に三つの対応策が存在する。

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 44)]	0
dense (Dense)	(None, 256)	11520
batch_normalization (BatchNo	(None, 256)	1024
activation (Activation)	(None, 256)	0
dense_1 (Dense)	(None, 256)	65792
batch_normalization_1 (Batch	(None, 256)	1024
activation_1 (Activation)	(None, 256)	0
dense_2 (Dense)	(None, 256)	65792
batch_normalization_2 (Batch	(None, 256)	1024
activation_2 (Activation)	(None, 256)	0
dense_3 (Dense)	(None, 5)	1285
vertex_output (Activation)	(None, 5)	0
<hr/>		
Total params:	147,461	
Trainable params:	145,925	
Non-trainable params:	1,536	

図 3.6: $b\bar{b}$ データセットについて崩壊点種予測モデルにおける各種パラメーターの出力

Oversampling や Undersampling は過学習や情報の欠損などの問題を孕んでいるため、本研究ではコスト考慮型学習を用いた。また重みは分類クラスの数の比の逆数を使用した。

- 終状態 $b\bar{b}$

$$L = -t_{NC} \log(y_{NC}) - t_{PV} \log(y_{PV}) - t_{SVCC} \log(y_{SVCC}) - t_{SVBB} \log(y_{SVBB}) - t_{SVBC} \log(y_{SVBC}) \quad (3.1)$$

- 終状態 $c\bar{c}$

$$L = -t_{NC} \log(y_{NC}) - t_{PV} \log(y_{PV}) - t_{SV} \log(y_{SV}) \quad (3.2)$$

ここで、NC : Not Connected, PV : Primary Vertex, SV : SecondaryVertex と訳した。

次に重み更新の最適化手法として、SGD を用いた。これは、Adam などでは収束が早すぎ、過学習になる恐れがあつたためである。

エポックを横軸に、正答率と損失を縦軸にプロットしたものを図??に示す。

3.3.3 ネットワークの性能

飛跡対についてのネットワークの性能を混合行列 (Confusion Matrix) にまとめる。(図??)
位置を見ているか
Chi2, Pos, Cov Mat, 位置を予想するネットワーク

Others クラスを加えた分類

ROC カーブ

3.4 任意の数の飛跡についてのネットワーク

ここでは 3.2 節で紹介した二つのネットワークの内、任意の数の飛跡のためのネットワークについて述べる。基本的には前節の飛跡対についてのネットワークと同様の手順での解説を行うが、この任意の数の飛跡についてのネットワークは、既存のネットワーク構造にはない独自のネットワークで構築している。これは本研究におけるデータの特殊性や問題解決のための最適なネットワークを考慮した結果である。このようなネットワークの詳細な構造については 3.4.2 項で述べる。

3.4.1 ネットワークの構造

任意の数の飛跡についてのネットワークではリカレントニューラルネットワークの技術を採用している。ただし、飛跡は本質的に順序を持っておらず、系列データではない為、リカレントニューラルネットワークをそのまま用いることはデータの性質に合わない。この為私は、リカレントニューラルネットワークの一つである LSTM を拡張し、新しい独自のリカレントニューラルネットワークの構造を構築した。この独自のネットワーク構造については、次項 3.4.2 にて解説する。ここでは、より大きな枠組みとしてのネットワークの構造について述べる。

図 3.7 は、リカレントニューラルネットワークを用いた最も簡単な崩壊点生成を表現したものである。ここで、左から崩壊点の種である飛跡対が入力されている。実際には全結合層を通して、リカレントニューラルネットワークの初期状態としている。この初期状態は、リカレントニューラルネットワーク内の重みと共に全結合層が学習される為、学習可能な初期状態 (Trainable (Learnable) Initial State) となっている。また、飛跡は下から一本ずつ入力され、1 事象分の全ての飛跡が使用されるが、リカレントニューラルネットワークである為、系列として扱う飛跡の本数は任意である。したがって、ある崩壊点のタネに対して、任意の数の飛跡が結合しているか否かを分類することができる。

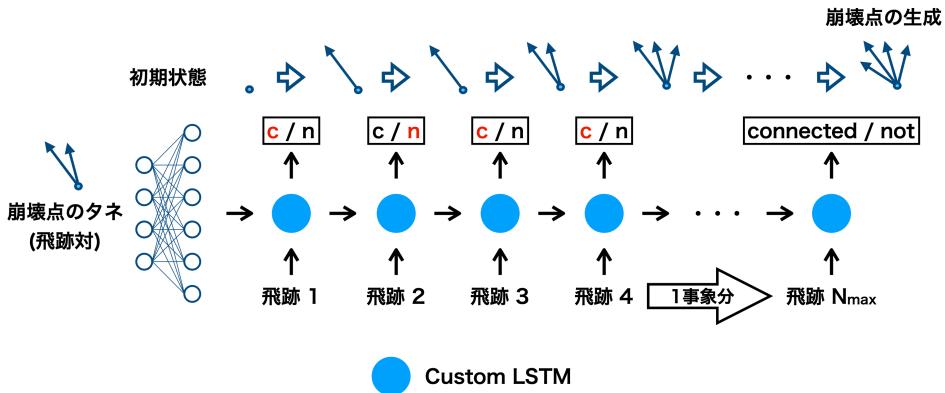


図 3.7: 独自のリカレントニューラルネットワーク構造を用いた崩壊点の生成

自然な発想として、今、飛跡について 1 事象分 (系列) の全ての情報を持っている為、エンコーダー・デコーダーモデルとすることで、その事象についての情報 (コンテキスト) を活用することができる。更に、エンコーダー・デコーダーモデルの間に Attention を組み込むことも同様に自然な発想である。その様なネットワークを図 3.8 に示す。この図では上から飛跡が入力されており、崩壊点の種である飛跡対は上部左右と左下から入力されている。上部がエンコーダー部、下部がデコーダ部である。個々の基本的な構造は図 3.7 と同一であるが、エンコーダー部には双方向リカレントニューラルネットワークを採用している。また、デコーダー部では独自のリカレントニューラルネットワークの構造を更に拡張し、エンコーダー部の出力を初期状態の一つとして Attention に対応させたネットワークを使用している。詳細は次項 3.4.2 にて述べる。

Attention を組み込むことによって、デコーダー部の”ある”飛跡はエンコーダー部によって抽出された飛跡の情報に任意の注意を払って、1 事象分の情報を取得することになる。エンコーダー・デコーダーモデルに拡張しても、このネットワークの基本構造がリカレントニューラルネットワークであることに変わりがない為、デコーダー部の飛跡の本数を任意に変えることが可能である。

3.4.2 ネットワークの詳細な構造

3.4.3 ネットワークの学習と戦略

訓練データ ゼロパディング

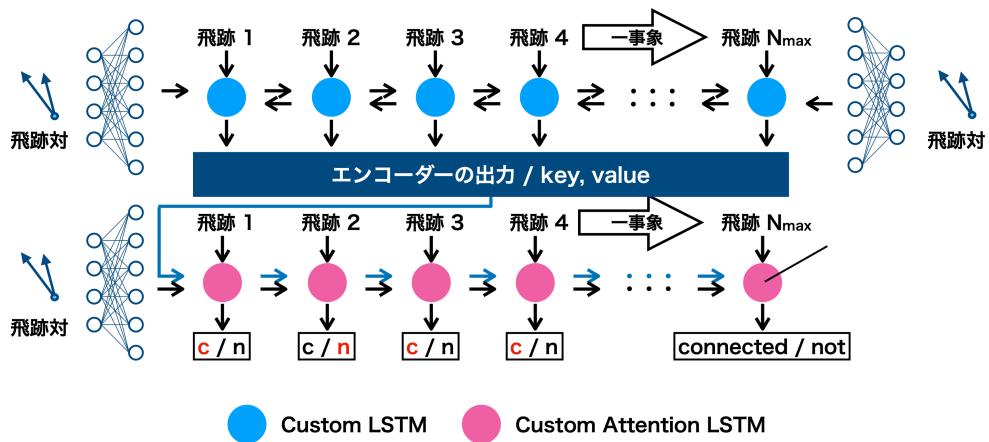


図 3.8: Attention を用いたエンコーダー・デコーダーモデルへの拡張

飛跡順のシャッフル

3.4.4 ネットワークの性能

ハイパーパラメータ・チューニング

事象についての情報による変化

Attention weight

bb, cc データセット

Layer (type)	Output Shape	Param #	Connected to
<hr/>			
Pair_Input (InputLayer)	[(None, 44)]	0	
Encoder_Dense_1 (Dense)	(None, 256)	11520	Pair_Input[0][0]
Encoder_b_Dense_1 (Dense)	(None, 256)	11520	Pair_Input[0][0]
Encoder_Activation_1 (Activation)	(None, 256)	0	Encoder_Dense_1[0][0]
Encoder_b_Activation_1 (Activation)	(None, 256)	0	Encoder_b_Dense_1[0][0]
Encoder_Input (InputLayer)	[(None, 53, 23)]	0	
Encoder_Dense_2 (Dense)	(None, 256)	65792	Encoder_Activation_1[0][0]
Encoder_b_Dense_2 (Dense)	(None, 256)	65792	Encoder_b_Activation_1[0][0]
Decoder_Dense_1 (Dense)	(None, 256)	11520	Pair_Input[0][0]
time_distributed (TimeDistribut	(None, 53, 256)	6144	Encoder_Input[0][0]
Encoder_Activation_2 (Activation)	(None, 256)	0	Encoder_Dense_2[0][0]
Encoder_b_Activation_2 (Activation)	(None, 256)	0	Encoder_b_Dense_2[0][0]
Decoder_Activation_1 (Activation)	(None, 256)	0	Decoder_Dense_1[0][0]
Decoder_Input (InputLayer)	[(None, None, 23)]	0	
bidirectional (Bidirectional)	(None, 53, 512)	1050624	time_distributed[0][0] Encoder_Activation_2[0][0] Encoder_Activation_2[0][0] Encoder_b_Activation_2[0][0] Encoder_b_Activation_2[0][0]
Decoder_Dense_2 (Dense)	(None, 256)	65792	Decoder_Activation_1[0][0]
time_distributed_1 (TimeDistrib	(None, None, 256)	6144	Decoder_Input[0][0]
reshape (Reshape)	(None, 27136)	0	bidirectional[0][0]
Decoder_Activation_2 (Activation)	(None, 256)	0	Decoder_Dense_2[0][0]
Decoder_Attention_VLSTM (RNN)	[(None, None, 1), (N 1246976	time_distributed_1[0][0] reshape[0][0] Decoder_Activation_2[0][0]	
<hr/>			
Total params: 2,541,824			
Trainable params: 2,541,824			
Non-trainable params: 0			

図 3.9: エンコーダー・デコーダーモデルにおける各種パラメーターの出力

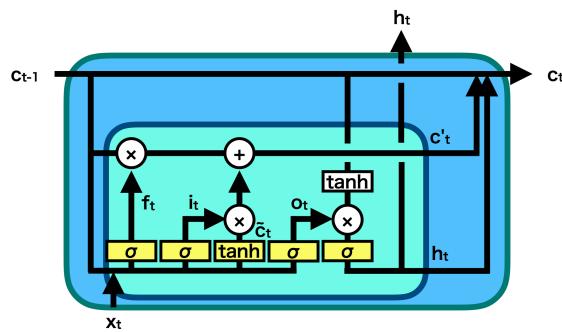


図 3.10: 自作リカレントニューラルネットワークの構造

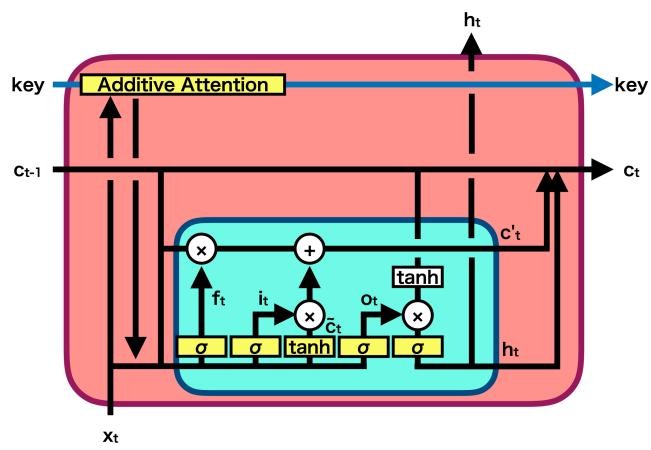


図 3.11: Attention を組み込んだ自作リカレントニューラルネットワークの構造

第 4 章

深層学習を用いた崩壊点検出

本章では、3 章で作成したネットワークを使用した崩壊点検出について紹介する。4.1 節で具体的なアルゴリズムについて概説し、4.1.2 項で Primary Vertex の再構成について、4.1.3 項で Secondary Vertex の再構成についての評価と調整をそれぞれ述べる。最終的な性能や評価方法については 4.2 節でまとめる。

4.1 崩壊点検出アルゴリズム

4.1.1 飛跡対についてのネットワークの拡張

4.1.2 Primary Vertex の再構成

4.1.3 Secondary Vertex の再構成

4.2 深層学習を用いた崩壊点検出の性能

4.2.1 時間計測

4.2.2 エネルギーの変化

第 5 章

現行の手法との比較

本章では、深層学習を用いた崩壊点検出と現行 (LCFIPlus) の崩壊点検出との比較を行う。まず 5.1 節では崩壊点検出単体での性能の比較をする。次に更なる比較の為の C++ への移行について 5.2 節で述べる。そのように LCFIPlus に実装された本研究の崩壊点検出を用いたフレーバータギングの性能までの詳細な性能の比較と評価を 5.3 節にて行う。

5.1 崩壊点検出単体での比較

5.2 C++ での推論

5.3 詳細な比較と評価

第 6 章

結論と今後の展望

謝辞

付録 A

ソースコード

A.1 飛跡対についてのネットワーク

A.2 任意の数の飛跡についてのネットワーク

A.3 崩壊点の再構成

付録 B

分散深層学習

付録 C

SiW-ECAL

参考文献

- [1] F. Hogehoge, Sample article, Nuclear Instruments and Methods in Physics Research A (2016).
- [2] Ilc photo gallery, <http://ilcgallery.com/term-of-use>.
- [3] **KEK International Working Group**, K. Desch *et al.*, Recommendations on ILC Project Implementation, (2019).
- [4] H. Baer *et al.*, The international linear collider technical design report - volume 2: Physics, 2013, 1306.6352.
- [5] P. Bambade *et al.*, The international linear collider: A global project, 2019, 1903.01629.
- [6] T. I. Collaboration, International large detector: Interim design report, 2020, 2003.01116.
- [7] ilcsoft, <https://github.com/iLCSof>.
- [8] lcfiplus/lcfiplus: Flavor tagging code for ilc detectors, <https://github.com/lcfiplus/LCFIPlus>.
- [9] T. Suehara and T. Tanabe, Lcfiplus: A framework for jet analysis in linear collider studies, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **808**, 109 – 116 (2016).
- [10] 斎. 康毅, ゼロから作る Deep Learning: Python で学ぶディープラーニングの理論と実装 ゼロから作る Deep Learning (オライリー・ジャパン, 2016).
- [11] 斎. 康毅, ゼロから作る Deep Learning 2: 自然言語処理編ゼロから作る Deep Learning (オライリー・ジャパン, 2018).
- [12] S. Raschka, クイープ, and 真. 福島, Python 機械学習プログラミング : 達人データサイエンティストによる理論と実践 Impress top gear (インプレス, 2016).
- [13] J. McCarthy, M. Minsky, N. Rochester, and C. E. Shannon, A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955., AI Magazine **27**, 12 (2006).
- [14] V. VAPNIK, Pattern recognition using generalized portrait method, Automation and Remote Control **24**, 774 (1963).
- [15] B. E. BOSE, A training algorithm for optimal margin classifiers, Proceedings

- of the 5th Annual ACM Workshop on Computational Learning Theory, Pittsburgh, Pennsylvania, United States (1992-7) (1992).
- [16] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review* **65**, 386 (1958).
 - [17] G. CYBENKO, Approximation by superpositions of a sigmoidal function, *Mathematics of Control, Signals and Systems* **2**, 303 (1989).
 - [18] Tensorflow, <https://www.tensorflow.org/?hl=ja>.
 - [19] Home - keras documentation, <https://keras.io/ja/>.
 - [20] Pytorch, <https://pytorch.org/>.
 - [21] Caffe — deep learning framework, <http://caffe.berkeleyvision.org/>.
 - [22] B. Widrow and M. E. Hoff, Adaptive switching circuits, 1960 IRE WESCON Convention Record , 96 (1960), Reprinted in *Neurocomputing* MIT Press, 1988 .
 - [23] T. Tieleman and G. Hinton, Lecture 6.5—rmsProp: Divide the gradient by a running average of its recent magnitude, COURSERA: Neural Networks for Machine Learning, 2012.
 - [24] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, 2017, 1412.6980.
 - [25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning representations by back-propagating errors, *Nature* **323**, 533 (1986).
 - [26] G. E. Hinton and R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, *Science* **313**, 504 (2006).
 - [27] I. J. Goodfellow *et al.*, Generative adversarial networks, 2014, 1406.2661.
 - [28] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Computation* **9**, 1735 (1997), <https://doi.org/10.1162/neco.1997.9.8.1735>.
 - [29] K. Cho *et al.*, Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014, 1406.1078.
 - [30] D. Bahdanau, K. Cho, and Y. Bengio, Neural machine translation by jointly learning to align and translate, 2016, 1409.0473.
 - [31] M.-T. Luong, H. Pham, and C. D. Manning, Effective approaches to attention-based neural machine translation, 2015, 1508.04025.
 - [32] A. Vaswani *et al.*, Attention is all you need, 2017, 1706.03762.
 - [33] Goto-k/vertexfinderwithdl, <https://github.com/Goto-K/VertexFinderwithDL>, (Accessed on 11/07/2020).
 - [34] T. Kramer, Track parameters in LCIO, (2006).