2024年度後期　修士論文

# 深層学習による口唇音声変換に関する研究

# A Study on the Conversion from Lip-Video to Speech Using a Deep Learning Technique

2025年月日

九州大学芸術工学府音響設計コース

2DS23095M

南 汰翼

MINAMI Taisuke

研究指導教員　鏑木 時彦　教授

# 概要

# 目 次

# 1　序論

## 1.1　背景

音声は基本的なコミュニケーション手段として，人々の日常生活において重要な役割を果たしている．しかし，癌などの病気で喉頭を摘出すると，声帯振動による音声生成が不可能になり，従来の発声手段を失う．このような場合の代用音声手法として，電気式人工喉頭や食道発声，シャント発声があるが，自然で聞き取りやすい音声を生成することが難しかったり，習得に訓練を必要としたり，器具の交換のための定期的な手術を必要とするといった課題がある．これに対して本研究では，新たな代用音声手法として，深層学習を活用して口唇の動きと音声波形の関係性を学習することで，口唇の動きから音声を合成するアプローチを提案する．この手法により，訓練や手術を必要とせずとも，自然な声でのコミュニケーションを可能にすることを目指す．

近年の動画音声合成では，LRS3[?] や VoxCeleb2[?] などの大規模データセットを活用して構築された，自己教師あり学習（Self Supervised Learning; SSL）モデルを FineTuning することの有効性が示されている．特に，近年の動画音声合成においては AVHuBERT[?] というモデルが動画からの特徴抽出器として採用される場合が多い．AVHuBERT は，動画と音声の対応関係を Masked Prediction という自己教師あり学習方法により捉えたモデルであり，Modality Dropout という工夫により，自己教師あり学習時には動画と音声の両方を入力とするものの，FineTuning 時には動画あるいは音声のみを入力とできる柔軟性を持ち合わせている．これに加え，従来の動画音声においては動画からの予測対象としてメルスペクトログラムが選択されることが多かったが，近年ではこれにテキストや，音声 SSL モデルである HuBERT を利用して得られた離散特徴量を合わせて予測対象とする，マルチタスク学習の有効性が示されている [?, ?].

## 1.2　目的

本研究では，動画音声合成モデルによって得られる合成音声の品質が依然として低く，自然音声に迫る合成音の実現が困難である点を課題とする．この課題に対し，近年高い精度を達成した手法 [?] における,「AVHuBERT を利用したメルスペクトログラムと HuBERT 離散特徴量を予測対象とするマルチタスク学習手法」をベースラインとして採用し，この性能を上回る新たなモデルを提案することを目的とする．

## 1.3　本論文の構成

# 2　音声信号処理

音声にはフォルマントや基本周波数（ピッチ）など，様々な周波数的な特徴が存在している．フォルマントは母音や子音を知覚するため，ピッチはアクセントやイントネーションを表現するために重要なものである．このような音声信号の持つ複雑さから，時間波形のままその特徴を分析することは困難である．これに対し，本節では音声の特徴を捉えやすくするための信号処理について説明する．

## 2.1　音声のフーリエ変換

音声の時間波形に対して，周波数領域での情報を得るためにはフーリエ変換（Fourier Transform）を使用する．特に，音声はマイクロフォンで収録され，コンピュータ内で処理されることが多い．この時，音声はアナログ信号ではなく，サンプリング周波数と量子化ビット数に従って離散化されたデジタル信号として扱われる．このような場合，離散信号に対してのフーリエ変換である離散フーリエ変換（discrete Fourier transform; DFT）が用いられる．また，信号の系列長をゼロパディングして 2 の冪乗の長さに調整することで，計算量を抑えた高速フーリエ変換（fast Fourier transform; FFT）を用いることができる．

離散信号を $x[n]$，それに対するフーリエ変換を $X[k]$ とする．ここで，n はサンプルのインデックス，k は周波数インデックスである．$X[k]$ は複素数であり，以下のように極座標表示することができる．

$$X[k] = \mathrm{Re}(X[k]) + j\mathrm{Im}(X[k]) \tag{2.1}$$

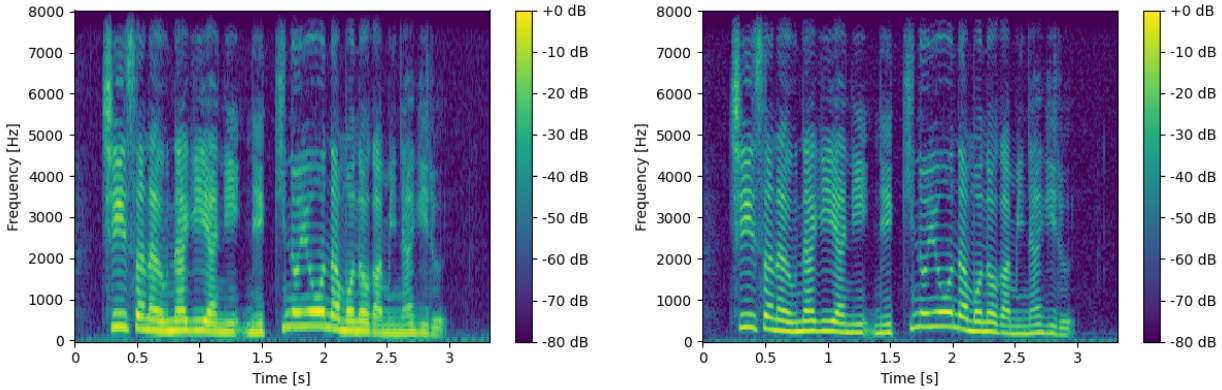
$$= |X[k]|e^{j\angle \mathrm{X[k]}} \tag{2.2}$$

ここで，$|X[k]|$ は振幅特性（振幅スペクトル），$\angle$X[k] は位相特性（位相スペクトル）であり，

$$|X[k]| = \sqrt{\mathrm{Re}(X[k])^2 + \mathrm{Im}(X[k])^2} \tag{2.3}$$

$$\angle \mathrm{X[k]} = \tan^{-1} \frac{\mathrm{Im}(X[k])}{\mathrm{Re}(X[k])} \tag{2.4}$$

と表される．また，$|X[k]|^2$ はパワースペクトルと呼ばれる．これにより，信号中にどのような周波数成分がどれくらい含まれているかを調べることができる．しかし，音声はフォルマントやピッチが時々刻々と変化するため，信号全体に対して直接フーリエ変換を適用したとしても有用な結果が得られない．このような音声の持つ非定常性の問題に対して，十分短い時間幅においては信号の定常性が成り立つという仮定のもと，短時間フーリエ変換（short-time Fourier transform; STFT）が用いられる．STFT では，音声信号に対して窓関数による窓処理を適用し，短時間に区切られた信号それぞれに対して DFT を適用する．ここで，窓処理とはある特定の窓関数と音声信号を時間領域でかけ合わせることであり，窓関数の時間幅を窓長という．また，窓関数を時間方向にシフトするときの時間幅をシフト幅という．STFT には時間分解能と




(a) 窓長 12.5ms，シフト幅 5ms

(b) 窓長 25ms，シフト幅 10ms

(c) 窓長 50ms，シフト幅 20ms

(d) 窓長 100ms，シフト幅 40ms

図 2.1: 「小さな鰻屋に，熱気のようなものがみなぎる」と発話した音声から計算された対数パワースペクトログラム

周波数分解能の間に不確定性が存在し，両者の間にトレード・オフの関係がある．窓長が長い場合には周波数分解能が向上する一方，時間分解能が低下する．窓長が短い場合にはその逆となる．音声信号 $x[n]$ の STFT を時刻 $j$，周波数インデックスを $k$ として $X[j,k]$ と表すと，$X[j,k]$ は時間周波数領域における複素数となる．これを複素スペクトログラムと呼ぶ．また，$|X[j,k]|$ を振幅スペクトログラム，$\angle \mathrm{X[j,k]}$ を位相スペクトログラム，$|X[j,k]|^2$ をパワースペクトログラムと呼ぶ.「小さな鰻屋に，熱気のようなものがみなぎる」と発話した音声に対し，窓関数としてハニング窓を用いた上で，複数の窓長・シフト幅によって計算した対数パワースペクトログラムを，図 **??**に示す．窓長が 100ms と長い場合には周波数分解能が高いが，時間分解能が低下することでスペクトルの時間変化が滑らかでないことがわかる．一方，窓長が 12.5ms と短い場合には時間分解能が高いが，周波数分解能が低下することでスペクトルがぼやけていることがわかる．これが窓長に対する時間分解能と周波数分解能とトレード・オフであり，窓長 25ms や 50ms が程よいパラメータであることがわかる．

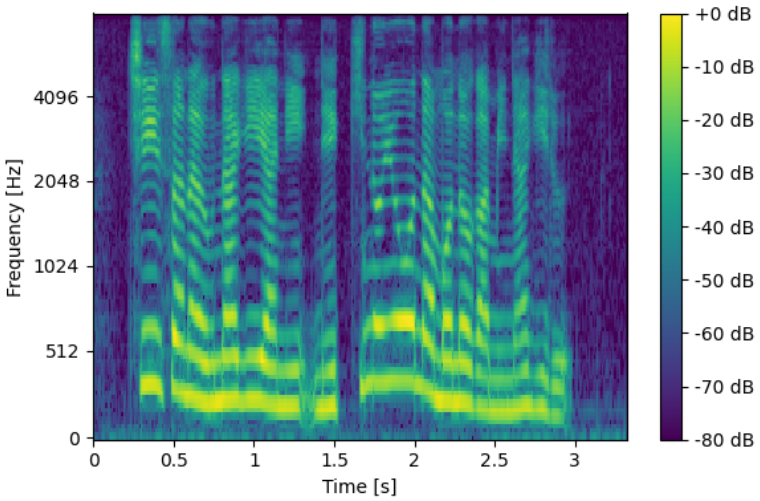



図 2.2: 「小さな鰻屋に，熱気のようなものがみなぎる」と発話した音声に対する対数メルスペクトログラム

## 2.2 メルスペクトログラム

メルスペクトログラムは，パワースペクトログラムを人間の聴感特性を考慮したメル尺度に変換することによって得られる．周波数軸をメル尺度に変換する際，以下の式を用いる．

$$\mathrm{Mel}(f) = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{2.5}$$

メル尺度は，1000 Hz，40 dB の純音を 1000 mel とする比率尺度である．メル尺度を用いることにより，低い周波数ほど細かく，高い周波数ほど荒い特徴量になる．メルスペクトログラムは，パワースペクトログラムに対してメルフィルタバンクを適用することによって得られる．メルフィルタバンクの数は任意に決定できるパラメータであり，メルスペクトログラムの周波数方向の次元はこれに一致する．音声合成においては，音声のサンプリング周波数を 16 kHz とするとき，メルフィルタバンクの数を 80 とし，8000 Hz までの帯域に対して適用することが多い．「小さな鰻屋に，熱気のようなものがみなぎる」と発話した音声に対し，窓関数にハニング窓を用い，窓長 25ms，シフト幅 10ms としてパワースペクトログラムを計算した上で，80 次元のメルフィルタバンクを適用して得られた対数メルスペクトログラムを，図 **??**に示す．

# 3　深層学習

深層学習とは，人間の神経細胞の仕組みを模擬したニューラルネットワークを用いる機械学習手法のことである．特に近年ではその層を深くしたディープニューラルネットワーク（Deep Neural Network; DNN）が用いられ，大量のパラメータによる表現力により，自然言語処理や画像処理，音声認識や音声合成など様々な分野で成果を上げている．本章では，DNN の構成要素及び，構築した DNN の学習方法について説明する．

## 3.1　DNN の構成要素

### 3.1.1　全結合層

全結合層は，入力に対して線型変換を施す層である．全結合層への入力を $\boldsymbol{x} \in \mathbb{R}^{D_{\text{in}}}$ とすると，出力 $\boldsymbol{y} \in \mathbb{R}^{D_{\text{out}}}$ は，

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \tag{3.1}$$

で与えられる．ここで，$\boldsymbol{W} \in \mathbb{R}^{D_{\text{out}} \times D_{\text{in}}}$ は重み，$\boldsymbol{b} \in \mathbb{R}^{D_{\text{out}}}$ はバイアスである．全結合層は DNN 内部での特徴量の次元の変換や，最終層において所望の出力に次元を合わせるのに用いられる．

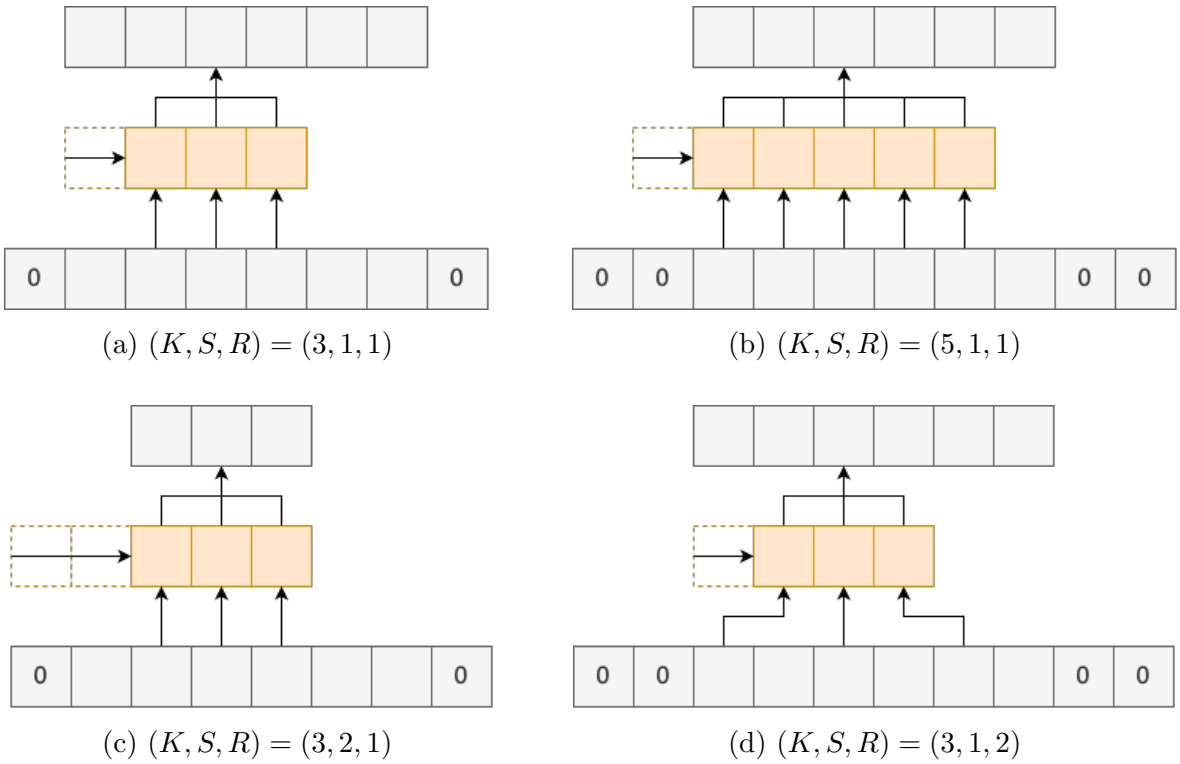
### 3.1.2　畳み込み層

畳み込み層は，入力に対して畳み込み演算を行う層である．一次元畳み込み層について，入力を $\boldsymbol{X} \in \mathbb{R}^{D_{\text{in}} \times T_{\text{in}}}$，出力を $\boldsymbol{Y} \in \mathbb{R}^{D_{\text{out}} \times T_{\text{out}}}$ とし，それぞれ $d$ 次元目，$t$ 番目の成分を $x_{d_{\text{in}},t}, y_{d_{\text{out}},t}$ で表す．このとき，$y_{d_{\text{out}},t}$ は，

$$y_{d_{\text{out}},t} = b_{d_{\text{out}}} + \sum_{d_{\text{in}}=1}^{D_{\text{in}}} \sum_{k=1}^{K} x_{d_{\text{in}},t-\left\lfloor \frac{K}{2} \right\rfloor+k} w_{d_{\text{in}},d_{\text{out}},k} \tag{3.2}$$

で与えられる．ここで，$K \in \mathbb{N}$ はカーネルサイズ，$w_{d_{\text{in}},d_{\text{out}},k} \in \mathbb{R}$ は入力の $d_{\text{in}}$ 次元目から出力の $d_{\text{out}}$ 次元目に割り当てられたカーネルの $k$ 番目の成分，$b_{d_{\text{out}}} \in \mathbb{R}$ は出力の $d_{\text{out}}$ 次元目に割り当てられたバイアスである．上式より，一次元畳み込み層の $t$ 番目の出力は，$t$ 番目を中心としたカーネルサイズの範囲分の入力から計算されることがわかる．これより，畳み込み層は入力の局所的な特徴を抽出するのに適した層だと考えられる．一次元畳み込みはテキストや音声など，データの形状が $(D \times T)$ となっている時に用いられる．ここで，$D$ は次元，$T$ は系列長である．

これに加えて，カーネルを二次元配列とすれば二次元畳み込み層，三次元配列とすれば三次元畳み込み層となる．二次元畳み込み層は画像など，データの形状が $(D \times H \times W)$ となっている場合に用いられる．ここで，$H$ は高さ，$W$ は幅に相当する．三次元畳み込み層は動画など，データの形状が $(D \times H \times W \times T)$ となっている場合に用いられる．




(a) $(K, S, R) = (3, 1, 1)$　(b) $(K, S, R) = (5, 1, 1)$

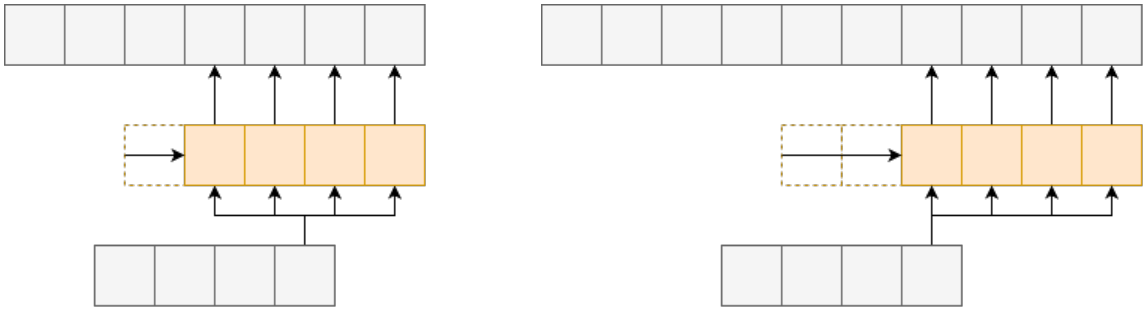
(c) $(K, S, R) = (3, 2, 1)$　(d) $(K, S, R) = (3, 1, 2)$

図 3.1: ある次元における一次元畳み込み層の処理．$K$ はカーネルサイズ，$S$ はストライド，$R$ はダイレーションを表し，図中の 0 はパディング部を表す．

畳み込み層における主要なパラメータは三つある．一つ目はカーネルサイズであり，これによって考慮できる入力特徴量の範囲が定まる．二つ目はストライドであり，これによってカーネルのシフト幅を設定できる．三つ目はダイレーションであり，これは畳み込み演算において計算対象となる入力特徴量の間隔を表す．ダイレーションを大きくすることで，カーネルサイズが同じでも考慮できる入力特徴量の範囲を広げることが可能である．また，出力系列長 $T_{\text{out}}$ を入力系列長 $T_{\text{in}}$ の整数倍に保つには，上記のパラメータに対して適切なパディング長を指定する必要がある．例えば，カーネルサイズを 3，ストライドとダイレーションを 1 とした場合には，入力の両端に 1 ずつゼロパディングすれば良い．図 ??に，ある次元における一次元畳み込み層の処理を示す．

### 3.1.3　転置畳み込み層

転置畳み込み層は，畳み込み層の逆演算に対応する層であり，主に入力のアップサンプリングに使用される．図 ??に，ある入出力チャンネル間における一次元転置畳み込み層の処理を示す．一次元転置畳み込み層では，$t$ 番目の入力とカーネルの積を計算し，その結果を $t$ 番目から $t + K$ 番目までの出力とする．ここで $K$ はカーネルサイズである．また，複数の入力から計算された出力がオーバーラップする場合，これらは加算される．図 ??は，カーネルサイズを 4，ストライドを 1 とした場合の様子である．アップサンプリングを行いたい場合は，ストライド



(a) $(K, S) = (4, 1)$　　(b) $(K, S) = (4, 2)$

図 3.2: ある次元における一次元転置畳み込み層の処理．$K$ はカーネルサイズ，$S$ はストライドを表す．

を 2 以上とすれば良い．図 **??**にカーネルサイズを 4，ストライドを 2 とした場合を示す．この時，入力系列長が 4 であるのに対して，出力系列長が 10 まで拡大されていることがわかる．ここで，出力系列長を入力系列長の整数倍に保つためには，出力の両端の削除数を適切に設定する必要がある．上述の例では両端の削除数を 1 とすることで，出力系列長を入力系列長の 2 倍である 8 に調整できる．

### 3.1.4　活性化関数

活性化関数は，ニューラルネットワークの出力に非線形性を与えるための関数である．これにより，DNN は単純な線形変換だけでは表現できない複雑な入出力の関係を学習可能になる．以下，活性化関数への入力を $x \in \mathbb{R}$ として，代表的なものを六つ述べる．また，本節で取り上げる活性化関数とその一階導関数のグラフを図 **??**に示す．

一つ目は，シグモイド関数である．シグモイド関数は

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{3.3}$$

で与えられ，その一階導関数は

$$\frac{d}{dx}\sigma(x) = \frac{\exp(-x)}{(1 + \exp(-x))^2} \tag{3.4}$$

となる．図 **??**より，シグモイド関数の一階導関数の最大値は $x = 0$ における 0.25 であり，$|x - 0|$ が大きくなるのに伴ってその値は小さくなることがわかる．DNN の各重みは，損失関数の勾配を利用することで更新されるから，シグモイド関数以前の層の重みにおける勾配は，シグモイド関数の一階導関数の値が乗算された結果となる．前述したように，シグモイド関数は一階導関数の値が小さくなりがちであるから，それ以前の層における勾配も小さくなり，重みの更新が進みづらくなる可能性がある．この問題を，勾配消失と呼ぶ．

二つ目は，tanh 関数である．tanh は

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{3.5}$$

で与えられ，その一階導関数は

$$\frac{d}{dx}\tanh(x) = \frac{4}{(\exp(x)+\exp(-x))^2} \tag{3.6}$$

$$= \frac{1}{\cosh(x)^2} \tag{3.7}$$

となる．tanh の値域は $[-1,1]$ となっており，図??より $|x-0|$ が小さいところではシグモイド関数より一階導関数の値が大きくなっていることがわかる．しかし，$|x-0|$ が大きくなればシグモイド関数と同様に一回導関数の値が小さく，勾配消失のリスクを抱えていることがわかる．

三つ目は，ReLU である．ReLU は

$$\mathrm{ReLU}(x) = \max(0, x) \tag{3.8}$$

で与えられ，その一階導関数は

$$\frac{d}{dx}\mathrm{ReLU}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x <= 0 \end{cases} \tag{3.9}$$

となる．ReLU は入力が 0 以上であれば恒等写像として振る舞うが，0 未満であれば 0 に写す．一階導関数は 0 あるいは 1 のみを取り，特に入力が正の値であれば常に微分係数は 1 となることから，シグモイド関数や tanh よりも勾配消失が起こりづらい．ReLU は現在，標準的な活性化関数として広く用いられている．しかし，ReLU への入力が 0 未満の値を取るとき，ReLU 入力についての出力の勾配は 0 になるから，ReLU 以前の層の重みが更新されず，学習が遅くなる可能性がある．

四つ目は，LeakyReLU[?] である．LeakyReLU は

$$\mathrm{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{if } x <= 0 \end{cases} \tag{3.10}$$

で与えられ，その一階導関数は

$$\frac{d}{dx}\mathrm{LeakyReLU}(x) = \begin{cases} 1 & \text{if } x > 0 \\ a & \text{if } x <= 0 \end{cases} \tag{3.11}$$

となる．ReLU と比較すると，0 未満の入力に対しても 0 でない値を出力し，一階導関数も 0 にならない点が異なっている．これにより，重みの更新が進まなくなる ReLU の課題を解消した．

五つ目は，PReLU[?] である．これは，LeakyReLU と似た活性化関数であるが，LeakyReLU のパラメータ $a$ を学習可能にすることで，その他の層と合わせて最適化が可能となったことが特徴である．

六つ目は，GELU[?] である．GELU は

$$\mathrm{GELU}(x) = x\Phi(x) \tag{3.12}$$

で与えられる．ここで，

$$\Phi(x) = P(X \leq x),\ X \sim \mathcal{N}(0,1) \tag{3.13}$$

である．GELU の一階導関数は，

$$\frac{d}{dx}\mathrm{GELU}(x) = \Phi(x) + \frac{x}{\sqrt{2\pi}}\exp\left(-\frac{x^2}{2}\right) \tag{3.14}$$

となる．GELU は，ReLU が入力に対して 0 あるいは 1 を確定的にかける活性化関数と捉えた上で，これを入力に依存した確率的な挙動に変更したものである．実際，$m \sim \mathrm{Bernoulli}(\Phi(x))$ とすると，

$$\begin{aligned}
\mathrm{GELU}(x) &= x\Phi(x) && (3.15)\\
&= 1x \cdot \Phi(x) + 0x \cdot (1 - \Phi(x)) && (3.16)\\
&= \mathbb{E}[mx] && (3.17)
\end{aligned}$$

となり，GELU の出力は確率的なバイナリマスク $m$ を入力 $x$ にかけた，$mx$ の期待値に等しいことがわかる．

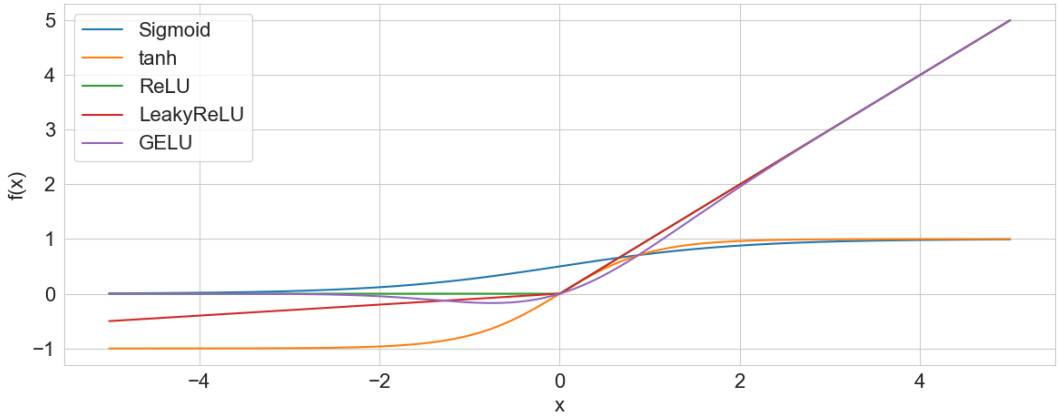
### 3.1.5 再帰型ニューラルネットワーク

再帰型ニューラルネットワーク（Recurrent Neural Network; RNN）は，自身の過去の出力を保持し，それをループさせる再帰的な構造を持ったネットワークである．

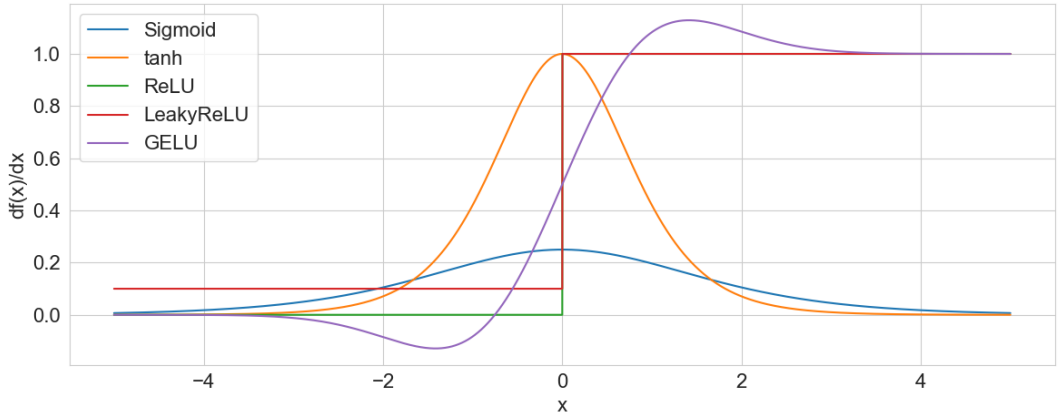
近年よく用いられる RNN として，長・短期記憶（Long Short-Time Memory; LSTM）[?] がある．LSTM は入力ゲート，忘却ゲート，出力ゲートの 3 つを持ち，これらゲートによってネットワーク内部の情報の取捨選択を行うことで，長い系列データからの学習を可能にした．LSTM のネットワーク内部で行われる計算を以下に示す．

$$\begin{aligned}
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_f[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_f) && (3.18)\\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_i[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_i) && (3.19)\\
\tilde{\boldsymbol{c}}_t &= \tanh(\boldsymbol{W}_h[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_h) && (3.20)\\
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t && (3.21)\\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_o[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_o) && (3.22)\\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) && (3.23)
\end{aligned}$$

ここで，$\boldsymbol{x}_t \in \mathbb{R}^{D_{\mathrm{in}}}$ は時刻 $t$ の入力，$\boldsymbol{f}_t \in [0,1]^{D_{\mathrm{out}}}$ は忘却ゲートの出力，$\boldsymbol{i}_t \in [0,1]^{D_{\mathrm{out}}}$ は入力ゲートの出力，$\boldsymbol{c}_t \in [-1,1]^{D_{\mathrm{out}}}$ が時刻 $t$ におけるセルの状態，$\boldsymbol{o}_t \in [0,1]^{D_{\mathrm{out}}}$ が出力ゲートの出力，$\boldsymbol{h}_t \in [-1,1]^{D_{\mathrm{out}}}$ が時刻 $t$ における隠れ状態を表す．また，$[\cdot,\cdot]$ はチャンネル方向の結合，$\odot$ は要素積を表す．$\boldsymbol{W}_f, \boldsymbol{W}_i, \boldsymbol{W}_h, \boldsymbol{W}_o \in \mathbb{R}^{D_{\mathrm{out}} \times (D_{\mathrm{in}} + D_{\mathrm{out}})}$ は重み，$\boldsymbol{b}_f, \boldsymbol{b}_i, \boldsymbol{b}_h, \boldsymbol{b}_o \in \mathbb{R}^{D_{\mathrm{out}}}$ はバイアスである．忘却ゲート出力 $\boldsymbol{f}_t$ が前時刻のセル状態 $\boldsymbol{c}_t$ に含まれる情報の選択，入力ゲート出力 $\boldsymbol{i}_t$




(a) 活性化関数




(b) 活性化関数の一階導関数

図 3.3: 活性化関数の例

が新たな入力 $\tilde{\boldsymbol{c}}_t$ に含まれる情報の選択に用いられ，$\boldsymbol{c}_t$ が決まる．その後，出力ゲート出力 $\boldsymbol{o}_t$ が $\boldsymbol{c}_t$ に含まれる情報の選択に用いられ，$\boldsymbol{h}_t$ が決まる．

また，LSTM が 3 つのゲートを必要とするのに対し，ゲートを 2 つに減らすことでネットワークの軽量化を図ったのがゲート付き回帰型ユニット（Gated Recurrent Unit; GRU）[?] である．GRU はリセットゲートと更新ゲートの 2 つを用いて隠れ状態を更新する．GRU のネットワーク内部で行われる計算を以下に示す．

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_z) \tag{3.24}$$

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_r) \tag{3.25}$$

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_h[\boldsymbol{x}_t, \boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}] + \boldsymbol{b}_h) \tag{3.26}$$

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t \tag{3.27}$$

ここで，$\boldsymbol{x}_t \in \mathbb{R}^{D_{\text{in}}}$ が時刻 $t$ における入力，$\boldsymbol{z}_t \in [0,1]^{D_{\text{out}}}$ が更新ゲートの出力，$\boldsymbol{r}_t \in [0,1]^{D_{\text{out}}}$ がリセットゲートの出力，$\boldsymbol{h}_t \in [-1,1]^{D_{\text{out}}}$ が時刻 $t$ における隠れ状態を表す．また，$\boldsymbol{W}_z, \boldsymbol{W}_r, \boldsymbol{W}_h \in$

$\mathbb{R}^{D_{\text{out}} \times (D_{\text{in}}+D_{\text{out}})}$ は重み，$\boldsymbol{b}_z, \boldsymbol{b}_r, \boldsymbol{b}_h \in \mathbb{R}^{D_{\text{out}}}$ はバイアスである．更新ゲート出力 $\boldsymbol{z}_t$ が $\boldsymbol{h}_{t-1}$ と $\tilde{\boldsymbol{h}}_t$ に含まれる情報の選択，リセットゲート出力 $\boldsymbol{r}_t$ が $\boldsymbol{h}_{t-1}$ に含まれる情報の選択に用いられる．

### 3.1.6 正規化層

DNN の学習過程では学習の進行に伴って重みが変化するため，その度に各層への入力の分布が変わってしまう．これは内部共変量シフトと呼ばれ，ネットワークの学習を不安定にする原因となる．これに対し，バッチ正規化（Batch Normalization）[?] が有効である．バッチ正規化は，ミニバッチ内における入力特徴量の期待値と分散を次元ごとに計算し，これらを用いて入力特徴量を次元ごとに標準化するものである．ここで，バッチサイズを $N$，バッチ正規化への入力特徴量を $\boldsymbol{x}_i \in \mathbb{R}^D$ $(n = 1, \ldots, N)$，出力特徴量を $\boldsymbol{y}_n \in \mathbb{R}^D$ $(n = 1, \ldots, N)$ とする．このとき，各 $n$ に対し入力特徴量 $\boldsymbol{x}_n$ の $d$ 次元目の成分を $x_{n,d}$，出力特徴量 $\boldsymbol{y}_n$ の $d$ 次元目の成分を $y_{n,d}$ とすると，$y_{n,d}$ は

$$\mu_d^B = \frac{1}{N} \sum_{n=1}^{N} x_{n,d} \tag{3.28}$$

$$\left(\sigma_d^B\right)^2 = \frac{1}{N} \sum_{n=1}^{N} \left(x_{n,d} - \mu_d^B\right)^2 \tag{3.29}$$

$$\tilde{x}_{n,d} = \frac{x_{n,d} - \mu_d^B}{\sqrt{\left(\sigma_d^B\right)^2 + \epsilon}} \tag{3.30}$$

$$y_{n,d} = \gamma_d \tilde{x}_{n,d} + \beta_d \tag{3.31}$$

で与えられる．ここで，$\gamma_d, \beta_d \in \mathbb{R}$ は学習可能なスカラーであり，$\epsilon \in \mathbb{R}$ はゼロ割を避けるためのスカラーである．バッチ正規化では $\gamma_d, \beta_d$ によって表現力を向上させており，実際

$$\gamma_d = \sqrt{\left(\sigma_d^B\right)^2 + \epsilon} \tag{3.32}$$

$$\beta_d = \mu_d^B \tag{3.33}$$

とすれば，標準化前の入力を再び得ることが可能である．学習時は，サンプルの標準化に用いる統計量とは別に，期待値の移動平均と不偏分散の移動平均を計算しておく．推論時は学習終了時に得られたこれら移動平均の値を用いることで，確率的な挙動を持たないよう工夫されている．

バッチ正規化は DNN の学習の安定化に貢献する一方，ミニバッチ全体における統計量を利用するため，バッチサイズが小さい場合はデータの分布を安定させることが難しくなる．また，テキストや音声といった系列長を持つデータを扱う場合，ミニバッチを構成するためにはゼロパディングによって系列長を揃える必要がある．この時，RNN の各ステップの出力に対しバッチ正規化を適用すると，ゼロパディングによって人為的に系列量を揃えているから，統計量が実際のデータの分布からかけ離れたものになる可能性がある．これら課題に対し，ミニバッチ内

の各サンプルごとに期待値と分散を求めて標準化する，レイヤー正規化（Layer Normalization）[?] がある．バッチ正規化のときと同様の表記を用いると，$y_{n,d}$ は

$$\mu_n^L = \frac{1}{D}\sum_{d=1}^{D} x_{n,d} \tag{3.34}$$

$$\left(\sigma_n^L\right)^2 = \frac{1}{D}\sum_{d=1}^{D}\left(x_{n,d} - \mu_n^L\right)^2 \tag{3.35}$$

$$\tilde{x}_{n,d} = \frac{x_{n,d} - \mu_n^L}{\sqrt{\left(\sigma_n^L\right)^2 + \epsilon}} \tag{3.36}$$

$$y_{n,d} = \gamma_d \tilde{x}_{n,d} + \beta_d \tag{3.37}$$

で与えられる．

上述したバッチ正規化およびレイヤー正規化は，特徴量を標準化することで学習を安定させる手法であった．一方，DNN 内のある層の重みを再パラメータ化することで学習を安定させる手法として，重み正規化（Weight Normalization）[?] がある．これは，ある層の重みベクトル $\boldsymbol{w}$ を，

$$\boldsymbol{w} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|} g \tag{3.38}$$

のように単位ベクトル $\boldsymbol{v}/\|\boldsymbol{v}\|$（ベクトルの向き）とスカラー $g$（ベクトルの大きさ）に再パラメータ化するものである．学習時は重みの更新を $\boldsymbol{v}$ と $g$ で別々に行う．重み正規化は，バッチ正規化やレイヤー正規化と同様に学習の安定化に役立つが，計算に入力特徴量の系列長が依存しない．そのため，例えば音声波形など系列長が非常に長くなりがちなデータを扱う場合，計算コストを下げながら同様の効果を狙える手段だと考えられる．

### 3.1.7 Transformer

Transformer[?] は，自己注意機構（Self-Attention）を用いて，入力系列全体に渡る依存関係を捉えることができるニューラルネットワークである．特に，再帰的な計算を必要とする RNN と比較して，Transformer は並列計算のみ行うため，GPU による計算の高速化が可能である．以下，入力特徴量を $\boldsymbol{X} \in \mathbb{R}^{T \times D_{\text{model}}}$ として，Transformer において行われる計算を説明する．また，Transformer 層の構造を図 **??**に示す．

まず，Transformer における Self-Attention の計算の流れを述べる．ここでは，はじめにクエリ $\boldsymbol{Q} \in \mathbb{R}^{T \times D_k}$，キー $\boldsymbol{K} \in \mathbb{R}^{T \times D_k}$，バリュー $\boldsymbol{V} \in \mathbb{R}^{T \times D_v}$ の計算を行う．これは，

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q \oplus \boldsymbol{b}_Q \tag{3.39}$$

$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K \oplus \boldsymbol{b}_K \tag{3.40}$$

$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V \oplus \boldsymbol{b}_V \tag{3.41}$$

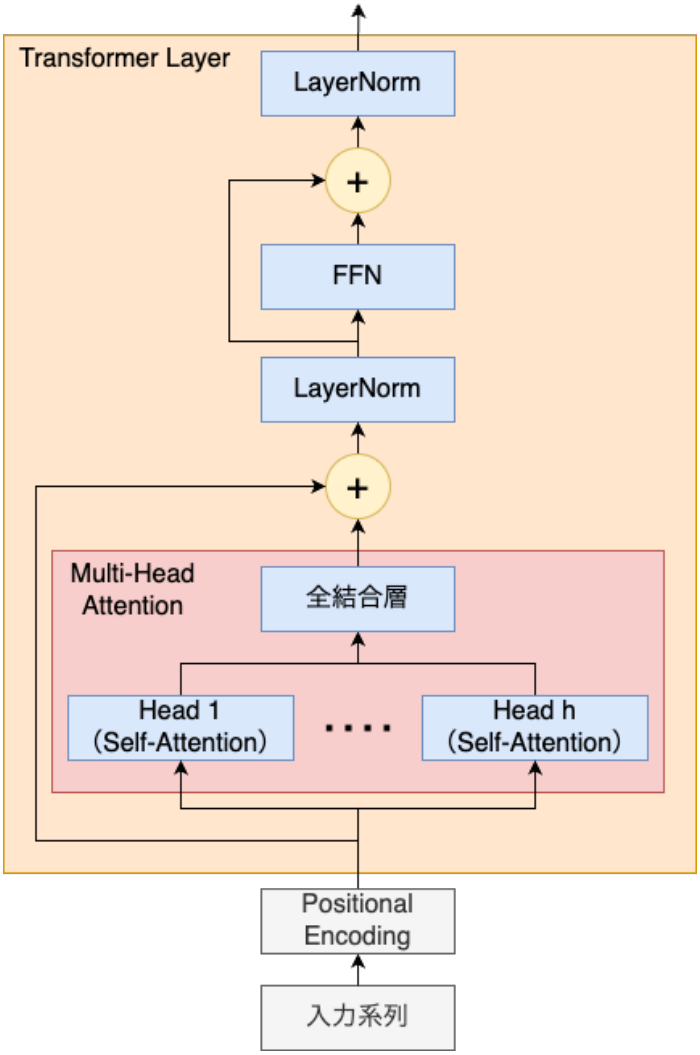



図 3.4: Transformer 層の構造

で与えられる．ここで，$\boldsymbol{W}_Q, \boldsymbol{W}_K \in \mathbb{R}^{D_{\text{model}} \times D_k}, \boldsymbol{W}_V \in \mathbb{R}^{D_{\text{model}} \times D_v}$ は重み，$\boldsymbol{b}_Q, \boldsymbol{b}_K \in \mathbb{R}^{D_k}, \boldsymbol{b}_V \in \mathbb{R}^{D_v}$ はバイアスである．また，演算子 $\oplus$ について，$\boldsymbol{A} \in \mathbb{R}^{D_1 \times D_2}$，$\boldsymbol{b} \in \mathbb{R}^{D_2}$ に対する $\boldsymbol{A} \oplus \boldsymbol{b} \in \mathbb{R}^{D_1 \times D_2}$ の $(i, j)$ 成分 $(\boldsymbol{A} \oplus \boldsymbol{b})_{i,j}$ は，

$$(\boldsymbol{A} \oplus \boldsymbol{b})_{i,j} = a_{i,j} + b_j \tag{3.42}$$

で与えられる．ここで，$a_{i,j}$ は $\boldsymbol{A}$ の $(i, j)$ 成分，$b_j$ は $\boldsymbol{b}$ の $j$ 成分である．

次に，クエリとキーを元にアテンション重みを求め，バリューに対する行列積を計算する．これは，

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{D_k}}\right)\boldsymbol{V} \tag{3.43}$$

で与えられる．softmax 関数は行方向に適用されるため，$\text{softmax}\left(\boldsymbol{Q}\boldsymbol{K}^\top/\sqrt{D_k}\right)$ の各行ベクトルが，各クエリ $\boldsymbol{q}_t \in \mathbb{R}^{D_k}$ からキー $\boldsymbol{k}_t \in \mathbb{R}^{D_k}$，$(t = 1, \ldots, T)$ に対する注意度になっている．

ここまでが Self-Attention の計算であったが，Transformer では Self-Attention を複数のヘッドで並列に計算し，各ヘッドの出力を結合して最終出力を得る Multi-Head Attention が採用され

ている．ヘッド数を $H$ とすると，各ヘッド $h$ におけるクエリ $\boldsymbol{Q}^h \in \mathbb{R}^{T\times\frac{D_k}{H}}$，キー $\boldsymbol{K}^h \in \mathbb{R}^{T\times\frac{D_k}{H}}$，バリュー $\boldsymbol{V}^h \in \mathbb{R}^{T\times\frac{D_v}{H}}$ の計算は，

$$\boldsymbol{Q}^h = \boldsymbol{X}\boldsymbol{W}_Q^h \oplus \boldsymbol{b}_Q^h \tag{3.44}$$

$$\boldsymbol{K}^h = \boldsymbol{X}\boldsymbol{W}_K^h \oplus \boldsymbol{b}_K^h \tag{3.45}$$

$$\boldsymbol{V}^h = \boldsymbol{X}\boldsymbol{W}_V^h \oplus \boldsymbol{b}_V^h \tag{3.46}$$

で与えられる．ここで，$\boldsymbol{W}_Q^h, \boldsymbol{W}_K^h \in \mathbb{R}^{D_{\text{model}}\times\frac{D_k}{H}}$，$\boldsymbol{W}_V^h \in \mathbb{R}^{D_{\text{model}}\times\frac{D_v}{H}}$ は重み，$\boldsymbol{b}_Q^h, \boldsymbol{b}_K^h \in \mathbb{R}^{\frac{D_k}{H}}, \boldsymbol{b}_V^h \in \mathbb{R}^{\frac{D_v}{H}}$ はバイアスである．この後の処理も Self-Attention と同様に，

$$\text{Attention}\left(\boldsymbol{Q}^h, \boldsymbol{K}^h, \boldsymbol{V}^h\right) = \text{softmax}\left(\frac{\boldsymbol{Q}^h(\boldsymbol{K}^h)^\top}{\sqrt{D_k}/H}\right)\boldsymbol{V}^h \tag{3.47}$$

となる．ここで得られた各ヘッドからの出力を結合し，さらに全結合層を通すことで Multi-Head Attetion の出力が得られる．すなわち，

$$\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \left[\text{Attention}\left(\boldsymbol{Q}^h, \boldsymbol{K}^h, \boldsymbol{V}^h\right), \ldots, \text{Attention}\left(\boldsymbol{Q}^h, \boldsymbol{K}^h, \boldsymbol{V}^h\right)\right]\boldsymbol{W}_o \oplus \boldsymbol{b}_o \tag{3.48}$$

と表される．ここで，$[\cdot, \ldots, \cdot]$ は次元方向の結合を表し，$\boldsymbol{W}_o \in \mathbb{R}^{D_v\times D_{\text{model}}}$ は重み，$\boldsymbol{b}_o \in \mathbb{R}^{D_{\text{model}}}$ はバイアスである．ヘッドを分割して複数パターンの Self-Attention を可能にすることで，より複雑な入出力の関係を学習できると考えられる．Multi-Head Attention 後は，残差結合とレイヤー正規化を適用する．すなわち，この出力 $\boldsymbol{Y} \in \mathbb{R}^{T\times D_{\text{model}}}$ は，

$$\boldsymbol{Y} = \text{LayerNorm}\left(\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) + \boldsymbol{X}\right) \tag{3.49}$$

で与えられる．その後，全結合層を通し，残差結合とレイヤー正規化を適用することで Transformer 層最終出力を得る．すなわち，

$$\boldsymbol{Y} = \text{LayerNorm}((\text{ReLU}(\boldsymbol{Y}\boldsymbol{W}_1 \oplus \boldsymbol{b}_1)\boldsymbol{W}_2 \oplus \boldsymbol{b}_2) + \boldsymbol{Y}) \tag{3.50}$$

となる．ここで，$\boldsymbol{W}_1 \in \mathbb{R}^{D_{\text{model}}\times D_{\text{fc}}}$，$\boldsymbol{W}_2 \in \mathbb{R}^{D_{\text{fc}}\times D_{\text{model}}}$ は重み，$\boldsymbol{b}_1 \in \mathbb{R}^{D_{\text{fc}}}$，$\boldsymbol{b}_2 \in \mathbb{R}^{D_{\text{model}}}$ はバイアスである．$D_{\text{fc}}$ は $D_{\text{model}}$ の 4 倍とされることが多い．Transformer 全体は，Transformer 層を多層積み重ねて構成される．

最後に，Transformer では RNN と違い，並列計算によって系列全体を一度に処理することが可能であるが，それと引き換えに入力の順序情報を考慮することができなくなる．これに対し，Transformer では Positional Encoding によって入力に位置情報を与える．Positional Encoding は sin と cos に基づいて，

$$\text{PositionalEncoding(t, d)} = \begin{cases} \sin\left(\frac{t}{10000^{2d/D_{\text{model}}}}\right) & \text{if } d \bmod 2 = 0 \\ \cos\left(\frac{t}{10000^{2d/D_{\text{model}}}}\right) & \text{if } d \bmod 2 = 1 \end{cases} \tag{3.51}$$

で与えられる．

## 3.2 学習方法

本節において，$\boldsymbol{\theta} \in \mathbb{R}^{N_{\text{model}}}$ はDNNの重みとバイアスをまとめて表す変数とする．また，文章中では簡潔さを優先し，特別な理由がない限りは重みと呼ぶ.

### 3.2.1 損失関数

損失関数は，DNNによって推定された結果と正解値との間の誤差を求める関数のことであり，扱う問題によって様々である．例えば，回帰問題において用いられる関数の一つに，MAE（Mean Absolute Error）Lossがある．推定対象を $\boldsymbol{y} \in \mathbb{R}^D$，DNNによる推定結果を $\hat{\boldsymbol{y}} \in \mathbb{R}^D$ とすると，MAE Lossは

$$L_{MAE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{D} \sum_{d=1}^{D} |y_d - \hat{y}_d| \tag{3.52}$$

で与えられる.

一方，分類問題において用いられる関数の一つに，Cross Entropy Lossがある．$C$ クラス分類の問題について，推定対象を $\boldsymbol{y} \in [0,1]^C$，DNNによる推定結果を $\hat{\boldsymbol{y}} \in [0,1]^C$ とすると，Cross Entropy Lossは

$$L_{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{c=1}^{C} y_c \log(\hat{y}_c) \tag{3.53}$$

で与えられる．ここで，$\boldsymbol{y}, \hat{\boldsymbol{y}}$ はどちらもクラスに対する確率分布であり，

$$\sum_{c=1}^{C} y_c = 1, \quad \sum_{c=1}^{C} \hat{y}_c = 1 \tag{3.54}$$

を満たす．推定対象 $\boldsymbol{y}$ は，正解となるクラスのみを1，それ以外を0としたOne-hotベクトルとされることが多い.

### 3.2.2 勾配降下法

勾配降下法（Gradient Descent）は，損失関数の重みについての勾配を利用して，損失関数の値を最小化するようにDNNを最適化するアルゴリズムである．ここで，学習データセットを $\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^N$ とする．各 $n$ に対し，$\boldsymbol{x}_n \in \mathbb{R}^{D_{\text{in}}}$ はDNNへの入力，$\boldsymbol{y}_n \in \mathbb{R}^{D_{\text{out}}}$ は予測対象を表す．DNNは $f : \mathbb{R}^{D_{\text{in}}} \to \mathbb{R}^{D_{\text{out}}}$ で表し，予測値を $\hat{\boldsymbol{y}}_n = f(\boldsymbol{x}_n; \boldsymbol{\theta})$ とする．損失関数は $L : \mathbb{R}^{D_{\text{out}}} \times \mathbb{R}^{D_{\text{out}}} \to \mathbb{R}$ とし，$\mathcal{L} : \mathbb{R}^{N_{\text{model}}} \to \mathbb{R}$ を，

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{I}) = \sum_{i \in \mathcal{I}} L(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i) = \sum_{i \in \mathcal{I}} L(\boldsymbol{y}_i, f(\boldsymbol{x}_n; \boldsymbol{\theta})) \tag{3.55}$$

で定義する．ここで，$\mathcal{I} \subset \{1, \ldots, N\}$ は各サンプルに対するインデックスの部分集合を表す．すなわち，$\mathcal{L}$ は学習データ $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i \in I}$ に対する損失を，DNNの重み $\boldsymbol{\theta}$ の関数として扱うもの

である．上の表記を用いれば，DNN の最適化問題は

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{N_{\mathrm{model}}}} \mathcal{L}(\boldsymbol{\theta};\{1,\dots,N\}) \tag{3.56}$$

と表される．この最適化問題に対し，勾配降下法による重み $\boldsymbol{\theta}$ の更新は，

$$\boldsymbol{\theta}_\tau = \boldsymbol{\theta}_{\tau-1} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_{\tau-1};\mathcal{I}_{\tau-1}) \tag{3.57}$$

で与えられる．ここで，$\tau\in\mathbb{N}$ は学習におけるイテレーション，$\eta\in[0,\infty)$ は学習率を表す．

勾配降下法には，三種類の方法がある [?]．一つ目は，バッチ勾配降下法である．これは，

$$\boldsymbol{\theta}_\tau = \boldsymbol{\theta}_{\tau-1} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_{\tau-1};\{1,\dots,N\}) \tag{3.58}$$

で与えられる．すなわち，各イテレーションで学習データ全てを用いる方法である．各サンプルのノイズの影響が低減されることで安定した学習が期待できるが，計算コストが高い．二つ目は，確率的勾配降下法である．これは，ランダムに選択された $n_\tau\in\{1,\dots,N\}$ に対し，

$$\boldsymbol{\theta}_\tau = \boldsymbol{\theta}_{\tau-1} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_{\tau-1};\{n_\tau\}) \tag{3.59}$$

で与えられる．すなわち，各イテレーションで単一サンプルのみを用いる方法である．計算コストが下がるが，各サンプルのノイズの影響が大きくなることで学習が不安定になる可能性がある．三つ目は，ミニバッチ勾配降下法である．これは，$1<|\mathcal{I}_\tau|<N$ を満たすランダムに選択された $\mathcal{I}_\tau\subsetneq\{1,\dots,N\}$ に対し，

$$\boldsymbol{\theta}_\tau = \boldsymbol{\theta}_{\tau-1} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_{\tau-1};\mathcal{I}_\tau) \tag{3.60}$$

で与えられる．ここで，$|\cdot|$ は集合の元の数を表す．これより，各イテレーションで二つ以上のサンプルからなる学習データセットの真部分集合を用いる方法だと言える．バッチ勾配降下法と確率的勾配降下法の間をとった方法であり，DNN の学習においては一般にミニバッチ勾配降下法が用いられる．ここで，ミニバッチに含まれるサンプルの数をバッチサイズと呼ぶ．

また，確率的勾配降下法やミニバッチ勾配降下法では，サンプルを学習データセットからランダムに非復元抽出する．ここで，毎回のサンプリングされた学習データに対する処理は 1 イテレーションとカウントし，学習データセットを一度全て使い切ることは 1 エポックとカウントする．実際には，データセットの総サンプル数 $N$ に対してバッチサイズを決定することで 1 エポックあたりの総イテレーション数は決まり，最大エポック数を設定して学習を回すこととなる．

### 3.2.3 正則化

DNN は大量のパラメータにより高い表現力を持つが，その分学習データに過剰に適合し，未知データに対する汎化性能が低いモデルとなる，過学習を引き起こす可能性がある．正則化は，このような DNN の過学習を防ぐための手段である．以下，具体的な方法を四つ述べる．