深層学習

知能情報工学科
232C1032 小田敢太

August 13, 2025

目次

まえがき

本レポートは,「深層学習」の講義で学んだ内容について自分なりに解釈し，また必要に応じてインターネット上の無数の資料を参考にしてまとめたものである．私の理解力不足のため，特に Transformer や生成モデルの内容は講義資料を振り返るだけでは自分の頭の中で整理して説明することができなかった．そのため，インターネットで調べて改めて勉強し，文章や図表にまとめたのでレポート提出が締め切り直前になってしまった．レポート課題として与えられた問題に対して**ただ答えるのではなく，授業で学習したことを整理して自らの理解を深めることを重視した**．また，**レポートの指示に従い，深層学習について知らない人に対して教える形を想定してまとめた**．そのため，ご覧の通り単なるレポートというより教科書を作ったというほうが正しいかもしれない．さらに，最近 Microsoft Word に飽きてきたので今回は LaTeX を使うことにした．個人的に LaTeX には慣れていないのでレポートの体裁については AI の力を借りたが，レポート中に現れる**あらゆる図はすべて自作**にこだわった．

念のため，レポートとして示された課題と本レポートとの対応関係を以下に示す．

	課題		本レポート
前半	問題 1	機械学習の原理	第 1 章
前半	問題 2	誤差逆伝播法	第 2 章
前半	問題 3	活性化関数	第 3 章，第 4 章
前半	問題 4	CNN	第 5 章
後半	問題 5	RNN	第 6 章
後半	問題 6	Transformer，BERT	第 7 章
後半	問題 7	正則化	第 8 章
後半	問題 8	生成モデル	第 9 章

1　機械学習の基本原理と損失関数

1.1　機械学習の基礎

機械学習の定義と分類

機械学習 (Machine Learning) は，データから規則性や知識を自動的に獲得し，それを用いて将来の予測や意思決定を行う技術である．伝統的には以下の三つに分類される．

- **教師あり学習 (Supervised Learning)**: 入力と正解ラベルが与えられたデータをもとに，入力から正解を推定する関数を学習する．
- **教師なし学習 (Unsupervised Learning)**: 正解ラベルのないデータから，データの構造や潜在的な分布を学習する．
- **強化学習 (Reinforcement Learning)**: エージェントが環境との相互作用を通じて報酬を最大化する行動方策を学習する．

このレポートの前半では，主に**教師あり学習**を扱う．

教師あり学習の形式化

教師あり学習では，入力空間 $\mathcal{X}$ と出力空間 $\mathcal{Y}$ を考える．訓練データは

$$\mathcal{D} = \{(x_i, y_i) \mid i = 1, \ldots, n\},$$

で表される．ここで $x_i \in \mathcal{X}$ は**特徴ベクトル**，$y_i \in \mathcal{Y}$ は**ラベル**である．目的は入力から出力を予測する関数

$$f : \mathcal{X} \to \mathcal{Y}$$

を見つけることである．この関数はパラメータ θ を含むことが多く，

$$f(x; \theta)$$

と表される．

頻度論的アプローチの概要

頻度論的立場では，データは未知の分布 $P(x, y)$ から独立に抽出されたと仮定する．したがって，ある損失関数 $\ell(y, f(x))$ に基づく**期待リスク**を最小化することが目標となる．

$$R(f) = \mathbb{E}_{(x,y)\sim P}[\, \ell(y, f(x))\,].$$

しかし，この分布 P は未知であるため，訓練データを使って**経験リスク**を近似的に計算する．

$$\hat{R}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i)).$$

学習とは，この経験リスクを最小化するパラメータ θ を探索する問題に帰着する．

Point　線形分類器の学習

2 次元平面上のデータ

$$x_i = (x_{i1}, x_{i2}) \in \mathbb{R}^2, \quad y_i \in \{-1, +1\}$$

に対し，線形分類器

$$f(x) = \mathrm{sign}(w^\top x + b)$$

を学習する場合，二乗損失を用いた経験リスクは次の式で表される．

$$\hat{R}(f) = \frac{1}{n} \sum_{i=1}^{n} \big(y_i - (w^\top x_i + b)\big)^2.$$

このリスクを最小化する w, b を**勾配降下法**などで求める．

まとめ

機械学習は，データから予測関数を構築する手法であり，教師あり・教師なし・強化学習に分類される．教師あり学習では，未知の分布に基づく期待リスクを損失関数で評価し，経験リスクを最小化してモデルを学習する．

1.2　リスク最小化の原理

経験リスクと期待リスク

機械学習では，学習した関数がどれだけ正確に予測できるかを定量的に評価する必要がある．その基準となるのが**リスク**である．理想的には，未知の分布 $P(x, y)$ において損失を平均した**期待リスク**を最小化することが目標である．

$$R(f) = \mathbb{E}_{(x,y)\sim P}[\, \ell(y, f(x)) \,].$$

ただし，分布 P は実際には不明なため，手元の訓練データ $\mathcal{D}$ を用いて代わりに**経験リスク**を計算する．

$$\hat{R}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i)).$$

学習とは，この経験リスクを最小化するようにパラメータを調整することである．

バイアス・バリアンスのトレードオフ

モデルの性能は単純に経験リスクの大きさだけで決まらない．複雑なモデルは訓練データにうまく適合するが，未知のデータに対する**汎化性能**が低下する場合がある．これは**バイアス・バリアンスのトレードオフ**として知られる（**図 1.1** 参照）．

- **バイアス** (bias): 予測値と真の値との誤差．モデルの表現力が不十分なために生じる．
- **バリアンス** (variance): 予測値の広がり具合．モデルが訓練データのノイズ（揺らぎ）に過度に適合（過学習）することで生じる誤差．

最適なモデルは，この二つの誤差をバランスさせる必要がある．

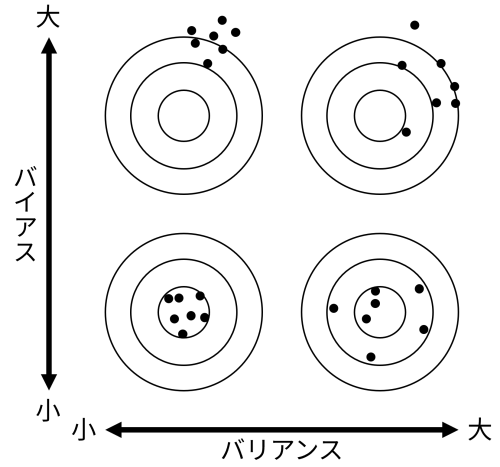

図 1.1: バイアスとバリアンスのトレードオフ

まとめ

学習では，期待リスクを最小化することが理想だが，未知の分布は得られないため，経験リスクを代用する．モデルの複雑さはバイアス・バリアンスのトレードオフを生み，汎化性能を考慮した選択が重要である．

1.3 損失関数の種類と性質

損失関数の役割

損失関数は，モデルの予測と正解のずれを数値化する尺度であり，学習の目的関数として機能する．損失関数の選択は，分類や回帰といった学習するタスクの特性に深く依存する．

分類タスクの損失関数

出力が離散的なクラスである分類問題では，以下のような損失関数が用いられる．$y \in \{-1, +1\}$ は真のラベル，$\hat{y}$ は予測スコア．$\mathbb{I}\{y \neq \hat{y}\}$ は，$y \neq \hat{y}$ であれば 1，そうでなければ 0 を返す．

- **0-1 損失**：

  $$\ell(y, \hat{y}) = \mathbb{I}\{y \neq \hat{y}\}$$

  正解と異なる予測に対して 1 を与える最も単純な損失．ただし非連続で最適化が難しい．

- **ヒンジ損失**：

  $$\ell(y, f(x)) = \max(0, 1 - yf(x))$$

  マージン最大化に基づき，**SVM(サポートベクターマシン)** で用いられる．

- **交差エントロピー損失**：

  $$\ell(y, \hat{p}) = -\sum_{k} \mathbb{I}\{y = k\} \log \hat{p}_k$$

  クラス確率の対数を用いる．ロジスティック回帰やニューラルネットワークで広く使われる．

- **指数損失関数**：

  $$\ell(y, \hat{y}) = \exp(-y\,\hat{y})$$

  分類誤差を指数的に強調する特徴をもつ．分類が正しく $(y\,\hat{y} > 0)$ スコアが大きいほど損失は指数的に小さくなる．分類が誤っている $(y\,\hat{y} < 0)$ 場合，損失は急激に増大する．特にアンサンブル学習の **Adaboost** において用いられる．

回帰タスクの損失関数

出力が連続値の回帰問題では，誤差を直接評価する損失関数が選ばれる．

- **二乗損失（二乗誤差）**：

$$\ell(y, \hat{y}) = (y - \hat{y})^2$$

最も一般的な損失関数．平均二乗誤差 (MSE) の最小化は最尤推定と一致する．

- **絶対値損失**：

$$\ell(y, \hat{y}) = |y - \hat{y}|$$

外れ値に対する頑健性が高い．

- **フーバー損失**：

$$\ell(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| \le \delta \\ \delta\left(|y - \hat{y}| - \frac{\delta}{2}\right) & \text{otherwise} \end{cases}$$

小さな誤差では二乗損失，大きな誤差では線形損失に切り替わる．

Point　損失関数と学習の安定性

損失関数の選択は，学習アルゴリズムの性質や性能に大きな影響を与える．たとえば，二乗損失は解析的に扱いやすいが外れ値に弱い．ヒンジ損失や交差エントロピー損失は分類性能を高めるが，モデルの出力に応じたスケーリングが必要になる．問題の性質に応じて損失関数を適切に選ぶことが重要である．

損失関数は単なる評価基準ではなく，学習の方向を決める重要な役割を持つ．次節では，分類や回帰モデルの具体的な構造について述べる．

まとめ

損失関数は，モデルがどれだけ正しく予測できているかを数値化する重要な指標である．0-1 損失やヒンジ損失は分類に，二乗損失やフーバー損失は回帰に適用される．目的に応じた適切な損失関数の選択が学習性能に直結する．

1.4　基本的な分類・回帰モデル

パーセプトロン

パーセプトロンは，線形分離可能なデータを分類するための古典的な分類モデルである．二値分類のタスクでは，入力ベクトル x に対して

$$f(x) = \mathrm{sign}(w^\top x + b)$$

で表される線形識別関数を用いる．ここで w は**重みベクトル**，b は**バイアス**である．学習は誤分類サンプルに基づいて w, b を更新することで行う．誤分類のたびに次の更新を行う．

$$w \leftarrow w + \eta y_i x_i, \quad b \leftarrow b + \eta y_i,$$

ここで y_i は**正解ラベル**，η は**学習率**である．

ロジスティック回帰

ロジスティック回帰は，確率的な出力を持つ線形の分類モデルである．入力 x に対するクラス 1 の事後確率は**シグモイド関数**でモデル化される（ただし，$z_i = w^\top x_i + b$）．

$$P(y = 1 \mid x) = \sigma(z_i) = \frac{1}{1 + \exp(z_i)}.$$

学習は**交差エントロピー損失**の最小化によって行われる．

$$L(w, b) = -\sum_{i=1}^{n} \big[y_i \log \sigma(z_i) + (1 - y_i) \log(1 - \sigma(z_i)) \big],$$

サポートベクターマシン (SVM)

SVM は**マージン最大化**の原理に基づく線形分類器である（**図 1.2** 参照）．データを完全に二分できることを前提とするハードマージン SVM では，以下を満たす w, b を求める．

$$y_i(w^\top x_i + b) \geq 1, \quad \forall i.$$

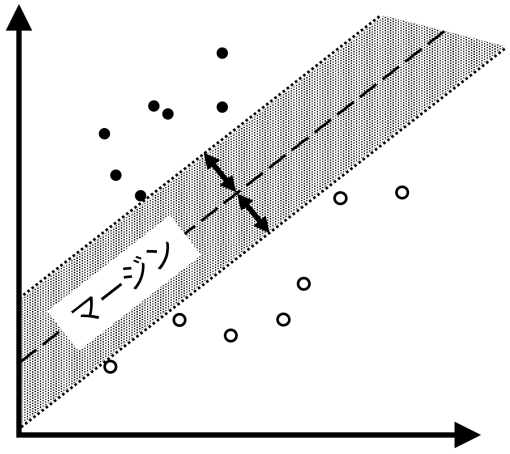

図 1.2: サポートベクターマシンの概念図

線形回帰

数値の連続値を予測する回帰タスクで最も基本的な方法が**線形回帰**である．線形回帰では，入力ベクトル x に対して以下の形の予測を行う．

$$f(x) = w^\top x + b$$

目的は，二乗損失を最小化するパラメータ (w, b) を求めることである．

$$\hat{R}(f) = \frac{1}{n} \sum_{i=1}^{n} \big(y_i - (w^\top x_i + b) \big)^2.$$

過学習防止のための改良版として，**L1 正則化**を用いた**ラッソ回帰**，**L2 正則化**を用いた**リッジ回帰**，両方を用いた **Elastic Net** など，損失関数にペナルティ項を加えて過剰なパラメータの増大を防ぐ手法がある（**第 8 章**参照）．

まとめ

パーセプトロンは単純な誤分類修正で学習するが，ロジスティック回帰は確率的なモデルであり，SVM はマージン最大化を目的とする．いずれも線形関数を基本とする分類モデルであるが学習基準が異なる．また，回帰モデルには線形回帰やそれを改良したラッソ回帰およびリッジ回帰がある．

2　最適化アルゴリズムと誤差逆伝播

2.1　最適化問題の定式化

最適化問題の概要

機械学習では，パラメータを更新することで損失関数を最小化することが目的となる．この問題は次のように**最適化問題**として定式化される．

$$\min_{\theta} \; L(\theta)$$

ここで θ はモデルのパラメータ，$L(\theta)$ は損失関数である．最適化の目標は，$L(\theta)$ が最小となる θ^* を見つけることである．

多くの機械学習アルゴリズムでは，勾配情報を利用してこの目的を達成する．最適化問題の性質（凸か非凸か）に応じて解の性質や困難さが異なる．

凸最適化と非凸最適化

最適化問題は，損失関数の形状によって大きく 2 つに分かれる．

- **凸最適化**：

$$\forall \theta_1, \theta_2, \quad L\big(\lambda\theta_1 + (1-\lambda)\theta_2\big) \leq \lambda L(\theta_1) + (1-\lambda)L(\theta_2), \quad \lambda \in [0,1]$$

  この条件を満たすとき，損失関数は**凸関数**であり，任意の局所解が大域解である．

- **非凸最適化**：ニューラルネットワークなどで用いられる損失関数は一般に非凸であり，複数の局所解や鞍点を持つ．このため，最適化の難易度が上がる．

非凸最適化は理論的な最適性が保証されない場合が多いが，大規模データにおける実践的手法の多くは経験的に高い性能を示す．

パラメータ更新の基本

最適化は，損失関数の**勾配**を用いてパラメータを更新する逐次的な手法が多く利用される．最も基本的な枠組みは以下の更新式で表される．

$$\theta \leftarrow \theta - \eta \, \nabla_\theta L(\theta),$$

ここで η は**学習率** (learning rate) である．学習率が大きすぎると最適点を飛び越えて発散し，小さすぎると収束が非常に遅くなる．実務では一定値を使うよりも，エポックの進行に応じて減衰させるスケジューリングがしばしば用いられる．

まとめ

最適化問題は，損失関数の最小化として定式化される．凸最適化では大域最適解が保証されるが，深層学習では多くが非凸最適化であり，複数の局所解や鞍点を含む．最適化では学習率をはじめとする設定が重要な役割を果たす．

2.2　勾配降下法とその改良

勾配降下法の基本

最も基本的な最適化手法は**勾配降下法** (Gradient Descent) である．これは，損失関数の勾配を用いてパラメータを更新するシンプルな手法であり，データの扱い方により次の 3 つのバリエーションがある．

- **バッチ勾配降下法**：全訓練データを使って勾配を計算する．安定するが計算コストが高い．
- **確率的勾配降下法 (SGD)**：データの順番を入れ替えながらランダムに勾配降下法を適用し，1 サンプルごとに勾配を計算する．このようにすることで，局所最適解に収束する確率を下げる．一方，学習率の設定が難しく収束が遅い．

適応的学習率の手法

近年は，パラメータごとに学習率を自動的に調整する手法が広く使われている．

- **Momentm SGD**：SGD には，最適化の進行がジグザグになりやすいという欠点がある（**図 2.1 参照**）．これを緩和するために**モメンタム** (Momentum) を導入し，過去の勾配を累積する**移動平均**を用いる．

$$v_t = \alpha v_{t-1} - \eta \nabla_\theta L(\theta)$$
$$\theta \leftarrow \theta + v_t$$

  ここで α はモメンタム係数 (一般に 0.9 程度) で，慣性を持たせる役割を果たす．

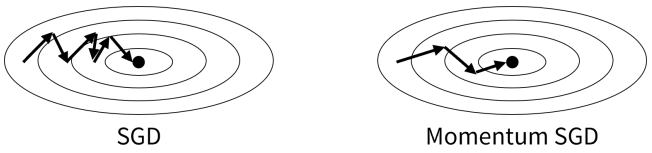

図 2.1: Momentum SGD の概念図

- **AdaGrad**：小刻みに探索したいパラメータは学習率を小さく，大雑把に探索したいパラメータは学習率を大きくすることで効率的に探索する．

$$\theta \leftarrow \theta - \frac{\eta}{\sqrt{G_t + \epsilon}} \circ \nabla_\theta L(\theta),$$

  G_t はこれまでの勾配の二乗和であり，頻繁に更新されるパラメータほど学習率が減衰する．

- **RMSProp**：AdaGrad の学習率減衰が早すぎる問題を緩和するために，勾配の二乗和を指数移動平均で計算してより最近の勾配を重視する．

- **Adam**：Momentum SGD と RMSProp を組み合わせた手法であり，最も汎用性が高いとされる．モメンタムによる慣性と，適応的な学習率調整の双方を兼ね備える．

パラメータが多い場合，単純な勾配降下法では収束が遅くなることや，学習が進むにつれて学習率を適切に減衰させないと最適化が不安定になる問題が生じやすい．さらに，学習率以外にも初期化の方法やバッチサイズの選択，正則化手法などが学習の安定性に影響を及ぼすため，最適化アルゴリズムとハイパーパラメータは総合的に設計する必要がある．

まとめ

最適化アルゴリズムは，SGD やそれを改良した Momentum SGD，AdaGrad，RMSProp，Adam など多様であり，それぞれ収束速度や安定性に特徴がある．モデルやデータに適した手法を選択することが，効率的な学習に不可欠である．

2.3　誤差逆伝播

勾配降下法と誤差逆伝播の関係

ニューラルネットワークの学習では，パラメータ（重みとバイアス）を損失関数が最小となるように更新する必要がある．これを実現する代表的な最適化手法が**勾配降下法**である．

具体的には，現在のパラメータ θ に対し，損失関数 $L(\theta)$ の勾配 $\nabla L(\theta)$ を計算し，以下のように更新する．

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$$

ここで $\eta\ (>0)$ は更新の大きさを調節するハイパーパラメータであり，**学習率**とよばれる．これを適切に設定しないと収束が遅くなったり，振動や発散が起こる．

連鎖律による勾配計算

多層ニューラルネットワークでは，出力層から順に損失関数の勾配を計算し，ネットワークの全パラメータを効率的に更新する必要がある．この仕組みが**誤差逆伝播法** (Backpropagation) である．

誤差逆伝播は，出力層で計算された損失の勾配を**連鎖律** (chain rule) を用いて入力層まで伝播させることで，各パラメータに対する偏微分を効率的に計算する．たとえば，損失 L が出力 y に依存し，y が中間変数 z に依存する場合，z に対する勾配は以下で表される．

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z}$$

出力層で計算した $\frac{\partial L}{\partial y}$ を入力層まで逆向きに再帰的に伝播することで，効率的に全ての勾配を計算できる．ネットワークが多層の場合でも同様に，この関係を繰り返してすべてのパラメータに対する勾配を計算できる（**図 2.2** 参照）．

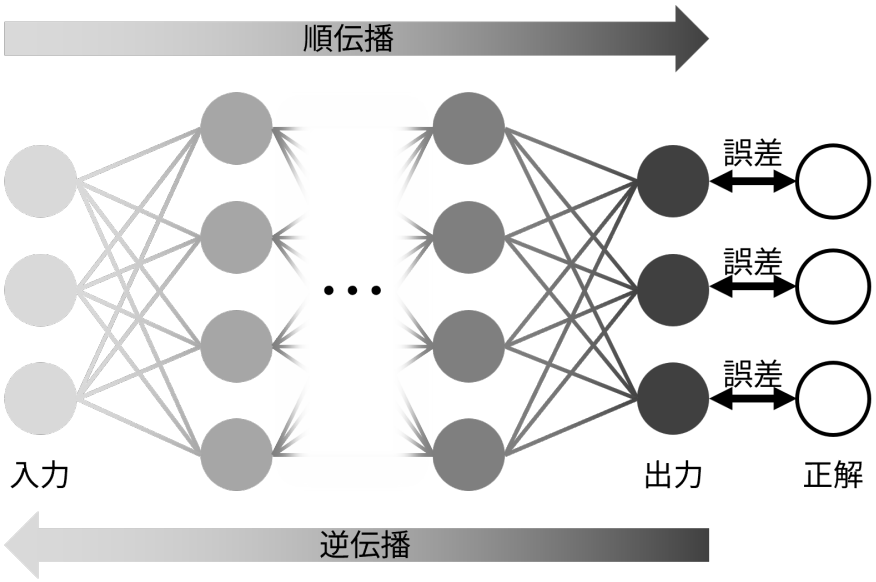

図 2.2: 誤差逆伝播の概念図

まとめ

誤差逆伝播は勾配降下法と組み合わせて用いることで，損失を減らすようにネットワークのパラメータを更新できる．連鎖律に基づき，出力層から入力層へ効率的に誤差を伝搬する仕組みが特徴である．

2.4　最適化における課題と工夫

局所解と鞍点

深層ニューラルネットワークは非凸最適化問題であり，損失関数の形は非常に複雑である．このため，以下のような性質が存在する．

- **局所解**: 真の最適解より損失が高いが，それ以上改善できない点（**図 2.3** 参照）．
- **鞍点**: 勾配はゼロではあるが，周囲に損失の低い方向が存在する点（**図 2.4** 参照）．

高次元空間では，鞍点が多く出現するため，学習の停滞を引き起こす場合がある．

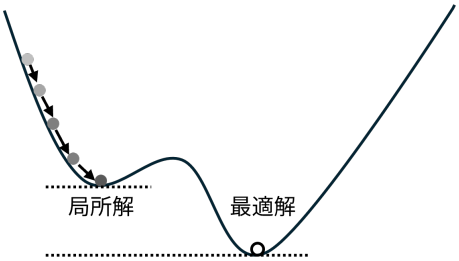

図 2.3: 局所最適解と大域最適解

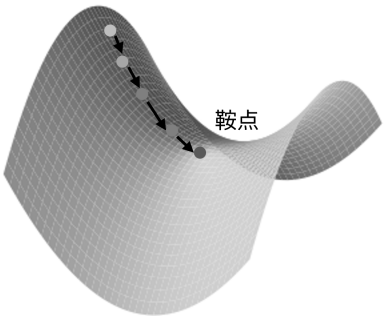

図 2.4: 鞍点での学習の停滞

勾配消失と勾配爆発

深いネットワークでは，出力から入力への勾配が層を遡るにつれて極端に小さくなる（**勾配消失**）か，逆に大きくなりすぎる（**勾配爆発**）問題が発生する．特にシグモイド関数を活性化関数に用いた場合には，勾配消失が問題となる．これを緩和するため，以下の工夫がある．

- 活性化関数に ReLU など非飽和関数を用いる（飽和：入力が大きな値に近づいたときに出力や勾配が一定値に近づいてほとんど変わらなくなること）．
- 重み初期化を工夫する．
- **バッチ正規化**を行い分布を安定させる．

重み初期化

学習の安定性と収束の速さは，重みの初期化に強く依存する．一般的な方法としては以下がある．

- **Xavier 初期化**: 入出力の次元に基づいて分散を調整する．
- **He 初期化**: ReLU 活性化関数に適した分散を与える．

バッチ正規化

バッチ正規化 (Batch Normalization) は，各層の出力を標準化する手法であり，勾配消失・爆発の緩和，学習の加速，汎化性能の向上に寄与する．各ミニバッチで平均 μ，分散 σ^2 を計算し，正規化後に学習可能なスケーリングとシフトを適用する（γ, β は学習可能なパラメータ）．

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad y = \gamma \hat{x} + \beta$$

まとめ

深層学習の最適化では，勾配消失や鞍点，初期化の不適切さが学習を妨げる要因となる．バッチ正規化や重み初期化，適切な活性化関数を組み合わせることで，安定かつ効率的な収束が可能になる．

3　活性化関数とその役割

3.1　活性化関数の意義

ニューラルネットワークは，複数の線形変換と活性化関数を組み合わせることで非線形かつ複雑な関数を表現できる．仮に活性化関数を用いず，すべての層を線形変換のみで構成すると，いくら層を積み重ねても最終的な出力は単なる線形変換に留まりネットワーク全体の表現力が制限されてしまう．この問題を解決するため，各層に活性化関数を適用して非線形性を導入する．活性化関数を適切に組み合わせた多層ネットワークは，任意の連続関数を任意の精度で近似できることが知られており，これを**普遍近似定理**という．

活性化関数は非線形性を導入するだけでなく，以下の観点も考慮して設計される．

- **微分可能性**：勾配降下法による学習のために，関数がほとんどの領域で微分可能である必要がある．
- **勾配消失問題**：シグモイドのように入力が大きいと勾配が小さくなる関数は，深いネットワークでは勾配消失を引き起こしやすい．
- **出力の分布**：出力の平均や分散が一定範囲に収まることで学習が安定する．

このように，活性化関数はニューラルネットワークの表現力と学習効率の双方に大きな影響を与えるため，タスクやモデルに適した関数を選択する必要がある．

3.2　さまざまな活性化関数

活性化関数は入力が大きくなると出力が一定の範囲に収まる**飽和型**と，入力の増加に伴って出力が線形に増加する**非飽和型**に大別される．

Sigmoid 関数

Sigmoid 関数 (Logistic 関数) は次の式で定義される．

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

出力範囲は (0,1) で確率的な解釈が可能であり，特に二値分類タスクの出力層に適している．一方で，入力が大きくなると勾配がほぼゼロになり，**勾配消失問題**を引き起こす．また，出力の分布の中心（平均）が 0 付近ではなく 0.5 付近であるため，出力の分布が正の領域に偏って最適化が遅くなる傾向がある（**図 3.1** 参照）．

tanh 関数

tanh 関数は Sigmoid 関数をスケーリング・平行移動した形であり，次のように定義される．

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

出力範囲は (-1,1) で活性化の分布の中心が 0 付近であるため，シグモイド関数に比べて重み更新で勾配の方向が偏りにくく，より学習が早く進むことが多い．ただし，入力が大きいときに飽和し勾配が消失する点は共通する（**図 3.2** 参照）．

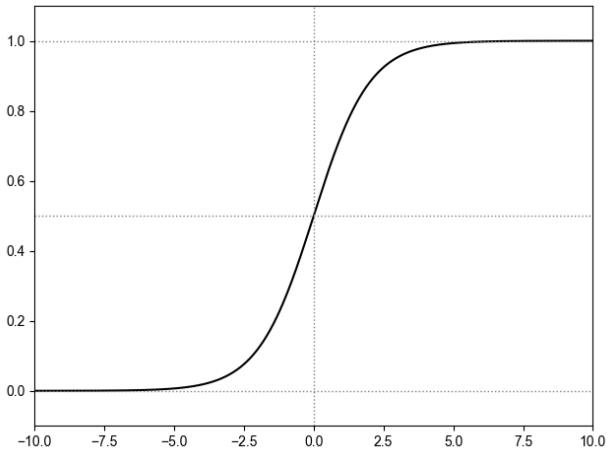

図 3.1: Sigmoid 関数

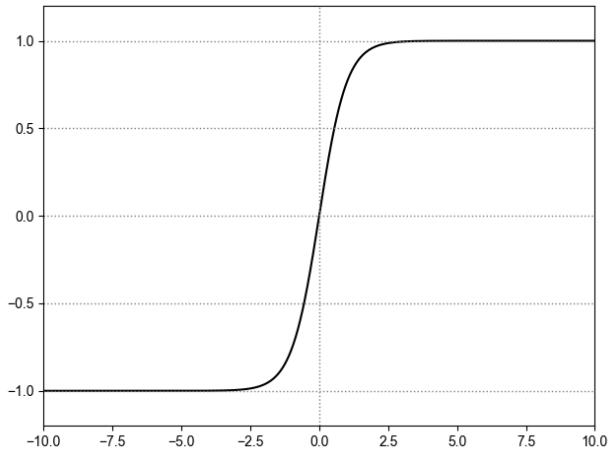

図 3.2: tanh 関数

Sigmoid 関数と tanh 関数はともに飽和型であり，入力が大きな正負の値になると，出力が上限・下限に張り付いて勾配がほぼ 0 になる．一方，次に述べる ReLU 関数は非飽和型の代表例であり，正の入力に対しては勾配 1 で飽和しない．

ReLU 関数

ReLU(Rectified Linear Unit) 関数は現在最も広く用いられる活性化関数であり，次の式で定義される．

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

計算コストが小さく，正の入力では勾配が消失しないため深いネットワークでの学習に適している．一方，負の領域では出力と勾配が常にゼロとなり，ニューロンが活性化しなくなる**ニューロンの死 (dead neurons)** とよばれる問題が発生することがある．

Leaky ReLU 関数

ReLU 関数の「ニューロンの死」を緩和するため，負の領域にも微小な傾きを与えるのが **Leaky ReLU 関数**である．

$$\mathrm{LeakyReLU}(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases}$$

ここで α は小さい正の定数（通常 0.01 程度）である．これにより負の入力に対する勾配が消失しない．α を学習可能パラメータとして扱う **PReLU(Parametric ReLU)** とよばれる関数もあり，データに応じて負の領域における傾斜を適応的に学習できる．

Softmax 関数

Softmax 関数は主に多クラス分類の出力層で利用される．入力ベクトル $z \in \mathbb{R}^K$ に対して，各出力は次のように計算される．

$$\mathrm{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

Softmax 関数の出力は各クラスの確率分布を表し，総和が 1 になる．このため，**交差エントロピー損失**と組み合わせて多クラス分類に用いられる．

まとめ

活性化関数は深層学習の性能や学習の安定性に直接影響する．シグモイドや tanh は古典的だが勾配消失しやすい．一方，ReLU 系は勾配消失を抑えられるがニューロンの死に注意が必要である．近年では Swish や ELU など新しい関数も実験的に使われている．タスクやネットワークの深さに応じて最適な関数を選択することが重要である．

3.3 活性化関数の性質と選択

活性化関数の特徴比較

活性化関数は学習の効率や収束に大きな影響を及ぼすため，性質を理解して選択することが重要である．代表的な活性化関数をいくつかの観点から比較する．

► 出力範囲

- Sigmoid：$0 \leq \sigma(x) \leq 1$
- tanh：$-1 \leq \tanh(x) \leq 1$
- ReLU：$0 \leq \max(0, x)$
- Softmax：$\forall k, \quad 0 \leq \hat{y}_k \leq 1, \quad \sum_k \hat{y}_k = 1$

► 0 中心性

- Sigmoid：0 中心ではない（平均が 0.5 に偏る）
- tanh：0 中心
- ReLU：0 未満の入力では 0

► 飽和性

- Sigmoid / tanh：大きな正・負の入力で勾配が 0 に近づく（飽和）
- ReLU：正の領域では飽和しない（非飽和）

► 微分可能性

- Sigmoid / tanh：全域で微分可能
- ReLU：$x = 0$ で微分が定義されないが実装上問題ない

Point　活性化関数の選択ガイド

Sigmoid 関数や tanh 関数では，入力が大きい場合に勾配がほぼ 0 になる「**勾配消失問題**」が発生しやすいため，層を深くすると学習が進みにくくなる．一方，ReLU 関数は正の領域で飽和しないため，深いネットワークでも勾配を保持しやすい．この特性により近年の深層学習では ReLU 関数が標準的に用いられている．しかし，ReLU 関数は $x < 0$ で勾配が完全に 0 となり，学習が停止する「ニューロンの死」の問題を生じることがある．これを緩和するために，Leaky ReLU 関数や PReLU 関数が提案されている．このように，出力の範囲や 0 中心性，勾配消失への影響を理解して使い分けることが重要である．

まとめ

活性化関数には出力範囲，飽和性，0 中心性，勾配消失耐性などの多様な性質がある．モデルやタスクに応じた適切な選択が学習性能に大きな影響を及ぼす．

4　ニューラルネットワークにおけるスキップ接続

4.1　スキップ接続の背景

深層化による問題

深層ニューラルネットワークでは，層を増やすと表現力が高まることが理論上は期待される．しかし，実際には層を増やしすぎると，活性化関数や重み行列が繰り返し適用されることで，（**2.4 節**で述べたバッチ正規化を用いたとしても）誤差逆伝播の際に勾配が次第に小さくなって重みの更新が滞ったり（**勾配消失**），あるいは大きくなって学習が不安定になったり（**勾配爆発**）する問題があった．

残差接続の導入

従来のネットワークでは，各層は単に次式のような非線形変換を適用する．

$$y = \mathcal{F}(x)$$

これに恒等写像を加えることで，入力をそのまま次の層に伝える**残差接続（スキップ接続）**とよばれる経路をつくる．ここで $\mathcal{F}(x)$ は入力と出力の差分であり，学習すべき変換が単なる恒等写像である場合は $\mathcal{F}(x)$ を 0 に近づければよい．

$$y = \mathcal{F}(x) + x$$

勾配降下法による学習時にスキップ接続を経由することで，深いネットワークであっても，層をショートカットした比較的大きな勾配が出力側から遠い層まで逆伝播していきやすい．この仕組みが残差学習の本質であり，深層化による性能低下を克服する鍵となった．これは，近年で特に重要なモデルである **Transformer** にも導入されている非常に重要な技術である．

まとめ

深層ネットワークでは，層を深くするほど勾配消失や過学習によって性能が低下する問題があった．そこで，スキップ接続を導入することにより，入力をそのまま次の層に伝える経路を持たせることで学習を安定させ，深い層でも性能を維持することができる．

4.2　ResNet の構造

ResNet (Residual Network) は，**スキップ接続**を利用して非常に深いネットワークの学習を成功させた代表的なモデルである．2015 年の ILSVRC(ImageNet Large Scale Visual Recognition Challenge) で登場し，152 層もの深さのネットワークでも学習が可能であることが示された．

残差ブロック

ResNet の基本単位は**残差ブロック** (residual block) である（**図 4.1** 参照）．残差ブロックは，入力 x と畳み込み層を通した変換結果を直接加算する構造となっており，勾配が直接伝わる経路が確保される．この仕組みにより，深いネットワークでも性能劣化を防いで理論通りに表現力を向上させることができる．

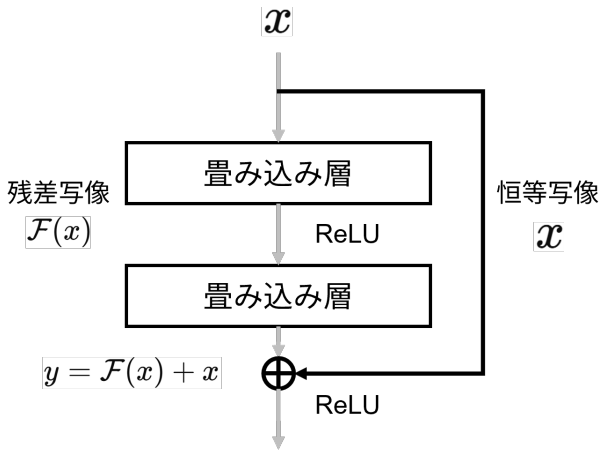

図 4.1: 残差ブロックの構造

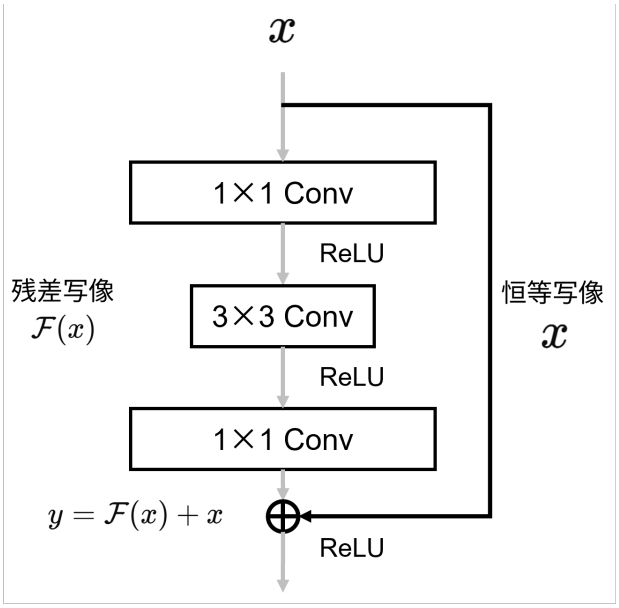

図 4.2: ボトルネック構造

ResNet-50 や ResNet-152 のような非常に深いネットワークでは，**ボトルネック構造**が採用される（**図 4.2** 参照）．これは，1×1 畳み込みを用いてチャンネル数を一時的に減らすことで計算コストを削減する．

4.3 DenseNet の構造

ResNet の改良版として知られる **DenseNet(Densely Connected Convolutional Networks)** は，Dense ブロックとよばれるスキップ接続を導入したネットワークである．ResNet が層ごとに恒等マッピングを足し合わせるのに対し，DenseNet では各層の出力を**後続すべての層に結合**する．具体的には，l 層目の入力は以下のようになる．

$$x_l = H_l\big([x_0, x_1, \ldots, x_{l-1}]\big)$$

ここで $[\cdot]$ はチャネル次元での連結を表す．これにより，特徴が層をまたいで直接共有され，勾配消失の緩和，パラメータ効率の向上，特徴の再利用が実現される（**図 4.3** 参照）．

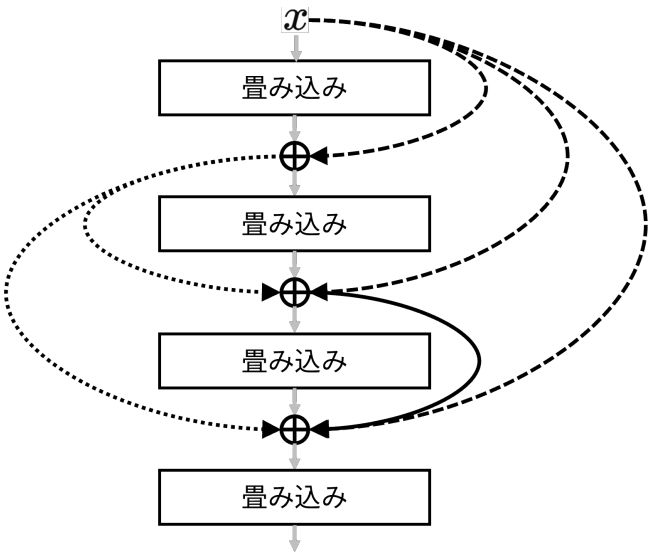

図 4.3: Dense ブロック

5　畳み込みニューラルネットワーク (CNN) の原理

5.1　CNN の基礎

CNN の設計思想

従来の多層パーセプトロン (MLP) は，すべての入力とニューロンを全結合するため，画像のように高次元なデータに適用すると膨大なパラメータが必要になる．この問題を解決するために考案されたのが**畳み込みニューラルネットワーク (Convolutional Neural Network)** であり，空間的な構造を効率よく捉えることができる．

CNN の設計は，以下の考え方に基づく．

- 空間的な局所的特徴が存在する入力に対して，全結合よりも局所的な演算が可能となる．
- 同じフィルタを画像全体に適用する，つまり重みを共有（パラメータ共有）することで，パラメータ数が劇的に削減される．
- 層を重ねることで抽象度の高い特徴を階層的に学習する．

画像認識タスクとの関連

CNN は，特に画像認識において高い性能を示す．たとえば，文字認識や物体分類などのタスクでは，入力画像から特徴を抽出し，どのクラスに属するかを判別する必要がある．これらのタスクでは，画像内の物体の位置が多少ずれても正しく認識できることが重要である．CNN では，同じフィルタ（カーネル）を画像全体に適用することで，特徴が画像のどの位置に現れても検出できる．また，CNN の層を深く積み重ねることで，浅い層がエッジやテクスチャなどの単純なパターンを検出し，深い層ではこれらの情報を統合して物体の部位や全体構造などの抽象的な特徴を学習する．

5.2　畳み込み演算

Point　畳み込み演算の基本

CNN における最も重要な処理は，入力に対して**畳み込み演算** (convolution) を行うことである．畳み込み演算では，学習可能な重みをもつ小さな行列（**カーネル**や**フィルタ**と呼ばれる）を入力画像上でスライドさせ，局所領域ごとに積和を計算する（**図 5.1** 参照）．

畳み込み層における出力は，次のように定義される．

$$y(i,j) = \sum_m \sum_n k(m,n)\, x(i+m, j+n)$$

ここで，$k(m,n)$ はカーネルの重み，$x(i+m, j+n)$ は入力の値である．

ストライドとパディング

- **ストライド (stride)**：カーネルをスライドさせる間隔を示す．ストライドが 1 の場合は 1 ピクセルずつ，2 の場合は 2 ピクセルずつ移動する．
- **パディング (padding)**：入力の周囲に余白を加える操作．主に以下の目的がある．
  - 出力サイズを調整し，空間解像度を保つ
  - 画像端の情報を損なわずに利用する
  - 出力の位置と入力の位置の対称性を保持する

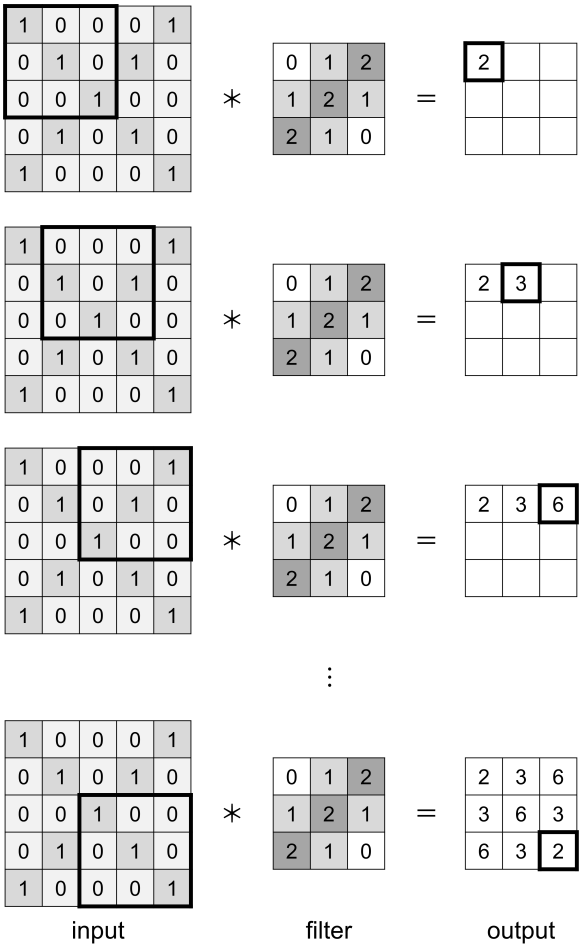

図 5.1: 畳み込み演算

入力の一辺の長さを I，カーネルの一辺の長さを K，ストライドを S，開始位置と終了位置のパディングをそれぞれ $P_{\text{start}}, P_{\text{end}}$ とすると，出力の対応する辺の長さは次の式で計算される．

$$O = \frac{I - K + P_{\text{start}} + P_{\text{end}}}{S} + 1$$

まとめ

畳み込み演算はCNNの基盤であり，カーネルを用いた局所的な積和計算により特徴を抽出する．ストライドとパディングを適切に設定することで，出力サイズを調整できる．

5.3 プーリングと逆プーリング

プーリングの概要

プーリング (pooling) は，畳み込み層の出力に対して空間的な次元を縮小し，重要な情報を集約する処理である．重要な情報を集約する処理である．通常，畳み込み層と活性化関数の後に配置される．たとえば，畳み込み → 活性化 → プーリング という流れでネットワークを構築し，段階的に空間解像度を減らしながら抽象度の高い特徴を抽出する．

プーリングには，以下の効果がある．

- 局所的な平行移動に対する不変性の獲得
- 特徴量の抽象化
- 計算量の削減
- 過学習の抑制

最大プーリング

最も一般的な手法が**最大プーリング** (max pooling) である．これは，ある領域内の最大値を代表値として抽出する（**図 5.2** 参照）．たとえば 2×2 のウィンドウをストライド 2 で適用する場合，次のように特徴マップの縦横のサイズが半分になる．ここで $\mathcal{R}_{i,j}$ はウィンドウ領域を表す．

$$Y_{i,j} = \max_{(m,n) \in \mathcal{R}_{i,j}} X_{m,n}$$

平均プーリング

平均プーリング (average pooling) は，ウィンドウ内の平均値を計算する（**図 5.3** 参照）．最大プーリングと比較すると，出力が滑らかになりやすい．

$$Y_{i,j} = \frac{1}{|\mathcal{R}_{i,j}|} \sum_{(m,n) \in \mathcal{R}_{i,j}} X_{m,n}$$

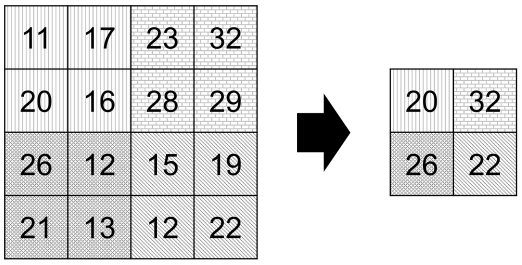

図 5.2: 最大プーリング

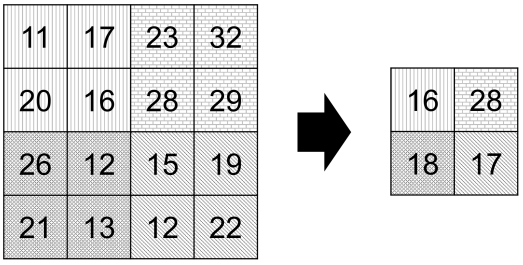

図 5.3: 平均プーリング

基本的な最大プーリングや平均プーリング以外にも，さまざまな応用的なプーリング手法が提案されている．入力特徴マップの中から任意の矩形領域を指定してその領域を固定サイズに変換する **ROI(Region of Interest) プーリング**は，物体検出タスクで用いられる．空間方向における平均をチャネルごとに 1 つの値に集約する**グローバル平均プーリング**は，パラメータ数を削減して過学習を抑制する効果がある．

逆プーリング

通常のプーリングは情報を縮約するため，元のサイズに戻すことはできない．しかし，**逆プーリング** (unpooling) と呼ばれる操作により，近似的に元の形に復元することが試みられる．主な方法は，プーリング時に記録しておいた最大値の位置（インデックス）に対応する位置に値を配置し，それ以外はゼロを埋める方法である（**図 5.4** 参照）．

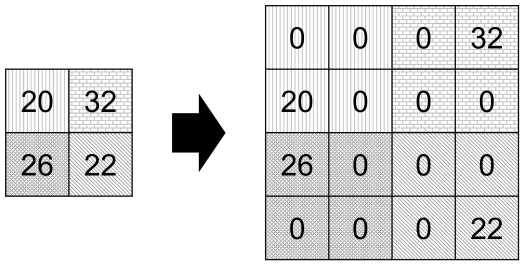

図 5.4: Max Unpooling

まとめ

プーリングは特徴マップの空間サイズを縮小し，重要な情報を集約する操作である．最大プーリングは最大値を抽出し，平均プーリングは平均値を用いる．逆プーリングでは縮約前の形を近似的に再構成できる．

5.4 特殊な畳み込み

近年の深層学習では，モデルの表現力や効率をさらに高めるため，さまざまな特殊な畳み込み手法が提案されている．

転置畳み込み（デコンボリューション）

転置畳み込み (transposed convolution) は，入力の空間サイズを拡大するために用いられる．主に**生成モデル**や**セグメンテーションモデル**のデコーダ部分で利用される．通常の畳み込みがサイズを縮小する演算であるのに対し，転置畳み込みは学習可能なパラメータを使ってアップサンプリング（データの追加）を行う．

膨張畳み込み

膨張畳み込み (dilated convolution) は，畳み込みカーネルの要素の間に空白を挿入することで，パラメータ数を増やさずに受容野を拡大する手法である．特に音声認識やセグメンテーションなど，グローバルな文脈が必要なタスクで有効である．

可分畳み込み

可分畳み込みは従来の畳み込みを計算効率よく分解する手法で，次の 2 段階で構成される．

- **Depthwise Convolution**：各チャネルに独立したフィルタを適用する．
- **Pointwise Convolution**：1×1 の畳み込みでチャネル間を線形結合する．

MobileNet など軽量モデルで広く利用され，計算量の大幅削減と性能維持を両立する．

グループ畳み込み

グループ畳み込み (group convolution) は，チャネルを複数のグループに分割し，それぞれ別々に畳み込みを行う手法である．AlexNet で初めて大規模に活用され，チャネル間の依存性を調整しながら計算量を抑える効果がある．

6　再帰型ニューラルネットワーク (RNN) の原理

6.1　RNN の基礎

RNN の必要性と設計思想

前章で述べた畳み込みニューラルネットワーク (CNN) は，局所的な空間構造の抽出に優れたモデルであり，入力間の時間的な依存関係や順序情報を扱うことが難しい．しかし，自然言語処理や音声認識，時系列予測などの多くのタスクでは，過去の情報が現在の判断に影響を与える．このようなタスクに適したモデルが，**リカレントニューラルネットワーク (Recurrent Neural Network)** である．

RNN では，現在の入力だけでなく，過去の入力から得られた内部状態（**隠れ状態**）を用いて処理を行う．これにより，系列内での文脈を反映した出力が可能となる．以下のような設計上の特徴をもつ．

- 過去の情報を隠れ状態として保持し，逐次的に更新しながら処理を行う．
- 同一のパラメータを全時刻で共有し，系列の長さに依存せずモデルを構築できる．
- 入力の時間的構造を保持したまま学習を行うため，系列全体の構造や依存性を捉えることができる．

系列データと RNN の応用例

RNN は，以下のような**系列データ**に対して効果的に機能する．

- **自然言語処理 (NLP)**：単語の順序が文の意味を決定するため，過去の単語に基づいて文脈を理解・生成する．
- **音声認識**：音の時間的変化を連続的に捉え，発話の意味を抽出する．
- **時系列予測**：株価，気象データ，センサーデータなどの連続的な予測する．
- **動画処理**：フレーム間の動きやイベントの継続性を捉える．

FNN/CNN との違い

RNN が FNN（フィードフォワードニューラルネットワーク）や CNN と本質的に異なるのは，**入力系列間の依存関係**を明示的にモデリングする点である．たとえば FFN は，位置情報や順序情報を内部に保持する仕組みがないため，時系列構造を考慮できない．CNN は局所的なパターンに対して強いが，系列順序の保持には適していない．一方，RNN は以下のような点でこれらと異なる．

- 入力の順序が出力に反映される（ある時刻における出力はそれ以前の情報に依存）
- 「前の入力の影響」がモデル内部に保持される（記憶能力）
- 時間軸に沿って同じ構造を繰り返すことで，可変長の系列を柔軟に処理できる

まとめ

RNN は，系列データに内在する時間的依存関係をモデリングするために設計されたニューラルネットワークである．FFN や CNN と異なり，時間順序を考慮した動的な内部状態を持ち，自然言語や音声，時系列予測などに強みを発揮する．

6.2 RNN の構造

RNN の基本構造

RNN では，入力系列 $\{x_0, x_1, x_2, \ldots, x_T\}$ に対して，時刻 t における出力 y_t と隠れ状態 h_t を逐次的に計算する．隠れ状態は，過去の情報を内部的に保持する役割を果たす．最も基本的な RNN の構造は次の式で表される．

Point　RNN の数式表現

$$h_t = \phi(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$
$$y_t = W_{hy}h_t + b_y$$

x_t：時刻 t の入力ベクトル
h_t：時刻 t の隠れ状態ベクトル（メモリのような役割）
y_t：時刻 t の出力
W_{xh}：入力から隠れ状態への重み行列
W_{hh}：前の隠れ状態から現在の隠れ状態への重み行列（再帰的に結合）
W_{hy}：隠れ状態から出力への重み行列
b_h：隠れ状態計算のバイアス項
b_y：出力計算のバイアス項
ϕ：活性化関数（通常は tanh や ReLU など）

逐次処理と再帰的な構造

この構造（**図 6.1** 参照）の特徴は，隠れ状態 h_t が時系列全体を通して再帰的に定義されている点である．つまり，ある時刻の出力はそれ以前の入力すべてに依存している．これにより，RNN は長期的な文脈や依存関係をモデリングできる．

RNN は，隠れ状態 h_t を介して時間的な文脈を次の時刻に伝えることができる．これにより，過去の入力が将来の出力に影響を与えるような動的な系列データを扱えるようになる．

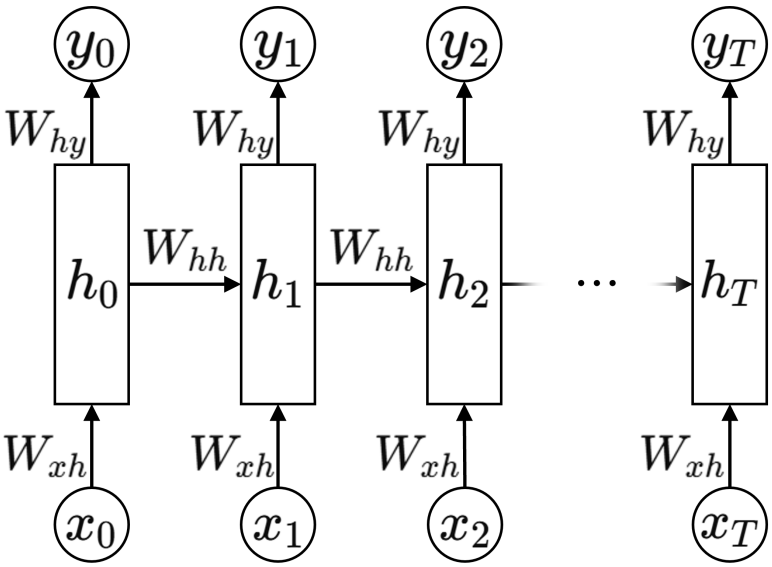

図 6.1: RNN の構造

6.3 RNNの学習と課題

時間方向の誤差伝播

RNNの学習では，出力誤差を逆伝播させて重みを更新する必要がある．ただし，RNNは時間方向に繰り返し構造を持つため，FNNと異なり，時間軸に沿って誤差を伝播する必要がある．このアルゴリズムは**通時的誤差逆伝播法 (Backpropagation Through Time; BPTT)** と呼ばれ，以下の手順で構成される．

1. 時間$t = 1$からTまでネットワークを展開し，隠れ状態と出力を計算
2. 出力誤差を計算し，時間$t = T$から1に向かって誤差を逆伝播
3. 各時刻における勾配を合計して重みを更新

この手法により，ネットワークは系列全体の損失に基づいて一貫した学習を行うことが可能となる．

RNNの課題

- **勾配爆発**：BPTTでは隠れ状態を通じて時間方向に誤差が伝播されるが，その過程で勾配が発散して重み更新が不安定になることがある．この問題に対処するためには，一定の閾値θを超えた勾配の大きさを制限する**勾配クリッピング (gradient clipping)** が有効である．$\nabla\mathcal{L}$を損失関数の勾配とすると，以下のように定式化される．

$$\text{if } \|\nabla\mathcal{L}\| > \theta, \quad \nabla\mathcal{L} \leftarrow \theta \cdot \frac{\nabla\mathcal{L}}{\|\nabla\mathcal{L}\|}$$

- **勾配消失**：逆に，勾配の値が指数的に小さくなって誤差が初期時刻まで届かず，RNNが長期的な文脈を学習することが困難になることがある．この課題を根本的に解決するため，次節以降で述べる **LSTM(Long Short-Term Memory)** や **GRU(Gated Recurrent Unit)** といった構造が提案された．

- **長期依存関係のハードル**：系列が長い場合，離れた要素の情報を保持するのが難しい．最後の隠れ状態は系列全体の情報を保持していることが望まれるが，系列の最初のほうの入力の情報は“ 薄まってしまう ”ことが知られており，この解決にもLSTMが有効である．

6.4 LSTMとGRU

LSTMの設計思想

前節で述べた通り，従来のRNNは長期依存関係の学習が困難であるという課題を抱えていた．この問題を解決するために提案されたのが **Long Short-Term Memory(LSTM)** であり，時間的に離れた情報を保持・伝搬するための**記憶セル (memory cell)** を導入している．

LSTMは，情報の流れを制御するために以下の3つの**ゲート機構**を備える（**図 6.2** 参照）．

- **忘却ゲート (forget gate)** f_t：過去の記憶セルからどのくらい情報を残すかを制御．
  過去の情報c_{t-1}をどれだけ保持するか決定．
- **入力ゲート (input gate)** i_t：現在の入力からどのくらい情報をセルに加えるかを制御．
  現在の入力x_tから新しい情報$\tilde{c}_t$をどれだけ反映するか決定．
- **出力ゲート (output gate)** o_t：現在の記憶セルからどのくらい情報を出力するかを制御．
  記憶セルc_tのどの情報から出力h_tを生成するか決定．

これらのゲートにより，必要な情報を保持し，不要な情報は削除することで，LSTMは長期間にわたる系列情報を安定して記憶・更新し，RNNの勾配消失の問題を大幅に軽減することができる．

LSTM の基本構造

LSTM における状態の更新は，以下の式によって表される．

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) && \text{（忘却ゲート）} \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) && \text{（入力ゲート）} \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) && \text{（出力ゲート）} \\
\tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) && \text{（新しいセル候補）} \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t && \text{（記憶セルの更新）} \\
h_t &= o_t \odot \tanh(c_t) && \text{（出力の計算）}
\end{aligned}
$$

ここで，$\odot$ は要素ごとの積（アダマール積），x_t は現在の入力，h_{t-1} は前の時刻の隠れ状態を表す．記憶セル c_t は，記憶を保持し続けるメインの経路として働き，必要な情報だけを通すことで学習を安定化させる．

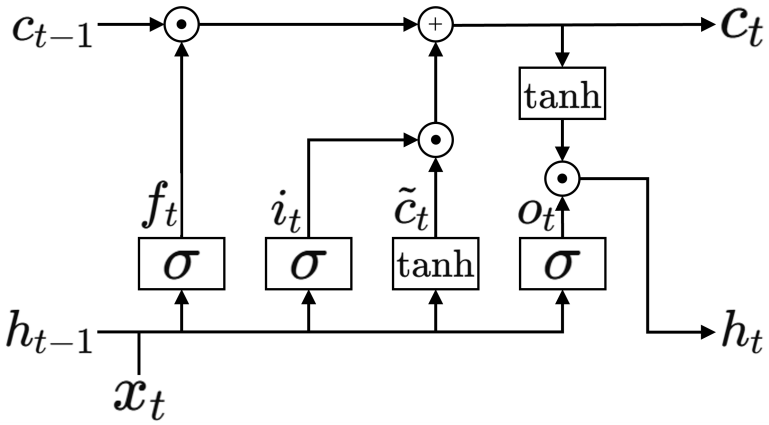

図 6.2: LSTM の構造

GRU の特徴

LSTM の登場によって，RNN では困難だった長期的特徴の学習が可能となったが，計算コストが大きいという問題があった．そこで，LSTM で用いられていた記憶セルと 3 つのゲートを簡略化し，**更新ゲート (update gate)** と**リセットゲート (reset gate)** の 2 つに統合したのが **GRU(Gated Recurrent Unit)** である．この簡略化により，パラメータ数が削減されて学習効率が向上する一方で，LSTM に匹敵する性能を発揮することが多い．

GRU の動作は以下の式で表現される．

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) && \text{更新ゲート：過去の状態をどれだけ保持するか制御} \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) && \text{リセットゲート：過去の情報を無視するかどうか制御} \\
\tilde{h}_t &= \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) && \text{現在の入力から計算される隠れ状態の候補} \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t && \text{新しい隠れ状態：実際の出力となる隠れ状態}
\end{aligned}
$$

更新ゲート z_t が 1 に近いと新しい情報を重視し，0 に近いと過去の状態を保持する．一方，リセットゲート r_t が 0 に近いと，過去の情報を無視して新しい入力だけに基づく計算が行われる．

まとめ

LSTM は，従来の RNN では困難だった長期依存関係の学習を可能にする構造である．記憶セルとゲート機構で重要な情報を保持しつつ，不要な情報を忘却することができる．これによって勾配消失を防ぎ，安定した学習が実現されたが，計算コストが大きいという問題があった．LSTM の簡略版である GRU は，簡潔な構造ながらも LSTM と遜色ない性能を持ち，更新ゲートとリセットゲートで状態の保持と更新を柔軟に制御する．

6.5　双方向 RNN

従来の RNN では，時系列データを**過去から未来**へと一方向に処理するため，現在の出力は**過去の情報にのみ依存**していた．しかし，自然言語処理など多くのタスクにおいては，**未来の情報も同様に重要**となる場合がある．たとえば，ある単語の品詞や意味を判定する際には，後続の単語も考慮した方が精度が向上する．

この問題を解決するために提案されたのが，**双方向 RNN(Bidirectional RNN; Bi-RNN)** である．BiRNN では，**順方向（過去から未来）**と**逆方向（未来から過去）**の 2 つの RNN を同時に動かし，それぞれの出力を統合することで，**文脈の前後関係の両方**を考慮した出力を得ることができる（**図 6.3** 参照）．

Bi-RNN の構造

入力系列 $\{x_0, x_1, x_2, ..., x_T\}$ に対して，順方向と逆方向の隠れ状態を以下のように計算する：

$$\overrightarrow{h}_t = \mathrm{RNN}_{\mathrm{forward}}(x_t, \overrightarrow{h}_{t-1}) \qquad \overrightarrow{h}_t：\text{順方向の隠れ状態}$$

$$\overleftarrow{h}_t = \mathrm{RNN}_{\mathrm{backward}}(x_t, \overleftarrow{h}_{t+1}) \qquad \overleftarrow{h}_t：\text{逆方向の隠れ状態}$$

$$y_t = \mathrm{concat}(\overrightarrow{h}_t, \overleftarrow{h}_t) \qquad y_t：\text{両方向の隠れ状態を結合した出力}$$

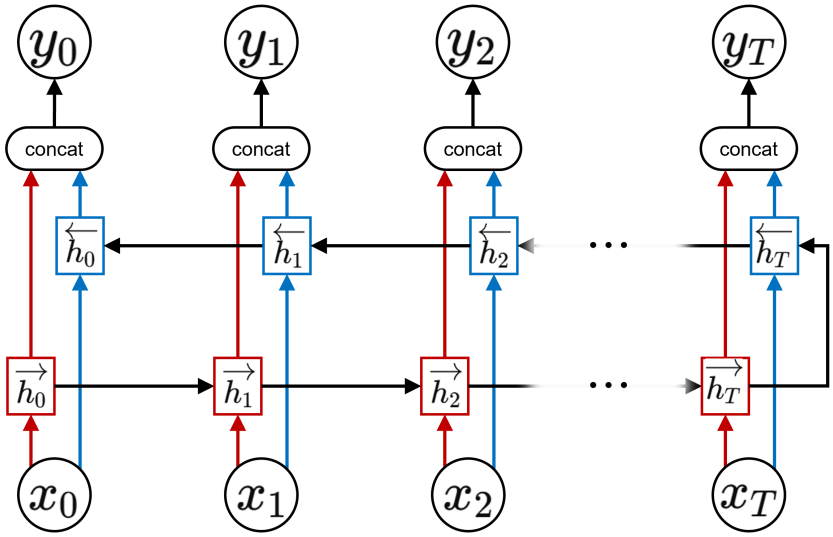

図 6.3: 双方向 RNN の構造

Bi-RNN の利点と応用

- **文脈の双方向性の活用**：各時点において，前後のコンテキストを利用することで，より豊かな表現を得ることができる．
- **自然言語処理への応用**：品詞タグ付け，固有表現抽出，機械翻訳，音声認識など多くのタスクで BiRNN が高い性能を示す．
- **可逆な系列処理**：特定のタスクにおいては，入力系列を前後から対称的に処理する必要があるため，BiRNN が適している．

まとめ

双方向 RNN は，順方向と逆方向の 2 つの RNN を用いて，時系列データの両方向の文脈を同時に捉えることができるモデルである．これにより，単方向 RNN よりも表現力が高まり，特に自然言語処理タスクにおいて強力な性能を発揮する．

6.6　RNN の応用と発展

Encoder-Decoder 構造

RNN は，系列データを扱う多くのタスクで利用されるが，その中でも特に重要な構造が **Encoder-Decoder アーキテクチャ**である．この構造は，入力系列 $\{x_0, x_1, x_2, ..., x_T\}$ を一旦ベクトルにエンコードし，その後，別の RNN によって出力系列 $\{y_0, y_1, y_2, ..., y_T\}$ をデコードする仕組みをもつ（**図 6.4** 参照）．

たとえば，日本語から英語への機械翻訳タスクでは，入力が「日本語文」，出力が「英語文」となる．Encoder は日本語文を 1 つの文脈ベクトル（コンテキスト）に変換し，Decoder はそのベクトルをもとに英語文を生成する．

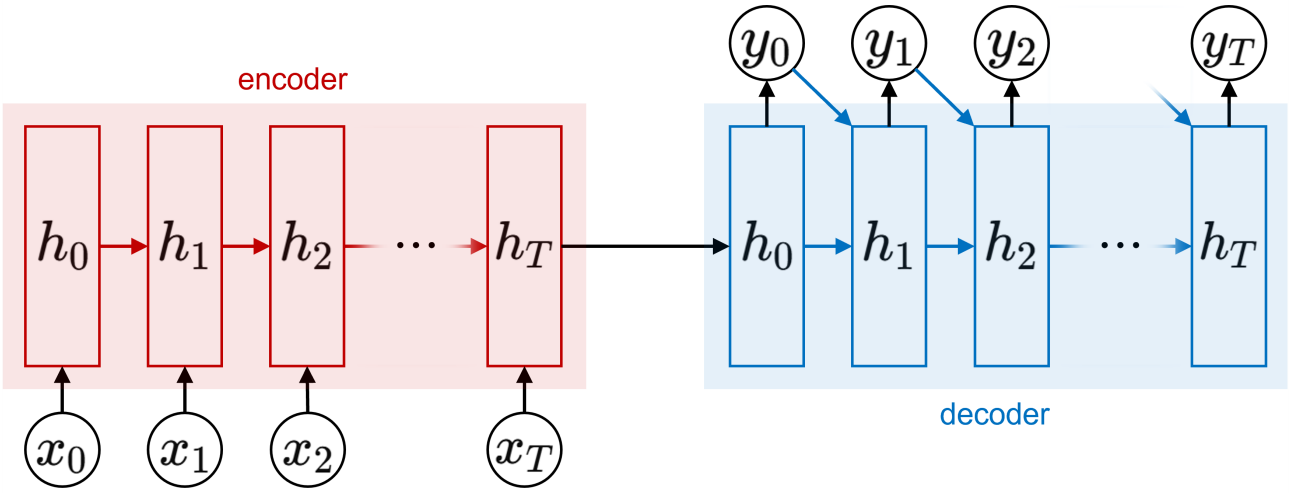

図 6.4: Encoder-Decoder アーキテクチャ

Attention 機構の導入

Encoder-Decoder の基本形では，入力系列全体を 1 つのベクトルに圧縮するため，長文や複雑な構造を十分に記憶できないという問題がある．この制約を解決するために導入されたのが，**Attention 機構**である．Attention では，出力の各ステップにおいて，入力系列のどの部分に注目すべきか（重み付け）を学習により自動で決定する．これにより，長文でも柔軟に重要な情報に焦点を当てることができるようになった．

一般に，Attention 付きの Encoder-Decoder では，以下の処理が行われる．

- ▶ Encoder は各時点の隠れ状態 $\{h_0, h_1, h_2, \ldots, h_T\}$ を出力する
- ▶ Decoder は，各時点でこれらに対して Attention の重みを計算し，文脈ベクトルを得る
- ▶ この文脈ベクトルを使って次の出力を生成する

RNN の発展

RNN は系列を逐次的に処理する特性上，処理の並列化には制約があり，学習や推論の高速化が難しい．しかし現在では，次章で述べる **Transformer** が RNN に代わって登場し，並列処理が可能となっている．

まとめ

RNN は Encoder-Decoder や Attention と組み合わせることで，翻訳や要約など高度な系列変換タスクに応用されている．一方で，構造上の制約もあり，Transformer などの後継モデルへと発展している．

7 Transformer と BERT

7.1 Transformer の原理

Transformer は，従来の RNN に代わる革新的な系列処理モデルであり，特に自然言語処理において大きな成功を収めている．時系列を 1 ステップずつ逐次的に処理する RNN や LSTM では，並列化が困難であり長期的な依存関係の学習にも限界があった．一方，Transformer では，全ての単語が互いに**注意 (Attention)** を向けることで，どの位置にある単語との関係も柔軟に捉えて系列データ全体を一括で処理する．これにより，時間的な順序を保持しつつも，計算の並列化を可能としている．

自己注意機構

自己注意 (Self-Attention) とは，入力系列の中で，各単語（トークン）が他のすべての単語を参照し，どこに注目すべきかを学習する仕組みである．入力が n 個の単語からなる場合，Transformer はそれぞれに対して，**クエリ (Query)**，**キー (Key)**，**バリュー (Value)** という 3 種類のベクトルを生成する．

ある単語の出力は，その単語の Query と他のすべての単語の Key との内積で計算される類似度に基づいて，その単語の Value を加重平均したものになる．この Attention の重みは次式で計算される．

$$\mathrm{Attention}(Q, K, V) = \mathrm{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

ここで，Q，K，V はそれぞれ Query，Key，Value 行列であり，d_k は Key の次元数である．スケーリングにより，Softmax の出力が極端に 0 と 1 に偏って（ほとんどの要素が 0 になって）勾配が極端に小さくなるのを防ぐ．

マルチヘッドアテンション

単一の自己注意では，特定の関係に偏る可能性がある．これを防ぐために，Transformer は複数の注意機構を並列に動作させる**マルチヘッドアテンション (Multi-Head Attention)** を用いる．それぞれのヘッドが異なる視点で情報を抽出し，その後に統合することで，より多様な文脈を捉えることができる．

位置符号化

Transformer では RNN のような時間順序の構造を持たないため，単語の並び順（位置情報）を明示的に与える必要がある．このとき用いられるのが**位置符号化 (positional encoding)** である．

もっとも一般的な方法は，正弦波と余弦波を用いた手法であり，位置 pos と次元 i に応じて以下のように定義される．

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$
$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

ここで，d_{model} は埋め込みベクトルの次元である．この方法により，Transformer は相対的・絶対的な位置情報の両方を利用できるようになる．

Transformer は，Encoder と Decoder から成る（**図 7.1** 参照）．Encoder は，入力系列を自己注意で処理し，その文脈情報を埋め込んだ特徴ベクトルに変換する．Decoder はその情報をもとに，新しい系列（例：翻訳文）を生成する．

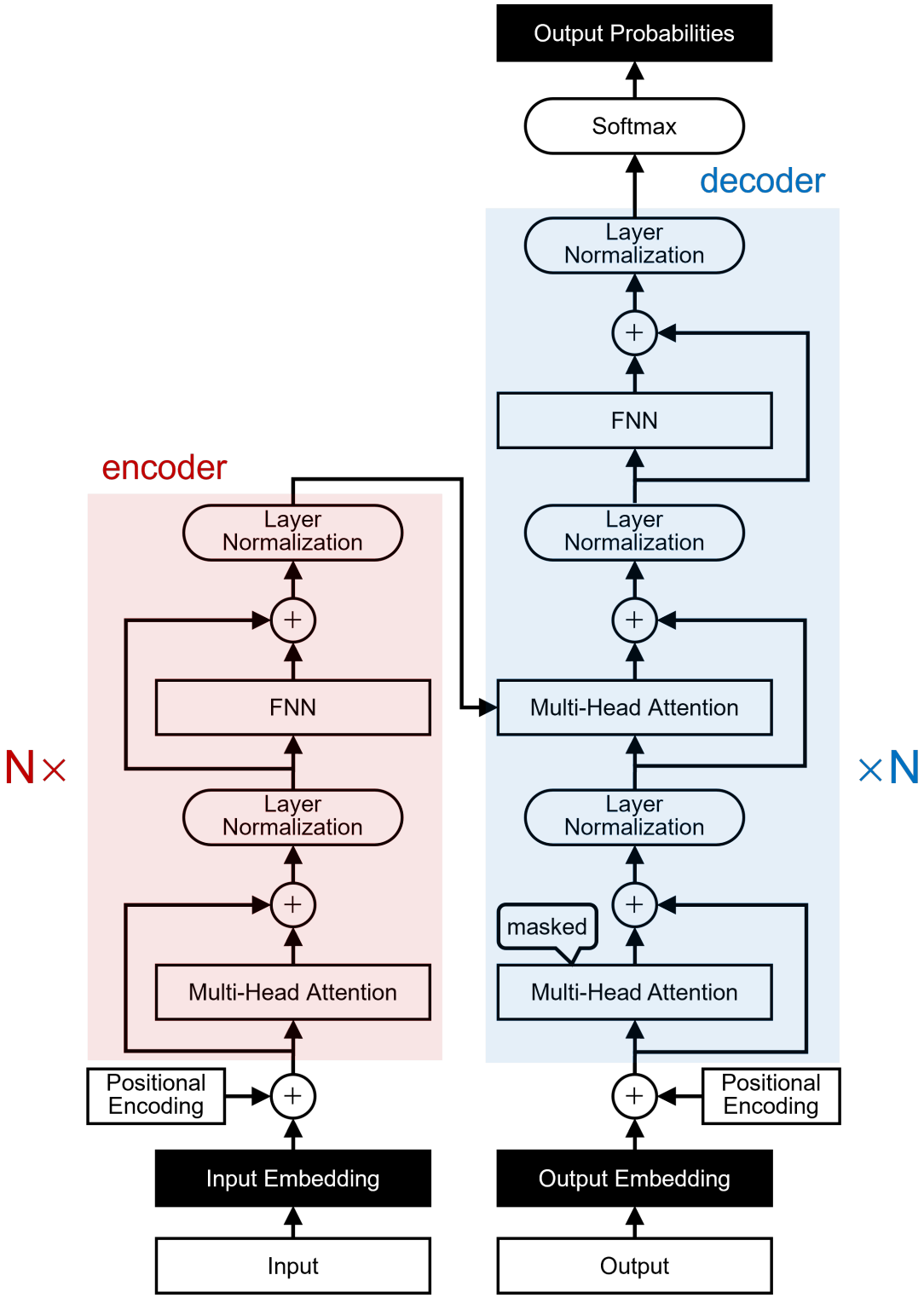

図 7.1: Transformer の構造

まとめ

Transformer は自己注意機構により，系列内の任意の位置に柔軟に注目することができるモデルである．これにより，Transformer は自然言語処理において高性能かつ並列処理可能なモデルとして，BERT や GPT などの基盤技術となっている．

7.2 BERT の仕組み

BERT(Bidirectional Encoder Representations from Transformers) は，Google によって 2018 年に提案された自然言語処理モデルであり，Transformer の Encoder 部分のみを用いた構造を持つ．BERT は双方向の文脈理解が可能な事前学習済みモデルとして，幅広い NLP タスク（質問応答，文の分類，名前付きエンティティ認識など）で画期的な成果を挙げた．

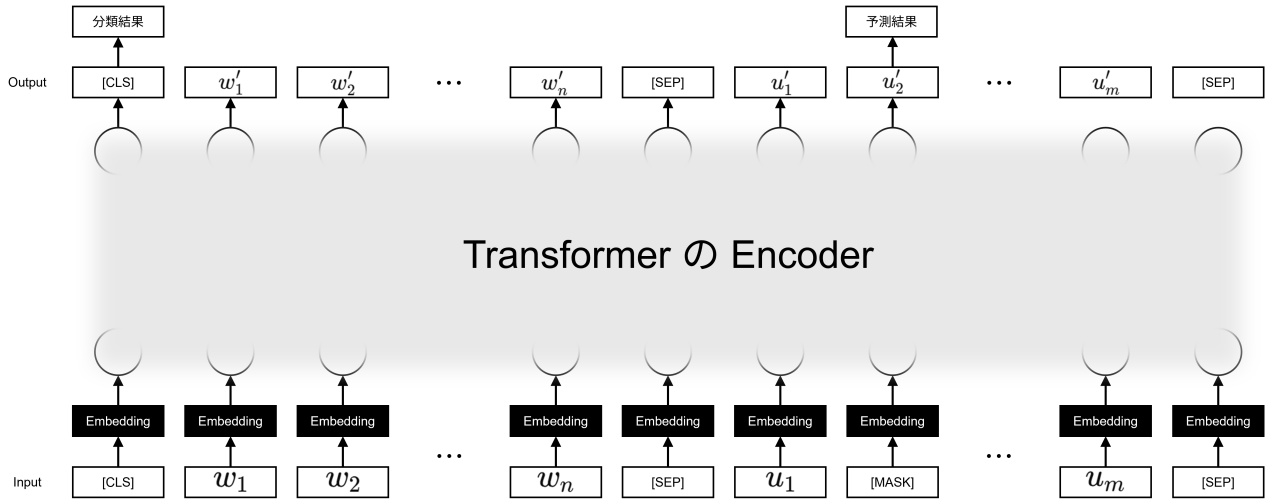

図 7.2: BERT の構造

[CLS]：文頭に挿入される分類用トークン．最終層のこの位置の出力がタスクに利用される．
[SEP]：文の区切りとして文の終端や文同士の境界に挿入される．

BERT の事前学習

BERT は，大規模コーパス上で以下の 2 つのタスクにより事前学習される．

- **Masked Language Modeling (MLM)**：入力文中の一部の単語を [MASK] トークンに置き換え，元の単語を予測させる．文の双方向の文脈を活用して単語の意味を精度よく予測する能力を向上させる．

  例： `The [MASK] barked at the stranger.` → `dog`

- **Next Sentence Prediction (NSP)**：2 文が連続した文かどうかを分類させる．文と文の関係性を学習することで，質問応答や自然言語推論に応用しやすくなる．

  文 A: `He went to the store.` 文 B: `He bought some milk.` → `IsNext`
  文 A: `He went to shopping.` 文 B: `He painted a picture.` → `NotNext`

この 2 つのタスクの損失を同時に最小化することで，BERT は文脈や文間関係の双方を理解する能力を習得する．

Point　BERT の事前学習の損失関数

BERT の事前学習では，MLM の損失 L_{MLM} と NSP の損失 L_{NSP} の和が最終的な損失関数 L となる．

$$L = L_{\text{MLM}} + L_{\text{NSP}}$$

まとめ

BERT は Transformer のエンコーダを基盤とし，双方向の文脈情報を活用する事前学習モデルである．MLM と NSP という 2 つの学習タスクを通じて，高精度な自然言語理解能力を獲得し，さまざまな下流タスクに転移可能な基盤モデルとなっている．

7.3 TransformerとBERTの比較

Transformer と BERT は密接に関連しているが，その目的や構成，応用範囲には明確な違いがある．本節では両者の違いを明確にするために，構造・学習方法・応用タスクの観点から比較する．

構造の違い

- **Transformer** は，エンコーダとデコーダの 2 つのモジュールから構成されている汎用的なアーキテクチャである．エンコーダは入力系列を特徴表現に変換し，デコーダはその表現をもとに出力系列を生成する．これは，機械翻訳のような「入力→出力」型のタスクに適している．
- **BERT** は Transformer のエンコーダ部分のみを利用しており，前から順にトークンを予測するような自己回帰的な生成は行わない．入力文の意味理解に特化しており，翻訳や生成よりも，分類や質問応答といった自然言語理解タスクに最適化されている．

学習方法の違い

- **Transformer** の学習は，教師あり学習・自己教師あり学習など様々な学習パラダイムに対応可能である．例えば，機械翻訳では教師あり学習として入力文と正解訳文のペアを使って学習する．例えば機械翻訳では，入力文に対応する正解の出力文（訳文）を与え，系列生成の確率を最大化するように学習する．デコーダは逐次的に出力を生成し，過去の出力を参照することで次の単語を予測する（自己回帰型）．
- **BERT** は自己教師あり学習を採用し，大量のテキストから単語のマスク予測 (MLM)，あるいは文の関係予測 (NSP) を学習する．双方向的な文脈を考慮できるように設計されており，文全体の理解に優れている．

応用タスクの違い

- **Transformer** はもともと機械翻訳のような入力から出力を生成するタスク向けに設計されたが，その構造は生成型・理解型の両方のタスクに応用可能である．また，Transformer のデコーダ部分を応用して，GPT 系列のような大規模言語モデルも開発されている．
- **BERT** は表現学習と自然言語の意味理解に特化しており，感情分析，文分類，質問応答，固有表現抽出などの下流タスクにおいて微調整 (fine-tuning) して活用される．出力は通常，分類ラベルやスパン（テキスト範囲）として得られる．

項目	Transformer	BERT
構成	エンコーダ + デコーダ	エンコーダのみ
処理方向	順方向 or 自己回帰型	双方向
学習方法	教師あり	自己教師あり
事前学習タスク	なし（通常タスクごとに学習）	MLM + NSP
応用タスク	翻訳・要約・生成	文分類・QA・NER など
使用場面	入力から出力を生成	入力文の意味理解

Table 7.1: Transformer と BERT の比較

8　過学習防止と正則化

8.1　正則化の概要

機械学習モデルは，訓練データに過度に適応しすぎると未知のデータに対する性能が低下する**過学習 (overfitting)** と現象がしばしば問題となる．過学習を防ぐためには，モデルの複雑さを制御する必要があり，その代表的な手法が**正則化 (regularization)** である．

正則化では，損失関数に**ペナルティ項**を追加し，大きすぎるパラメータが学習されることを抑制する．一般的な損失関数 $\mathcal{L}_{\text{data}}$ に正則化項 $\mathcal{R}(\mathbf{w})$ を加えると，最小化すべき目的関数は次のようになる．

$$\mathcal{L}_{\text{total}}(\mathbf{w}) = \underbrace{\mathcal{L}_{\text{data}}(\mathbf{w})}_{\text{元の損失関数}} + \underbrace{\lambda\,\mathcal{R}(\mathbf{w})}_{\text{正則化項}}$$

ここで，$\mathbf{w}$ はモデルの学習可能なパラメータ，λ は正則化係数であり，ペナルティの強さを調整する役割を持つ．

8.2　L1 正則化

L1 正則化は，パラメータの絶対値の和に比例したペナルティを課す手法である．損失関数 $\mathcal{L}_{\text{data}}$ に対して，L1 正則化項を加えると次のようになる．

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda \sum_{i} |w_i|$$

ここで，w_i は i 番目のパラメータ，λ は正則化係数である．

特徴と効果

- パラメータの多くをゼロにする傾向があり，**スパース性** (sparsity) を生む．
- 不要な特徴量を自動的に排除するため，**特徴選択** (feature selection) としての役割を持つ．
- 幾何的には，制約領域がダイヤ型になるため，座標軸との交点（ゼロ成分）が解として選ばれやすい（**図 8.1** 参照）．

ラッソ回帰との関係

L1 正則化を線形回帰の損失関数に適用したものが**ラッソ回帰 (Lasso regression)** である．ラッソ回帰は以下の式で表される．

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \left\{ \frac{1}{2n} \sum_{j=1}^{n} (y_j - \mathbf{x}_j^\top \mathbf{w})^2 + \lambda \sum_{i} |w_i| \right\}$$

ここで，n はサンプル数，$\mathbf{x}_j$ は j 番目の入力ベクトル，y_j は対応する真のラベルである．ラッソ回帰は高次元データにおいても不要な特徴量をゼロにし，解釈性を向上させる効果がある．

まとめ

L1 正則化はスパース性を生み，不要な特徴量を自動的に除去できる．ラッソ回帰はその代表的な応用であり，特に特徴選択が重要な場面で有効である．

8.3　L2 正則化

L2 正則化は，パラメータの二乗和に比例したペナルティを課す手法である．損失関数 $\mathcal{L}_{\text{data}}$ に対して，L2 正則化項を加えると次のようになる．

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda \sum_i w_i^2$$

ここで，w_i は i 番目のパラメータ，λ は正則化係数である．

特徴と効果

- パラメータを小さく保つことでモデルの複雑さを抑える．
- 全てのパラメータを均等に縮小する傾向があり，スパース性は生じない．
- 幾何的には，制約領域が球形になるため，解は原点方向へ滑らかに縮小される（**図 8.1** 参照）．

リッジ回帰との関係

L2 正則化を線形回帰の損失関数に適用したものが**リッジ回帰 (Ridge regression)** である．リッジ回帰は次の式で表される．

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \left\{ \frac{1}{2n} \sum_{j=1}^{n} (y_j - \mathbf{x}_j^\top \mathbf{w})^2 + \lambda \sum_i w_i^2 \right\}$$

リッジ回帰は，全ての特徴量を保持しつつその影響を均等に小さくするため，**多重共線性**（異なる特徴量間の相関）のあるデータや過学習を防ぐ場合に有効である．

まとめ

L2 正則化は全パラメータを均等に縮小し，モデルの複雑さを抑える．リッジ回帰はその代表的な応用であり，多重共線性対策や汎化性能の向上に適している．

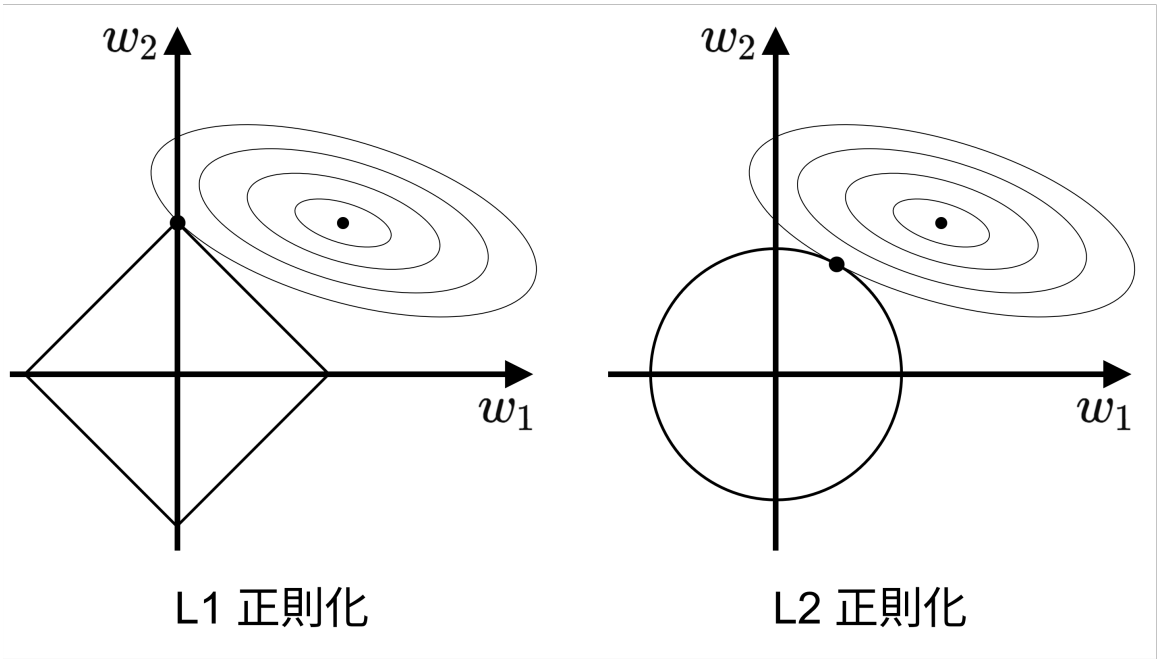

図 8.1: 正則化の幾何的解釈

8.4 L1・L2 正則化の比較と Elastic Net

L1 と L2 の比較

L1 正則化（ラッソ回帰）はパラメータの一部をゼロにして特徴選択を行う効果がある一方，L2 正則化（リッジ回帰）は全てのパラメータを均等に縮小しスパース性は生まない．両者の特性を**表 8.1** に示す．

項目	L1 正則化	L2 正則化
主な効果	特徴選択（スパース性）	パラメータ縮小
解の性質	一部の係数が 0 になる	全ての係数が小さくなる
幾何的制約	ダイヤモンド型	球形
代表的応用	ラッソ回帰	リッジ回帰

Table 8.1: L1 正則化と L2 正則化の比較

Elastic Net

Elastic Net は，L1 正則化と L2 正則化を組み合わせた手法であり，次のように表される．

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda_1 \sum_i |w_i| + \lambda_2 \sum_i w_i^2$$

ここで，λ_1 と λ_2 はそれぞれ L1 と L2 の正則化係数である．

特徴

- L1 のスパース性と L2 の安定性の両方を享受できる．
- 高次元データで多重共線性が存在する場合に有効．
- ラッソが選択しやすい単一特徴への依存を緩和し，よりバランスの取れた特徴選択を実現．

まとめ

L1 正則化は特徴選択に，L2 正則化はパラメータの安定化に有効である．Elastic Net はその両者を組み合わせ，高次元かつ多重共線性のあるデータに強い．

【過学習を防止する主な手法】

これらの手法は単独で使うだけでなく，モデルの特性やデータの性質，利用可能な計算資源を踏まえて適切に組み合わせることが重要である．状況に応じた戦略的な選択と調整により，過学習を抑えつつ高い汎化性能を維持できる．

- **データ拡張**：学習データに回転・平行移動・ノイズ付加などの加工を加え，見かけのデータ数を増やして汎化性能を高める．
- **ドロップアウト**：学習時にランダムにニューロンを無効化し，特定のニューロンや経路への依存を抑制する．
- **早期終了**：検証データの損失が改善しなくなった時点で学習を停止し，過学習を防ぐ．
- **正則化**：パラメータの大きさに応じたペナルティを加え，線形和の重みの増大を防ぐ．
- **バッチ正規化**：各ミニバッチごとに出力を正規化し，内部共変量シフトを軽減することで過学習を防ぐ．
- **モデルの簡略化**：パラメータ数を減らすことで表現力を抑制し，過学習のリスクを下げる．
- **クロスバリデーション**：データを分割して複数回学習・評価を行い，モデルの汎化性能をより正確に見積もる．

9　生成モデルの原理

9.1　生成モデルの基礎

生成モデル (generative model) は，学習データの確率分布 $p_{\text{data}}(x)$ そのものを学習し，その分布に従う新しいサンプル $\tilde{x}$ を生成することを目的としたモデルである．**1.4 節**で述べたロジスティック回帰やサポートベクターマシン (SVM) のような**識別モデル (discriminative model)** が入力 x に対してクラス y の条件付き確率 $p(y|x)$ を学習するのに対し，生成モデルは入力データ x の周辺分布 $p(x)$，または条件付き生成の場合は $p(x|y)$ そのものを学習する．これにより，訓練データに似た新しいサンプルを作り出すことが可能となった．

Point　生成モデルと確率分布

生成モデルは，学習データが従う確率分布 $p_{\text{data}}(x)$ を近似するモデル $p_\theta(x)$ を学習し，サンプリングによって新しいデータ $\hat{x}$ を生成する．条件付き生成（ラベルや入力に応じた特定の出力生成）の場合は，条件変数 y を与えて $p_\theta(x \mid y)$ を学習する．ここで，θ は確率分布の形を決める可変なパラメータの集合である．

$$\hat{x} \sim p_\theta(x) \quad \text{または} \quad \hat{x} \sim p_\theta(x \mid y)$$

主要なタスク

生成モデルは以下のようなタスクで活用される．

- **画像生成**：人物・風景・物体などの高精細画像の生成，画像修復 (Inpainting)
- **音声生成**：音声合成や音声変換，音楽生成
- **動画生成**：動画合成，動画予測，動画編集
- **テキスト生成**：自然言語文の生成（例：文章作成，翻訳，要約，画像キャプション生成）
- **データ拡張**：学習データ不足を補うための合成データ生成，シミュレーションデータ生成

代表的な生成モデルの種類

生成モデルは，その目的や条件付けの有無によって以下のようなタスクに分類される．

- **無条件生成 (Unconditional Generation)**　**学習対象：$p(x)$**
  ラベルや追加情報を一切与えずにデータ分布 $p(x)$ を直接学習し，新しいサンプルを生成．
  例：GAN による人物画像生成，拡散モデルによるアート画像生成

- **条件付き生成 (Conditional Generation)**　**学習対象：$p(x|y)$**
  追加情報（クラスラベル y，テキスト説明，他の画像など）に基づいてデータを生成．
  例：テキストから画像を生成する Text-to-Image モデル，音声から映像を生成するモデル

- **変換タスク（ドメイン変換）(Transformation / Translation)**　**学習対象：$p(x_{\text{tgt}}|x_{\text{src}})$**
  入力データ x_{src} を別の形式やドメインに変換．
  例：英語 → 日本語，白黒画像 → カラー画像，昼の写真 → 夜の写真

- **補完タスク (Inpainting / Completion)**　**学習対象：$p(x_{\text{missing}}|x_{\text{observed}})$**
  欠損部分を持つデータを入力とし，欠けた領域を自然に埋める生成．
  例：画像の一部削除後の復元，文章中の欠落単語の補完 (Masked Language Modeling)

- **拡張タスク (Data Augmentation)**　**学習対象：$p(\tilde{x}|x)$**
  既存データからバリエーションを生成し，学習データの多様性を向上．
  例：GAN で作った偽画像による学習データ拡張，音声のピッチやスピードの変更

9.2　主要な生成モデルのアプローチ

本節では，代表的な生成モデルである**変分自己符号化器 (VAE)**，**敵対的生成モデル (GAN)**，**自己回帰モデル**，**拡散モデル**について，それぞれの生成過程や学習方法，利点と限界を明らかにする．これらのモデルは，いずれも高次元データの分布を学習し，新しいサンプルを生成する能力を持つが，そのアプローチは大きく異なる．

変分自己符号化器

＜自己符号化器＞

まず，**自己符号化器 (Autoencoder)** とは，入力データを一旦低次元の潜在表現に圧縮し，そこから元のデータを復元することを目的としたニューラルネットワークである．教師なし学習の代表的な手法であり，データの特徴抽出や次元削減，ノイズ除去などに利用される．

オートエンコーダは大きく次の 2 つの部分から構成される．

- **エンコーダ (Encoder)**：入力 $\mathbf{x}$ を潜在変数 $\mathbf{z}$ に変換する．次元削減と特徴抽出を担う．
- **デコーダ (Decoder)**：潜在変数 $\mathbf{z}$ から元の入力を復元する．$\hat{\mathbf{x}}$ を出力する．

オートエンコーダは，元の入力 $\mathbf{x}$ と復元 $\hat{\mathbf{x}}$ の差を最小化するように学習される．一般的には平均二乗誤差 (MSE) が損失関数として用いられる．

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2$$

＜変分自己符号化器＞

変分オートエンコーダ (Variational Autoencoder; VAE) は，オートエンコーダを確率的生成モデルとして拡張したものである．従来のオートエンコーダが潜在変数を決定論的に扱うのに対し，VAE では潜在変数を確率分布（例えばガウス分布）に従ってサンプリングし，その分布から新しいサンプルを生成することができる．オートエンコーダの枠組みを拡張し，エンコーダで潜在変数の分布を推定し，デコーダでデータを復元する．VAE の大きな特徴は，単なる圧縮ではなく，データの生成可能な潜在空間を学習することである．

通常のオートエンコーダでは，エンコーダは入力 $\mathbf{x}$ を潜在変数 $\mathbf{z}$ に直接マッピングする．しかし VAE では，エンコーダは $\mathbf{z}$ の確率分布のパラメータ（平均 $\boldsymbol{\mu}$ と分散 $\boldsymbol{\sigma}^2$）を出力し，その分布に基づいて潜在変数 $\mathbf{z}$ をサンプリングする．

$$\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$$

このアプローチにより，潜在空間は連続的かつ確率的な構造を持つようになり，データ生成や補間を滑らかに行えるようになる．

＜変分自己符号化器の学習＞

学習可能なネットワークに直接乱数サンプリングを入れると，誤差逆伝播ができなくなる．そこで，VAE では**再パラメータ化トリック**を用いる．乱数 $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ を別途生成し，次式で $\mathbf{z}$ を構成する．これにより，$\boldsymbol{\mu}$ と $\boldsymbol{\sigma}$ に対して勾配計算が可能になる．

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$$

VAE では，次の**証拠下限 (ELBO)** を最大化するように学習する．

$$\log p(\mathbf{x}) \geq \underbrace{\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})]}_{\text{再構成項（精度の良さ）}} - \underbrace{D_{\mathrm{KL}}(q(\mathbf{z}|\mathbf{x}) \,\|\, p(\mathbf{z}))}_{\text{正則化項（ズレの大きさ）}}$$

- **復元誤差（再構成誤差）**：元の $\mathbf{x}$ をどれだけうまく再現できるか表す再構成項
- **KL ダイバージェンス**：潜在変数の確率分布 $q(\mathbf{z}|\mathbf{x})$ と事前分布 $p(\mathbf{z})$ の差異を表す正規化項

敵対的生成モデル

敵対的生成モデル (Generative Adversarial Network; GAN) は，**生成器 (Generator)** と**識別器 (Discriminator)** の 2 つのネットワークを対立させて学習する．生成器は本物と見分けがつかないデータを生成することを目指し，識別器は入力が本物か偽物かを判定することを目指す．

- **生成器**：ノイズ $z \sim \mathcal{N}(0, I)$ から $G(z)$ を生成
- **識別器**：入力が実データか生成データかを判定

GAN は次のような**ミニマックスゲーム**として定式化される．生成器 G は乱数 $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$ を受け取り，生成サンプル $G(\mathbf{z})$ を出力する．識別器 D は入力が訓練データ分布 p_{data} 由来か G 由来かを判定し，確率 $D(\mathbf{x})$ を出力する．

$$\min_G \max_D \quad \underbrace{\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})]}_{\text{本物画像を本物と判定したい}} + \underbrace{\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))]}_{\text{偽画像を偽物と判定したい}}$$

学習は，D が正しく判定できるように更新しつつ，G が D を欺く方向に更新することを交互に繰り返す．

GAN は学習速度が速く，高解像度な画像が生成できる一方で，生成器が限られたパターンしか生成しなくなる**モード崩壊 (mode collapse)** や，生成器と識別器のバランスが崩れると学習が進まなくなるという問題がある．

自己回帰モデル

自己回帰モデル (Autoregressive Models) は，データの系列的構造を利用し，系列データを一要素ずつ順に生成する方式の生成モデルである．時刻 t の出力は，それ以前に生成された要素に依存して決定される．この仕組みは次のような確率分解で表される．ここで，x_t は系列の t 番目の要素，T は系列の長さである．

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \,|\, x_1, x_2, \ldots, x_{t-1})$$

学習時には，既知の系列を用いて次の要素の条件付き確率を最大化するようにパラメータを更新する．生成時には，初期トークン（または開始シンボル）から順に次の要素をサンプリングしていく．高精度な系列生成が可能でモデルが出力の一貫性を保ちやすいが，並列生成が難しく，逐次的生成により計算時間が長くなる．

拡散モデル

拡散モデル (Diffusion Model) は，データに段階的にノイズを加える**順方向過程 (forward process)** と，ノイズからデータを復元する**逆方向過程 (reverse process)** を学習するモデルである．

- **順方向過程：**データ $\mathbf{x}_0$ に正規分布に従うノイズを段階的に加え，$\mathbf{x}_t$ まで変換する．これはマルコフ連鎖で表される．データ $\mathbf{x}_0$ に対して $t = 1$ から T ステップにわたり，次のように少しずつノイズを加える（β_t はステップ t におけるノイズ量を決定するパラメータ）．

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

- **逆方向過程 (Reverse Process)：**順方向過程で加えたノイズを取り除き，$\mathbf{x}_{t-1}$ を再構成する過程である．真の逆方向分布 $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ は未知のため，パラメータ θ をもつ分布 $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ で近似する．一般的に次のように正規分布として定式化される（μ_θ と σ_θ はそれぞれ，ニューラルネットワークによって推定される平均と標準偏差）．

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}\left(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_\theta(\mathbf{x}_t, t)\right)$$

拡散モデルは生成の安定性に優れているが，推論に多くのステップが必要で計算コストが高いという課題もある．

参考文献

[1] 大北剛,「深層学習」講義資料

[2] 宮野英次，江藤宏,「最適化」講義資料

[3] 株式会社アイデミー,「図解即戦力　機械学習＆ディープラーニングのしくみと技術がこれ1冊でしっかりわかる教科書」，技術評論社，2019

[4] CVML エキスパートガイド
https://cvml-expertguide.net/

[5] AIsmiley
https://aismiley.co.jp/