令和3年度 修士論文

機械学習を用いたアナログ回路の
トポロジーと素子値の自動設計

指導教員 髙井 伸和 准教授

群馬大学大学院 理工学府 理工学専攻
電子情報・数理教育プログラム
髙井研究室

T201D034
齋藤 彰寛

令和4年3月

目 次

第1章　研究背景・目的

1.1　研究背景

近年、IC 設計において開発コストの削減や開発期間の短縮、高集積化、アナログデジタルの混載化が不可欠となってきている。しかし複雑な回路設計や設計者の育成には時間と労力がかかってしまうという問題がある。この問題の解決法として CPU や ROM、RAM などのデジタル回路領域では HDL と論理合成技術による自動設計が実用化され、設計時間の大幅な短縮に成功している。一方 ADC や DAC、アンプなどのアナログ回路領域では特性にトレードオフの関係があり、様々なパラメータが一度に変わってしまうため設計の難易度が高く、実用化には至っていない。アナログ回路設計は設計者自身の知識や経験に基づいて修正を行うことで、要求仕様を満たす回路を試行錯誤しながら設計する必要がある。そのためデジタル回路に比べ、アナログ回路はアナログデジタル混載 LSI の開発期間の中で大きな割合を占めている。よって高性能な混載 LSI の設計に掛かるコストや時間の削減には、計算機支援による効率的なアナログ集積回路の設計手法確立が重要となる。

1.2　研究目的

一般的にアナログ回路設計の流れとしては、1. 仕様決定、2. 回路トポロジー決定、3. 素子値決定、4. レイアウト設計、5. 製品テストの順で行われる。この中でも回路トポロジーの選択は設計者の経験による部分が大きく、回路の知識もなければ適切な選択を行うことは困難である。この問題の解決方法として深層学習を用いて所望特性から回路トポロジーを選択するという手法 [1] がある。また素子値決定はあるトポロジーに対して各素子の素子値を決め、シミュレーションを行い、要求仕様を満たしているかを確認する必要がある。しかし一回で完了することは少なく、何度も出戻りが発生し、このサイクルを繰り返す必要がある。これがアナログ回路設計の時間がかかる要因になっている。解決策として MOS の I-V 特性の式を用いて解析を行い、素子値を算出する数式ベースの手法や SPICE のシミュレーション結果を利用した最適化アルゴリズムを用いた手法 [2, 3, 4, 5, 6] がある。これらは補助的に用いられているが、設計者の知識や経験に依存し、完全な自動化には至っていない。深層強化学習を用いた素子値探索の手法 [7] が提案されているが、そもそも所望特性を得にくいトポロジーで探索を行ってもその時間は無駄になってしまう。よって効率的な自動設計のためには、「適切なトポロジーに対して効率的に素子値探索を行う手法」が必要となる。

そこで本論文では演算増幅器の回路トポロジー決定と素子値決定に機械学習を用いて、同時に決定する自動設計手法を提案する。目標となる要求仕様を入力することで、計算機自身がそれを得られるトポロジー及び素子値を決定し、所望特性を実現することを目的とする。提案手法ではまず様々なトポロジーに対して素子値を変化させたときの利得やスルーレート、出力抵抗などの特性データを作成・収集する。次に収集したデータに処理を行った後、入力

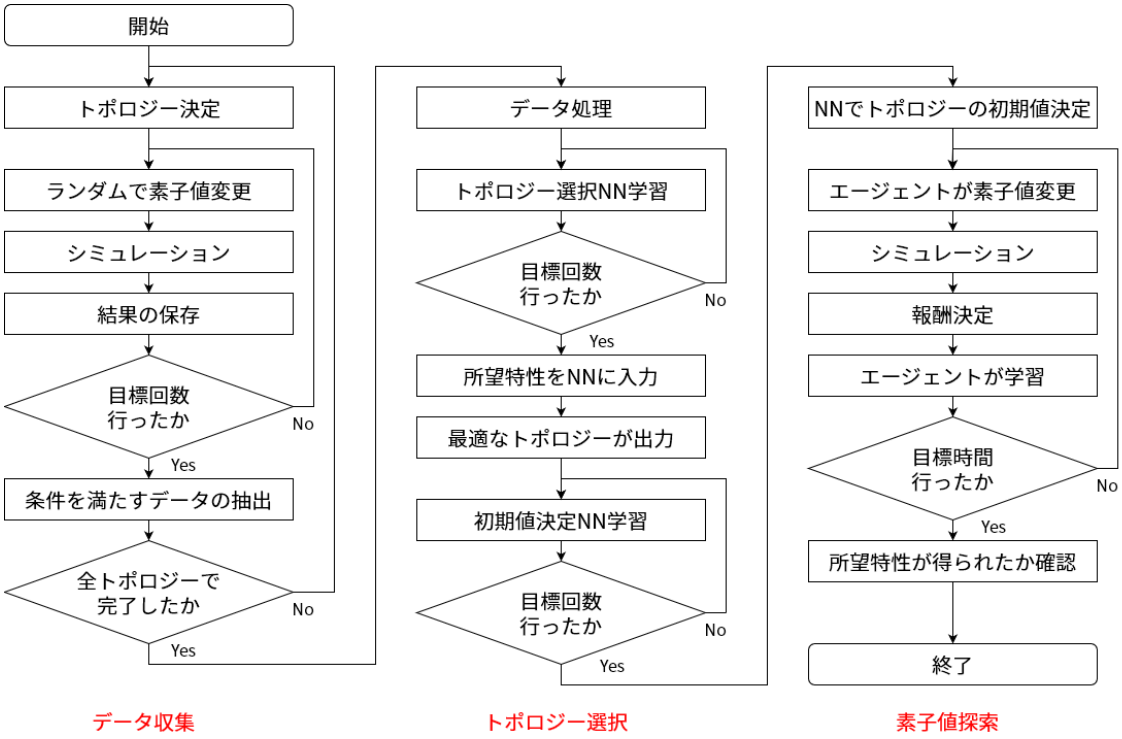

図 1.1: 提案手法のフローチャート

が回路特性、出力が回路トポロジーとしたニューラルネットワーク (NN: Neural Network) の学習をする。最後に所望特性を目標として深層強化学習の一つである Importance Weighted Actor-Learner Architecture (IMPALA) を用いて素子値を決定する。この際、素子値探索を行うトポロジー初期値を収集したデータを用いて学習した入力が回路特性、出力が素子値という NN で決定する。その後最終的に得られた特性と目標との比較を行う。提案手法により、設計者の知識・経験に依存せず、所望特性を決めることで計 24 種類の回路から最適なものを選択し、最終的に目標とシミュレーション結果の平均一致率が 90%以上の回路を自動設計できることを確認した。

1.3 提案手法の流れと本論文の構成

提案手法のフローチャートを図 1.1 に示す。本論文は全 6 章で構成される。第 2 章で学習データの収集について述べる。第 3 章で NN の概要と回路トポロジーの選択について述べる。第 4 章で深層強化学習の概要と素子値探索について述べる。第 5 章で提案手法のプログラムとシミュレーションの実行結果を示し、所望特性との比較を行う。最後に第 6 章で本研究のまとめと今後の課題を述べる。

第2章　学習データの収集

2.1　収集する特性と条件

第 3 章で用いる NN で構築したモデルの学習には大量の学習データが必要となる。学習データの量・質によって学習後のモデルの精度が大きく変化するため、目的に応じたデータの収集は NN を用いた機械学習において不可欠である。本研究では演算増幅器の特性とトポロジーとの関係性を NN で近似し、最適なトポロジーを選択する。そのため各トポロジーごとに素子値を振って、得られる特性をデータ化し保存する。収集を行う特性は消費電力 (PC: Power Consumption) [W]、直流利得 (DC Gain) [dB]、位相余裕 (PM: Phase Margin) [°]、利得帯域幅積 (GBW: Gain BandWidth product) [Hz]、スルーレート (SR: Slew Rate) [V/s]、全高調波歪 (THD: Total Harmonic Distortion) [%]、同相除去比 (CMRR: Common Mode Rejection Ratio) [dB]、電源電圧変動除去比 (PSRR: Power Supply Rejection Ratio) [dB]、出力電圧範囲 (OVR: Output Voltage Range) [%]、同相入力範囲 (CMIR: Common Mode Input Range) [%]、占有面積 (Area) [mm^2]、出力抵抗 (OR: Output Resistance) [Ω]、入力換算雑音 (IRN: Input Referred Noise) [V/Hz] の計 13 項目とする。動作しないまたは不安定な回路の特性データが混ざると NN の学習に悪影響を及ぼすため、基準として 2020 年演算増幅器設計コンテスト [8] の最低要件を満たすもののみを採用する。その条件を表 2.1、表 2.2 に示す。各特性の算出方法も同じく 2020 年演算増幅器設計コンテストの方式を用いるため、ここでは省略する。

また、シミュレーションの条件は以下のように設定する。

- 電源電圧は 3V 固定 (V_{DD}=1.5V、V_{SS}=-1.5V)
- PMOS のバルクは V_{DD}、NMOS のバルクは V_{SS} に接続
- シミュレータは HSPICE を使用
- プロセスパラメータは TSMC0.18um を使用
- 同一トポロジーでも MOS の L=2.0um と L=0.2um の 2 パターンで収集
- 素子値の変更はランダム
- MOS: W/L=3:1 とし Multiplier を 1〜32 の間で刻みで振る
- RES: 20kΩ〜500kΩ の間で 20kΩ 刻みで振る
- CAP: 0.2pF〜2pF の間で 0.2pF 刻みで振る
- 対となる MOS の素子値は同一とする

この条件で 1 種類あたり約 6 万回シミュレーションを行い、最低要件を満たす特性のみを抽出して CSV ファイルにまとめて学習データとする。

表 2.1: 各特性の最低要件

特性	最低要件
PC [W]	規定条件下 (表 2.2) にてバイアス電流の変動が 50%以下かつ 100mW 以下
DC Gain [dB]	40dB 以上
PM [°]	45 ° 以上
GBW [Hz]	1MHz 以上
SR [V/s]	立ち上がり立ち下がりともに絶対値が 0.1V / μs 以上
THD [%]	1% 以下
CMRR [dB]	40dB 以上
PSRR [dB]	V_{DD} 側、V_{SS} 側いずれか悪い側の 0.1Hz での値が 40dB 以上
OVR [%]	0V を中心とする出力電圧が正負電源電圧の 5% 以上
CMIR [%]	0V を中心とする同相入力が正負電源電圧の 5% 以上
Area [mm^2]	$1mm^2$ 以下
OR [Ω]	要件無し
IRN [V/Hz]	要件無し

表 2.2: 消費電流を計測する条件

	low	typ	high
温度	-40°C	25°C	80°C
電源電圧	typ - 10%	設計値	typ + 10%

2.2 登録トポロジー

収集を行うトポロジーは資料 [9, 10] を参考に以下の計 12 種類の特徴的な回路で行う。条件を統一するため、バイアス回路は全て同じもの [11] を用いる。

1. NMOS 入力基本差動対+ソース接地出力 (図 2.1)
2. PMOS 入力基本差動対+ソース接地出力 (図 2.2)
3. NMOS 入力基本差動対+AB 級出力 (図 2.3)
4. PMOS 入力基本差動対+AB 級出力 (図 2.4)
5. NMOS 入力フォールデッドカスコード差動対+ソース接地出力 (図 2.5)
6. PMOS 入力フォールデッドカスコード差動対+ソース接地出力 (図 2.6)
7. NMOS 入力フォールデッドカスコード差動対+AB 級出力 (図 2.7)
8. PMOS 入力フォールデッドカスコード差動対+AB 級出力 (図 2.8)
9. NMOS 入力フォールデッドカスコード差動対+スーパーソースフォロワ (図 2.9)
10. PMOS 入力フォールデッドカスコード差動対+スーパーソースフォロワ (図 2.10)

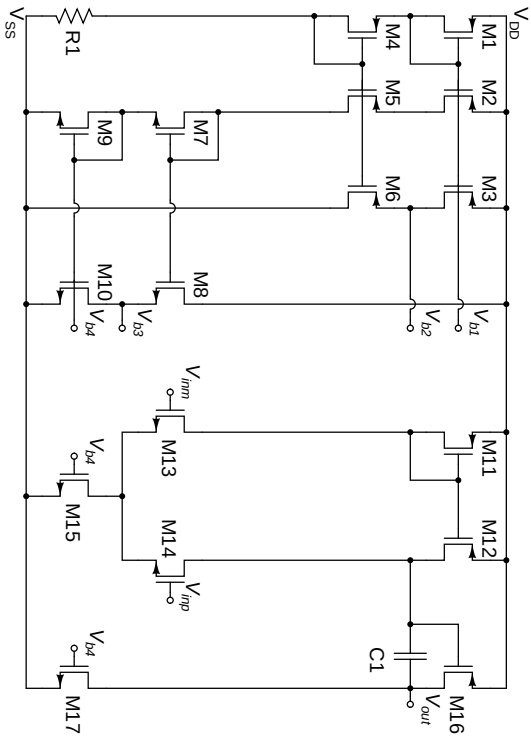

図 2.1: NMOS 入力基本差動対+ソース接地出力

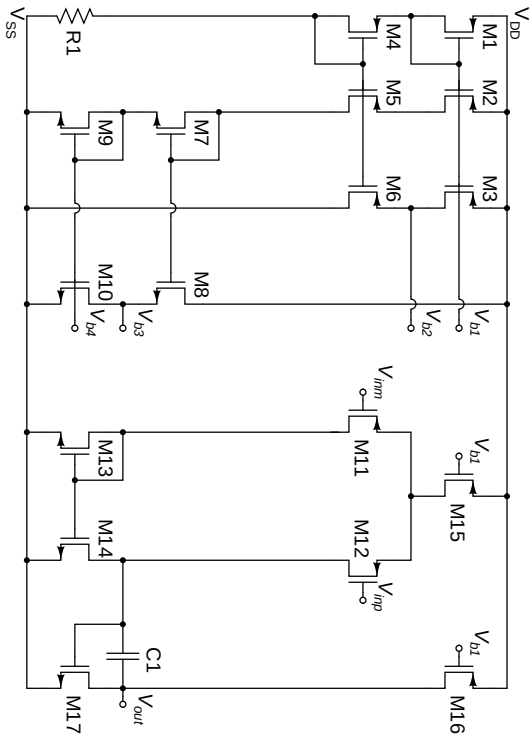

図 2.2: PMOS 入力基本差動対+ソース接地出力

11. Rail to Rail 基本差動対+AB 級出力 (図 2.11)

12. Rail to Rail フォールデッドカスコード差動対+AB 級出力 (図 2.12)

よって L=2.0um (W=6.0um) と L=0.2um (W=0.6um) の 2 パターンで収集するため、計 24 種類となる。以降「1 のトポロジーで L=0.2um のデータ→ 01-S」、「2 のトポロジーで L=2.0um のデータ→ 02-L」のように表現する。

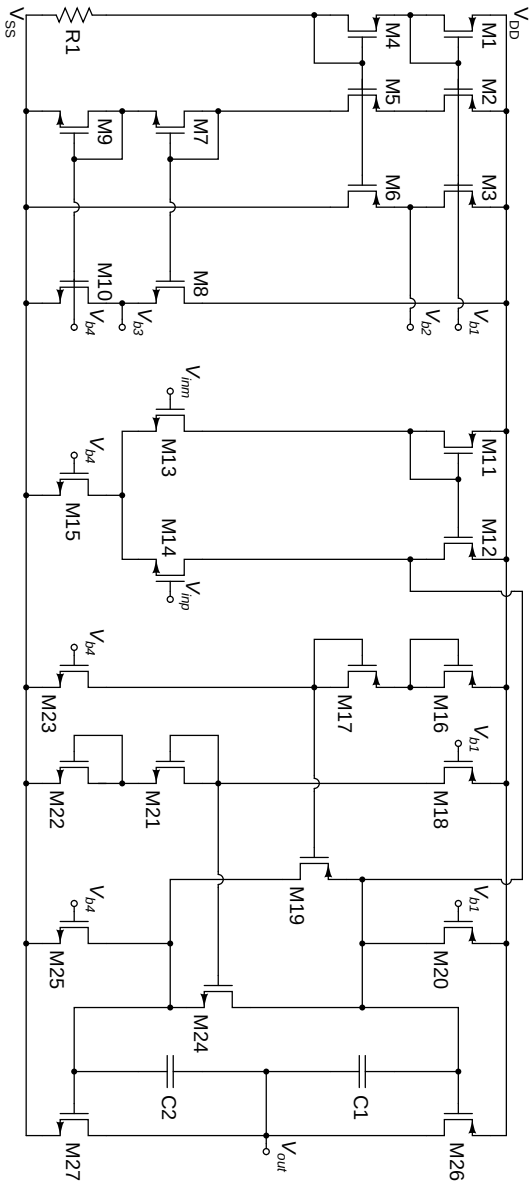

図 2.3: NMOS 入力基本差動対+AB 級出力

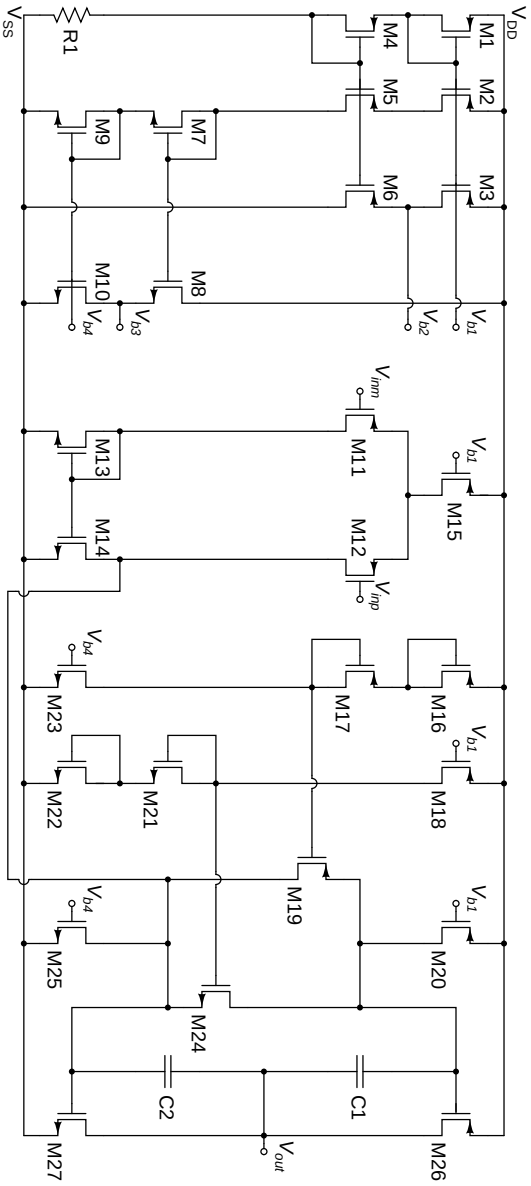

図 2.4: PMOS 入力基本差動対+AB 級出力

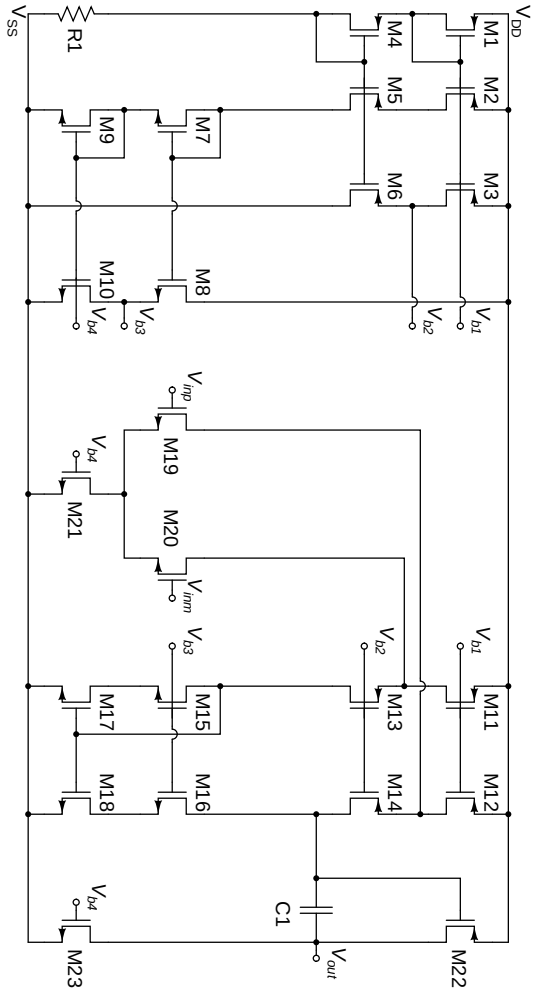

図 2.5: NMOS 入力フォールデッドカスコード差動対+ソース接地出力

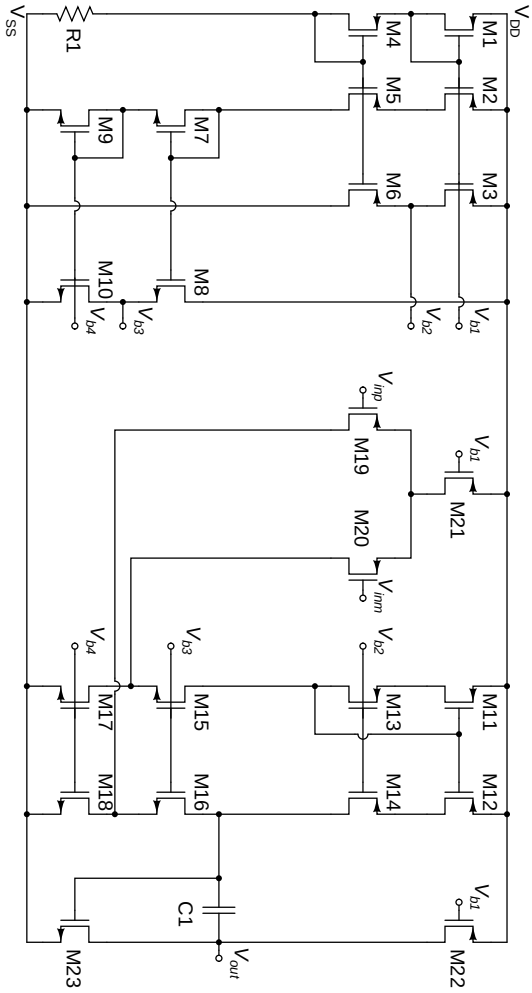

図 2.6: PMOS 入力フォールデッドカスコード差動対+ソース接地出力

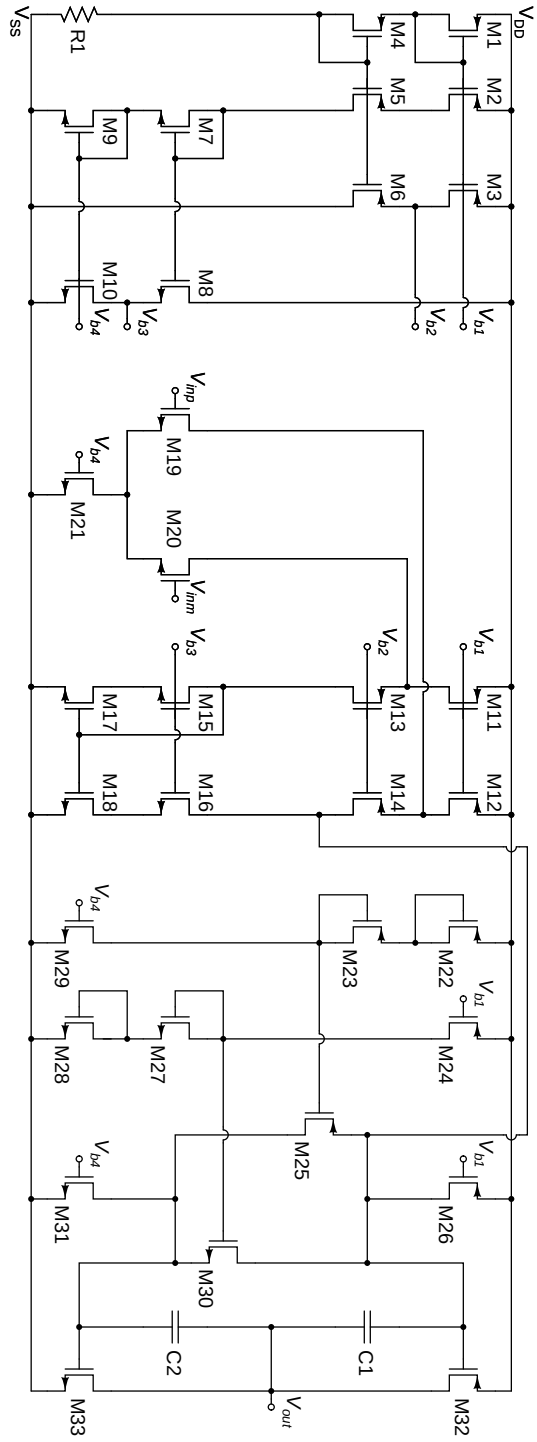

図 2.7: NMOS 入力フォールデッドカスコード差動対+AB 級出力

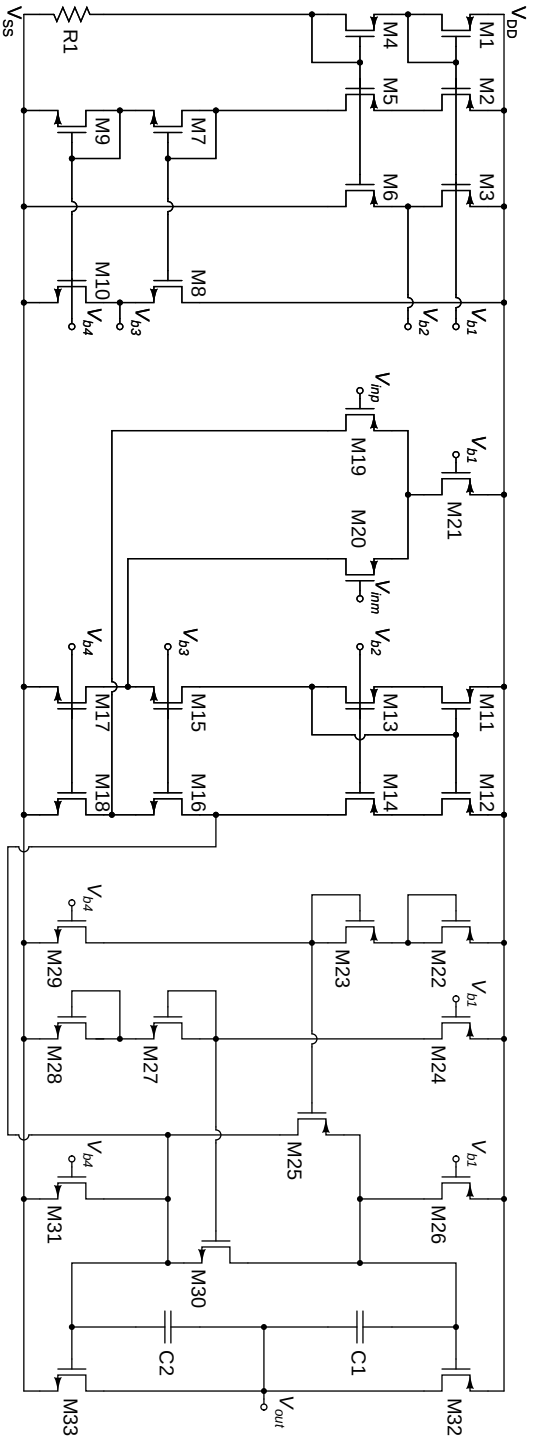

図 2.8: PMOS 入力フォールデッドカスコード差動対+AB 級出力

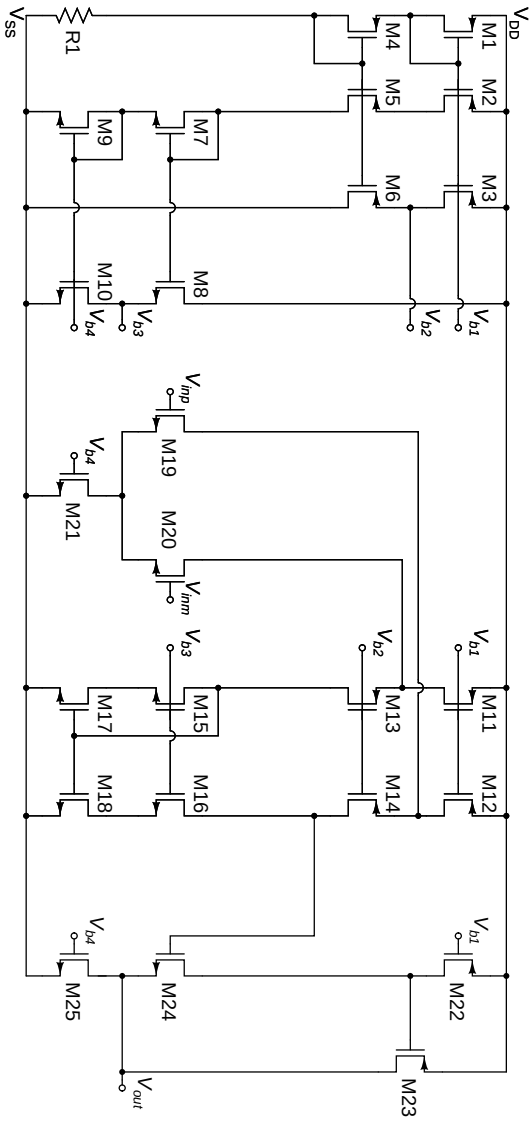

図 2.9: NMOS 入力フォールデッドカスコード差動対+スーパーソースフォロワ

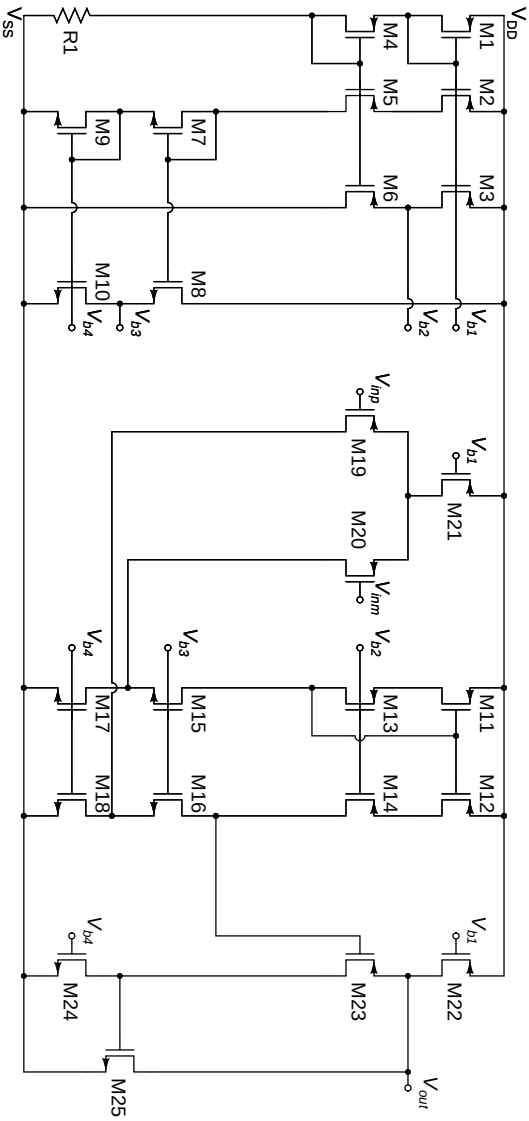

図 2.10: PMOS 入力フォールデッドカスコード差動対+スーパーソースフォロワ

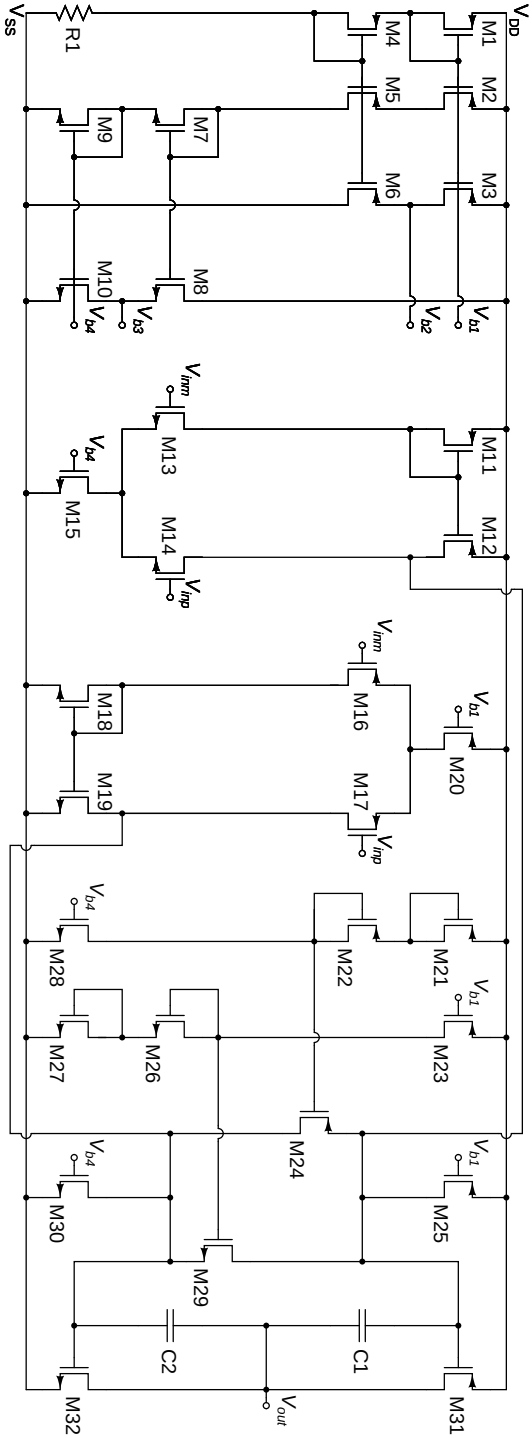

図 2.11: Rail to Rail 基本差動対+AB 級出力

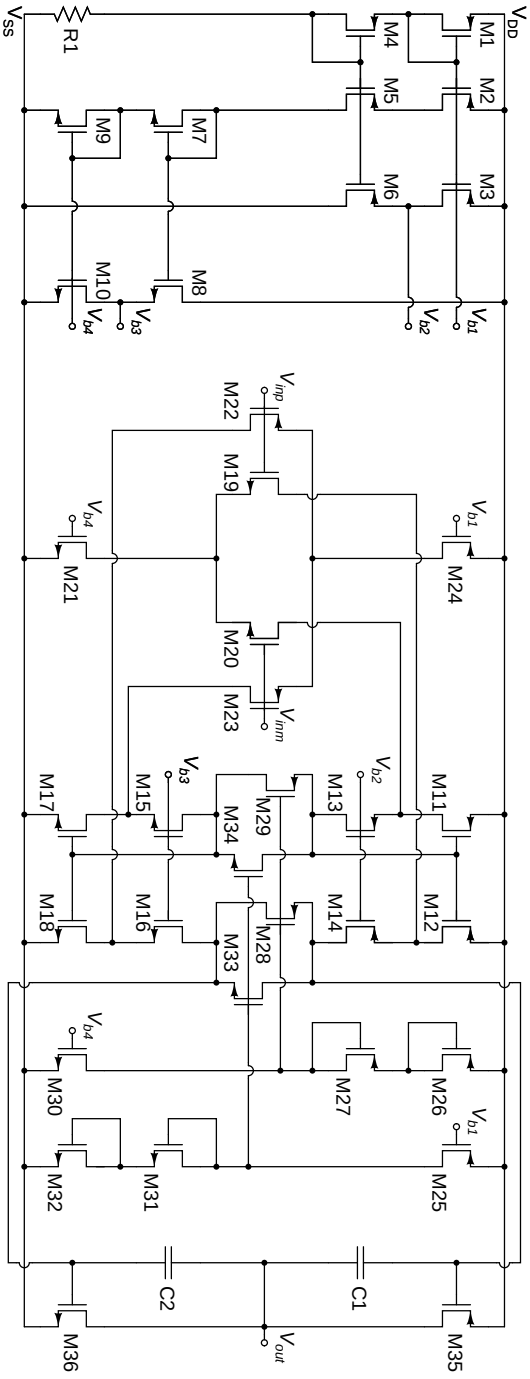

図 2.12: Rail to Rail フォールデッドカスコード差動対+AB 級出力

第3章　回路トポロジーの選択

3.1　Neural Network

3.1.1　パーセプトロンの概要

NN はパーセプトロンという数式的モデルの組み合わせによってなりたっている。まず図 3.1 にパーセプトロンの仕組みを示す。このとき信号の入力側のパーセプトロン群を入力層、出力側のパーセプトロン群を出力層と呼ぶ。b_0 はバイアス、x_1、x_2 は入力信号を表す。また w_1、w_2 は重みであり、この値を変化させることで各信号に対して重みづけを行う。そしてパーセプトロンに入力される総信号は u と表される。その後シグモイド関数や ReLU 関数などの活性化関数 f を u に適用し、最終的に出力信号として z が得られる。この重みを調節することで入出力の関係を関数近似することができる。

$$u = b_0 + x_1 w_1 + x_2 w_2 \tag{3.1}$$

$$z = f(u) \tag{3.2}$$

3.1.2　NN の概要

NN は脳の神経細胞 (ニューロン) とそのつながりをパーセプトロンの組み合わせによって表現し、より複雑な関数近似を実現する技術である。内容としては入力層と出力層の間に中間層 (隠れ層) を追加し、さらに 1 層あたりのパーセプトロン数 (ユニット数) を増やすことでネットワークの表現力を向上させている。特に隠れ層が多数存在するネットワークを用いた学習をディープラーニングと呼ぶ。例えば画像認識は入力層に画像の各ピクセルの色情報、出力層にラベル (どんな画像かを表したもの) としてパラメータを学習させることで実現される。

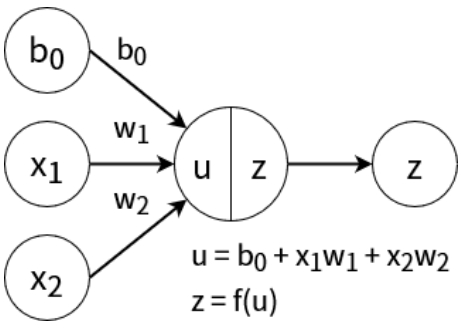

図 3.1: パーセプトロンの仕組み

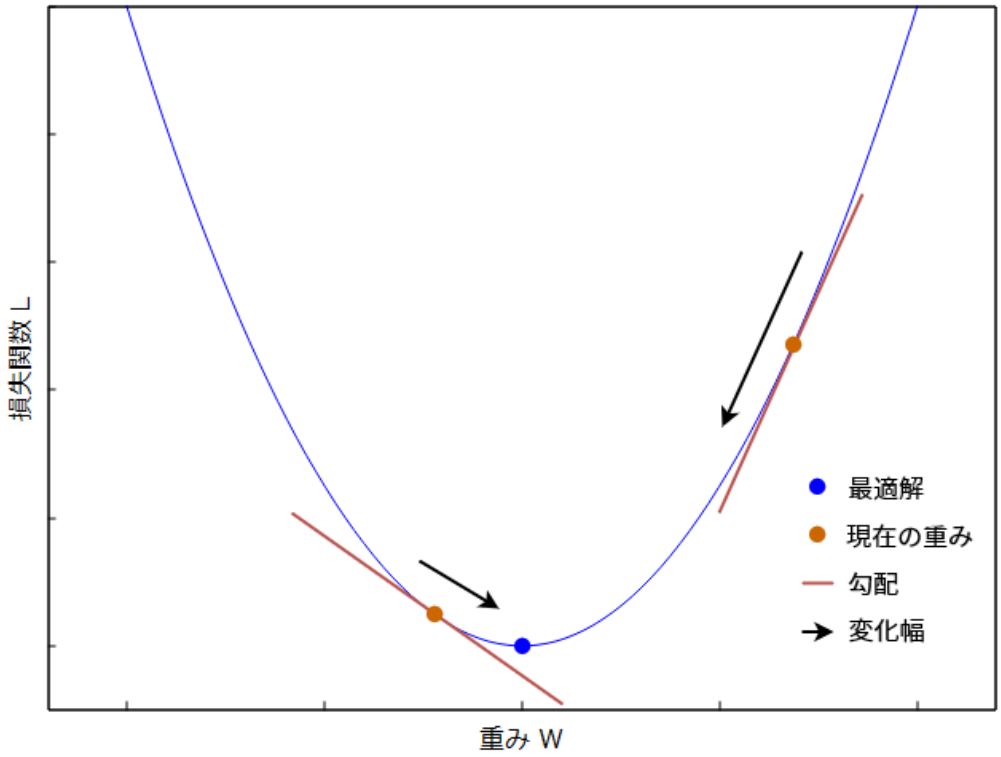

図 3.2: 勾配降下法の概略図

NN への入力を x_n、出力を y_n とし、入力 x_n に対する正解を d_n とする。NN の学習とはどんな入力データに対しても $y_n = d_n$ に近づけるため、重み W を適切な値に調節することである。このときに用いられる手法が勾配降下法である。その概略図を図 3.2 に示す。y_n と d_n がどれだけ近いかを誤差関数 L とすると、勾配は、

$$\Delta L = \frac{\partial E}{\partial W} \tag{3.3}$$

で表される。基本理論は勾配が正の場合は重み W を小さくする方向に更新し、負の場合は重み W を大きくする方向に更新することで誤差関数 L が最小となる点を探索する。この方式は誤差逆伝搬法 (バックプロパゲーション) と呼ばれ、現在 NN において主流の学習法である。しかし単純に更新を行うだけでは局所解に陥る場合や、無駄な探索を行ってしまい収束が遅くなってしまう場合がある。その解決策としてより発展させたアルゴリズムとして RMSProp や Adam が NN の学習に用いられることが多い。

3.2 学習データの処理

収集したデータは特性ごとに数値のスケールが異なっており、そのままでは NN の学習に悪影響がある。よって数値のスケーリング (正規化) を行うために、それぞれの特性に対して以下の式を適用する。

$$x_i' = \frac{x_i}{|x_{\max}|} \tag{3.4}$$

x_{max} はその特性の最大値、x_i は正規化前の値、x_i' は正規化後の値を表している。これにより全ての値は 0 より大きく 1 以下の値に正規化される。

また特性データにトポロジーの情報を One-Hot ベクトル形式のラベルで付与しておく。One-Hot ベクトルは $(0, 1, 0, 0)$ のように 1 つの成分が 1 で残りの成分が全て 0 であるベクトルである。単純に自然数でラベル付けを行うよりも数値の大きさという要素を排除できるため、学習精度の向上につながる。今回の場合は 01-L なら $(1, 0, 0, \cdots, 0, 0)$、01-S なら $(0, 1, 0, \cdots, 0, 0)$、12-S なら $(0, 0, 0, \cdots, 0, 1)$ のようにラベル付けを行う。

全てのデータを学習に使用してしまうと、そのモデルの評価が適切に行えなくなってしまう。そのため全データの 80%をモデルの学習に用いる訓練データ、20%を学習には用いずモデルの確認・評価に用いるテストデータとする。その選出はランダムで行う。加えて今回のデータは時系列データではなく連続性は存在しないため、データの順序をシャッフルし、順番による影響を排除する。

3.3 使用したトポロジー選択 NN の条件

3.3.1 トポロジー選択への応用

次に 13 項目の回路特性を入力、24 種類の回路トポロジーを出力とした隠れ層が 3 層の NN を構築する。これによりそれぞれの相関関係を学習し、目標特性を得るために最適なトポロジーを選択する。実装には言語は Python を使用し、ライブラリとして TensorFlow と TensorFlow Addons を用いた。その構造を図 3.3 に示す。加えて学習の際の設定を以下に示す。

- ミニバッチサイズ: 2048
- 学習回数: 2000Epoch
- 損失関数: Cross entropy loss
- 最適化アルゴリズム: Ranger

入力層のユニット数は収集した特性数、出力層のユニット数は登録する回路数と同じである。以下の節 3.3.3〜節 3.3.8 で用いた手法の詳細について述べる。

3.3.2 トポロジーの初期値決定への応用

トポロジーの初期値決定に用いる NN は図 3.3 とほぼ同じものを使用が、変更点は以下の通りである。

- 出力: 回路トポロジー → 素子値
- NN 構造: 最後に Softmax 関数を適用 → 最後に活性化関数無し
- 損失関数: Cross entropy loss → Mean Absolute error

またこちらで用いる学習データは全てのトポロジーのものでは無く、選択されたトポロジーのデータのみを用いる。

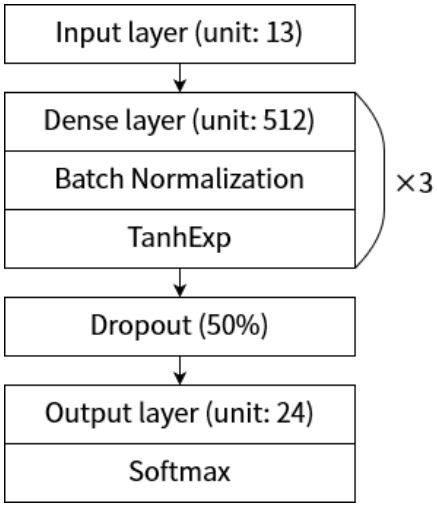

図 3.3: トポロジー選択に用いるニューラルネットワークの構造

3.3.3 ミニバッチ学習と Epoch

ミニバッチ学習は全データを小さなグループに分割してモデルの学習を行う手法である。また、分割を行わず一度に全てのデータで学習を行う手法をバッチ学習、1 データづつ抜き出して学習を行う手法をオンライン学習と呼ぶ。NN の学習においてはデータ量が膨大でバッチ学習では必要なメモリが大きくなってしまう。しかし、オンライン学習では 1 データごとに学習を行うため速度が遅く、学習が安定しにくい。そのため多くの場合、ミニバッチ学習が用いられる。

例としてデータ数 10000 個のデータセットをミニバッチサイズ 1000 個で１０個に分割したとする。このとき、ミニバッチ 1 個を使用した学習を 1Iteration、ミニバッチ 10 個全てを使用した学習を 1Epoch と呼ぶ。

3.3.4 Batch Normalization

NN において隠れ層を増やしていくと、Internal Convariate Shift と呼ばれる問題が発生する。これはミニバッチごとに入力される特徴量の分布が変化し、学習を不安定・低速にしてしまう。その解決策として Batch Normalization[13] が用いられる。これはミニバッチごとにデータを平均 0、分散 1 に正規化する手法である。Batch Normalization で適用する式は以下の通りである。

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{3.5}$$

$$y_i = \gamma \hat{x}_i + \beta \tag{3.6}$$

x_i は正規化前のデータ、$\hat{x}_i$ は正規化後のデータ、μ_B はミニバッチの平均、σ_B^2 はミニバッチの分散、ϵ は安定化のための定数、y_i は最終的に出力されるデータ、γ と β は学習によって決定される新しい平均と分散を表している。Batch Normalization により隠れ層を増やせるだけでなく、学習の安定化・効率化にも効果がある。

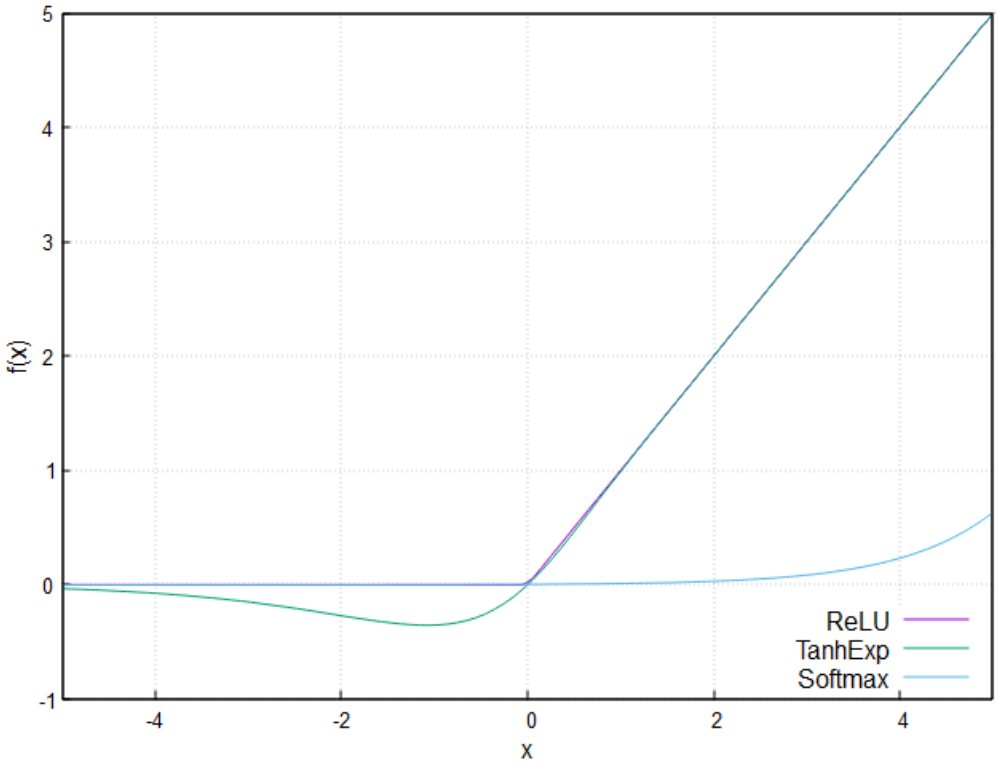

図 3.4: 活性化関数の比較

3.3.5 TanhExp と Softmax

図 3.4 に ReLU、TanhExp、Softmax の比較を行ったグラフを示す。グラフの横軸が入力 x で縦軸が活性化関数を通した後の出力 $f(x)$ を表している。

TanhExp 関数は以下の式で定義される。

$$f(x) = x \tanh(e^x) \tag{3.7}$$

同じく NN の隠れ層で用いられる活性化関数である ReLU は入力が負の時、微分値 (勾配) が 0 となるため、学習が進まないという問題があった。そこで TanhExp を用いることで計算コストの増大を抑えつつ、学習を精度を向上させることができる [14]。

Softmax 関数は以下の式で定義される。

$$f(x_i) = \frac{x^{x_i}}{\sum_{k=1}^{n} e^{x_k}} \tag{3.8}$$

Softmax 関数は多クラスの分類問題の出力層に用いられることが多い。その理由として出力値の合計が 1 となるという特徴があり、そのまま結果を確率とみなして扱うことができるためである。

3.3.6 Dropout

Dropout[15] は NN で用いられる手法一つで、学習時に一定割合のニューロンを不活性化させながら学習を行う。機会学習の分野にはアンサンブル学習という複数のモデルを組み合わせる学習方法があり、汎化性能の向上が見込めることが知られている。Dropout は学習を行う際にランダムにニューロンを不活性化することで毎回異なる NN で学習しているとみなすことができる。よって Dropout を適用することで複数の NN を用意することなく汎化性能の向上に繋がる。

3.3.7 Cross entropy loss

Cross entropy loss (交差エントロピー誤差) は以下の式で定義される。

$$L = -\sum_k t_k \log y_k \tag{3.9}$$

L は損失、k はクラスの数、t_k は正解ラベル、y_k は予測ラベルを表す。このときラベルは One-Hot ベクトル形式である。正解と予測が正しいほど損失は 0 に近づき、異なるほど損失は大きくなる。例として $k = 3$ で正解ラベルが $(0, 1, 0)$、予測ラベルが $(0.2, 0.7, 0.1)$ の場合の Cross Entropy Loss は以下のようになる。

$$\begin{aligned} L &= -(0 \times \log 0.2 + 1 \times \log 0.7 + 0 \times \log 0.1) \\ &= 0.1549 \cdots \end{aligned}$$

3.3.8 Ranger

Ranger は RAdam[16] と Lookahead[17] を組み合わせた最適化アルゴリズムである。

RAdam は Rectified Adam の略で、Adam[18] をベースに発展させた最適化アルゴリズムである。Adam ではより良い結果を得るために、人間が場合に応じて学習率を試行錯誤しながら調節する必要があった。そこで RAdam では Warm start 呼ばれる学習初期には小さい学習率で始め、その後徐々に通常の学習率まで大きくしていく手法を用いている。これによりハイパーパラメータの調節が不要となり、Adam の問題だった適応学習率の分散の増大を抑制している。また、Adam 自体の計算式にも補正項を追加し、同様に適応学習率の分散を抑えている。

Lookahead は NN の重みを通常の最適化アルゴリズムで更新する fast weights だけでなく、数ステップごとに fast weights を用いて更新する slow weights も用いる手法である。つまり、数ステップ通常の最適化アルゴリズムで重みを更新し、その後更新前と更新後の間の重みに戻って更新する。これにより収束速度・汎化性能向上が見込める。fast weights は k 回以下の式で更新される。

$$\theta_{t,i+1} = \phi_{t,i} + A(L, \theta_{t,i-1}, d) \tag{3.10}$$

θ は fast weights、A は最適化アルゴリズム、L は損失関数、d はミニバッチデータの分散に基づいて決定されるパラメータを表している。slow weights は k 回に 1 回以下の式で更新される。

$$\phi_{t+1} = \phi_t + \alpha(\theta_{t,k} - \phi_t) \tag{3.11}$$

ϕ は slow weights、α はステップサイズ (ハイパーパラメータ) を表している。

第4章　素子値探索

4.1　深層強化学習

4.1.1　強化学習の概要

強化学習は機械学習の一分野で、システム自身が試行錯誤しながら行動を自律的に最適化していく手法である。NN のような教師あり学習と異なり、明確な答えが無い問題に対して有効である。その概要を図 4.1 に示す。強化学習は意思決定を行う「エージェント」と制御対象となる「環境」によって成り立っている。基本的な学習の流れは以下の通りである。

1. エージェントが時刻 t で環境の状態 s に応じて行動 a を選択・実行する
2. 行動 a によって更新された状態 s と報酬 r をエージェントにフィードバックする
3. 環境からのフィードバックを元に方策 π を修正する
4. 時刻 t を $t+1$ に進める

この 1〜4 の流れを繰り返し行うことで、エージェントは報酬を最大化する方策 π を学習する。このときのモデルはマルコフ決定過程 (MDP: Markov Decision Process) に従う。強化学習には大きく分けて価値反復法と方策反復法に分けられる。価値反復法の例としては Q-Learning や SARSA、方策反復法の例としては方策勾配法や REINFORCE がある。

4.1.2　深層強化学習の概要

深層強化学習は強化学習と NN を組み合わせたものである。強化学習ではとりうる状態や行動が増加するとその組み合わせが膨大になり、現実的に表現できる限界を超えてしまうという問題があった (将棋やビデオゲームなど)。この解決策としてその保存場所に NN を用いて、近似を行ったのが深層強化学習である。例として強化学習の一つである Q-Learning では行動価値 (Q 値) を Q-Table と呼ばれる 状態数 $a \times$ 行動数 s の領域に貯蓄・保存していたが、Q-Table を NN で置き換えたのが Deep-Q-Network (DQN) である。

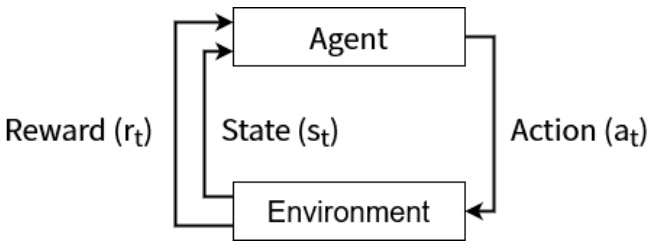

図 4.1: 強化学習の概要

4.2 使用した素子値探索の条件

4.2.1 素子値決定への応用

一般的な強化学習の形式と同様に、ネットリストからシミュレーションで特性を計算し、そこから報酬 r を出力する「環境」と、現在の状態 s と出力された報酬 r によって行動 a (素子値の変更) を選択する「エージェント」によって成り立っている。状態は離散的な素子値とし、そのとりうる値は第 2 章のシミュレーション条件と同様とする。行動は 1 素子に対して大きくするか小さくするかの 2 通りで、1 ステップでとれる行動数は素子数× 2 となる。例として M=10 の素子 A、B、C の 3 つからなる回路があった場合、とれる行動は 1. 素子 A を M=11 にする、2. 素子 A を M=9 にする、3. 素子 B を M=11 にする、4. 素子 B を M=9 にする、5. 素子 C を M=11 にする、6. 素子 C を M=9 にする、の 6 通りの中から 1 つ選択される。報酬はそれぞれの特性に対して以下の式で計算し、目標値とシミュレーション値が近ければ近いほど高評価となるようにする。

$$r = \begin{cases} \displaystyle\sum_{i=1}^{n} \frac{s_i}{t_i} & (t_i > s_i) \\ \displaystyle\sum_{i=1}^{n} \frac{t_i}{s_i} & (t_i \leq s_i) \end{cases} \tag{4.1}$$

r は報酬、t は目標の特性、s はシミュレーションで得られた特性、n は特性数を表している。つまり考慮する特性は 13 項目あるため、報酬は最高で 13 点となる。また最低要件を満たさない特性が含まれている場合は、報酬を 0 とする。

アルゴリズムは深層強化学習の手法の一つであり A3C[20] の発展形である IMPALA[21] を用いる。Actor 数は 4 つ、ステップ数は 100 とし、所定の時間まで実行し続ける。実装には言語は Python を使用し、環境は OpenAI Gym、エージェントは RLlib を使用して実装した。アルゴリズムのハイパーパラメータや NN の構造は RLlib で設定されているデフォルトのものを使用している。A3C と IMPALA についてはそれぞれ節 4.2.2 と節 4.2.3 で詳細を述べる。

4.2.2 A3C

Asynchhronous Advantage Actor-Critic (A3C) は深層強化学習の手法の一つである。以下でその特徴である Actor-Critic、Advantage、Asynchronous について順に手法の詳細を述べる。

Actor-Critic

Actor-Critic は状態 s に対して各行動を選択する確率 $\pi(s, a)$ を出力する Actor と、その時の価関数 $V(s)$ を出力する Critic により構成される。これら 2 つを同時に学習する価値反復法と方策反復法を組み合わせた学習手法である。Actor-Critic の動作の流れは以下の通りである。

1. Actor が方策 π をもとに行動 a_t を選択・実行する。
2. 環境が報酬 r を計算し、次の状態 s_{t+1} に遷移する。

3. 得られた状態 s_{t+1} と報酬 r を用いて Critic が価値関数 $V(s_t)$ と $V(s_{t+1})$ を推定し、以下の式で TD (Temporal Difference) 誤差 δ を算出する。

$$\delta = r + \gamma V(s_{t+1}) - V(s_t) \tag{4.2}$$

γ は割引率を表し、$0 \leq \alpha \leq 1$ で設定する。

4. TD 誤差を用いて Actor の方策 π を更新する。

   - $\gamma > 0$ の場合: 実行した行動 a_t の選択確率を小さくする。
   - $\gamma = 0$ の場合: 変化なし。
   - $\gamma < 0$ の場合: 実行した行動 a_t の選択確率を大きくする。

5. TD 誤差を用いて Critic における価値関数 $V(s)$ を以下の式で更新する。

$$V(s_t) \leftarrow V(s_t) + \alpha\gamma \tag{4.3}$$

α は学習率を表し、$0 < \alpha \leq 1$ で設定する。

A3C では Actor-Critic は 1 つの NN を用いて出力を分割して表される。メリットとして報酬の揺らぎの影響を受けにくく、学習の安定化・高速化が可能であることが挙げられる。

Advantage

通常の Actor-Critic では TD 誤差 δ で 1 ステップ先の報酬を用いて状態価値 $V(s)$ を更新していたが、この方法だと収束まで時間がかかってしまうという問題があった。その解決策として Advantage は数ステップ先の報酬 r を考慮して NN の更新を行う。A3C での Advantage は以下の式で表される。

$$Advantage = \sum_{i=0}^{k-i} \gamma^i r_{t+1} + \gamma^k V(s_{t+k}) - V(s_t) \tag{4.4}$$

k は用いるステップ数、r_t は報酬、$V(s)$ は状態価値の推定値、γ は割引率で $0 \leq \alpha \leq 1$ で設定する。これにより 1 ステップ先のみを考慮する場合に比べ、早く学習が進むというメリットがある。また、Advantage と Actor-Critic を組み合わせ、Asynchronous ではないアルゴリズムを Advantage Actor-Critic (A2C) と呼ぶ。

Asynchronous

Asynchronous は複数のエージェントと環境を用意し、それぞれを非同期で分散学習を行うことを表している。その概要を図 4.2 に示す。A3C では環境とエージェントによって成り立つ Worker と各 Worker の学習データを共有する Global Network によって構成される。動作の流れは以下の通りである。

1. Worker がその環境で経験を貯蓄する。
2. 経験をもとにより多くの報酬が得られるように NN の重みを更新する勾配を計算する。
3. 計算した勾配を Global Network に送信・更新する。

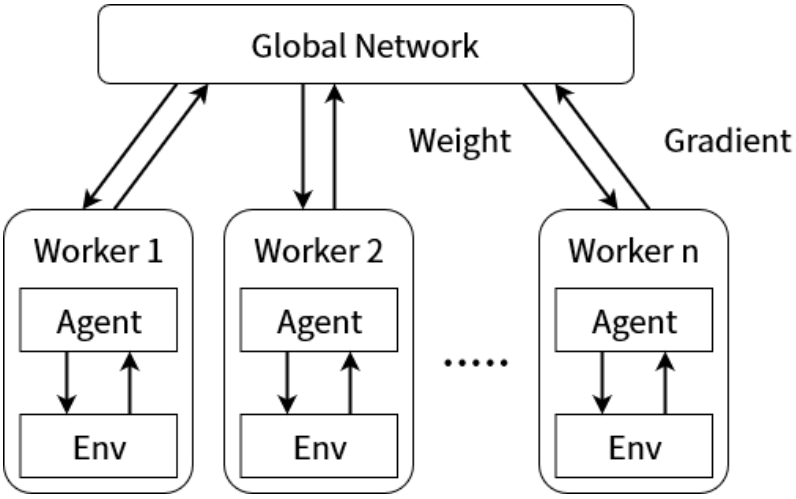

図 4.2: A3C の Asynchronous の概要

4. 更新された Global Network の重みを Worker に送信する。

この 1〜4 の流れを繰り返し、Global Network を強化していく。これにより並列分散で学習が行えるため高速化が可能、複数の Worker によって様々なパターンの学習が行えることによりデータの相関が減り、学習が安定するというメリットがある。

4.2.3 IMPALA

Importance Weighted Actor-Learner Architecture (IMPALA) は節 4.2.2 で述べた A3C を発展させたアルゴリズムある。他のアルゴリズムと比較しデータ効率が高いことが特徴である。演算増幅器のシミュレーションには 1 回あたり 10 秒弱の時間をかかり、試行回数を減らすため採用した。

基本的な学習の概要は図 4.2 で示した A3C と同じだが、IMPALA では Worker が Actor、Global Network が Learner と名前が変化する。また、A3C では Worker → Global Network では NN の勾配、Global Netowork → Worker では NN の重みを渡していたが、IMPALA では Actor → Learner では環境情報 (状態、行動、報酬)、Learner → Actor にパラメータ (方策、状態価値の推定値) を渡す。 これにより経験を生成するプロセスと方策 π と状態価値 $V(s)$ の学習のプロセスが切り離され、GPU を用いた高速な学習が行いやすくなる。更に環境情報のデータサイズ < 勾配情報のデータサイズであるため、Actor-Learner 間の通信性能が向上する。しかし、この構成により経験を貯蓄・保存する時の方策 μ と学習を行っている方策 π が必ずしも一致しないという問題が発生する。そこで IMPALA では V-trace という手法を用いている。V-trace については以下で詳細を述べる。

V-trace

V-trace は経験の貯蓄・保存時の方策 μ と学習時の方策 π 間のズレを補正するアルゴリズムである。n ステップの V-trace ターゲット v_s を以下の式で定義する。

$$v_s \stackrel{\text{def}}{=} V(x_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s} \left(\prod_{i=s}^{t-1} c_i \right) \delta_t V \tag{4.5}$$

この時

$$\delta_t V \stackrel{\text{def}}{=} \rho_t \left(r_t + \gamma V(x_{t+1}) - V(x_t) \right) \tag{4.6}$$

は重点サンプリングによって重みづけされた TD 誤差を表している。また、

$$\rho_t \stackrel{\text{def}}{=} \min \left(\bar{\rho}, \frac{\pi(a_t \mid x_t)}{\mu(a_t \mid x_t)} \right) \tag{4.7}$$

$$c_i \stackrel{\text{def}}{=} \min \left(\bar{c}, \frac{\pi(a_i \mid x_i)}{\mu(a_i \mid x_i)} \right) \tag{4.8}$$

は切り捨てされた重点サンプリングの重み係数を表し、$\bar{\rho} \geq \bar{c}$ を満たす。$\bar{\rho}$ は v_s における収束値に影響し、$\bar{c}$ はある TD 誤差が過去の時刻にどの程度影響を与えるかに影響している。

V-trace Actor-Critic

V-tarce の考えを Actor-Critic に応用したのが V-tace Actor-Critic である。通常の Actor-Critic では更新に TD 誤差を用いていたが、こちらでは上の V-trace ターゲット v_s を用いて算出する。方策 π_ω をパラメータ ω で、価値関数 V_θ をパラメータ θ で関数近似する。学習時間 s で θ は勾配降下法によって更新され、その勾配は以下の式で表される。このときパラメータ θ の勾配は以下の式で計算される。

$$\left(v_s - V_\theta(x_s) \right) \nabla_\theta V_\theta(x_s) \tag{4.9}$$

そしてパラメータ ω の勾配は以下の式で計算される。

$$\rho_s \nabla_\omega \log \pi_\omega(a_s \mid x_s) \left(r_s + \gamma v_{s+1} - V_\theta(x_s) \right) \tag{4.10}$$

更に局所解に収束してしまうのを防ぐためにエントロピーボーナスを追加し、以下の式で表される。

$$-\nabla_\omega \sum_a \pi_\omega(a \mid x_s) \log \pi_\omega(a \mid x_s) \tag{4.11}$$

IMPALA ではハイパーパラメーターである係数に従い、上記の 3 つの勾配を合計した値をもとに行われる。

第5章　実行結果

5.1　学習データ収集とNN学習

プログラムは「CPU: Intel Xeon Silver 4210R × 2、Memory: 64GB、GPU: なし」というマシンで実行した。まず学習データの収集の結果として、最低要件を満たす特性・素子値のデータを24種類の合計で26万2305個収集した。所要時間は1回路 (2種類) あたり約20時間で合計で約240時間であった。これは並列数を増やすことで時間短縮が可能である。種類ごとの個数の内訳は以下の通りである。

- 01-L: 10406個
- 01-S: 18633個
- 02-L: 15847個
- 02-S: 18599個
- 03-L: 7504個
- 03-S: 10719個
- 04-L: 10998個
- 04-S: 13524個
- 05-L: 7994個
- 05-S: 6040個
- 06-L: 7504個
- 06-S: 10719個
- 07-L: 1374個
- 07-S: 1599個
- 08-L: 2221個
- 08-S: 2417個
- 09-L: 3164個
- 09-S: 3206個
- 10-L: 3677個
- 10-S: 2892個
- 11-L: 6958個
- 11-S: 29590個
- 12-L: 23450個
- 12-S: 43170個

また収集したデータがそのトポロジーの特性を引き出せているかを確認する。そのため01-Sと05-Sの利得の比較を図5.1に、05-Sと09-Sの出力抵抗の比較を図5.2に示す。図5.1について、グラフの横軸はデータの個数を表し、縦軸は利得を表している。このグラフより差動対がフォールデッドカスコードになっている05-Sの方が、全体的に利得が大きくなっていることが確認できる。また図5.2について、グラフの横軸は同じくデータ個数を表し、縦軸は対数スケールで出力抵抗を表している。このグラフより出力回路がスーパーソースフォロワになっている09-Sの方が、出力抵抗が大幅に小さくなっていることが確認できる。

次に収集したデータを用いてNNを学習させる。学習の所要時間はデータ処理を含めて約1時間であった。こちらもGPUを使用できる環境であれば大幅な短縮が可能である。NNの学習結果をEpoch数を横軸、正解率を左縦軸、損失を右縦軸としたグラフを図5.3に示す。正解率は予測と正解が一致した割合を表し、損失は予測と正解のズレの大きさを表している。つまり正解率は高い程、損失は小さい程良いモデルと言える。24種類の回路トポロジーを特性から98.3%という高い精度で分類できていることが確認できた。

5.2　素子値探索の結果

学習済みNNに目標となる所望特性を入力してトポロジーの選択を行い、そのトポロジーに対して素子値探索を行う。その際の初期値にはNNで決定したものを用いる。検証のため

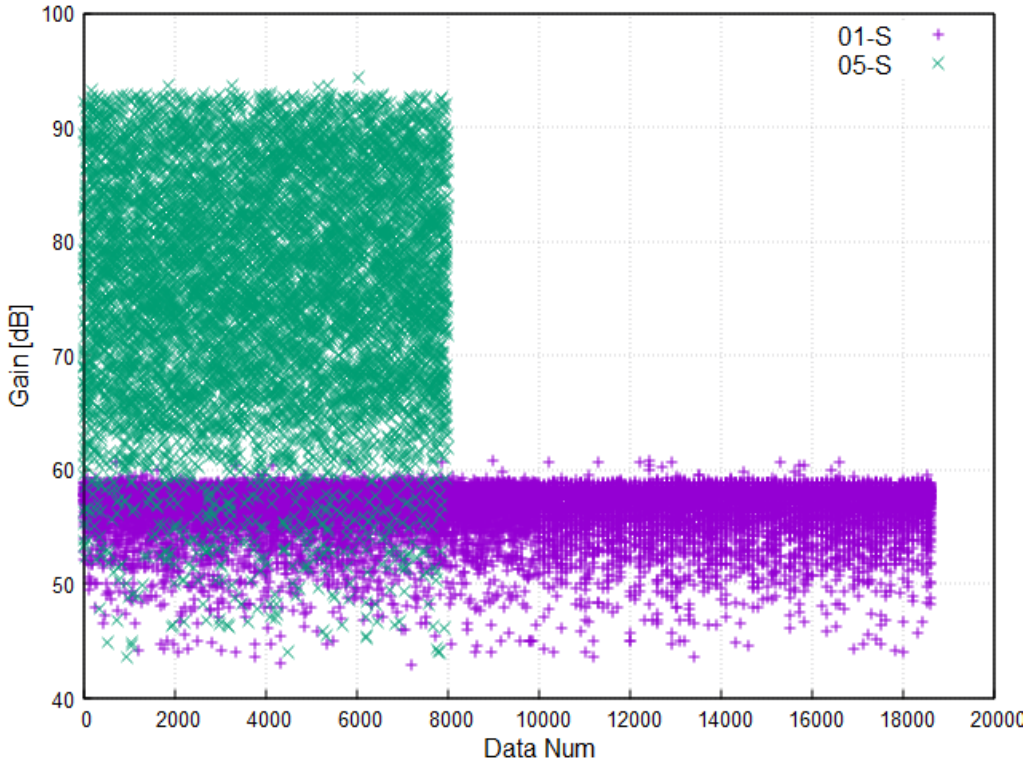

図 5.1: 01-S と 05-S の利得の比較

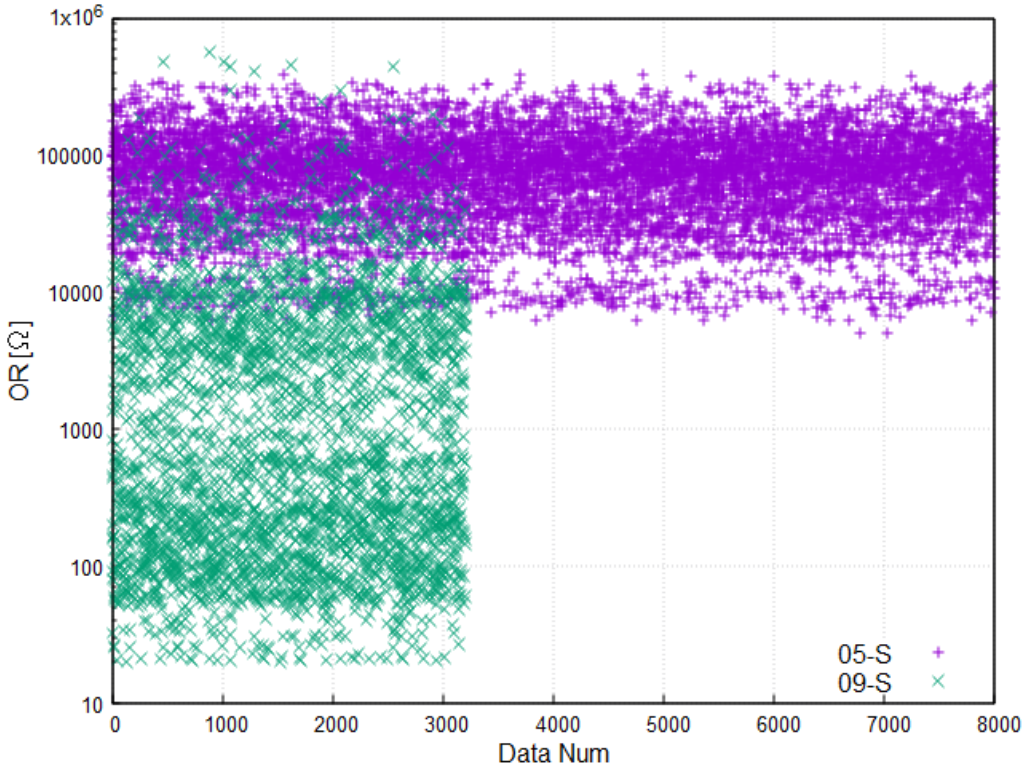

図 5.2: 05-S と 09-S の出力抵抗の比較

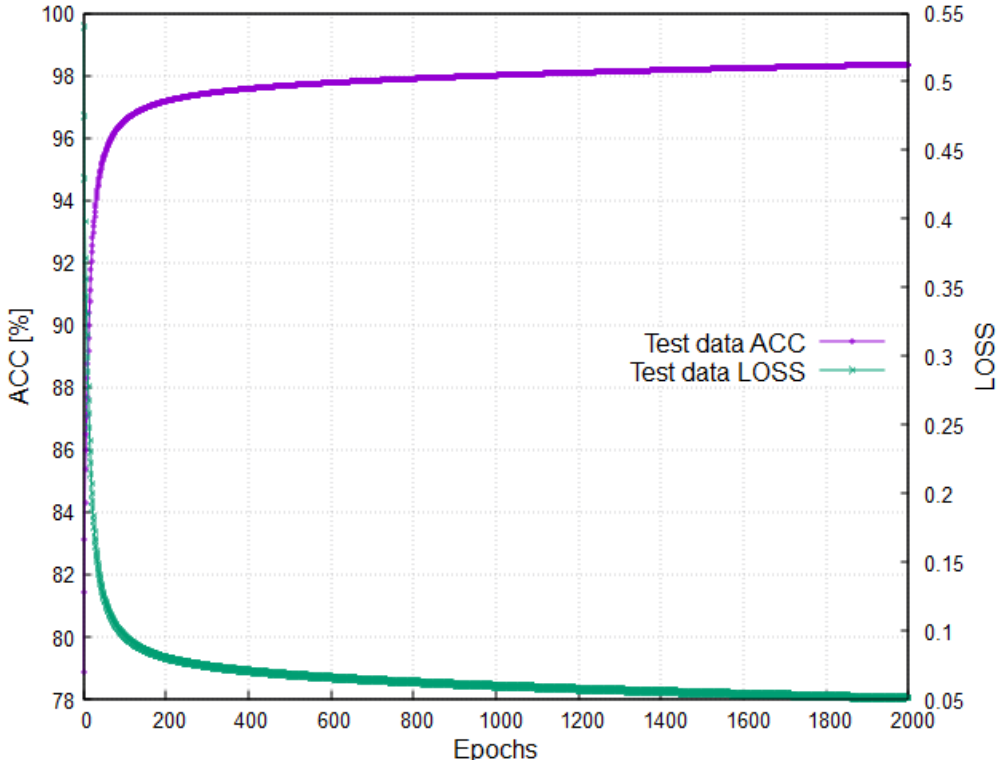

図 5.3: トポロジー選択 NN の学習結果

特徴的な特性を持つ 2 つの目標値に対しての結果を節 5.2.1 と節 5.2.2 示す。結果を評価する際に一致率と改善率を用いる。y は出力、t は目標の特性、s はシミュレーションで得られた特性として、一致率は以下の式で計算する。

$$y_i = \begin{cases} \dfrac{s_i}{t_i} \times 100 \quad (t_i > s_i) \\ \dfrac{t_i}{s_i} \times 100 \quad (t_i \leq s_i) \end{cases} \tag{5.1}$$

また、改善率は大きい程良い特性 (利得、スルーレートなど) の場合は以下の式で計算し、

$$y_i = \frac{s_i - t_i}{t_i} \times 100 \tag{5.2}$$

小さい程良い特性 (消費電流、面積など) の場合は以下の式で計算する。

$$y_i = \frac{t_i - s_i}{t_i} \times 100 \tag{5.3}$$

5.2.1 素子値探索の結果 (目標 1)

CMIR と OVR が 100%で DC Gain、CMRR、PSRR が 100dB となる特性を目標 1 とし、表 5.1 の目標値の列に示す。これを NN に入力すると、69.0%の確率で Rail to Rail フォールデッドカスコード差動対+AB 級出力の回路 (L=2.0um、W=6.0um) である図 2.12 が選択された。次に素子値探索を行った結果を表 5.1、目標値と最終的に得られた結果を表 5.2 に示す。実行

時間は 20 時間とし、最も平均一致率が高かったものを結果とした。このときのシミュレーション回数を横軸、報酬を縦軸としたグラフを図 5.4 に示す。表 5.2 より、最低一致率は Area の 70.7%で平均一致率は 94.6%であった。比較として同じ時間ランダムで素子値の変更を行った結果、最低一致率は Area の 68.9%で平均一致率は 89.4%であった。また、図 5.4 より、シミュレーション回数が 2800 回あたりで解にたどり着き、その後は一定で推移していることが確認できた。

5.2.2 素子値探索の結果 (目標 2)

OR が 100Ω で GBW が 100MHz、SR が 100MV/s となる特性を目標 2 とし、表 5.3 の目標値の列に示す。これを NN に入力すると、98.8%の確率で PMOS 入力フォールデッドカスコード差動対+スーパーソースフォロワの回路 (L=0.2um、W=0.6um) である図 2.10 が選択された。次に素子値探索を行った結果を表 5.3、目標値と最終的に得られた結果を表 5.4 に示す。実行時間は 20 時間とし、最も平均一致率が高かったものを結果とした。このときのシミュレーション回数を横軸、報酬を縦軸としたグラフを図 5.5 に示す。表 5.4 より、最低一致率は SR の 79.6%で平均一致率は 93.6%であった。比較として同じ時間ランダムで素子値の変更を行った結果、最低一致率は OR の 53.1%で平均一致率は 86.8%であった。また、図 5.5 より、学習開始から終盤まで緩やかに報酬が増加していることが確認できた。

表 5.1: 目標 1 の探索された素子値

素子名	素子値
R1	340kΩ
M1〜M3	M=10
M4〜M6	M=3
M7, M8	M=27
M9, M10	M=6
M11, M12	M=5
M13, M14	M=22
M15, M16	M=7
M17, M18	M=2
M19, M20	M=18
M21	M=12
M22, M23	M=16
M24	M=7
M25〜M27	M=16
M28, M29	M=8
M30〜M32	M=22
M33, M34	M=11
M35	M=30
M36	M=26
C1, C2	2pF

表 5.2: 目標 1 の実行結果

	目標値	結果	一致率 [%]	改善率 [%]
PC [W]	5.00E-04	5.28E-04	94.7	-5.6
DC Gain [dB]	1.00E+02	9.89E+01	98.9	-1.1
PM [°]	6.00E+01	5.80E+01	96.6	-3.3
GBW [Hz]	1.00E+06	1.00E+06	100.0	0.0
SR [V/s]	1.00E+06	9.97E+05	99.7	-0.3
THD [%]	4.82E-01	4.82E-01	100.0	0.0
CMRR [dB]	1.00E+02	9.37E+01	93.7	-6.3
PSRR [dB]	1.00E+02	8.29E+01	82.9	-17.1
OVR [%]	1.00E+02	1.00E+02	100.0	0.0
CMIR [%]	1.00E+02	1.00E+02	100.0	0.0
Area [mm^2]	1.00E-02	1.42E-02	70.7	-42.0
OR [Ω]	1.00E+05	1.06E+05	94.1	-6.0
IRN [V/Hz]	2.00E-03	1.98E-03	98.8	1.0
Avg.			94.6	-6.2

表 5.3: 目標 2 の探索された素子値

素子名	素子値
R1	60kΩ
M1〜M3	M=12
M4〜M6	M=14
M7, M8	M=10
M9, M10	M=21
M11, M12	M=23
M13, M14	M=15
M15, M16	M=3
M17, M18	M=2
M19, M20	M=7
M21	M=16
M22	M=19
M23	M=28
M24	M=2
M25	M=6

表 5.4: 目標 2 の実行結果

	目標値	結果	一致率 [%]	改善率 [%]
PC [W]	5.00E-04	6.00E-04	82.2	-20.0
DC Gain [dB]	5.00E+01	4.65E+02	93.0	-7.5
PM [°]	6.00E+01	5.95E+01	99.2	-0.8
GBW [Hz]	1.00E+08	9.89E+07	98.9	-1.1
SR [V/s]	1.00E+08	1.26E+08	79.6	26.0
THD [%]	4.82E-01	4.82E-01	100.0	0.0
CMRR [dB]	5.00E+01	4.94E+01	98.8	-1.2
PSRR [dB]	5.00E+01	4.65E+01	93.0	-7.0
OVR [%]	5.00E+01	4.56E+01	91.2	-8.8
CMIR [%]	7.50E+01	7.69E+01	97.5	2.5
Area [mm^2]	5.00E-04	4.99E-04	99.7	-0.2
OR [Ω]	1.00E+02	9.11E+01	91.1	8.9
IRN [V/Hz]	2.00E-02	2.17E-02	92.3	-8.5
Avg.			93.6	-1.4

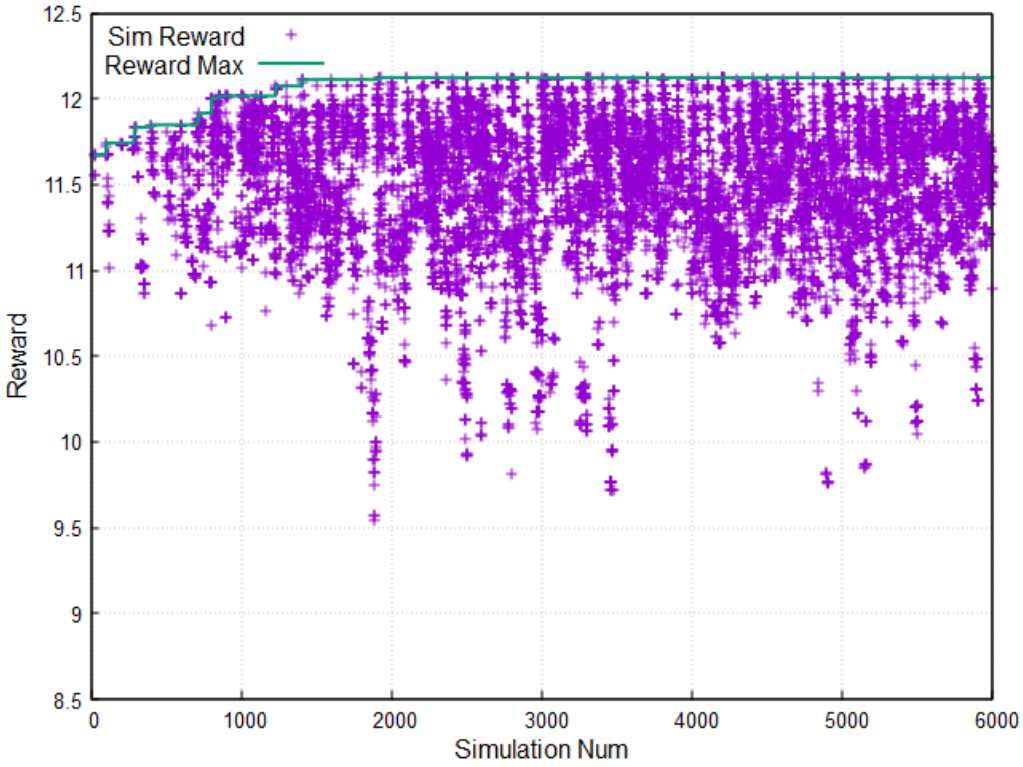

図 5.4: 目標 1 の報酬の変化

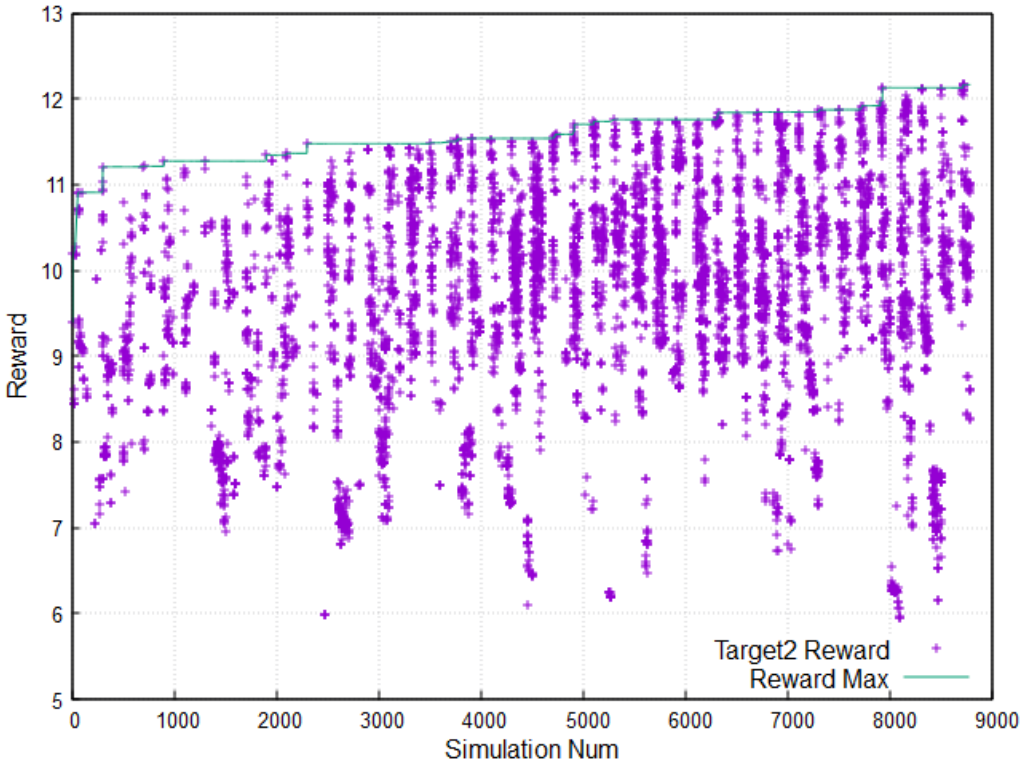

図 5.5: 目標 2 の報酬の変化

第6章　まとめと今後の課題

6.1　まとめ

本論文ではNNによる分類と深層強化学習 (IMPALA) を用いて、アナログ回路設計におけるトポロジー選択と素子値決定を自動化する手法を提案した。例として2つの特徴的な特性の演算増幅器で手法を評価した。結果として計24種類の登録回路から98.3%の精度でトポロジーを選択し、その回路の素子値を探索することで90%以上の一致率で所望特性を得られる回路の自動設計に成功した。また、深層強化学習の初期値依存という問題をNNで予測した値を用いることで解決し、従来よりも高速な収束を実現した。本手法はアナログ回路設計者の知識・経験によらないため、設計時間の短縮と設計者の負担軽減が可能である。

6.2　今後の課題

今回は12トポロジー24種類のデータを収集し、その中から所望特性を得るために最適なトポロジーを選択した。より多くのトポロジー・条件で登録を行うことで選択肢の幅が広がり、更に適切なものを決定できるようになると考えられる。またトポロジー選択にはNNを用いた、グラフ理論とNNを組み合わせたGraph Neural Network (GNN) という技術も近年登場している。これを応用することで精度を向上させられる可能性がある。

素子値探索では一致率は90%は超えたものの、改善率はマイナスになってしまった。これは報酬の計算式上、目標値と近い程高評価としており改善したかは評価に含めていないためである。そのため評価式を改善することで、目標よりも高性能な回路設計が可能であると考えられる。また、アルゴリズムにはIMPALAを用いたが、深層強化学習のアルゴリズムは日々進化を続けている。その評価にAtari2600という家庭用ゲーム機のゲームが良く用いられるが、そこでより高い性能を示しているもの (例: Agent57など) がある。そのためアルゴリズムをより高性能なものに変更することで探索精度の向上や時間短縮が見込める。時間短縮については、より高性能なマシンを使用して並列実行数を増やすことでも高速化・効率化が可能と考えられる。

謝辞

本研究を進めるにあたり、有益な助言をいただいた髙井伸和准教授、同研究分野の加藤博己氏、酒向諒氏、佐藤充氏、関井菜乃氏に心より感謝申し上げます。また、論文審査をしていただきました弓仲康史教授、伊藤直史准教授に心より感謝申し上げます。

参考文献

[1] 松場 輝樹, 髙井 伸和, 福田 雅史, 久保 友助, “深層学習を用いた最適アナログ回路トポロジーの推論”, 電気学会 電子回路研究会, ECT-020-030, Mar. 2020

[2] H.Y. Koh, C.H. Sequin, P.R. Gray, “OPASYN: a complier for CMOS operational amplifiers”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 9, Issue 2, Feb. 1990

[3] K. Swings, W. Sansen, “DNALD: a workbench for interactive design space exploration and sizing of analog circuits”, Proceedings of the European Conference on Design Automation, pp.475-479, Feb. 1991

[4] M.M. Hershenson, S.P. Boy, T.H. Lee, “Optimal Design of a CMOS Op-Amp via Geometric Programming”, IEEE Transactions on Computer-Aided Design, Volume 22, No.1, pp.1-21, Jan. 2001

[5] JIANHAIYU, ZHIGANGMAO, “A Design Method in CMOS Operational Amplifier Optimization Based on Adaptive Genetic Algorithm”, WSEAS Transactions on Circuits and Systems archive, Volume 8, Issue 7, pp.548-558, Jul. 2009

[6] J. Marsik, O. Subrt, P. Martinek, “Developing Automated Design Procedure for Operational Amplifier blocks”, 2008 International Conference on Signals and Electronic Systems, Sep. 2008

[7] H. Wang, J. Yang, H. Lee, S. Han, “Learning to Design Circuits”, arXiv:1812.02734, Dec. 2018

[8] 演算増幅器設計コンテスト運営委員会, “2020 年演算増幅器設計コンテスト”, https://www.ec.ict.e.titech.ac.jp/opamp/2020/, 2020

[9] Behzad Razavi, “Design of Analog CMOS Integrated Circuits”, McGraw-Hill Companies, Oct. 2003

[10] Phillip E. Allen, Douglas R. Holberg, “CMOS Analog Circuit Design”, Oxford University Press, Aug. 2011

[11] P.Prasad Rao, K.Lal Kishore, “A 80Ms/sec 10bit PIPELINED ADC Using 1.5Bit Stages And Built-in Digital Error Correction Logic”, International Journal of VLSI Design and Communication Systems, Sep. 2011

[12] 木村 優志, “現場で使える! Python 深層学習入門 Python の基本から深層学習の実線手法まで”, 翔泳社, Jun. 2019

[13] Sergey Ioffe, Christian Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, arXiv:1502.03167, Feb. 2015

[14] Xinyu Liu, Xiaoguang Di, “TanhExp: A Smooth Activation Function with High Convergence Speed for Lightweight Neural Networks”, arXiv:2003.09855, Mar. 2020

[15] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, Journal of Machine Learning Research 15 (2014) 1929-1958, 2014

[16] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, Jiawei Han, “On the Variance of the Adaptive Learning Rate and Beyond”, arXiv:1908.03265, Aug. 2019

[17] Michael R. Zhang, James Lucas, Geoffrey Hinton, Jimmy Ba, “Lookahead Optimizer: k steps forward, 1 step back”, arXiv:1907.08610v1, Jul. 2019

[18] Diederik P. Kingma, Jimmy Ba, “Adam: A Method for Stochastic Optimization”, arXiv:1412.6980 Dec. 2014

[19] 伊藤 多一, 今津 義充, 須藤 広大, 仁ノ平 将人, 川﨑 悠介, 酒井 裕企, 魏 崇哲, “現場で使える! Python 深層強化学習入門 強化学習と深層学習による探索と制御”, 翔泳社, Aug. 2019

[20] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu, “Asynchronous Methods for Deep Reinforcement Learning”, arXiv:1602.01783, Feb. 2016

[21] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, Koray Kavukcuoglu, “IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures”, arXiv:1802.01561, Feb. 2018

学会成果

[1] 齋藤 彰寛, 髙井 伸和, 松場 輝樹, “機械学習を用いた所望特性を満たす回路トポロジーの選択及び素子値決定”, 第 10 回 電気学会 東京支部 栃木・群馬支所 合同研究発表会, ETG-20-90, pp. 275-278, 群馬工業高等専門学校, Mar. 2020

[2] 松場 輝樹, 髙井 伸和, 齋藤 彰寛, 今野 哲史, “幅広い所望特性に対する適切トポロジーの推論”, 電気学会 電子回路研究会, ECT-020-030, 東京, Jun. 2020

[3] 齋藤 彰寛, 髙井 伸和, 今野 哲史, “機械学習による回路トポロジーの選択及び素子値決定”, 電気学会 電子回路研究会, ECT-020-046, Web 開催, Jun. 2020

[4] A. Saito, N. Takai, S. Konno, “Determination of Circuit Topology and Element Values from Desired Characteristics by Machine Learning”, Conference on Electronics Circuits & Systems (ICECS 2020), pp. 1-4, Glasgow Scotland, Nov. 2020