100. 深層学習 day4

秋葉洋哉

2024 年 7 月 21 日

1　強化学習

1.1　概要

強化学習は、エージェントが環境と相互作用し、環境から報酬を受け取ることで、最適な行動を学習する手法である。強化学習の特徴は、報酬を最大化するような行動を学習することである。ただし、報酬を得るためにはどうしたらよいか、という問題は、非自明である場合が多く、報酬のみではスパースで扱いづらいため、途中の過程で出てくる状態や、エージェントの行動に価値を付与し、その価値を最大化させる問題に置き換えて考える。よって、強化学習では、状態、行動、報酬、方策、価値関数の 5 つの要素で構成される。

要素	変数	定義	(例) 将棋の場合
状態	s(state)	エージェントが環境と相互作用する際の状態	将棋盤の状態
行動	a(action)	エージェントが環境に対して取る行動	駒を動かす
報酬	r(reward)	エージェントが環境から受け取る報酬	勝敗
方策	π(policy)	エージェントが状態に対して取る行動を決定するための戦略	次の一手
価値関数	V(Value), Q(Quality)	エージェントが状態や行動に対してどれだけ価値があるかを表す関数	評価値

表 1: 強化学習で用いられる単語の定義と例

強化学習の手法には、方策勾配法、Q 学習、SARSA、DQN、DDPG、PPO などがある。主に 3 つの方針が用いられる。

1. 方策ベース：状態価値 V を推測し、以降の状態価値の現在割引和 (同じ報酬が得られるなら早い方がいいという考え方) を最大化するような方策 π を学習する

2. 価値ベース：エージェントの行動価値 Q を推測し、以降の行動価値の現在割引和を最大化するような方策 π を学習する
3. モデルベース：環境が分かっているという仮定のもと、価値最大化をプランニングする

以下に、強化学習システムの画像を示す。

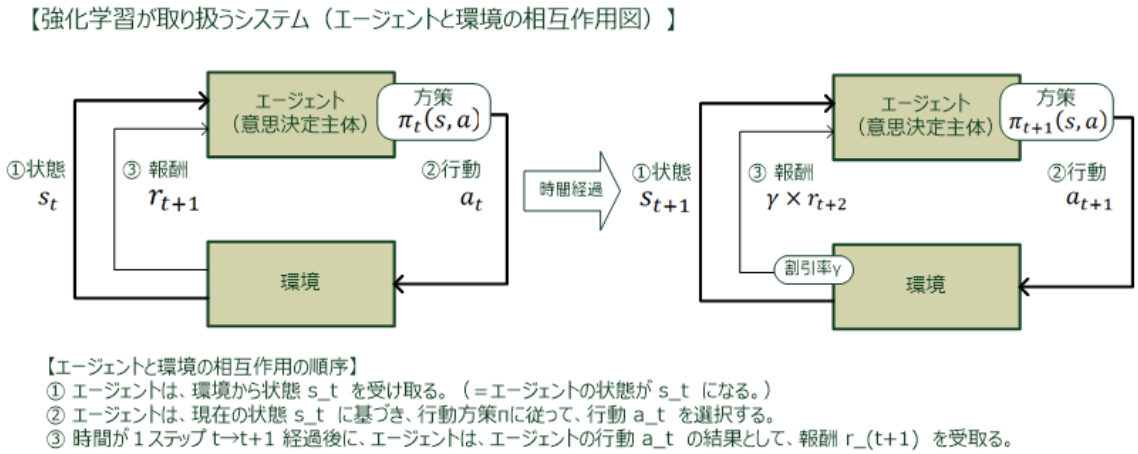


図 1: 強化学習システムの概要

ここで重要な点は、環境について事前に完璧な知識があれば最適な行動を決定することができるが、環境についての知識が不完全である場合最適な行動を学習することが困難であるということである。強化学習の場合、不完全な知識を元に行動しながら、データを収集することで最適な行動を学習していく。

強化学習は、価値関数をどのように設定するか、そして、その価値関数を最大にするにはどのような方策関数が必要かという問題を解くことが目的となる。

1.2 簡単な歴史

年	説明
1950 年代	強化学習の始まり。Richard Bellman が動的計画法を提案。
1960 年代	Richard Sutton が TD 学習を提案。
1989 年	Christopher Watkins が Q 学習を提案。
1992 年	Gerald Tesauro がバックギャモンの AI を開発。
2006 年	Geoffrey Hinton が深層強化学習を提案。
2013 年	DeepMind が DQN を提案し、Atari 2600 のゲームで人間を超える性能を達成。
2016 年	AlphaGo が囲碁の世界チャンピオンを破り、強化学習の可能性を示す。

表 2: 強化学習の歴史

かつて、強化学習はテーブルベースの手法が主流だった。その手法とは、状態と行動の組み合わせに対し

て価値を表すテーブルを用意し、そのテーブルを更新することで最適な行動を学習する手法である。しかし、テーブルベースの手法は状態や行動の数が増えるとテーブルのサイズが指数関数的に増加するため、現実の問題には適用しづらいという問題があった。

その後、そのテーブルの対応付けを関数で表現することで、状態や行動の数が増えても対応できる手法が提案された。この手法が、ディープラーニングを用いた強化学習である。ディープラーニングを用いた強化学習は、状態や行動の数が増えても対応できるため、現実の問題に適用できるようになり、強化学習が注目されるようになった。

1.3 価値関数

価値関数には状態価値関数と行動価値関数の 2 つがある。近年の強化学習では、行動価値関数を用いることが一般的である。

1. 状態価値関数 $V(s)$：状態 s において得られる報酬の期待値 (状態のみを入力とする関数)
2. 行動価値関数 $Q(s,a)$：状態 s において、行動 a を取った後に得られる報酬の期待値 (状態と行動を入力とする関数)

1.4 方策関数

方策関数 π は、状態 s において行動 a を取る確率を表す関数である。方策関数の目的は、価値関数を最大にするような行動を決定することである。例えば、将棋の場合は、価値関数が評価値の値を決定し、方策関数が次の手を決定する。

1.5 最適化問題の解き方

強化学習における最適問題の解き方は、主に以下の 3 つの方法がある。

1. 動的計画法：選択肢を一つずつ増やしていき、逐次的に最適解を求めていく方法
2. モンテカルロ法：方策ベースの手法で用いられ、ロールアウトによって疑似的にいくつかの選択肢を比較し、方策を決定する方法
3. TD 法：価値ベースの手法で用いられ、t ステップ後の行動価値までを考えたうえでの行動選択の最適化を行う方法

1.5.1 方策勾配法

方策勾配法は、方策関数を直接更新する手法である。方策勾配法は、方策関数の勾配を求め、その勾配の方向に方策関数を更新することで、最適な方策関数を学習する。方策勾配法では、方策の勾配 θ を更新していく。

$$\theta^{(t+1)} = \theta^{(t)} + \epsilon \nabla J(\theta) \tag{1}$$

ここで、$J(\theta)$ は方策関数の期待報酬であり、ϵ は学習率である。この式では、期待報酬 $J(\theta)$ を、方策関数の期待報酬を最大化するように更新していく。第二項は、方策関数の勾配を求めるための式であり、以下のよう

に表される。

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_{a \in A} \pi_\theta(a|s) Q^\pi(s,a) \tag{2}$$

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)] \tag{3}$$

ここで、π_θ は方策関数、$Q^{\pi_\theta}(s,a)$ は行動価値関数である。下付きの θ や、π_θ は、方策関数のパラメータであることを表している。

内容	チャンネル数	説明
石の色	3	自分の石、相手の石、空き目
オール 1	1	すべての要素が 1 である盤面 (1 チャンネルすべてが 1 である盤面)
着手履歴	8	過去 8 手の着手 (チャンネル 1 つに 1 つの石が対応)
呼吸点	8	該当の位置に石がある場合、その石を含む連の呼吸点の数
取れる石の数	8	該当の位置に石を打った場合、取れる石の数
取られる石の数	8	該当の位置に石を打たれた場合、取られる石の数
着手後の呼吸点の数	8	該当の位置に石を打った場合、その石を含む連の呼吸点の数
着手後にシチョウで取れるか	1	該当の位置に石を打った場合、シチョウで隣接連を取れるかどうか
着手後にシチョウで取られるか	1	該当の位置に石を打った場合、シチョウで隣接連を取られるかどうか
合法手	1	ルールに見合った手であるかどうか
オール 0	1	すべての要素が 0 であるかどうか
手番	1	現在の手番が黒番であるかどうか (ValueNet のみ)

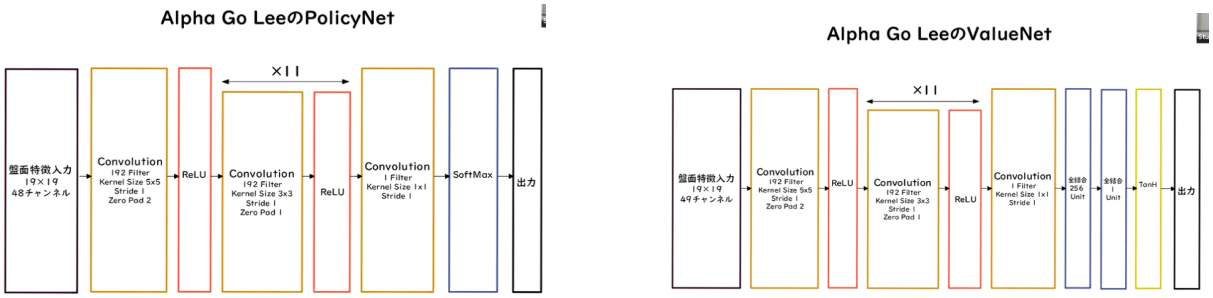
表 3: AlphaGo Lee の入力データ

2 AlphaGo

2.1 AlphaGo Lee

2.1.1 概要

AlphaGo Lee は、2016 年に Google DeepMind が開発した囲碁の AI である。19×19×48 の囲碁の盤面を入力すると、次の一手・評価値を出力する。各目毎の 48 チャンネルは、表 3 以下の通りである。Policy Net では、19×19 のすべての目の着手確率、Value Net では、評価値を出力する。



(a) 方策関数: Policy Net　　(b) 価値関数: Value Net

図 2: AlphaGo Lee で用いられる方策関数と価値関数

2.1.2 AlphaGo Lee の学習

AlphaGo の学習は以下のステップで行われる。

1. 教師あり学習による RollOut Policy と Policy Net の学習

2. 強化学習による Policy Net の学習
3. 強化学習による Value Net の学習

以下では、それぞれのステップについて説明する。

RollOut Policy とは、ニューラルネットワークではなく、線形の方策関数で、高速に着手確率を出すために用いられる。強化学習では Policy Net の学習 1 回に対して 3 ミリ秒、計 10^8 オーダーの学習回数が必要となるため、1 手打つのに莫大な時間がかかる。そのため、RollOut Policy を用いて、アバウトだが高速に着手確率を出す (1マイクロ秒/1 学習) ことで、後々の Policy Net の学習を効率化する。RollOut Policy は、教師あり学習によって学習される。教師データは、人間の棋譜が用いられる。また、Policy Net も同様に人間の棋譜を用いて教師あり学習によって学習する。

PolicyNet は、いわば Bot であり、自分自身が最も身近な対戦相手となるため、PolicyNet 同士で対戦させることで、PolicyNet の学習を行うのが効率的である。しかし、現状の PolicyNet 同士で対戦させると、PolicyNet 同士が同じような戦略を取るため、過学習が発生しやすい。

そこで、PolicyNet の学習には PolicyPool と呼ばれる複数の PolicyNet の集合を用いる。PolicyPool は、PolicyNet の学習過程を 500Iteration 毎に保存したものである。PolicyPool から PolicyNet をランダムに選択したモデルと、現状の PolicyNet を対戦させ、その結果を用いて方策勾配法で学習を行う。こうすることで、PolicyNet 同士が同じような戦略を取ることを防ぎ、過学習を防ぐことができる。

2.1.3 強化学習による Value Net の学習

ValueNet では、PolicyNet の学習時の対局結果を教師データとして、教師あり学習を行う。以下に教師データの作成手順を示す。

1. まず、教師あり学習で作成した PolicyNet (SL PolicyNet) で N 手までを打つ。
2. その後、$N+1$ 手目の手をランダムに選択し、その手で進めた局面を $S(N+1)$ とする。
3. $S(N+1)$ から強化学習で作成した PolicyNet (RL PolicyNet) で終局まで打ち、その結果の報酬を R とする。
4. $S(N+1)$ と R を教師データ対として ValueNet の入力とし、平均二乗誤差を最小化するように学習を行う。

AlphaGo における強化学習の学習手法として、モンテカルロ木探索という手法が最も有効とされている。

2.1.4 モンテカルロ木探索

モンテカルロ木探索は、盤面の価値や勝率予測値を考えずに、現局面から末端局面までランダムにシュミレーション (PlayOut と呼ぶ) を行い、その勝敗を集計して着手の優劣を決定する手法である。

モンテカルロ木探索は、ある手を選んだ場合のシュミレーション回数が一定数を超えた時にその手を開始とする局面からの計算を打ち切り、その手を着手した後の局面をシュミレーション開始局面として、再度シュミレーションを行って、探索木を成長させていく。

2.2 AlphaGo Zero

2.2.1 概要

AlphaGo Zero は、2017 年に Google DeepMind が開発した囲碁の AI である。AlphaGo Zero は、AlphaGo Lee と比較して以下の点が改善されている。

1. 教師あり学習を一切行わず、強化学習のみで作成
2. 特徴入力からヒューリスティックな要素 (表 3) を排除し、石の配置のみを入力とした
3. PolicyNet と ValueNet の統合
4. Residual Network を導入
5. モンテカルロ木探索から RollOut Policy を排除

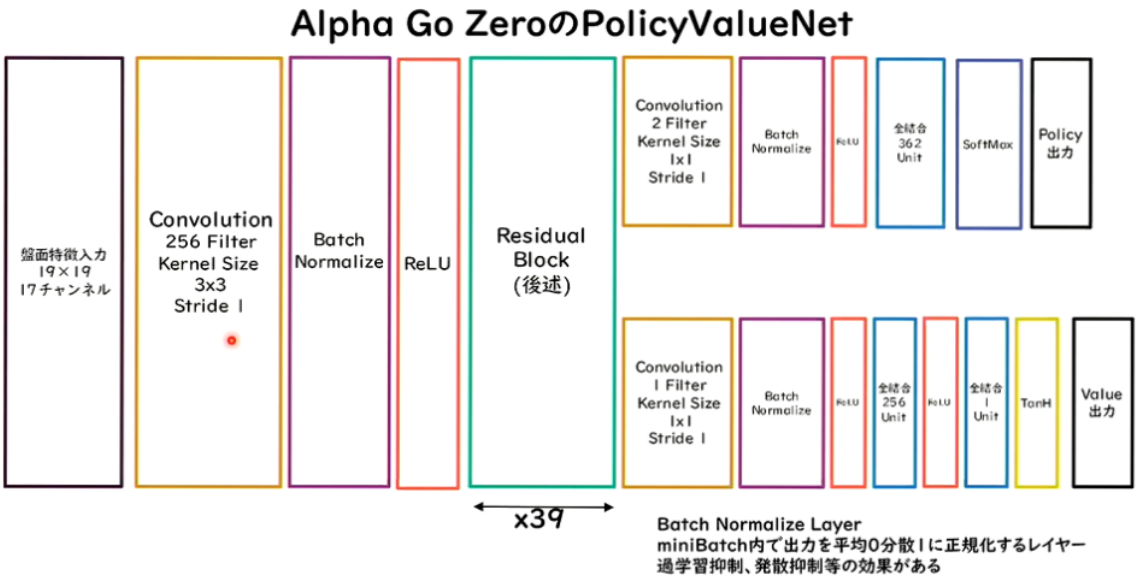


図 3: AlphaGo Zero の方策価値関数： Residual Network の後にネットワークが 2 つに分岐していることが特徴である。

2.2.2 Residual Network

Residual Network とは、勾配消失問題を解決するために提案されたニューラルネットワークの構造である。Residual Network は Residual Block と呼ばれる単位を重ね合わせた (AlphaGo Zero の場合は 39 層) ネットワークである。Residual Block の中では入力に対して畳み込み・プーリング・活性化関数を適用した結果を、元の入力と足し合わせる。このショートカット結合は、いわばその Block 内の畳み込み層をドロップアウトさせているものだと考えることができる。ドロップアウトではランダムにノードを削除するが、Residual Network では、層を残したまま、その層の重みを 0 にすることで無効化する。つまり、スルーする層を学習するドロップアウトと考えることができる。Residual Network は、深いネットワークに対して勾配消失問題を解決するだけでなく、学習の収束を早める効果がある。

2.2.3 AlphaGo Zero における Network の工夫

AlphaGo Zero では、以下の工夫が施されている。

1. Residual Block に対して Bottleneck の導入 (1x1 畳み込みを用いて次元削減)
2. Residual Block に対して PreActivation の導入 (Batch Normalization と活性化関数の順序を入れ替える)
3. Network 構造に対して WideResNet の導入 (フィルター数を k 倍に増やし、GPU を効率的に用いる)
4. WideResNet に対して PyramidNet の置き換え (各層で Filter 数を増やす)

3　軽量化・高速技術

3.1　分散深層学習

深層学習は多くのデータを使用したり、パラメータ調整のための計算量が多いため、複数の計算資源 (ワーカー) を使用して、並列的にニューラルネットを構成することで、効率の良い学習を行うことが必要である。そのために行われる高速化は大きく分けて

1. データ並列化 (図 4 左)
2. モデル並列化 (図 4 右)
3. GPU による高速化

がある。ちなみに、機械学習モデルの計算量は 10 倍/年の割合で増加している。一方で、GPU の性能は 2 倍/年の割合で増加しているため、GPU の性能を最大限に引き出すことが重要である。

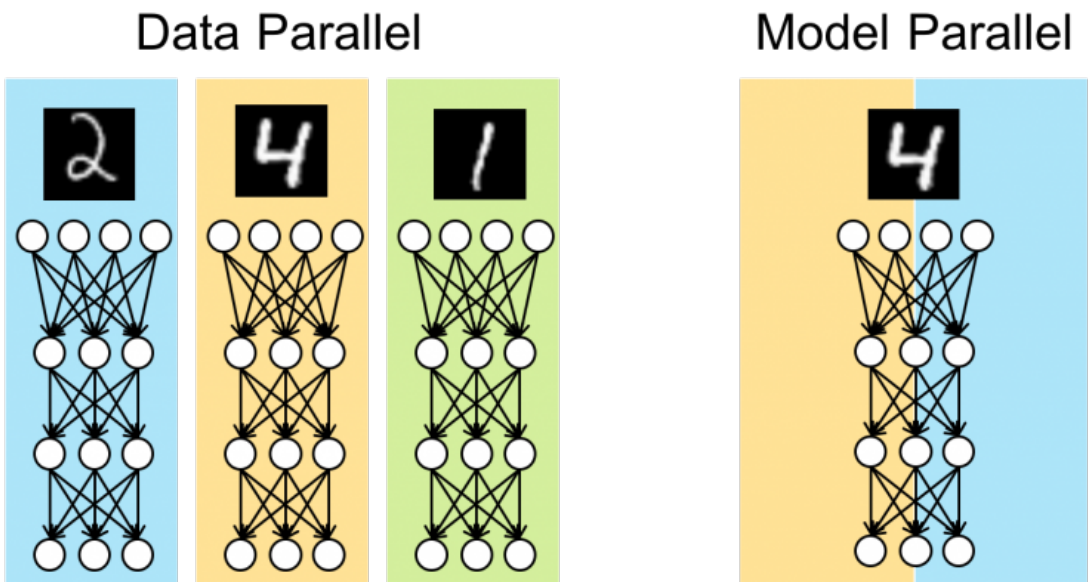


図 4: データ並列化とモデル並列化のイメージ図 (https://tech.preferred.jp/ja/blog/model-parallelism-in-dnn より引用)

3.1.1　データ並列化

データ並列化は、元のビッグデータを複数の計算機で分割して学習させる方法で、同期型と非同期型の 2 つの手法がある。

同期型は、親モデルとなる 1 つのモデルを複数のワーカーに分配し、それぞれのワーカーで学習を行い、各ワーカーで学習した時の誤差の勾配を平均化して、親のモデルに反映し、パラメータが更新する手法である。そうして、更新されたモデルを再度ワーカーに分配して学習を進める。同期型は、各ワーカーが計算が終わるのを待ち、全ワーカーの勾配が出たところで勾配の平均を計算し、親モデルのパラメータを更新する。

非同期型は、各ワーカーが独立して学習を行い、出来上がったモデルをパラメータサーバーに送り (push)、送った後、ワーカーはパラメータサーバーにあるモデルを取得 (pop) して新しく学習を始める。パラメータサーバーは、先入れ先出しのキューであるため、ワーカーが送ったモデルが順番に処理される。

同期型は、全ワーカーの計算が終わるのを待つため、計算時間がかかるが、パラメータの更新が安定している。一方、非同期型は、ワーカーが独立して学習を行うため、計算時間が短いが、パラメータの更新が不安定である。例えば、世界中のスマートフォンを使って、分散深層学習を行う場合、非同期型が適しているが、GPU を使って分散深層学習を行う場合、同期型が適している。

3.1.2 モデル並列化

モデル並列化は、親モデルとなる 1 つのモデルを複数のワーカーに分配し、それぞれのワーカーでモデルの一部分を学習させる方法である。モデルの分割方法は、層ごとに分割する方法や、分岐した部分を分割する方法がある。モデル並列化は 1 台の計算機内で GPU を用いて学習を行わせる場合が多く、モデルのパラメータ数が多いほどスピードアップの効率も向上する。

3.1.3 GPU による高速化

GPU(図 5b) は、CPU(図 5a) に比べて大量のデータを並列処理することが得意である。そのため、深層学習の学習には GPU を用いることが一般的である。もともとは、グラフィック描画のために開発された GPU であるが、その並列処理能力を活かして、深層学習の学習に用いられるようになり、そうしたグラフィック用途以外の用途で用いられる GPU を GPGPU (General-Purpose computing on Graphics Processing Units) と呼ぶ。

CUDA (Compute Unified Device Architecture) は、NVIDIA が提供する GPU 向けの開発環境である。CUDA を用いることで、GPU を用いた高速な計算が可能となる。OpenCL は、Apple が提唱した GPU 向けのプログラミング環境である。OpenCL は、NVIDIA の CUDA と比較して、ハードウェアに依存しないため、様々なハードウェアで利用することができる。

Tensorflow や PyTorch を用いる場合、ユーザーは CUDA や OpenCL を意識することなく、GPU を用いた高速な計算を行うことができる。

(a) CPU　　(b) GPU

図 5: CPU と GPU の比較 : CPU は計算を行った結果をメモリに書き込む。GPU は算術論理演算装置 (ALU) を 2500∼5000 個搭載し、数千の乗算と加算を行う。(https://cloud.google.com/tpu/docs/intro-to-tpu?hl=ja より引用)

3.1.4 TPU による高速化

TPU (Tensor Processing Unit) (図 6) は、Google が開発した AI 向けの ASIC (Application-Specific Integrated Circuit) である。TPU は、GPU に比べて、深層学習の学習に特化しており、高速な計算が可能である。

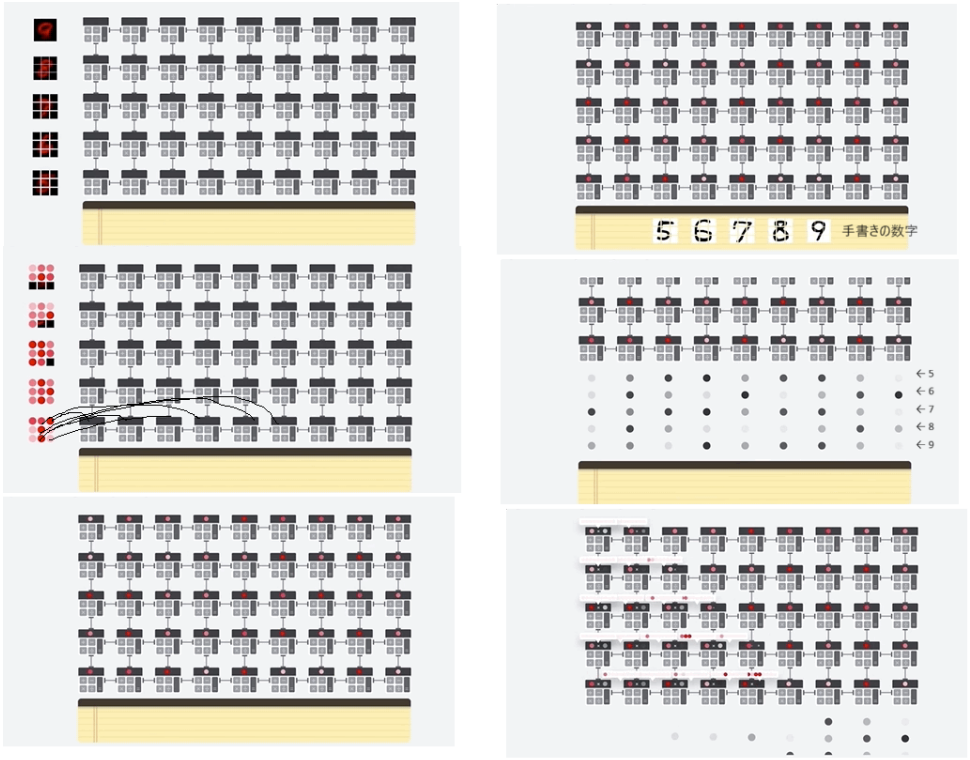


図 6: TPU では、メモリから Matrix Multiplication Unit(MXU) にパラメータを読み込み (画像左列)、次にメモリからデータを読み込む (画像右列)。行列乗算演算中にメモリアクセスが無いため、高いスループットを達成できる。(https://cloud.google.com/tpu/docs/intro-to-tpu?hl=ja 動画よりキャプチャ)

3.1.5 量子化 (Quantization)

ネットワークが大きくなると、大量のパラメータをメモリで保持する必要があり、メモリの使用量が増加する。そのため、64bit の浮動小数点を 32bit などの下位の精度に変換することで、メモリの使用量を削減することができる。このような変換を量子化と呼ぶ。

浮動小数点とは、指数部と仮数部に分かれている数値表現のことであり、64bit の浮動小数点は、指数部 11bit、仮数部 52bit で構成されている。一方、32bit の浮動小数点は、指数部 8bit、仮数部 23bit で構成されている (図 7)。機械学習用の bfloat16 は、指数部 8bit、仮数部 7bit で構成されている。

例えば、64bit の浮動小数点を使った場合、パラメータが 1024 個だった時には、8byte×1024=8KB、1024×1024 個だった時には、8MB、1024×1024×1024 個だった時には、8GB のメモリが必要となる。一方、32bit の浮動小数点を使った場合、パラメータが 1024 個だった時には、4KB、1024×1024 個だった時には、

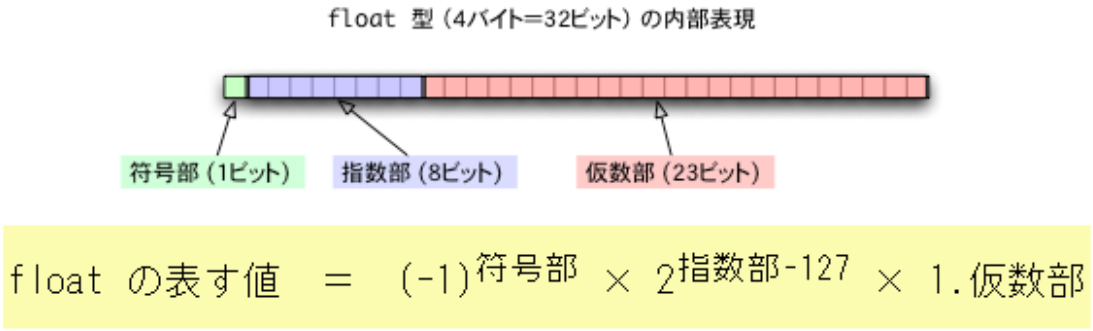


図 7: 32bit 浮動小数点の内部表現 (https://www.cc.kyoto-su.ac.jp/ yamada/programming/float.html より引用)

4MB、1024×1024×1024 個だった時には、4GB のメモリが必要となる。BERT では、およそ 0.6GB のメモリが必要となるが、量子化を行うことで、0.3GB のメモリで動作するようになる。

ただし、浮動小数点の性質上、量子化を行うことで、精度が下がる。例えば、Yin et al.(2017) は図 8 のように、量子化を行うことで、検出時間は 1/4 1/5 になり、精度が下がったり、余白が広くなっていることを示している。

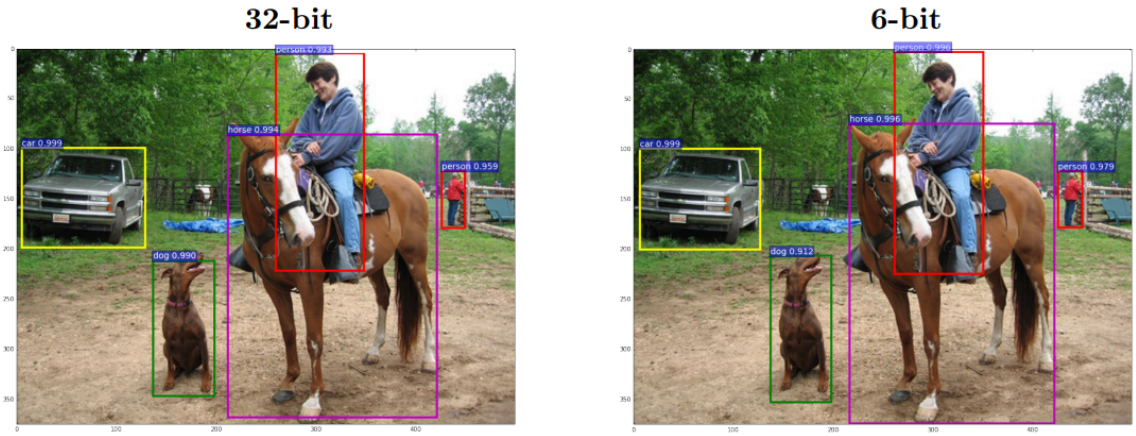
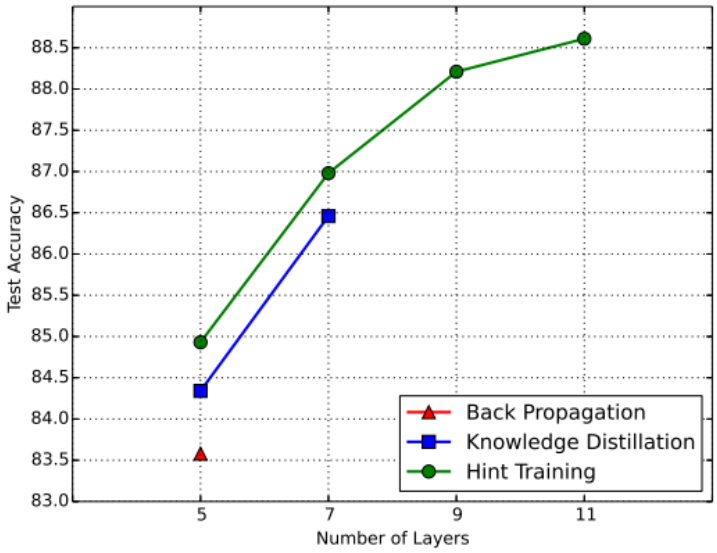


図 8: 量子化による精度の低下 (Yin et al.(2017) より引用): 6bit にしたときに、各分別結果の値が若干異なることがわかる。

3.2 蒸留

蒸留とは、大きなモデル (教師モデル) の知識を小さなモデル (生徒モデル) に継承する手法である。蒸留では、教師モデルは学習済みとし重みを固定し、生徒モデルの重みを学習する。データを与えた時の教師モデルの出力をソフトマックス関数で確率分布に変換し、生徒モデルの出力との誤差を最小化するように学習を行う。少ない層でも高い精度の出力を行うことができる (図 9)。



(a) 30M Multiplications

図 9: 蒸留後の生徒モデルの層数 (横軸) と精度 (縦軸) (Romero et al.(2015) より引用)

3.3 プルーニング

プルーニングとは、モデルのパラメータを削減する手法である。プルーニングでは、学習後にモデルのパラメータのうち、重要でないパラメータを削除する。例えば、閾値を決めて、その閾値を超えるパラメータを削除する。プルーニングを行うことで、モデルのサイズを削減し、計算量を削減することができる (図 10)。

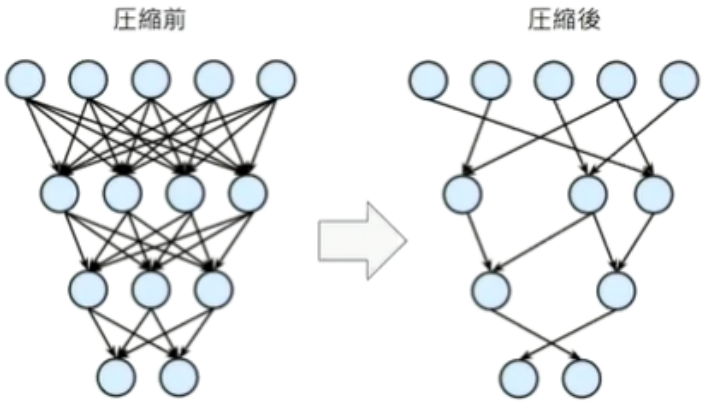


図 10: プルーニングのイメージ図

例えば、佐々木など (2015) は、全結合層のパラメータ削減手法を比較している。閾値パラメータ α が最小の 0.5 の時にはすでにおよそ 50% のパラメータが削減されるが、精度は 92% からわずかに 0.34% 下がるだけであることがわかっている (図 11)。

表 5　CaffeNet における結合単位でのパラメータ削減時の各層のパラメータ削減率

	パラメータ削減率（%）				
α	全結合層 1	全結合層 2	全結合層 3	全結合層全体	精度 (%)
0.5	49.54	47.13	57.21	48.86	91.66
0.6	57.54	55.14	64.99	56.86	91.39
0.8	70.96	69.04	76.44	70.42	91.16
1.0	80.97	79.77	83.74	80.62	91.11
1.3	90.51	90.27	90.01	90.43	91.02
1.5	94.16	94.28	92.45	94.18	90.69
1.6	95.44	95.64	93.38	95.49	90.53
1.7	96.41	96.66	94.20	96.47	89.84
1.8	97.17	97.43	94.88	97.23	89.94
1.9	97.75	98.02	95.45	97.81	89.45
2.0	98.20	98.45	95.94	98.26	89.56
2.2	98.80	99.03	96.74	98.85	88.97
2.4	99.19	99.40	97.41	99.24	87.74
2.6	99.46	99.64	97.95	99.50	87.08

図 11: もともとのモデルの精度は 92% 程度 (佐々木など (2015) より引用)

4 応用技術

4.1 MobileNet : Efficient Convolutional Neural Networks for Mobile Vision Applications

MobileNets は、軽量な画像認識畳み込みニューラルネットワークである。深層学習による画像分類予測は 2017 年でほとんどが完成されており、それ以降は軽量化、高速化、精度向上が求められてきた。MobileNets は、そうした流れの中で開発された。

まず、一般的な畳み込みレイヤーでは、入力 $H \times W \times C$ の画像に対して、$K \times K \times C$ のフィルタを M 個適用して、$H \times W \times M$ の出力を得る。このときの計算量は、$H \times W \times C \times K \times K \times M$ である。

MobileNets では、Depthwise Separatable Convolution と Pointwise Convolution という畳み込み手法を用いて、計算量を削減している。

Depthwise Separatable Convolution は、フィルタのチャンネル数が 1 で、チャンネルを通した畳み込みを行わず、チャンネルごとに畳み込みを行う手法である。この手法では、入力 $H \times W \times C$ の画像に対して、$K \times K \times 1$ のフィルタを適用して、$H \times W \times C$ の出力を得る。このときの計算量は、$H \times W \times C \times K \times K$ となり、通常の畳み込みに比べて、C 倍の計算量削減が可能である。

Pointwise Convolution は、GoogLeNet にも用いられた畳み込み手法で、$1 \times 1 \times C$ のフィルタを M 個適用して、チャンネル数を変換する手法である。この手法では、入力 $H \times W \times C$ の画像に対して、$1 \times 1 \times M$ のフィルタを適用して、$H \times W \times M$ の出力を得る。このときの計算量は、$H \times W \times C \times M$ となり、通常の畳み込みに比べて、$K \times K$ 倍の計算量削減が可能である。

通常の畳み込みに相当する部分を、Depthwise Separatable Convolution と Pointwise Convolution で担うことで、計算量は以下の削減される。

$$\text{通常の畳み込み} : H \times W \times C \times K \times K \times M \tag{4}$$

$$\text{MobileNets} : H \times W \times C \times K \times K + H \times W \times C \times M \tag{5}$$

4.2 DenseNet : Densely Connected Convolutional Networks

DenseNet は、2016 年に提案された画像認識畳み込みニューラルネットワークである。特徴は、Dense Block と呼ばれるブロックを用いて、畳み込み層間を結合する手法である。Dense Block 内は、複数の畳み込み層が結合されており、l 番目の入力は、$l-1$ 番目までの出力を全て結合して入力とする。

$$x_l = H_l([x_0, x_1, \cdots, x_{l-1}]) \tag{6}$$

ここで、H_l は、l 番目の畳み込み層を表し、$[x_0, x_1, \cdots, x_{l-1}]$ は、$l-1$ 番目までの出力を結合したものである。最初のチャンネル数が k_0 で、出力が k チャンネルの場合、Block 内の畳み込み層を経るごとに、出力は k チャンネル増加する。結果、Dense Block の層数が n、フィルタ数が k の場合、Block に k_0 チャンネルの入力を与えると、$k_0 + k \times n$ の出力が得られることとなる。

こうして得られた出力は、Transition Layer と呼ばれる畳み込み層で、チャンネル数を削減する。Transition Layer は、Batch Normalization、畳み込み、プーリングの 3 つの層から構成されている。

図 12 では、通常の畳み込み、ResNet、DenseNet の結合方法の違いを示している。通常の畳み込みでは、l 番目の畳み込み層の入力は、$l-1$ 番目の出力のみである。ResNet では、l 番目の畳み込み層の入力は、$l-1$

番目の出力と $l-2$ 番目の出力を結合して入力とする。DenseNet では、l 番目の畳み込み層の入力は、$l-1$ 番目までの出力を全て結合して入力とする。

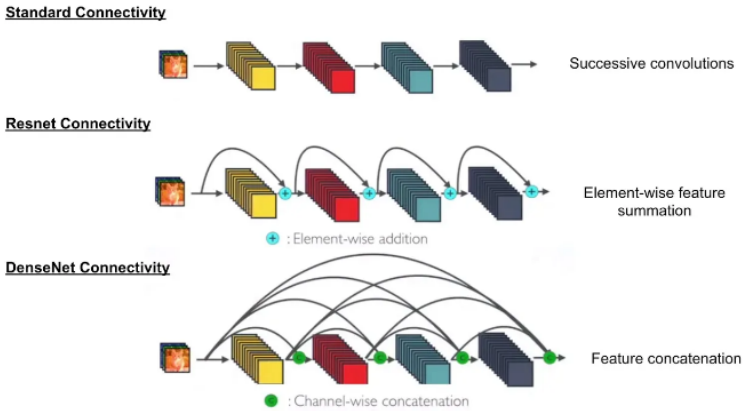


図 12: 畳み込み層間の結合方法：(上) 通常の畳み込み, (中)ResNet の結合方法, (下)DenseNet の結合方法 (https://cvinvolution.medium.com/why-isnt-densenet-adopted-as-extensive-as-resnet-1bee84101160 より引用)

4.3 Batch Normalization, Layer Normalization, Instance Normalization

Batch Normalization, Layer Normalization, Instance Normalization は、ニューラルネットワークの正則化手法である。

Batch Normalization は、ミニバッチに含まれるサンプルの同一チャンネルが同一分布に従うように正規化する。効果がミニバッチの数に依存し、ミニバッチの数が少ないと効果が低下する。

Layer Normalization は、それぞれのサンプルのすべてのピクセルが同一分布に従うように正規化する。ミニバッチの数に依存しないため、ミニバッチの数が少なくても効果がある。Instance Normalization は、それぞれのサンプルの、同一チャンネルのすべてのピクセルが同一分布に従うように正規化する。コントラストの強調や色調・テクスチャの変換に使われることが多い。

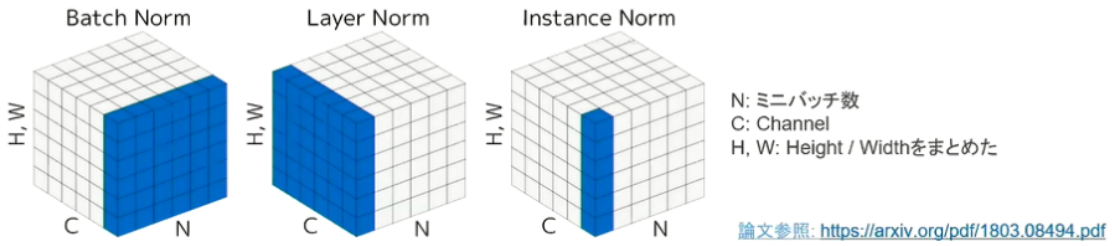
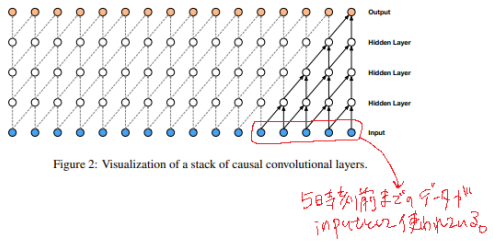


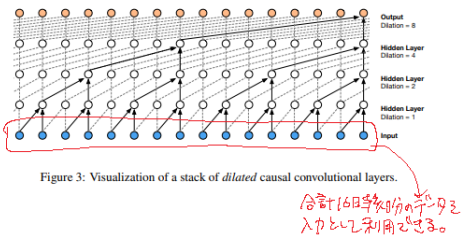
図 13: Batch Normalization, Layer Normalization, Instance Normalization の比較 (https://arxiv.org/pdf/1803.08494.pdf より引用)

4.4 WaveNet : A Generative Model for Raw Audio

WaveNet は、2016 年に提案された音声生成モデルである。WaveNet は、Dilatad Convolution を用いて、



(a) 既存の畳み込み



(b) Dilatad Convolution

図 14: WaveNet の畳み込み手法： (左) 既存の畳み込み, (右)Dilatad Convolution(飛ばし飛ばしで畳み込むことでより多くの受容野を入力とできる特徴がある) (https://arxiv.org/pdf/1609.03499.pdf より引用)

音声の生成を行う。Dilatad Convolution は、飛ばし飛ばしで畳み込むことで、より多くの受容野を入力とできる特徴がある。図 14a では、既存の畳み込みの受容野は、5 つのピクセルであるが、Dilatad Convolution を用いることで、受容野を 16 つのピクセルに拡張することができる (図 14b)。

■参考文献

1. 岡谷貴之/深層学習 改訂第 2 版 [機械学習プロフェッショナルシリーズ]/ 講談社サイエンティフィク/ 2022-01-17
2. 深層学習とゲーム理論 https://www.slideshare.net/slideshow/marl/234128570#6
3. 強化学習 星の本棚/ https://yagami12.hatenablog.com/entry/2019/02/22/210608 2024 年 6 月 21 日閲覧
4. 分散深層学習とモデル並列性 Preferred Networks https://tech.preferred.jp/ja/blog/model-parallelism-in-dnn
5. Cloud TPU の概要 Google Cloud https://cloud.google.com/tpu/docs/intro-to-tpu?hl=ja
6. 浮動小数点数型と誤差 京都産業大学 https://www.cc.kyoto-su.ac.jp/~yamada/programming/float.html
7. Quantization and Training of Low Bit-Width Convolutional Neural Networks for Object Detection (Yin et al.(2016) https://arxiv.org/abs/1612.06052)
8. ニューラルネットワークの全結合層におけるパラメータ削減手法の比較 佐々木など (2015)https://db-event.jpn.org/deim2017/papers/62.pdf
9. Why isn’ t DenseNet adopted as extensive as ResNet? https://cvinvolution.medium.com/why-isnt-densenet-adopted-as-extensive-as-resnet-1bee84101160
10. Group Normalization (Wu and He(2018)) https://arxiv.org/pdf/1803.08494.pdf
11. WAVENET: A GENERATIVE MODEL FOR RAW AUDI (Oord et al.(2016)) https://arxiv.org/pdf/1609.03499.pdf