Section1：強化学習

長期的に報酬を最大化できるように環境のなかで行動を選択できるエージェントを作ることを目標とする機械学習の分野の一つ。

行動の結果として与えられる利益(報酬)をもとに、行動を決定する原理を改善していく仕組み。

応用例

マーケティングの場合

環境	エージェント	行動	報酬
会社の販売促進部	プロフィールと購入履歴に基づいて、キャンペーンメールを送る顧客を決めるソフトウェア	顧客ごとに送信、非送信の 2 つの行動を選ぶ	キャンペーンのコストという負の報酬とキャンペーンで生み出されると推測される売上という正の報酬を受ける

探索と利用のトレードオフ

環境について事前に完璧な知識があれば、最適な行動を予測・決定することは可能であり、

応用例でのマーケティングの場合については、どのような顧客にキャンペーンメールを送信すると、どのような行動を行うのかが既知であれば予測できる。

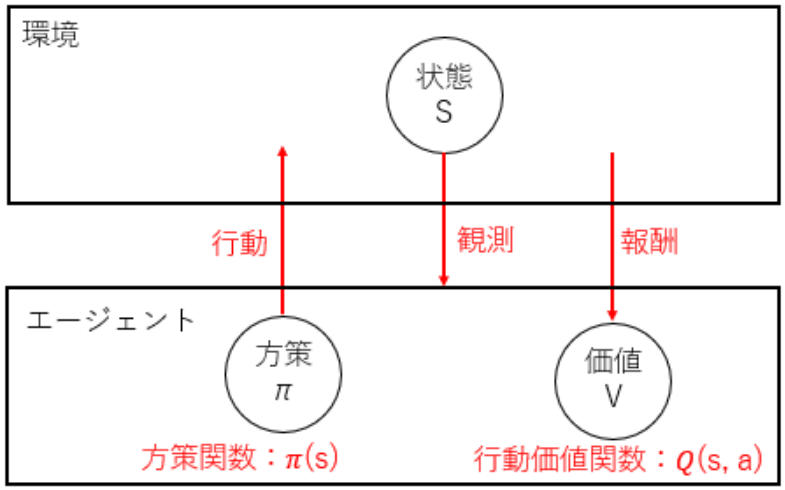
強化学習の場合は、この顧客の行動が既知であるという仮定は成り立たず、不完全な知識をもとに行動しながらデータを収集して、最適な行動を見つけることになる。

上記の最適な行動を見つける際に、下記の探索と利用のトレードオフが発生する。

① 過去のデータでベストとされる行動のみ常にとり続けた場合、他にもっとベストな行動があっても見つけることができない。
→探索が足りない状態。

② 未知の行動のみを常にとり続けた場合、過去の経験を生かすことができない。
→利用が足りない状態。

イメージ



差分

強化学習は教師あり学習、教師なし学習と目標が違う。

教師あり学習、教師なし学習は、データに含まれるパターンを見つけ出すおよびそのデータから予測することが目標であるのに対して、

強化学習では、優れた方策を見つけることが目標である。

歴史

強化学習には冬の時代があったが、計算速度の進展により大規模な状態を持つ場合の強化学習が可能となりつつある。

関数近似法と Q 学習を組み合わせる手法の登場により利用が広がっている。

Q 学習とは、行動価値関数を行動する毎に更新することにより学習を進める方法のこと。

関数近似法とは、価値関数や方策関数を関数近似する手法のこと。

価値関数

価値を表す関数としては、状態価値関数と行動価値関数の 2 種類がある。

状態価値関数とは、ある状態の価値に注目する関数。

行動価値関数とは、状態と価値を組み合わせた価値に注目する関数。

状態関数：$V^{\pi}(s)$

行動価値関数：$Q^{\pi}(s,a)$ = 状態 + 行動関数

s：状態、a：行動

$V^{\pi}(s)$と$Q^{\pi}(s,a)$からゴールまで方策を続けた時の報酬の予測値が得られる。

方策関数

方策ベースの強化学習手法において、ある状態でどのような行動を採るのかの確率を与える関数のこと。

$a = \pi(s)$

s：状態、a：行動

方策勾配法

方策反復法は方策をモデル化して最適化する手法。

下記の式で表される方策勾配法から最適化する。

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

$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}}\left[(\nabla_{\theta} log \pi_{\theta}(a \mid s) Q^{\pi}(s,a)\right]$

θ：重み、J：期待される報酬(= 方策の良さを表す)

$\varepsilon \nabla J(\theta)$を加算するのは、報酬に関する部分であり、報酬を大きくする方向に学習を進めるのが目的だからである。

Section2：AlphaGo

AlphaGo (Lee)

学習ステップ

1. 教師あり学習による RollOutPolicy と PolicyNet の学習

   KGS Go Server(ネット囲碁対局サイト)の棋譜データから 3000 万局面分の教師データを用意し、教師と同じ着手を予測できるように学習を行う。

   具体的には、教師が着手した手を 1 として残りを 0 とした 19×19 次元の亜飛列を教師データとして、それを分類問題として学習を行う。

   この学習で作成した PolicyNet は 57%ほどの精度。

2. 強化学習による PolicyNet の学習

   現状の PolicyNet と PolicyPool からランダムに選択された PolicyNet と対局シミュレーションを行い、その結果を用いて方策勾配法で学習を行う。

   PolicyPool とは、PolicyNet の強化学習の過程を 500Iteration ごとに記録し保存しておいたもの。

   現状の PolicyNet 同士の対局ではなく、PolicyPool に保存されているものとの対局を使用するのは、対局に幅を持たせて過学習を防ぐためである。

   この学習を mini batch size 128 で 1 万回行う。

3. 強化学習による ValueNet の学習

   PolicyNet を使用して対局シミュレーションを行い、その結果の勝敗を教師として学習を行う。

   教師データ作成の手順は以下となっている。

   ① まず SL PolicyNet(教師あり学習で作成した PolicyNet)で N 手まで打つ。

   ② 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 RL PolicyNet(強化学習で作成した PolicyNet)で終局まで打ち、その勝敗報酬を R とする。

   S(N+1)と R を教師データ対とし、損失関数を平均二乗誤差とし、回帰問題として学習を行う。

   この学習を mini batch size 32 で 5000 万回行う。

   N 手までと N+1 てからの PolicyNet を別々にしているのは、過学習を防ぐためであると説明されている。

RollOutPolicy

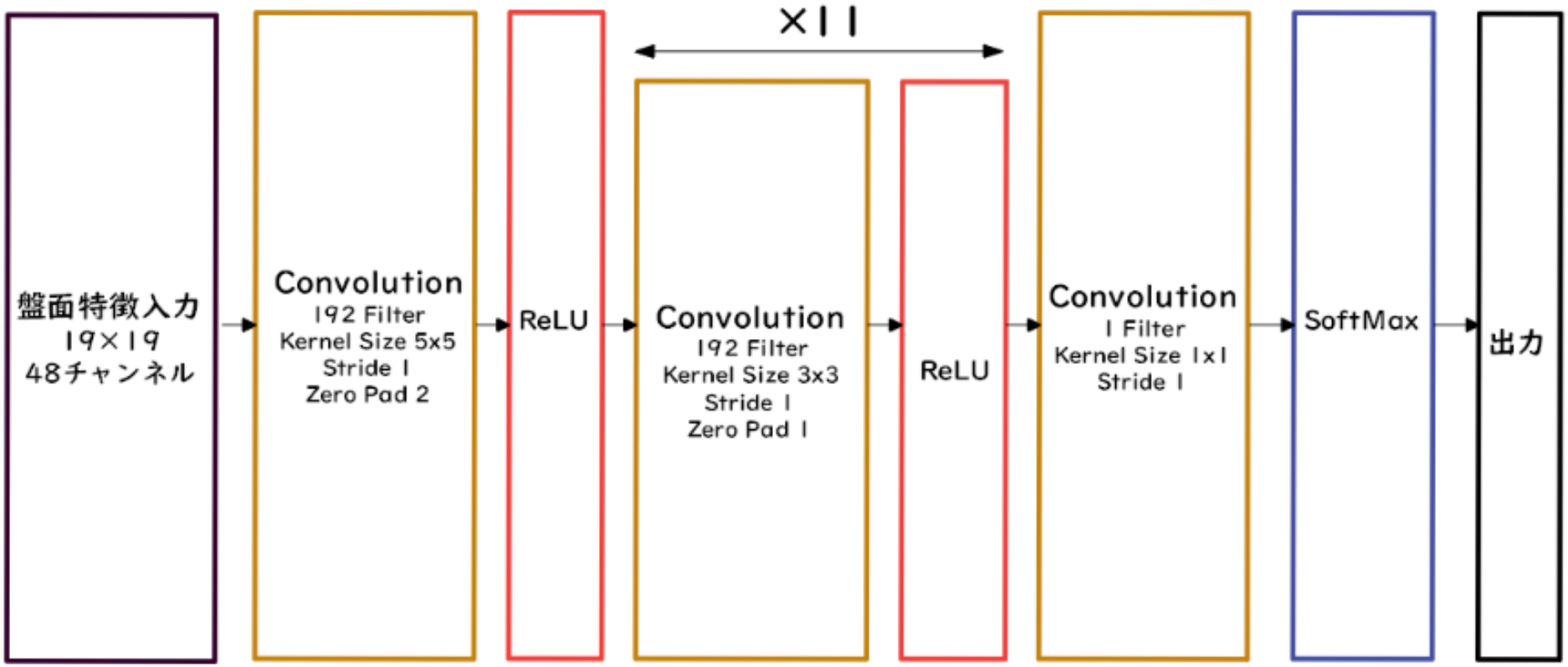
NNではなく線形の方策関数
探索中に高速に着手確率を出すために使用される

特徴	次元数	説明
１２近傍マッチ	1	一つ以上の12近傍パターン（下記）にマッチしたかどうか
アタリ救助	1	アタリを逃げる手であるかどうか
隣接	8	直前の着手と隣接したマスへの着手であるか
ナカデ	8192	ナカデのパターンにマッチするか
１２近傍（直前）	32207	直前の着手中心のダイアモンド型12近傍パターンにマッチするか
3x3近傍	69338	着手しようとしているマス中心の3×3近傍パターンにマッチするか
アタリ	1	アタリをつける手か
距離	34	直前の着手と着手しようとしているマスのマンハッタン距離
12近傍	69338	着手しようとしているマス中心のダイアモンド型12近傍パターンにマッチするか

赤字の特徴はRollOut時には使用されず、TreePolicyの初期値として使用されるときに使われる

上記の特徴が19×19マス分あり、出力はそのマスの着手予想確率となる

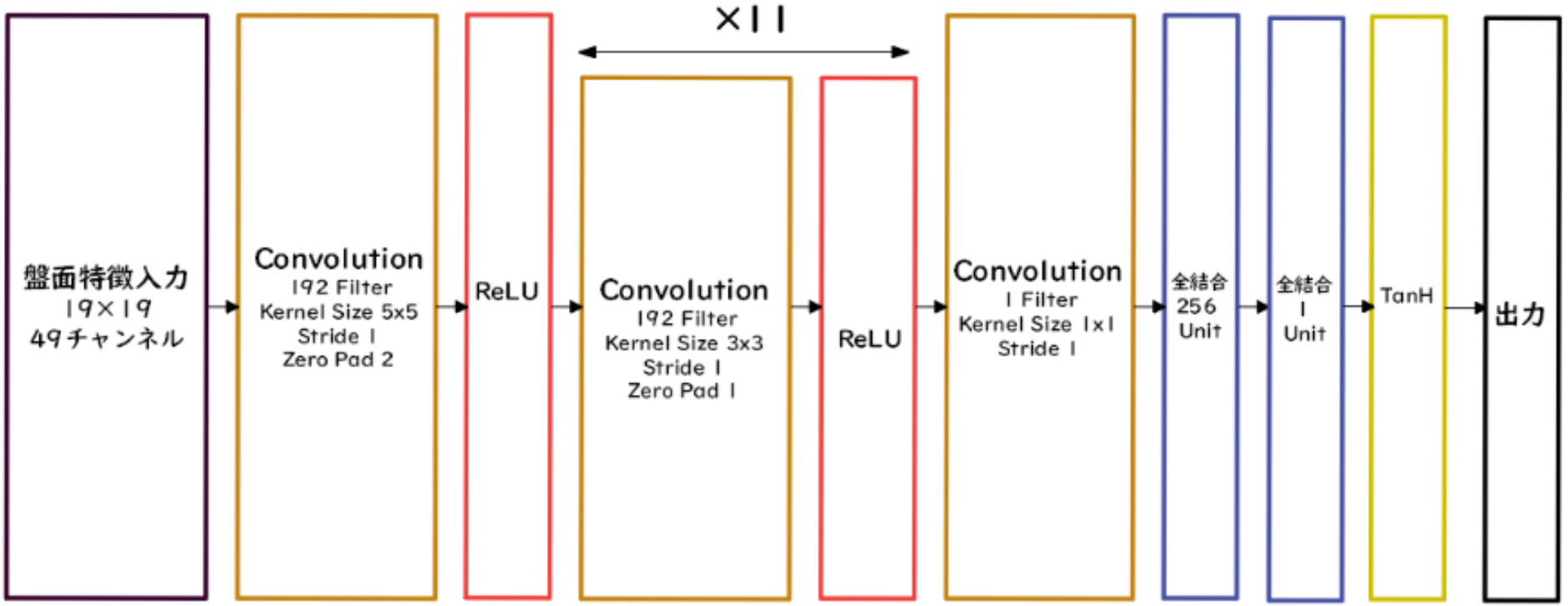
Alpha Go (Lee)のPolicyNet



SoftMax Layer 多次元を総和１の確率分布に変換するLayer

出力は19×19マスの着手予想確率が出力される

Alpha Go (Lee) のValueNet



TanH　Layer　-∞～∞の値を-1～1までに変換するLayer

出力は現局面の勝率を-1～1で表したものが出力される

Policy Net、Value Netの入力

特徴	チャンネル数	説明
石	3	自石、敵石、空白の３チャンネル
オール１	1	全面1
着手履歴	8	8手前までに石が打たれた場所
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たれた場合、シチョウで隣接連を取られるかどうか
合法手	1	合法手であるかどうか
オール0	1	全面0
手番	1	現在の手番が黒番であるか？（Policy Netにはこの入力はなく、ValueNetのみ）

モンテカルロ探索

コンピュータ囲碁ソフトでは現在もっとも有効とされている探索法。

他のボードゲームでは minmax 探索やその派生形の $\alpha\ \beta$ 探索を使うことが多いが、盤面の価値や勝敗予想値が必要となるが、囲碁では盤面の価値や勝敗予想を出すのが困難とされていた。

そこで、盤面評価値に頼らず末端評価値、つまり勝敗のみを使って探索を行うことができないかという発想で生まれた探索法である。囲碁の場合、他のボードゲームと違い最大手数はマスの数でほぼ限定されているため、末端局面に到達しやすい。

具体的には、現局面から末端局面まで PlayOut と呼ばれるランダムシミュレーションを多数回行い、その勝敗を集計して着手の優劣を決定する。

また、該当手のシミュレーション回数が一定数を超えたら、その手を着手したあとの局面をシミュレーション開始局面とするよう、探索木を成長させる。

この探索木の成長を行うというのがモンテカルロ木探索の優れているところ。

モンテカルロ木探索はこの木の成長を行うことで、一定条件下において探索結果は最善手を返すということが理論的に証明されている。

学習手法

選択、評価、バックアップ、成長という 4 ステップで構成される。

選択

Root 局面にて着手選択方策$\pi = Q(s,a) + cP(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$に従って手を選択する。$c$ は定数、$P(s, a)$は PolicyNet による選択確率、$N(s, a)$はその手が探索中に選ばれた数、$\Sigma N(s, b)$は現局面の全合法手が選ばれた数の合計。

$Q(s, a)$が現状の勝敗期待値、$cP(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$がバイアス項となり、基本的には勝敗期待値のより高い手を選択するが、選択数が少ない手には高いバイアスがかかり、選択されやすくなる。また、従来のモンテカルロ木探索では $P(s, a)$がバイアス項にそもそもないものや、人間が手作業で作ったヒューリスティックな方策評価が使われていたが、これに PolicyNet を使用するとしたのが AlphaGO の特徴の一つ。

この選択、着手を Leaf ノードに到着するまで行う。

評価

着手選択方策によって選ばれた手で進めた局面 sa が Leaf ノードであればその局面 sa を ValueNet で評価する。

また、局面 sa を開始局面とした末端局面までのシミュレーションを行い、勝敗値を出す。この時シミュレーション時には RollOut 方策を使用する。

バックアップ

評価フェイズで評価した値を積算する。局面 sa での ValueNet の評価の積算値 Wv、RollOut での勝敗値での積算値 Wr が積算され、N(s,A)と ΣN(s, b)が 1 加算される。それらの値から勝敗期待値が再計算される。これを root 局面までさかのぼって更新する。

$Q(s,a) = (1 - \lambda)\frac{Wv}{N(s,a)} + \lambda\frac{Wr}{N(s,a)}$とし、 λは 0≦ λ ≦1 の定数。AlphaGo (Lee)では 0.5 を使用。

成長

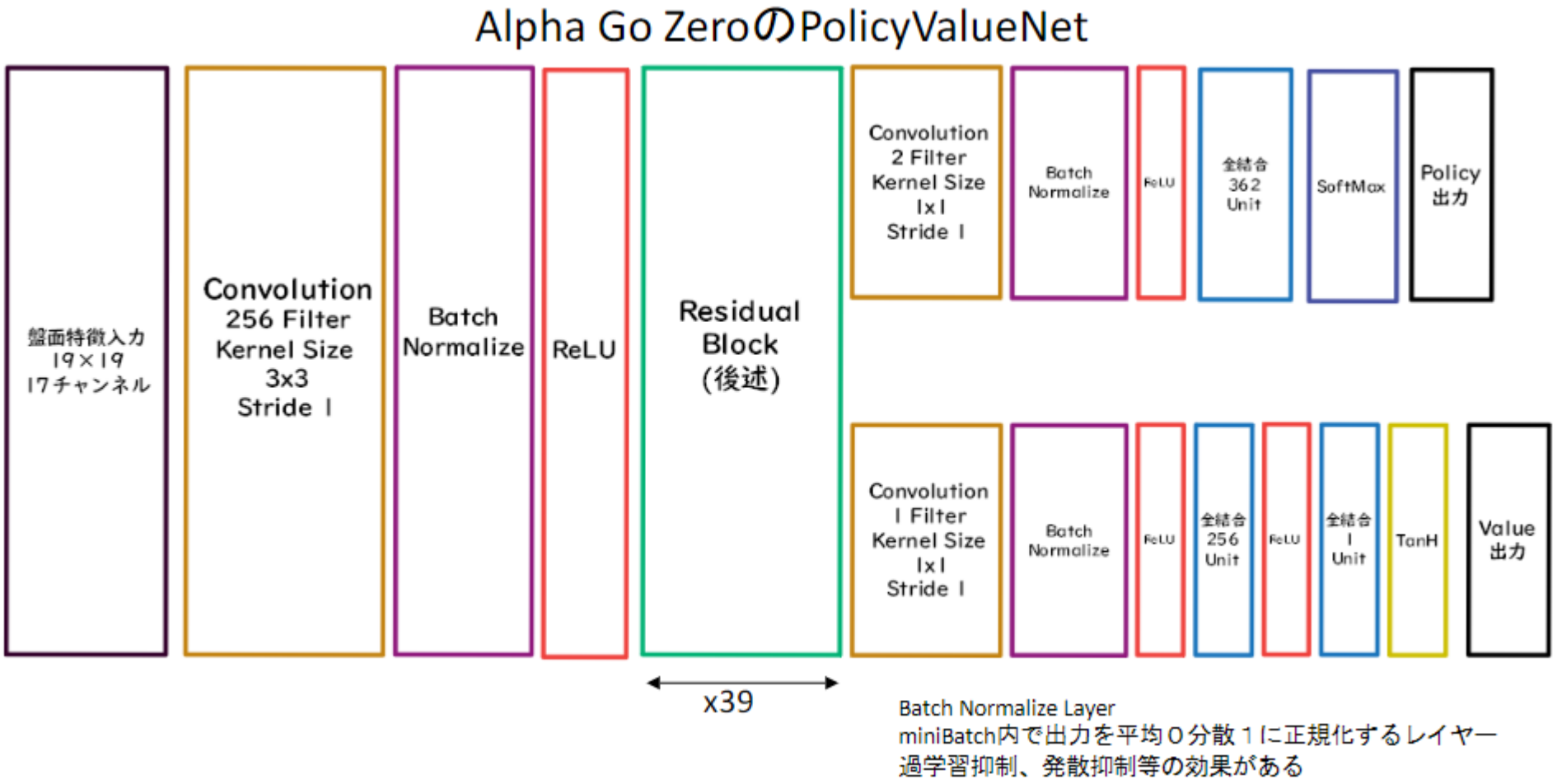
選択、評価、バックアップを繰り返し一定回数選択された手があったら、その手で進めた局面の合法手ノードを展開し、探索木を成長させる。

AlphaGo Zero

AlphaGo (Lee)との差異

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を排除し、石の配置のみ変更。
3. PolicyNet と ValueNet を 1 つのネットワークに統合。
4. モンテカルロ木探索から RollOut シミュレーションを削除。

PolicyValueNet



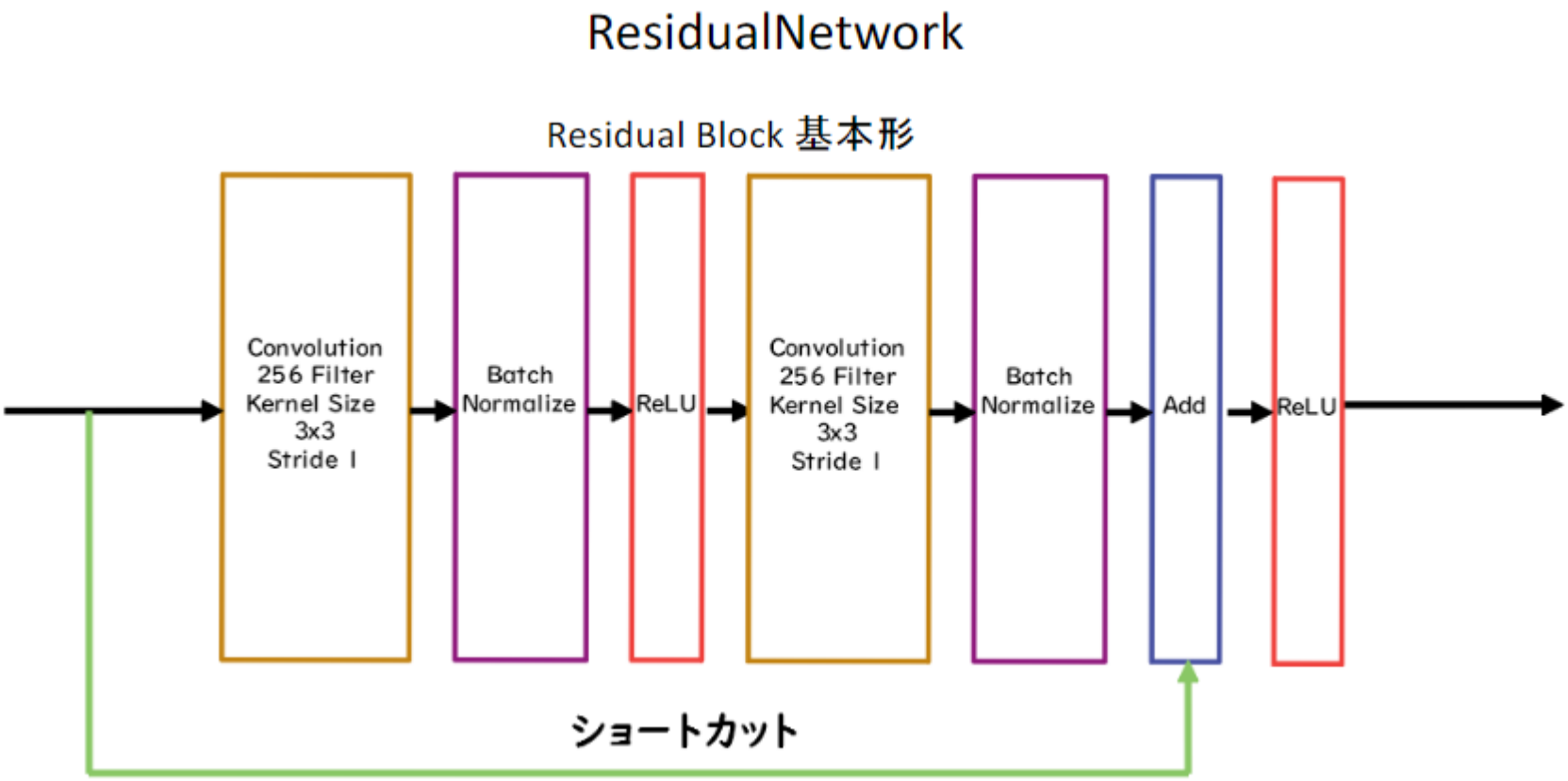
Residual Network

ネットワークにショートカット構造を追加して、勾配の爆発・消失を抑える効果を狙ったもの。

使用することにより、100 層を超えるネットワークでの安定した学習が可能となった。

基本構造は下図であり、Convolution→BatchNorm→ReLU→Convolution→BatchNorm→Add→ReLU までの Block を 1 単位にして積み重ねた形。

また、使用することで総数の違う Network のアンサンブル効果が得られているという説がある。



Residual Network の派生形

Residual Block の工夫

Bottleneck

1×1Kernel の Convolution を利用し、1 層目で次元削減を行って 3 層目で次元復元する 3 層構造にし、2 層のものと比べて計算量はほぼ同じだが 1 層増やせるメリットがあるとしたもの。

PreActivation

Residual Block の並びを BatchNorm→ReLU→Convolution→BatchNorm→ReLU→Convolution→Add とすることで性能が上昇したとするもの。

Network 構造の工夫

WideResNet

Convolution の Filter 数を k 倍にした ResNet。1 倍→k 倍×ブロック→2*k 倍 y ブロックと段階的に幅を増やしていくのが一般的。

Filter 数を増やすことにより、浅い層数でも深い層数のものと同等以上の精度となり、また GPU をより効率的に使用できるため学習も早い。

PyramidNet

WideResNet で幅が広がった直後の層に過度の負担がかかり精度を落とす原因となっているとし、段階的にではなく、各層で Filter 数を増やしていく ResNet。

Residual Network の性能

Residual Network を imageNet の画像分類に使ったときのネットワーク構造、およびエラーレートが以下となっている。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResidualNetworkの構造

出典：Kaiming He Deep Residual Learning for Image Recognition
arXiv.org 10 December 2015
https://arxiv.org/abs/1512.03385

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

ImageNetのエラー率

出典：Kaiming He Deep Residual Learning for Image Recognition
arXiv.org 10 December 2015
https://arxiv.org/abs/1512.03385

18, 34Layer のものは基本形、50、101、152Layer のものは BottleNeck 構造。

性能は従来の VGG-16 と比較して Top5 で 4%程度物エラー率の低減に成功している。

モンテカルロ木探索

選択、評価および成長、バックアップという 3 ステップで構成される。

選択

Root 局面にて着手選択方策$\pi = Q(s,a) + cP(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$に従って手を選択する。

選択された合法手 a で進めた局面が Leaf ノードでなければ、そのノードの着手選択方策に従って選択を行い、局面を進める。
選択された合法手 a で進めた局面が Leaf ノードであれば評価および成長ステップに移行する。

評価および成長

Leaf ノードまで進めた局面 sa を PolicyValueNet で評価する(RollOut は行わない)。
また、局面 sa の合法手ノードを展開し木を成長させる。

バックアップ

評価フェイズで評価した値を積算する。局面 sa での ValueNet の評価の積算値 Wv が積算され、N(s,A)と ΣN(s, b)が 1 加算される。

それらの値から勝敗期待値$Q(s,a) = \frac{Wv}{N(s,a)}$が計算される。これを Root 局面までさかのぼって更新する。

学習手法

自己対局による教師データの作成、学習、ネットワークの更新の 3 ステップで構成される。

自己対局による教師データの作成

現状のネットワークでモンテカルロ木探索を用いて自己対局を行う。
まず 30 手までランダムで打ち、そこから探索を行い勝敗を決定する。
自己対局中の各局面での着手選択確率分布と勝敗を記録する。
教師データの形は(局面、着手選択確率分布、勝敗)が 1 セットとなる。

学習

自己対局で作成した教師データを使い学習を行う。
Network の Policy 部分の教師に着手選択確率分布を用い、Value 部分の教師に勝敗を用いる。
損失関数は Policy 部分は CrossEntropy、Value 部分は平均二乗誤差。

ネットワークの更新

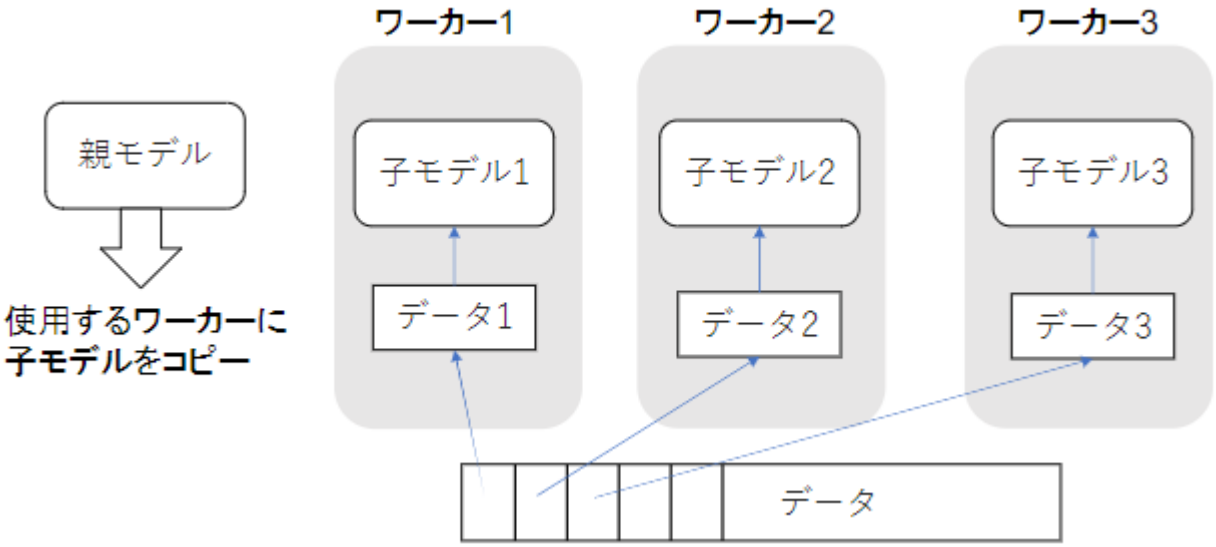
学習後、現状のネットワークと学習後のネットワークとで対局テストを行い、学習後のネットワークの勝率が高かった場合、学習後のネットワークを現状のネットワークとする。

Section3：軽量化・高速化技術

データ並列

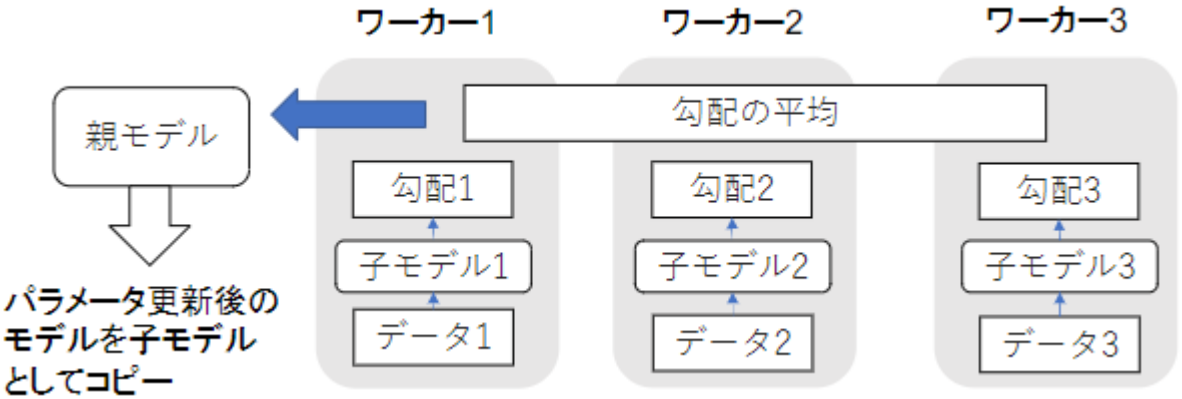
親モデルを各ワーカーに子モデルとしてコピーし、データを分割してワーカーごとに計算させる手法。

各モデルのパラメータの合わせ方で、同期型と非同期型の２つのタイプがある。



データ並列化：同期型

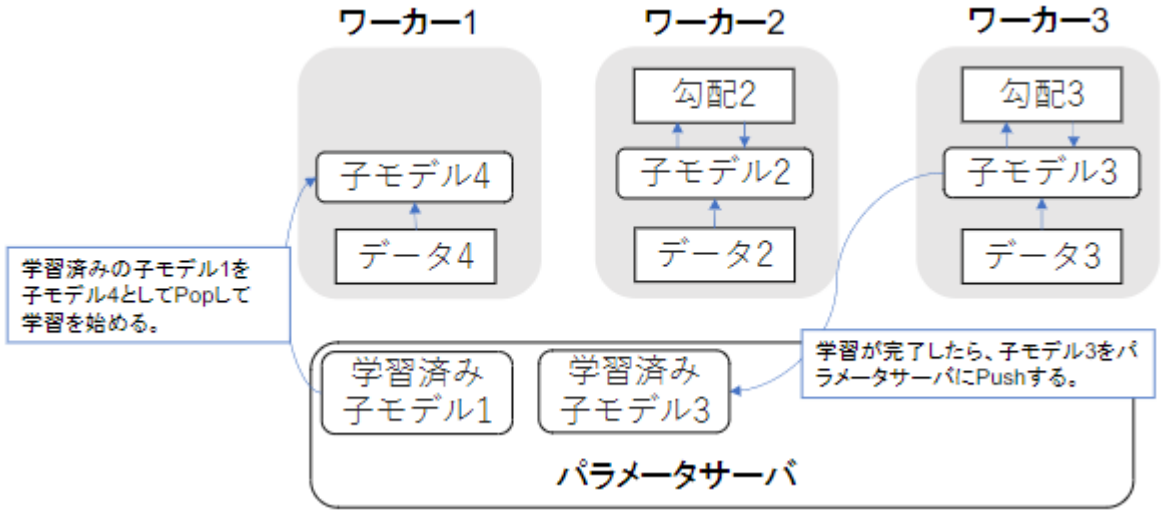
各ワーカーの計算が終わるのを待ち、前ワーカーの勾配が出たところで勾配の平均を計算し、親モデルのパラメータを更新する手法。



データ並列化：非同期型

各ワーカーはお互いの計算を待たず、子モデルごとに更新を行う手法。

学習が終わった子モデルはパラメータサーバに Push され、新たに学習を始める時は、パラメータサーバから Pop したモデルに対して学習を行う。



同期型と非同期型の比較

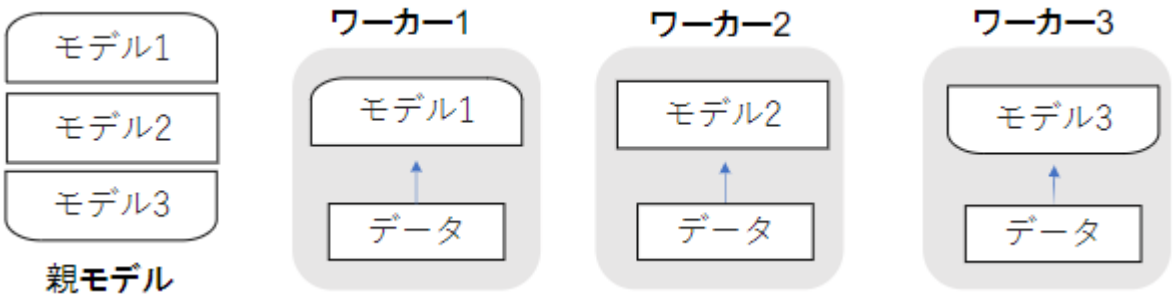
項目	同期型	非同期型	備考
処理スピード	×	○	お互いのワーカーの計算を待たないため、非同期型の方が早い。
安定性	○	×	最新のモデルのパラメータを利用できないため、非同期型の方が不安定となりやすい。
精度	○	×	現在は同期型の方が精度が良いことが多い。

モデル並列

親モデルを各ワーカーに分割し、それぞれのモデルを学習させる手法。

全てのデータで学習が終了した後で、一つのモデルに復元する。

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が良い。



モデル並列化とデータ並列化の選択

モデルとデータの大きさに着目して選択する。

モデルが大きい場合：モデル並列化

データが大きい場合：データ並列化

GPU

GPGPU(General-purpose on GPU)

元々の使用目的であるグラフィック以外の用途で使用される GPU の総称である。

CPU と GPU の比較

項目	CPU	GPU
コアの性能	高性能	比較的低性能
コア数	少数	多数
得意な処理	複雑で連続的な処理	簡単な並列処理

ニューラルネットの学習は単純な行列演算が多いため、GPU の方が高速化できる。

GPGPU 開発環境

Deep Learning フレームワーク(Tensorflow, Pytorch)内で以下の 2 つのプラットフォームが実装されているため、指定すればこれらを使用することができる。

CUDA

GPU 上で並列コンピューティングを行うためのプラットフォーム。
NVIDIA 社の GPU のみで使用可能。
Deep Learning 用に提供されており使いやすい。

OpenCL
オープンな並列コンピューティングのプラットフォーム。
NVIDIA 社以外の GPU でも使用可能。
Deep Learning 用の計算に特化していない。

軽量化
モデルの精度を維持しつつパラメータや演算回数を低減する手法の総称。
モデルの軽量化は、計算の高速化と省メモリ化を行うため、モバイル、IoT 機器と相性が良い。
代表的な手法として、量子化、蒸留、プルーニングの 3 つがある。

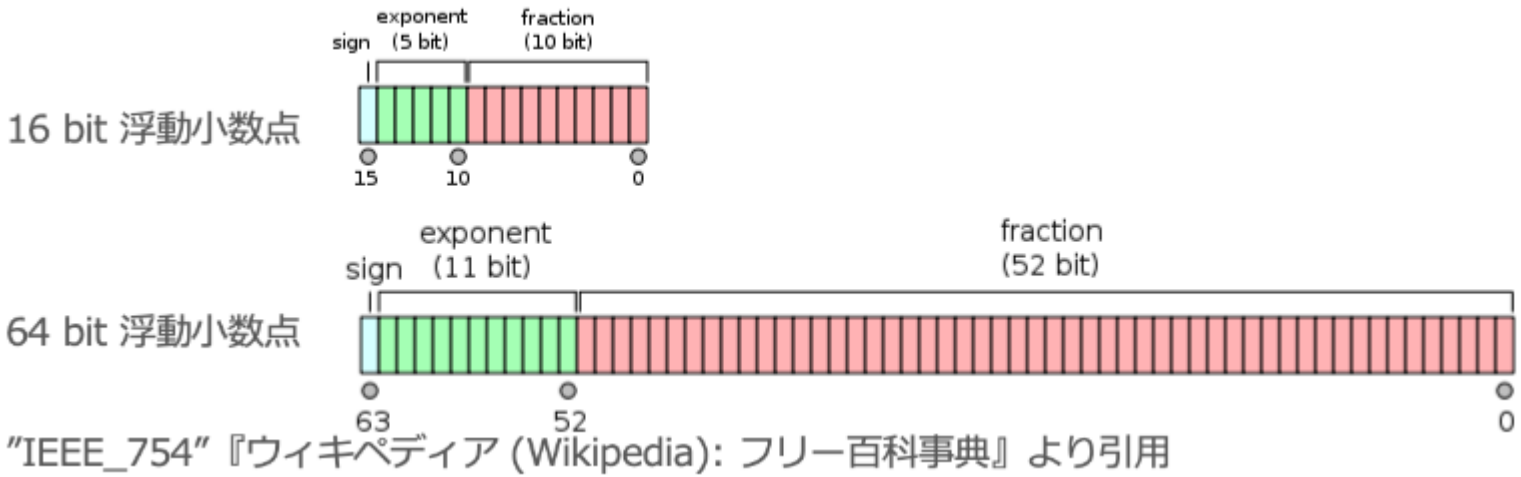
量子化
通常のパラメータの 64bit 浮動小数点を 32bit など下位の精度に落とすことでメモリと演算処理の削減を行う手法。
利点：計算の高速化や省メモリ化ができる。
欠点：精度が低下する。

64bit と 32bit での演算速度の比較
NVIDIA 社製の GPU では以下のような違いがあり、単精度(32bit)の方が約 2 倍の速度となっている。

GPU	単精度(32bit)	倍精度(64bit)
NVIDIA Tesla V100	15.7 TeraFLOPS	7.8 TeraFLOPS
NVIDIA Tesla P100	9.3 TeraFLOPS	4.7 TeraFLOPS

省メモリ化について
ニューロンの重みを浮動小数点の bit 数を少なくして有効桁数を下げることで、ニューロンのメモリサイズを小さくすることができ、多くのメモリを消費す

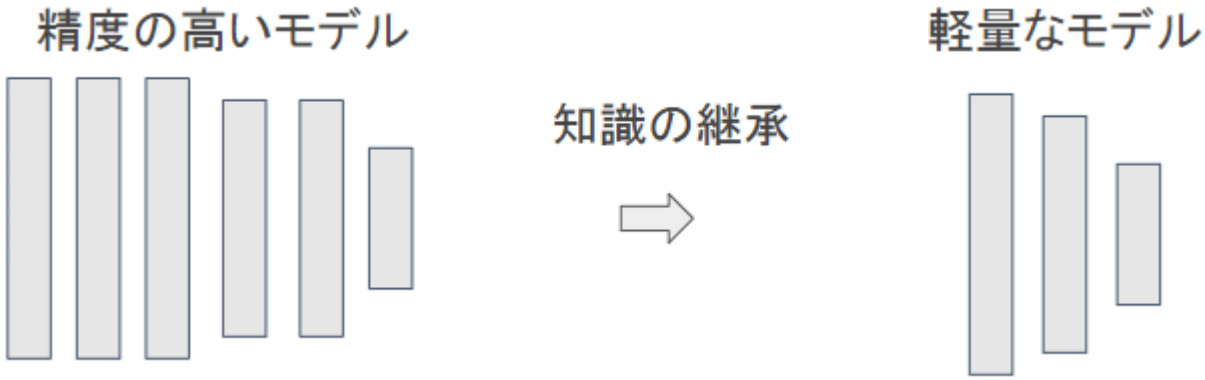
るモデルのメモリ使用料を抑えることができる。



蒸留

精度の高いモデルはニューロンの規模が大きなモデルとなっており、多くのメモリと演算処理が必要となる。

そのため、学習済みの精度の高いモデル(=規模の大きなモデル)の知識を使い軽量なモデルを作成する手法。



精度の高いモデルの知識を継承することで、軽量でありながら複雑なモデルに匹敵する精度を得ることが期待できる。

利点：少ない学習回数でより精度の良いモデルを作成することができる。

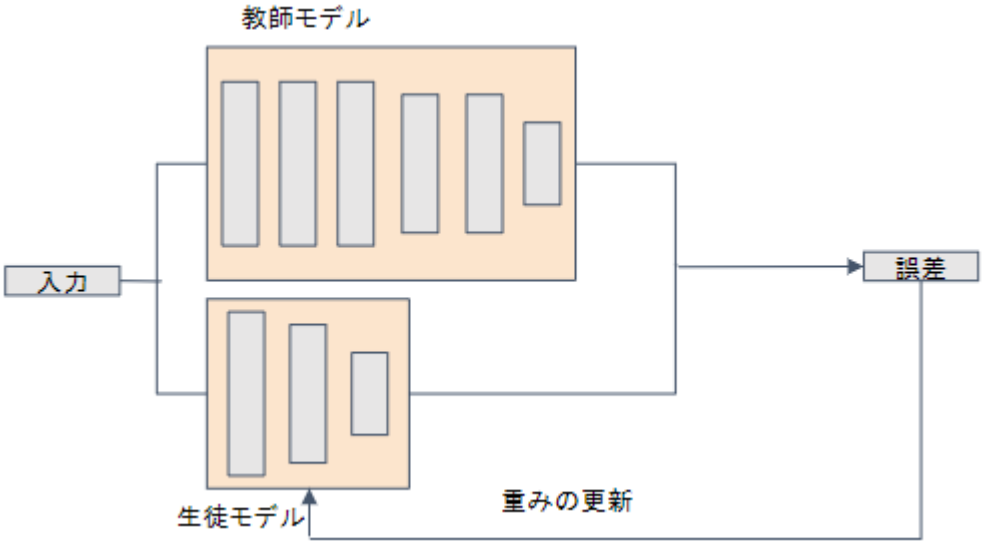
教師モデルと生徒モデル

蒸留は教師モデルと生徒モデルの 2 つで構成されている。

教師モデル：予測精度の高い、複雑なモデルやアンサンブルされたモデル。

生徒モデル：教師モデルをもとに作られる軽量なモデル。

教師モデルの重みを固定し生徒モデルの重みを更新し、誤差は教師モデルと生徒モデルのそれぞれの誤差を使い重み更新をする。

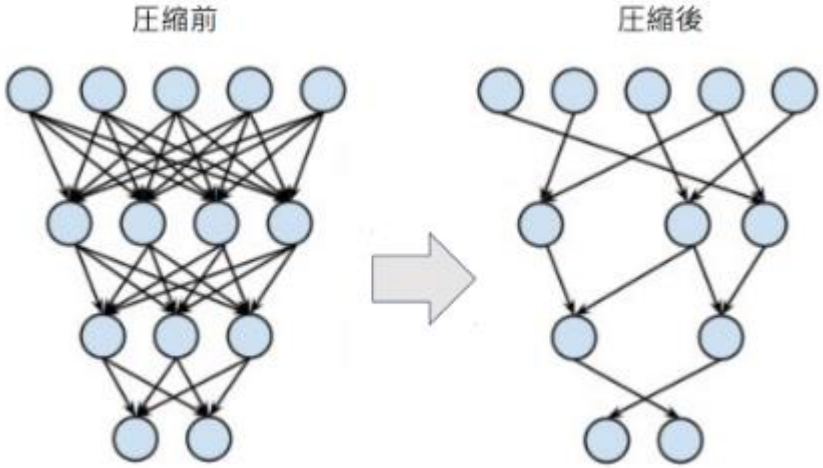


プルーニング

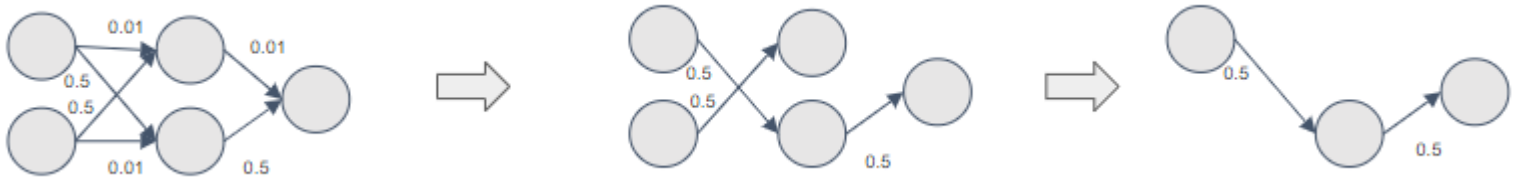
ネットワークが大きくなると大量のパラメータが全てのニューロンの計算に寄与するわけではないため、モデルの精度に寄与が少ないニューロンを削除することでモデルの軽量化、高速化を行う手法。

計算の高速化

寄与の少ないニューロンを削減してモデルの圧縮を行うことで計算の高速化ができる。



下図は、重みが 0.1 以下のニューロンを削減しており、重みが閾値以下の場合はニューロンを削減するという手法一般的である。



ニューロン数と精度

下図は Oxford 102 category ower dataset を CaffeNet で学習したモデルのプルーニングの閾値による各層と全体のニューロンの削減率と精度をまとめたものであり、閾値を高くするとニューロンは大きく削減できるが精度も減少していることがわかる。

	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

	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 (%)
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

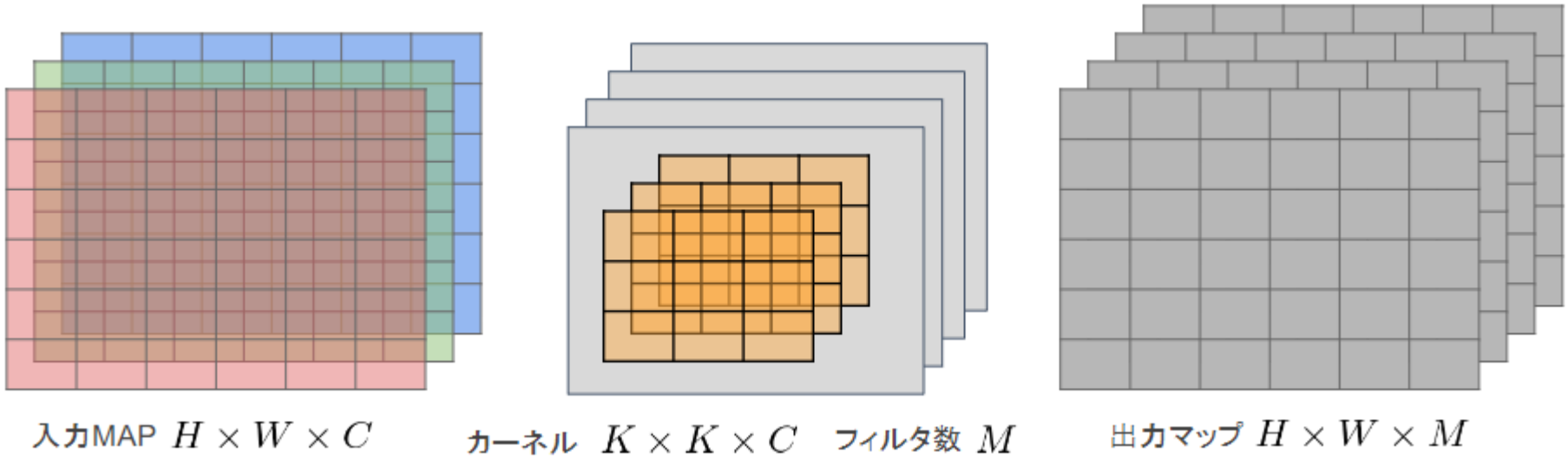
Section4：応用モデル

MobileNet

一般的な畳み込みレイヤーは計算量が多いため、Depthwise Convolution と Pointwise Convolution の組み合わせで軽量化を実現する手法。

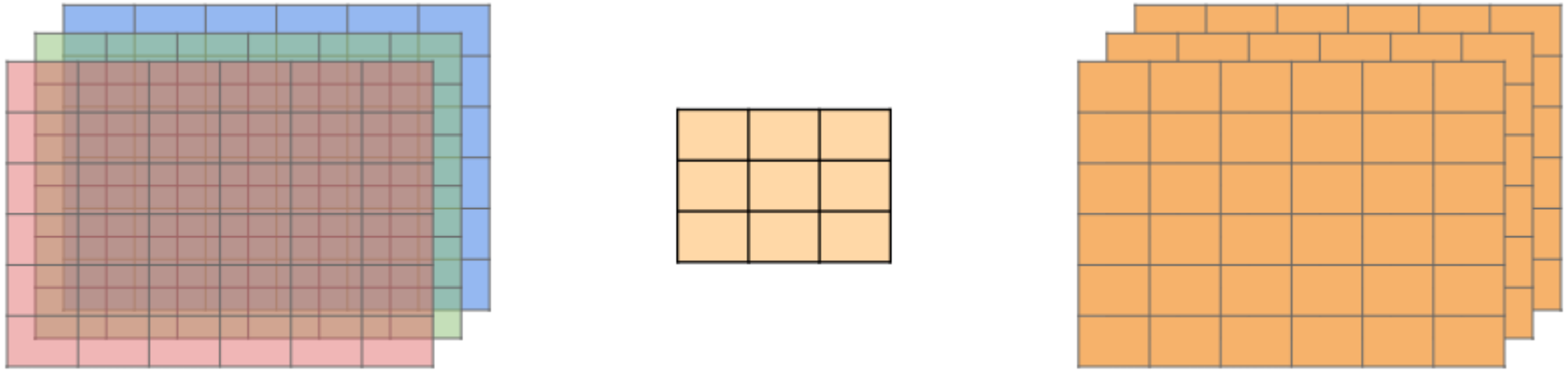
一般的な畳み込みレイヤー

入力特徴マップ　　：H×W×C
畳み込みカーネルのサイズ　　：K×K×C
出力チャネル数(フィルタ数)　　：M
出力マップの計算量　　：H×W×K×K×C×M



Depthwise Convolution

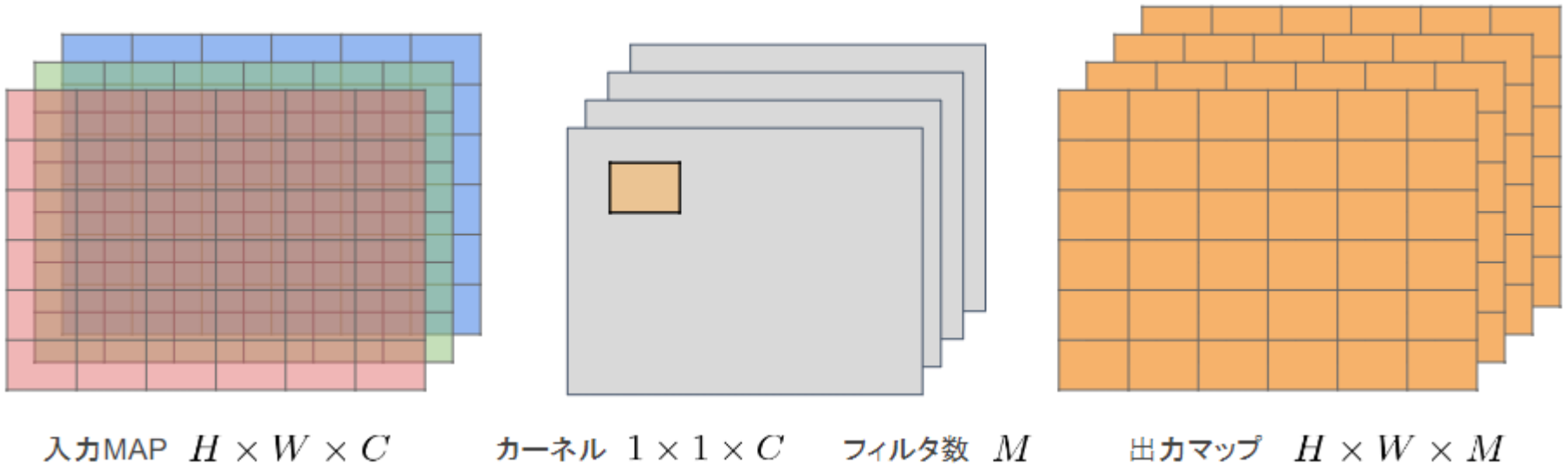
入力マップのチャネルごとに畳み込みを実施し、出力マップをそれらと結合する(入力マップのチャネル数と同じになる)。
通常の畳み込みカーネルは全ての層にかかっていることを考えると計算量が大幅に削減される。
各層ごとの畳み込みなので層間の関係性は全く考慮されないが、通常は PW 畳み込みとセットで使うことで解決できる。
出力マップの計算量：H×W×C×K×K



Pointwise Convolution

1×1 conv とも呼ばれており、入力マップのポイントごとに畳み込みを実施し、出力マップ(チャネル数)はフィルタ数分だけ作成可能(任意のサイズが指定可能)である。

出力マップの計算量：H×W×C×M



DenseNet

Dense Convolution Network(DenseNet)は、畳み込みニューラルネットワーク(CNN)アーキテクチャの一種であり、ニューラルネットワークでは層が深くなるにつれて、学習が難しくなるという問題があったが、Residual Network(ResNet)などの CNN アーキテクチャでは前方の層から後方の層へアイデンティティ接続を介してパスを作ることで問題対処している。DenseBlock と呼ばれるモジュールを用いた DenseNet もそのようなアーキテクチャの一つ。

DenseNet と ResNet の差異

DenseBlock では前方の各層からの出力全てが後方の層への出力として用いられるのに対して、RessidualBlock では前 1 層の入力のみ後方の層へ入力される。

成長率(Growth Rate)

DenseNet 内で使用される DenseBlock と呼ばれるモジュールでは成長率と呼ばれるハイパーパラメータが存在する。

DenseBlock 内の各ブロック毎に k 個ずつ特徴マップのチャネル数が増加してく時の k を成長率と呼んでいる。

正則化

BatchNorm

ミニバッチに含まれる sample の同一チャネルが同一分布(平均=0、分散=1)に従うように正規化する手法。

ニューラルネットワークにおいて、学習時間の短縮や初期値への依存低減、過学習の抑制などの効果がある。

問題点

Batch Size が小さい条件下では学習が収束しないことがある。

そのため、Layer Norm などの正規化手法が代わりに使われる。

LayerNorm

それぞれの sample の全ての pixels が同一分布に従うように正規化する手法。

バッチサイズに依存しないため、BatchNorm のバッチサイズが小さいと学習が収束しないという問題点が解消できている。

入力データや重み行列に対して、以下の操作を行っても出力が変わらない。

・入力データのスケールに関してロバスト

・重み行列のスケールやシフトに関してロバスト

InstanceNorm

各 sample のチャネルごとに同一分布に従うように正規化する手法。

コントラストの正規化に寄与しており、画像のスタイル転送やテクスチャ合成タスクなどで利用されている。

Wavenet

生の音声波形を生成する深層学習モデル。

Pixel CNN(高解像度の画像を精密に生成できる手法)を音声に応用したものであり、時系列データに対して畳み込み(Dilated convolution)を適用する。

畳み込み(Dilated convolution)

層が深くなるにつれて畳み込むリンクを離すことで、受容野を簡単に増やすことができるという利点がある手法。

下図では、Dilated=1,2,4,8 としている。

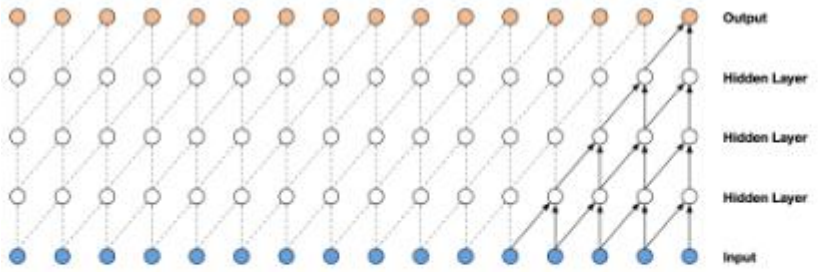


Figure 2: Visualization of a stack of causal convolutional layers.

既存の畳み込み

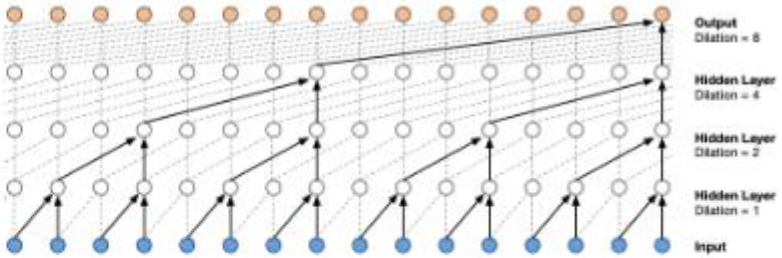


Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

畳み込み (Dilated)

<確認テスト 1>

Q. Depthwise separable Convolution という手法を用いて計算量を削減している。通常の畳込みが空間方向とチャネル方向の計算を同時に行うのに対して、Depthwise Separable Convolution ではそれらを Depthwise Convolution と Pointwise Convolution と呼ばれる演算によって個別に行う。

Depthwise Convolution はチャネル毎に空間方向へ畳み込む。すなわち、チャネル毎に $D_K \times D_K \times 1$ のサイズのフィルターをそれぞれ用いて計算を行うため、その計算量は(い)となる。

次に Depthwise Convolution の出力を Pointwise Convolution によってチャネル方向に畳み込む。すなわち、出力チャネル毎に $1 \times 1 \times M$ サイズのフィルターをそれぞれ用いて計算を行うため、その計算量は(う)となる。

A. (い)：H×W×C×K×K、(う)H×W×C×M
Depthwise Convolution の出力マップの計算量は H×W×C×K×K であるため、(い)にはこの値が入る。
Pointwise Convolution の出力マップの計算量は H×W×C×M であるため、(う)にはこの値が入る。

<確認テスト 2>

Q. 深層学習を用いて結合確率を学習する際に、効率的に学習が行えるアーキテクチャを提案したことが WaveNet の大きな貢献の 1 つである。
提案された新しい Convolution 型アーキテクチャは(あ)と呼ばれ、結合確率を効率的に学習できるようになっている。
(あ)を用いた際の大きな利点は、単純な Convolution Layer と比べて(い)ことである。

(あ)の選択肢
(1) Dilated causal convolution
(2) Depthwise separable convolution
(3) Pointwise convolution
(4) Deconvolution

(い)の選択肢
(1) パラメータ数に対する受容野が広い
(2) 受容野あたりのパラメータ数が多い
(3) 学習時に並列計算が行える
(4) 推論時に並列計算が行える

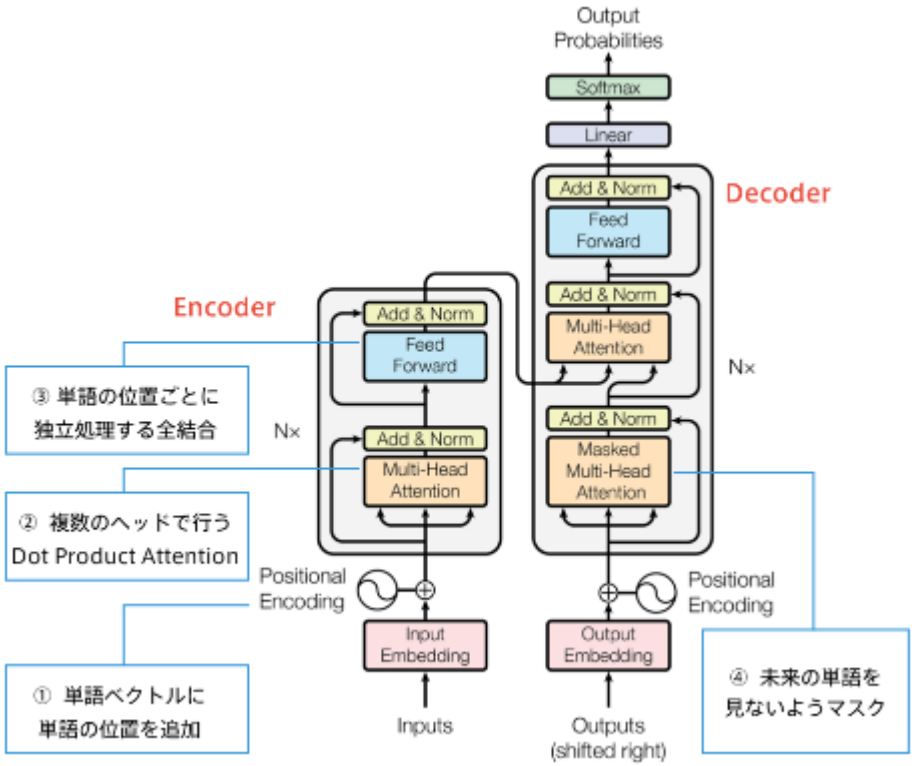
A. (あ)：(1) Dilated causal convolution、(い)：(1) パラメータ数に対する受容野が広い
WaveNet の Convolution は、Dilated convolution と呼ばれる畳み込みであるため(1)が正解。
Dilated convolution は、層が深くなるにつれて畳み込むリンクを離すことで、既存の畳み込みとは異なり、層が深くなるほど受容野が広くなるようになっている。

Section5：Transformer

Transformer は 2017 年 6 月に登場した RNN を使わない必要なのは Attention だけという深層学習モデル。

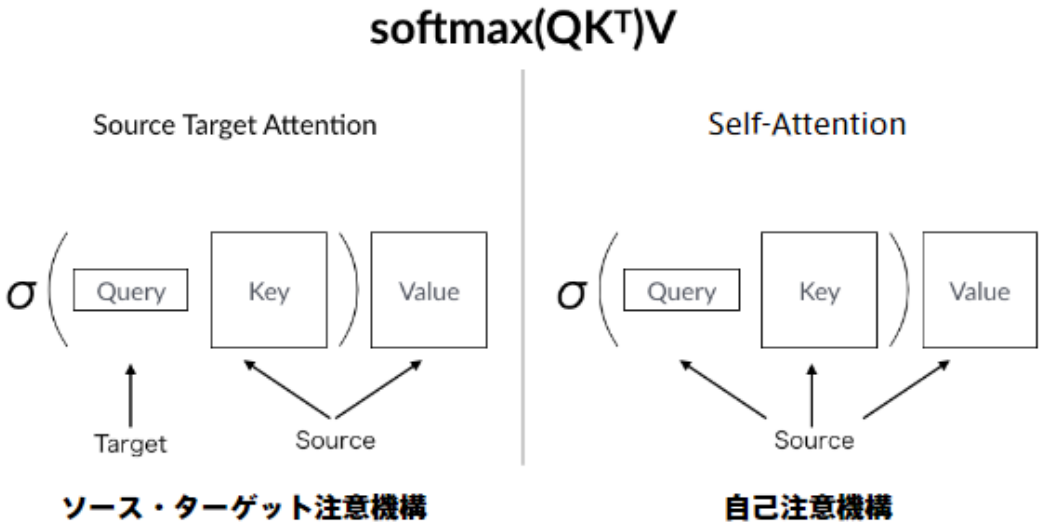
当時の SOTA をはるかに少ない計算量で実現した(英仏(3600 万文)の学習を 8GPU で 3.5 日で完了)。

モジュール構成



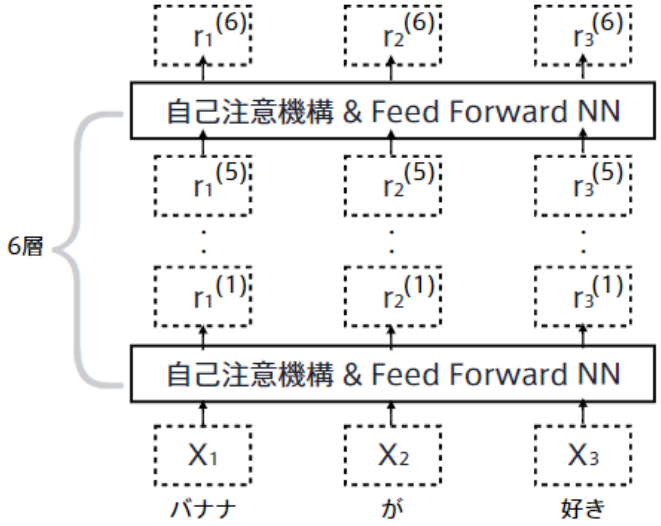
Attention

注意機構にはソース・ターゲット注意機構と自己注意機構の 2 種類ある。



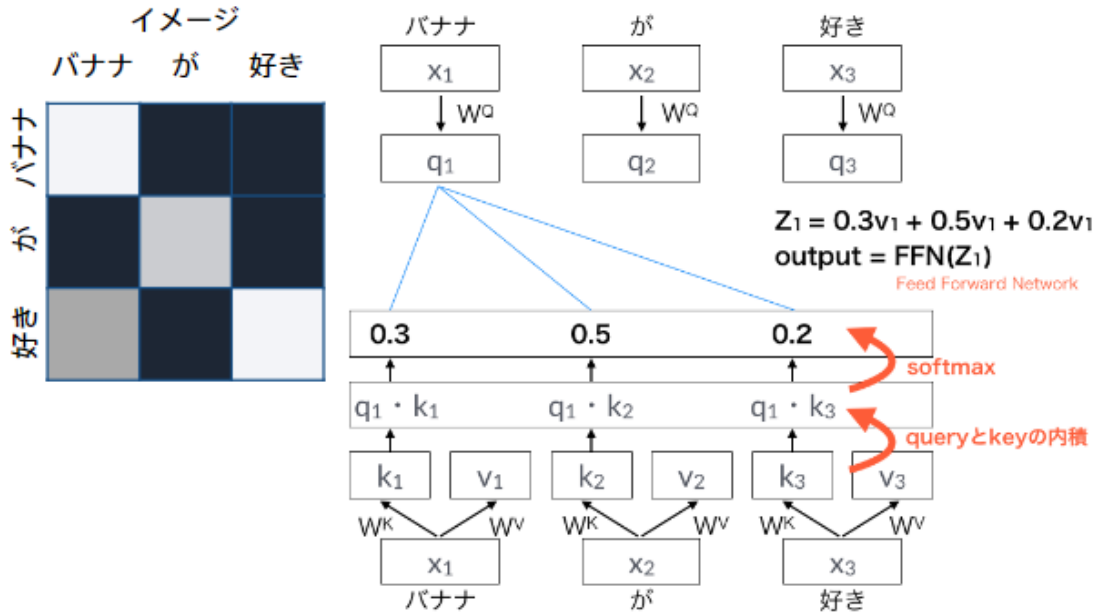
Transformer-Encoder

自己注意機構により文脈を考慮して各単語をエンコードする。



Self-Attention

入力を全て同じにして学習的に注意箇所を決めていく。



Position-Wise Feed-Forward Networks

位置情報を保持したまま順伝播させる。

各 Attention 層の出力を決定。

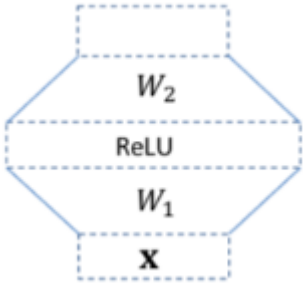
2 層の全結合 NN

線形変換→ReLU→線形変換

$FFN(x) = \max(0, xW_1 + b_1)\,W_2 + b_2$

$W_1 \in \mathbb{R}^{512\times2048}$ $b_1 \in \mathbb{R}^{2048}$

$W_2 \in \mathbb{R}^{2048\times512}$ $b_2 \in \mathbb{R}^{512}$



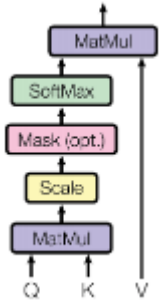
Scaled dot product attention

全単語に関する Attention をまとめて計算する。

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$Q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}、K = \begin{bmatrix} k_1 & k_2 & k_3 \end{bmatrix}、QK = \begin{bmatrix} q_1k_1 & q_1k_2 & q_1k_3 \\ q_2k_1 & q_2k_2 & q_2k_3 \\ q_3k_1 & q_3k_2 & q_3k_3 \end{bmatrix}$$

$$softmax(QK) = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.3 & 0.3 \\ 0.8 & 0.1 & 0.1 \end{bmatrix}、V = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}、softmax(QK^T)V = \begin{bmatrix} 0.1v_1 & 0.2v_1 & 0.3v_1 \\ 0.4v_2 & 0.3v_2 & 0.3v_2 \\ 0.8v_3 & 0.1v_3 & 0.1v_3 \end{bmatrix}$$

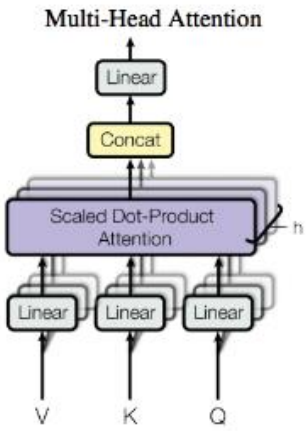


Multi-Head Attention

重みの異なる 8 個のヘッドを使用する。

8 個の Scaled Dot-Product Attention の出力を Concat。

それぞれのヘッドが異なる種類の情報を収集する。



Decoder

Encoder と同じく 6 層から成る。

各層で 2 種類の注意機構があり、注意機構の仕組みは Encoder とほぼ同じである。

自己注意機構

生成単語列の情報を収集し、直下の層への出力へのアテンション。

未来の情報を見ないようにマスクしている。

Encoder-Decoder-attention

入力文の情報を収集し、Encoder の出力へのアテンション。

Add&Norm

Add(Residual Connection)

入出力の差分を学習させる。

実装上は出力に入力をそのまま加算するだけである。

効果：学習・テストエラーが低減される。

Norm(Layer Normalizatin)

各層においてバイアスを除く活性化関数への入力を平均 0、分散 1 に正則化する。

効果：学習が高速化される。

Position Encoding

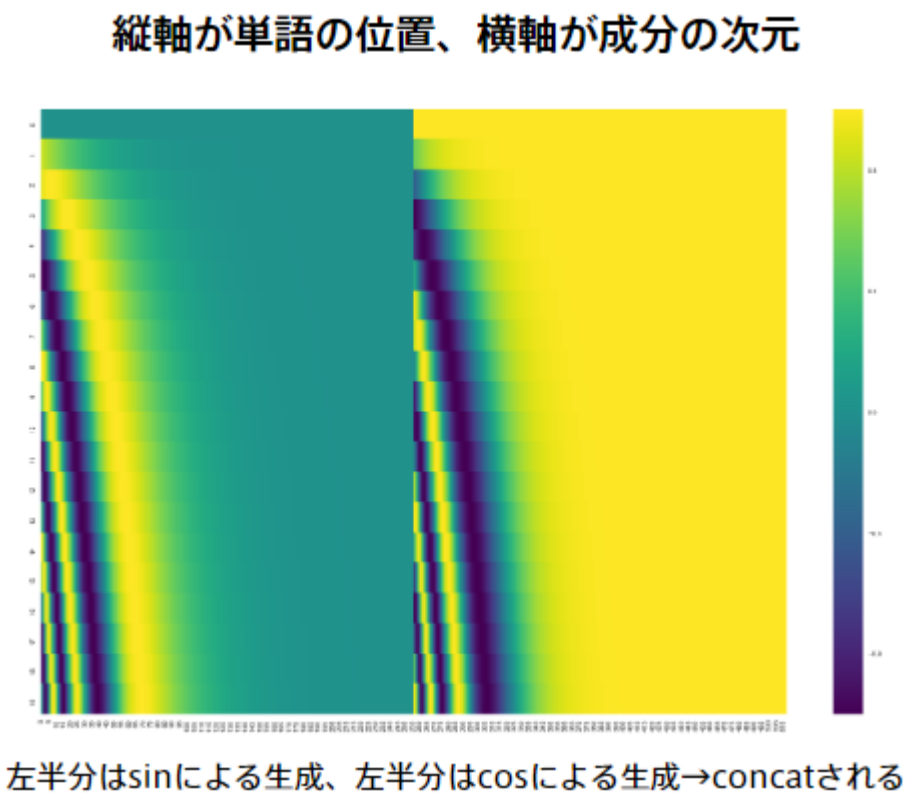
RNN を用いないので単語列の語順情報を追加する必要がある。

単語の位置情報をエンコード

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{512}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{512}}}\right)$$

Pos の(ソフトな)2 進数表現



<実装演習>

lecture_chap1_exercise_public.ipynb

lecture_chap2_exercise_public.ipynb

Section6：物体検知・セグメンテーション

入力

画像(カラー・モノクロのどちらでも良い)

出力

種類	説明	物体の位置	インスタンスの区別
分類	(画像に対して単一または複数の)クラスラベル	興味なし	興味なし
物体検知	Bounding Box [bbox/BB]	興味あり	興味なし
意味領域分割	(各ピクセルに対して単一の)クラスラベル	興味あり	興味なし
個体領域分割	(各ピクセルに対して単一の)クラスラベル	興味あり	興味あり

タスクの難易度は、分類<物体検知<意味領域分割<個体領域分割である。

代表的データセット

物体検出のコンペティションで用いられた VOC12、ILSVRC17、MS COCO18、OICOD18 などがある。

データセット	クラス	Train+Val	Box/画像	Instance Annotation	画像サイズ
VOC12	20	11,540	2.4	○	470×380
ILSVRC17	200	476,668	1.1	×	500×400
MS COCO18	80	123,287	7.3	○	640×480
OICOD18	500	1,743,042	7.0	○	一様ではない

VOC12(Visual Object Classes12)

PASCAL VOC Object Detection Challenge というコンペで用いられた。

主要貢献者が 2012 年に亡くなりコンペも終了したためさ新バージョンが VOC12 となっている。

ILSVRC17(ImageNet Scale Visual Recognition Challenge17)

コンペ自体は 2017 年に終了しているが、後継のコンペとして Open Images Challenge がある。

ImageNet(21,841 クラス/1400 万枚以上)のサブセットである。

MS COCO18(MS Common Object in Context Object Detection Challenge18)

物体位置推定に対する新たな評価指標を提案している。

OICOD18(Open Images Challenge Object Detection18)

ILSVRC や MS COCO とは異なる annotation process となっている。

Open Images V4(6000 クラス以上/900 万枚以上)のサブセットである。

Box/画像

値が小さい場合：アイコン的な映りであり、日常感とはかけ離れやすい。

値が大きい場合：部分的な重なりなども見られ、日常生活のコンテキストに近い。

目的に応じた Box/画像の値に対応したデータセットを選択する。

クラス数について

クラス数が大きい場合、同じようなものが別なものとして区別されるため、単純にクラス数が大きいほど良いわけではない。

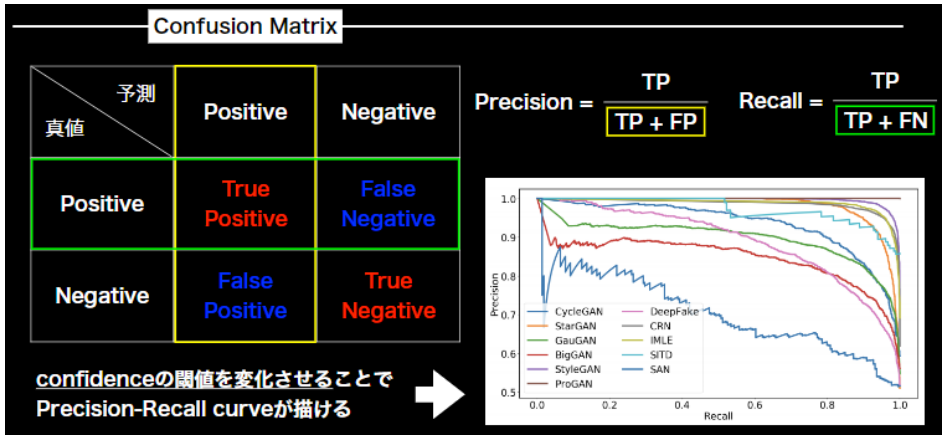
評価指標

分類問題における評価指標の復習

Accuracy　：全データ中正しく判定した確率($\frac{TP+TN}{TP+FN+FP+TN}$)

Precision　：Positive と予測したデータについて、実際に Positive である確率

Recall　　：Positive なデータについて、正しく予測できる確率

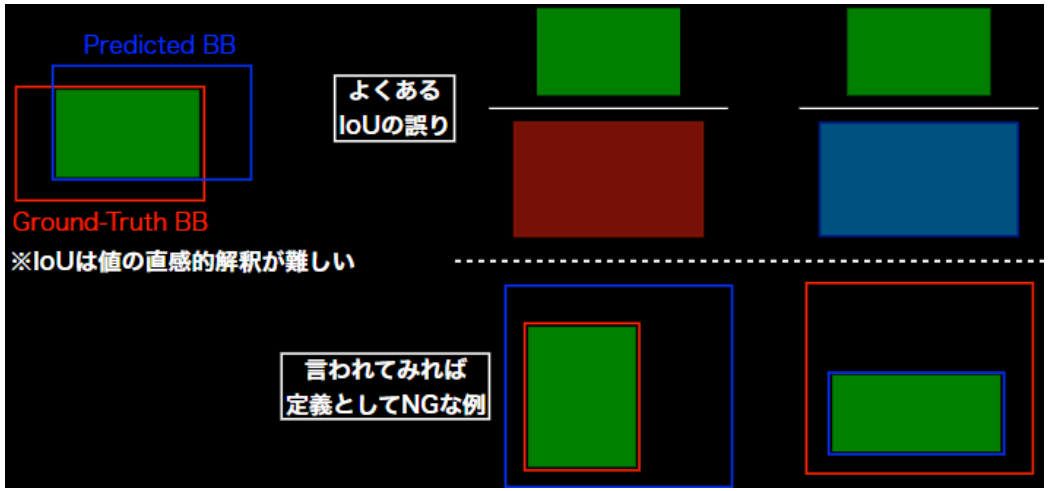


IoU(Intersection over Union)

物体検出においてはクラスラベルだけでなく、物体位置の予測精度も評価したいことから生まれた。

Jaccard 係数とも呼ばれている。

IoU = $\frac{TP}{TP+FP+FN}$



AP：Average Precision

Conf の閾値をβとするとき、

Recall = R(β)

Precision = P(β)

P = f(R)

$AP = \int_0^1 P(R)dR$　PR 曲線の下側の面積

※厳密には各 Recall のレベルに対して最大の Precision にスムージング。

※積分は Interpolated AP として有限点で計算される。

mAP：mean Average Precision

AP の平均(AP はクラスごとに計算される)

$mAP = \frac{1}{C}\sum_{i=1}^{C} AP_i$

C：クラス数

mAP COCO

MS COCO で導入された指標。

例：0.5 から 0.95 まで 0.05 刻みで AP&mAP を計算して算術平均を計算する。

$$mAP_{coco} = \frac{mAP_{0.5} + mAP_{0.55} + \cdots + mAP_{0.95}}{10}$$

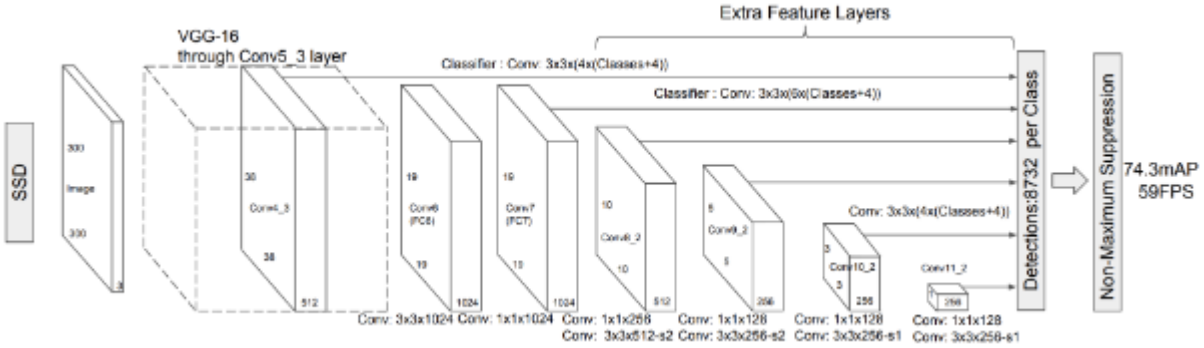
FPS：Flmes per Second

検出精度に加えて検出速度も問題となる。

SSD：Single Shot Detector

R-CNN ではバウンディングボックスを色々と動かしてそのたびに CNN による演算を行っているので、1 枚の画像から物体検出を行うのにかなりの処理時間を要していた。SSD では”Single Shot”という名前が暗示しているように、1 度の CNN 演算で物体の「領域候補検出」と「クラス分類」の両方を行う。これにより物体検出処理を高速化できる。

SSD のネットワークアーキテクチャ



Semantic Segmentation の概略

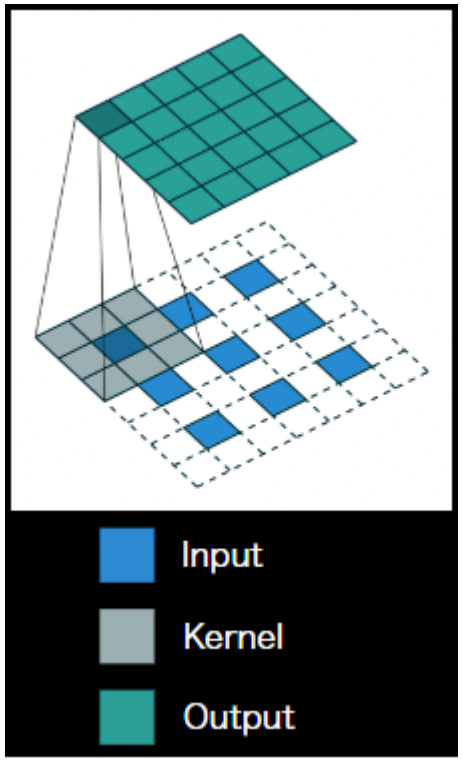
Deconvolution/Transposed convolution

通常の Conv.層と同様、kernel size、padding、stride を指定。

処理手順

1. 特徴マップの pixel 間隔を stride だけ空ける。
2. 特徴マップのまわりに(kernel size – 1) – padding だけ余白を作る。
3. 畳み込み演算を行う。

下図は keernel size = 3、padding=1、stride=1 の Deconv.により 3×3 の特徴マップが 5×5 に Up-sampling される様子。

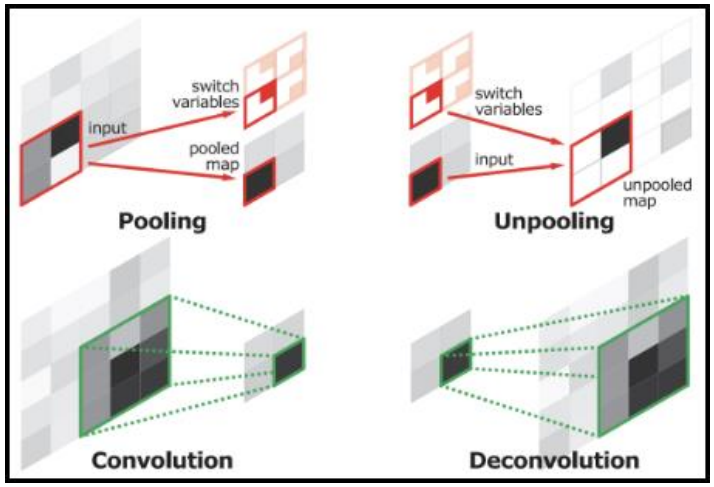


逆畳み込みと呼ばれることも多いが畳み込みの逆演算ではないことに注意が必要。

当然、pooling で失われた情報が復元されるわけではない。

Unpooling

プーリングした時の位置情報を保持しておく方法。



Dilated Convolution

Convolution の段階で受容野を広げる工夫。