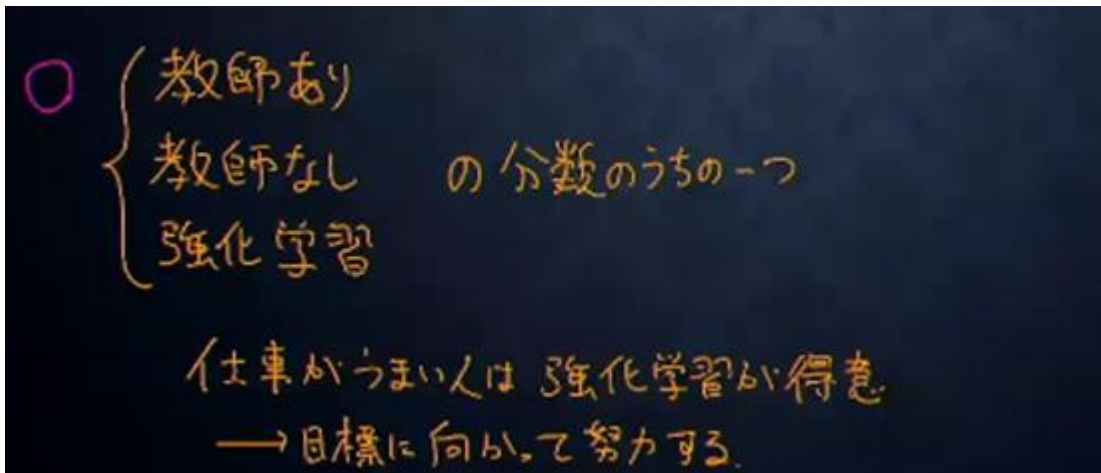


1. section1_強化学習

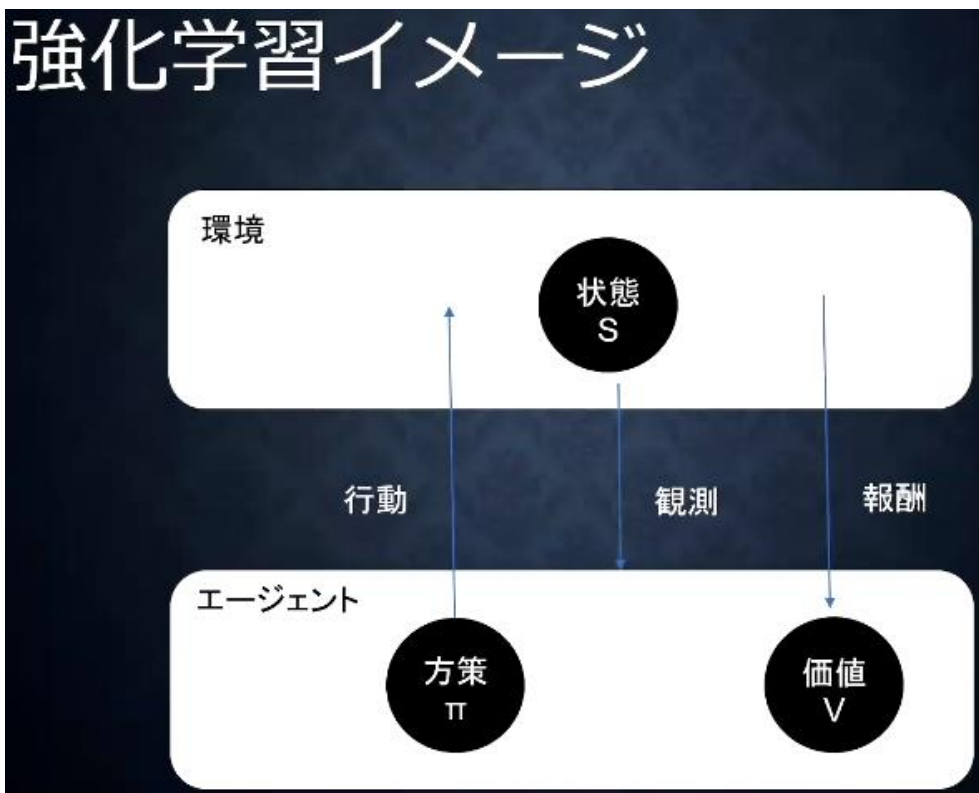
1. 1 強化学習とは

長期的に報酬を最大化できるように環境の中で行動を選択できるエージェントを作ること为目标とする機械学習の1分野である。

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



強化学習イメージ



1. 2 強化学習の応用例

マーケティングの場合

環境：会社の販売促進部

エージェント：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

行動：顧客ごとに送信、非送信の2つの行動を選ぶことになる。

報酬：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。

1. 3 探索と利用のトレードオフ

(1) 環境について事前に完璧な知識があれば、最適な行動を予測し決定することは可能。

→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既知であるが状況。

強化学習の場合、上記仮定は成り立たないとする。不完全な知識を元に行動しながら、データを収集。最適な行動を見つけていく。

(2) 過去のデータで、ベストとされる行動のみを常に取り続けければ他にももっとベストな行動を見つけることはできないか

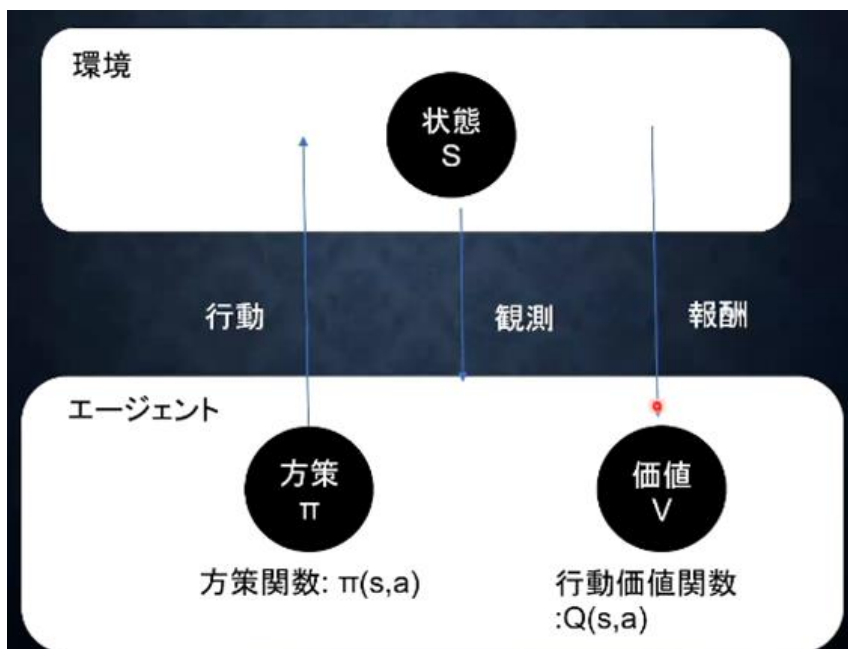
→探索が足りない状態

トレードオフの関係性

未知の行動のみを常に取り続けければ、過去の経験が生かせない。

→利用が足りない状態

1. 4 強化学習のイメージ



方策関数

行動価値関数

1. 5 強化学習の差分

- (1) 強化学習と通常の教師あり、教師なし学習の違い。

結論：目標が違う

①教師あり、なし学習では、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

②強化学習では、優れた方策を見つけることが目標
方策を探す。

- (2) 強化学習について

①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を可能としつつある。

②関数近似法、Q 学習を組み合わせる手法の登場

- (3) Q 学習

①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。

- (4) 関数近似法

①価値関数ら方策関数を関数近似する手法のこと。

入力が有ったら値を返す。表を作ってやっていたが関数で行う。機械学習する。

1. 6 行動価値関数

- (1) 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：ある状態に注目する場合

行動価値関数：状態と価値を組み合わせた価値に注目する場合

1. 7 方策関数

- (1) 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かの確率を与える関数のことである。

方策関数: $\pi(s) = a$

関数の関係

エージェントは方策に基づいて行動する

$\pi(s, a)$: V や Q を基にどういう行動をとるか \Rightarrow その瞬間、その瞬間の行動をどうするか
 \Rightarrow 経馬金を活かす or チェンソ切る など.

$\left\{ \begin{array}{l} V(s): \text{状態関数} \\ Q(s, a): \text{状態 + 行動関数} \end{array} \right\}$ $\left\{ \begin{array}{l} \text{ゴールまで今の方策を続けたときの} \\ \text{報酬州の予測値が得れる} \end{array} \right.$

\Rightarrow やり続けたら最終的にどうなるか.

1. 8 方策勾配法

(1) 方策反復法

方策をモデル化して最適化する手法

→ 方策勾配法

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

J とは方策の良さ → 定義しなければならない。

関数が出てきたらニューラルネットワークを使う。

方策勾配法

$$\pi(s, a | \theta)$$

NNでは誤差関数、ここでは期待収益 → 大きくしたい。

(2) 定義方法

① 平均報酬

② 割引報酬和

上記の定義に対して、行動価値関数： $Q(s,a)$ の定義を行い、方策勾配定理が成り立つ。

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

The image shows a handwritten derivation of the policy gradient theorem. At the top, the formula $\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}}[(\nabla_{\theta} \log \pi_{\theta}(a | s) Q^{\pi}(s, a))]$ is written in white on a black background. Below it, the same formula is written in orange on a dark blue background. An orange arrow points from the term $Q^{\pi}(s, a)$ in the handwritten formula to the text below it, which says "ある行動をとるときの報酬" (reward when taking a certain action).

$$\text{またこの式: } \nabla_{\theta} J(\theta) = \nabla_{\theta} \sum_{a \in A} \pi_{\theta}(a | s) \underbrace{Q^{\pi}(s, a)}_{\text{ある行動をとるときの報酬}}$$

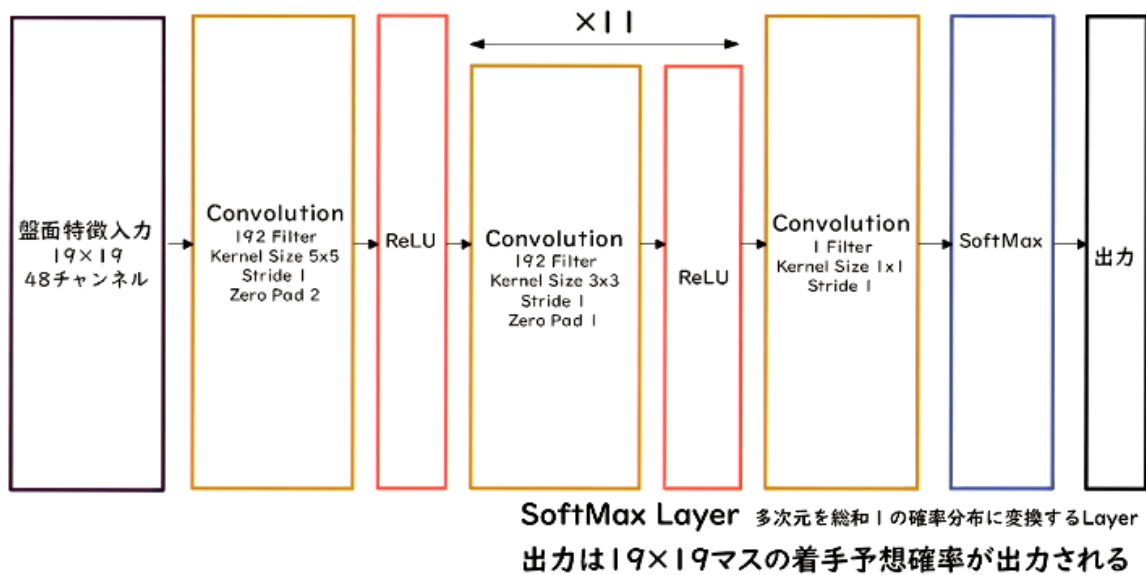
π_{θ} : エージェントがとる行動

2. section2_AlphaGo

AlphaGo には AlphaGo_Lee と AlphaGo_Zero がある。

2. 1 AlphaGo_Lee

Alpha Go LeeのPolicyNet

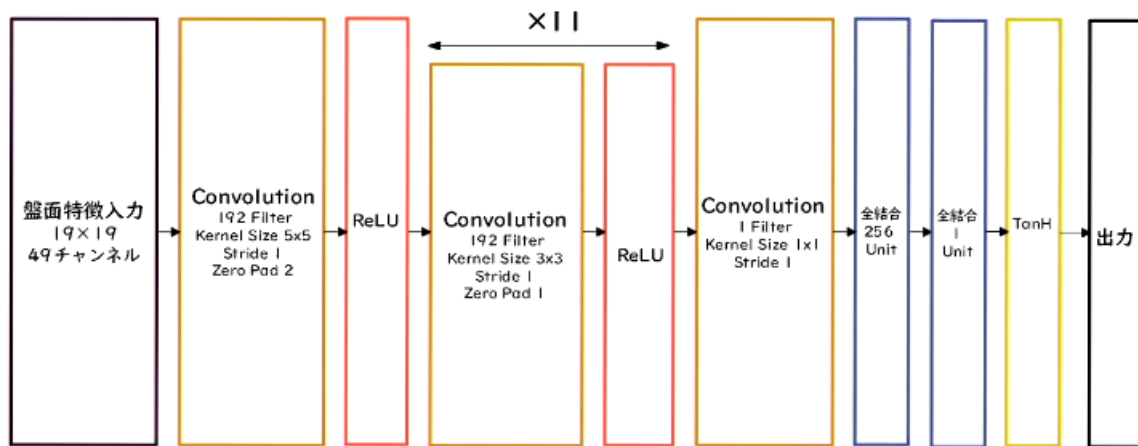


方策関数

(1) Policy Net, Value Net の入力

特徴	チャンネル数	内容
石	3	白石、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	合法手であるかどうか
オール0	1	全面0
手番	1	現在の手番が黒番か? (Policy Net にはこの入力はなく、Value Net のみ)

Policy Net は48チャンネル



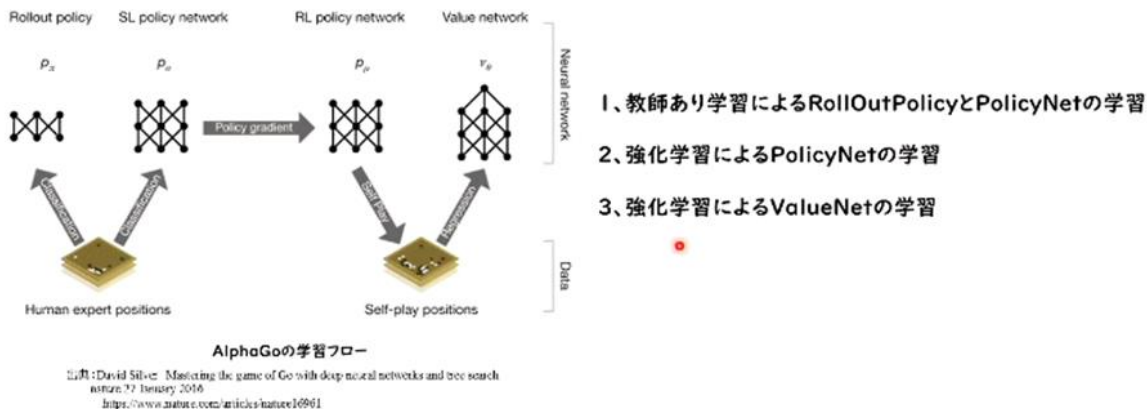
TanH Layer $-\infty \sim \infty$ の値を $-1 \sim 1$ までに変換するLayer
出力は現局面の勝率を $-1 \sim 1$ で表したものが出力される

価値関数

Value Net は49チャンネル

2. 1. 1 AlphaGo の学習

AlphaGo の学習は以下のステップで行われる。



2. 1. 2 PolicyNet の強化学習

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った。

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

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理由は、対局に幅を持たせて過学習を防ごうというのが主である。

この学習を mini batch size 128 で1万回行った。

2. 1. 3 ValueNet の学習

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

教師データの作成の手順は

- (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、損出関数を平均二乗誤差とし、回帰問題として学習した。

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

N 手までと N+1 手からの PolicyNet を別々にしてある理由は、過学習を防ぐためであると論文では説明されている。

2. 1. 4 RollOutPolicy

NN ではなく線形の方策関数

探索中に高速に着手確立を出すために使用される。

特徴	次元数	内容
1 2 近傍マッチ	1	1 つ以上の 1 2 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	8 1 9 2	ナカデのパターンにマッチするか
1 2 近傍 (直前)	3 2 2 0 7	直前の着手中心のダイヤモンド型 1 2 近傍パターンにマッチするか
3 × 3 近傍	6 9 3 3 8	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	3 4	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
1 2 近傍	6 9 3 3 8	着手しようとしているマス中心のダイヤモンド型 1 2 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 1 9 × 1 9 マス分あり、出力はそのマスの着手予想確立となる。

ポリシーネットでどこに置くかを教えてくれる (方策関数)

計算量が多いネットワーク

どうにかするための方法—>RollOutPolicy

3mm秒かかる。—>計算量の少ないものを使用する。

基本的にはPolicyNetと同じだがスピードが速い—>3 μ 秒

2. 1. 5 PolicyNet の教師あり学習

KGS Go Sever（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、教師と同じ着手予想できるように学習を行った。

具体的には、教師が着手した手を 1 とし残りを 0 とした 19×19 次元の配列を教師とし、それを分類問題として学習した。

この学習で作成した PolicyNet は 57% ほどの精度である。

過去の人間同士の打った手を教師あり学習で学習させる。

強化学習の学習手法

2. 1. 6 モンテカルロ木探索

コンピュータ囲碁ソフトでは現在最も有効とされている探索法。

他のボードゲームでは minmax 探索やその派生型の $\alpha\beta$ 探索を使うことが多いが、盤面の価値や確率予想値が必要となる。

しかし囲碁では盤面の価値や確率予想値を出すのが困難であるとされてきた。

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、その手を着手した後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である。

モンテカルロ木探索はこの木の成長を行うことによって、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。

価値関数を学習させるときの手法

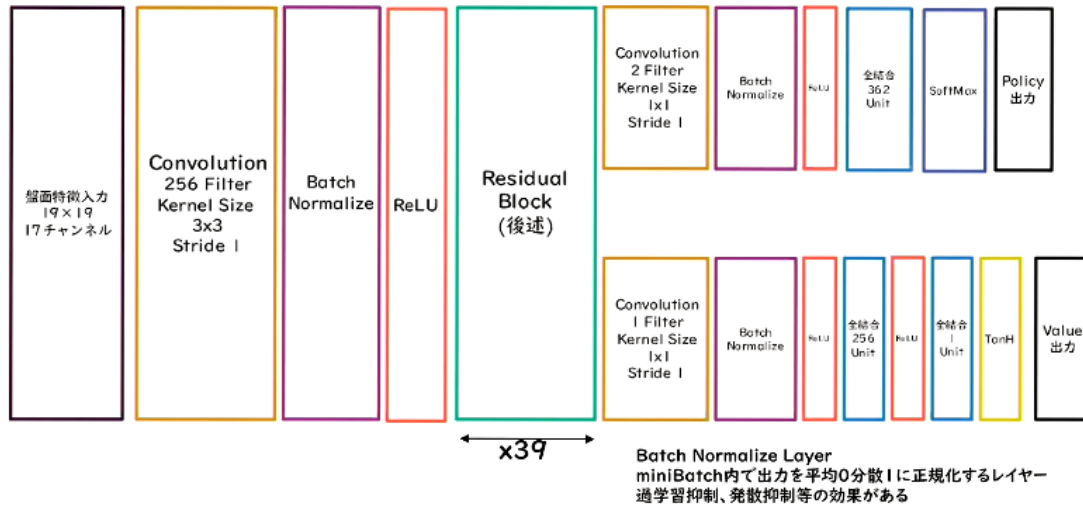
Roll Out Policy を使用する。

2. 2 AlphaGo_Zero

2. 2. 1 AlphaGo_Lee と AlphaGo_Zero の違い

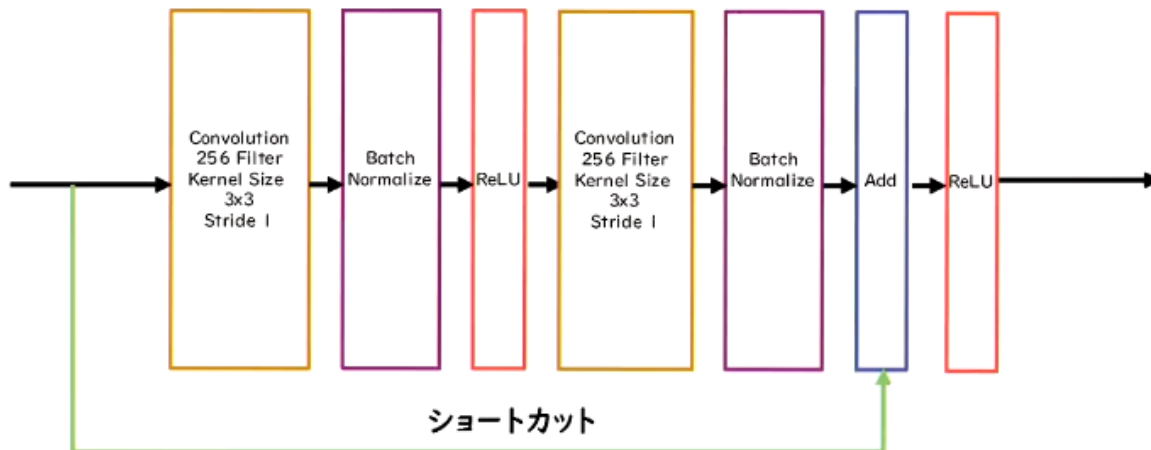
- (1) 教師あり学習を一切行わず、強化学習のみで作成
- (2) 特徴入力からヒューリスティックな要素を排除し、石の配置のみした
- (3) PolicyNet と ValueNet を1つのネットワークに統合した
- (4) Residual Net を導入した
- (5) モンテカルロ木探索から RollOut シミュレーションを無くした

Alpha Go ZeroのPolicyValueNet



ResidualNetwork

Residual Block 基本形



ショートカット：

ネットワークの深さを抑える。→勾配消失・勾配爆発等が起きづらくなる。

2. 2. 2 Residual Network

ネットワーク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。
 Residual Network を使うことにより、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
 Convolution->BatchNorm->ReLU->Convolution->BatchNorm->Add->ReLUのBlockを1単位にして積み重ねる形となる。
 また、Residual Network を使うことにより総数の違う Network のアンサンブル効果が得られるという説もある
 アンサンブル効果がある。

1) Residual Block の工夫

■Bottleneck

1×1 Kernel の Convolution を利用し、1層目で次元削減を行って3層目で次元を復元する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

ResidualBlock の並びを BatchNorm->ReLU->Convolution->BatchNorm->ReLU->Convolution->Add とすることにより性能が上昇したとするもの。

2) Network 構造の工夫

■WideResNet

Convolution の Filter 数をk倍にした ResNet。1倍->k倍 x 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、制度的にはなく、各層で Filter 数を増やしていく ResNet。

結) 畳み込み、プーリング、RNN、アテンションここに活性化関数。これを組み合わせる。

2. 2. 3 Residual Network の性能

ResidualNetworkの性能

layer name	output size	18 layer	34 layer	50 layer	101 layer	152 layer
conv1	112×112					
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	7.8×10 ⁹	11.3×10 ⁹

ResidualNetworkの構造

出典: Kaiyang 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
PRReLU-net [13]	24.27	7.38
resnet-34	28.54	10.02

左記はResidualNetwork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である。
 18、34Layerのものは基本形、50、101、152LayerのものはBottleNeck構造である。

性能は従来のVGG-16と比較してTop5で4%程度ものエラーの低減に成功している。

ResidualNetworkの性能



layer name	output size	16 layer	34 layer	50 layer	101 layer	152 layer
conv1	112 × 112	7 × 7, 64, stride 2				
7 × 7 max pool, stride 2						
conv2_x	56 × 56	$\begin{bmatrix} 1 \times 1, 64 \\ 1 \times 1, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 64 \\ 1 \times 1, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 1 \times 1, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28 × 28	$\begin{bmatrix} 1 \times 1, 128 \\ 1 \times 1, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 128 \\ 1 \times 1, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 1 \times 1, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
conv4_x	14 × 14	$\begin{bmatrix} 1 \times 1, 256 \\ 1 \times 1, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 256 \\ 1 \times 1, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 1 \times 1, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 1 \times 1, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 1 \times 1, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7 × 7	$\begin{bmatrix} 1 \times 1, 512 \\ 1 \times 1, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 512 \\ 1 \times 1, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 1 \times 1, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
fc	1 × 1	average pool, 1000-d.f. softmax				
Top5 err.		1.8 × 10 ²	3.6 × 10 ¹	3.6 × 10 ¹	7.6 × 10 ¹	11.5 × 10 ¹

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
PRoLNet-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>

左記はResidualNetwork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である。
18、34Layerのものは基本形、50、101、152LayerのものはBottleNeck構造である。

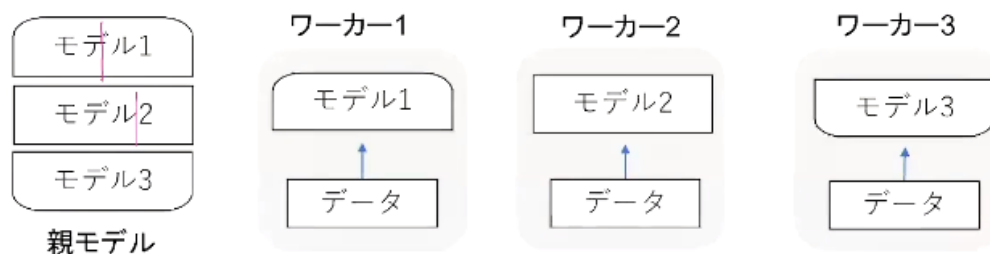
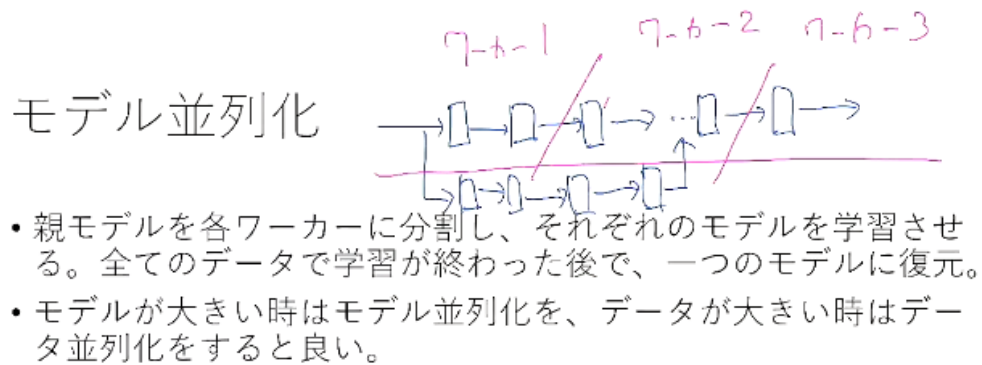
性能は従来のVGG-16と比較してTop5で4%程度ものエラー率の低減に成功している。

3. section3_計量化_高速化技術

3. 1 分散深層学習とは

- (1) 深層学習は多くのデータを利用したり、パラメータ調整のために多くの時間を使用したりするため、高速な計算が求められる。
- (2) 複数の計算資源（ワーカー）を使用し、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いたい。
- (3) データ並列化、モデル並列化、GPU による高速技術は不可欠である。

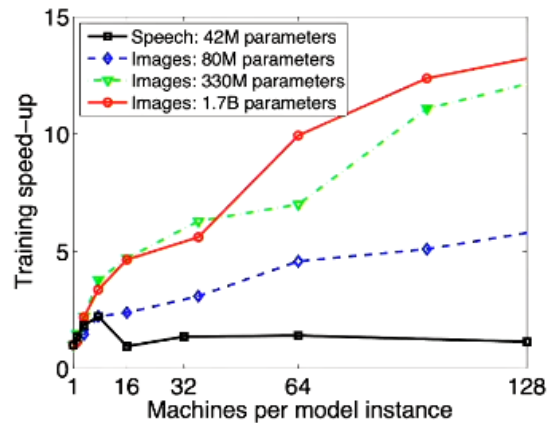
3. 2 モデル並列



モデルを分割して、並列に学習させる。

モデル並列

- 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。



(Jeffrey Dean et al. 2016)
Large Scale Distributed Deep Networks

枝別れ

一カで学習させるのが主流。

た部分んを別のワ

大きなモデルの場合有効

(1) 参照論文

① Large Scale Distributed Networks

- Google 社が 2016 年に出した論文
- Tensorflow の前身といわれている

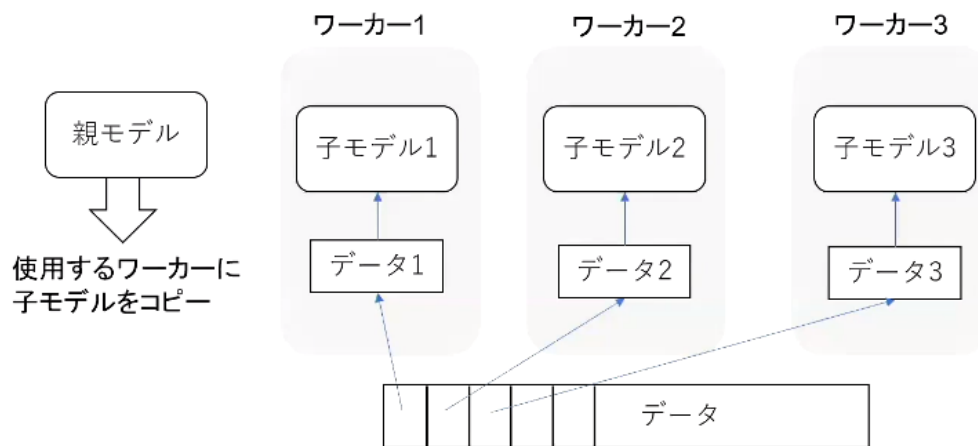
② 並列コンピューティングを用いることで大規模なネットワークを高速に学習させる仕組みを提案。

③ 主にモデル並列とデータ並列（非同期型）の提案をしている。

3. 3 データ並列

データ並列化

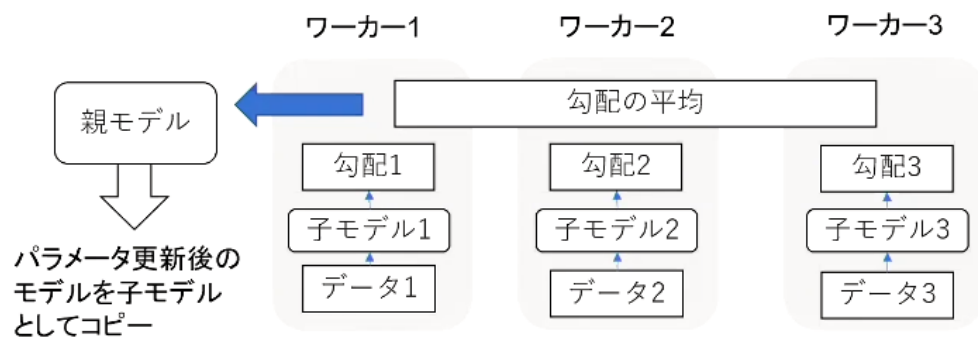
- 親モデルを各ワーカーに子モデルとしてコピー
- データを分割し、各ワーカーごとに計算させる



ワーカー：コンピュータ

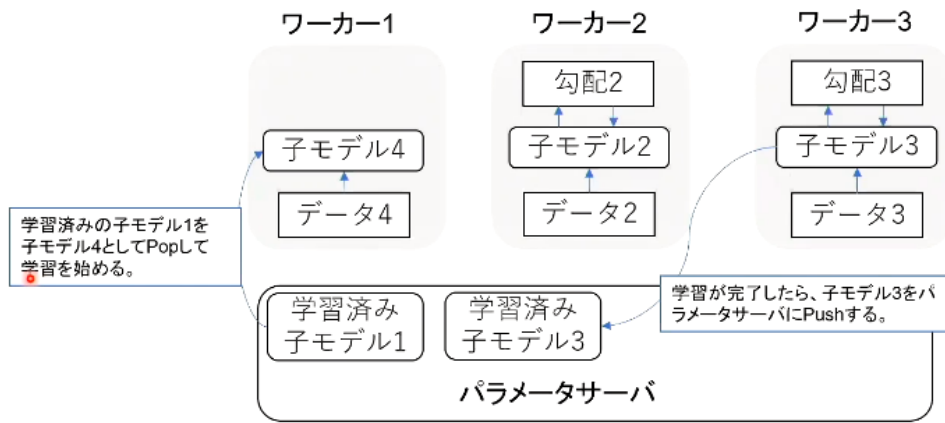
データ並列化: 同期型

- データ並列化は各モデルのパラメータの合わせ方で、同期型か非同期型が決まる。



同期型のパラメータ更新の流れ。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。

データ並列化: 非同期型



非同期型のパラメータ更新の流れ。
各ワーカーはお互いの計算を待たず、各子モデルごとに更新を行う。
学習が終わった子モデルはパラメータサーバにPushされる。
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していく。

3. 3. 1 同期型と非同期型の違い

- (1) 処理のスピードは、お互いのワーカーの計算を待たない非同期型の方が早い。
- (2) 非同期型は最新のモデルのパラメータを利用できないので、学習が不安定になりやすい。

→ Stale Gradient Problem

- (3) 現在は同期型の方が精度が良いことが多いので、主流となっている。

3. 4 GPU

3. 4. 1 GPU による高速化

(1) GPGPU(General-purpose GPU)

- ・元々の使用目的であるグラフィック以外の用途で使われる GPU の総称

(2) CPU

- ・高性能なコアが少数
- ・複雑で連続的な処理が得意

(3) GPU

- ・比較的 low 性能なコアが多数
- ・簡単な並列処理が得意
- ・ニューラルネットの学習は単純な行列演算が多いので、高速化が可能

3. 4. 1 GPGPU 開発環境

(1) 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 社が開発している GPU のみで使用可能。
- ・Deep Learning 用に提供されているので、使いやすい。

(2) 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 社以外の全者 (Intel,AMD,ARM など) の GPU からでも使用可能
- ・Deep Learning 用の計算に特化しているわけではない。

(3) Deep Learning フレームワーク (Tensorflow,Python) 内で実装されているので、使用する際は指定すればよい

3. 5 量子化

3. 5. 1 量子化とは

ネットワークが大きくなると大量のパラメータが必要になり、学習や推論に多くのメモリと演算処理が必要



通常のパラメータの 64bit 浮動小数点を 32bit などの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。

BERT : 大量のパラメータがある。

3. 5. 2 省メモリ化

ニューロンの重みを浮動小数点の bit 数を少なくし有効桁数を下げることによって、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量を抑えることができる。

3. 5. 3 量子化の利点と欠点

量子化の利点と欠点

利点	メモリ	16	32	64
○計算の高速化	計算量	少	多	大
○省メモリ化	速度	遅	速	遅
欠点	精度の低下	精度	悪	良

(1) 計算の高速化

倍精度演算（64bit）と単精度（32bit）は演算性能が大きく違うため、量子化により精度を落とすことにより、より多くの計算をすることができる。

深層学習で用いられる NVIDIA 社製の GPU の性能は下記の用になる

	単精度	倍精度
NVIDIA® Tesla V100™	15.7 TeraFLOPS	7.8 TeraFLOPS
NVIDIA® Tesla P100™	9.3 TeraFLOPS	4.7 TeraFLOPS

16ビット：単精度→15.7 TeraFLOPS

(2) 精度の低下

ニューロンが表現できる少数の有効桁が小さくなる。

→モデルの表現力が低下する

学習した結果ニューロンが重みを表現できなくなる。ただし実際の問題ではほぼ精度が変わらない。

機械学習の場合は16ビット（単精度）で十分

(3) 極端な量子化

極端な量子化を考える。表現できる値が0，1の1ビットの場合、誤差が大きくなる。

量子化する際は精度が落ちない程度に量子化しなければならない。

3. 6 蒸留

3. 6. 1 蒸留とは

精度の高いモデルはニューロンの大きなモデルになっている。

そのため、推論に多くのメモリと演算処理が必要になる。

蒸留により、規模の大きなモデルの知識を使用して、軽量のモデルの作成を行う。

3. 6. 2 モデルの簡約化

学習図の精度の高いモデルの知識を軽量のモデルへ継承させる。

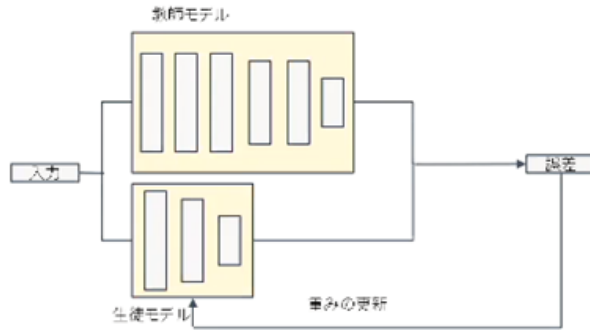
知識の継承により、計量でありながら複雑なモデルに匹敵する制度のモデルを得ることができる。

3. 6. 3 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計量なモデル



重みの更新の時、生徒モデルだけを学習させる。

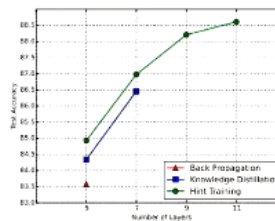
3. 6. 3 蒸留の利点

蒸留の利点

下記のグラフは Cifar 10 データセットで学習を行ったレイヤー数と精度のグラフになる

表の back propagation は通常の学習、Knowledge Distillation は先に説明した蒸留手法、Hint Taraing は蒸留は引用論文で提案された蒸留手法

図から蒸留によって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ている



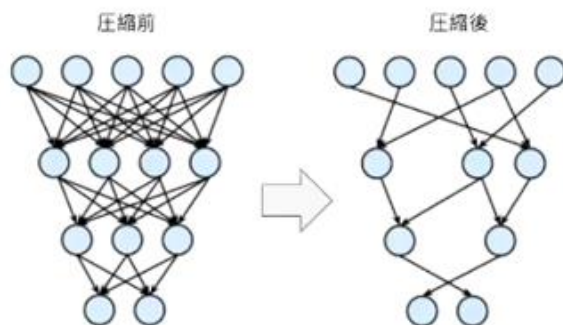
Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, Yoshua Bengio (2015) [FITNETS: HINTS FOR THIN DEEP NETS] ,<https://arxiv.org/pdf/1412.6550.pdf>.

3. 7 プルーニング

3. 7. 1 プルーニングとは

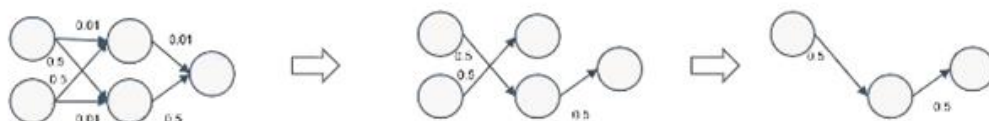
ネットワークが大きくなると大量のパラメータになるが、全てのニューロンの計算が制度の寄与しているわけではない。

→モデルの精度に寄与が少ないニューロンを削減することでモデルの軽量化、高速化が見込まれる。
寄与の少ないニューロンの削減を行いモデルの圧縮を行うことで高速化に計算を行うことができる。



3. 7. 2 ニューロンの削減

ニューロンの削減の手法は重みが閾値以下の場合、ニューロンを削減し、再学習を行う。



3. 7. 2 ニューロン数と精度

ニューロン数と精度

下記の表は 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になる。 α は閾値のパラメータ、閾値は各層の標準偏差 σ とパラメータの積
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する

α	パラメータ削減率 (%)				精度 (%)
	全結合層 1	全結合層 2	全結合層 3	全結合層全体	
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

α	パラメータ削減率 (%)				精度 (%)
	全結合層 1	全結合層 2	全結合層 3	全結合層全体	
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

佐々木健太, 佐々木勇和, 鬼塚 真 (2015) 「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」, <https://db-event.ipn.org/deim2017/papers/62.pdf>.

3. 8 軽量化まとめ

量子化：重みの精度を下げることにより計算の高速化と省メモリ化を行う技術。

蒸留：複雑で精度のよい教師モデルから軽量の生徒モデルを効率よく学習を行う技術。

プルーニング：寄与の少ないニューロンをモデルから削減し高速化と省メモリ化を行う技術。

4. section4_応用モデル

4. 1 MobileNets

画像認識モデル

(1) 論文タイトル

Efficient Convolution Neural Network for Mobile Vision Application

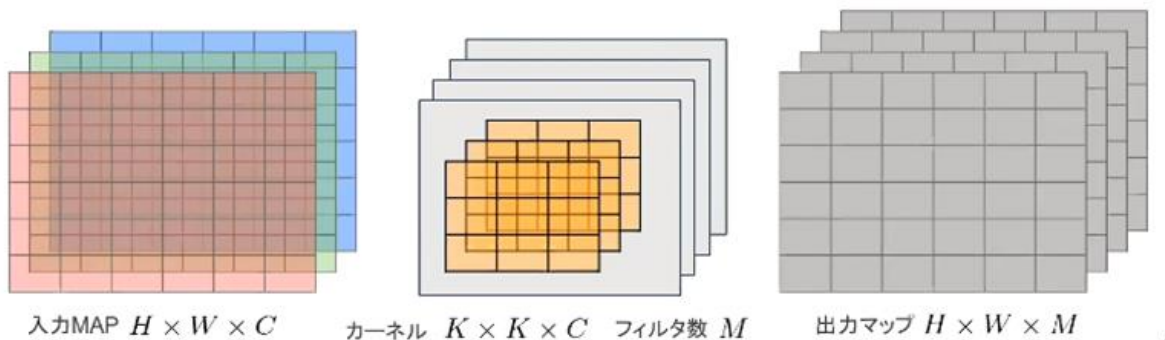
(2) 提案手法

- ・ディープラーニングモデルは精度はよいが、その分ネットワークが深くなり計算量が増える。
- ・計算量が増えると、多くの計算リソースが必要で、お金がかかってしまう。
- ・ディープラーニングモデルの軽量化・高速化・高精度化を実現

(3) 近年の画像認識タスクに用いられる最新のニューラルネットワークアーキテクチャは、多くのモバイル及び組み込みアプリケーションの実行環境を上回る高い計算資源を必要とされる。

4. 1. 1 一般的な畳み込みレイヤー

- ・入力特徴マップ (チャンネル数) : $H \times W \times C$
- ・畳み込みカーネルのサイズ : $K \times K \times C$
- ・出力チャンネル数 (フィルタ数) : M
- ・ストライド1でパディングを適用した場合の畳み込み計算の計算量



MobileNet は特殊な畳み込み演算を行う。

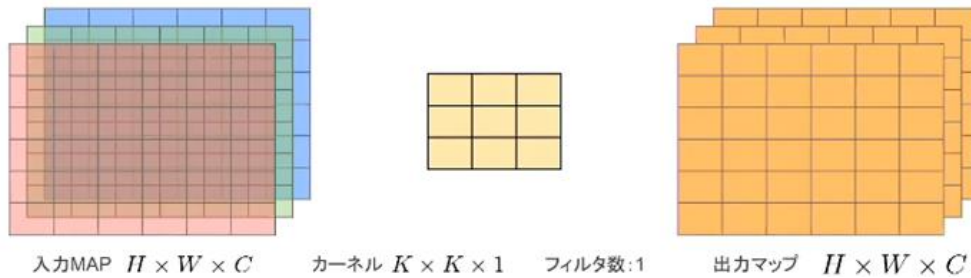
この点を計算するための計算量は $H \times W \times K \times K \times C \times M$

一般的な畳み込みレイヤーの計算量が多い。

MobileNets は Depthwise Convolution と Pointwise Convolution の組み合わせで軽量化を実現

(1) Depthwise Convolution の仕組み

- ・入力マップのチャンネル毎に畳み込みを実施
- ・出力マップをそれらと結合 (入力マップのチャンネル数と同じになる)



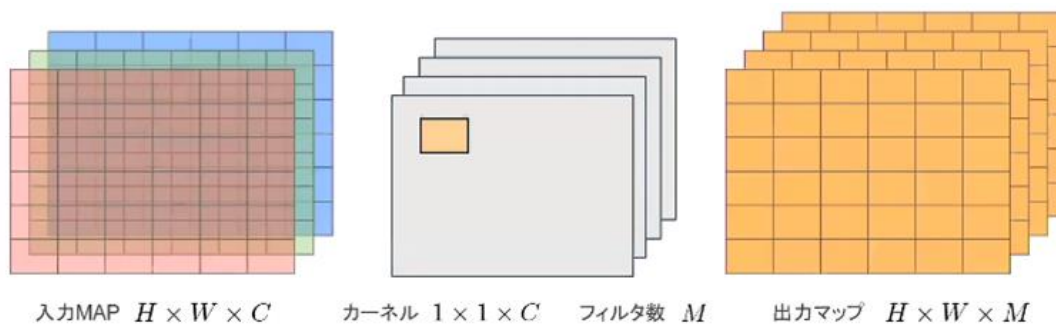
フィルタの数は1固定。1チャンネルのみ畳み込みを行う。

出力マップの計算量は（一般的な畳み込み層）： $H \times W \times K \times K \times C \times M$

出力マップの計算量： $H \times W \times C \times K \times K$

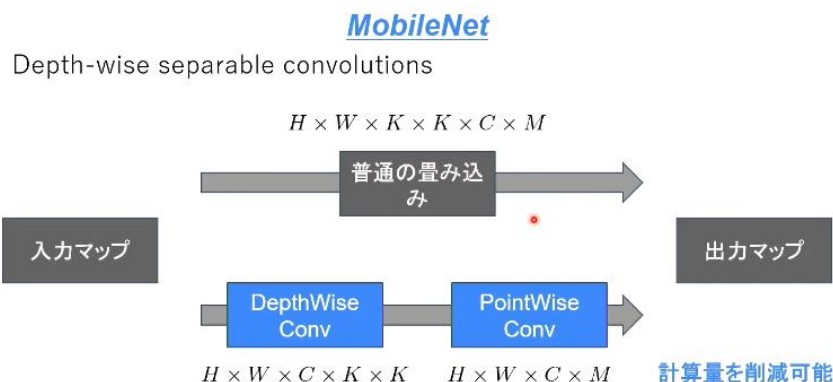
(2) Pointwise Convolution の仕組み

- 1×1 conv と呼ばれる（正確には $1 \times 1 \times c$ ）
- 入力マップのポイントごとに畳み込みを実施する
- 出力マップ（チャンネル数）はフィルタ数分だけ作成可能（任意のサイズ指定可能）



4. 1. 2 MobileNet のアーキテクチャ

- (1) Depthwise Separable Convolution という手法を用いて計算量を削減している。通常の畳み込みが空間方向とチャン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。
- (2) Depthwise Convolution はチャンネルごとに空間方向へ畳み込む。すなわち、チャンネル毎に $D_k \times D_k \times 1$ のサイズのフィルタをそれぞれ用いて計算を行うため、その計算量は小さくなる。
- (3) 次に Depthwise Convolution の出力を Pointwise Convolution によってチャンネル方向に畳み込む。すなわち、出力毎に $1 \times 1 \times M$ サイズのフィルタをそれぞれ用いて計算を行ため、その毛産量は小さくなる。



4. 2 DenseNet

画像認識のネットワーク

(1) 論文タイトル

Densely Connected Convolutional Networks.G.Huang et.,al.2016

(2) 概要

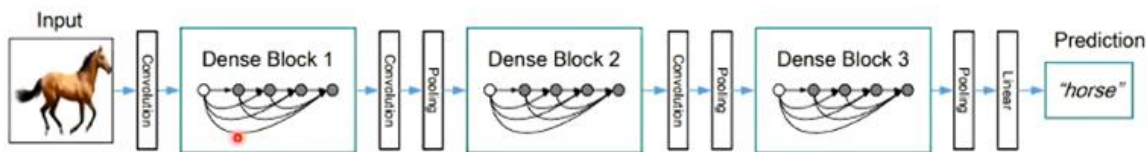
Dense Convolution Network は、畳み込みニューラルネットワークアーキテクチャの一種である。ニューラルネットワークでは層が深くなるにつれて、学習が難しくなるという問題があったが Residual Network などの CNN アーキテクチャでは前方の層から後方の層へアイデンティティ接続を介してパスを作ることで問題を対処した。DenseBlock と呼ばれるモジュールを用いた。DenseNet もそのようなアーキテクチャの 1 つである。

(3) 初期の畳み込み

(4) Dense ブロック

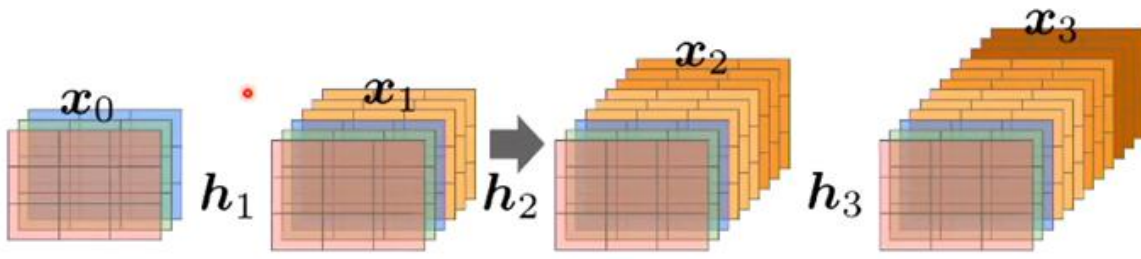
(5) 変換レイヤー

(6) 判別レイヤー



(7) 出力層に前の層の入力を足し合わせる。

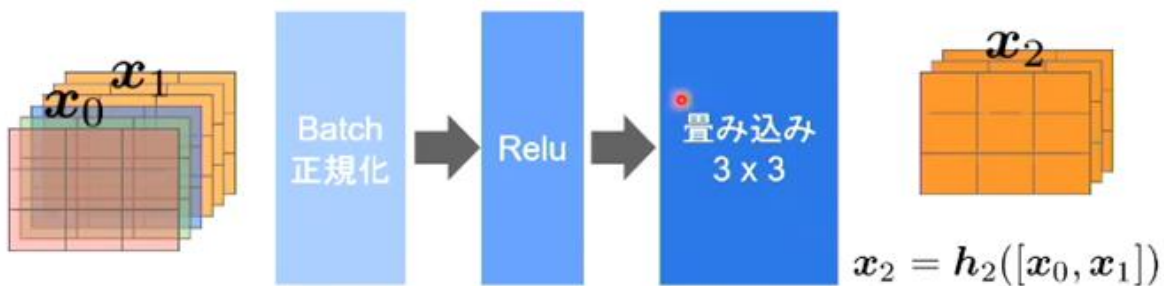
層間の情報の伝達を最大にするために全ての同時微量サイズ層を結合する



Dense Block を通過するとチャンネルが増える。

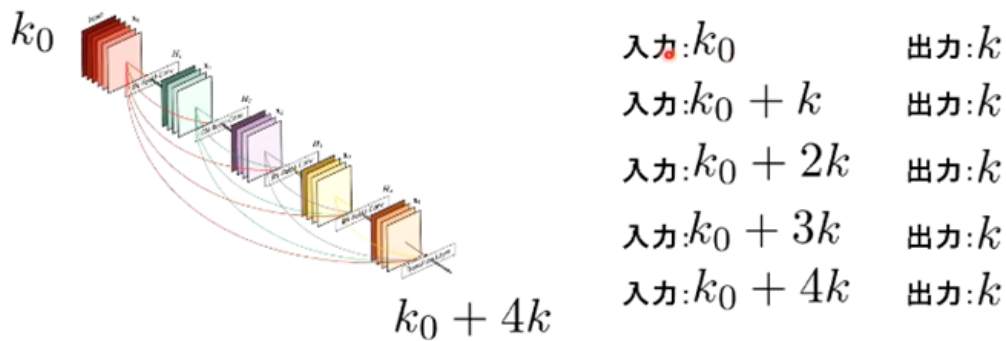
(8) 特徴マップの入力に対し、下記の処理で出力を計算する。

- Batch 正規化
- ReLU 関数による変換
- 3×3 畳み込み層による処理

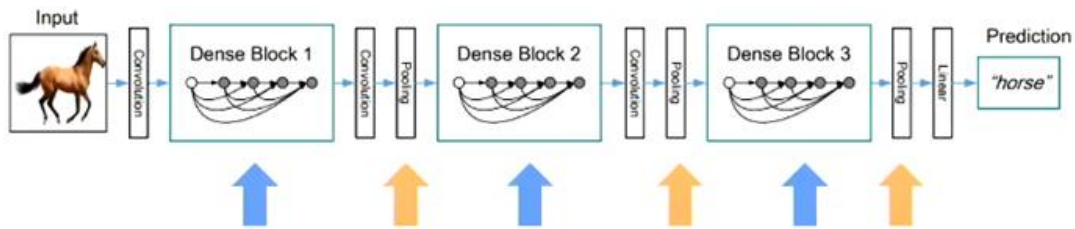


(9) k をネットワークの **growth rate** と呼ぶ

k が大きくなるほど、ネットワークが大きくなるため、小さな整数に設定するのが良い。



Transition Layer : Convolution+Pooling : チャンネル数を減らす



プーリングは特徴マップのサイズを削減

各ブロック内で特徴マップのサイズは一致

(10) DenseNet と ResNet の違い

- DenseBlock では前方の各層からの出力全てが後方の層への入力として用いられる。
- ResidualBlock では前1層の入力のみ後方の層へ入力

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

DenseBlock 内各ブロック毎に k 個ずつ特徴マップのチャンネル数が増加していく時、 k を成長率と呼ぶ。

4. 3 Layer 正規化／Instance 正規化

(1) BatchNorm

- ・レイヤー間を流れるデータの分布をミニバッチ単位で平均が 0・分散が 1 になるように正規化
- ・Batch Normalization は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、過学習の抑制など効果がある。

(2) Batch Norm の問題

- ・Batch Size が小さい条件下では、学習が収束しないことがあり、代わりに Layer Normalization などの正規化手法が使われることが多い。
- 正規化：平均が 0、分散が 1

(3) Batch Norm

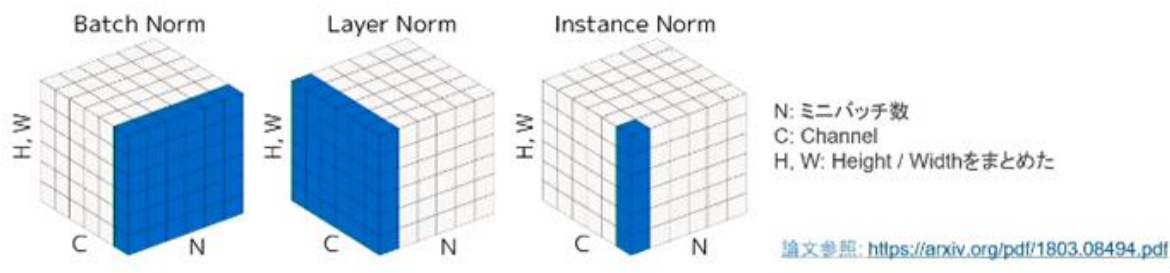
- ・ミニバッチに含まれる sample の同一チャンネルが同一分布に従うような正規化

(4) Layer Norm

- ・それぞれの sample の全ての pixels が同一分布に従うよう正規化

(5) Instance Norm

- ・さらに channel も同一分布に従うような正規化



Batch Normalization はあまり使いたくない。

Batch Norm :

縦 1 直線が画像 1 枚分 (6 チャンネル)

バッチサイズが 6 画像分

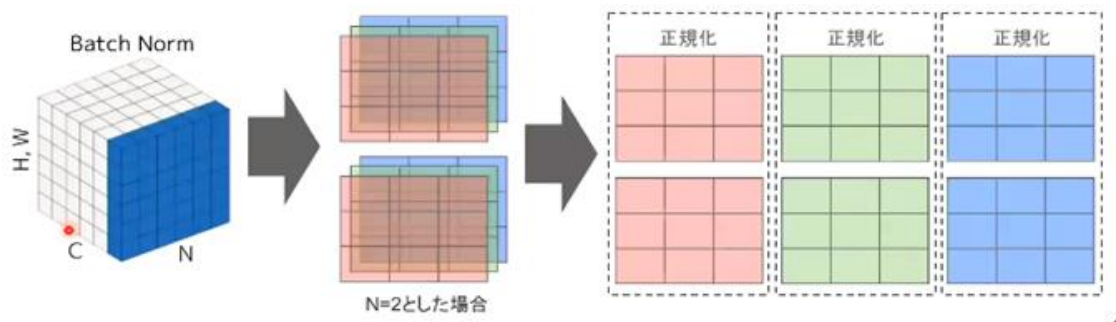
Layer Norm :

Instance Norm :

(6) $H \times W \times C$ の sample が N 個あった場合に、 N 個の同一チャンネルが正規化の単位

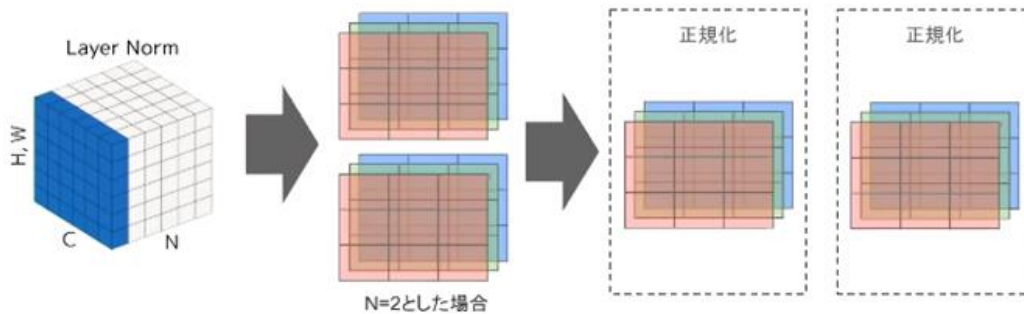
- ・RGB の 3 チャンネルの sample が N 個の場合は、それぞれのチャンネルの平均と分散を求め正規化を実施。チャンネル毎に正規化された特徴マップを出力。

(7) ミニバッチのサイズを大きく取れない場合には、効果が薄くなってしまう。



8) Layer Norm

- N 個の sample のうち 1 つに注目。 $H \times W \times C$ の全ての pixel が正規化の単位。
- RGB 3 チャンルの sample が N 個の場合は、ある sample を取出し、全てのチャンネルの平均と分散を求め正規化を実施。特徴マップごとに正規化された特徴マップを出力。
- ミニバッチの数に依存しないので、上記の問題を解消できると考えられる。

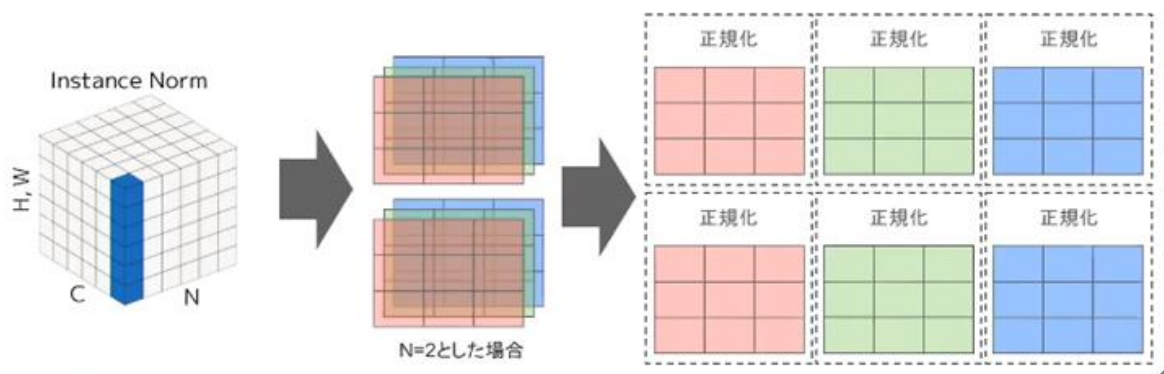


(9) Layer Norm は、入力データや重み行列に対して、以下の操作を実施しても、出力が変わらないことが知られている。

- 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

(10) Instance Norm

- 各 sample の各チャンネル毎に正規化
Batch Normalization のバッチサイズが 1 の場合と等価
- コントラストの正規化に寄与・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



4. 4 WaveNet

(1) WaveNet

- Aaron van den Oord et.al.,2016 により提案
AlphaGo のプログラムを開発しており、2014年に Google に買収される。
- 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。

(2) WaveNet のメインアイデア

- 時系列データに対して畳み込み (Dilated convolution) を適用する。
- Dilated convolution
層が深くなるにつれて畳み込むリンクを離す。
受容野を簡単に増やすことができるという利点がある。

- 図(右)では、Dilated = 1,2,4,8となっている

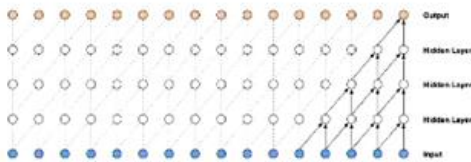


Figure 2: Visualization of a stack of causal convolutional layers.

既存の畳み込み

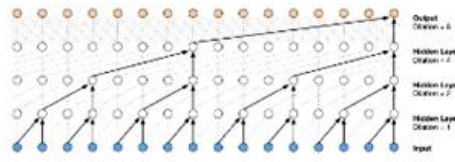


Figure 3: Visualization of a stack of dilated causal convolutional layers.

畳み込み (Dilated)

入力データがずれても大丈夫

より時間的に幅広いデータを用いながら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。

(3) Deconvolution : 逆畳み込み

画像の解像度を上げる。昔の画像を 4 K 画像に変換数。

- (4) パラメータ数に対する受容野が広いことを用いた際大きな利点は、単純な Convolution layer と比べて学習時に並列計算が行えることである。

5. section5_Transformer

- (1) Encoder-Decoder モデルの理解
- (2) BERT までのロードマップ
 - ①Encoder-Decoder Model
 - ②Transfer (Encoder-Decoder×Attention)
 - ③BERT

5. 1 Seq2Seq とは

系列 (Sequence) を入力して、系列を出力するもの

- (1) Encoder-Decoder モデルとも呼ばれる。
- (2) 入力系列が Encode (内部状態に変換) され、内部状態から Decode (系列に変換) する。
- (3) 実応用上も、入力・出力共に系列情報なものは多い
 - ①翻訳 (英語→日本語)
 - ②音声認識 (波形→テキスト)
 - ③チャットボット (テキスト→テキスト)

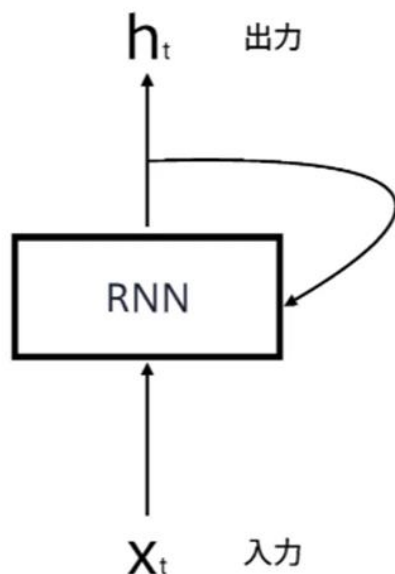
5. 2 Seq2Seq の理解に必要な材料

要は言語モデルを二つ連結した形になっている。

- (1) RNN の理解
 - ①RNN の動作原理
 - ②LSTM などの改良版 RNN の理解
- (2) 言語モデルの理解

5. 3 RNN とは

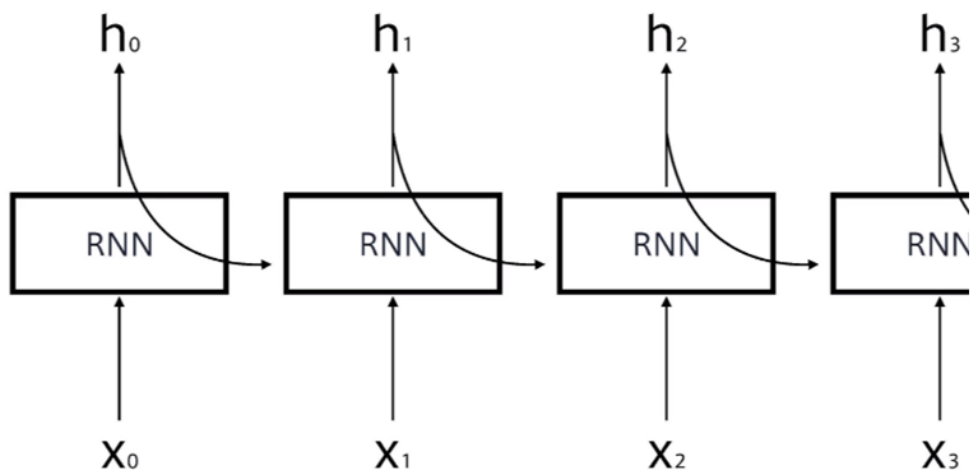
系列データを読むために再帰的に動作する NN (ニューラルネットワーク)



(1) 再帰的とは

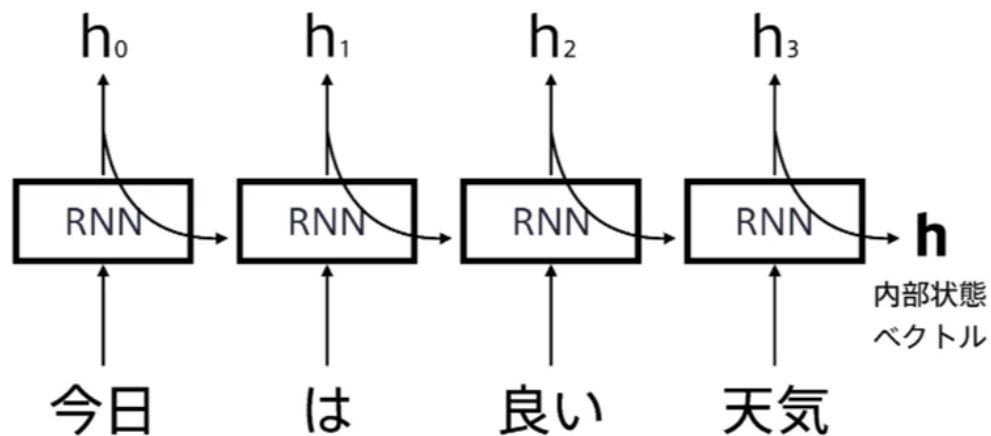
出力が再び入力になる

再帰処理は時間軸方向に展開できる。



前の時刻の出力を現在の時刻の入力にする。

(2) 系列情報を舐めて内部状態に変換できる。



5. 4 言語モデルとは

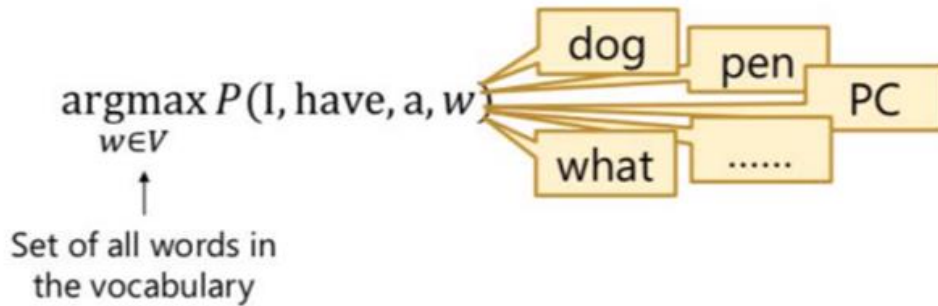
(1) 言語モデルは単語の並びに確率を与える。

①単語の並びに対して尤度（それがどれだけ起こり得るか）すなわち、文章として自然かを確率で評価する。

②数式的には同時確立を事後確率に分解して見せる。

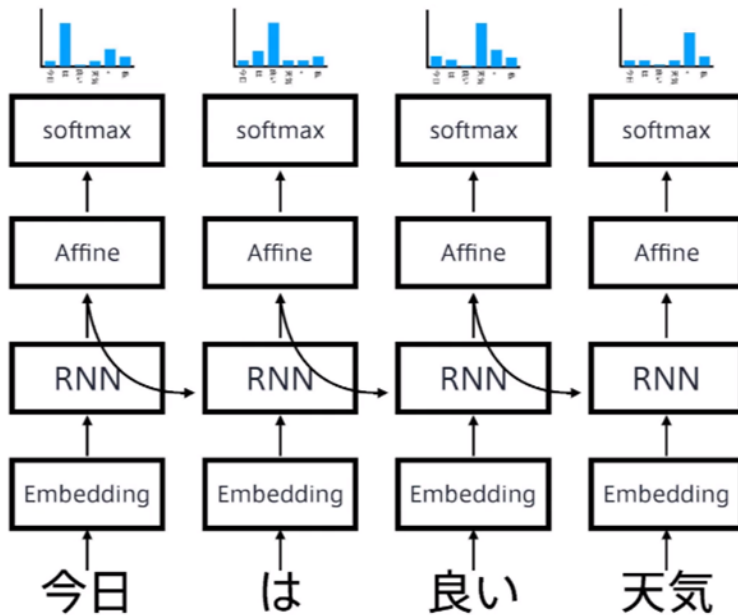
$$P(w_1, \dots, w_m) = \prod_{i=1}^m P(w_i | w_1, \dots, w_{i-1})$$

2) 時刻 $t-1$ までの情報で、時刻 t の事後確率を求めることが目標→これで同時確率が計算できる。



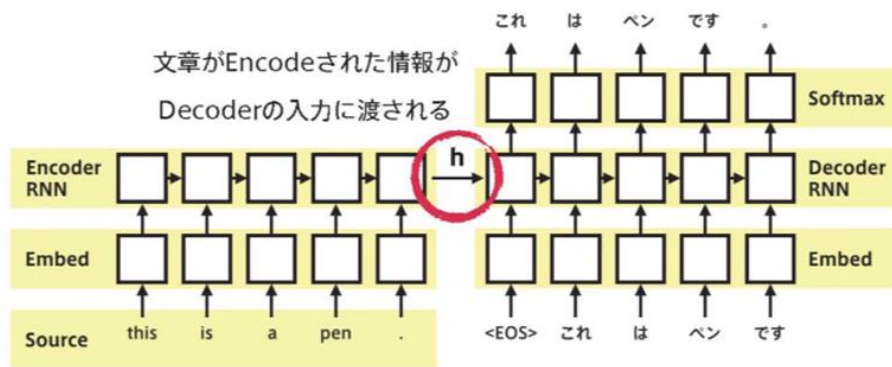
5. 5 RNN×言語モデル

(1) 各地点で次にどの単語が来れば自然（事後確率最大）かを出力できる。



5. 6 RNN×言語モデル・まとめ

- (1) RNN は系列情報を内部状態に変換することである。
- (2) 文章の各単語が現れる際の同時確率は、事後確率で分解できる。
したがって、事後確率を求めることが RNN の目標になる。
- (3) 言語モデルを再現するように RNN の重みが学習されていれば、ある時点の次の単語を予測することができる。
先頭単語を与えれば文章を生成することも可能
言語の自然な並び方を学習すると、逆に自然な文章を生成できる。
- (4) Seq2seq
Encoder から Decoder に渡される内部状態ベクトルが鍵



**Decoder側の構造は言語モデルRNNとほぼ同じだが
隠れ状態の初期値にEncoder側の内部状態を受け取る**

Decoder 側の構造は言語モデル RNN とほぼ同じだが、隠れ状態の初期値に Encoder 側の内部状態を受け取る。
Decoder の入力に Encoder の出力
教師あり学習が可能となる。

6. section6_物体検知_セグメンテーション

6. 1 物体検知の基礎とセマンティックセグメンテーション

(1) 広義の物体認識タスク

処理	出力
分類	(画像に対し単一または複数の) クラスラベル
物体検知	Bounding Box[bbox/BB]
意味領域分割	(各ピクセルに対し単一の) クラスラベル
固体領域分割	(各ピクセルに対し単一の) クラスラベル

(2) 代表的データセット

データセット	クラス	Train+Val	Box/画像	画像サイズ
VOC12	20	1、1540	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	

①VOC12

Visual Object Class

②ILSVRC17

- ImageNet Scale Visual Recognition Challenge
- ImageNet のサブセット

③MS COCO18

- Common Object in Context
- 物体位置推定に対する新たな評価指標を提案

④OICOD18

- Open Image Challenge Object Detection
- ILSVRC や MS COCO とは異なる annotation process
- Open Image V4 のサブセット

(3) IoU(Intersection over Unit)

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。

$\text{IoU} = \text{Area Of Overlap} / \text{Area of Union}$

①Confusion Matrix の要素を用いて表現

$\text{Iou} = \text{TP} / (\text{TP} + \text{FP} + \text{FN})$

②別名 Jaccard 係数という

(6) 深層学習以降の物体検知

AlexNet の登場を皮切りに、時代は SIFT から DCNN へ

SIFT : Scale Invariant Feature Transform

DCNN : Deep Convolutional Neural Network

(7) 物体検知のフレームワーク

- DetectorNet
- RCNN
- SPPNet
- Fast RCNN
- YOLO
- Faster RCNN
- SSD
- RFCN
- YOLO9000
- FPN
- RetinaNet
- Mask RCNN
- ConerNet

(8) One-stage Detector と Two-stage Detector

- Two-stage Detector

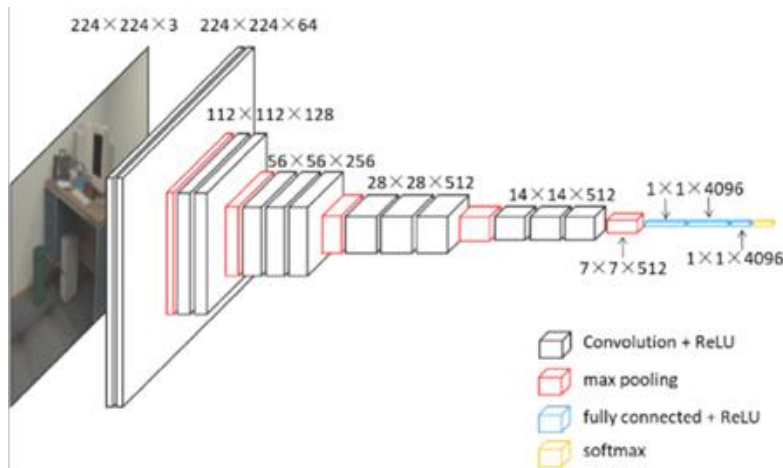
物体の候補領域の推定とクラスの推定をそれぞれ 2 つの独立のネットワークで行うもの

- One-stage Detector

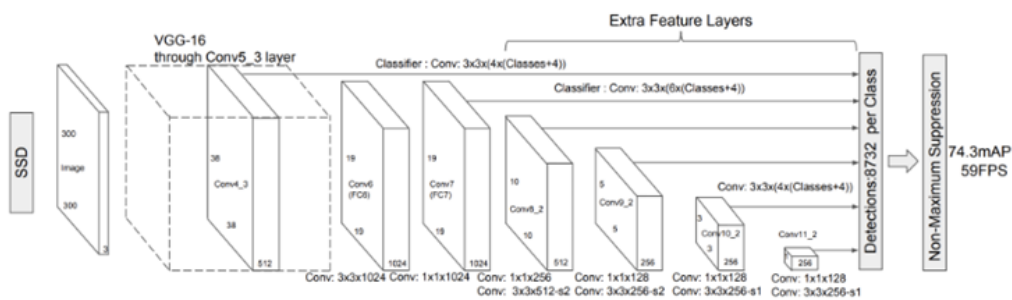
上記を一つのネットワークで同時に推定するもの

6. 2 SSD

- (1) Default Box を用意
- (2) Default Box を変形し、conf.を出力
- (3) VGG16 のアーキテクチャ



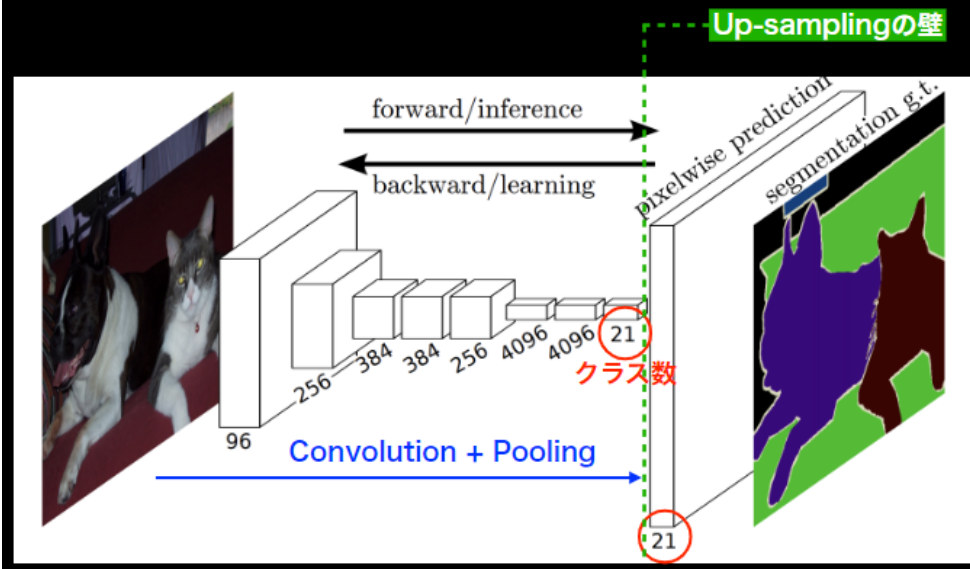
- (4) SSD のネットワークアーキテクチャ



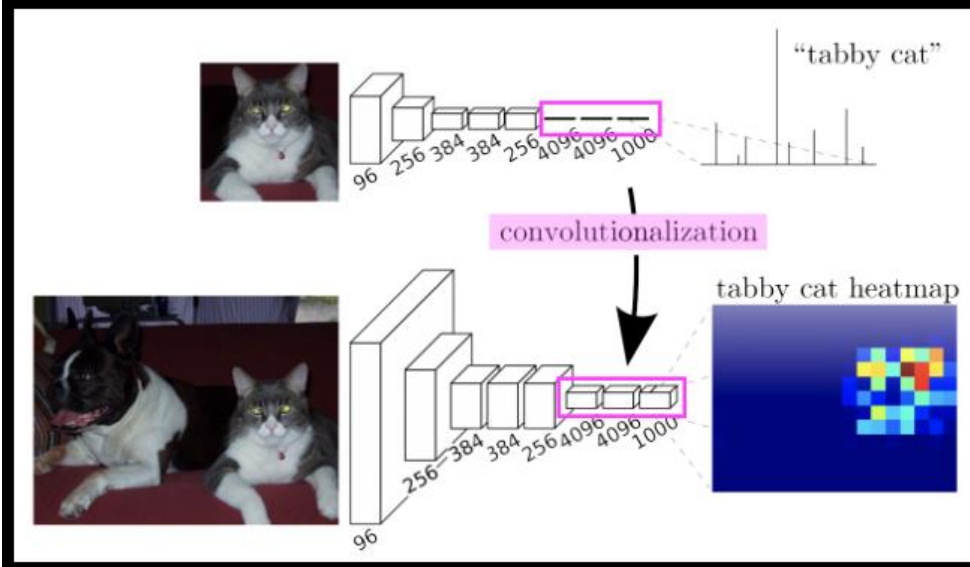
入力サイズに応じて SSD300,SSD512 と表記される。

6. 3 Semantic Segmentation の概要

ざっくり分かるSemantic Segmentationの肝

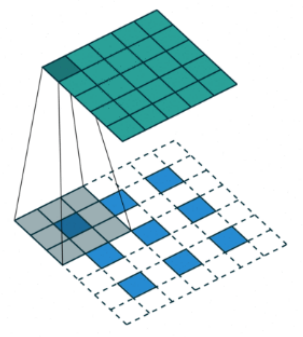


FCN (Fully Convolutional Network) の基本アイデア



6. 3. 1 Deconvolution/Transposed convolution

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

左図は**kernel size = 3, padding = 1, stride = 1**のDeconv.により3x3の特徴マップが5x5にUp-samplingされる様子.

Input

Kernel

Output

※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
→当然, **pooling**で失われた**情報**が復元されるわけではない

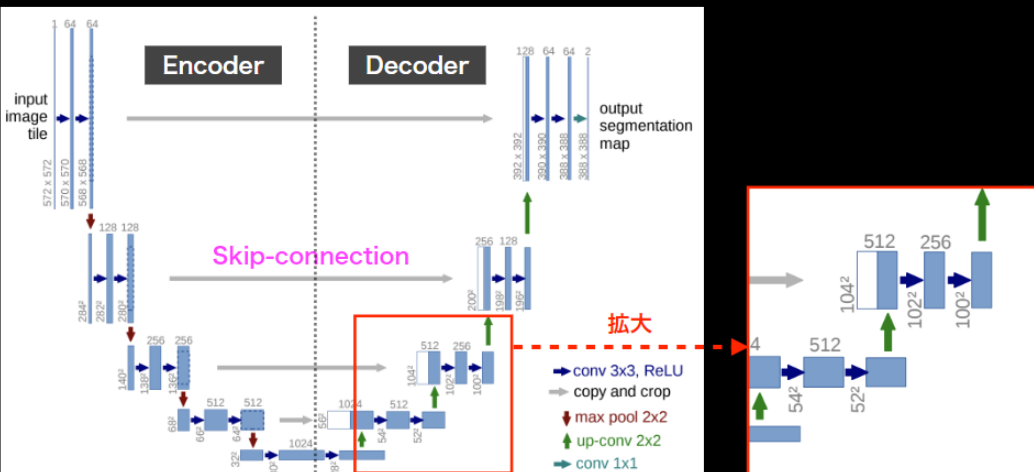
矩形範囲指定(R)Ctrl+Alt+FF

https://github.com/ydmucolin/conv_arithmetic

115

6. 3. 2 U-Net

U-Net



input image tile

Encoder

Decoder

output segmentation map

Skip-connection

拡大

conv 3x3, ReLU

copy and crop

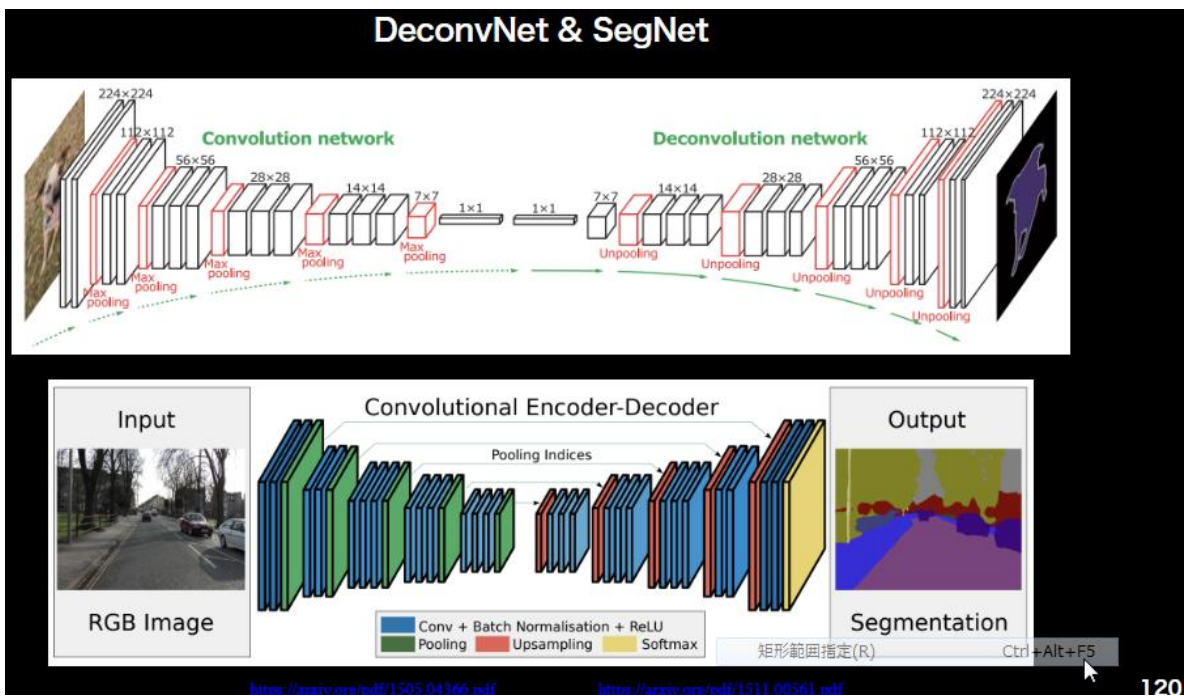
max pool 2x2

up-conv 2x2

conv 1x1

36

6. 3. 3 DeconvNet & SegNet



6. 3. 3 Dilated Convolution

