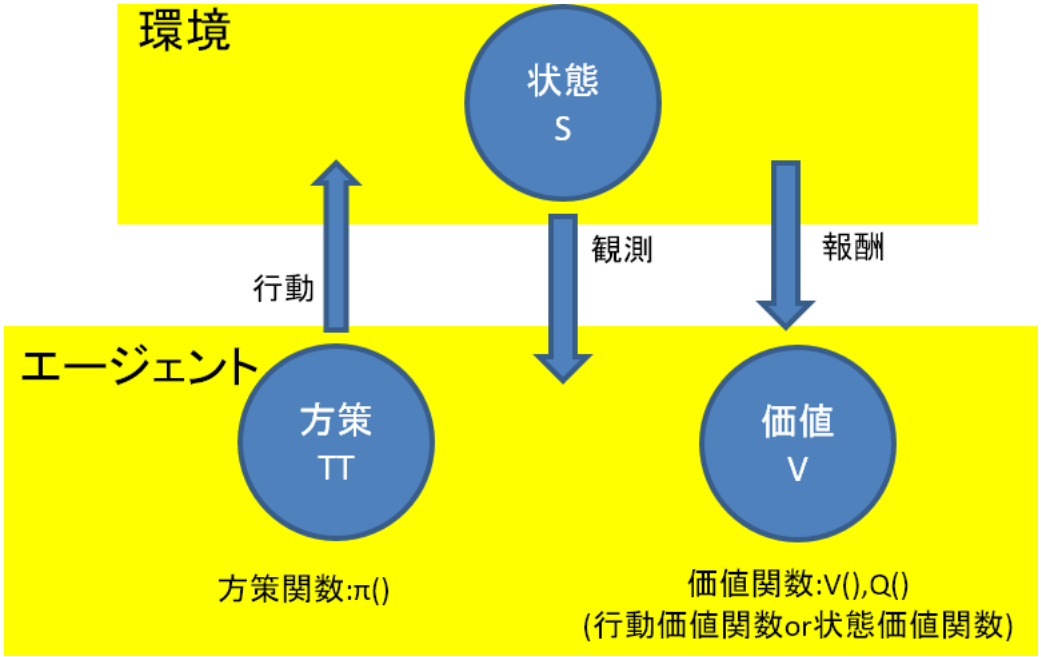
1　強化学習

・機械学習の手法 (DNN にも適用できる)
・教師あり学習、教師なし学習、と並んで手法として定義されている

・エージェントという仕組みがモデルの動かし方 (方策) を学習する
・例えば同一の機械学習モデル同士を競わせて、より勝率が高い方策を得る様に学習する
・エージェントは環境より得る報酬 (スコア) より、良い方策か否か認識する
・報酬 (スコア) はメリットとデメリット (コスト等) を包括して決める



・方策は最初はランダムに選び、より報酬 (スコア) が高い方策を採用する
・過去の良い方策にこだわり新たな探索を怠る様になると発展がない
・かといって過去の良い方策を利用しないと効率が悪い
・何事もバランス感覚と説明 (この辺、人間の行動そのもの)

・強化学習は以下 2 つが重要となる
　Q 学習 (行動価値関数を、行動する毎に更新する事により学習を進める方法)
　関数近似法 (価値関数や方策関数を関数近似する手法)

・強化学習を数学モデルに置き換える (2 か所)
　方策 (方策関数:π ())
　価値 (価値関数:$Q(), V()$)

・方策関数は π ()、ある状態でどのような行動を取るのかの確率を与える関数

・価値を表す関数としては、状態価値関数 $V()$ と行動価値関数 $Q()$ の 2 種類がある
　ある状態の価値にのみ注目する場合、状態価値関数を利用する
　状態と行動を組み合わせた価値に注目する場合、行動価値関数 (=コストを加味した関数)

・という事は各関数の引数も相応に異なってくる
　引数 s:状態 (state)
　引数 a:行動 (action)
　方策関数:π (s, a) ならば状態や過去の行動をもとに方策を決める (=経験値を生かす)
　方策関数:π (s) ならば状態のみをもとに方策を決める
　方策関数の出力が a という事になる
　状態価値関数:状態のみに着目なので $V(s)$
　行動価値関数:状態と行動に着目なので $Q(s, a)$

・価値関数の出力が大きくなる様に強化学習を行う (誤差という概念は無い)

・方策勾配法という方法で学習を行う (以下が方策勾配法の式)
　$\theta^{t-1} = \theta^t + \epsilon \nabla J(\theta)$　　(θ は重みやバイアス、$J(\theta)$ は方策関数の指標 (良し悪し) を示す関数)

・方策関数自体を DNN で構成し、$J(\theta)$ の出力値をより大きくする様に θ を学習するという事

・$\nabla J(\theta)$ の数式は方策勾配定理等より以下の形に簡素化される (証明省略)
　$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[(\nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s, a))]$

2 Alpha Go

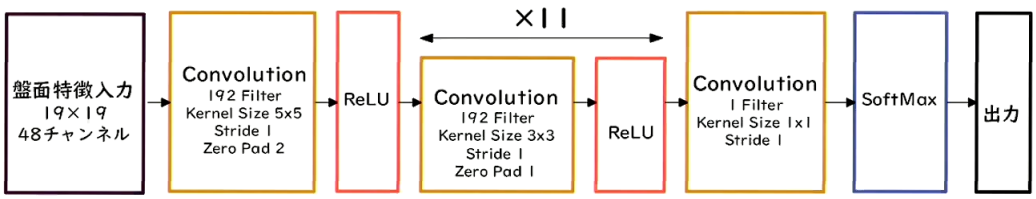
・有名な囲碁対戦モデル)
・Alpha Go Lee(2016) と Alpha Go Zero(2017) の 2 つについて説明

2.1 Alpha Go Lee

・PolicyNet と ValueNet の 2 つの DNN で構成される

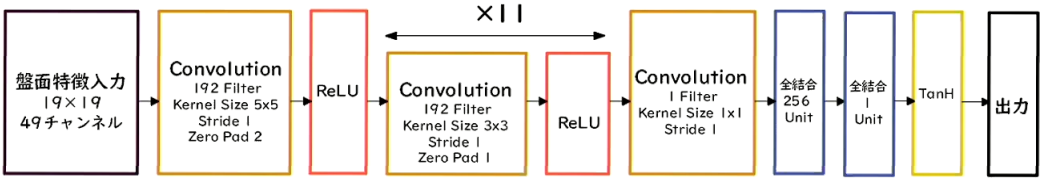
・PolicyNet

方策関数となる
出力は 19 マス x19 マスの着手予想確率
多層の畳み込み層で構成され、活性化関数は ReLU、出力は Softmax(総和 1 の確率分布)



・ValueNet

価値関数となる
出力は勝率を-1〜1 で表したものになる
多層の畳み込み層で構成され、活性化関数は ReLU、出力は TanH(-∞〜∞の値を-1〜1 に変換)



・PolicyNet と ValueNet の入力チャンネルはほとんど同じ説明変数となる

・すべて 19 x 19 の情報となる (以下)

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）

2.2 Alpha Go Lee の学習

・以下ステップとのこと

1. 教師あり学習による RollOutPolicy と PolicyNet の学習
2. 強化学習による PolicyNet の学習
3. 強化学習による ValueNet の学習

・教師あり学習による PolicyNet の学習

教師データは棋士の棋譜データを使う (棋士と同じ手を打つ様に学習)

→RollOutPolicy(線形方策関数) も学習する

→RollOutPolicy は強化学習フェーズにおけるモンテカルロ木探索に用いる (高速化)

・強化学習による PolicyNet および ValueNet の学習

モンテカルロ木探索を用いて強化学習する

これは PlayOut と呼ばれるランダムシミュレーションを多数回実行である

ランダムなので初期方策は精度が低い (が高速な)RollOutPolicy を採用していると理解する

モンテカルロ木探索によるシミュレーション回数が既定数超過ならば

その時点の手を初期値としてさらに探索木を成長させる

モンテカルロ木探索により勝率の高い手を探索 (強化学習) する

2.3 Alpha Go Zero

・Alpha Go Lee より以下を見直ししたネットワークとなる

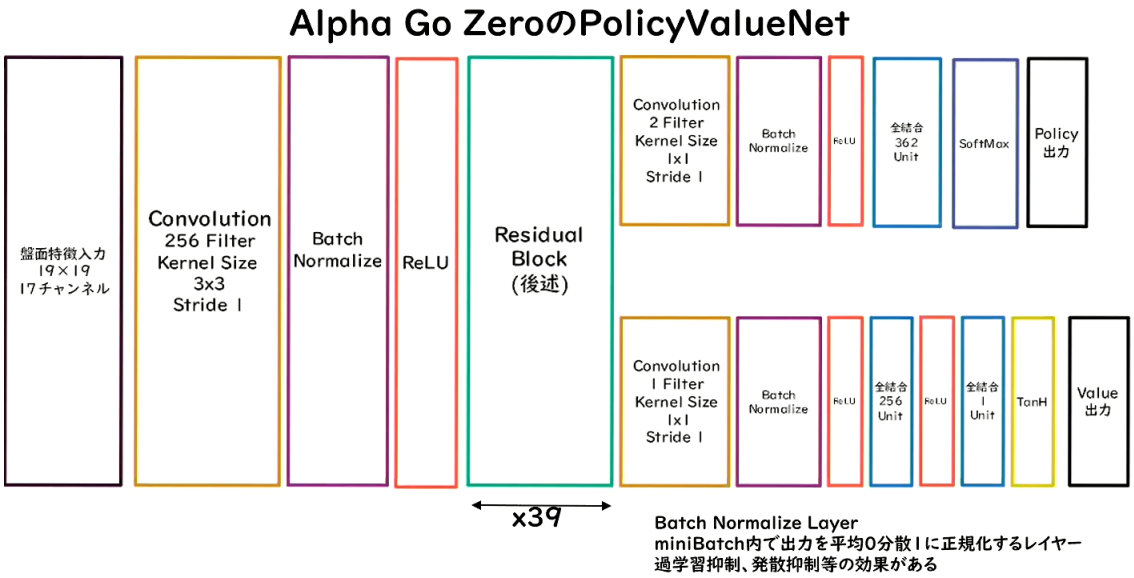
教師あり学習を廃止し強化学習のみで作成

ヒューリステックなパラメータを廃止した (石の配置にのみ着目する)

PolicyNet と ValueNet を統合 (パラメータ含め共通部分を統合した)

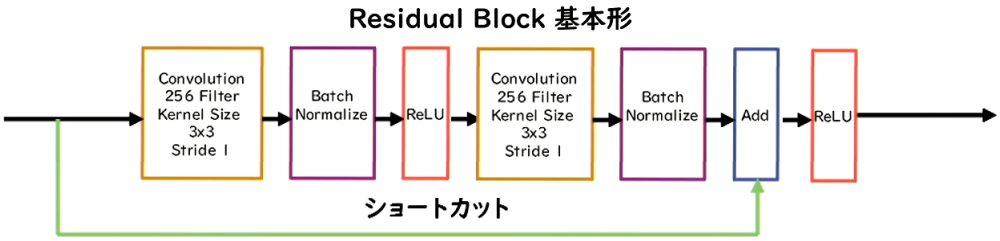
ネットワークの中間層に ResidualNet を導入

モンテカルロ木探索から RollOut シミュレーションをなくした



2.4 Alpha Go Zero の ResidualNet について

・多層構造による勾配消失 (爆発) 問題を解決するような構造
・入力を最終段に加算することにより勾配消失 (爆発) 問題を回避
・Alpha Go Zero では下記の Block が 39 層積み重なる



・ResidualBlock の取捨選択が可能なためネットワークの多様性に寄与すると理解
(説明ではアンサンブル効果とされている)

2.5 閑話休題

・ResidualNet もそうだが、大体の「新技術」と発表されるものは「既存技術の組み合わせ変更」
・基本部品 (既存の活性化関数や畳み込みやプーリング) の構成を変えて性能 UP した → 有用な技術
・という説明が講師からされている
(無論それが 100% ではないのは確かだが、応用技術とはそういうものと理解)

・私見：量子ゲート方式の計算機が実用化されたら手法も色々と変わるんだろうと思う

3　軽量化・高速化技術

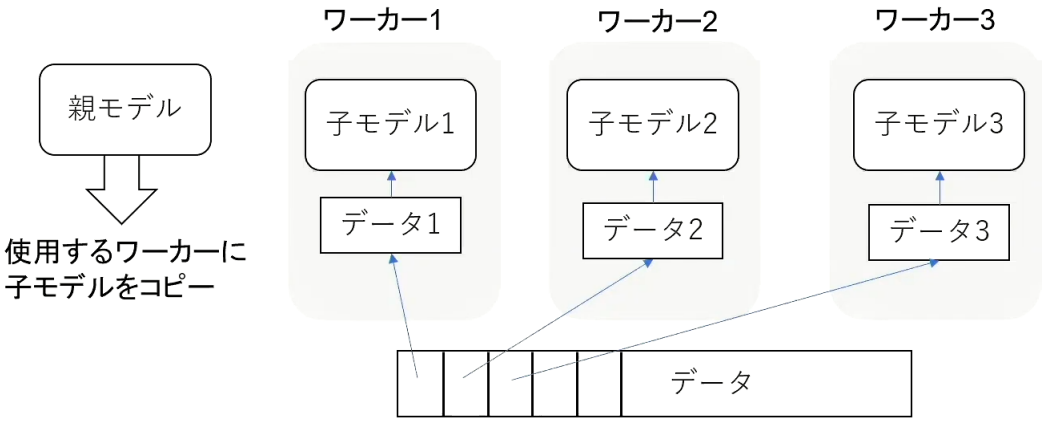
・深層学習は多くのデータをつかう、またその NN 構成はどんどん複雑化する (データも増える)
・対して計算機の進化が追いつかないのが現状
　→ なので何等かの工夫が必要となると説明

3.1　分散深層学習

・NN 自体の並列化 (複数計算機 (グリッドコンピューティング) で同時に学習可能な構成)
・並列化 (データ並列化とモデル並列化)、GPU による高速化 (Node レベルの並列化)、が不可欠

3.1.1 データ並列化

・データを分割して複数のワーカー (他の独立したコンピュータ) に同一モデルの学習をさせる
・同期型と非同期型の 2 種類のデータ並列化がある

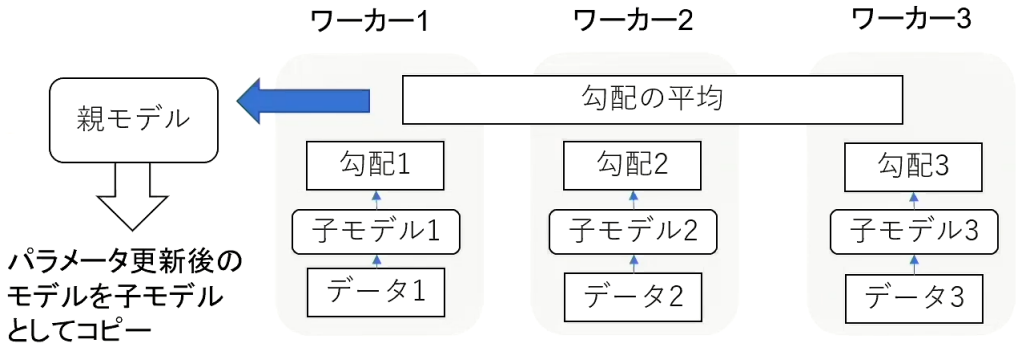


・同期型は学習が安定する (常に最新パラメータを参照) が、学習速度が遅い
・非同期型は学習が安定しない (任意学習度パラメータを参照) が、学習速度が速い

・同期型と非同期型は用途に応じて使い分けされる
・ワーカーが管理下に存在ならば同期型 (を採用するケースが多い)
・ワーカーが任意 (世界中に非管理下になど) に点在するなら非同期型 (にせざるを得ない)

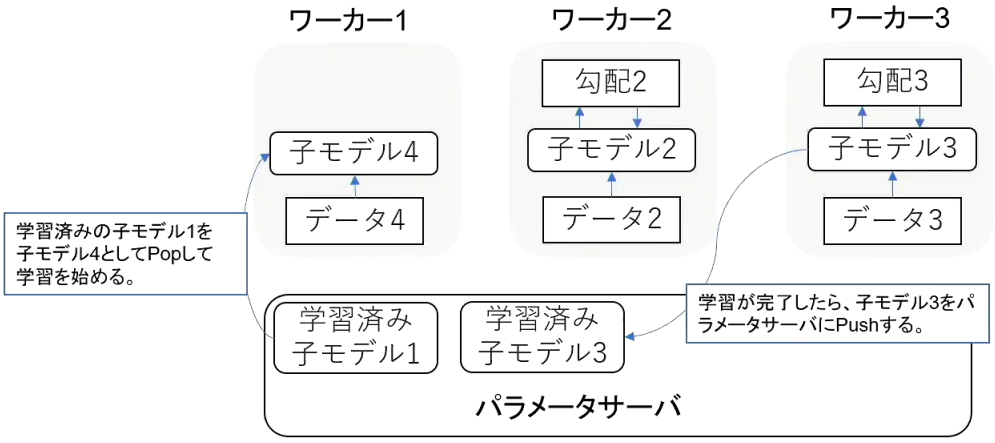
・同期型と非同期型の仕組みは以下に説明する (講義コンテンツより)

データ並列化：同期型



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。

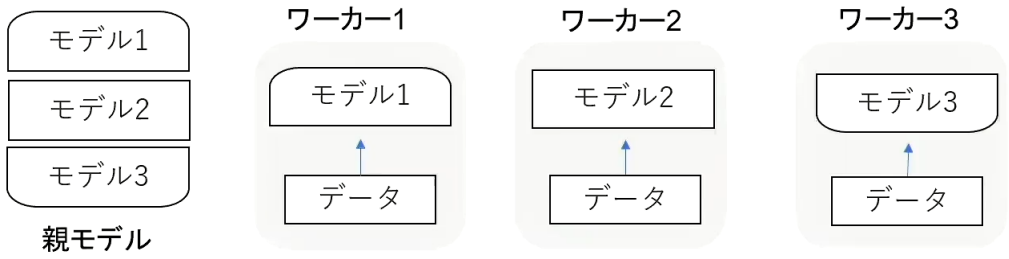
データ並列化：非同期型



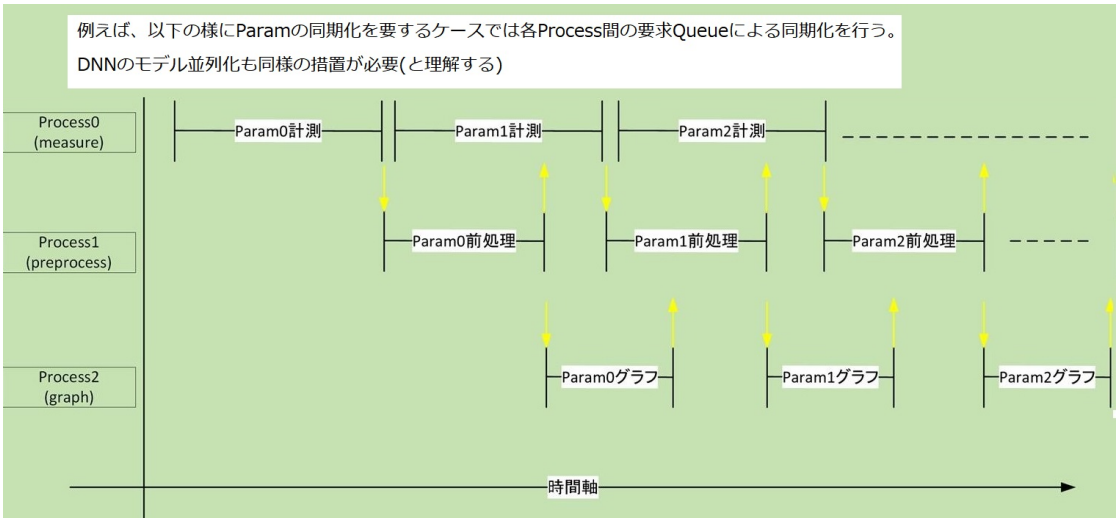
非同期型のパラメータ更新の流れ。
各ワーカーはお互いの計算を待たず、各子モデルごとに更新を行う。
学習が終わった子モデルはパラメータサーバにPushされる。
新たに学習を始める時は、パラメータサーバからPopしたモデルに対して学習していく。

3.1.2 モデル並列化

・モデルを分割して複数のワーカー (他の独立したコンピュータ) に同一データの学習をさせる



上記イメージではモデルの途中を切断して各々の断片を独立に学習する様に見えるが、
原理上、独立計算は無理なので何等かの方法で学習結果の同期化が必要
(ex:マルチプロセス処理における要求 Queue(以下) に似た)



同期化手段はネットワーク (Internet) しかなく、速度の問題が残る
なので、どちらかというと元々並列なモデルの並列部分をモデル分割して学習させる事が多いとの事

・モデル並列化はそのモデルが大規模である (パラメータが多い) ほど効果が高くなる

3.1.3 GPU による並列化

・元々は PC のグラフィックボードを汎用計算に使う為のプロセッサとなる

・GPU は低レベルなプロセッサが数 1000 個レベルで集積されている
・DNN の様な行列表現された「単純な積和演算」を「同時並列」に実行するのに適している

・GPU 自体は各社 (AMD、Intel、NVIDIA、ARM、等) から提供される
→ が、DeepLearning に特化したのは NVIDIA のみ (CUDA を提供し抽象化している)
→NVIDIA 含めて他社は OpenCL には対応しているがそれは DeepLearning 用ではない

・CUDA 自体もさらに抽象化 (Tensorflow 等の DeepLearning フレームワークで Wrap される)
→ 結果的に CUDA を理解しなくても使える

3.1.4 量子化 (Quantization)

・現代の PC で浮動小数点の桁数を決める bit 数は 2 のべき乗 (8,16,32,64,...) 表現
・現代の PC の CPU は 64bit が主流

・DNN の個々の (Node レベルの) 計算精度は左程重要ではない (64bit 浮動小数精度不要)
→32bit 精度 (あるいはもっと落とす) でも十分
→ 相対的にメモリ消費量削減と演算処理の削減 (速度 UP) ができる
→ 当然、省電力化にも寄与できる (はず)

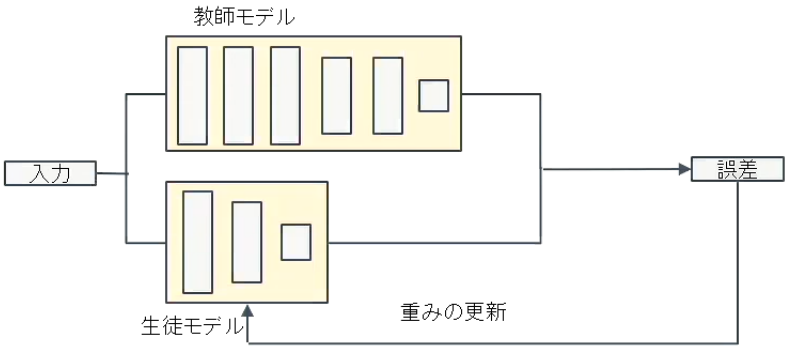
・メモリ消費量は bit 数に比例 (bit 数が半分ならばメモリ量も半分)
・演算速度は bit 数に相反 (bit 数落とせば速度 UP 方向)

・精度を十分に確保しつつ bit 数を落とす試みがされている
・浮動小数点の指数部と仮数部の比率変更し、学習結果のベンチマーク比較、等の研究

・Google の TPU 等の専用プロセッサもそれになる

3.1.5 蒸留

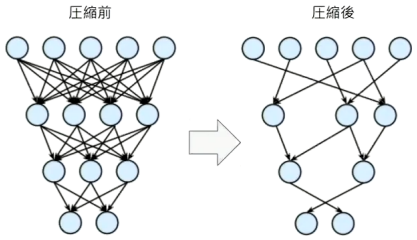
・精度が高いモデルは規模が大きく学習に時間がかかる
→ 推論に利用 (を限定) する場合でもメモリを多く使い演算処理も多くなる
→ 規模の大きなモデルの振る舞いを軽量な (推論専用) モデルの学習に使う



・引用図の様に、教師モデル (学習済み) と生徒モデル (教師モデルを軽量化、学習対象) を並列接続
・それぞれのモデルが出力する誤差をとり、誤差を小さくする様に生徒モデルのパラメータを学習する

3.1.6 プルーニング

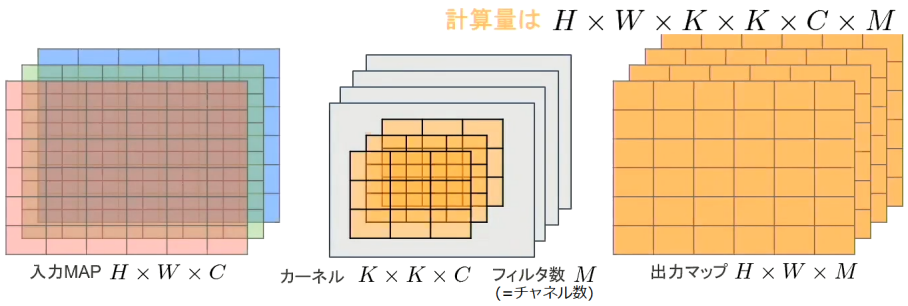
・大規模な DNN は大量のパラメータを有する
→ 必ずしもすべてのニューロンが精度に寄与しているわけではない
→ 精度への寄与が少ないニューロンを削除する (=計算量を削減&メモリ削減)
→ 削除の指標は重みの値とその層の標準偏差の積 (閾値より小なら削除とする等)



4　応用技術

4.1　MobileNet

・CNN 畳み込み層は出力情報のサイズより逆算して演算量 (掛け算のみ) を見積もる (以下)
(出力の Total 画素数から逆算 → 適合する様にパディングやストライドを暗黙的に決定)

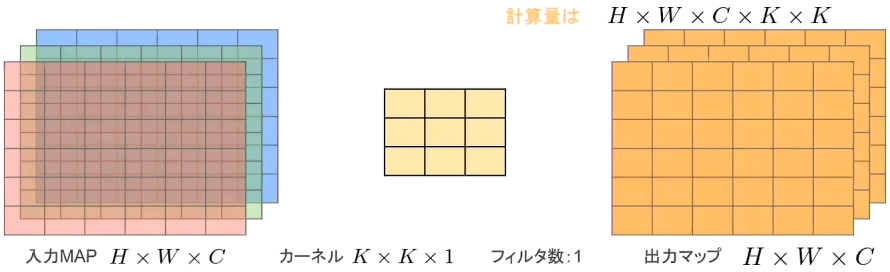


→ 一般的な畳み込み演算は計算量が多い
→Depthwise Convolution と Pointwise Convolusion の 2 つで計算量を削減する

・次に Depthwise Convolution と Pointwise Convolusion について説明する

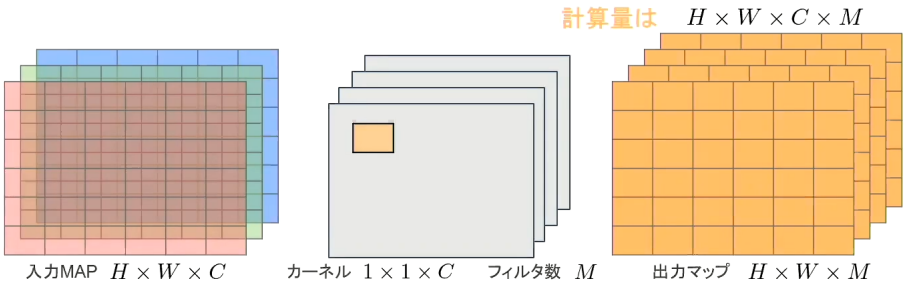
4.1.1 Depthwise Convolution

・入力マップのチャネル毎に畳み込みを実施
・出力マップをそれらと結合 (入力マップのチャネル数と同じになる)
(出力の Total 画素数から逆算 → 適合する様にパディングやストライドを暗黙的に決定)



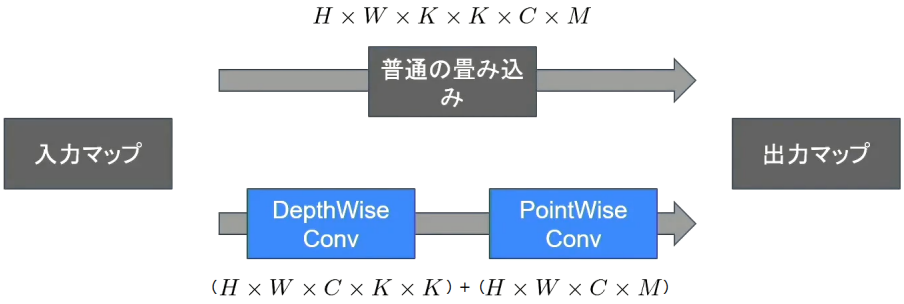
4.1.2 Pointwise Convolution

・1x1conv とも呼ばれる
・入力マップのポイント毎に畳み込み (というより色フィルタっぽい) を実施
・出力マップはフィルタ数分となる (よって任意数を指定する)



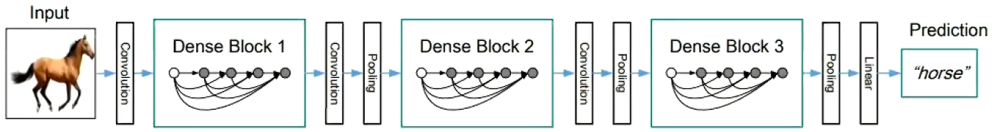
4.1.3 Depthwise Convolution と Pointwise Convolution を結合

・例えば以下の様になる (計算量が削減される)

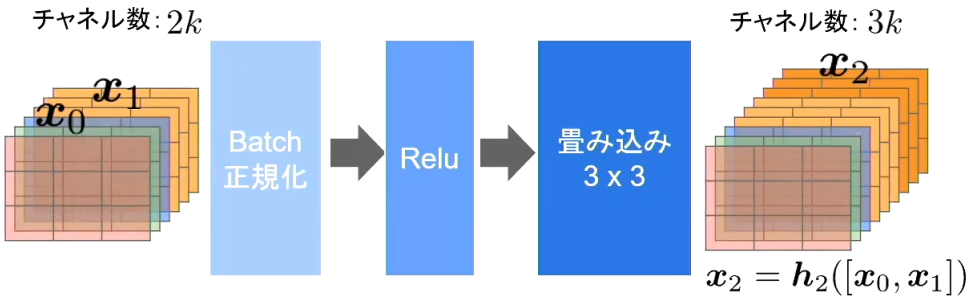


4.2 DenseNet

・ResidualNet と同じような思想を加えたネットワーク (画像認識用)
・DenseBlock の中身が 4 層で構成される
・DenseBlock 中身の各層を通るか通らないかのパスを夫々の層で全結合する様な構成 (以下図)
・DenseBlock 内で増えた出力を TransitionLayer(Conv+Pooling の層) で結合し元に戻す
・ResidualNet はひとつ前の層からのみショートカットされる構造 (DenseNet との相違点)



・多層のネットワークにおける勾配消失問題に対応したもの
・DenseBlock 内の層は以下 (手前層からの総出力にその層を通った出力を加える)



・DenseBlock における成長率 (Growth Rate)k がハイパーパラメータとなる
・成長率 k は DenseBlock の個別の出力チャネル数となる

4.3 正規化

・平均 0 分散 1 になるように正規化する

4.3.1 BatchNormalization(バッチ正規化)

・ミニバッチ単位で入力の各チャネルを一括りとして正規化を行う
→ 例えば RGB のチャネルならばバッチサイズ分の画像について
R チャネル,G チャネル,B チャネルの括りで夫々正規化する

・ミニバッチサイズは学習環境等に依存する (正規化都合で決められない)
→ ミニバッチサイズが小さいと正規化が安定しない (逆に十分大きいならば効果大)
→ 良くないので LayerNormalization 等の正規化手法を使う場合もある

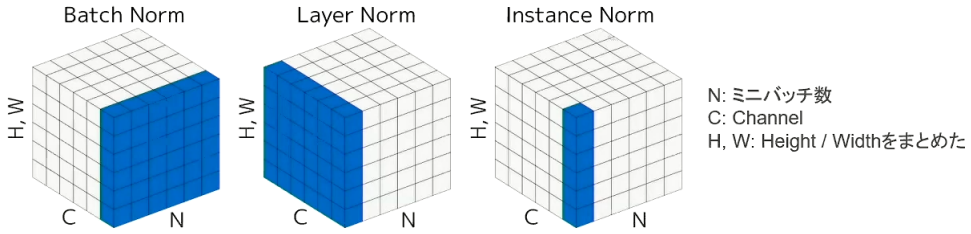
4.3.2 LayerNormalization

・夫々のサンプル (=画像と考える) の全チャネルの画素を正規化
→ 例えば一枚の画像の RGB チャンネルを対象として正規化するという事

4.3.3 InstanceNormalization

・夫々のサンプル (=画像と考える) の各チャネルの画素を夫々正規化
→ 例えば一枚の画像の RGB チャンネル夫々に対して正規化するという事

4.3.4 各正規化の範囲



4.4 Wavenet

・畳み込み演算を行い音声を生成する
・DilatedConvolution を採用している
→ 予測に用いるネットワーク中間層を一定単位で間引く
→ 入力範囲を長く (サンプリング数を多く) 取れる

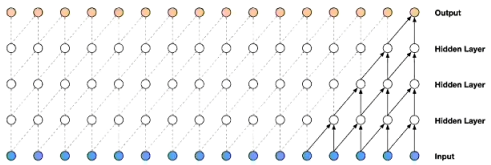


Figure 2: Visualization of a stack of causal convolutional layers.

既存の畳み込み

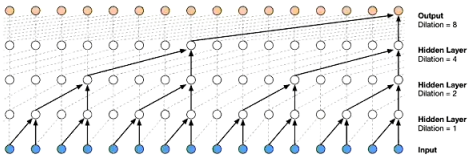


Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

畳み込み (Dilated)