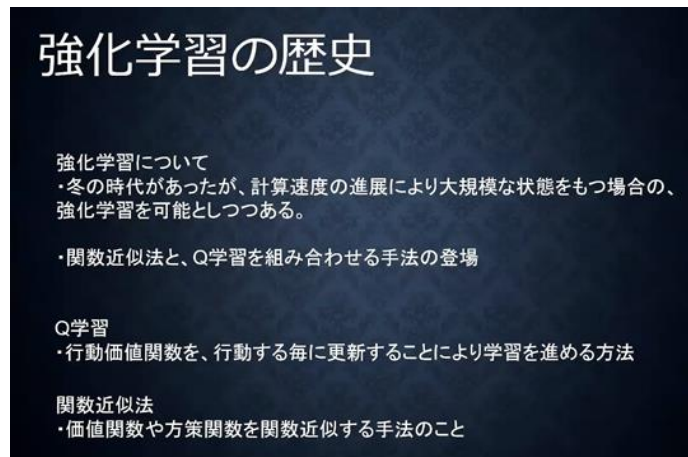
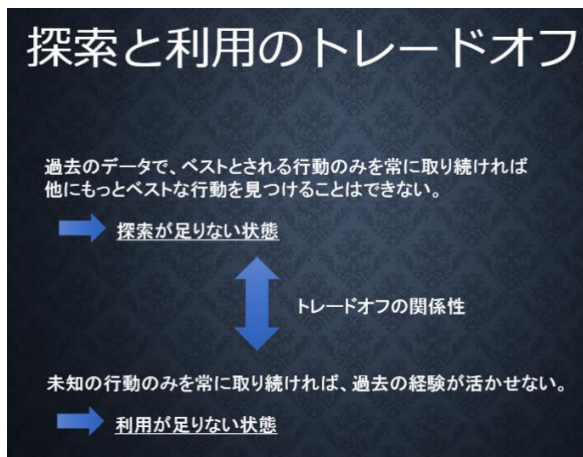


# Stage4『深層学習 Day4』レポート

## ●Section1：強化学習

長期的に報酬(reward)を最大化できる様に、環境(state)の中で行動(action)を選択できるエージェントを作る事を目標とする機械学習の一分野。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。



強化学習では、長期的な報酬を最大化する優れた方策(policy)を見つける事が目標となる。

（教師なし/あり学習では、データに含まれるパターンを見つけ出す、およびデータから予測する事が目標である。）

- ・ 学習初期はランダムな行動を選択する“探索”が中心。
- ・ 学習が進むにつれ、それまでの結果を“利用”して最適な行動を選択し始める。



- ・ 方策関数：  $\pi(s,a)$

方策ベースの強化学習において、ある状態でどの様な行動を探るのか？の確率を与える関数の事。

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（経験を活かす or チャレンジする）？  $\Rightarrow$  その瞬間の行動をどうするか？

$V^\pi(s)$ : 状態価値関数、 $Q^\pi(s,a)$ : 状態+行動関数…今の方策をゴールまで続けた時の報酬予測値が得られる。  $\Rightarrow$  やり続けたら最終的にどうなるか？

◇累積報酬：方策  $\pi$  の良さは、得られる報酬(Reward)の累積で決まる。

$$\text{累積報酬} : U_t = \sum_{k=0}^{\infty} R_{t+k+1} (= R_{t+1} + R_{t+2} + R_{t+3} + \dots)$$

$\gamma$  : 割引率

$$\therefore \text{割引き累積報酬} : U_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} (= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots)$$

$t$ : 現在時刻、  $R_t$ : 時刻  $t$  における報酬

◇方策  $\pi$  の下における価値関数（価値を表す関数）は、

- ・ 状態価値関数  $V^\pi(s_t)$  : 時刻  $t$  での状態  $s_t$  の価値

$$V^\pi(s_t) = E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \right] = E_\pi [R_{t+1} + \gamma V^\pi(s_{t+1})]$$

- ・ 行動価値関数  $Q^\pi(s_t, a_t)$  : 時刻  $t$  での状態  $s_t$  で、行動  $a_t$  を採用する事の価値

$$Q^\pi(s_t, a_t) = E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \right] = E_\pi [R_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1})]$$

※  $E$ …期待値 = (試行によって出る値×値が出る確率) の総和

の 2 種類がある。

◇価値関数の最適化

$$V^{\pi}(s_t) = \max_{a_t} E_{\pi}[R_{t+1} + \gamma V^{\pi}(s_{t+1})]$$

$$Q^{\pi}(s_t, a_t) = E_{\pi}\left[R_{t+1} + \gamma \max_{a_{t+1}} Q^{\pi}(s_{t+1}, a_{t+1})\right]$$

ただし、成立条件として学習対象が「マルコフ性」を満たす必要がある。

マルコフ性… 未来の挙動が現在の値のみによって決定される性質（現在より過去の値は未来に影響を及ぼさない）。

マルコフ性を満たす強化学習は、マルコフ決定過程（MDP）と呼ばれる。

◇方策勾配法

方策を最適化したいが、現実的には困難なケースが多い・・・。

・方策反復法…行動は方策に基づき選択され、方策を更新（方策改善）していく事で方策を最適化する方法の事。“policy ベース”と呼ばれる。

方策  $\pi$  をモデル化（パラメータで表される関数）し、パラメータを最適化する手法

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

※  $\theta$ : パラメータ、 $J_{(\theta)}$ : 最大化したい目的関数(期待収益)

$$J_{(\theta)} = \mathcal{V}_{\pi_{\theta}}(s_0)$$

$s_0$ : 初期状態、 $\mathcal{V}_{\pi_{\theta}}$  は  $\theta$  に依存する状態価値関数

$$\nabla J_{(\theta)} = E_{\pi_{\theta}}[\nabla_{\theta} \log \pi_{\theta}(a|s) Q^{\pi_{\theta}}(s, a)]$$

・価値反復法…方策は「価値が最大となる行動を選択する」という仮定のもとで、

価値を繰り返し更新しながら、価値を推定する方法。“value ベース” と呼ばれる。

◇方策勾配定理

$$\nabla J(\theta) \propto \sum_{s \in S} \mu(s) \sum_{a \in A} \nabla \pi_{\theta}(a|s) Q^{\pi_{\theta}}(s, a)$$

ここで、

$\mu(s)$  : 状態  $s$  に遷移する確率

$\pi_{\theta}(a|s)$  : 状態  $s$  で行動  $a$  をとる確率（方策関数）

$Q_{\pi_{\theta}}(s,a)$  : その価値

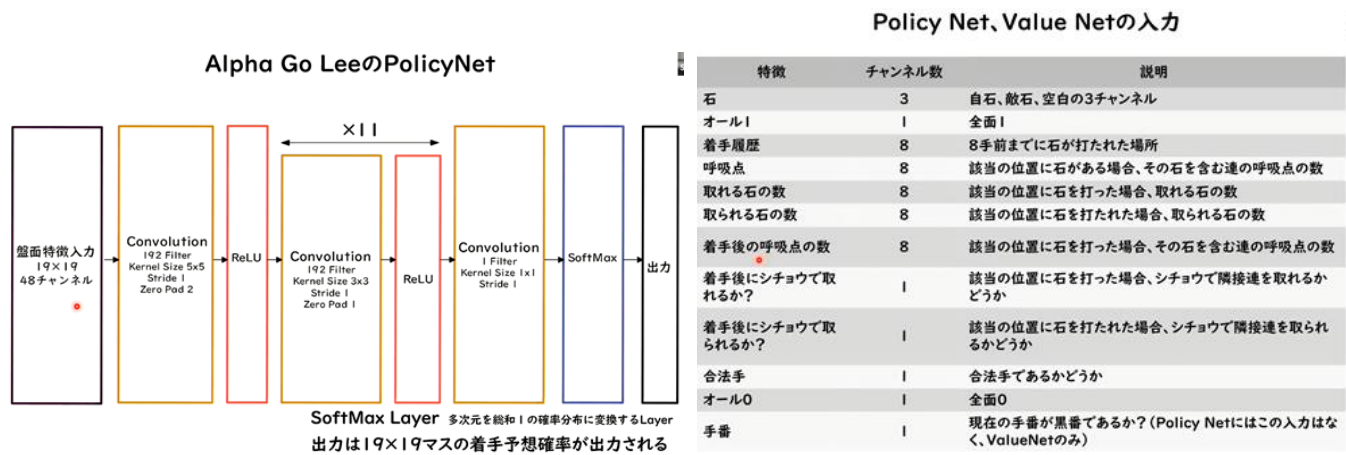
を示す。

方策勾配法は、エージェントの行動確率 $\pi(a|s, \theta)$ をニューラルネットワークで表現するためのテクニック（ $a$  は行動、 $s$  は状態）。

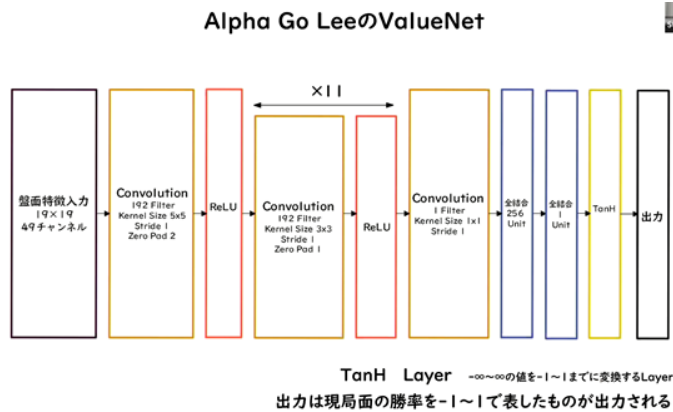
●Section2 : Alpha Go

◇Alpha Go Lee

- ・ PolicyNet（方策の決定）



## ・ ValueNet（優劣の予測：勝率を出力）



## 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

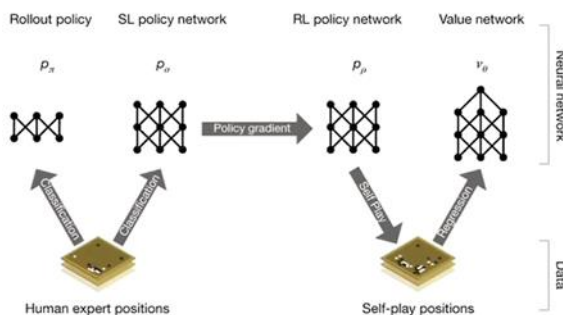
| 特徴       | 次元数   | 説明                                    |
|----------|-------|---------------------------------------|
| 12近傍マッチ  | 1     | 一つ以上の12近傍パターン(下記)にマッチしたかどうか           |
| アタリ救助    | 1     | アタリを逃げる手であるかどうか                       |
| 隣接       | 8     | 直前の着手と隣接したマスへの着手であるか                  |
| 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ヤモンド型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3x3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ヤモンド型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19x19マス分あり、出力はそのマスの着手予想確率となる

## 学習ステップ

- 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



AlphaGoの学習フロー

出典: David Silver: Mastering the game of Go with deep neural networks and tree search  
nature 27 January 2016  
<https://www.nature.com/articles/nature16961>

## ・ PolicyNet の教師あり学習

過去の棋譜データ 3,000 万局分の教師を用意し、教師（棋譜）と同じ着手を予測できるように

## ・ 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（mini batch size:128×

10,000 万回)。

※PolicyPool…PolicyNet の強化学習の過程を 500iteration 毎に記録したもの。

(PolicyNet 同士の対局ではなく、PolicyPool との対局をする理由は、対局に幅を持たせて過学習を防ぐ事)

### ・ 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。教師データ作成手順は、

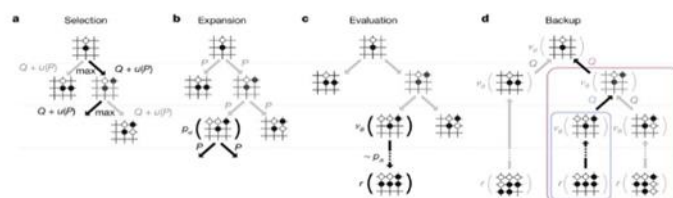
1. 教師あり学習で作成した PolicyNet (SL PolicyNet) で N 手まで打つ。
2. N+1 手目をランダムに選択し、その局面を S(N+1)とする。
3. S(N+1)から RL PolicyNet (強化学習で作成した PolicyNet) で終局まで打ち、その勝敗報酬を R とする。

S(N+1)と R を教師データ対とし、損失関数を平均 2 乗誤差とし、回帰問題として学習 (mini batch size:32×5,000 万回)。

Alpha Go 強化学習において、モンテカルロ木探索が最も有効な探索法。

### Alpha Go (Lee) のモンテカルロ木探索

Alpha Go のモンテカルロ木探索は選択、評価、バックアップ、成長という 4 つのステップで構成される



AlphaGoのモンテカルロ木探索

出典 David Silver Mastering the game of Go with deep neural networks and tree search  
nature 27 January 2016  
<https://www.nature.com/articles/nature16961>

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、 $\sum_b N(s,b)$ は現局面の全合法手が選ばれた数の合計である。

$Q(s,a)$ 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の特徴のひとつである。この選択、着手をLeafノードに到達するまで行う。



# Alpha Go (Lee) のモンテカルロ木探索

Alpha Goのモンテカルロ木探索は選択、評価、バックアップ、成長という4つのステップで構成される

## 評価

着手選択方針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。このときのシミュレーション時にはRollOut方針を使用する。

## 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 $W_v$ 、RollOutでの勝敗値での積算値 $W_r$ が積算され、 $N(s,a)$ と $\Sigma N(s,b)$ 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。

$$Q(s,a) = (1-\lambda) \frac{W_v}{N(s,a)} + \lambda \frac{W_r}{N(s,a)}$$

とし、 $\lambda$ は $0 \leq \lambda \leq 1$ の定数である。AlphaGoLeeでは0.5が使用されている。

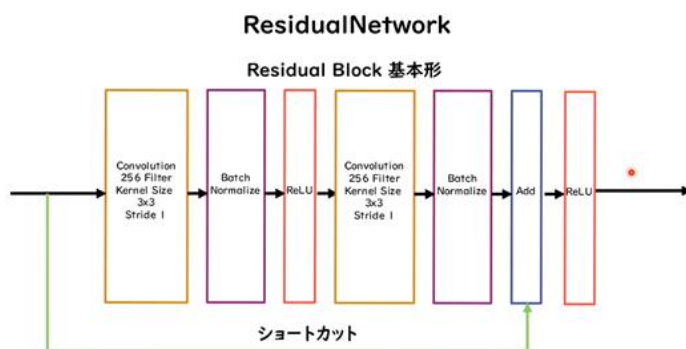
## 成長

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。

## ◇Alpha Go Zero

### ・ Alpha Go Lee との違い

1. 教師あり学習学習を一切行わず、強化学習のみ
2. 特徴入力から人間らしい要素を排除し、石の配置のみ
3. PolicyNet と ValueNet を1つのネットワークに統合
4. Residual Net を導入（勾配爆発/消失を防止）



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

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を使うことにより層数の違うNetworkのアンサンブル効果が得られているという説もある

### 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

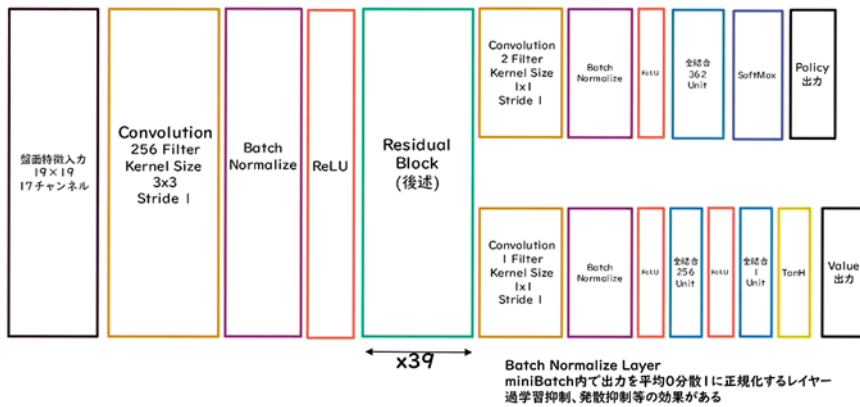
Residual Networkを使うことにより、100層を超えるネットワークでの安定した学習が可能となった

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を使うことにより層数の違うNetworkのアンサンブル効果が得られているという説もある

### 5. モンテカルロ木探索から RollOutPolicy シミュレーションを排除

## Alpha Go ZeroのPolicyValueNet



## Residual Networkの派生形

### 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層のものとは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

### 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、段階的にではなく、各層でFilter数を増やしていくResNet。

## ●Section3：軽量化・高速化技術

分散深層学習：

深層学習は多くのデータをしようしたり、パラメータ調整のために多くの時間を使用する為、高速な計算が求められる。

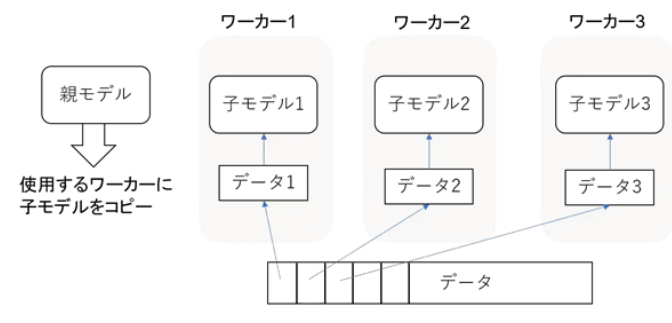
⇒複数の計算資源（ワーカー）を使用し、並列にニューラルネットワークを構成する事で、効率の良い学習を行いたい。

⇒データ並列化・モデル並列化・GPUによる高速化技術は不可欠。



## ◇モデル並列

- ・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

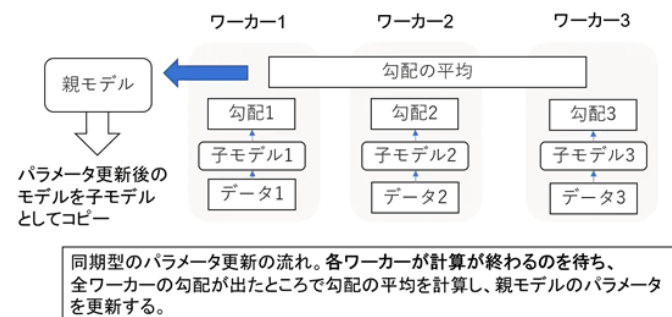


## ◇データ並列

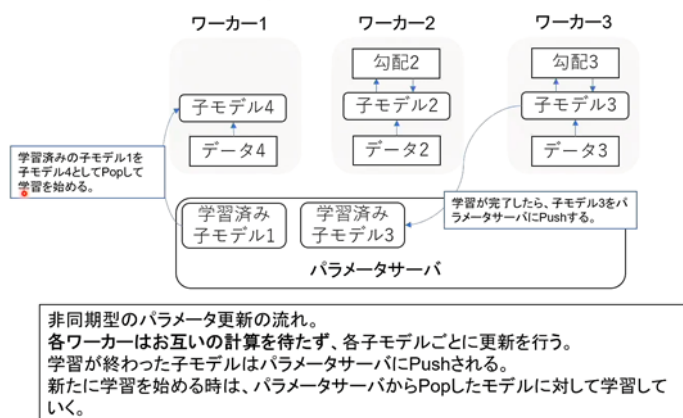
同期型と非同期型の2種類ある。

### 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。



### データ並列化: 非同期型



- ・処理スピードは、お互いのワーカー計算を待たない非同期型の方が速い。
- ・非同期型は最新モデルのパラメータを利用できないので、学習が不安定になる。

⇒現在は、同期型の方が精度が良いことが多い（主流になっている）。

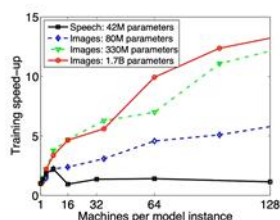
## ◇GPU

### モデル並列化

- ・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。全てのデータで学習が終わった後で、一つのモデルに復元。
- ・モデルが大きい時はモデル並列化を、データが大きい時はデータ並列化をすると良い。



- ・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

- 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われる GPU の総称

- CPU

高性能なコアが少数、複雑で連続的な処理が得意

- GPU

比較的低性能なコアが多数、簡単な並列処理が得意、

NN の学習は単純な行列演算が多いので高速化が可能

## GPGPU 開発環境

- 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。

DeepLearnig 用に提供されているので使いやすい

- 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 社以外の会社(Intel,AMD,ARM など)の GPU からでも使用可能

DeepLearnig 用に特化しているわけではない

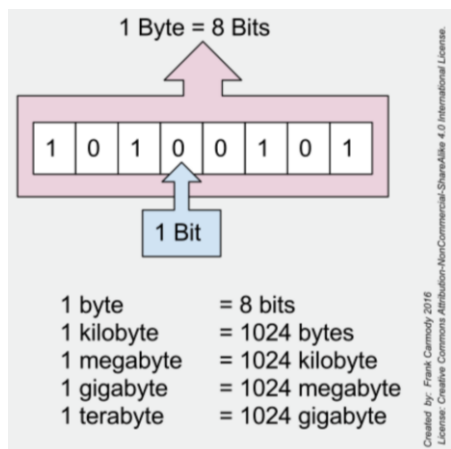
※DeepLearnig フレームワーク(Tendorflow,Pytouch)内で実装されているので、使用時に指定すれば良い。

## ◇量子化(Quautization)

大量のパラメータが必要な大きなネットワークの学習/推論に多くのメモリと演算

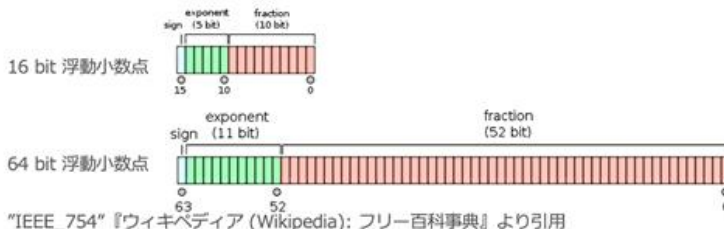
処理が必要となる

⇒64bit 浮動小数点(8byte)を 32bit(4byte)や 16bit(2byte)などの下位の精度に落とす。



## 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量を抑えることができる



・ メリット：計算量低下⇒ 計算高速化、省メモリ化

深層学習で用いられる NVIDIA 社製の GPU の性能は下記のような

16bit 半精度

|                     | 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 |

<https://www.nvidia.com/ja-jp/data-center/tesla-v100/>

<https://www.nvidia.co.jp/object/tesla-p100-ja.html>

※FLOPS…秒あたりの計算回数

・ デメリット：精度の低下

先に述べたようにニューロンが表現できる小数の有効桁が小さくなる



モデルの表現力が低下する

単精度の下限は  $1.175494 \times 10^{-38}$ 、倍精度の下限は  $2.225074 \times 10^{-308}$  になる

⇒ 学習した結果ニューロンが  $1.175494 \times 10^{-38}$  未満の値になる場合など  
重みを表現できなくなる



ただし、実際の問題では倍精度を単精度にしてもほぼ精度は変わらない

(機械学習では、16bit で十分な場合が多い)

## 速度の実験結果

右の図は 32bit と 6bit のモデルで検出を行った画像になる

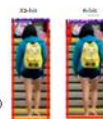
それぞれの検出の時間は下記の表のようになる

| 32 bit  | 6 bit  |
|---------|--------|
| 0.507s  | 0.098s |
| 0.441s  | 0.106s |
| 32.269s | 5.113s |



## 精度の実験結果

右の図は量子化を行い検出した矩形の図になる。  
量子化することにより目的のオブジェクト以外の領域が多くなっている



下記の図は量子化を行わない場合と量子化を行った際の精度になる。

| R-FCN, ResNet-50      | mAP    | R-FCN, ResNet-101     | mAP    |
|-----------------------|--------|-----------------------|--------|
| 4-bit LBW             | 74.37% | 4-bit LBW             | 76.79% |
| 5-bit LBW             | 76.99% | 5-bit LBW             | 77.85% |
| 6-bit LBW             | 77.05% | 6-bit LBW             | 78.24% |
| 32-bit full-precision | 77.66% | 32-bit full-precision | 78.94% |

## ◇蒸留

精度の高いモデルは、ニューロンの規模が多い=多くのメモリと演算処理が必要  
⇒規模の大きなモデルの知識を使い、軽量なモデルの作成を行う。

### モデルの簡約化

学習済みの精度の高いモデルの知識を軽量なモデルへ継承させる



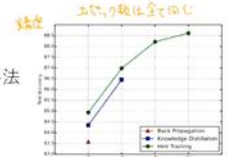
知識の継承により、軽量でありながら複雑なモデルに匹敵する精度のモデルを得ることが期待できる

### 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>.

### 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

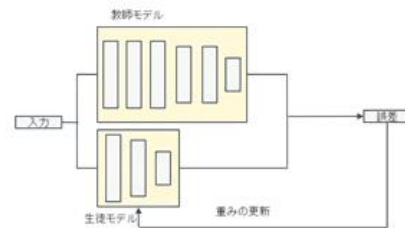
予測精度の高い、複雑なモデルやアンサンブルされたモデル

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ていく

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を更新していく

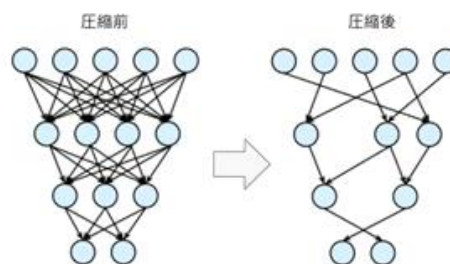


Geoffrey Hinton, Oriol Vinyals, Jeff Dean (2015) 「Distilling the Knowledge in a Neural Network」, <https://arxiv.org/abs/1503.02531>.

## ◇プルーニング

ネットワークが大きくなる=大量のパラメータとなるが、全てのニューロンが精度に寄与しているわけではない・・・。

⇒制度への寄与が少ないニューロンを削減する事で、モデルの軽量化/高速化が可能

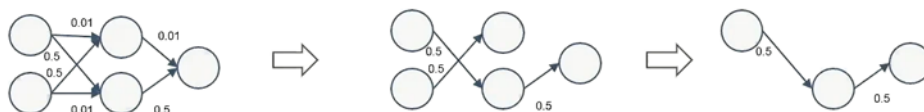


Song Han, Jeff Pool, John Tran, William J. Dally (2015) 「Learning both Weights and Connections for Efficient Neural Networks」, <https://arxiv.org/pdf/1506.02626.pdf>.

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。

下記の例は重みが 0.1 以下のニューロンを削減した



## ニューロン数と精度

STAP

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になる。 $\alpha$  は閾値のパラメータ、閾値は各層の標準偏差  $\sigma$  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

|          | パラメータ削減率 (%) |        |        |        |        |
|----------|--------------|--------|--------|--------|--------|
| $\alpha$ | 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 (%) |        |        |        |        |
|----------|--------------|--------|--------|--------|--------|
| $\alpha$ | 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        | 96.06  | 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>.

## ●Section4：応用モデル（深層学習の適応手法）

### ◇MobileNet（画像認識）：

ディープラーニングは計算量が大きく、計算リソースが高コスト。

通常の畳み込みは、空間方向とチャネル方向の計算を同時に行う。

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、多くのモバイルおよ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。



<https://arxiv.org/pdf/1704.04861.pdf>

### ● 一般的な畳み込みレイヤー

- 入力特徴マップ (チャネル数):  $H \times W \times C$
- 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?

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

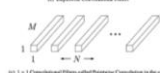
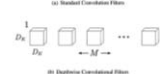


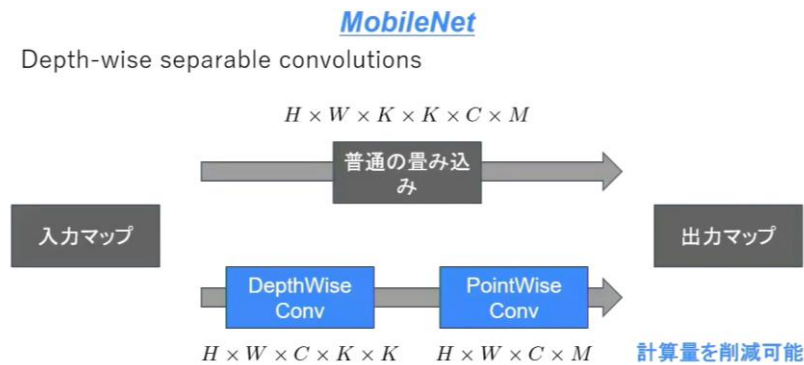
Figure 2. The standard convolutional filters in (a) are replaced by one-layer, depthwise convolutional filters in (b) and grouped convolutional filters in (c) to build a depthwise separable filter.

入力MAP  $H \times W \times C$     カーネル  $K \times K \times C$     フィルタ数  $M$     出力マップ  $H \times W \times M$

⇒モデルの軽量化/高速化/高精度化を実現するモバイルなネットワーク



Depthwise Separable Convolution という手法を用いて計算量を削減している。

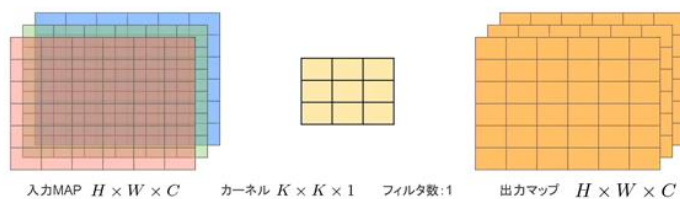


⇒Depthwise Convolution と Pointwise Convolution と呼ばれる演算によって個別に行い、足し合わせる。

- Depthwise Convolution : チャネルごとに空間方向に畳み込む。

#### Depthwise Convolution

- 仕組み
  - 入力マップのチャネルごとに畳み込みを実施
  - 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)



- 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可能
- 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。通常はPW畳み込みとセットで使うことで解決

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

チャネル毎に  $D_K \times D_K \times 1$  のサイズのフィルターを各々用いて計算を行う為、計算量は少なくなる。

- Pointwise Convolution : Depthwise Convolution の出力を、チャネル方向に畳み込む

#### Pointwise Convolution

- 仕組み
  - $1 \times 1$  convとも呼ばれる (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
  - 出力マップ(チャネル数)はフィルタ数だけ作成可能 (任意のサイズが指定可能)

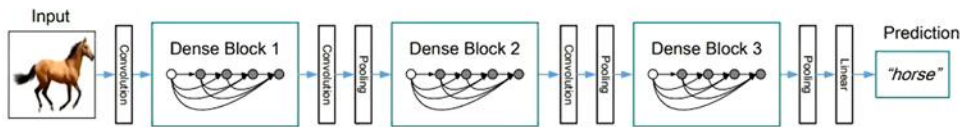


出力チャネル毎に  $1 \times 1 \times M$  サイズのフィルターをそれぞれ用いて計算を行う。



## ◇DenseNet (Dense Convolution Network : 画像認識)

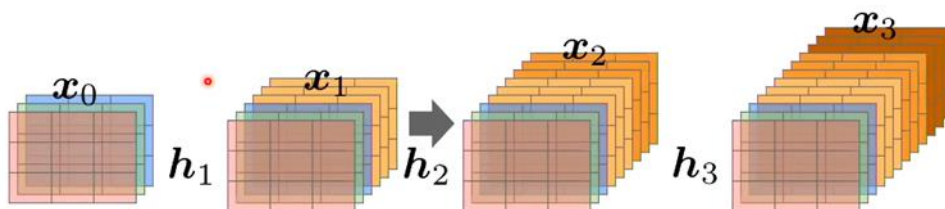
- 初期の畳み込み
- Denseブロック
- 変換レイヤー
- 判別レイヤー



<https://arxiv.org/pdf/1608.06993.pdf>

出力に前層の入力を足し合わせる

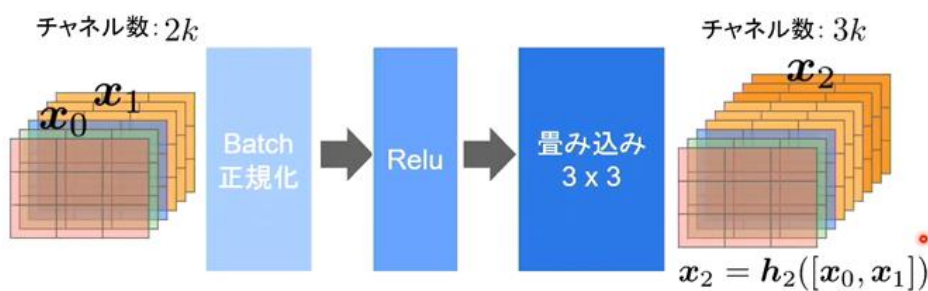
全ての特数量サイズの層を結合する事で、層間の情報の伝達を最大にする。



(少しずつチャンネルが増えていく)

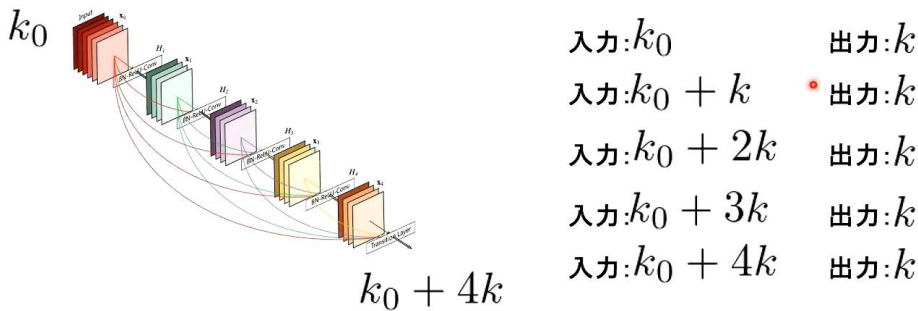
- 特徴マップの入力に対し、下記の処理で出力を計算
  - Batch正規化
  - Relu関数による変換
  - 3 x 3畳み込み層による処理
- 前スライドで計算した出力に入力特徴マップを足し合わせる
  - 入力特徴マップのチャンネル数が  $l \times k$  だった場合、出力は  $(l+1) \times k$  となる
- 第  $l$  層の出力をとすると

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



- Groth Rate (成長率) …ハイパーパラメーター

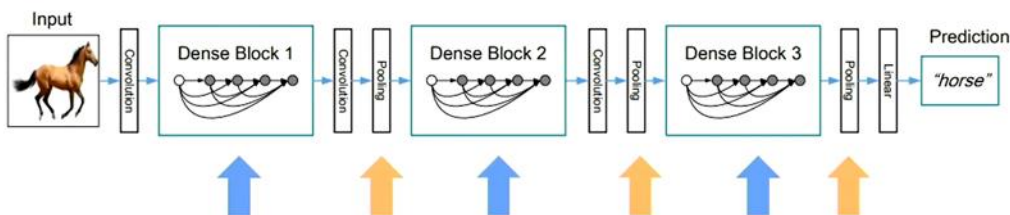
- $k$ をネットワークのgrowth rateと呼ぶ
  - $k$ が大きくなるほど、ネットワークが大きくなるため、小さな整数に設定するのがよい



<https://arxiv.org/pdf/1608.06993.pdf>

Dense Block 内の各ブロック毎に  $k$  個ずつ特徴マップのチャンネル数が増加していく

- Transition Layer
  - CNNでは中間層でチャンネルサイズを変更し
  - 特徴マップのサイズを変更し、ダウンサンプリングを行うため、Transition Layerと呼ばれる層でDense blockをつなぐ



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

<https://arxiv.org/pdf/1608.06993.pdf>

## ◇ResNet

- DenseNet との違い

Dense Block…前方の各層からの出力全てが後方の層への入力となる事に対し、

Residual Block…前1層の入力のみ後方の層へ入力される。  $k$ （成長率）が無い。

## ◇Batch Norm

レイヤー間を流れるデータの分布を、mini バッチ単位で、

平均：0 、 分散：1

になる様に正規化する

⇒ニューラルネットワークにおいて、

- ・学習時間の短縮
- ・初期値への依存低減
- ・過学習の抑制

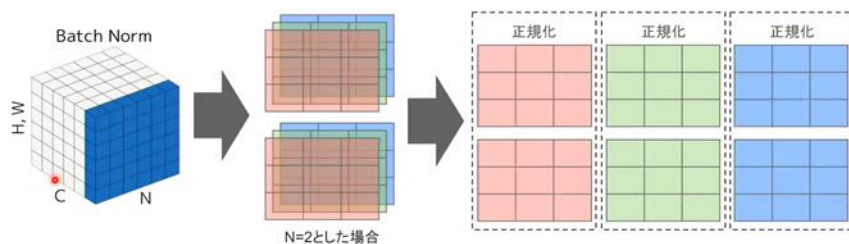
などに効果がある。

〈問題点〉

PC 仕様の制約で、バッチサイズが大きく出来ない条件下では、学習が収束しない事がある。

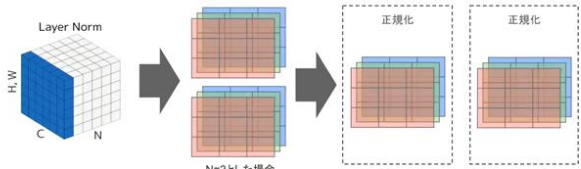
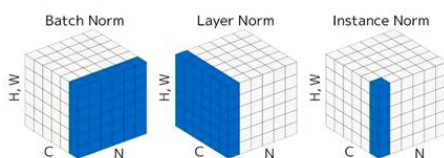
#### Batch Norm.

- $H \times W \times C$ 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（図の青い部分に対応）。チャンネルごとに正規化された特徴マップを出力。
- 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。



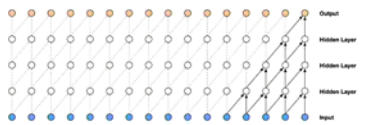
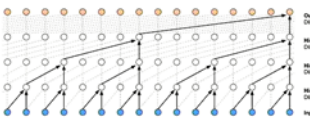
論文参照: <https://arxiv.org/pdf/1803.08494.pdf>

⇒代わりに **Layer Normalization** などの正規化手法が使われる事が多い。

| Layer Norm.  | Batch Norm以外の正規化  |
|--|---|
| <ul style="list-style-type: none"><li>● Layer Norm<ul style="list-style-type: none"><li>○ N個のsampleのうち一つに注目。H x W x Cの全てのpixelが正規化の単位。</li><li>○ RGBの3チャンネルのsampleがN個の場合は、あるsampleを取り出し、全てのチャンネルの平均と分散を求め正規化を実施 (図の青い部分に対応)。特徴マップごとに正規化された特徴マップを出力</li><li>○ 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ていると考えられる。</li></ul></li></ul>  <p>論文参照: <a href="https://arxiv.org/pdf/1803.08494.pdf">https://arxiv.org/pdf/1803.08494.pdf</a></p> | <ul style="list-style-type: none"><li>● Batch Norm<ul style="list-style-type: none"><li>○ 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正規化</li></ul></li><li>● Layer Norm<ul style="list-style-type: none"><li>○ それぞれのsampleの全てのpixelsが同一分布に従うよう正規化</li></ul></li><li>● Instance Norm<ul style="list-style-type: none"><li>○ さらにchannelも同一分布に従うよう正規化</li></ul></li></ul>  <p>N: ミニバッチ数<br/>C: Channel<br/>H, W: Height / Widthをまとめた</p> <p>論文参照: <a href="https://arxiv.org/pdf/1803.08494.pdf">https://arxiv.org/pdf/1803.08494.pdf</a></p> |

## ◇WaveNet

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** の大きな貢献の一つである。

| Wavenet   |
|---|
| <ul style="list-style-type: none"><li>● Wavenetのメインアイディア<ul style="list-style-type: none"><li>○ 時系列データに対して畳み込み(Dilated convolution)を適用する</li><li>○ Dilated convolution<ul style="list-style-type: none"><li>■ 層が深くなるにつれて畳み込むリンクを離す</li><li>■ 受容野を簡単に増やすことができるという利点がある<ul style="list-style-type: none"><li>● 図(右)では、Dilated = 1,2,4,8となっている</li></ul></li></ul></li></ul></li></ul>  <p>Figure 2: Visualization of a stack of causal convolutional layers.</p>  <p>Figure 3: Visualization of a stack of dilated causal convolutional layers.</p> <p>既存の畳み込み</p> <p>畳み込み (Dilated)</p> <p><a href="https://arxiv.org/pdf/1609.03499.pdf">https://arxiv.org/pdf/1609.03499.pdf</a></p> |

提案された新しいConvolution型アーキテクチャはDilated causal convolutionと呼ばれ、結合確率を効率的に学習出来る様になっている。

Dilated causal Convolution を用いる大きな利点は、単純な Convolution Layer と比べて、パラメータ数に対する受容野が広い。