# 深層学習 day4

# Section1: 強化学習

- 強化学習
  - 行動の結果として与えられる利益をもとに、行動を決定する原理を改善していく仕組み
  - 教師なし、あり学習はデータに含まれるパターンを見つけ出すことだが、強化学習は優れた方策を見つけることが目標
- 状態価値関数：状態のみで価値を評価する関数
- 行動価値関数：状態と行動で評価を行う関数
  - 最近もてはやされているのはこちら。
- 方策関数：価値を最大にするような行動を決める関数

## 方策勾配法

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

ニューラルネットワークの学習の式に似ている。
ニューラルネットワークでは誤差を「小さく」するためマイナス記号が付いていたが、
強化学習では期待収益を「大きく」する方向に更新したいのでプラス記号が付いている。
Jとは方策の良さを表す。

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

# Section2: AlphaGo

## Alpha Go LeeのPolicy Net

- 入力として19x19の盤面情報、48チャンネル分を持つ。
  - 石情報が3チャンネル（自石、敵石、空白）、着手履歴8チャンネルなど
- Conv->ReLUを繰り返し、最後にSoftmaxして出力
- どの手が良いかを確率で出力

## Alpha Go LeeのValue Net

- Policy Netとほぼ同じ構造だが、SoftmaxではなくTanHを通って出力となる

- TanHにより-1~1までの値を出力。このまま行って勝てるかどうかを出力。

## RollOutPolicy

- 過去の棋譜データを使用し、教師あり学習させたモデル
- PolicyNetに比べて約1000倍高速だが、精度はPolicy Netの半分程度。

## Alpha Go Zero

1. 教師あり学習を一切行わず強化学習のみ
2. 特徴入力からヒューリスティックなパラメタ（人が役に立ちそうと思っていれたパラメタ）をなくし、石の配置のみにした
3. Policy NetとValueNetを1つのネットワークに統合
4. Residual Netを導入
5. 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

## Residual Network

深いネットワークモデルで、勾配の消失・爆発を防ぐため、
あるブロックでショートカットを設けたもの。
これを1つのブロックとして、何層も重ねることでうまく学習が可能になる。

# Section3: 軽量化・高速化技術

深層学習では毎年10倍計算量が増えるのに対し、
演算器は18~24か月で2倍の性能向上と、演算器の向上が追いついていないため、
軽量化、高速化技術が必須となっている。

## データ並列化

ワーカーを1台のPCに割り当てるといったハード資源を増やして計算負荷を分散する手法。
同期型と非同期型がある。

- 同期型
  - 各ワーカーが終わるのを待ち、勾配の平均を出して親モデルのパラメータを更新
  - ★現在の主流は同期型。性能も非同期型より良い。
- 非同期型
  - 他のワーカーの終了を待たずに、各子モデルごとに更新を行う。
  - 子モデルで更新した結果はパラメータサーバにpush
  - 次の学習を始める際はパラメータサーバから学習済モデルをpopして使用

    - 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

## モデル並列化

ニューラルネットワークの処理モデルの一部を別のハードに処理させる手法。
1つのPCで複数のGPUに割り当てるといったやり方が多い。
ネットワークモデルで枝分かれになっている部分を別のGPUに割り当てたりすることが多い。

## GPUによる並列化

- GPGPU
    - General-purpose on GPU
    - ゲーム用途などグラフィック専門だったGPUを汎用的に使う手法
- GPGPUフレームワーク
    - CUDA: NVIDIAのGPUが使用可能。TensorFlowでも使用されており、現在の主流。
    - OpenCL: NVIDIA以外のGPUも動作可能。

## 量子化

- 量子化(Quantization)
    - パラメータの64bit 浮動小数点数を32-bitなど下位の精度に落とす
    - 精度は落ちるものの、メモリと演算処理の大幅な削減
    - ニューラルネットワークでは半精度の16bitで十分実用レベルの精度が得られる。
- 参考）Tesla V100の演算性能
    - double(FP64): 7.8TFLOPS
    - float(FP32): 15.7TFLOPS
    - half(FP16): ~150TFLOPS

# Section4: 応用モデル

## MobileNet

- 軽量化、高速化、高精度化を実現したモデル
- 計算量の多いConvolutionを以下2つの組合せで軽量化
    - Depthwise Convolution
        - フィルタを1枚とし、出力チャンネル数＝入力チャンネル数とする
        - 計算量はH×W×C×K×K（通常のConvolutionの1/M）
    - Pointwise Convolution
        - フィルタサイズ1x1で畳み込み

    - 計算量はH×W×C×M（通常のConvolutionの1/(k*k)）

## DenseNet

- DenseBlockと呼ばれる処理ブロック内で、以下の処理を繰り返す
    - Batch正規化→ReLU→Conv3x3
    - 出力に入力特徴マップを足し合わせる
    - これを何回繰り返すかをkとし、growth rateと呼ぶ
- Denseブロック後はTransition Layerで増えたチャンネル数を元のサイズに戻す
- ResNetでは前1層の入力のみ後方の層に受け渡していたが、DenseNetでは前の層全てが後方の層へと渡される。

# Section5: Transformer

- Transformer
    - 2017年6月に登場した
    - RNNを使わず、Attentionのみを使用
    - とても少ない計算量でSOTAを達成（SOTA: State of the Art. 最も高精度であることを示す）
    - RNNを使用しないので、単語の位置情報をエンコードして使用
    - 順に処理する必要がないため、従来のNNより並列化が可能

# Section6: 物体検知・セグメンテーション

- 物体検知のフレームワークにはRCNN、Fast RCNN, SSD, YOLOなどがある。

| 種類 | 備考 | 代表的なフレームワーク |
| --- | --- | --- |
| 2段階検出器 | 候補領域の検出とクラス推定を別々に行う。<br>精度高いが遅い | RCNN, SPPNet, RFCN,<br>Mask RCNN |
| 1段階検出器 | 候補領域の検出とクラス推定を同時に行う。<br>精度低いが高速 | DetectorNet, SSD, YOLO,<br>CornerNet |