進捗報告

1 今週やったこと

- Augment 操作の Random Apply の実装
- 問題の定式化

1.1 Augment 操作の Random Apply の実装

森先生と以前話していた内容を実装．方針としては訓練データ x_i に対して拡張操作 f を 1 回だけ適用したデータ $f(x_i)$ でデータを拡張する．ただし，計算量の増加を抑えるために $\mathcal{F}(D) = \bigcup_{f\in\mathcal{F}}\{(f(x), y) : (x, y) \in D\}$ とするのではなく，確率的に候補となる操作を抽出し，適用する or 適用しないを選択し，近似的に操作候補の集合を評価する．ここで操作が適用されなかった場合，オリジナル画像から変化しないことを意味する．現在の適用確率は (操作候補数)/(操作候補数+1) としてある．つまり，x_i, $f_1(x_i)$, $f_2(x_i)$... が等確率で選択されることになる．また，各操作の適用強度に関しては先行研究の良い精度を出したものを参考に partial 関数でハイパラとして固定してある (とはいえ，けっこう値がばらばらなので雑に体感の平均値をとっている程度．ここも将来的には最適化タスクに加えたい)．基本的な GA のタスク設計および実装はひとまずこれくらいで精度検証はサーバが使えるようになってからという感じで．レポは `https://github.com/1g-hub/DEAP_CLS` に．

1.2 問題の定式化

先行研究の問題定式をまとめた．参考サイト [1]

この定式を GA の問題に置き換えるのはサブポリシーとして扱う遺伝子型の改良が必要ではあるが，概ね立式としては以下のような感じになりそう．今のままの組み合わせ問題として扱う場合には下の枠組みよりも AutoAugment の child model を学習する方式のほうがフィットしている．

1.2.1 探索空間

先行研究では $\mathbb{O}$ を入力空間 $\mathcal{X}$ 上の拡張操作 $\mathcal{O} : \mathcal{X} \to \mathcal{X}$ の集合とする．各オペレーション $\mathcal{O}$ は確率 p と強度 λ をもつ．S を sub-policy の集合として，sub-policy $\tau \in S$ は N_τ 個の 操作集合 $\{\bar{\mathcal{O}}_n^{(\tau)}(x; p_n^{(\tau)}, \lambda_n^{(\tau)}) : n = 1, ..., N_\tau\}$ として定義される．各 operation は入力画像に対して $n = 1$ から順番に確率 p に従って次のように適用される．

$$\bar{\mathcal{O}}_n(x; p, \lambda) = \begin{cases} \mathcal{O}_n(x; \lambda) : \text{with probability p} \\ x : \text{with probability } 1 - \text{p} \end{cases}$$

よって sub-policy $\tau(x)$ の出力は

$$\tilde{x}_{(n)} = \bar{\mathcal{O}}_n^{(\tau)}(\tilde{x}_{(n-1)}), n = 1, ..., N_\tau$$

と定義される．

1.2.2 探索戦略

$\mathcal{D}$ を $\mathcal{X} \times \mathcal{Y}$ 上の確率分布とし，$D \in \mathcal{D}$ をデータセットとする．パラメータ θ をもつ分類モデル $\mathcal{M}(\cdot|\theta)$ の期待される精度と損失を $\mathcal{R}(\theta|D), \mathcal{L}(\theta|D)$ と定義する．

任意のペア D_{train} と D_{valid} が与えられたとき，目標はモデルの汎化性能を向上させることとなる．ここでは D_{train} と D_{valid} の分布を一致させるような augmentation の policy を見つけることを目標とする．しかし，すべての policy の候補において 2 つの分布の密度関数を比較することは不可能であるため，ここではあるデータセットがもう一方のデータセットのパターンにどれだけ従うかを両方のデータセットに対するモデルの予測を使って測る．具体的には D_{train} を $D_{\mathcal{M}}$ と $D_{\mathcal{A}}$ の 2 つに分け， θ の学習と $\mathcal{T}$ の探索に利用する．policy の探索のために，下記の目的関数を定義する．

$$\mathcal{T}_* = \arg\max_{\mathcal{T}} \mathcal{R}(\theta^*|\mathcal{T}(D_{\mathcal{A}}))$$

ただし，モデルパラメータ θ^* は $D_{\mathcal{M}}$ 上で学習した結果とする．この目的関数は近似的に $D_{\mathcal{M}}$ と $\mathcal{T}(D_{\mathcal{A}})$ 間の分布の距離をモデルのパフォーマンスの最大化という観点で最小化している．

Fast AutoAugment では以下の手続きをとる．

- まず K-fold stratified shuffling により D_{train} を $D_{\text{train}}^{(1)}, ..., D_{\text{train}}^{(K)}$ の K 個に分割する．各 $D_{\text{train}}^{(k)}$ は 2 つのデータセット $D_{\mathcal{M}}^{(k)}$ と $D_{\mathcal{A}}^{(k)}$ からなる．
- 次に，$D_{\mathcal{M}}$ を使い， data augmentation なしでモデルを学習する．
- 学習後， $1 \leq t \leq T$ の間 B 個の policy の候補 $\mathcal{B} = \mathcal{T}_1, ..., \mathcal{T}_B$ をベイズ最適化により探索空間 $\mathcal{S}$ から探索する．より具体的には $\mathcal{T}(D_{\mathcal{A}})$ での $\mathcal{L}(\theta|\cdot)$ を最小化するように適用確率 $\{p_1, ..., p_{N_t}\}$ と強度 $\{\lambda_1, ..., \lambda_{N_t}\}$ をもつ policy $\mathcal{T} = \{\tau_1, ..., \tau_{N_t}\}$ を探索する．

肝は policy の探索の間モデルのパラメータを学習する必要が一切なく，これによって実行速度を大幅に改善している点．また，それと同時に AutoAugment の child model の概念をなくし，実際に学習したいネットワークに対して直接 policy を探索することを可能にしている．

探索終了後は $\mathcal{B}$ の中から上位 N 個の policy を選び，それらを $\mathcal{T}_t$ とする．その後 $\mathcal{T}_t$ を $\mathcal{T}_*$ に併合し，$\mathcal{T}_*$ を得る．最終的には $\mathcal{T}_*$ を D_{train} 全体に対して適用してモデルを学習する．

policy は学習済みモデルに対してモデルの精度を上げるように探索されているため，次の関係を満たすことが期待される．

$$\mathcal{R}(\theta|\mathcal{T}_*(D_{\mathcal{A}})) \geq \mathcal{R}(\theta|D_{\mathcal{A}})$$

2 来週のタスク

実験は自粛期間が明けてからでないとどうしようもないので，ひとまずは論文のかけるところから少しずつ書いていくという形で．

参考文献

[1] Fast AutoAugment を読んだのでメモ. http://peluigi.hatenablog.com/entry/2020/03/09/131252.