進捗報告

1　今週やったこと

- 論文読んだ
- CNN の縮小

2　論文

NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING

2.1　導入

アーキテクチャの設計には、専門知識と十分な時間が必要

ニューラルアーキテクチャ探索　文字列をネットワークの構造に変換

1. コントローラ（RNN）で文字列を生成
2. ネットワーク生成
3. 訓練して検証データの精度を得る
4. 精度の勾配からコントローラを更新

2.2　手法

2.2.1　コントローラリカレントニューラルネットワークを用いたモデル記述の生成

コントローラからニューラルネットワークのハイパーパラメータを生成

例）Conv 層のみのネットワーク
RNN でフィルタの高さ、幅、ストライドの高さ、幅、フィルタ数をレイヤーごとに予測

2.2.2　REINFORCE による学習

コントローラ自体の学習は, 方策勾配法系の REINFORCE で θ を更新

並列性と非同期更新で訓練を加速　子ネットワークの訓練には何時間もかかる．分散訓練と非同期パラメータ更新を使用.

- パラメータサーバ　s 個：パラメータを保存
- コントローラ（レプリカ）　k 個：　θ の勾配を計算
- 子ネットワーク　m 個：勾配のために増やす？、並列実行

2.2.3　スキップ接続や他のレイヤを用いたアーキテクチャの複雑性の向上

- N - 1 のコンテンツベースのシグモイドを持つアンカーポイントを追加
  接続する必要がある前のレイヤーを示す
- 学習率、畳み込み層以外の層（層のタイプとそのハイパーパラメータ）の追加

コンパイルの失敗　レイヤの互換性がない, レイヤが入力・出力を持たないなどの理由
回避策は

1. 入力がないとき、画像を入力レイヤとして使用する
2. 接続されていないすべてのレイヤ出力を連結した最終レイヤは隠れていた状態を分類器に送る
3. 連結する入力レイヤのサイズが異なる場合、小さいほうを 0 で埋める

2.2.4　リカレントセル・アーキテクチャの生成

(リカレントネットワークを生成する場合)

リカレントセル　入力 (x_t, h_{t-1}, c_{t-1}) → 出力 (h_t, c_t)

ノード 組み合わせ方法（加算、アダマール積）＋活性化関数（tanh, sigmoid）

基底数 ノードの数。実験では基底数 8

コントローラで

- ノードの設定 x 基底数
- セルへ挿入するノード設定
- 状態ノード (c_{t-1}, c_t) の接続位置

を生成し、計算グラフ（リカレントセル）を組み立てる

2.3 実験と結果

2.3.1 Cifar-10 の畳み込みアーキテクチャの学習

データセット オーギュメント手順は、アップサンプル＝＞ランダムクロップ＝＞ランダム水平反転

探索空間 ReLU, Batch Norm, スキップ接続を持つ畳み込みアーキテクチャ。畳み込み層はカーネルサイズ縦横 [1, 3, 5, 7], チャンネル数 [24, 36, 48, 64] から選択、ストライドは 1 固定と [1, 2, 3] から選択する実験に分ける

トレーニング詳細 コントローラＲＮＮ: 2 層 LSTM(各層の隠れユニット 35 個), 初期重みは-0.08　0.08、オプティマイザは Adam(lr=0.0006)、スケジューラーは子モデルの層数を増やす (6layer + 2layer / 1600 sample)
分散学習：パラメータサーバ数 S=20, コントローラレプリカ数 K=100, 子レプリカ数 m=8　(800 のネットワーク同時学習)
50 epoch 訓練
報酬：最後の 5 epoch の検証精度の最大値の 3 乗
検証セット/ 訓練セット：5000 / 45000
子モデルの訓練：Momentum Optimizer(lr=0.1, weight decay=1e-4, momentum=0.9)

結果

1. 12800 のアーキテクチャを訓練
2. 最高精度のアーキテクチャを見つける
3. グリッド探索：学習率、重みの減衰、Batch Norm の ϵ、学習率の減衰
4. 検証精度を計算

ｖ１：プレーン Error rate 5.50
ｖ２：ストライドを予測 Error rate 6.01
ｖ３：プーリング層 Error rate 4.47

2.3.2 Penn Treebank のリカレントセルの学習

データセット　Penn Treebank 正則化手法（過学習を抑制）: embedding dropout, recurrent dropout, 入力と出力 embedding の共有

探索空間　コントローラ：組み合わせ法 (add, elem mult) ＋活性化関数 (identity, tanh, sigmoid, relu) 基数 8、6x10[1]6

訓練の詳細　Cifar-10 ほぼ同様
最低の validation perplexity となる RNN セルを選択グリッド探索：学習率、重みの初期値、ドロップアウト率、減衰 epoch

結果　3.6 perplexity の改善, 2 倍の高速化

3 考察

基本的な考え方は, コントローラにアーキテクチャのパラメータを生成させること。RNN だと長さを自由に変えられるので適している。ほかの強化学習手法を使うこともあるらしい。

1. ネットワークアーキテクチャの探索
2. ハイパーパラメータ調整
3. ネットワークの重み学習

普通は 2. 3. だけ行うが, NAS では全て学習する.
学習には膨大な時間がかかるのを、分散訓練で対処しているが, NAS では GPU 800 台で 28 日かかっている. NAS を参考にした ENAS では, 重みを転移学習することで, GPU 1 台で半日程度で訓練できるらしい.

分からなかったこと

- RNN の仕組みと内部の計算グラフ（使ったことがないので試してみる）
- コントローラによるネットワークの組み立て（github にコードがある？）
- GPU の分散学習の具体的な手順

4　CNN の縮小

前回の CNN は学習に 1 時間近くを必要とするためより高速なネットワークを探した．単純にレイヤーを 16 層から 9 層に削減することで, 時間と精度がどう変化するかを実験した. 5 分で精度が 7 割, 30 分で 9 割近くの CNN に改善できた.

5　今後の予定

- RNN でネットワークを作るシステムの調査と構築

6　ソースコード

載せていなかった, 前回のソースコードです
https://github.com/tatsuya-sugiyama/WeeklyReport/blob/2020_0626/report/2020_06_26/5%20to%2010.ipynb