Theoretical Science Group

理論科学グループ

部報 309 号
— 駒場祭パンフレット号 —

目　次

展示企画

企画紹介 2048 AIs

@semiexp

あらすじ

2 年の semiexp です．駒場祭で展示する (されるはずの) 2048 AI 展示について紹介します．

2048 とは

2048 とは，少し前に流行ったパズルゲームです．ルールは単純で，

- 4×4 の盤面があり，各マスは空きマスもしくは，2 の累乗数のタイルが置かれている．
- 各ターン，上下左右方向に重力をかけることができる．タイルは重力に引かれる．
- このとき，同じ数のタイルがぶつかると，2 倍の数の 1 つのタイルになる．この際，新たにできるタイルの数だけ得点が入る．
- ターンの終わりに，ランダムに 2 か 4 のタイルが出現する．
- 盤面がいっぱいになり，動かせなくなったら (動かしてくっつくタイルがなくなったら) 終了．

というものです．

このゲームは，どんなにうまく操作をしても有限回で終了することを明確に示すことができます．しかし，理論上は 2^{16} くらいを作るのは可能ですが，実際はゲームが進むと大きい数のタイルで盤面がどんどん埋まっていき，2^{16} どころか 2^{14} を作るのも困難です．このゲームの名前「2048 $(= 2^{11})$」の示すとおり，ゲームの最初の目標は 2048 のタイルを作ることになります．

遊び方

AI の種類を選ぶボタンがついているので，それをクリックすると自動でプレイが始まります．なお，AI を切り替えても盤面は自動ではリセットされません．そのため，ゲームの途中で AI を切り替えてプレイさせることも可能です．

AI の動作

公開されているブラウザ版 2048 を改造し，いくつかの AI を導入しました．

- 単純評価関数
  実際に遊んでみると，大きい数があまり真ん中でうろうろしていると邪魔ということがわかります．このバージョンの AI では，動かした直後の盤面の評価関数を，$\sum \log_2$(タイルに書かれた数) $\times$ (そのタイルのマスの隅からの距離) のようにして，これを最小化する方向に移動を行うようにしています．
- 評価関数+ミニマックス法
  単純評価関数バージョンでは，1 手の先読みしかしていません．このバージョンでは，ミニマックス法[1] により「移動した後数手以内に手詰まりになる確率」も求めるようにして，これも評価関数に組み込んでいます．
- モンテカルロ法
  このゲームでは，移動の方向によってすぐに有利，不利が現れにくいという特徴があることがわかります．そのため，ある盤面からどれくらい得点を得られるかを調べるときに,「動かせる方向をランダムに選んで動かすことを，ゲーム終了まで繰り返す」という (一見いい加減に見える方法で) かなりうまくいきます．このバージョンの AI では，評価関数としてこのランダム動の後のスコアのみを用いています．
- 評価関数+モンテカルロ法
  モンテカルロ法の評価関数によるスコアと，単純評価関数によるスコアを併用します．これにより，大きい数をできるだけ隅に寄せようとする傾向が強くなります．

[1] 厳密には，自動で置かれるタイルがランダムに置かれるとして手詰まりの確率の最小値を求めているので，ミニマックス法とは少し違います

Alea jacta est.

umedaikiti

Alea jacta est. について

駒場祭で展示する (予定) の”Alea jacta est.”はオープンソースの連続音声認識エンジン Julius(http://julius.sourceforge.jp/) を利用した、音声認識による将棋プログラムです。棋譜での表記のように「7 六歩」などと喋るとそれを認識して駒を動かします。

名前について

”Alea jacta est.”は古典ラテン語で「賽は投げられた」という意味でガイウス・ユリウス・カエサルが軍を率いてルビコン川を渡る際に言ったとされています。音声認識のライブラリの名前である Julius にちなんでいます。

遊び方

マイクに向かって駒の動かし方を話してください。駒の動かし方は棋譜の表記方法 (http://www.shogi.or.jp/faq/kihuhyouki.html) にしたがって指定してください。持ち駒ももちろん使えます。詰みの判定や対局終了の判断はしていないので玉が取られたとしても続きます。終了するときは Ctrl-C で強制終了してください。

指示の方法

指示文は

1. 到達地点の筋
2. 到達地点の段
3. 駒の種類

4. 駒の相対位置
5. 駒の動作
6. 成・不成・打

の順番に並べたものです。

左右方向の番号を筋、前後方向の番号を段と呼びます。大抵の場合筋は算用数字、段は漢数字で表記します。先手からみると「1 一」は右上に来ます。一手前と同じ位置に移動するとき (相手の駒を取るとき) は 1,2 の代わりに「同」と言います (このプログラムでは 1,2 の部分をそのまま「筋、段」で指示しても許容します)。4,5,6 は複数の動きが考えられるときに区別するためのもので駒の動かし方によっては必要ありません。

駒の相対位置

右 指す側から見て右側の駒を動かした場合
左 指す側から見て左側の駒を動かした場合

駒の「動作」

上 1 段以上、上に動く
寄 1 マス以上、横に動く
引 1 段以上、下に動く

持ち駒を打つとき

打つ駒と同じ種類の駒で打つ場所に移動できるものがある場合、駒の移動と持ち駒を区別するため「打」をつけます。そこに移動できる駒がないときは「打」はつけません。

駒を移動させるときで、移動した駒が成れるとき

「成」「不成」を必ずつけます。ルール上成らなくてはならない場合でも「成」「不成」はつけます。

同じ地点に複数の駒が移動可能なとき

まず、「上」「寄」「引」をつけて駒が一つに定まるときは「上」「寄」「引」をつけます。

そうでない場合、「左」「右」で区別できる場合、「左」「右」をつけます。金銀が横に 2 枚以上並んでいる場合は 1 段上に上がる時に「直」をつけます。竜馬は「直」は使わず「左」「右」で区別します。

それでも区別できない時は「上」「寄」「引」と「左」「右」を組み合わせて駒が一つに定まるようにします。

プログラムの処理について

プログラムは起動後、JuliusLib の初期化やその他の初期化を行います。JuliusLib の初期化には文法ファイルの指定も含まれます。そして、音声を認識して一手ずつ駒を動かす処理のループに入ります。このループでは JuliusLib から音声認識の結果を受け取り、結果がエラーでなければそれを解析して駒を動かす、という処理を繰り返します。

駒をどう動かすべきかの解析についての処理は大まかに言うと

1. 駒の種類と行き先を決める
2. そこに移動できる駒を全て探す
3. そこに移動できる駒それぞれについて、その駒を動かすとした場合の「正しい指示文」を作成する
4. 「正しい指示文」と入力を比較して、一致するものがあればその駒を動かし、なければ何もしない

という流れになっています。

1 の処理は入力文の一部を取り出すだけです。入力が「3 二金上」なら金を 3 二に、その次が「同銀」なら銀を一手前と同じ位置である 3 二に移動させる、と分かります。

2 の処理は駒によって違うので少し面倒ですが、同じ方向にいくらでも動けるタイプの動きとそうでない動きに分けて考え、それぞれに対応する 2 つの関数を用意し、駒ごとに別の配列を渡すことで探すことができるようにしました。

3 の処理は 2 の処理の結果、移動できる駒の数が

0 個のとき 持ち駒を打つ (「打」は省略されている)
1 個以上で「打」と指示されているとき 持ち駒を打つ
1 個で「打」と指示されていないとき その駒を移動させる
2 個以上で「打」と指示されていないとき 「正しい指示文」を作成し、比較する

となります。駒を移動させるとき、駒が成ることができる場合、「成」「不成」が指示される必要があることに注意します。

複数の駒が移動できる時の「正しい指示文」の作り方ですが、`http://www.kmc.gr.jp/~okuji/kifu.html` をかなり参考にしました。

1. それぞれの駒について「移動種別」を作成します。移動種別とは筋の移動方向を表す「左・中・右」と段の移動方向を表す「上・寄・引」の一つずつの組み合わせのことです。駒の位置を筋の番号、段の番号の順に (x, y) のように表すと、駒が (x_1, y_1) から (x_2, y_2) に移動

するとき、先手から見ると $x_1 > x_2$ が左、$x_1 = x_2$ が中、$x_1 < x_2$ が右で、$y_1 > y_2$ が上、$y_1 = y_2$ が寄、$y_1 < y_2$ が引です。打つ人から見た方向で表すので後手では逆になります。また、筋移動情報と段移動情報のそれぞれについて、同じ情報を持つ駒がいくつあるかを数えておきます。

2. 移動先へ移動可能な駒の中で同一の段移動情報を持つ駒が一つしか無い場合は筋移動情報を省略します。
3. 移動先へ移動可能な駒の中で同一の筋移動情報を持つ駒が一つしか無い場合は段移動情報を省略します。
4. 省略の結果、「中」「中上」が残った場合、竜か馬ならもう一方の駒との位置関係に合わせて「左」または「右」に変更し、そうでなければ「直」に変更します。

こうしてできた移動情報が相対位置と動作の指示になります。

文字認識エンジン Shrift OCR Engine

levelfour

昨今のスマートフォン、タブレット端末といったタッチ型デバイスの普及に伴って，手書き文字認識の重要性が高まっている．既に IME 等の手書き入力では文字認識が非常に高い精度で実用化されているが，複数文字の同時認識技術は未だ発展途上といったところである．Shrift OCR Engine は複数の手書き文字（差し詰め手書き文字列といったところか）を認識することを意図して設計された，OCR エンジンである．

OCR

OCR とは Optical Character Recognition の略で，和訳すると「光学文字認識」となる．OCR 分野自体は非常に古くから（といっても IT のスパンで見ると，という意味だが）研究されており，近年のスマートフォン等の台頭によりその重要性を増している．まずはいくつかの既存プロジェクトを紹介する．

Tesseract-OCR

Tesseract-OCR(https://code.google.com/p/tesseract-ocr/) とは Google の OSS の一つである．このプロジェクトはアナログ書籍の電子化を主目的として進められており，認識対象は基本的には活字である．対応言語も多く，勿論日本語も対応している．英語の精度自体は非常に高く，入力画像の状態にもよるが誤認識は少ない．ところが，日本語をはじめとした英語圏外の言語の対応状況はあまり芳しくなく,「語」が「言吾」,「ル」が「ノレ」の 2 文字に分かれてしまう等といった誤認が目立つ．勿論日本語の文字空間が巨大であることを鑑みると十分に大きな成果を残していると言える．

Tomoe

Tomoe(http://tomoe.sourceforge.jp/cgi-bin/ja/blog/index.rb) は Tegaki Online MOji-ninshiki Engine の略で，その名の通り日本語の手書き文字認識エンジンである．データや辞書も豊富のよう．

Shrift の概要

Shrift は日本語の手書き文字列を認識するために開発された（している）文字認識エンジンである．開発に着手してから 2ヶ月程であるので未だ実用化に至ったわけでは決してないのだが，想定している主な用途としては，ノートアプリとの連携や，アナログのスケジュール帳と Google カレンダー等の同期といったことを考えている．

Shrift はサイボウズが 2014 年度夏に開催したサイボウズ・ラボユース Hackathon 東京大会にて開発した．3 日間の開発期間だったことも踏まえ，現在は日本語の中でもひらがなのみに認識対象を絞って開発されている．精度の向上に成功した後に，カタカナや漢字に対象文字種を広げることを考えている．

名前の由来はドイツ語の Schrift(=文字) である．しかし，github のリポジトリを作成するときに c を抜かすというスペルミスしてしまったので，その名前を引きずっている．蛇足だが，shrift は英語で「告解/赦罪」といった意味である．つまりどういうことかというと，スペルミスしてごめんなさい．

Shrift のアルゴリズム

実装の概要

Shrift は複数文字の認識を同時に行うため，文字抽出と文字認識という 2 段階の処理を行っている．

文字抽出

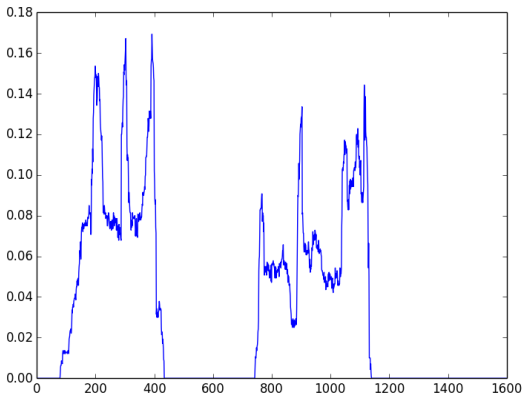
認識対象として入力される画像は文字列を含んでいるため，画像の中から上手に一文字ずつ文字領域を抽出しなければならない．これが文字抽出の段階である．

基本的な考え方は単純で，まずは行方向を検出して行を抽出し，次に各行ごとに文字方向を検出して一文字ずつ抽出する．

抽出処理ではスペクトルから山の範囲を決定して抽出を行っている．図 3.1 の (a) が入力画像である．これを横方向に平均をとると (b) のようになる．この濃淡をスペクトルとしてグラフにしたのが (c) である．こうするとどの範囲が 1 行なのかわかる．ここから行を抽出して縦方向に同様の処理を施したものが図 3.2 である．(ただし図 3.1 と 3.2 の (b) は見やすさを考慮して，実際よりもコントラストを大きくしている)

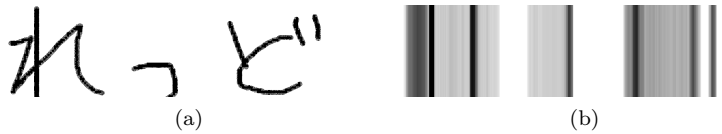
「い」「は」「ふ」のように縦に見たときにスペクトルが 2 つの山になってしまう文字については，まず 2 つの山にわけた抽出候補と 1 つの山にまとめた抽出候補を用意し，両方で文字認識をさせた後に尤度の高い方を採用するようになっている.「う」「え」のように横に見たときに 2 つの山に分かれてしまう文字についても同様である．

(a) (b)

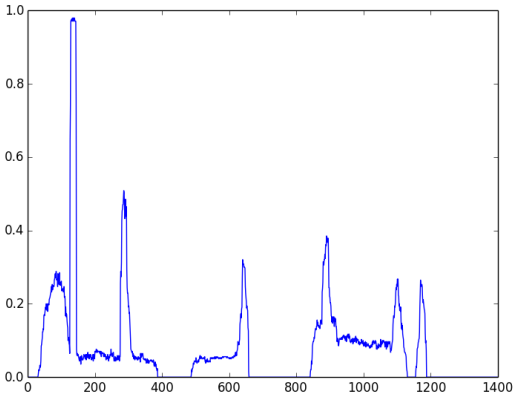


(c)

図 3.1: 行抽出の様子



(a) (b)



(c)

図 3.2: 列抽出の様子

文字認識

当然のことながら文字認識は自動化しなければならないため，機械学習を用いて実装されている．機械学習と聞くと漠然としたイメージを抱く方もいるかもしれないが，基本的なコンセプトとしては既存の学習データを元にデータをうまく分類できる分類器をつくることである．文字認識の場合は，既存の手書き文字データ群を学習させ，それらを正しく分類できるような分類器をつくっておけば，未知の文字データを入力されたときに正しく分類できるという方向になる．機械学習の詳細については，現在執筆中の拙著の記事 (https://github.com/levelfour/machine-learning-2014/wiki/第 1 回#機械学習の基礎) があるので参考になれば幸いである．

機械学習の学習アルゴリズムとして採用したのは，RandomForest である．RandomForest を採用したことに理論的な背景は特別にあるわけではなく，単に実験的に高い精度が得られたので採用している．

また，特徴抽出は入力文字を 20×20 でグリッド状に区切り，400 次元の特徴ベクトルとしている (図 3.3 を参照)．

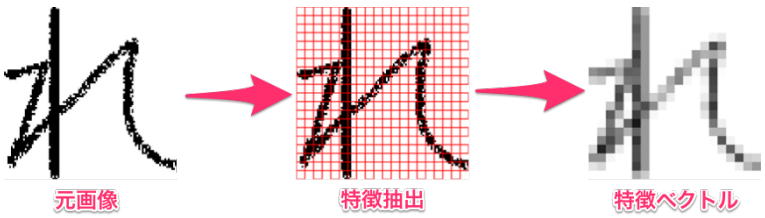


図 3.3: 特徴抽出の様子

問題点・改善点

認識精度

現在の状態では決して認識精度が高いとは言い難い．その原因は色々あるが，まずは文字抽出がそこまで正確ではないことが挙げられる．

現在は考えられうる抽出候補をすべて総当たり的に認識にかけることで尤もらしい抽出結果を選択するようになっているが，実際問題として認識対象文字列長が長くなるにつれて処理時間が指数関数的に増大してしまう．そういったことを考慮するとすべての抽出候補を検討するというのは現実的ではなく，入力文字列画像のみから抽出できることが望ましい．この記事（http://www.ohnishi.nuie.nagoya-u.ac.jp/ kudo/research/ifv/characters.segment.htm）では視覚の誘導場を利用した文字抽出方法を提案しているので，参考にしてみると面白いかもしれない．

また，文字認識自体の精度に関しては，圧倒的に学習データ数が不足しているのでまずは学習データを「偏りなく」増やすことを考えたい．そして，各段階におけるパラメータがあるので，そういったパラメータのチューニングをしっかり行うべきである．例えば，文字抽出時のしきい値や RandomForest のパラメータ，入力画像の分割画素数等．こういったパラメータを同時にチューニングするには計算資源を要するため，未だ実現していない．

余談だが，学習データを増やす方法については，テストデータとして入力されたデータを学習データとして再利用することを考えている．つまり，何らかしらの手段でユーザが文字認識を行ったときに，認識結果に対して正しいか間違っているかをたずね，可能であれば誤認していた場合に答えを教えさせるようにしたい．

認識の前段階としての補正

先程の話に繋がる部分が多分にあるが，文字認識を行う前に入力画像に対して補正をかける必要がある．例えば，現在のアルゴリズムでは日本語の文字は基本的に正方形状であることを仮定しているが，手書き文字では勿論人によって縦細かったり上下方向に歪んでいる等の文字の歪みがある．文字に対して大まかな外接四角形をつくり，正方形状に変換することで文字の歪みを補正することは認識精度の向上に役立つはずである．

また，特徴抽出した濃淡ベクトルのコントラストを大きくすることも有効だと考えられる．

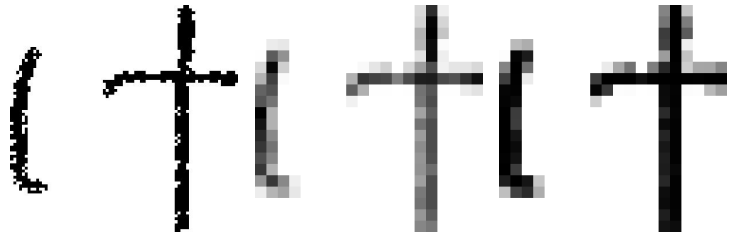


図 3.4: 入力画像を変換してコントラストを調整した様子．(a) が入力画像，(b) は濃淡平均を単純にとった画像，(c) は濃淡平均をとり $y = x^4$ でコントラストをスケーリングした画像．

認識結果の精査

認識結果についても結果をそのまま返すのではなく，吟味する必要がある．例えば,「単言吾」という認識結果が返ってくれば，人間ならこれはどう見ても「単語」の誤認であるとわかる．これと同じことを機械にも行わせると万が一誤認した場合でも修正をきかせることができる．つまり，認識結果に対して辞書整合性を見るのである．(勿論認識段階で誤認をなくせればそれがベストである)

オンライン認識の導入

OCR の分野では認識方法の違いでオフライン認識とオンライン認識という二種類がある．オフライン認識とは入力画像のピクセルデータのみを情報源として文字認識をすることであり，オンライン認識とは筆順等のストローク情報も情報源として認識することである．一般にオンライン認識の方が情報量が多いため，認識精度が上がると言われている．今回の開発動機としてはタッチデバイスをフル活用したいということがあったので，タッチデバイスであればストローク情報も容易に取得できるため，オンライン認識の導入をすべきだと考えている．

参考 URL

[1] gihub リポジトリ https://github.com/levelfour/shrift
[2] 機械学習入門（拙著，執筆中） https://github.com/levelfour/machine-learning-2014/wiki
[3] ラボユース Hackathon での発表資料 (SlideShare) http://www.slideshare.net/levelfour/shrift
[4] U-22 プログラミング・コンテスト入選作品 http://www.u22procon.com/sakuhin.html
[5] ハンズラボ関連記事 https://www.hands-lab.com/contents/?p=893

編集後記

★ TSG へお越しくださってありがとうございます.
★ おそらく，これが自分が編集する最後の部報になると思います.
★ 今年は記事は少し少ないですが，楽しんでいただけたら幸いです.
★ TSG にお越しいただいた方に, 今一度礼を重ねて失礼します。

理論科学グループ　部報 第 309 号

2014 年 11 月 22 日　発行
発行者　内藏理史
編集者　村井翔悟
発行所　理論科学グループ
〒 153–0041　東京都目黒区駒場 3–8–1
東京大学教養学部内学生会館 313B
Telephone: 03–5454–4343

理論科学グループ部報 第 309 号
— 駒場祭パンフレット号 —
2014 年 11 月 22 日

THEORETICAL SCIENCE GROUP