［ディープラーニング活用事例と使いこなしの勘所］ 最適化・推論分野

6 AIによるソースコードのレビュー

―ディープラーニングでコードの美しさを診断する―

森崎雅稔 | 富士通アプリケーションズ（株）

AIとの出会い

AIにソースレビューをやらせるという我々の発想の起点は，2012年にGoogleが行った猫の画像認識に遡る．大量の画像をニューラルネットワークで学習し，猫の画像を1つのグループにまとめてほかの画像と分類することに成功したというものだった．つまり「猫」というグループを，ラベル付けしていないデータからの学習（自己教示学習）によって作り上げたということだ．この衝撃的な研究成果にディープラーニングによる画像認識の可能性を強く意識し，仕事でディープラーニングを使うことに憧れの気持ちを持った．

この憧れが現実の仕事に転換したきっかけはGoogleのTensorFlowの登場（2015年）と，ハッカソン／アイデアソンなどのイノベーション発掘のブームである．新ビジネス発掘のイベントから，「ソースコード⇒画像化⇒ディープラーニング⇒AIソースレビュー⇒プロダクト品質改善に役立てる」というアイディアが生まれ，TensorFlowを使って概念実証する活動に発展した．

ディープラーニングへのアプローチ

ディープラーニングで何を認識させるのかは，アイディア発掘の時点で決まっていた．ディープラーニングに学習させるのはソースコードの「美しさ」である．

美しく書かれたソースコードは，本来の実装者以外が後に修正を担当してもミスが起きにくいといえる．つまり，美しく書かれていないソースコードの潜在リスクは，後の保守作業や仕様変更の際に起きる品質面での不具合や修正コストの増大である．これは長年プログラマを務めて得た自らの経験則である．

また，近年はアジャイル開発の浸透もあり，開発現場は常に実装と改修とレビューの連続である．美しく読みやすいソースコードを書くことは，開発の生産性向上や品質安定化には欠かせない要因となっている．継続的なレビューで美しいソースコードを維持することは重要であり，レビューの生産性向上にAIを応用したソースレビューシステムが有益であると考えた．

このレビューシステムの学習手段には，教師あり学習を選択した．これはGoogleの猫認識（教師なし学習）とは異なる方針である．その理由は，どのようなソースコードが美しくて，どのようなソースコードが美しくないかは，自らがはっきりとした指針を持っていたからである．ソースコードを学習対象とする際に工夫したことは，学習用に画像化する方式である．前述のGoogleの研究では，画像を200ピクセル×200ピクセルに揃えて集めたそうだ．我々のソースコード認識もそれに倣い，120ピクセル×40ピクセルの画像に納めることを思いついた．コンピュータの画面に表示したソースコードの一般的な特徴は，横幅は大きくても120桁程度，縦の長さは数十行から数千行と，縦方向に細長い形状をしている．これを機械的に40行ずつに分割して，横120桁×縦40行のカードを作成し，さらに各文字に文字コードに紐づけた色を割り振った．そして，1文字を着色した1ピクセルに変換することで，120桁×40行のカードを，120ピクセル×40ピクセルの画像にマッピングした（図-1）．画像に使用した色はおよそ100色である．

さっそく社内に蓄積しているJavaのソースコード資産から教師データの抽出を始めた．ディープラーニングの成功の可否に，教師データの出来栄えが重要なこと

は自明である．そのため教師データには，業務アプリケーション，フレームワーク，および開発ツールなどさまざまな特徴ある開発資産から幅広くデータを集めた．また，開発経験の浅い社員のコードから，ベテラン社員，上級プログラマのコード，さらには構成管理サーバに残っていた古い版数のコード，新入社員が研修中に書いたコードまで集めた．また，学習結果の検証用には，前述の多様なソースコードに加え，GitHub 上のオープンソースや，過去の成功プロジェクト，およびトラブルプロジェクトのソースコードも集めた．

これらのソースコードを用いたディープラーニングと，学習済みニューラルネットワークによるパターン認識のイメージを図 -2 に示す．レビュー時のチェック項目である，読みやすく実装されているかの○×を判断する学習である．そのため美しいコードを「良いコード」とラベル付けし，読みづらいコード，複雑すぎるコード，コメント不足なコードなど，レビューアの作業負荷を高めてしまう，美しくないコードを「悪いコード」とラベル付けして学習させた．

この取り組み用に集めた教師データの数は画像の枚数換算で，「良いコード」が 5,296 枚，「悪いコード」が 5,100 枚である．意図的に「良い」と「悪い」がほぼ同数になるように教師データを抽出した．40 行ずつカード化したソースコードに対して「良い」「悪い」の格付けをするには，さまざまな検討，工夫をした．格付け手法の詳細は本稿では割愛する．格付け作業用には 3 人の上級プログラマを擁し，均質に格付けできるよう勉強会を行い，観点のぶれが出ないよう約 4 週間で一気に教師データを作成した．約 1 万件の教師データ作成に，投入した時間は延べ 250 時間である．3 人が毎日 4 時間程度教師データ作成に没頭した．

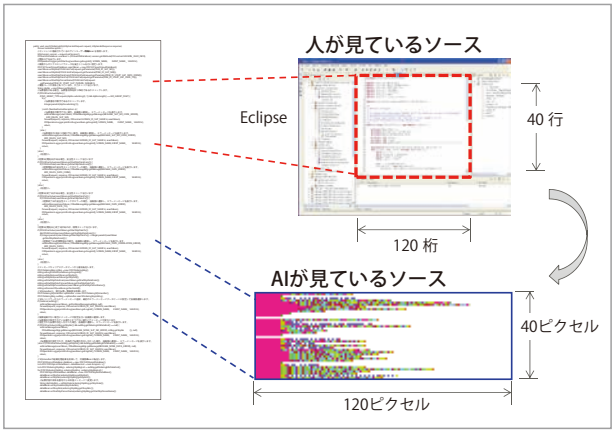
ニューラルネットワークの設計と学習

私たちがディープラーニングの実践を開始したのは 2016 年 6 月である．このときに使用したツールは 2015 年 11 月に公開されたばかりの Google の TensorFlow である．最初は TensorFlow を 64bit OS の Ubuntu で動作させた．その後，NVIDIA の GPU を搭載したマシンを活用した．

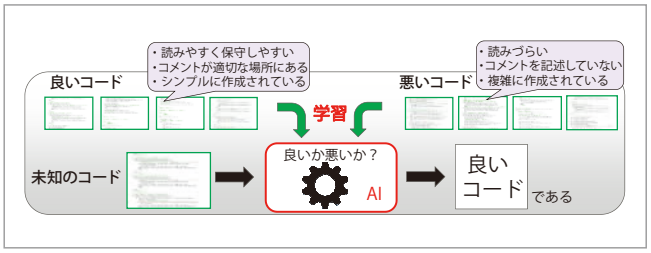
ニューラルネットワークの設計スタイルには画像認識の領域で多用される，畳込みニューラルネットワーク（以後 CNN）を採用した．ニューラルネットワークの各層の設計は，手書き文字認識用に公開されていた構造を参考にした．最初は 6 層のシンプルな CNN だった．その後，さまざまな実験と改修を経て，現在は 8 層の CNN を使用している．

初めてのディープラーニングのため，ニューラルネットワークの設計知識はまったくなかったが，インターネットから入手した事例に基づいたパラメータ設定でも，学習がちゃんと収束することを確認できた．

初めて構築した学習済みニューラルネットワークの認識力の測定結果は，良いコードと悪いコードに正しく分類できたものが 79％あり，21％が誤認識であった．この検証では，学習に使用していない教師データを評価用に使用した．この識別能力から，ソースレビュー現



■図 -1　ソースコードの画像化



■図 -2　ソースコードの美しさの学習と認識

場で，潜在的な品質の診断に活用できる技術として期待が湧きあがった．その後，教師データの見直し（格付けの再調整），学習パラメータの修正，ニューラルネットワークの設計変更をしつつ認識精度の改善を行った．具体的には半年間で 8 回の設計変更と 47 回の再学習（パラメータの変更を伴う）を経て，認識率（正解率）を 88%まで向上させることに成功した．現在はこの精度の実用性について，実プロジェクトでの実践において検証中である．なお，学習の所要時間は，CPU 版で数時間程度，GPU 版は 1 時間以内だった．

結果の見せ方の工夫

学習済みのニューラルネットワークがソースコードの美しさを見分けることができ，それがソースコードの可読性や保守性の「良い」「悪い」といった潜在的な品質を客観的に評価する指標として使えることが分かった．これを開発現場の実務に活かすためには，診断結果を見やすくする必要がある．そのために，膨大なソースコード資産の，可読性や保守性を一目で見渡せることを目標とした．最初に作成したのは，図 -3 の赤青マップである．

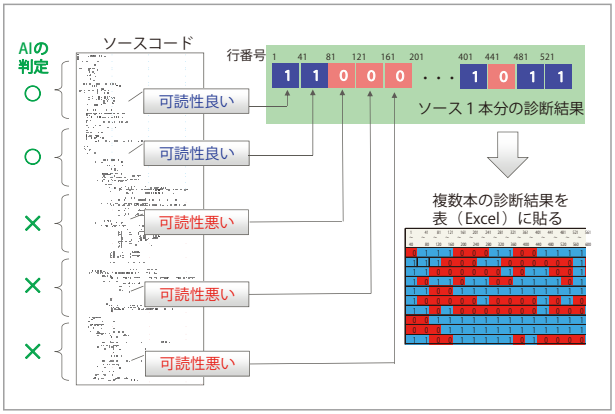
赤青マップは，1 本のソースコードを 40 行単位に分割し，40 行 1 単位の範囲内で，可読性，保守性の「良い」と「悪い」を，青と赤で表している．複数のソースコードの診断結果を表（Excel）に張り付けると，ソースコードの潜在的な品質を俯瞰して確認するためのマップとして利用できる．表上の 1 行はソースコード 1 本分の情報であり，それを列方向に辿ると，ソースコードの何行目あたりに可読性，保守性の悪い個所があるのかを赤いセルの存在によって知ることができる．

可読性，保守性に関するコーディング規約を遵守し，レビューや改修のしやすさに配慮したソースコード資産は，マップ上は青いセルが多くを占めるように見える．逆に赤いセルが多数存在している個所は，読みづらいソースコード，改修しづらいソースコードの存在を示している．赤青マップで可読性，保守性が悪いと指摘されたソースコードは，それがアジャイル開発中の資産なら，各スプリントの生産性や品質を劣化させる要因として認識できる．あるいはウォーターフォール開発の実装工程の成果物なら，後続のテスト工程での品質問題の増大や，将来の仕様変更や障害修正時の生産性の悪化や，改修時に新たな障害を生み出すリスクの予測として認識できる．

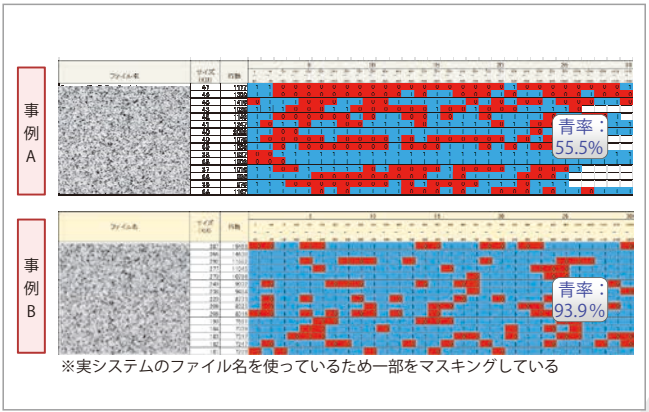
図 -4 は実プロジェクトに AI ソースレビューを適用した際の赤青マップの具体例である．各々，作成した赤青マップから 15 件分だけを抜粋した．

AI ソースレビューによる診断結果は，実際にはそのままの形では使用せず，赤青マップとソースコードの両方を見たレビューアからの専門家所見という形で引き出される．

事例 A は，資産総数 590 本，合計 13 万行の Java コー



■図 -3　赤青マップの作成



■図 -4　赤青マップ診断事例

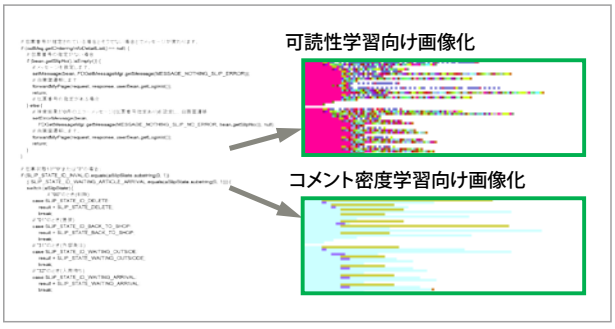
ドで，全体の 55.5%が青（良い）と診断された．専門家所見は，『可読性・保守性にムラがある．プログラマのスキルのバラつきに対して十分な品質統制が取れていない可能性がある．赤が集中しているソースコードを重点的にレビューし，原因分析が必要』となった．

事例 B は，資産総数 1 万 2 千本，合計 600 万行の Java コードで，全体の 93.9%が青（良い）と診断された．専門家所見は『可読性・保守性が均質であり，全体的に読みやすいソースコードが揃っている．コーディング規約の順守，レビューの徹底などの品質統制が行き届いている』となった．このような所見のメッセージの自動合成が現在の課題である．

なお，AI ソースレビューシステムの実行速度は，画像化とニューラルネットワークでの認識を合わせて 1,000 行あたり 40 秒ほどであった（Intel Core i5 搭載のノート PC で測定）．現在の速度ではリアルタイムな診断は難しい．変更の起きたソースコードだけを夜間バッチで診断して，翌朝フィードバックを得るという運用が標準である．

複数のニューラルネットワークでの診断

現在は前述の専門家所見を自動合成する取り組みに挑戦中である．その方法は，異なる教師データを与えた複数のニューラルネットワークによる診断結果から，



■図 -5　レビュー観点をコメント部に絞る画像化

あらかじめ用意したメッセージを組み合わせる方式である．たとえば，図 -5 のようにコメントの記述を強調した新しい画像化による学習を行い，コメント記述の場所や分量の「良い」「悪い」をディープラーニングで評価できるようにしている．

具体的には，ニューラルネットワークが学習するレビュー観点をコメント部に絞るために，画像化で使用する色の種類を削減した．たとえば，図 -5 の例では，Java 予約語の if, while, for, case 等，コメントを付記することが好ましい語彙は「紫色」，その他の予約語，関数名，変数名，空白，演算子，括弧は「水色」，コメントは「黄色」で画像化している．

コメントが適切な位置に適量で存在していることを「良い」とし，コメントがない，足りない，あるいは位置が不適切な状態を「悪い」として教師データを再格付けし，新たなニューラルネットワークを構築した．

最初の取り組みにおける画像化でも，コメント量の不備を「悪い」と検出できていたが，この新しい画像化ではコメント量の不備のみに顕著に反応する．何をもって「悪い」と診断されたのかをメッセージで示せるようになった．

このほかにもいくつかのレビュー観点ごとの画像化を工夫して，複数ニューラルネットワークのアンサンブルによる診断機能の高度化に努めている．美しいソースコードの特徴の強調が教師データの加工の工夫で実現できる．人間のレビューアの感性に，より近づけた学習と認識を目指している．

（2018 年 7 月 31 日受付）

■森崎雅稔（正会員）　morisaki.masato@jp.fujitsu.com

1986 年富士通入社．第 2 次 AI ブームの中で Lisp プログラマとしてエキスパートシステムの開発に従事．AI 冬の時代には流通業向けの基幹業務パッケージの SE として活動．2016 年より AI に返り咲き，AI 人材育成にも取り組んでいる．