人工知能 課題番号 23「ニューラルネットの応用例」

工学部電子情報工学科 03-175001 浅井明里

2017 年 12 月 20 日

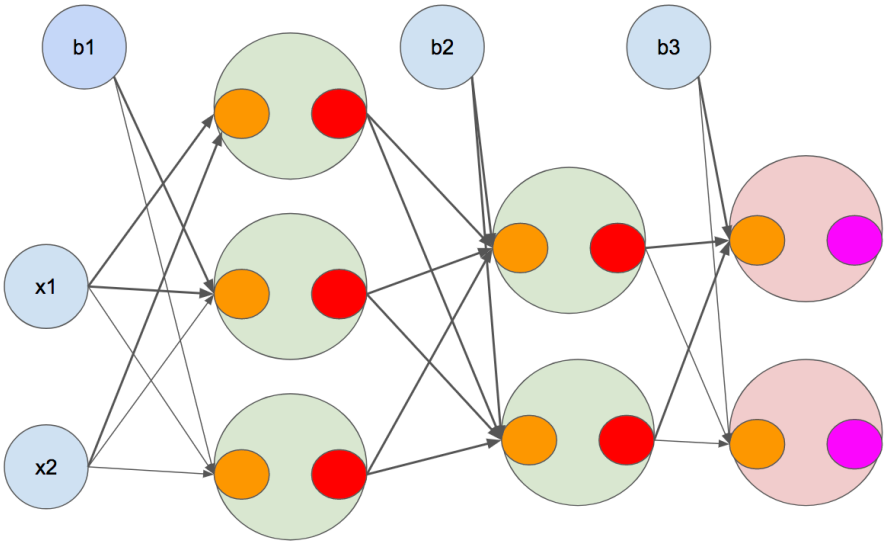

図 1　二層ニューラルネットワークの構造

1　ニューラルネットを用いて Speed Dating 問題を解く

本課題では、Python の標準ライブラリ及び Numpy を用いて二層のニューラルネットワークモデルを実装し、このモデルを用いて Speed Dating 問題を解いた。データセットについては講義サイトで公開されている前処理済みの full_matching.csv を用い、この 70% を学習データセットとして学習し、残りのテストデータセットでマッチングが成立するかどうかを二値分類して予測した。実装したニューラルネットを用いた予測の正答率は 83% 程度であり、また特徴選択を行ない、正解ラベルと相関の大きい上位 20 項目のみを用いた場合の予測正答率は 85% 程度を達成した。

2　ニューラルネットワークの実装

2.1　実装したモデル

今回は元々の Speed Dating データセットの数及び入力の次元がそこまで多くないため、二層のシンプルなニューラルネットワークを実装した。ネットワークは以下の図 1 のような構造となっている。まず一層目では、それぞれのニューロンに対して、重み付き和を計算する（図の 1 段目の緑色のニューロンの中の橙色の部分)。

$$A^{(1)} = XW^{(1)} + B^{(1)}$$

次に、活性化関数にこの重み付き和を入力し、出力を次の層へ渡す（図の赤色の部分)。実験ではこの活性化関数について、Sigmoid 及び ReLU で検証を行なった。

$$Z^{(1)} = Activation(A^{(1)})$$

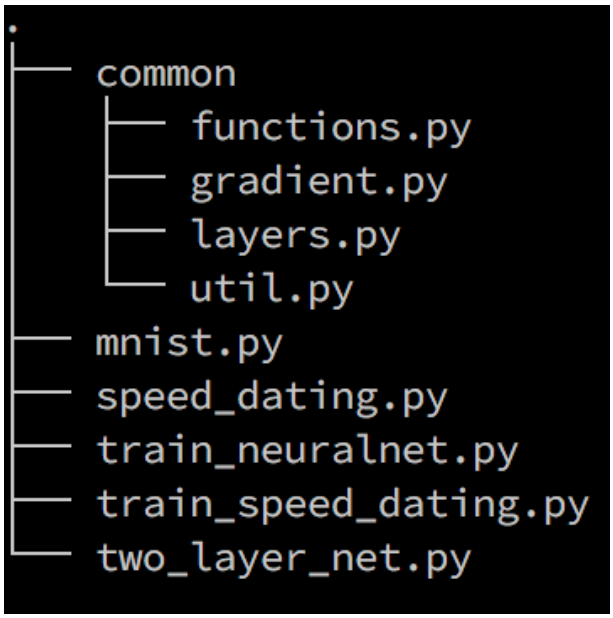

図 2　ディレクトリ構成

同様のことをもう一層繰り返す。3 段目の桃色のニューロンは出力層となっており、まず一つ前の層の出力を入力として重み付き和を計算する。

$$A^{(3)} = Z^{(2)}W^{(2)} + B^{(2)}$$

次にこの $A^{(3)}$ についてソフトマックス関数を適用する。k 番目の出力 y_k は以下の式で求めることができる。

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

ソフトマックス関数は 0 から 1.0 の実数になり、また出力の総和は 1 になる。つまりこの出力はそれぞれの出力ラベルがどこまで確からしいかという確率として解釈できる。このソフトマックス関数の損失関数として交差エントロピー誤差を用い、この損失関数の勾配を求め、パラメータを勾配方向に微笑寮だけ更新することを繰り返して学習を行なった。また勾配計算に当たって数値微分ではなく誤差逆伝播法 (Back Propagation) を用いた。

2.2　実装の詳細

図 2 はディレクトリ構成を示している。speed_dating.py においてデータの読み込みや正解ラベルのワンホットベクトル化などの処理を行いデータセットを作成し、train_speed_dating.py で two_neural_net.py で定義された二層ニューラルネットワーククラスのインスタンスを生成し、任意のエポック数だけ学習を行う。common 以下の各ファイルでは誤差伝搬法で用いるレイヤクラスの定義、勾配計算及や活性化関数の定義などを行なっている。なお、多層ニューラルネットワークの実装、誤差伝播法の実装については Stanford 大学の「CS231n: Convolutional Neural Networks for Visual Recognition Spring 2017」講義資料等を参考にした。

実装に誤りがないか確認するため、一般に多層ニューラルネットモデルであれば 95% 以上の精度を出すことができるとされている mnist データセットに対する実験を行なった結果、最終的に 97% を超える test

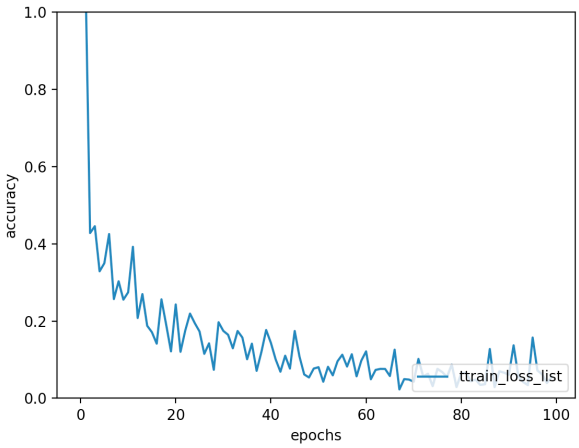

図 3　mnist データセット学習時の損失の推移

sccuracy を達成した。図 3 は mnist データセットを用いた学習で loss がエポック数に応じてどう変化したかを可視化したグラフであり、学習が進むにつれて損失が小さくなっていることが確認できる。

2.3　プログラムの実行

プログラムを実行する際には speed_dating ディレクトリ以下で以下のコマンドを入力すれば実行できる。学習中は train 及び test accuracy, loss が epoch 数の経過とともに表示され、また学習終了時に loss の推移の様子などを可視化したグラフが表示される。また speed_dating については speed_dating.py のデータへのパスが私のマシンでの絶対パスとなっているため、相対パスなどへの変更が必要である。

```
$ python3 train_speed_dating.py
$ python3 train_mnist.py
```

3　full_matching_data における特徴選択

full_matching_data はそれぞれの男女ペアについて、マッチしたかどうかを表す match を除くと 174 項目の情報が与えられている。ニューラルネットワークモデルを用いて予測を行うのにあたってこれらの入力を全て使うのではなく、実際にラベルとなんらかの相関のある項目のみを入力変数とすることで精度が上がるかどうかについても検証を行なった。実際の特徴選択の過程及び sckit-learn を用いたロジスティック回帰によるナイーブな予測を Jupiter notebook 上で行なった。結果は ipython notebook を起動することで確認できる。

まず、全ての match を含む全ての変数について、ヒートマップでどういった相関関係があるか、可視化を行なった。このヒートマップでは項目が多すぎてどの項目が match と相関関係にあるのか目視では確認できないが、変数間にも正もしくは負の相関関係があるもの (正の相関関係は赤で、負の相関関係は青で表現される) と全く相関のないもの（灰色で表現される）があり、無駄な入力情報をいくつも保持するより、ある程度相関のある変数を利用した方がより精度が高くなるのではないかという仮説が立てられる。

match 変数と全ての変数で相関係数を求め、相関係数の大きさに基づいてソートすると、以下の図 3 で表すような変数が相関関係が大きいことがわかる。実際に match を除き最も相関が高いとされた male_fun3_1 について、マッチングした人を赤、しなかった人を青としてヒストグラムを描画すると、male_fun3_1 の値が多

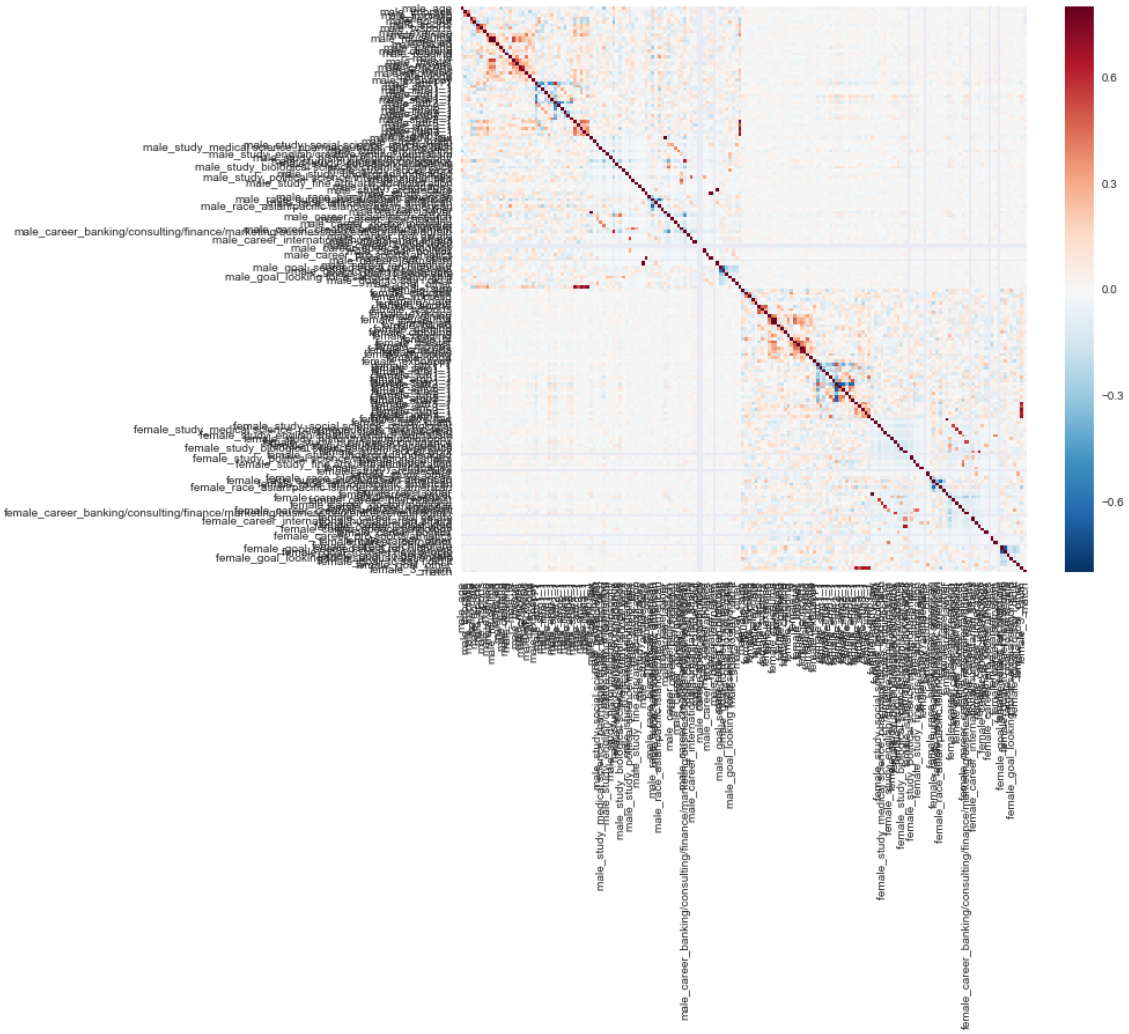

図 4　全ての項目間でのヒートマップ

```
full_matching_df.corr().match.sort_values(ascending=False)[:10]
```

```
match                                                    1.000000
male_fun3_1                                              0.086226
female_study_medical science, pharmaceuticals, and bio tech    0.079165
female_clubbing                                          0.066871
male_3_1sum                                              0.065327
female_art                                               0.063774
female_fun1_1                                            0.056416
male_clubbing                                            0.052261
female_concerts                                          0.051516
female_museums                                           0.050616
Name: match, dtype: float64
```

図 5　match 変数と相関の大きい変数の一覧

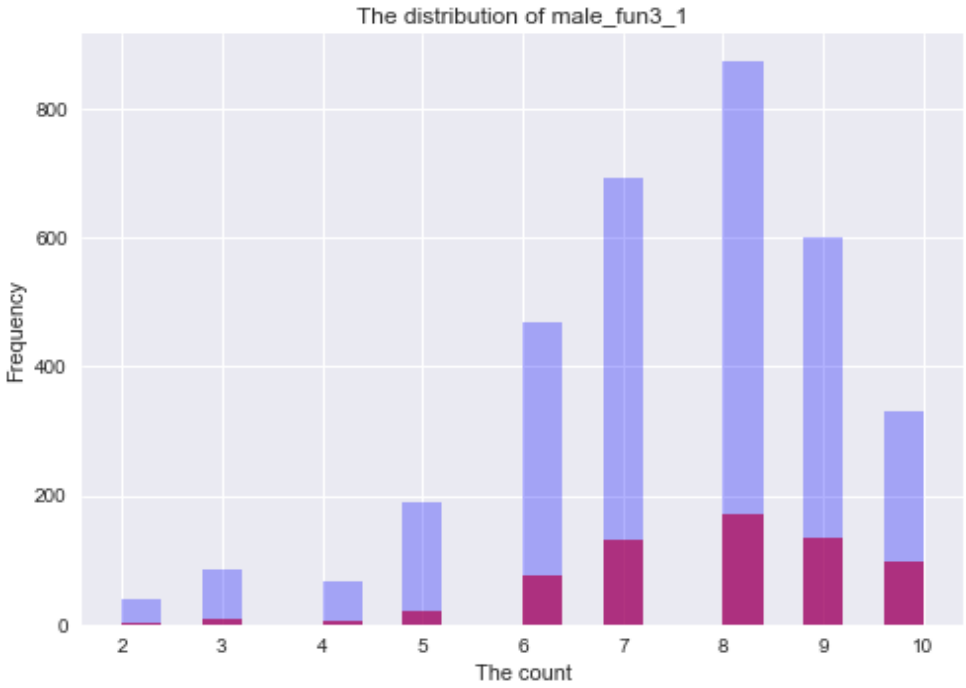

図 6　male_fun3_1 のヒストグラム

いほど、マッチングしたユーザーをマッチングしていないユーザーと比較した時の相対的な割合が大きくなっていることがわかる。

以上のような分析の結果、match との相関係数の絶対値の大きい 18 の変数を選択し、それ以外の変数を入力情報より排除した上で match ラベルの予測する実験も行なった。

4　予測結果

4.1　学習率、活性化関数を変更し結果を比較する

まず、二層ニューラルネットワークモデルの学習率、活性化関数を変更し、こういったハイパーパラメータの変更が結果にどのような影響を与えうるのか観察した。

活性化関数の変更

活性化関数を ReLU にした場合及び Sigmoid にした場合で、loss の変化の様子、accuracy に変化が現れるか観測した。またエポック数は共に 10000 である。

Sigmoid

$$h(x) = \frac{1}{1 + \exp(-x)}$$

ReLU

$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x\ leq 0) \end{cases}$$

結果は次の通りになる。表 1 より、ReLU の方がテストデータに対する accuracy が高く、また loss

表 1　活性化関数ごとの accuracy 及び loss の比較

活性化関数	training accuracy	test accuracy	loss
Sigmoid	0.841	0.826	0.604
ReLU	0.835	0.842	0.424

についても 10000 エポック終了時により小さくなっていることが確認できる。また次の二つの図は Sigmoid 及び ReLU での loss の推移を観測したものであり、Sigmoid についてはエポックが進んでも loss が上昇するなど、値がうまく収束いていないことがわかる。

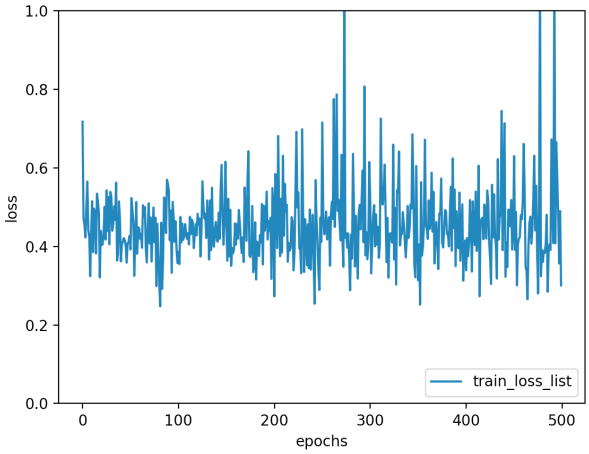

図 7　Sigmoid を活性化関数に用いた時の loss の推移

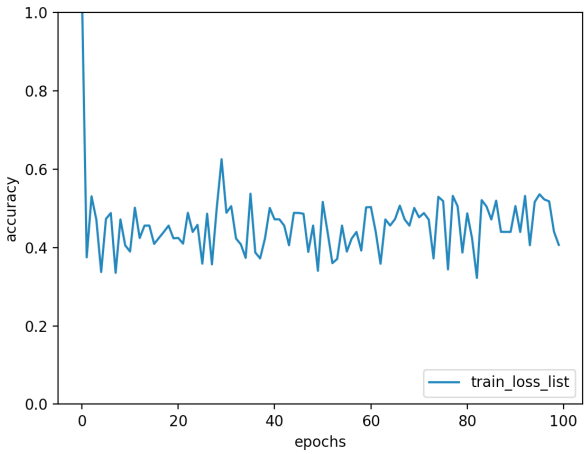

図 8　ReLU を活性化関数に用いた時の loss の推移

学習率の変更

学習率を 0.1, 0.5, 1.0 と変化した時の最終エポック時の accuracy 及び loss の値は次のようになった。

表 2　学習率を変化させた時の accuracy 及び loss の比較

学習率	training accuracy	test accuracy	loss
0.1	0.838	0.833	0.439
0.5	0.834	0.842	0.460
1.0	0.834	0.843	0.387

表 2 の結果を見ると、学習率については少なくとも今回のタスクにおいてはあまり大きく結果に影響を与えているわけではないように見受けられる。

4.2 特徴選択の有無

特徴選択を有りにした（すなわち入力変数の数を 20 以下に制限した）時と全ての特徴を入力変数として用いた時のパフォーマンス及び計算時間の比較を行なった。表 3 のように、特徴選択は精度の向上にある程度貢

表 3　学習率を変化させた時の accuracy 及び loss の比較

特徴選択	training accuracy	test accuracy	loss	計算時間 [sec]
有り	0.831	0.851	0.487	19.76
無し	0.834	0.843	0.460	44.51

献しているように思われる。また特筆すべきは計算時間で有り、特徴選択を行なった場合、50000 エポックにかかる時間が特徴選択を行わない場合と比較し半分以下となっている。精度の面でも上回るもしくは差異がない程度であれば、ある程度使用する特徴を絞ってしまった方が特に計算時間になんらかの制約条件がある場合は良いのではないかと感じた。