物理学とディープラーニング (ゼミ)

B4 柳瀬調知

2023 年 5 月 18 日

5.2 波動関数の変分計算

前節で回帰問題についてのニューラルネットの表現能力をチェックした。次に、1 次元井戸型ポテンシャルの基底状態をニューラルネットによる数値計算で求める。

・問題設定

3.1.3 節の例題にあるような、1 次元無限井戸型ポテンシャル (井戸の幅 $L = 1$ とする) の基底状態における波動関数 $\psi(x)$ とエネルギー ϵ を求める。ただし、境界条件 $\psi(0) = \psi(1) = 0$ を課す。シュレディンガー方程式から解析的に求めると、

$$\psi(x) = \sqrt{2}\sin \pi x \tag{1}$$

$$\epsilon = \frac{\hbar^2 \pi^2}{2m} \tag{2}$$

となる。いま、簡単のため $\frac{\hbar^2}{2m} = 1$ とすると、$\epsilon = \pi^2$ である。

・ニューラルネットで変分計算

$$\epsilon[\psi] = \frac{\langle\psi|\,\hat{H}\,|\psi\rangle}{\langle\psi|\psi\rangle} = \frac{1}{\pi^2}\frac{\int_0^1 |\psi'(x)|^2}{\int_0^1 |\psi(x)|^2} \tag{3}$$

シュレディンガー方程式を解くほかに、変分法という基底状態の求め方がある。これは上の式で表せる、汎関数を最小とするような $\psi(x)$ を決める方法である。ここで、解析解から $\epsilon = \pi^2$ であるため、真の基底エネルギーが 1 になるように規格化している。なお、ここからの話は波動関数が実数にとれると仮定する。

ここから、規格化条件と境界条件は考慮しつつ、エネルギーを最小とするように波動関数の形を最適化することを考える。前節と同じくニューラルネットは設計するが、訓練データがないので教師あり学習とはいえない。エネルギーを最小とする強化学習のようなものといえる。

なぜ強化学習か：損失関数は誤差の大きさを見積もるためのものだから、普通は推定量と教師データの差 $y-\hat{y}$ の関数になっているはず。エネルギー汎関数はそうなっていない。

表 1　今回の計算を他の計算と比較

	5.1 節の回帰	波動関数の変分計算
未知関数	$f(x) = (10x^2 - 6x^3 + x^4)e^{-x}$ (5-1) 式	$\psi(x) = \sqrt{2}\sin\pi x$ (3-90) 式
モデル (NN なし)	$f_{pol}(x) = \sum_{n=0}^{39} c_n x^n$ (5-2) 式	$\phi_\alpha(x) = Cx(1-x)^\alpha$ (3-91 式)
モデル (NN あり)	中間層 (LeakyReLU) が 2 層と出力層 (linear)	中間層 (sigmoid) が 1 層と出力層 (linear)
損失関数	平均二乗誤差	なし (エネルギー汎関数)

ニューラルネットワークなしの変分計算は p65 の例題にあって、この節でやるのはニューラルネットワークを使った変分計算である。

・プログラムの説明

Listing 1　データやライブラリのロード

```
# 波動関数の変分計算のための関数定義
import numpy as np
from numpy import pi, sin, sqrt

# 離散化する数を指定
N = 3000
x = np.linspace(0, 1, N)
dummy = np.sqrt(2) * np.sin(np.pi*x) # 仮に正解を入れておく
```

np.linspace(0,1,N) は [0,1] の範囲を N 分割した Numpy 配列を返す。各要素は、離散化された x 座標の値を表している。これにユニバーサル演算で基底状態の波動関数 (p66,(3-90) 式) にかませたのが、dummy で x と同じく、N 個の成分をもつ。ただし、dummy はあくまで教師あり学習のアーキテクチャを使うためなので、なんでもよい。

例: $x = [0., 0.1, \ldots, 0.9, 1.]$ に対し、$y = \sqrt{2}\sin\pi x$ とすると、$y = [0., \ldots, -1.4]$

Listing 2　損失関数 (エネルギー汎関数) の定義

```
# 関数を対称化して端をゼロに合わせる
def psi(y):
  y_rev = K.reverse(y, 0)
  y_symmetrized = y + y_rev - y[0] - y[-1]
  return y_symmetrized

# 関数の微分を計算
def dpsi(y):
  y_shifted_f = tf.roll(y, shift=-1, axis=0)
  y_shifted_b = tf.roll(y, shift=+1, axis=0)
  dy = (y_shifted_f - y_shifted_b)/2
```

```
  return dy

# 最小化したいエネルギーを損失関数とする
def variationalE(y_true, y_pred):
  wave = psi(y_pred)
  wave_nom = K.l2_normalize(wave, 0)
  dwave = dpsi(wave_nom)
  return N**2 * K.sum(K.square(dwave)) / pi**2
```

variationalE を定義するために必要なので、任意の配列 y が与えられたときに (5-5) 式の境界条件を満たすように対象化する関数 phi(y)、y の微分を計算する関数 dphi(y) を定義している。ここでは、$\psi(x) = f(x)+f(1-x)-f(0)-f(1)$(ただし、$f(x)$ は任意) という対象化を行うことで、境界条件を自動的に満たすようにしてある。それから、微分の分母にある Δx が抜けている気がするが、variaionalE の最後の N^2 のところでつじつまが合っている。なお、プログラム中の K は tensorflow.keras からインポートした backend モジュール。

Listing 3　ニューラルネットワークの設計

```
# 波動関数の変分計算を機械学習で実装
model = Sequential()
model.add(Dense(2, input_dim=1, activation='sigmoid'))
model.add(Dense(1, activation='linear'))
```

ここでは、1 次元から 1 次元への写像を作る。隠れ層は 1 層として、そのユニット数は 2 とする。初めに、keras の Sequetial というクラスのインスタンスを作成している。活性化関数はシグモイド関数と出力層は線形ユニット (恒等写像) としている。(波動関数が単純な形のため、少ないニューロン数、隠れ層でも十分学習できる)

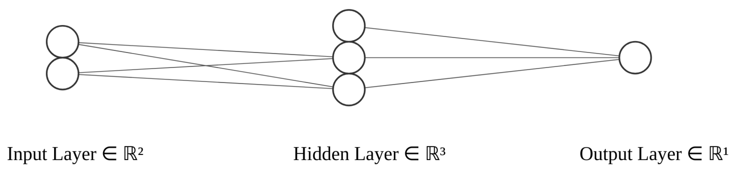

図 1　ニューラルネットの概形

Listing 4　ニューラルネットネットワークの学習

```
model.compile(loss=variationalE, optimizer='Adam')
# model.summary()
results = model.fit(x, dummy, epochs=300, steps_per_epoch =1, verbose=0, shuffle=False)

pred = model.predict(x)
```

model.compile で学習の細かい設定を行っている。損失関数は、先に定義した variationalE(エネルギー汎関数) で、勾配の更新方法が Adam となっている。今回は検証データにあたるものがない。学習は 300 エポックを行い、一度に使うデータが 1 なので、オンライン学習である。

Listing 5　結果の評価

```
# 結果をプロット
pred = model.predict(x)
func = psi(pred)
func = np.abs(func) / np.sqrt(np.sum(func**2)/N) ←波動関数の規格化
plt.xlim(0,1) ←横軸の範囲設定
plt.plot(x, func, label='fitted')
plt.plot(x, dummy, label='answer')
plt.legend()
plt.xlabel('$x$')
plt.ylabel(r'$\psi(x)$')
plt.show()
```

この出力結果が図 5.6(p188) である。左図のようにエネルギーは 1 に漸近していくから、規格化などがうまく機能しているとわかる。右図より、たしかに見分けつかないほど解析解と一致している。

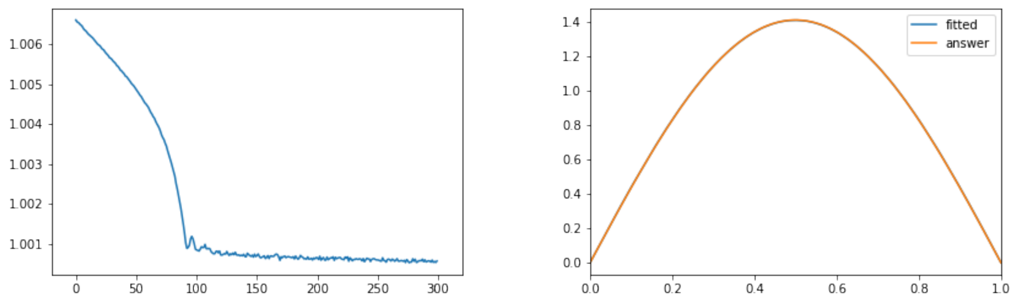

図 2　自分で実行した結果

・パラメータについて (補足)

第 l 層の j 番目のニューロンと第 $l-1$ 層の i 番目のニューロンの重みを $w_{ij}^{(l)}$ とする。$\sigma(x) = \frac{1}{1+e^{-x}}$ とする。

$$z_1^{(1)} = \sigma(w_{11}^{(1)} x + b_1^{(1)}) \tag{4}$$

$$z_2^{(1)} = \sigma(w_{12}^{(1)} x + b_2^{(1)}) \tag{5}$$

$$y = f(w_{11}^{(2)} z_1^{(1)} + w_{21}^{(2)} z_2^{(1)} + b_1^{(2)}) \tag{6}$$

f は恒等写像より、

$$y = w_{11}^{(2)} \sigma(w_{11}^{(1)} x + b_1^{(1)}) + w_{21}^{(2)} \sigma(w_{12}^{(1)} x + b_2^{(1)}) + b_1^{(2)} \tag{7}$$

パラメータは 7 個ある。学習後の各パラメータの値は、

```
[array([[0.9557099, 1.152671 ]], dtype=float32), array([ 0.24239989, -0.33635545],
    dtype=float32), array([[ 0.1871245 ],[-0.39245972]], dtype=float32), array
    ([-0.02212993], dtype=float32)]
```

となる。この値で (7) 式の関数をプロットすると、次のようになる。

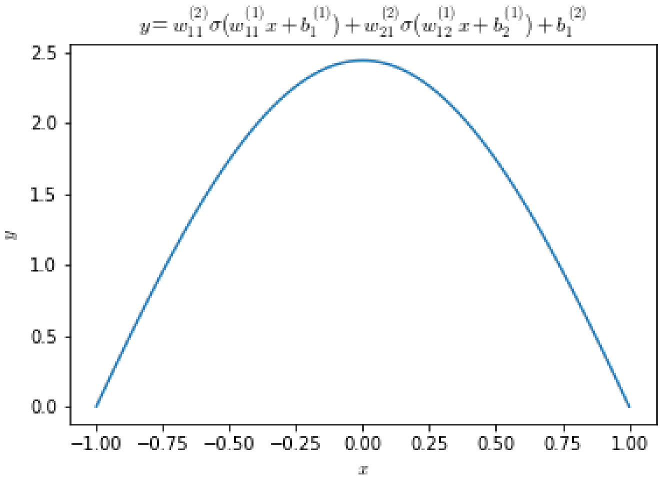

図 3　(7) 式のプロット

内部で (7) の計算をしていることが確かめられた。

```
params=model.get_weights()
print(params)
# print(params[0][0][1])

x=np.linspace(-1,1,1000)
z_1=sigmoid(params[0][0][0]*x+params[1][0])
z_2=sigmoid(params[0][0][1]*x+params[1][1])
y=params[2][0]*z_1+params[2][1]*z_2+params[3][0]
y=psi(y)
y=np.abs(y)/np.sqrt(np.sum(y**2)/N)
plt.plot(x,y)
plt.title("$y=w_{11}^{(2)} \sigma{(w_{11}^{(1)} x+b_{1}^{(1)})}+w_{21}^{(2)} \sigma{(w_
    {12}^{(1)} x+b_{2}^{(1)})}+b_{1}^{(2)}$",math_fontfamily='cm')
plt.xlabel("$x$",math_fontfamily='cm')
plt.ylabel("$y$",math_fontfamily='cm')
# plt.grid()
plt.savefig("check.png")
plt.show()
```

・参考文献

[1] 富谷昭夫「これならわかる機械学習入門」

[2] ゼロから作る DeepLearning