深層学習 day1

Section1: 入力層～中間層

- 入力層
  - 入力となるデータを数字の並びで表現したもの
  - 動物種類の分類の例では、体長、体重、ひげの本数、毛の平均長などが入力データとなる
- 中間層
  - 入力層のデータに重みを掛けた後、バイアスを加えたもの

確認テスト1

$u = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$　をpythonで書け。

答え： `u1 = np.dot(x, W1) + b1`

確認テスト2

`1_1_forward_propagation.ipynb` から中間層の出力を定義しているソースを抜き出せ。

答え： `z = functions.relu(u)`

実装演習結果

- ファイル： `my_1_1_forward_propagation.ipynb`
- 実装演習結果の抜粋
  - 順伝播（単層・複数ユニット）課題の試してみよう_配列の初期化の実行結果

```
*** 重み ***
[[0.78448836 0.5519249  0.05299646]
 [0.28541838 0.0874703  0.19128375]
 [0.75817506 0.07897251 0.16248218]
 [0.80977524 0.13379177 0.74521453]]

*** バイアス ***
[0.1 0.2 0.3]

*** 入力 ***
[ 1.  5.  2. -1.]

*** 総入力 ***
[3.01815512 1.21342967 0.88916503]

*** 中間層出力 ***
[0.95338761 0.77090523 0.70871783]
```

Section2: 活性化関数

- ニューラルネットワークにおいて次の層への出力の大きさを決める非線形の関数

中間層用の活性化関数

- ステップ関数
  - 最近はあまり使われない。
  - 0以上なら1, 0より小さいなら0を出力。0か1かしか表現できない。
- シグモイド（ロジスティック）関数
  - 0~1の間を緩やかに変化
  - 1/(1+e^(-x))
  - 大きな値では出力の変化が微小なため勾配消失問題を引き起こすことがあった
- ReLU
  - これが今最も使われている。
  - x > 0の時はxを、xが0以下の時は0を出力
  - 勾配消失問題の回避とスパース化に貢献。
  -

出力層用の活性化関数

- ソフトマックス関数
  - 多クラス分類に使用される
- 恒等写像

- シグモイド関数

確認テスト

- 中間層z=f(u)に相当するソースを抜き出せ

答え： `z = functions.sigmoid(u)`

実装演習結果

- ファイル： `my_1_1_forward_propagation.ipynb`
  - 多クラス分類のノードの構成を2-3-4から3-5-4に変更

```
##### ネットワークの初期化 #####
*** 重み1 ***
[[0.39584932 0.34003881 0.95012385 0.83734055 0.62053755]
 [0.67485755 0.12538868 0.59715302 0.04805491 0.04911069]
 [0.46974373 0.30237175 0.72931983 0.88260397 0.17032201]]

*** 重み2 ***
[[0.74174043 0.99298396 0.93496933 0.39263016]
 [0.83137794 0.95624954 0.68359426 0.22060711]
 [0.79603207 0.70925198 0.49614996 0.75663965]
 [0.36314909 0.67968242 0.36471733 0.56427986]
 [0.30406935 0.10250141 0.54503293 0.73483335]]

*** バイアス1 ***
[0.1 0.2 0.3 0.4 0.5]

*** バイアス2 ***
[0.1 0.2 0.3 0.4]

##### 順伝播開始 #####
*** 総入力1 ***
[3.25479561 1.69793142 4.63238936 3.98126229 1.72972495]

*** 中間層出力1 ***
[3.25479561 1.69793142 4.63238936 3.98126229 1.72972495]

*** 総入力2 ***
[ 9.58511486 11.2244305   9.19698244  9.07516187]

*** 出力1 ***
[0.13458064 0.6933114  0.09128908 0.08081888]

出力合計: 1.0

##### 結果表示 #####
*** 出力 ***
[0.13458064 0.6933114  0.09128908 0.08081888]

*** 訓練データ ***
[0 0 0 1]

*** 誤差 ***
2.515543432929862
```

Section3: 出力層

- 中間層からの出力を出力層の活性化関数に掛けて、最終的な出力yを出す

- 出力yと教師データdの誤差を誤差関数にて求め、誤差を最小にするよう中間層のパラメータを調整する

確認テスト（二乗誤差）

二乗誤差の式：

$$E_n(w) = 1/2\Sigma(y_i - d_i)^2$$

- Q1: 誤差関数で引き算ではなく2乗しているのは何故か？
  - 答え：それぞれのラベルでの誤差を正の値にするため
- Q2: 式の1/2はどういう意味を持つか？
  - 答え：誤差関数の微分を行う際の計算式を簡単にするため。本質的な意味はない。

出力層の種類と活性化関数

	回帰	二値分類	多クラス分類
活性化関数	恒等写像f(u)=u	シグモイド関数	ソフトマックス関数
誤差関数	二乗誤差	交差エントロピー	

ソフトマックス関数： $f(i,u) = e^x/\Sigma e^x$

交差エントロピー： $E_n(w) = -\Sigma d_i log y_i$

確認テスト（交差エントロピー）

交差エントロピー： $E_n(w) = -\Sigma d_i log y_i$
の式において(1)左辺、(2)右辺に該当するソースコードの行を示せ。

- (1)左辺： `def cross_entropy_error(d, y)`
- (2)右辺: `-np.sum(np.log(y[np.arrange(batch_size), d] + 1e-7)) / batch_size`
  - log(x)は0に近づくと値が-∞となるため、ごく小さい値として1e-7を足している

Section4: 勾配降下法

- 誤差を最小化するパラメータwを見つける
- 学習率εを用いる
  - 学習率が大きすぎると発散してしまう
  - 学習率が小さすぎると収束までに時間がかかる

$$w^{(t+1)} = w^{(t)} - \varepsilon \Delta E$$

勾配降下法のアルゴリズム

- Momentum
- AdaGrad
- AdaDelta
- Adam
    - 最近ではこれがよく使われる

確率的勾配降下法(SGD)

$$w^{(t+1)} = w^{(t)} - \varepsilon \Delta E_n$$

- 勾配降下法では全サンプルの平均誤差を取るのに対し、ランダムに抽出したサンプルの誤差を使用
- メリット
    - データが冗長な場合の計算コスト軽減
    - 望まない局所極小解に収束するリスクの軽減
    - オンライン学習ができる

確認テスト

- オンライン学習とは何か？
    - 答え：学習データが入ってくる度に都度パラメータを更新し、学習を進めていく手法。
    - これに対し、バッチ学習では全ての学習データを使ってパラメータ更新を行う。
    - 最近ではオンライン学習が主流。バッチ学習だとメモリに載らないこともある。

ミニバッチ勾配降下法

- ランダムに分割したデータの集合（ミニバッチ）Dtに属するサンプルの平均誤差
  例えば10万枚の画像を小分けにして学習する際、
  500枚ずつ2000バッチ、500枚の誤差をEとすると
  全体の誤差は2000*Eとできる。これを1/2000することで平均誤差が求まる。
- メリット
    - SGDのメリットを損なわず、CPUを利用したスレッド並列化、GPUでのSIMD並列化が可能。

Section5: 誤差逆伝播法

- 誤差を出力層側から順に微分し、前の層へと伝播
- 微分の連鎖律を使用し、不要な再帰的演算を避けて微分を算出する
- 順伝播（forward）に対し、逆伝播（backward）という

確認テスト

(1) delta2.funcitons.d_mean_squared_error(d, y)

(2) delta1 = np.dot(delta2, w2.T) * functions.d_sigmoid(z1)

(3) grad['w2'] = np.dot(x.T, delta1)