ニューラルネットの学習

前田利之（まえだ　としゆき）
aipr@e-chan.jp

阪南大学　経営情報学部

(2024.11.08)

データから学習する

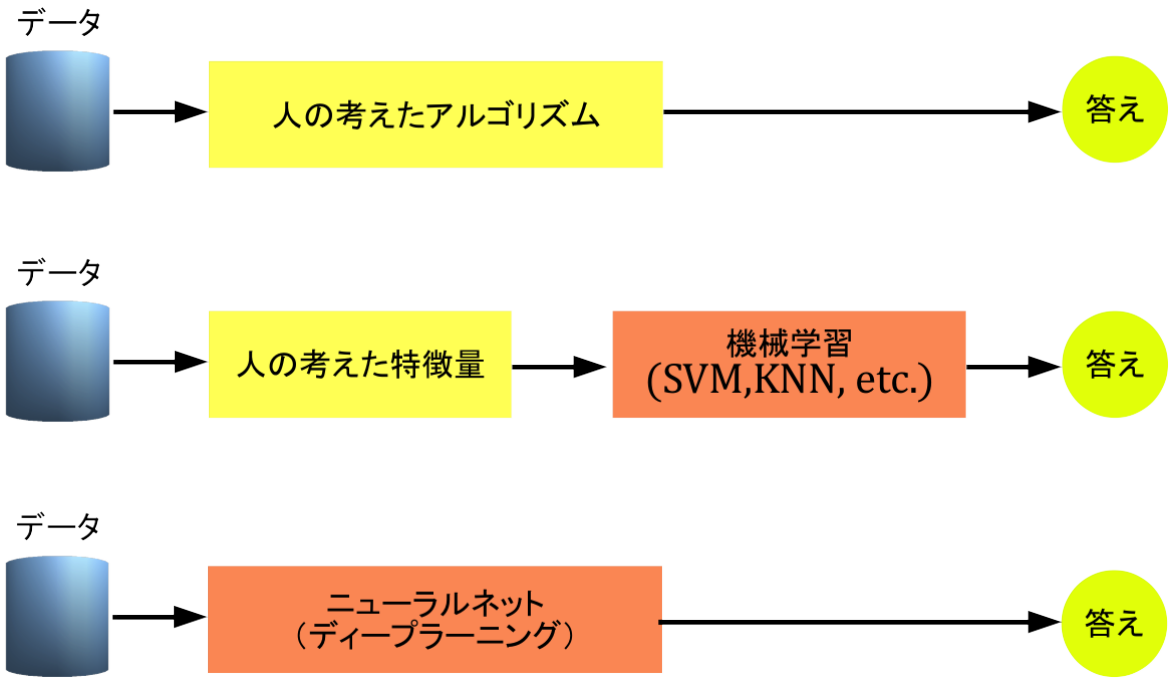
- ニューラルネットの最も重要な特徴の１つ：データから学習できる→ 
- つまり、重みパラメータの値をデータから自動設定できる
  - そうじゃなければ手作業で決めるのは大変
  - 本格的なディープラーニングだと億を軽く越える…事実上無理

データから学習する

- ニューラルネットの最も重要な特徴の１つ：データから学習できる→ **データ駆動**
- つまり、重みパラメータの値をデータから自動設定できる
  - そうじゃなければ手作業で決めるのは大変
  - 本格的なディープラーニングだと億を軽く越える…事実上無理

データ駆動

- 人中心アプローチからの脱却（パラダイムシフト）



訓練データとテストデータ

- 機械学習では普通、データを訓練データとテストデータに分け、学習と評価をおこなう
- もし分けないと、 [　　] がつかない
  - [　　]: 未知の（訓練データに含まれない）データにも対応できる能力
  - データセット１つで学習と評価してしまうと、そのデータセットには対応できても、他のデータセットには対応できない可能性がある
  - そのような、特定のデータセットに過度に対応した状況を [　　] といい、避けなくてはならない

訓練データとテストデータ

- 機械学習では普通、データを訓練データとテストデータに分け、学習と評価をおこなう
- もし分けないと、 **汎化能力** がつかない
  - [　　]: 未知の（訓練データに含まれない）データにも対応できる能力
  - データセット１つで学習と評価してしまうと、そのデータセットには対応できても、他のデータセットには対応できない可能性がある
  - そのような、特定のデータセットに過度に対応した状況を [　　] といい、避けなくてはならない

訓練データとテストデータ

- 機械学習では普通、データを訓練データとテストデータに分け、学習と評価をおこなう
- もし分けないと、 **汎化能力** がつかない
  - **汎化能力**: 未知の（訓練データに含まれない）データにも対応できる能力
  - データセット１つで学習と評価してしまうと、そのデータセットには対応できても、他のデータセットには対応できない可能性がある
  - そのような、特定のデータセットに過度に対応した状況を [　　] といい、避けなくてはならない

訓練データとテストデータ

- 機械学習では普通、データを訓練データとテストデータに分け、学習と評価をおこなう
- もし分けないと、 **汎化能力** がつかない
  - **汎化能力**: 未知の（訓練データに含まれない）データにも対応できる能力
  - データセット１つで学習と評価してしまうと、そのデータセットには対応できても、他のデータセットには対応できない可能性がある
  - そのような、特定のデータセットに過度に対応した状況を **過学習** といい、避けなくてはならない

学習アルゴリズムの実装

1. [　　]: 訓練データの中からランダムに一部のデータ [　　] を選ぶ。
2. [　　]: [　　] の [　　] を減らすために、各重みパラメータの [　　] を求める。[　　] は [　　] の値を最も減らす方向を示す。
3. [　　]: 重みパラメータを [　　] 方向に微小量だけ [　　] する
4. 上を繰りかえす

学習アルゴリズムの実装

1. ミニバッチ: 訓練データの中からランダムに一部のデータ [　　] を選ぶ。
2. [　　]: [　　] の [　　] を減らすために、各重みパラメータの [　　] を求める。[　　] は [　　] の値を最も減らす方向を示す。
3. [　　]: 重みパラメータを [　　] 方向に微小量だけ [　　] する
4. 上を繰りかえす

学習アルゴリズムの実装

1. ミニバッチ: 訓練データの中からランダムに一部のデータ（ミニバッチ）を選ぶ。
2. [　　]: [　　]の[　　]を減らすために、各重みパラメータの[　　]を求める。[　　]は[　　]の値を最も減らす方向を示す。
3. [　　]: 重みパラメータを[　　]方向に微小量だけ[　　]する
4. 上を繰りかえす

学習アルゴリズムの実装

1. **ミニバッチ**: 訓練データの中からランダムに一部のデータ**（ミニバッチ）**を選ぶ。
2. **勾配の算出**: [　　] の [　　] を減らすために、各重みパラメータの [　　] を求める。[　　] は [　　] の値を最も減らす方向を示す。
3. [　　]: 重みパラメータを [　　] 方向に微小量だけ [　　] する
4. 上を繰りかえす

学習アルゴリズムの実装

1. **ミニバッチ**: 訓練データの中からランダムに一部のデータ**（ミニバッチ）**を選ぶ。
2. **勾配の算出**: **ミニバッチ**の［　　］を減らすために、各重みパラメータの［　　］を求める。［　　］は［　　］の値を最も減らす方向を示す。
3. ［　　］: 重みパラメータを［　　］方向に微小量だけ［　　］する
4. 上を繰りかえす

学習アルゴリズムの実装

1. **ミニバッチ**: 訓練データの中からランダムに一部のデータ **（ミニバッチ）** を選ぶ。
2. **勾配の算出**: **ミニバッチ** の **損失関数** を減らすために、各重みパラメータの [　　] を求める。[　　] は [　　] の値を最も減らす方向を示す。
3. [　　]: 重みパラメータを [　　] 方向に微小量だけ [　　] する
4. 上を繰りかえす

学習アルゴリズムの実装

1. **ミニバッチ**: 訓練データの中からランダムに一部のデータ **（ミニバッチ）** を選ぶ。
2. **勾配の算出**: **ミニバッチ** の **損失関数** を減らすために、各重みパラメータの **勾配** を求める。［　　］は［　　］の値を最も減らす方向を示す。
3. ［　　］: 重みパラメータを［　　］方向に微小量だけ［　　］する
4. 上を繰りかえす

学習アルゴリズムの実装

1. **ミニバッチ**: 訓練データの中からランダムに一部のデータ **（ミニバッチ）** を選ぶ。
2. **勾配の算出**: **ミニバッチ** の **損失関数** を減らすために、各重みパラメータの **勾配** を求める。 **勾配** は [　　] の値を最も減らす方向を示す。
3. [　　]: 重みパラメータを [　　] 方向に微小量だけ [　　] する
4. 上を繰りかえす

学習アルゴリズムの実装

1. **ミニバッチ**: 訓練データの中からランダムに一部のデータ **（ミニバッチ）** を選ぶ。
2. **勾配の算出**: **ミニバッチ** の **損失関数** を減らすために、各重みパラメータの **勾配** を求める。 **勾配** は **損失関数** の値を最も減らす方向を示す。
3. [　　]: 重みパラメータを [　　] 方向に微小量だけ [　　] する
4. 上を繰りかえす

学習アルゴリズムの実装

1. **ミニバッチ**: 訓練データの中からランダムに一部のデータ **（ミニバッチ）** を選ぶ。
2. **勾配の算出**: **ミニバッチ** の **損失関数** を減らすために、各重みパラメータの **勾配** を求める。 **勾配** は **損失関数** の値を最も減らす方向を示す。
3. **パラメータの更新**: 重みパラメータを [ ] 方向に微小量だけ [ ] する
4. 上を繰りかえす

学習アルゴリズムの実装

1. **ミニバッチ**: 訓練データの中からランダムに一部のデータ **（ミニバッチ）** を選ぶ。
2. **勾配の算出**: **ミニバッチ** の **損失関数** を減らすために、各重みパラメータの **勾配** を求める。 **勾配** は **損失関数** の値を最も減らす方向を示す。
3. **パラメータの更新**: 重みパラメータを **勾配** 方向に微小量だけ [　　] する
4. 上を繰りかえす

学習アルゴリズムの実装

1. **ミニバッチ**: 訓練データの中からランダムに一部のデータ**（ミニバッチ）**を選ぶ。
2. **勾配の算出**: **ミニバッチ**の**損失関数**を減らすために、各重みパラメータの**勾配**を求める。**勾配**は**損失関数**の値を最も減らす方向を示す。
3. **パラメータの更新**: 重みパラメータを**勾配**方向に微小量だけ**更新**する
4. 上を繰りかえす

損失関数

- 学習過程では、ある指標= [ ] によって現在の状態をあらわし、それを基準として最適な重みパラメータの探索をおこなう
- 任意の関数が使えるが、よく使われるのは [ ] や [ ]

損失関数

- 学習過程では、ある指標= **損失関数** によって現在の状態をあらわし、それを基準として最適な重みパラメータの探索をおこなう
- 任意の関数が使えるが、よく使われるのは [　　] や [　　]

損失関数

- 学習過程では、ある指標= **損失関数** によって現在の状態をあらわし、それを基準として最適な重みパラメータの探索をおこなう
- 任意の関数が使えるが、よく使われるのは **２乗和誤差** や

損失関数

- 学習過程では、ある指標= **損失関数** によって現在の状態をあらわし、それを基準として最適な重みパラメータの探索をおこなう
- 任意の関数が使えるが、よく使われるのは **２乗和誤差** や **交差エントロピー誤差**

2 乗和誤差

- $E = \frac{1}{2}\sum_k (y_k - t_k)^2$

  y_k はニューラルネットの出力、t_k は教師データ
- 実装例は以下のようになる

```
def sum_squared_error(y,t):
    return 0.5 * np.sum((y-t)**2)
```

- 教師データは、正解のインデックスだけ１、他は０という表記になる ⇒ ワンホット (one-hot) 表現

2乗和誤差（実習）

☆ 以下を Colab 上で実行する

```
# coding: utf-8
import numpy as np
def sum_squared_error(y,t):
  return 0.5 * np.sum((y-t)**2)
t = np.array([0,0,1,0,0])
y1 = np.array([0.1,0.05,0.6,0.0,0.1])
print(sum_squared_error(y1,t))
y2 = np.array([0.1,0.05,0.1,0.0,0.1])
print(sum_squared_error(y2,t))
```

◎結果は 0.09125000000000003 と 0.41625 になる
→ y1 のほうが誤差が小さい=正解に近い

交差エントロピー誤差

- $E = -\sum_k t_k \log y_k$

  log は底が e の自然対数…対数とはとりあえず log（対数）の復習 を参照のこと
- 実装例は以下のようになる

```
def cross_enthropy_error(y,t):
    delta = 1e-7
    return -np.sum(t*np.log(y+delta))
```

- delta は微小な数 (python での 1e-7 は $1 \times 10^{-7} = 0.0000001$) で、y に 0 が入っても大丈夫なようにしている

ミニバッチ学習

- 訓練データで学習するとは、訓練データにたいする [　　] を求め、その値を [　　] するパラメータを探す、ということなので損失関数は [　　] が対象
- なので、全訓練データの損失関数の和は、データ N 個の交差エントロピー誤差の場合、以下になる [　　]

$$E = -\frac{1}{N}\sum_{n}\sum_{k} t_{nk} \log y_{nk}$$

ミニバッチ学習

- 訓練データで学習するとは、訓練データにたいする **損失関数** を求め、その値を [　　] するパラメータを探す、ということなので損失関数は [　　] が対象
- なので、全訓練データの損失関数の和は、データ N 個の交差エントロピー誤差の場合、以下になる [　　]

$$E = -\frac{1}{N}\sum_{n}\sum_{k} t_{nk} \log y_{nk}$$

ミニバッチ学習

- 訓練データで学習するとは、訓練データにたいする **損失関数** を求め、その値を **小さく** するパラメータを探す、ということなので損失関数は [　　] が対象
- なので、全訓練データの損失関数の和は、データ N 個の交差エントロピー誤差の場合、以下になる [　　]

$$E = -\frac{1}{N}\sum_{n}\sum_{k} t_{nk} \log y_{nk}$$

ミニバッチ学習

- 訓練データで学習するとは、訓練データにたいする **損失関数** を求め、その値を **小さく** するパラメータを探す、ということなので損失関数は **全訓練データ** が対象
- なので、全訓練データの損失関数の和は、データ N 個の交差エントロピー誤差の場合、以下になる

$$E = -\frac{1}{N}\sum_{n}\sum_{k} t_{nk} \log y_{nk}$$

ミニバッチ学習

- 訓練データで学習するとは、訓練データにたいする **損失関数** を求め、その値を **小さく** するパラメータを探す、ということなので損失関数は **全訓練データ** が対象
- なので、全訓練データの損失関数の和は、データ N 個の交差エントロピー誤差の場合、以下になる **（平均する）**

$$E = -\frac{1}{N}\sum_{n}\sum_{k} t_{nk} \log y_{nk}$$

ミニバッチ学習

- MNIST データセットは 60000 個あったので、全てのデータの損失関数の和を求めるには時間がかかる
- なので、 を選び出して、それを全体の近似として利用する
- →

ミニバッチ学習

- MNIST データセットは 60000 個あったので、全てのデータの損失関数の和を求めるには時間がかかる
- なので、一部のデータを選び出して、それを全体の近似として利用する
- →

ミニバッチ学習

- MNIST データセットは 60000 個あったので、全てのデータの損失関数の和を求めるには時間がかかる
- なので、一部のデータを選び出して、それを全体の近似として利用する
- → ミニバッチ学習

バッチ対応交差エントロピー誤差（実装）

- データが１つの場合とバッチの場合と両方に対応

```
def cross_enthropy_error(y,t):
  if y.ndim == 1:
    t = t.reshape(1,t.size)
    y = y.reshape(1,y.size)
  batch_size = y.shape(0)
  return -np.sum(t*np.log(y+1e-7))
```

なぜ損失関数を設定するのか

- 認識問題は、[　　]が高くなるようなパラメータを獲得したい
- 何故わざわざ損失関数を導入する？(二度手間？)
- →最適なパラメータを求めるためには損失関数の値が小さくなるようにパラメータを変更したい→パラメータの[　　]を計算し更新する、という手続きを踏むから
- 認識精度の場合は[　　]なので更新の手掛かりにならないから

なぜ損失関数を設定するのか

- 認識問題は、**認識精度**が高くなるようなパラメータを獲得したい
- 何故わざわざ損失関数を導入する？（二度手間？）
- →最適なパラメータを求めるためには損失関数の値が小さくなるようにパラメータを変更したい→パラメータの　　を計算し更新する、という手続きを踏むから
- 認識精度の場合は　　なので更新の手掛かりにならないから

なぜ損失関数を設定するのか

- 認識問題は、**認識精度**が高くなるようなパラメータを獲得したい
- 何故わざわざ損失関数を導入する？（二度手間？）
- →最適なパラメータを求めるためには損失関数の値が小さくなるようにパラメータを変更したい→パラメータの**微分（勾配）**を計算し更新する、という手続きを踏むから
- 認識精度の場合は［　　］なので更新の手掛かりにならないから

なぜ損失関数を設定するのか

- 認識問題は、**認識精度**が高くなるようなパラメータを獲得したい
- 何故わざわざ損失関数を導入する？（二度手間？）
- →最適なパラメータを求めるためには損失関数の値が小さくなるようにパラメータを変更したい→パラメータの**微分（勾配）**を計算し更新する、という手続きを踏むから
- 認識精度の場合は**微分できない（ほとんど０）**なので更新の手掛かりにならないから

微分

- そもそも微分とは何かについては、は、 先端技術論の資料（１）と、 先端技術論の資料（２） を参照のこと
- ここでは数値的に微分を求めることを考える （数値的じゃないのは [　　] という
- （上のリンクは [　　] のみの説明）

微分

- そもそも微分とは何かについては、は、 先端技術論の資料（１）と、 先端技術論の資料（２） を参照のこと
- ここでは数値的に微分を求めることを考える （数値的じゃないのは 「解析的」 という
- （上のリンクは のみの説明）

微分

- そもそも微分とは何かについては、は、 先端技術論の資料（１）と、 先端技術論の資料（２） を参照のこと
- ここでは数値的に微分を求めることを考える （数値的じゃないのは 「解析的」 という
- （上のリンクは 解析的微分 のみの説明）

数値微分

- $$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$
- 直観的には
- 数値解を求める場合 h は 0 にはできないし、極端に小さな場合はアンダーフローを起こしてしまう（コンピュータで計算できなくなる）ので、適当な小さな値 (10^{-4} くらい）を用いる。

数値微分

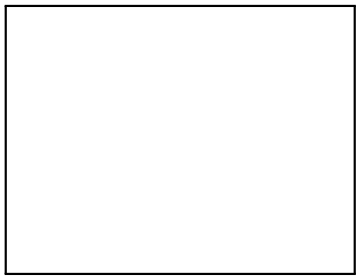
- $$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$
- 直観的には **グラフの接線の傾き**
- 数値解を求める場合 h は 0 にはできないし、極端に小さな場合はアンダーフローを起こしてしまう（コンピュータで計算できなくなる）ので、適当な小さな値 (10^{-4} くらい）を用いる。

数値微分

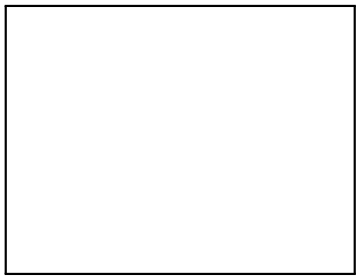
- また、$f(x+h)-f(x)$ だと、真の接線から結構ずれることがある
- そこで、$f(x+h)-f(x)$ ではなく $f(x+h)-f(x-h)$ として、それを $2h$ で割ると、ずれを押えられる（ずれは0にはならない）

```
def numerical_diff(f,x):
  h = 1e-4 # 0.0001
  return (f(x+h)-f(x-h))/(2*h)
```

偏微分

- $f(x, y) = x^2 + y^2$ のような、２つ以上変数がある関数を微分する場合を考える
- これを、ある１つの変数で微分する場合、どの変数に対しての微分かを区別する必要がある
- これが  ...  のように書く
  - ちなみに ∂ はいろんな読み方がある が、個人的には「でる」に１票 (?)
  - さらにちなみに 全微分というのもある（詳細略）

偏微分

- $f(x, y) = x^2 + y^2$ のような、２つ以上変数がある関数を微分する場合を考える
- これを、ある１つの変数で微分する場合、どの変数に対しての微分かを区別する必要がある
- これが 偏微分...  のように書く
  - ちなみに ∂ はいろんな読み方がある が、個人的には「でる」に１票 (?)
  - さらにちなみに 全微分というのもある（詳細略）

偏微分

- $f(x, y) = x^2 + y^2$ のような、２つ以上変数がある関数を微分する場合を考える
- これを、ある１つの変数で微分する場合、どの変数に対しての微分かを区別する必要がある
- これが **偏微分**... $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}$ のように書く
  - ちなみに ∂ はいろんな読み方がある が、個人的には「でる」に１票 (?)
  - さらにちなみに 全微分というのもある（詳細略）

偏微分

- その場合、微分しないほうの変数は □ として扱う（固定化）
- たとえば関数 $f(x, y) = x^2 + y^2$ の $(x, y) = (3, 4)$ での偏微分は
  - $\dfrac{\partial f}{\partial x} = 2x = 6$ (y^2 は定数なので無くなる)
  - $\dfrac{\partial f}{\partial y} = 2y = 8$ (x^2 は定数なので無くなる)

偏微分

- その場合、微分しないほうの変数は **定数** として扱う（固定化）
- たとえば関数 $f(x, y) = x^2 + y^2$ の $(x, y) = (3, 4)$ での偏微分は
  - $\dfrac{\partial f}{\partial x} = 2x = 6$ (y^2 は定数なので無くなる)
  - $\dfrac{\partial f}{\partial y} = 2y = 8$ (x^2 は定数なので無くなる)

偏微分（実習）

☆ 以下を Colab 上で実行する

```
def num_diff(f,x):
  h = 1e-4 # 0.0001
  return (f(x+h)-f(x-h))/(2*h)
def func_tmpx(x):
  return x*x + 4.0**2
def func_tmpy(y):
  return 3.0**2 + y*y
print(num_diff(func_tmpx,3))
print(num_diff(func_tmpy,4))
```

◎結果は 6.00000000000378 と
7.999999999999119 となり、誤差の範囲となる

勾配

- すべての変数の [　　] を [　　] として表したもの
- 例えば前述の $f(x, y) = x^2 + y^2$ の場合、この関数の勾配は以下の通り：

[　　]

勾配

- すべての変数の **偏微分** を

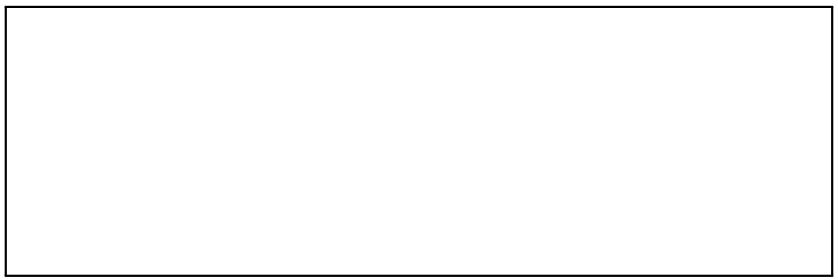
  [　　]として表したもの

- 例えば前述の $f(x, y) = x^2 + y^2$ の場合、この関数の勾配は以下の通り：

  [　　]

勾配

- すべての変数の **偏微分** を **ベクトル（行列）** として表したもの
- 例えば前述の $f(x, y) = x^2 + y^2$ の場合、この関数の勾配は以下の通り：



勾配

- すべての変数の **偏微分** を **ベクトル（行列）** として表したもの
- 例えば前述の $f(x, y) = x^2 + y^2$ の場合、この関数の勾配は以下の通り：

$$\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) = (2x, 2y)$$

勾配（実装）

☆ 参考：勾配の実装例

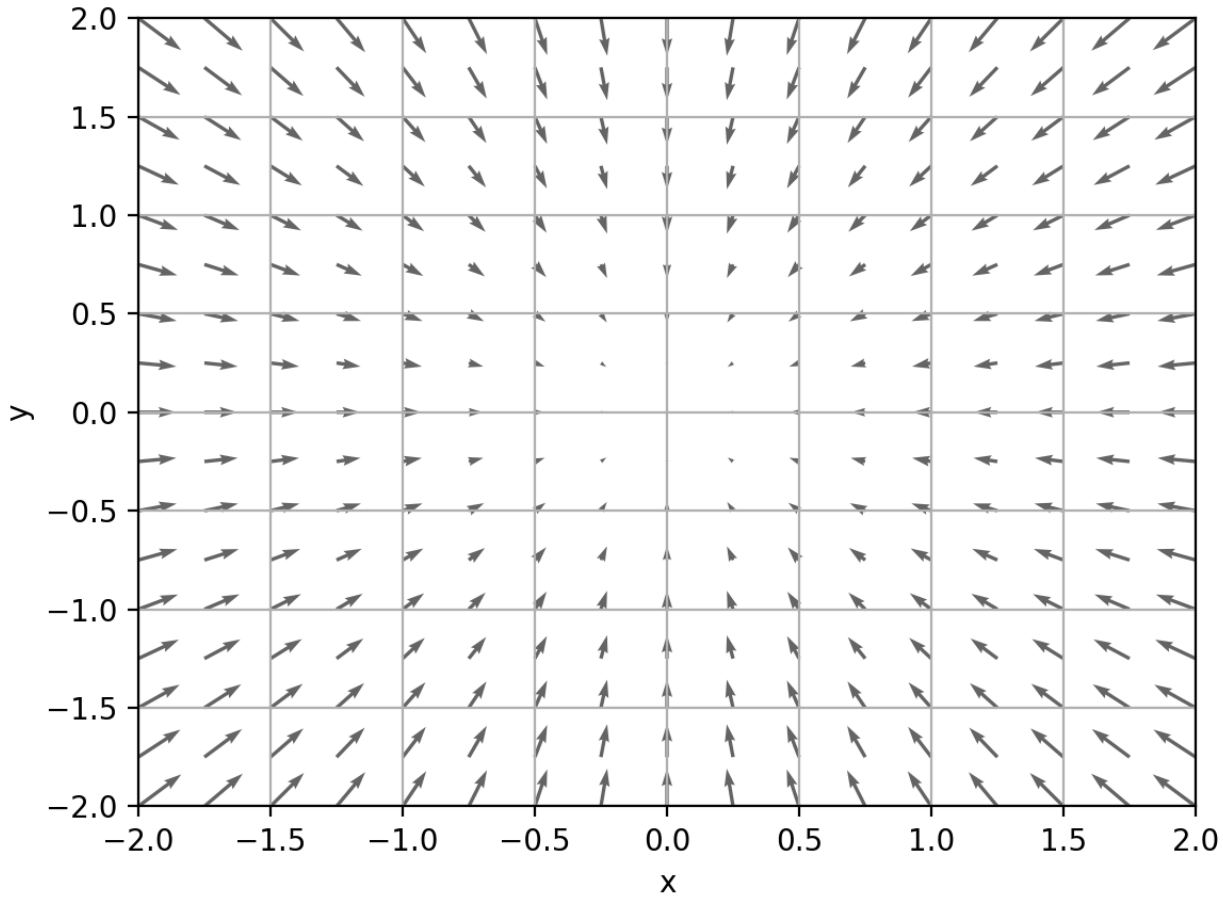
（２変数に限定せず、ベクトル x を引数としていることに注意）

```
# coding: utf-8
import numpy as np
def numerical_gradient(f, x):
    h = 1e-4  # 0.0001
    grad = np.zeros_like(x)
    for idx in range(x.size):
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + h
        fxh1 = f(x)  # f(x+h)
        x[idx] = tmp_val - h
        fxh2 = f(x)  # f(x-h)
        grad[idx] = (fxh1 - fxh2) / (2*h)
        x[idx] = tmp_val  # 値を元に戻す
    return grad
```

勾配

☆ $f(x, y) = x^2 + y^2$ の勾配のグラフ

（矢印は低くなる方向を示していて、中央が一番低い）



勾配法

- 機械学習：学習の際に [　　] を探索する
- [　　]：損失関数が最小なとき
- そこで、勾配をうまく使う=勾配法
  - 現在の場所から勾配方向に一定の鋸歯
  - これを繰り返し関数の値を徐々に減らしていく

勾配法

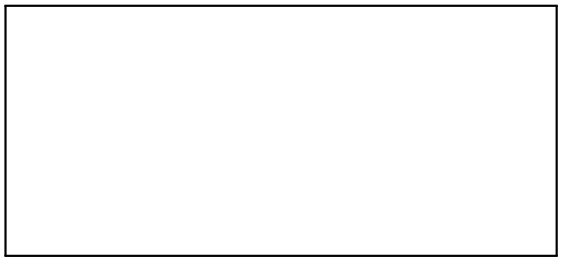
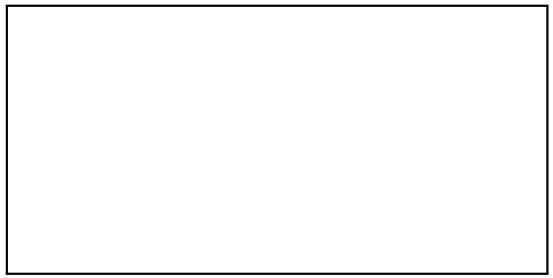
- 機械学習：学習の際に **最適なパラメータ** を探索する
- [　　]: 損失関数が最小なとき
- そこで、勾配をうまく使う=勾配法
  - 現在の場所から勾配方向に一定の鋸歯
  - これを繰り返し関数の値を徐々に減らしていく

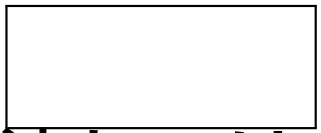
勾配法

- 機械学習：学習の際に **最適なパラメータ** を探索する
- **最適なパラメータ**：損失関数が最小なとき
- そこで、勾配をうまく使う=勾配法
  - 現在の場所から勾配方向に一定の鋸歯
  - これを繰り返し関数の値を徐々に減らしていく

勾配法

- 式であらわすと


,


- η（イータ）は

：大きすぎても少なすぎても良い場所に辿りつけない。変更しながら確認する作業が必要


-

とも言う（上昇も可、いずれにせよ本質は同じ）

勾配法

- 式であらわすと

$$x = x - \eta \frac{\partial f}{\partial x},$$

- η（イータ）は ：大きすぎても少なすぎても良い場所に辿りつけない。変更しながら確認する作業が必要
-

とも言う（上昇も可、いずれにせよ本質は同じ）

勾配法

- 式であらわすと

$$x = x - \eta \frac{\partial f}{\partial x}, \quad y = y - \eta \frac{\partial f}{\partial y}$$

- η（イータ）は [　　]：大きすぎても少なすぎても良い場所に辿りつけない。変更しながら確認する作業が必要 [　　]
- [　　]

とも言う（上昇も可、いずれにせよ本質は同じ）

勾配法

- 式であらわすと

$$x = x - \eta \frac{\partial f}{\partial x}, \quad y = y - \eta \frac{\partial f}{\partial y}$$

- η（イータ）は **学習率**：大きすぎても少なすぎても良い場所に辿りつけない。変更しながら確認する作業が必要
-
とも言う（上昇も可、いずれにせよ本質は同じ）

勾配法

- 式であらわすと

$$x = x - \eta \frac{\partial f}{\partial x}, \quad y = y - \eta \frac{\partial f}{\partial y}$$

- η（イータ）は **学習率**：大きすぎても少なすぎても良い場所に辿りつけない。変更しながら確認する作業が必要 **（ハイパーパラメータ）**
- [ ]

とも言う（上昇も可、いずれにせよ本質は同じ）

勾配法

- 式であらわすと

$$x = x - \eta \frac{\partial f}{\partial x}, \quad y = y - \eta \frac{\partial f}{\partial y}$$

- η（イータ）は **学習率**：大きすぎても少なすぎても良い場所に辿りつけない。変更しながら確認する作業が必要 **（ハイパーパラメータ）**
- **勾配降下法 (gradient descent method)** とも言う（上昇も可、いずれにせよ本質は同じ）

勾配法（実装）

☆ 勾配法の実装例（前の numerical_gradient を利用。**動かす必要なし**）

```
def gradient_descent(f,init_x,lr=0.01, \
                     step_num=100):
    x = init_x
    x_history = []
    for i in range(step_num):
        x_history.append( x.copy() )
        grad = numerical_gradient(f, x)
        x -= lr * grad
    return x, np.array(x_history)
```

ニューラルネットに対する勾配

- 多層ニューラルネットの場合、当然勾配も行列になる
- たとえば 2x3 の重みだけをもつニューラルネットの場合、その重みとその勾配は、損失関数を L とすると

$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix},$$

$$\frac{\partial L}{\partial W} = \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{13}} \\ \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{23}} \end{pmatrix}$$

学習アルゴリズムの実装（再掲）

1. ミニバッチ: 訓練データの中からランダムに一部のデータ（ミニバッチ）を選ぶ。
2. 勾配の算出: ミニバッチ の 損失関数 を減らすために、各重みパラメータの 勾配 を求める。 勾配は損失関数 の値を最も減らす方向を示す。
3. パラメータの更新: 重みパラメータを 勾配 方向に微小量だけ 更新する
4. 上を繰りかえす

2層ニューラルネットのクラス

☆ 主要構造のみ。全コードは two_layer_net.py をリンク先から Colab. の mymodules フォルダにコピーする

```
class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size, \
                 weight_init_std=0.01):
        # 重みの初期化
    def predict(self, x):
        # 推論を行なう
    def loss(self, x, t):
        # x:入力データ, t:教師データ
        # 損失関数の値を求める
    def accuracy(self, x, t):
        # 認識精度を求める
    def numerical_gradient(self, x, t):
        # 勾配を求める
    def gradient(self, x, t):
        # 勾配を求める. 上の高速版
```

２層ニューラルネットのクラス

☆パラメータは大きく２つ

- [　　]: ネットワークのパラメータ全般…重み、バイアス
- [　　]: 勾配を保持する辞書変数（params 変数と対応）

２層ニューラルネットのクラス

☆パラメータは大きく２つ

- params: ネットワークのパラメータ全般…重み、バイアス
- : 勾配を保持する辞書変数（params 変数と対応）

2層ニューラルネットのクラス

☆パラメータは大きく2つ

- params: ネットワークのパラメータ全般…重み、バイアス
- grads: 勾配を保持する辞書変数（params 変数と対応）

ミニバッチ学習の実装と評価

- 学習の実装は、ミニバッチ学習で行なう
- → [　　]に [　　]のデータを取り出してそれを対象に勾配法によりパラメータを更新する

ミニバッチ学習の実装と評価

- 学習の実装は、ミニバッチ学習で行なう
- → 無作為に □ のデータを取り出してそれを対象に勾配法によりパラメータを更新する

ミニバッチ学習の実装と評価

- 学習の実装は、ミニバッチ学習で行なう
- → 無作為に 一部のデータを取り出してそれを対象に勾配法によりパラメータを更新する

ミニバッチ学習の実装と評価

- ニューラルネットの学習では訓練データ [　　] のデータを正しく認識できるかどうか確認する必要がある
- これは過学習していないかの確認＝そもそも、[　　] を身につけることが目標
- そこで、学習を行なう過程で定期的に訓練データとテストデータを対象に [　　] を記録する
- ここでは１ [　　] ごとに記録

ミニバッチ学習の実装と評価

- ニューラルネットの学習では訓練データ 以外 のデータを正しく認識できるかどうか確認する必要がある
- これは過学習していないかの確認＝そもそも、 を身につけることが目標
- そこで、学習を行なう過程で定期的に訓練データとテストデータを対象に を記録する
- ここでは１ ごとに記録

ミニバッチ学習の実装と評価

- ニューラルネットの学習では訓練データ **以外** のデータを正しく認識できるかどうか確認する必要がある
- これは過学習していないかの確認＝そもそも、**汎化能力** を身につけることが目標
- そこで、学習を行なう過程で定期的に訓練データとテストデータを対象に [　　] を記録する
- ここでは１ [　　] ごとに記録

ミニバッチ学習の実装と評価

- ニューラルネットの学習では訓練データ **以外** のデータを正しく認識できるかどうか確認する必要がある
- これは過学習していないかの確認＝そもそも、**汎化能力** を身につけることが目標
- そこで、学習を行なう過程で定期的に訓練データとテストデータを対象に **認識精度** を記録する
- ここでは１ [　　] ごとに記録

ミニバッチ学習の実装と評価

- ニューラルネットの学習では訓練データ**以外**のデータを正しく認識できるかどうか確認する必要がある
- これは過学習していないかの確認＝そもそも、**汎化能力**を身につけることが目標
- そこで、学習を行なう過程で定期的に訓練データとテストデータを対象に**認識精度**を記録する
- ここでは１**エポック**ごとに記録

ミニバッチ学習の実装と評価

☆ エポックとは

- 学習の単位
- １エポックとは学習において訓練データを全て　　ときの回数
  - 例：10000 個の訓練データに対して 100 個のミニバッチで学習する場合、勾配法を 100 回繰り返したら、全てのデータを「見た」ことになる
  - → 100 回＝１エポック

ミニバッチ学習の実装と評価

☆ エポックとは

- 学習の単位
- １エポックとは学習において訓練データを全て使いきったときの回数
  - 例：10000 個の訓練データに対して 100 個のミニバッチで学習する場合、勾配法を 100 回繰り返したら、全てのデータを「見た」ことになる
  - → 100 回＝１エポック

ミニバッチ学習の実装と評価

- 全コードが tr_nn.py にあるので、リンク先からコピーして Colab. で実行する
- １エポックごとに認識精度を計算しているところに注意

ミニバッチ学習の実装と評価

```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
bd = 'drive/My Drive/Colab Notebooks/mymodules'
import sys, os
import numpy as np
import matplotlib.pyplot as plt
sys.path.append(bd)
from mnist import load_mnist
from two_layer_net import TwoLayerNet
```

ミニバッチ学習の実装と評価

```
# データの読み込み
(x_train, t_train), (x_test, t_test)
    = load_mnist(normalize=True, one_hot_label=True
network = TwoLayerNet(
        input_size=784,hidden_size=50,output_size=10
iters_num = 10000  # 繰り返しの回数を適宜設定する
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1
train_loss_list = []
train_acc_list = []
test_acc_list = []
iter_per_epoch = max(train_size / batch_size, 1)
```

ミニバッチ学習の実装と評価

```
for i in range(iters_num):
    batch_mask = np.random.choice(train_size, bat
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]
    # 勾配の計算
    #grad = network.numerical_gradient(x_batch, t
    grad = network.gradient(x_batch, t_batch)
    # パラメータの更新
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * gr
```

ミニバッチ学習の実装と評価

```
loss = network.loss(x_batch, t_batch)
train_loss_list.append(loss)
if i % iter_per_epoch == 0:
    train_acc = network.accuracy(x_train,t_train)
    test_acc = network.accuracy(x_test, t_test)
    train_acc_list.append(train_acc)
    test_acc_list.append(test_acc)
    print("train acc, test acc | "
        + str(train_acc) + ", " + str(test_acc))
```

ミニバッチ学習の実装と評価

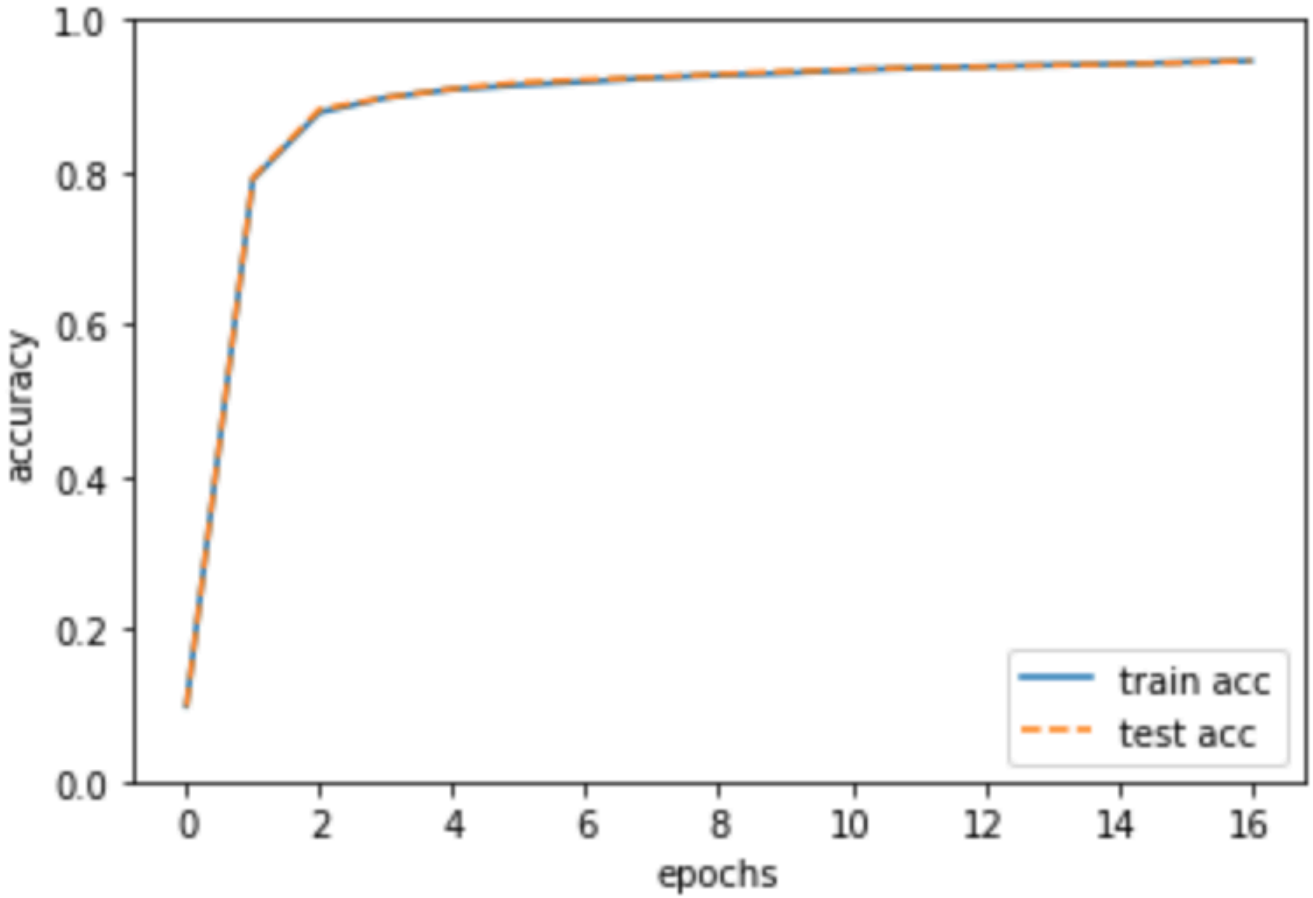
```
# グラフの描画
markers = {'train': 'o', 'test': 's'}
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, label='train acc')
plt.plot(x, test_acc_list, label='test acc', line
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
```

ミニバッチ学習の実装と評価

◎結果（図は次頁）

```
Mounted at /content/drive
train acc, test acc | 0.09871666666666666, 0.098
train acc, test acc | 0.7904666666666667, 0.7922
train acc, test acc | 0.8782833333333333, 0.8822
（途中略）
train acc, test acc | 0.93885, 0.9381
train acc, test acc | 0.9403833333333333, 0.9403
train acc, test acc | 0.9419666666666666, 0.9416
train acc, test acc | 0.9443166666666667, 0.9436
train acc, test acc | 0.9459333333333333, 0.9462
```

ミニバッチ学習の実装と評価



まとめ

- 機械学習で使用するデータセットは [　　] 用と [　　] 用にわける
- ニューラルネットの学習は [　　] を指標として、損失関数が小さくなるように [　　] を更新する
- 重みパラメータを更新するときには [　　] を用いる

まとめ

- 機械学習で使用するデータセットは 訓練 用と　　用にわける
- ニューラルネットの学習は　　を指標として、損失関数が小さくなるように　　を更新する
- 重みパラメータを更新するときには　　を用いる

まとめ

- 機械学習で使用するデータセットは 訓練 用と テスト 用にわける
- ニューラルネットの学習は [　　] を指標として、損失関数が小さくなるように [　　] を更新する
- 重みパラメータを更新するときには [　　] を用いる

まとめ

- 機械学習で使用するデータセットは 訓練 用と テスト 用にわける
- ニューラルネットの学習は 損失関数 を指標として、損失関数が小さくなるように [　　] を更新する
- 重みパラメータを更新するときには [　　] を用いる

まとめ

- 機械学習で使用するデータセットは 訓練 用と テスト 用にわける
- ニューラルネットの学習は 損失関数 を指標として、損失関数が小さくなるように 重みパラメータ を更新する
- 重みパラメータを更新するときには [　　] を用いる

まとめ

- 機械学習で使用するデータセットは **訓練** 用と **テスト** 用にわける
- ニューラルネットの学習は **損失関数** を指標として、損失関数が小さくなるように **重みパラメータ** を更新する
- 重みパラメータを更新するときには **勾配法** を用いる

おしまい

質問・コメント等あれば、何でもお気軽に
aipr@e-chan.jp に
メールください！