

機械学習講習会 第六回

- 「ニューラルネットワークの実装」 です！

Jupyter / Google Colaboratoryで今回使う新規ファイルを作成して
構えて待機しておいてください

機械学習講習会 第六回

- 「ニューラルネットワークの実装」

traP アルゴリズム班 Kaggle部

2023/7/3

第一回: 学習

第二回: 勾配降下法

第三回: 自動微分

第四回: ニューラルネットワークの構造

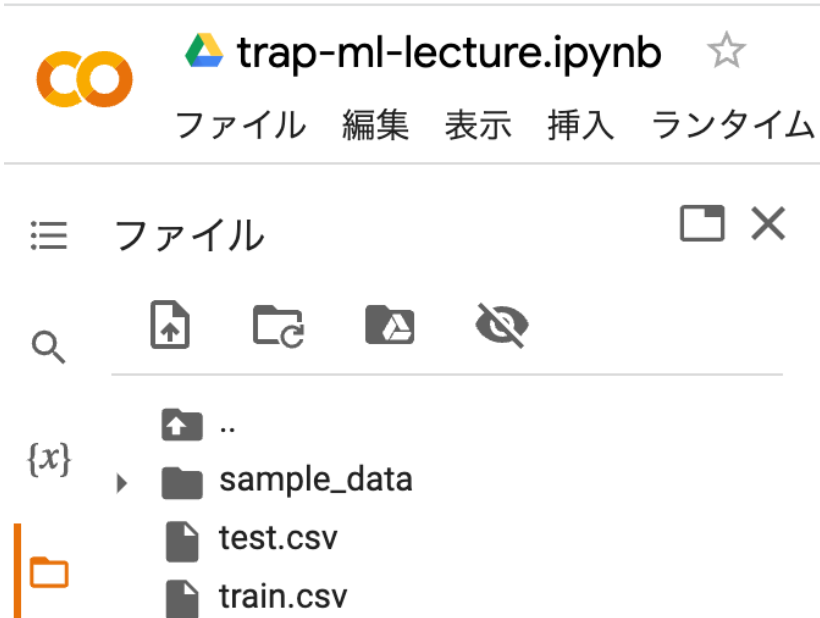
第五回: ニューラルネットワークの学習と評価

第六回: ニューラルネットワークの実装

今回解くタスク

セルに、以下のコマンドを入力:

```
!curl -L https://abap34.com/ml-lecture/train.csv -o train.csv  
!curl -L https://abap34.com/ml-lecture/test.csv -o test.csv
```



データ

- train.csv
3211 行 x 17列
- test.csv
803 行 x 16列

test.csvtrain.csv

1 to 10 of 3211 entriesFilter

最高気温(°C)	平均気温(°C)	降水量の合計(mm)	日照時間(時間)	平均風速(m/s)	平均雲量(10分比)	平均湿度(%)	子供割合	ice1	ice2	ice3	ice4	ice5	ice6	ice7	ice8	売り上げ
16.7	12.6	0.0	7.7	5.5	5.5	61.0	11.4	250	150	150	300	300	350	250	350	0.5955326509621945
25.900000000000002	22.599999999999998	21.14597132295255	2.0	2.3	10.0	75.0	12.6	450	300	100	150	200	350	350	350	0.44641586554590684
26.7	20.0	21.75470734052112	13.5	4.7	3.3	45.0	10.9	100	200	400	450	450	150	400	200	0.45303182441582446
31.3	27.099999999999998	0.0	0.9	2.7	9.8	85.0	10.8	350	200	250	450	450	300	150	150	0.6442353972371553
26.2	19.7	0.0	3.3	2.4	10.0	85.0	11.3	150	450	350	150	150	300	250	200	0.6159198470772944
22.400000000000002	15.2	0.0	11.5	3.1	3.5	44.0	12.6	100	400	150	250	250	200	450	100	0.6001764011884118
37.0	31.4	0.0	9.4	3.4	7.5	71.0	10.8	100	400	250	450	400	400	450	300	0.6813853990468928
24.900000000000002	23.099999999999998	0.0	0.2	2.0	10.0	93.0	10.8	450	150	200	150	200	450	350	150	0.6190911886951989
34.5	29.0	0.11080070863841747	10.5	3.0	5.8	66.0	11.0	400	300	250	400	200	150	300	450	0.6492066638924047
30.3	24.599999999999998	61.70445455784176	13.0	3.7	2.0	56.0	11.0	100	450	100	350	300	100	450	150	0.44859429084515184

Show10per page1210100300320322

データ



はキッチンカーでアイスを売っています。

なるべく多くの売り上げをあげたいので、売り上げを予測するモデルを構築して、売り上げを予測することにしました。

そこで、ある日に行ったところの気温や降水量などの気象データ、いく地域の人口に占める子供の割合、アイスの値段などなどと、売り上げをまとめたデータを作りました。(train.csv)



そこで、 のために気象データや値段などから、売り上げを予測するモデルを作って、未知のデータ(test.csv)に対して予測してください！

せっかくなので

<https://dacq.trap.show/>

予測結果を投稿して、精度を競えます！！！！すごい！！！！


DacQ

投稿

ファイル

選択されていません

Upload

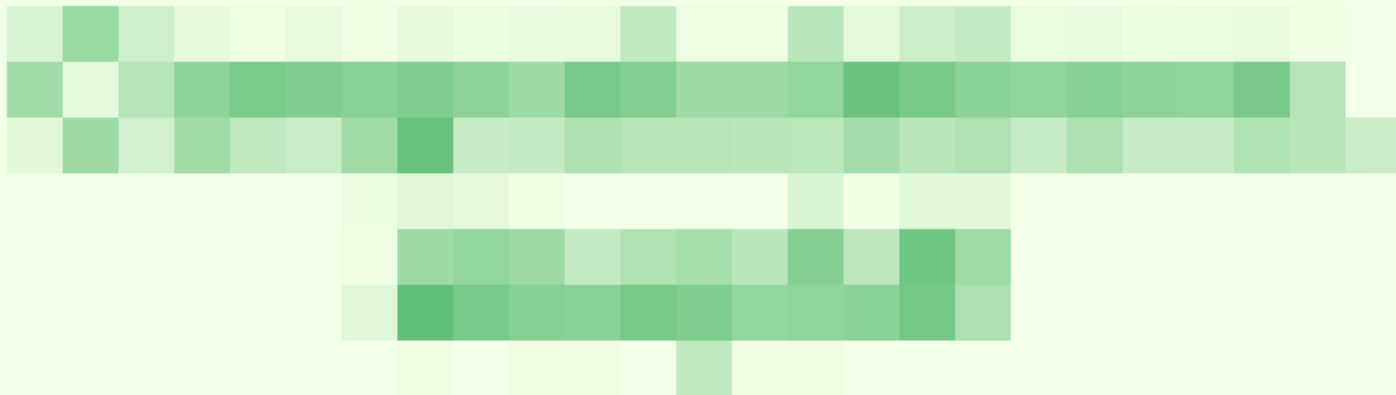
Rank	User	Loss
1	 abap34	0.003894386693650532

せっかくなので

<https://dacq.trap.show/>

自分のベストスコアを更新すると、激アツアニメーションが見れます！！！！すごいで！！！！！！！！！！！！！！！！！！！！！！

最高スコアを更新してtraQに投稿しよう！



全体の流れ

1. データの読み込み
2. モデルの構築
3. モデルの学習
4. 新規データに対する予測
5. 順位表への提出

1-1. データの読み込み



1-2. データの前処理



1-2. PyTorchに入力できる形に

1-1. データの読み込み

```
# pandasパッケージを`pd`という名前をつけてimport
import pandas as pd

# これによって、pandasの関数を`pd.関数名`という形で使えるようになる
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```



`pd.read_csv(path)` で、 `path` にあるcsvファイルを読み込める

パスとは、ファイルの場所を示す文字列のことです。

今回は、train.csvとtest.csvが、ノートブックと同じ場所にあるので、"train.csv"と"test.csv"という文字列をパスとして指定しています。例えば、"/home/abap34/train.csv"というパスを指定すると、"/home/abap34"というフォルダの中にあるtrain.csvというファイルを読み込みます。他にもノートブックの位置からの相対パスを指定することもできます。例えば、"./train.csv"というパスを指定すると、ノートブックの一つ上のフォルダにあるtrain.csvというファイルを読み込みます。

1-1. データの読み込み

train

とだけセルに入力すると

✓
0
秒

▶

1

train

	最高気温(℃)	平均気温(℃)	降水量の合計(mm)	日照時間(時間)	平均風速(m/s)	平均雲量(10分比)	平均湿度(%)	子供割合	ice1	ice2	ice3	ice4	ice5	ice6	ice7	ice8	売り上げ
0	16.7	12.6	0.000000	7.7	5.5	5.5	61.0	11.4	250	150	150	300	300	350	250	350	0.595533
1	25.9	22.6	21.145971	2.0	2.3	10.0	75.0	12.6	450	300	100	150	200	350	350	350	0.446416
2	26.7	20.0	21.754707	13.5	4.7	3.3	45.0	10.9	100	200	400	450	450	150	400	200	0.453032
3	31.3	27.1	0.000000	0.9	2.7	9.8	85.0	10.8	350	200	250	450	450	300	150	150	0.644235
4	26.2	19.7	0.000000	3.3	2.4	10.0	85.0	11.3	150	450	350	150	150	300	250	200	0.615920
...
3206	12.9	10.9	4.958982	2.2	2.2	8.3	63.0	11.0	150	100	300	400	450	150	150	250	0.415770
3207	32.0	24.9	22.364472	9.0	3.5	8.5	58.0	11.3	100	100	150	450	100	450	250	350	0.448675
3208	17.1	10.1	3.278023	10.9	2.0	3.5	46.0	11.0	350	300	250	350	450	100	400	350	0.453402
3209	12.1	6.7	42.488025	9.5	2.5	5.0	56.0	11.0	250	150	150	200	250	350	300	400	0.415192
3210	15.0	11.3	2.858941	2.2	5.0	7.3	87.0	11.4	200	350	200	250	200	250	200	200	0.428246

3211 rows x 17 columns

1-1. データの読み込み

test

とだけセルに入力すると

0秒

1test

↑

↓

🔗

💬

⚙️

	最高気温(℃)	平均気温(℃)	降水量の合計(mm)	日照時間(時間)	平均風速(m/s)	平均雲量(10分比)	平均湿度(%)	子供割合	ice1	ice2	ice3	ice4	ice5	ice6	ice7	ice8
0	10.0	4.1	0.000000	10.5	3.8	0.5	51.0	11.0	250	100	300	400	200	450	250	300
1	32.2	27.4	65.503225	2.7	2.4	8.5	85.0	11.9	200	450	200	350	250	400	150	200
2	13.7	9.6	0.599091	3.2	3.7	2.0	70.0	11.0	150	450	350	150	250	100	200	100
3	25.3	19.3	11.622781	10.7	2.8	6.0	52.0	11.0	150	400	300	200	100	100	100	150
4	12.0	7.7	0.000000	6.6	3.1	8.5	62.0	11.3	450	350	250	300	450	150	300	350
...
798	12.8	8.1	45.979216	4.1	3.7	5.3	72.0	11.0	200	350	300	400	300	300	200	450
799	37.8	32.9	0.000000	10.1	4.1	6.3	58.0	10.8	300	200	300	200	150	200	100	250
800	18.7	15.2	0.000000	1.1	1.8	7.8	82.0	11.3	150	150	150	250	450	150	200	350
801	16.1	10.6	35.940064	3.8	2.0	7.0	72.0	11.3	450	350	100	100	300	200	200	150
802	8.3	4.2	0.000000	3.9	2.2	5.0	73.0	11.0	350	350	200	150	300	300	200	150

803 rows × 16 columns

1-1. データの読み込み

今までは...

```
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

def loss(a):
    n = len(x)
    s = 0
    for i in range(n):
        s += (y[i] - a * x[i])**2

    return s / n
```

⇒ 今回のデータも入力と出力(の目標) に分けておく

1-1. データの読み込み

```
train['カラム名']
```

で「カラム名」という名前の列を取り出せる



今回の予測の目標は

```
train['売り上げ']
```

1-1. データの読み込み

セルに、

```
train_y = train['売り上げ']
```

と入力して実行

⇒ `train_y` に売り上げの列が入る

1-1. データの読み込み

逆に、モデルに入力するデータは、`train` から売り上げの列を除いたもの

```
train.drop(columns=['カラム名'])
```

を使うと、`train` から「カラム名」という名前の列を除いたものを取り出せる



今回は「売り上げ」を除けば良いので、

```
train.drop(columns=['売り上げ'])
```

1-1. データの読み込み

セルに、

```
train_x = train.drop(columns=['売り上げ'])
```

と入力して実行

⇒ `train_x` に売り上げの列を除いたデータが入る

1-1. データの読み込み

✓ データの読み込みが完了!

今の状況...

- `train_x` ...モデルに入力するデータ(気温、値段、etc...)
- `train_y` ...モデルの出力の目標(売り上げ)
- `test` ...予測対象のデータ(気温、値段、etc...)

が入っている

1-2. データの前処理

✓ データをそのままモデルに入れる前に処理をすることで、
学習の安定性や精度を向上可能

今回は、
各列に対して「標準化」と呼ばれる処理を行う

1-2. データの前処理

標準化

$$x' = \frac{x - \mu}{\sigma}$$

(μ は平均、 σ は標準偏差)

- 平均 μ_1 のデータの全ての要素から μ_2 を引くと、平均は $\mu_1 - \mu_2$
- 標準偏差 σ_1 のデータの全ての要素を σ_2 で割ると、標準偏差は $\frac{\sigma_1}{\sigma_2}$

⇒ 標準化によって、平均を0、標準偏差を1にできる

1-2. データの前処理

`scikit-learn` というライブラリの `StandardScaler` を使うと、簡単に標準化できる！

```
# sklearn.preprocessingに定義されているStandardScalerを使う
from sklearn.preprocessing import StandardScaler

# StandardScalerのインスタンスを作成
scaler = StandardScaler()
scaler.fit(train_x)
train_x = scaler.transform(train_x)
test = scaler.transform(test)
```

`scaler.fit` 関数によって引数で渡されたデータの各列ごとの平均と標準偏差を計算され、`scaler` に保存されます。そして、`scaler.transform` 関数によってデータが実際に標準化されます。勘がいい人は、「`test` に対しても `train_x` で学習した平均と標準偏差を使って標準化しているけど大丈夫なのか？」と思ったかもしれないですね。結論から言うとそうなのですが、意図しています。ここに理由を書いたら信じられないくらいはみ出てしまったので、省略します。興味がある人は「Kaggleで勝つデータ分析の技術」p.124あたりを参照してみてください。

1-2. データの前処理

```
train_x
```

```
test
```

などをセルに入力して実行してみると、
確かに何かしらの変換がされている
(ついでに、結果が数字だけになっている)

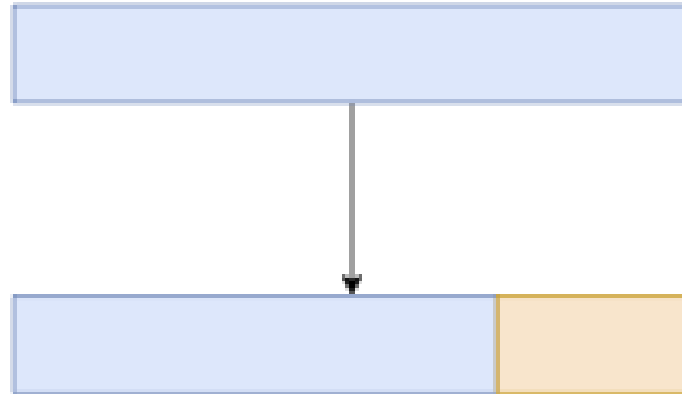
ので、`train_y` も数字だけにしておく

```
train_y = train_y.values.reshape(-1, 1)
```

最初のテーブルっぽい情報を持ったまま計算を進めたい場合は、`train_x[:] = scaler.transform(train_x)` のようにすると良いです。

1-2. データの前処理 - バリデーション

バリデーションのためにデータを分割しておく



(あんまり前処理には含めない人が多いと思いますが。。。。。。。。ここでやっておきます！)

1-2. データの前処理 - バリデーション

scikit-learn の train_test_split を使うと、簡単にデータを分割できる！

```
from sklearn.model_selection import train_test_split
train_x, val_x, train_y, val_y = train_test_split(train_x, train_y, test_size=0.3, random_state=34)
```

```
train_test_split(train_x, train_y, test_size=0.3, random_state=34)
```

- (train_x, train_y)を、学習データ:検証データ = 7:3に分割

```
train_x.shape
```

```
val_x.shape
```

を確認すると、確かに7:3に分割されていることがわかる

1-3. PyTorchに入力できる形に

- ✅ このあとこれらをPyTorchで扱うので、PyTorchで扱える形にする

1-3. PyTorchに入力できる形に

数として**Tensor型**を使って自動微分などを行える

```
>>> x = torch.tensor(2.0, requires_grad=True)
>>> def f(x):
...     return x ** 2 + 4 * x + 3
...
>>> y = f(x)
>>> y.backward()
>>> x.grad
tensor(8.)
```

($f(x) = x^2 + 4x + 3$ の $x = 2$ における微分係数8)

⇒ データをTensor型に直しておく

1-3. PyTorchに入力できる形に

torch.tensor関数によるTensor型のオブジェクトの作成

```
torch.tensor(data, requires_grad=False)
```

- `data`: 保持するデータ(配列**っぽいもの**のなんなんでも)
 - リスト、タプル、NumPy配列、スカラ、...
- `requires_grad`: 勾配を保持するかどうかのフラグ
 - デフォルトはFalse
 - 勾配計算を行う場合はTrueにする

1-3. PyTorchに入力できる形に

```
import torch

train_x = torch.tensor(train_x)
train_y = torch.tensor(train_y)
val_x = torch.tensor(val_x)
val_y = torch.tensor(val_y)
test = torch.tensor(test)
```

我々が勾配降下法で使うのは、

損失についての各パラメータの勾配

⇒ 入力データに対する勾配は不要なので

`requires_grad=True` とする必要はない

✓ 1-1. データの読み込み



✓ 1-2. データの前処理



✓ 1-2. PyTorchに入力できる形に

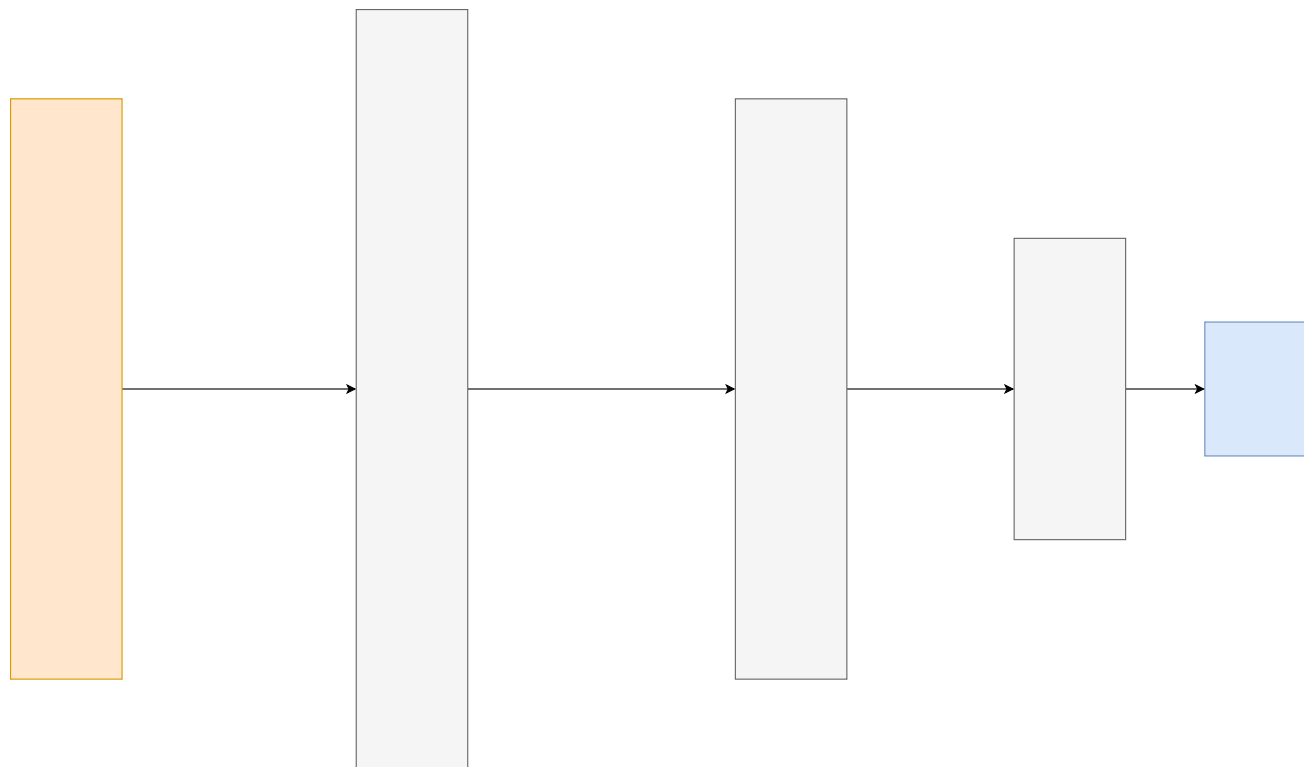
全体の流れ

1. データの読み込み
2. モデルの構築
3. モデルの学習
4. 新規データに対する予測
5. 順位表への提出

2. モデルの構築

今からすること...

$f(x; \theta)$ をつくる



2. モデルの構築

torch.nn.Sequentialによるモデルの構築

入力層は16次元の入力を受け取り、出力が32次元の全結合層

隠れ層は一つあり、32次元の入力を受け取り、出力が64次元の全結合層

出力層は64次元の入力を受け取り、出力が1次元の全結合層

```
import torch.nn as nn

model = nn.Sequential(
    nn.Linear(16, 32),
    nn.Sigmoid(),
    nn.Linear(32, 64),
    nn.Sigmoid(),
    nn.Linear(64, 1)
)
```

2. モデルの構築

```
import torch.nn as nn

model = nn.Sequential(
    nn.Linear(16, 32),
    nn.Sigmoid(),
    nn.Linear(32, 64),
    nn.Sigmoid(),
    nn.Linear(64, 1)
)
```

`nn.Sequential` は、順番に層をつなげていくモデルを作るためのクラス
引数に層を順番に渡すことで、モデルを構築してくれる

⇒ すでにこの時点でパラメータの初期化などは終わっている

2. モデルの構築

`model.parameters()` または `model.state_dict()` でモデルのパラメータを確認できる

```
model.state_dict()
```

を実行すると、モデルが持つパラメーター一覧を確認できる

2. モデルの構築

構築したモデルは、関数のように呼び出すことができる

```
import torch
dummy_input = torch.rand(1, 16)
model(dummy_input)
```

`torch.rand` で、ダミーのインプットを作成

⇒ モデルに入力

(現段階では、初期化されて学習されていない重みによる計算)

✓ $f(x; \theta)$ をつくる

⇒ あとはこれを勾配降下法の枠組みで学習させる！



思い出すシリーズ

確率的勾配降下法
(ミニバッチ学習)

全体の流れ

1. データの読み込み
2. モデルの構築
3. モデルの学習
4. 新規データに対する予測
5. 順位表への提出

3-1. ミニバッチ学習のための準備



3-2. 確率的勾配降下法の実装

3-1. ミニバッチ学習のための準備

ミニバッチ学習

損失関数は

$$L(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N (y_i - f(\boldsymbol{x}_i; \boldsymbol{\theta}))^2$$

- ⇒ これを計算するには、 N 回の計算が必要
- ⇒ データ数が少ない場合はいいが、数GB, TB,となっていく
- ⇒ メモリに乗り切らず実用的な速度で計算することが難しくなる！

3-1. ミニバッチ学習のための準備

✓ データをランダムにいくつか選んで、そのデータだけを使って損失関数を計算する

例) データ数 $N = 10000$ から $s = 100$ 個のデータをランダムに選ぶ
⇒ メモリに乗り切って計算できる！

この選んだ小さいデータの集合のことをミニバッチと呼び、そのサイズを **バッチサイズ(batch size)** と呼ぶ

そしてこれを使った勾配降下法を「確率的勾配降下法」という

確率的勾配降下法を単一データのみの損失を計算して行う手法と呼ぶ派閥もいるみたいですが、ミニバッチ学習に対しても確率的勾配降下法と呼ぶ人が多いと思います。

3-1. ミニバッチ学習のための準備

つまり...

我々がやらなきゃいけないこと

- データをいい感じにランダムに選んで供給する仕組みを作る

...

...

🔥 < 私がやろう

✅ `torch.utils.data.Dataset` と `torch.utils.data.DataLoader` を
使うと、簡単に実装できる！

- 現状確認...

`train_x`, `train_y`, `val_x`, `val_y`, `test` をTensor型で保持している

3-1. ミニバッチ学習のための準備

DatasetとDataLoaderの作成

```
# データセットの作成
train_dataset = TensorDataset(train_x, train_y)
val_dataset = TensorDataset(val_x, val_y)

# データローダの作成
batch_size = 32
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, drop_last=True)
val_dataloader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
```



1. Datasetの作成(`Dataset`型)

- データセット(データの入出力のペア)を表すオブジェクト

`TensorDataset` に

- モデルの入力データ(`train_x`)と
- 出力の目標データ(`train_y`)を渡すことで `Dataset` 型のオブジェクトが作られる

実際は `torch.utils.data.Dataset` を継承したクラスを作ることでも `Dataset` 型 (のサブクラス) のオブジェクトを作ることができます。この方法だと非常に柔軟な処理が行えるためこの方法が主流流です。(今回は簡単のために `TensorDataset` を使いました。)

3-1. ミニバッチ学習のための準備

```
from torch.utils.data import TensorDataset

# データセットの作成

# 学習データのデータセット
train_dataset = TensorDataset(train_x, train_y)
# 検証データのデータセット
val_dataset = TensorDataset(val_x, val_y)
```

1. DataLoaderの作成(`DataLoader`型)

- `Dataset` からミニバッチを取り出して供給してくれるオブジェクト

つまり....

- データをいい感じにランダムに選んで供給する仕組みを作る

をやってくれる

3-1. ミニバッチ学習のための準備

1. DataLoaderの作成(DataLoader型)

- Datasetからミニバッチを取り出して供給してくれるオブジェクト

```
DataLoader(dataset, batch_size=batch_size, shuffle=shuffle)
```

```
from torch.utils.data import DataLoader

batch_size = 32
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, drop_last=True)
val_dataloader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
```

⇒ これをfor文で回すことでデータを取り出すことができる

3-1. ミニバッチ学習のための準備

1. DataLoaderの作成(DataLoader型)

```
for inputs, targets in train_dataloader:  
    print('inputs.shape', inputs.shape)  
    print('targets.shape', targets.shape)  
    print('-----')
```



```
inputs.shape torch.Size([32, 16])  
targets.shape torch.Size([32, 1])  
-----  
inputs.shape torch.Size([32, 16])  
targets.shape torch.Size([32, 1])  
...
```

3-1. ミニバッチ学習のための準備

我々がやらなきゃいけないこと

- データをいい感じにランダムに選んで供給する仕組みを作る

をやってくれる



3.2 確率的勾配降下法の実装

データは回るようになった

⇒ あとは学習を実装しよう！

TODOリスト

1. 損失関数を設定する
2. 勾配の計算を行う
3. パラメータの更新を行う

3.2 確率的勾配降下法の実装

1. 損失関数を設定する

今回は、平均二乗和誤差(Mean Squared Error)を使う

⇒ これもPyTorchには用意されている

```
criterion = nn.MSELoss()
```

とすれば、

```
criterion(torch.tensor([1.0., 2.0, 4.0]), torch.tensor([2.0, 3.0, 4.0]))
```

と計算してくれる！

3.2 確率的勾配降下法の実装

TODOリスト

- ☑ 1. 損失関数を設定する
- 2. 勾配の計算を行う
- 3. パラメータの更新を行う

2. 勾配の計算を行う

やりかたは....？

3.2 確率的勾配降下法の実装

[illegible]

3.2 確率的勾配降下法の実装

```
# ここから
import torch.nn as nn

model = nn.Sequential(
    nn.Linear(16, 32),
    nn.Sigmoid(),
    nn.Linear(32, 64),
    nn.Sigmoid(),
    nn.Linear(64, 1)
)
# ここまではすでに入力したやつ

dummy_input = torch.rand(1, 16)
dummy_target = torch.rand(1, 1)

# 計算
pred = model(dummy_input)
loss = criterion(pred, dummy_target)

# backward
loss.backward()
```


3.2 確率的勾配降下法の実装

チェックポイント

1. **loss**に対する勾配を計算している

```
# backward  
loss.backward()
```

2. 勾配は**パラメータ**に対して計算される

```
for param in model.parameters():  
    print(param.grad)
```

(`dummy_input`, `dummy_target` は `requires_grad=False` なので勾配は計算されない)

3.2 確率的勾配降下法の実装

TODOリスト

- ☒ 1. 損失関数を設定する
- ☒ 2. 勾配の計算を行う
- 3. パラメータの更新を行う

3.2 確率的勾配降下法の実装

```
for epoch in range(epochs):
    for inputs, targets in train_dataloader:
        # 計算
        outputs = model(inputs)
        loss = criterion(outputs, targets)

        # backward
        loss.backward()

        # -----
        # ....
        # ここにパラメータの更新を書く
        # ....
        # -----
```

3.2 確率的勾配降下法の実装

これまでは、我々が手動(?)で更新するコードを書いていた

⇒ 🔥 < 私がやろう

✅ `torch.optim`のオプティマイザを使うことで、簡単にいろいろな最適化アルゴリズムを使える

3.2 確率的勾配降下法の実装

(完成版ではないです！)

```
optimizer = optim.SGD(model.parameters(), lr=lr)

# 学習ループ
for epoch in range(epochs):
    for inputs, targets in train_data_loader:
        # 勾配の初期化
        optimizer.zero_grad()

        # 計算
        outputs = model(inputs)
        loss = criterion(outputs, targets)

        # backward
        loss.backward()

        # パラメータの更新
        optimizer.step()
```

3.2 確率的勾配降下法の実装

```
optimizer = optim.SGD(params, lr=lr)
```

のようにすることで、`params`を更新の対象とするオプティマイザを作成できる(`lr`は学習率)

他にも、`optim.Adam`が使いたければ

```
optimizer = optim.Adam(params, lr=lr)
```

とするだけでOK！

⇒ 勾配を計算したあと、`optimizer.step()`によってパラメータを更新できる！

3.2 確率的勾配降下法の実装

⚠ 注意点

`optimizer.step()` で一回パラメータを更新するたびに
`optimizer.zero_grad()` で勾配を初期化する必要がある！
(これをしないと前回の `backward` の結果が残っておかしくなる)

↓ 次のページ...

学習の全体像を貼ります！！！！

3.2 確率的勾配降下法の実装

```
n_epoch = 10
for epoch in range(n_epoch):
    running_loss = 0.0

    for inputs, targets in train_dataloader:
        # 前の勾配を消す
        optimizer.zero_grad()

        # 計算
        outputs = model(inputs)
        loss = criterion(outputs, targets)

        # backwardで勾配を計算
        loss.backward()

        # optimizerを使ってパラメータを更新
        optimizer.step()

    running_loss += loss.item()

val_loss = 0.0
with torch.no_grad():
    for inputs, targets in val_dataloader:
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        val_loss += loss.item()

# エポックごとの損失の表示
train_loss = running_loss / len(train_dataloader)
val_loss = val_loss / len(val_dataloader)
print(f'Epoch {epoch + 1} - Train Loss: {train_loss:.4f} - Val Loss: {val_loss:.10f}')
```


3.2 確率的勾配降下法の実装

各行の解説...

- 1行目. `for epoch in range(n_epoch) n_epoch` 回分の学習をする
- 2行目. `running_loss = 0.0` 1エポックごとの訓練データの損失を計算するための変数
- 4行目. `for inputs, targets in train_dataloader` 訓練データを1バッチずつ取り出す(`DataLoader` の項を参照してください！)
- 6行目. `optimizer.zero_grad()` 勾配を初期化する。二つ前のページのスライドです！
- 9, 10行目. `outputs = ...` 損失の計算をします。

3.2 確率的勾配降下法の実装

- 13行目. `loss.backward()` 勾配の計算です。これによって `model` のパラメータに**損失に対する**勾配が記録されます
- 16行目. `optimizer.step()` `optimizer` が記録された勾配に基づいてパラメータを更新します。
- 18行目. `running_loss += loss.item()` 1バッチ分の損失を `running_loss` に足しておきます。
- 20行目~25行目. 1エポック分の学習が終わったら、検証データでの損失を計算します。検証用データの内容は、学習に影響させないので勾配を計算する必要がありません。したがって、`torch.no_grad()` の中で計算します。

3.2 確率的勾配降下法の実装

- 28行目～30行目. 1エポック分の学習が終わったら、訓練データと検証データの損失を表示します。 `len(train_data_loader)` は訓練データが何個のミニバッチに分割されたかを表す数、 `len(val_data_loader)` は検証データが何個のミニバッチに分割されたかを表す数です。
- 32行目. 損失を出力します。

3.2 確率的勾配降下法の実装

TODOリスト

- ✓ 1. 損失関数を設定する
- ✓ 2. 勾配の計算を行う
- ✓ 3. パラメータの更新を行う

3. 学習が完了！！！！

+ オプション 学習曲線を書いておこう

1. 各エポックの損失を記録する配列を作っておく

```
train_losses = []  
val_losses = []
```

1. 先ほどの学習のコードの中に、損失を記録するコードを追加する

```
train_loss = running_loss / len(train_dataloader)  
val_loss = val_loss / len(val_dataloader)  
train_losses.append(train_loss) # これが追加された  
val_losses.append(val_loss) # これが追加された  
print(f'Epoch {epoch + 1} - Train Loss: {train_loss:.4f} - Val Loss: {val_loss:.10f}')
```

3. 学習が完了！！！！

+ オプション 学習曲線を書いておこう

`matplotlib` というパッケージを使うことでグラフが書ける

```
# matplotlib.pyplot を plt という名前でimport  
import matplotlib.pyplot as plt
```

```
plt.plot(train_losses, label='train')  
plt.plot(val_losses, label='val')  
plt.legend()  
plt.xlabel('epoch')  
plt.ylabel('loss')  
plt.show()
```

⇒ いい感じのプロットを見よう！

全体の流れ

1. データの読み込み
2. モデルの構築
3. モデルの学習
4. 新規データに対する予測
5. 順位表への提出

4. 新規データに対する予測

思い出すシリーズ...

`test` に予測したい未知のデータが入っている

```
model(test)
```

⇒ 予測結果が出る

5. 順位表への提出

```
import csv

def write_pred(predictions, filename='submit.csv'):
    pred = predictions.squeeze().tolist()
    with open(filename, 'w', newline='') as f:
        writer = csv.writer(f)
        writer.writerows([[x] for x in pred])
```

をコピー

→

5. 順位表への提出

`write_pred(predictions)` を実行すると、`submit.csv` というファイルが作成されて、予測結果が出力される！！⇒これを提出しよう！

<https://dacq.trap.show/>


DacQ

投稿

ファイル

選択されていません

Upload

Rank	User	Loss
1	 abap34	0.003894386693650532

5. 順位表への提出

めざせ No.1 !

できること

- モデルの構造を変えてみる
- オプティマイザーを変えてみる
- 精度が出やすいようにデータを加工する
- 任意に決めた値(`batch_size` , `lr` , `n_epochs` など)を変えてみる

次回:

「ニューラルネットワークの発展」

「ニューラルネットワークの発展」

- MLP以外のニューラルネットワークのアーキテクチャ
- 最近アツい話
- 今後機械学習を勉強していくにあたって

etc...

部内データ分析コンペを開始します！！！！！！

- 昨年度も行った部内でのコンペを今年もやります
- 今年は、(ここだけに話) ちょっと豪華になりそうです
- 乞うご期待！！