Lecture 1 : Introduction of Deep Learning

Create at 2022/01/30

- 上課資源 :
  1. 課程 Youtube (https://www.youtube.com/watch?v=7XZR0-4uS5s)
  2. ML 2022 PyTorch Tutorial 1 (https://www.youtube.com/watch?v=85uJ9hSaXig)
  3. ML 2022 PyTorch Tutorial 2 (https://www.youtube.com/watch?v=VbqNn20FoHM)
  4. ML 2022 PyTorch Tutorial Colab (https://www.youtube.com/watch?v=YmPF0jrWn6Y)

介紹

- **機器學習** : 幫我們找一個人類寫不出來的複雜函式
- 這門課著重在機器學習裡的一個關鍵技術 **深度學習**
- **深度學習** : 找到的函式是一個類神經網路 (Neural Network)
- 提供機器有 Label 的資料
- 機器判斷一個函式的好壞 Loss (越小越好)
  - 執行函式用輸出結果判斷正確與否
- 機器自動找出 Loss 最低的函式

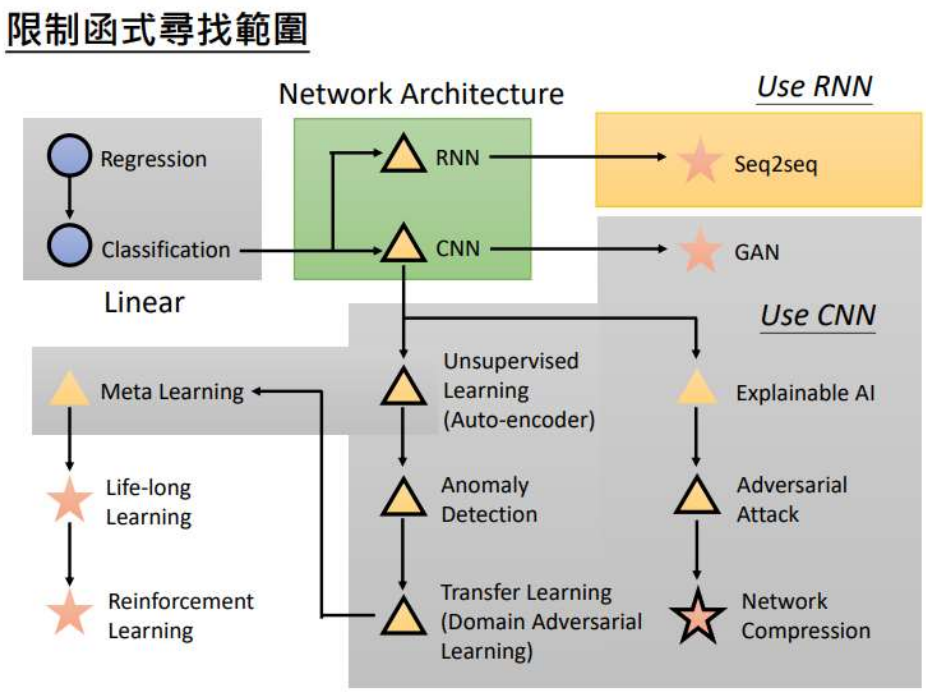
作業

- HW1 : COVID-19 確診率預測

    - 輸入 : vector
    - 輸出 : scalar
- HW2 : Phoneme Classification 語音辨識的簡化版
    - 輸入 : vector
    - 輸出 : classification
- HW3 : Image Classification 影像分類
    - 輸入 : matrix (圖片)
    - 輸出 : classification (圖片所對應的類別)
- HW4 : Speaker Classification 語者辨認
    - 輸入 : sequence (聲音)
    - 輸出 : classification (判斷是誰的聲音)
- HW5 : Machine Translation 機器翻譯
    - 輸入 : sequence (句子)
    - 輸出 : text (翻譯)
- HW6 : Anime Face Generation 動漫人臉生成
- HW7 : BERT (沒 Label 資料)
- HW8 : Anomaly Detection 異常檢測 (機器回答我不知道)
- HW9 : Explainable AI 可解釋性 AI (機器告訴我們他為甚麼知道答案)
- HW10 : Model Attack 模型攻擊
- HW11 : Domain Adaptation
- HW12 : Reinforcement Learning (不知道資料怎麼 Label，但是可以定義甚麼是成功的時候)
- HW13 : Network Compression 模型壓縮
- HW14 : Life-long Learning
- HW15 : Meta Learning 學習如何學習

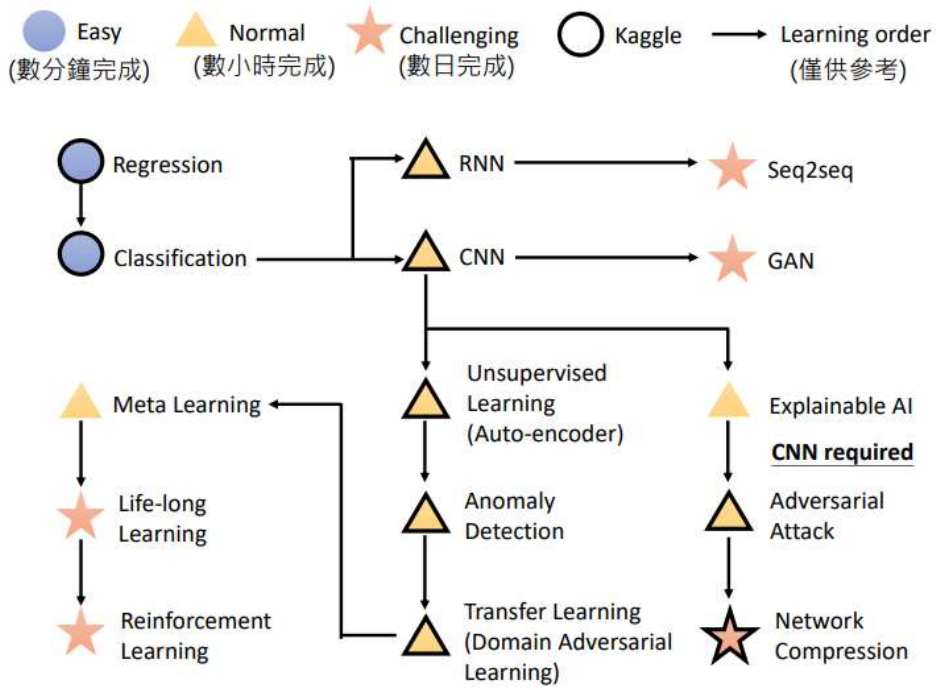
機器怎麼找出你想要的函式？

- 考慮 Linear function
- 考慮 Network Architecture



- Meta Learning = Learn to learn
- 學習如何學習的能力

作業說明



What is PyTorch ?

- 一個機器學習的框架
- 高維矩陣運算 **Tensor** 用 GPU 做加速運算
- 在神經網路上做梯度運算

Training & Testing Neural Networks

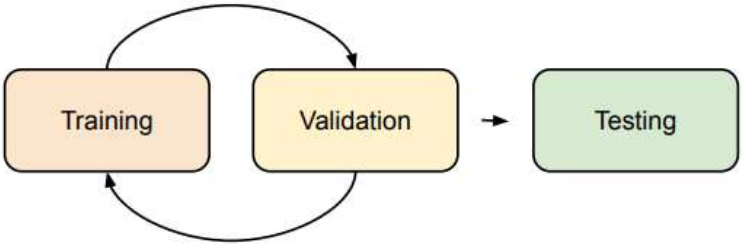
訓練神經網路前有三個步驟

Step 1 : Define Neural Network

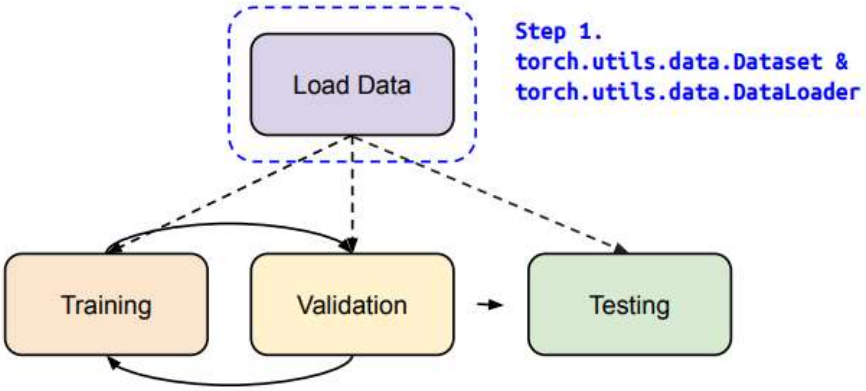
Step 2 : Loss Function

Step 3 : Optimization Algorithm

訓練與驗證會反覆操作，最後 Testing



Step 1 : 如何將資料讀進檔案 in Pytorch




```
torch.utils.data.Dataset
and
torch.utils.data.DataLoader

...

dataset = MyDataset(file)
# shuffle - Training:True、Testing : False
dataloader = DataLoader(dataset, batch_size, shuffle=True)
```

- Dataset : 將原始資料一筆一筆讀進來，用 python 的 class 打包好用來呼叫，儲存原始資料以及預測資料。
- Dataloader : 把 dataset 的一筆一筆資料，合併成各個 batch，每個 batch 裡面會有幾筆資料

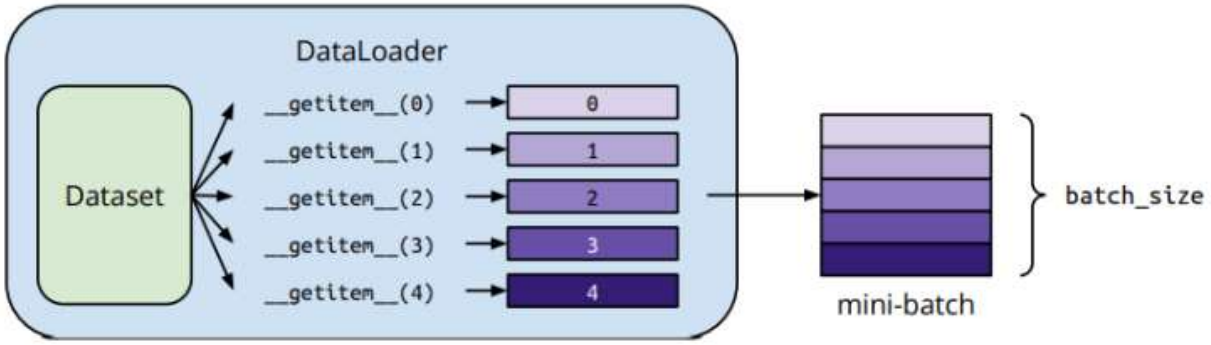
延伸參考資料 : 批次 (batch) 與動量 (momentum) (https://www.youtube.com/watch?v=zzbr1h9sF54)

```
from torch.utils.data import Dataset, DataLoader

class MyDataset(Dataset):
    # Read data & preprocess
    def __init__(self, file):
        self.data = ...

    # Returns one sample at a time
    def __getitem__(self, index):
        return self.data[index]

    # Returns the size of the dataset
    def __len__(self):
        return len(self.data)
```

呼叫 MyDataset class 產生一個 object 時，會呼叫第一個 initialization 的 function

- __ init __ : 將資料讀進來，並且去做資料的前處理
- __ getitem __ : 查看第 i 筆資料的內容
- __ len __ : 得到資料總數，得到 batch 數

```
dataset = MyDataset(file)
dataloader = Dataloader(dataset, batch_size = 5, shuffle = False)
```



- Using different data types for model and data will cause errors.

Data type	dtype	tensor
32-bit floating point	torch.float	torch.FloatTensor
64-bit integer (signed)	torch.long	torch.LongTensor

- Many functions have the same names as well

PyTorch	NumPy
x.reshape / x.view	x.reshape
x.squeeze()	x.squeeze()
x.unsqueeze(1)	np.expand_dims(x, 1)

讓 PyTorch 在顯卡上做運算

- Tensors & modules will be computed with **CPU** by default
- Use **.to()** to move tensors to appropriate devices
- CPU

```
x = x.to('cpu')
```

- GPU

```
x = x.to('cuda')
```

檢查電腦是否有 GPU

```
torch.cuda.is_available()
```

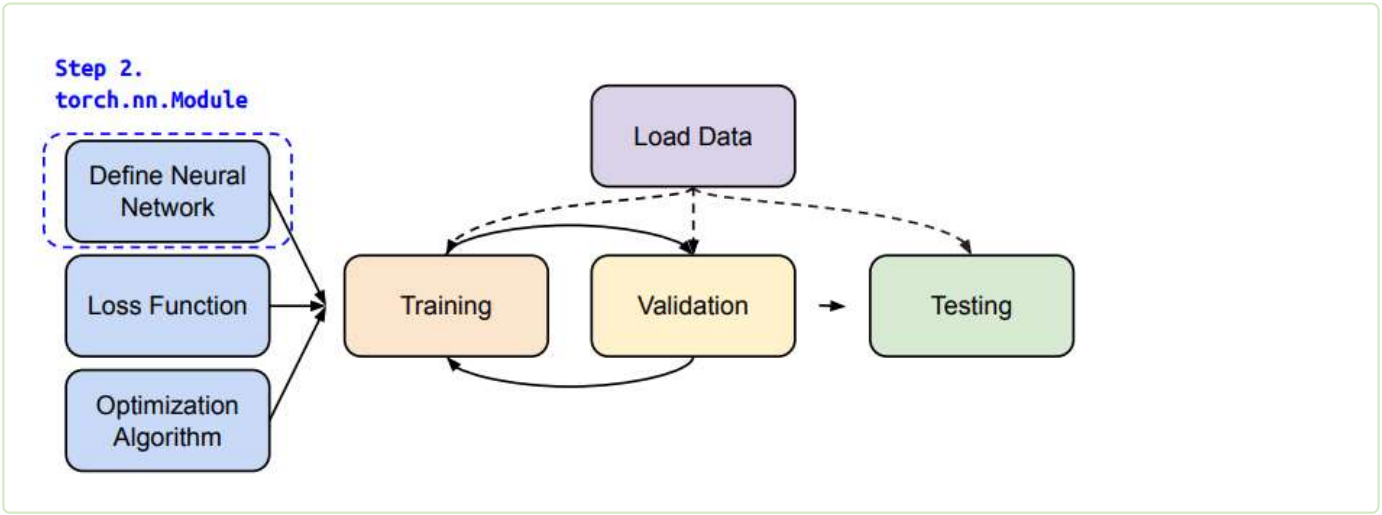
```
>>> x = torch.tensor([[1., 0.], [-1., 1.]], requires_grad=True)
>>> z = x.pow(2).sum()
>>> z.backward()
>>> x.grad
tensor([[ 2.,  0.],
        [-2.,  2.]])
```

See here to learn about gradient calculation.

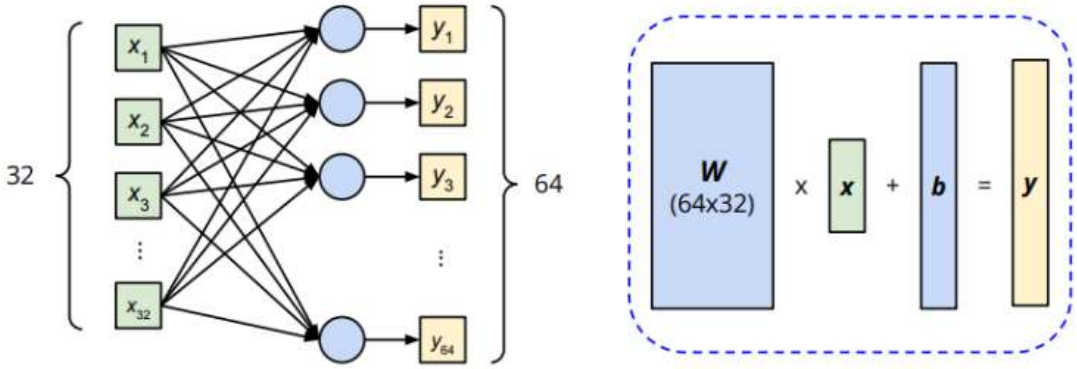
1. $x = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$
2. $z = \sum_i \sum_j x_{i,j}^2$
3. $\frac{\partial z}{\partial x_{i,j}} = 2x_{i,j}$
4. $\frac{\partial z}{\partial x} = \begin{bmatrix} 2 & 0 \\ -2 & 2 \end{bmatrix}$

延伸參考資料：如何利用梯度的計算去優化神經網路 (https://www.youtube.com/watch?v=ibJpTrp5mcE)

Step 2：如何利用 PyTorch 去定義神經網路



- Linear Layer (**Fully-connected** Layer)

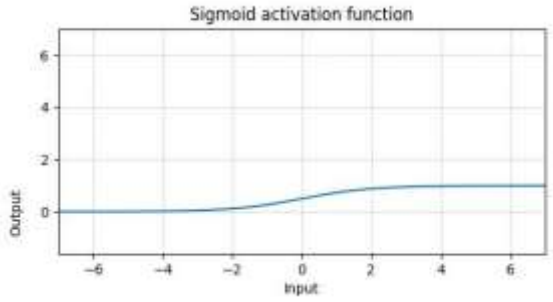



```
layer = torch.nn.Linear(32, 64)

layer.weight.shape
# torch.Size([64, 32])

layer.bias.shape
# torch.Size([64])
```

非線性函數

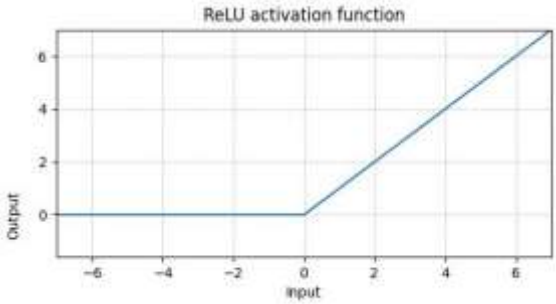
- Sigmoid Activation

```
nn.Sigmoid()
```



- ReLU Activation

```
nn.ReLU()
```



延伸參考資料：為甚麼要使用非線性函數？(https://www.youtube.com/watch?v=bHcJCp2Fyxs)

```
import torch.nn as nn

class MyModel(nn.Module):

    # Initialize your model & define layers
    def __init__(self):
        super(MyModel, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(10, 32),
            nn.Sigmoid(),
            nn.Linear(32, 1)
        )

    # Compute output of your NN
    def forward(self, x):
        return self.net(x)
```

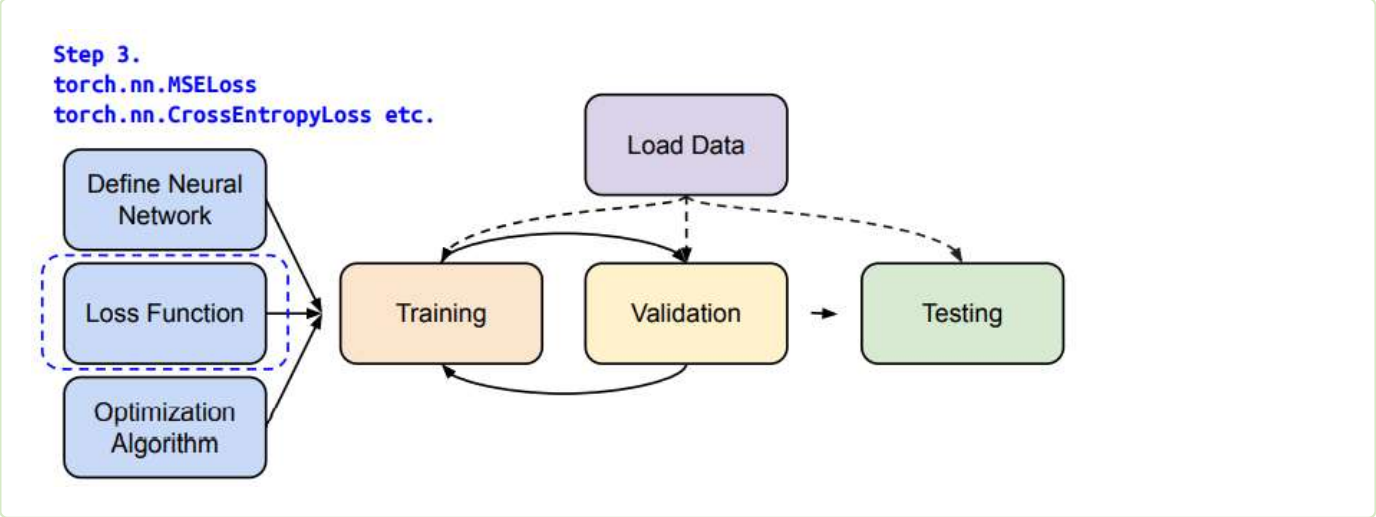
首先去 override 原本 torch 寫好的 .Module 的 class
去定義自己的模型，去 override __ **init** __ function
去定義模型是由哪幾層所組成 (這個例子定義三層 layer)

接著去定義模型要如何做運算
去 override forward function，定義模型得到一個輸入資料時，要如何做運算

也可以用以下方式寫

```
import torch.nn as nn

class MyModel(nn.Module):

    # Initialize your model & define layers
    def __init__(self):
        super(MyModel, self).__init__()
        self.layer1 = nn.Linear(10, 32),
        self.layer2 = nn.Sigmoid(),
        self.layer3 = nn.Linear(32, 1)

    # Compute output of your NN
    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        return out
```

Step 3 : 如何定義 Loss Function



常用的 Loss Function

- Mean Squared Error (for regression tasks)

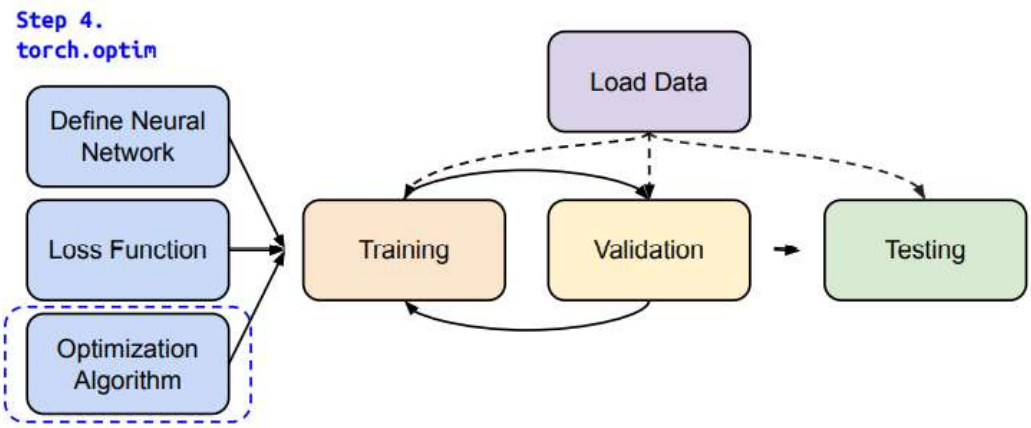
```
criterion = nn.MSELoss()
```

- Cross Entropy (for classification tasks)

```
criterion = nn.CrossEntropyLoss()
```

```
loss = criterion(model_output, expected_value)
```

Step 4 : 選擇 Optimization Algorithm



常見的優化模型演算法

演算法 : 如何用不同基於梯度下降的演算法去調整模型的參數

```
optimizer = torch.optim.SGD(model.parameters(), lr, momentum = 0)
```

For every batch of data

- reset gradients of model parameters

```
optimizer.zero_grad()
```

- backpropagate gradients of prediction loss

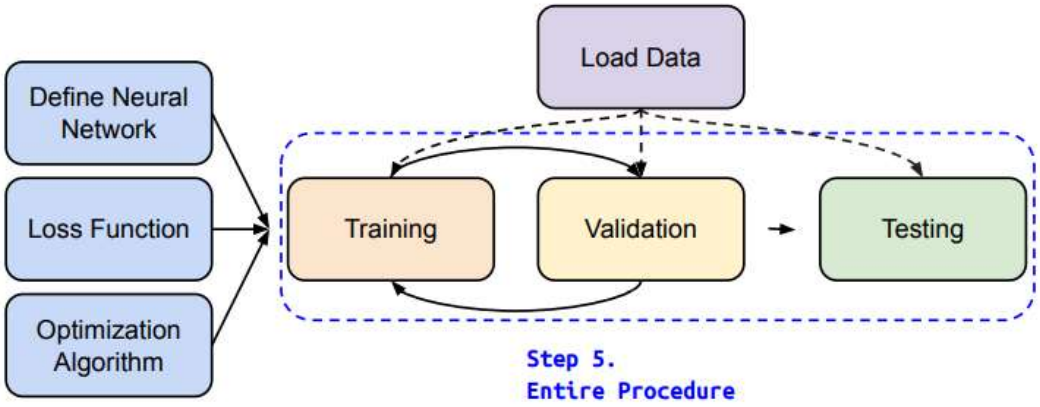
```
loss.backward()
```

- adjust model parameters

```
optimizer.step()
```

1. 把演算法前一個步驟計算的梯度所歸零
2. 把計算出來的結果，回推去算每一層的梯度
3. 根據上一個步驟所計算的梯度去調整模型的參數

Step 5：把前述的步驟連貫在一起



Neural Network Training Setup

```
# read data via MyDataset
data = MyDataset(file)

# put dataset into Dataloader
tr_set = DataLoader(dataset, 16, shuffle = True)

# construct model and move to device (cpu/cuda)
model = MyModel().to(device)

# set loss function
criterion = nn.MSELoss()

# set optimizer
optimizer = torch.optim.SGD(model.parameters(), 0.1)
```

Neural Network Training Loop

```
# iterate n_epochs
for epoch in range(n_epochs):

    # set model to train mode
    model.train()

    # iterate through the dataloader
    for x, y in tr_set :

        # set gradient to zero
        optimizer.zero_grad()

        # move data to device (cpu/cuda)
        x, y = x.to(device), y.to(device)

        # forward pass (computer output)
        pred = model(x)

        # compute loss
        loss = criterion(pred, y)

        # compute gradient (backpropagation)
        loss.backward()

        # update model with optimizer
        optimizer.step()
```

Neural Network Validation Loop

```
# set model to evaluation mode
model.eval()

total_loss = 0

# iterate through the dataloader
for x, y in dv_set :

    # move data to device (cpu/cuda)
    x, y = x.to(device), y.to(device)

    # disable gradient calculation
    with torch.no_grad():

        # forward pass (compute output)
        pred = model(x)

        # compute loss
        loss = criterion(pred, y)

    # accumulate loss
    total_loss += loss.cpu().item() * len(x)

    # compute averaged loss
    avg_loss = total_loss / len(dv_set.dataset)
```

Neural Network Testing Loop

```
# set model to evaluation mode
model.eval()

preds = []

# iterate through the dataloader
for x in tt_set :

    # move data to device (cpu/cuda)
    x = x.to(device)

    # disable gradient calculation
    with torch.no_grad :

        # forward pass (compute output)
        pred = model(x)

        # collect prediciton
        preds.append(pred.cpu())
```

- **model.eval()**

- Changes behaviour of some model layers, such as dropout and batch normalization.

- **with torch.no_grad()**
  - Prevents calculations from being added into gradient computation graph. Usually used to prevent accidental training on validation/testing data.

Save/Load Trained Models

- Save

```
torch.save(model.state_dict(), path)
```

- Load

```
ckpt = torch.load(path)
model.load_state_dict(ckpt)
```

Example of Using Pytorch

(HW1 的內容)

Task Description

- Given survey results in the past 5 days in a specific state in U.S., then predict the percentage of new tested positive cases in the 5th day.

Day1&2&3&4 **Day5**

Data

- In this case, data is included in a .csv file
- Each row represents a sample of data, containing 118 feature (id + 37 states + 16 features * 5 days)
- the last element of a row is its label

id	AL	AK	AZ	AR	CA	CO
0	0	0	0	0	0	0
1	0	0	0	0	0	1
2	0	0	0	0	0	0
3	0	0	0	0	0	0

smoothed_worried_finances	smoothed_tested_positive_14d
37.3295118	7.4561538
32.5088806	8.010957
36.7455876	2.9069774
38.6801619	12.5758159

Load data / Preprocessing

Load data: You can use **pandas** to load a csv file.

```
train_data = pd.read_csv('./covid.train.csv').drop(columns=['date']).values
```

Preprocessing: Get model inputs and labels.

```
x_train, y_train = train_data[:,:-1], train_data[:,-1]
```

```
print(x_train.shape)
print(y_train.shape)
```

```
(2699, 117)
(2699,)
```

Dataset

- __init__: Read data and preprocess
- __getitem__: Return one sample at a time. In this case, one sample includes a 117 dimensional feature and a label
- __len__: Return the size of the dataset. In this case, it is 2699

```
class COVID19Dataset(Dataset):
    '''
    x: Features.
    y: Targets, if none, do prediction.
    '''
    def __init__(self, x, y=None):
        if y is None:
            self.y = y
        else:
            self.y = torch.FloatTensor(y)
        self.x = torch.FloatTensor(x)

    def __getitem__(self, idx):
        if self.y is None:
            return self.x[idx]
        else:
            return self.x[idx], self.y[idx]

    def __len__(self):
        return len(self.x)
```

```
train_dataset = COVID19Dataset(x_train, y_train)
```

Dataloader

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, pin_memory=True)
```

- Group data into batches
- If you set **shuffle=True**, dataloader will permutes the indices of all samples automatically.
- We often set **shuffle=True** during training
- You can check this page Advantage to shuffle a dataset if you are curious about why we should shuffle the data during training

Model

- The input dimension of our model will be 117
- The output of our model will be a scalar, which represents the predicting value of the percentage of new tested positive cases in the 5th day

```
class My_model(nn.Module):
    def __init__(self, input_dim):
        super(My_model, self).__init__()
        # TODO: modify model's structure, be aware of dimension.
        self.layers = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, 1)
        )

    def forward(self, x):
        x = self.layers(x)
        x = x.squeeze(1) # (B, 1) -> (B)
        return x
```

```
model = My_Model(input_dim=x_train.shape[1]).to('cuda')
```

Criterion

We are doing a regression task, choosing mean square error as our loss function would be a good idea !

```
criterion = torch.nn.MSELoss(reduction='mean')
```

Optimizer

We need to declare a optimizer that adjust network parameters in order to reduce error.

Here we choose stochastic gradient descent as our optimization algorithm.

```
optimizer = torch.optim.SGD(model.parameters(), lr=1e-5, momentum=0.9)
```

Training loop

```
for epoch in range(3000):
      model.train() # Set your model to train mode.
      # tqdm is a package to visualize your training progress.
      train_pbar = tqdm(train_loader, position=0, leave=True)
      for x, y in train_pbar:
          x, y = x.to('cuda'), y.to('cuda')   # Move your data to device.
          pred = model(x)
          loss = criterion(pred, y)
          loss.backward()                     # Compute gradient(backpropagation).
          optimizer.step()                    # Update parameters.
          optimizer.zero_grad()               # Set gradient to zero.
```

Get model prediction, compute gradient, update parameters and reset the gradient of model parameters.

Pytorch Documentation and Common Errors

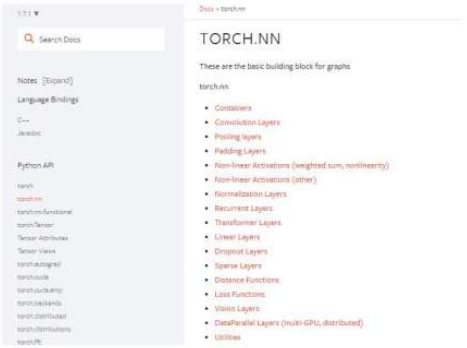
PyTorch Documentation

https://pytorch.org/docs/stable/

torch.nn -> neural network

torch.optim -> optimization algorithms

torch.utils.data -> dataset, dataloader



PyTorch Documentation Example

Function inputs and outputs

TORCH.MAX

`torch.max(input)` → Tensor

Returns the maximum value of all elements in the `input` tensor.

WARNING

This function produces deterministic (sub)gradients unlike `max(dim=0)`

Data type and explanation of each input

Parameters

input (*Tensor*) – the input tensor.

PyTorch Documentation Example

Some functions behave differently with different inputs

Parameters : You don't need to specify the name of the argument (Positional Arguments)

Keyword Arguments : You have to specify the name of the argument

They are separated by *

`torch.max(input, dim, keepdim=False, *, out=None) -> (Tensor, LongTensor)`

Returns a namedtuple `(values, indices)` where `values` is the maximum value of each row of the `input` tensor in the given dimension `dim`. And `indices` is the index location of each maximum value found (argmax).

If `keepdim` is `True`, the output tensors are of the same size as `input` except in the dimension `dim` where they are of size 1. Otherwise, `dim` is squeezed (see `torch.squeeze()`), resulting in the output tensors having 1 fewer dimension than `input`.

NOTE

If there are multiple maximal values in a reduced row then the indices of the first maximal value are returned.

Parameters

- **input** (*Tensor*) – the input tensor.
- **dim** (*int*) – the dimension to reduce.
- **keepdim** (*bool*) – whether the output tensor has `dim` retained or not. Default: `False`.

Keyword Arguments

out (*tuple, optional*) – the result tuple of two output tensors (max, max_indices)

PyTorch Documentation Example

Some functions behave differently with different inputs

Arguments with default value : Some arguments have a default value (keepdim=False), so passing a value of this argument is optional

`torch.max(input, dim, keepdim=False, *, out=None) -> (Tensor, LongTensor)`

Returns a namedtuple `(values, indices)` where `values` is the maximum value of each row of the `input` tensor in the given dimension `dim`. And `indices` is the index location of each maximum value found (argmax).

If `keepdim` is `True`, the output tensors are of the same size as `input` except in the dimension `dim` where they are of size 1. Otherwise, `dim` is squeezed (see `torch.squeeze()`), resulting in the output tensors having 1 fewer dimension than `input`.

NOTE

If there are multiple maximal values in a reduced row then the indices of the first maximal value are returned.

Parameters

- **input** (*Tensor*) – the input tensor.
- **dim** (*int*) – the dimension to reduce.
- **keepdim** (*bool*) – whether the output tensor has `dim` retained or not. Default: `False`.

Keyword Arguments

out (*tuple, optional*) – the result tuple of two output tensors (max, max_indices)

PyTorch Documentation Example

Three Kinds of torch.max

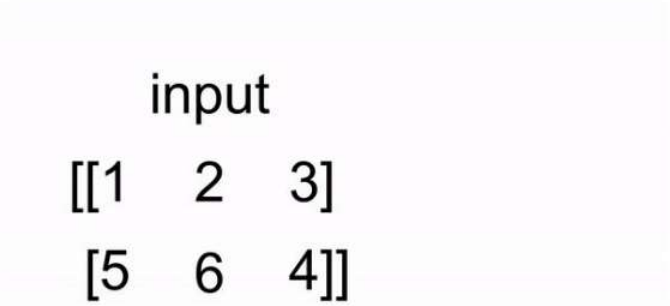
1. `torch.max(input) → Tensor`
2. `torch.max(input, dim, keepdim=False, *, out=None) → (Tensor, LongTensor)`
3. `torch.max(input, other, *, out=None) → Tensor`

```
input : Tensor, dim : int, keepdim : bool
other : Tensor
```

PyTorch Documentation Example

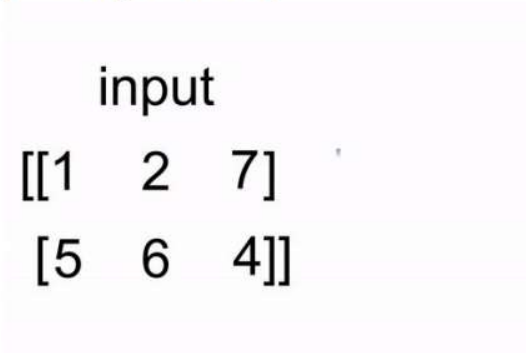
1.torch.max(input) → Tensor

Find the maximum value of a tensor, and return that value.



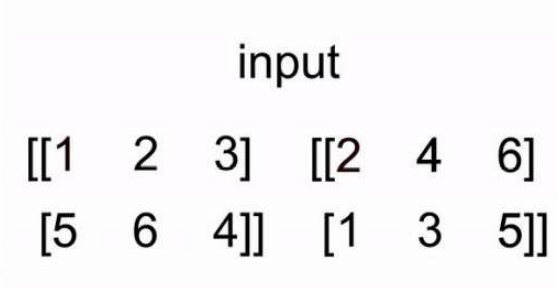
2. torch.max(input, dim, keepdim=False, *, out=None) → (Tensor, LongTensor)

Find the maximum value of a tensor along a dimension, and return that value, along with the index corresponding to that value.



3.torch.max(input, other) → Tensor

Perform element-wise comparison between two tensors of the same size, and select the maximum of the two to construct a tensor with the same size.



PyTorch Documentation Example (Colab)

Three Kinds of torch.max

1. `torch.max(input) → Tensor`
2. `torch.max(input, dim, keepdim=False, *, out=None) → (Tensor, LongTensor)`
3. `torch.max(input, other, *, out=None) → Tensor`

```
input : Tensor
dim : int
keepdim : bool
other : Tensor
```

Colab code

```
x = torch.randn(4,5)
y = torch.randn(4,5)
1.  m = torch.max(x)
2.  m, idx = torch.max(x,0)→O
    m, idx = torch.max(input = x,dim=0)→O
    m, idx = torch.max(x,0,False)→O
    m, idx = torch.max(x,0,keepdim=True)→O
    m, idx = torch.max(x,0,False,out=p)→O
    m, idx = torch.max(x,0,False,p)→x
        *out is a keyword argument
    m, idx = torch.max(x,True)→x
        *did not specify dim
3.  t = torch.max(x,y)
```

Common Errors -- Tensor on Different Device to Model

```
model = torch.nn.Linear(5,1).to("cuda:0")
x = torch.randn(5).to("cpu")
y = model(x)
```

Tensor for * is on CPU, but expected them to be on GPU

=> send the tensor to GPU

```
x = torch.randn(5).to("cuda:0")
y = model(x)
print(y.shape)
```

Common Errors -- Mismatched Dimensions

```
x = torch.randn(4,5)
y = torch.randn(5,4)
z = x + y
```

The size of tensor a (5) must match the size of tensor b (4) at non-singleton dimension 1

=> the shape of a tensor is incorrect, use **`transpose`, `squeeze`, `unsqueeze`** to align the dimensions

```
y = y.transpose(0,1)
z = x + y
print(z.shape)
```

Common Errors -- Cuda Out of Memory

```
import torch
import torchvision.models as models
resnet18 = models.resnet18().to("cuda:0") # Neural Networks for Image Recognition
data = torch.randn(512,3,244,244) # Create fake data (512 images)
out = resnet18(data.to("cuda:0")) # Use Data as Input and Feed to Model
print(out.shape)
```

CUDA out of memory. Tried to allocate 350.00 MiB (GPU 0; 14.76 GiB total capacity; 11.94 GiB already allocated; 123.75 MiB free; 13.71 GiB reserved in total by PyTorch)

=> The batch size of data is too large to fit in the GPU. Reduce the batch size.

Common Errors -- Cuda Out of Memory

If the data is iterated (batch size = 1), the problem will be solved. You can also use DataLoader

```
for d in data:
 out = resnet18(d.to("cuda:0").unsqueeze(0))
print(out.shape)
```

Common Errors -- Mismatched Tensor Type

```
import torch.nn as nn
L = nn.CrossEntropyLoss()
outs = torch.randn(5,5)
labels = torch.Tensor([1,2,3,4,0])
lossval = L(outs,labels) # Calculate CrossEntropyLoss between outs and labels
```

expected scalar type Long but found Float

=> labels must be long tensors, cast it to type “Long” to fix this issue

```
labels = labels.long()
lossval = L(outs,labels)
print(lossval)
```

課程網頁 (https://speech.ee.ntu.edu.tw/~hylee/ml/2022-spring.php)

下一篇：Lecture 2 : What to do if my network fails to train (https://hackmd.io/@_DpzRc_ARk-2NkdnL1lU2g/Sk7IaeMzq)

tags: 2022 李宏毅_機器學習