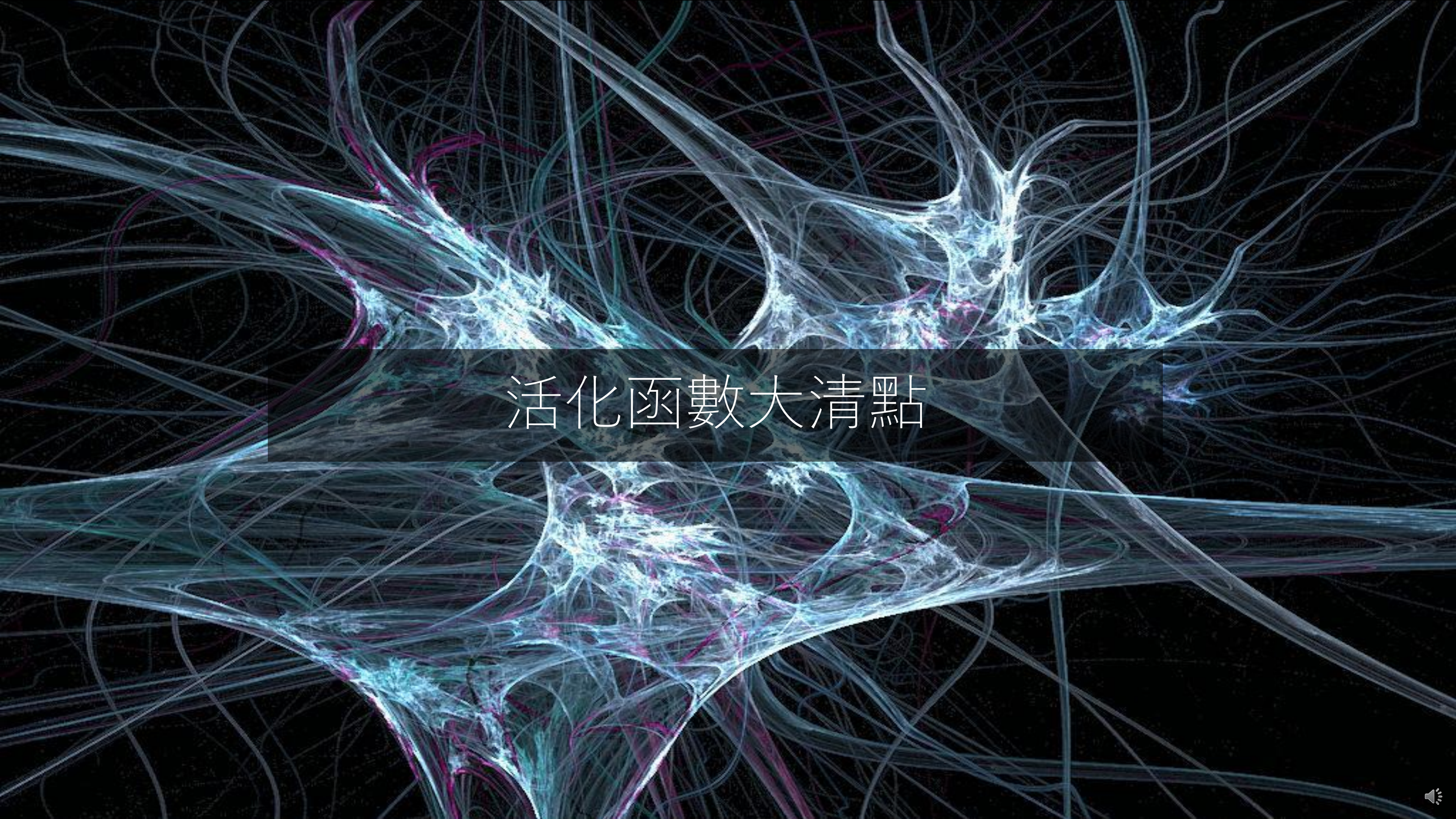

活化函數大清點

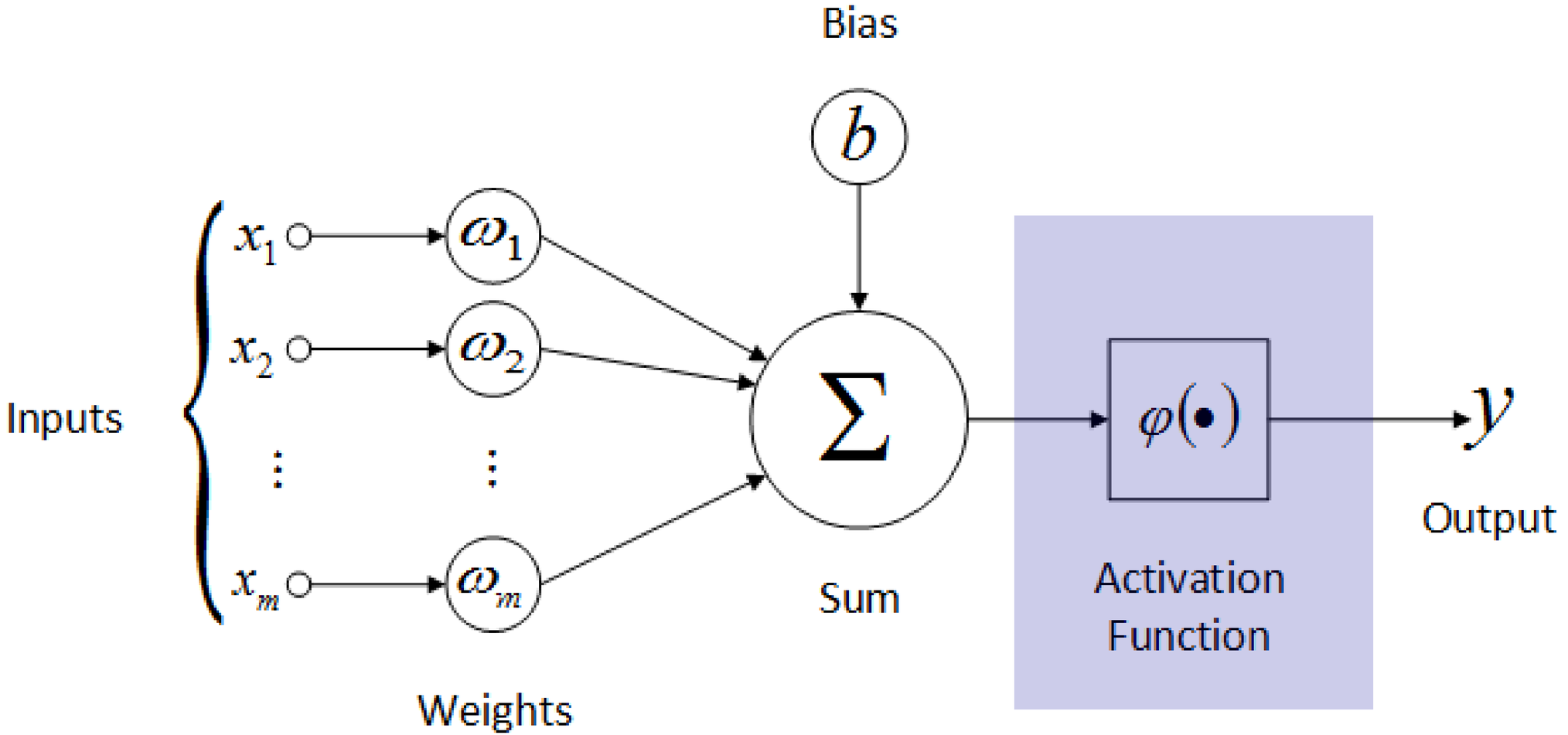
實做案例位置

github

https://github.com/AllanYiin/DeepBelief_Course5_Examples

colab

https://drive.google.com/open?id=1ppB5QOt4bE7ZprMZAZ37rSWZCpJ4DSLG


Bias
b
Inputs
x1
x2
xm
ω1
ω2
ωm
Σ
φ(•)
y
Output
Sum
Activation
Function
Weights

File Edit View Insert Cell Kernel Help Python [Root]

Code CellToolbar

In [ ]:
```
%matplotlib inline

import tkinter
import matplotlib
import platform
if platform.system() not in ['Linux', 'Darwin'] and not platform.system().startswith('CYGWIN'):
    matplotlib.use('TKAgg')
import matplotlib.pyplot as plt
from IPython import display
plt.rcParams.update({'font.size': 8})
```

活化函數大清點 (pytorch)

支援python 版本: 3.5以上
支援pytorch版本 : 1.2以上

這個實作是關於活化函數的理解。在神經元的底層結構中，活化函數扮演著將收到的信號賦予非線性特性的關鍵角色，也因此活化函數雖然不是太顯眼，但是仍有不少研究者推出新的活化函數希望可以藉由活化函數來提升模型效度。

In [ ]:
```
import gc
import glob
import os
import cv2
os.environ['TRIDENT_BACKEND'] = 'pytorch'
import trident as T
from trident import *
from trident.layers.pytorch_activations import __all__
```

在我這次為課程封裝的 API trident 中，只需要利用get_activation('活化函數名稱')即可回傳對應的活化函數，我們可以利用它結合Matplotlib 來繪製值域以及一階導數分布圖。

In [ ]:
```
fig=plt.figure(figsize=(16,16))
plt.subplots_adjust(wspace=0.4, hspace=0.6)
n=1
items= __all__[int( len(__all__)//2):-1]
plt.clf()
for k in  items :
    if k not in ('p_relu','prelu'):
        try:
            act_fn=get_activation(k)
            x =np.arange(-10, 10, 0.1).astype(np.float32)
```

File Edit View Insert Cell Kernel Help Python [Root]

Code CellToolbar

```
import trident as T
from trident import *
from trident.layers.pytorch_activations import __all__
```

在我這次為課程封裝的 API trident 中，只需要利用get_activation('活化函數名稱')即可回傳對應的活化函數，我們可以利用它結合Matplotlib 來繪製值域以及一階導數分布圖。

In [ ]:

```
fig=plt.figure(figsize=(16,16))
plt.subplots_adjust(wspace=0.4, hspace=0.6)
n=1
items= __all__[int( len(__all__)//2):-1]
plt.clf()
for k in  items :
    if k not in ('p_relu','prelu'):
        try:
            act_fn=get_activation(k)
            x =np.arange(-10, 10, 0.1).astype(np.float32)
            tensor_x=to_tensor(x)
            y=to_numpy(act_fn(tensor_x))
            ax1 = fig.add_subplot(6, 4, n)
            ax1.plot(x,y)
            ax1.plot(x[1:], np.diff(y) /(np.diff(x)+1e-8),ls=':')
            ax1.set_title(k)

        except Exception as e:
            print(e)
            pass
        n+=1
display.display(fig)
```

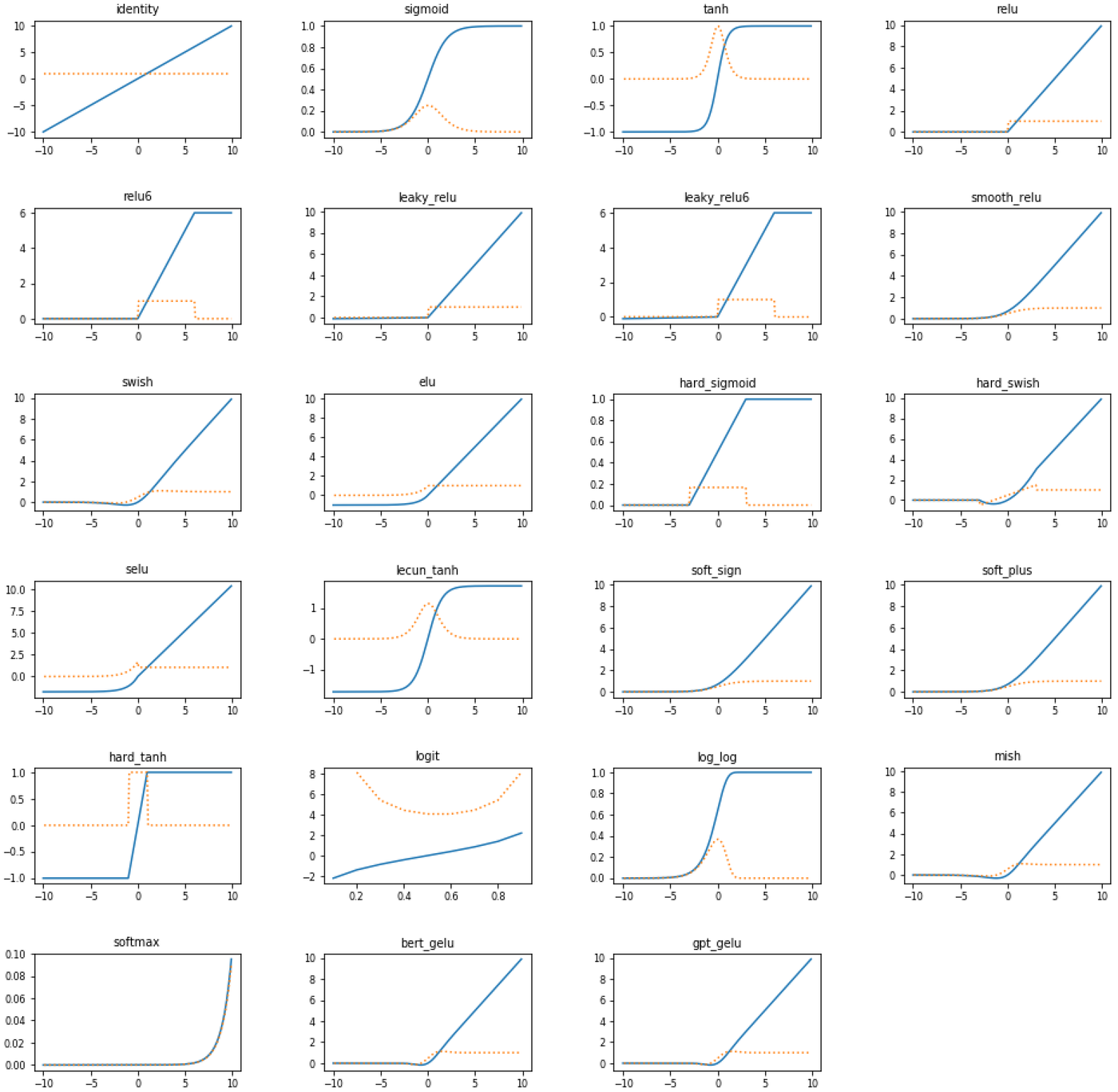
接下來我們想要評測一下幾個傳統活化函數與新的活化函數之間的效能差異，我們將從幾點來評估它:

1. 計算時長(表示計算的開銷)
2. 跑 mnist 1000 minibatch 的訓練階段最佳accuracy以及最後10個批次的accuracy
3. 使用一樣的數據進行推論
4. 使用加入大量噪聲的數據進行推論(確認跑出來模型的通用性)
5. 檢視訓練過程的梯度與權重分布
6. 梯度為零比率

下面的語法是使用trident API 讀取mnist數據集
你也可以換成T.load_mnist(' fashion-mnist','train',is_flatten=True)來改讀取fashion mnist

In [ ]:

```
dataset=T.load_mnist('mnist','train',is_flatten=True)
dataset.image_transform_funcs=[normalize(127.5,127.5)]
```


identity
sigmoid
tanh
relu
relu6
leaky_relu
leaky_relu6
smooth_relu
swish
elu
hard_sigmoid
hard_swish
selu
lecun_tanh
soft_sign
soft_plus
hard_tanh
logit
log_log
mish
softmax
bert_gelu
gpt_gelu


Q&A