3.6.4~3.6.8

p.3-92~3-106

- **3.6.4~3.6.8**
  - **3.6.4多分類交叉熵損失**
    - **one-hot encoding**
  - **3.6.5透過加權和計算交叉熵損失**
  - **3.6.6 softmax回歸的梯度計算**
    - **1.交叉商損失關於加權和的梯度**
    - **2.交叉熵損失關於權值參數的梯度**
  - **3.6.7 softmax回歸的梯度下降法實現 p.3-103**
  - **3.6.8 spiral 資料集的softmax回歸模型**
- **3.7 批次梯度下降法和隨機梯度下降法**
  - **3.7.1 MNIST手寫數字集**
  - **3.7.2 用部份訓練樣本訓練邏輯回歸模型**
  - **3.7.3 批次梯度下降法**
  - **3.7.4 隨機梯度下降法**
  - **範例**
- **4.1～4.1.3**
  - **生物上的神經網路**
  - **機器學習中的神經元**
    - **感知機**
    - **神經元**
    - **啟動函數**
  - **前饋神經網路**
    - **前向傳播**
- **4.1.4～4.1.7**
  - **1.多個樣本的正向計算**
  - **2.損失函數**
    - **1.均方差損失函數**
    - **2.二分類交叉熵損失函數**
    - **3.多分類交叉熵損失函數**
  - **3.神經網路的訓練與實作**
    - **1.製作一個兩層的神經網路，進行參數的初始化並return**

- 2.進行正向計算
- 3.計算交叉熵
- 4.製作一個丟入f和parameters就能返回的權重的函數
- 5.簡單的梯度下降優化器
- 4 2~4 2 4
  - 4.2 反向求導
    - 補充:正向求導
  - 4.2.1 正向計算反向求導
  - 4.2.2 計算圖
  - 4.2.3 損失函數關於輸出的梯度
    - 1. 二分類交叉熵損失函數關於輸出的梯度
    - 2. 均方差損失函數關於輸出的梯度
    - 3. 多分類交叉熵損失函數關於輸出的梯度
  - 4.2.4 2層神經網路的反向求導
    - 1. 單樣本的反向求導
    - 2. 反向求導的多樣本向量化表示
- 4.2.5 ~ 4.2.6

3.6.4多分類交叉熵損失

- 樣本屬於每個分類的機率：$(f_1^{(i)}, f_2^{(i)}, \ldots, f_c^{(i)})$
- 樣本屬於目標分類$y^{(i)}$的機率：$f_{y^{(i)}}^{(i)}$
- m個樣本(x^i,y^i)均以他們對應的目標分類出現的機率：$\prod_{i=1}^m f_{y^{(i)}}^{(i)}$
- 最佳參數: W 使這m個樣本有最大的正確出現的機率

> 但$\prod$容易使數值快速的趨近無限大或0
> –>改成求代價函數(上面機率的負對數的平均值)

- 代價函數：$L(W) = -\frac{1}{m}\sum_{i=1}^m \log\left(f_{y^{(i)}}^{(i)}\right)$
- 其中"$-\log\left(f_{y^{(i)}}^{(i)}\right)$"稱為交叉熵損失
- 問題從求$\prod_{i=1}^m f_{y^{(i)}}^{(i)}$最大，變成求$-\log\left(f_{y^{(i)}}^{(i)}\right)$最小的問題

舉例:

2個樣本(m=2)，對應的機率矩陣F、目標向量y如下

$F = \begin{bmatrix} 0.2 & 0.5 & 0.3 \\ 0.2 & 0.6 & 0.2 \end{bmatrix}$，$y = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$

則$F_y = \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix}$

因此其平均交叉商損失為:

$L(W) = -\frac{1}{2}(\log(0.3) + \log(0.6))$

計算程式如下

```
import numpy as np
def cross_entropy(F,y):
    m = len(F) #y.shape[0]
    log_Fy = -np.log(F[range(m),y]) #o.s. 什麼神奇的寫法
    return np.sum(log_Fy)/m

F = np.array([[0.2,0.5,0.3],[0.2,0.6,0.2]])
y = np.array([2,1])

print(cross_entropy(F,y))
```

若是用one-hot vector表示$y^{(i)}$，程式如下

```
import numpy as np
def cross_entropy_one_hot(F,y):
    m=len(F)
    return -np.sum(Y*np.log(F)) #-(1.0/m)*np.sum(np.multiply(y,np.log(F)))

F = np.array([[0.2,0.5,0.3],[0.2,0.6,0.2]])
y = np.array([[0,0,1],[0,1,0]])

print(cross_entropy_one_hot(F,y))
```

one-hot encoding

Dict	queen	king	man	woman	boy	girl
queen	1	0	0	0	0	0
king	0	1	0	0	0	0
man	0	0	1	0	0	0
woman	0	0	0	1	0	0

可能有用的補充

(https://axk51013.medium.com/%E4%B8%8D%E8%A6%81%E5%86%8D%E5%81%9Aone-hot-encoding-b5126d3f8a63)

3.6.5透過加權和計算交叉熵損失

- 一個樣本的加權和**z**的softmax函數的輸出就是機率f

```
#https://www.parasdahal.com/softmax-crossentropy
def softmax(Z):
    A = np.exp(Z-np.max(Z,axis=1,keepdims=True))
    return A/np.sum(A,axis=1,keepdims=True)

def softmax_cross_entropy(Z,y):
    m = len(Z)
    F = softmax(Z)
    log_Fy = -np.log(F[range(m),y])
    return np.sum(log_Fy)/m
```

舉例:

```
Z = np.array([[2,25,13],[54,3,11]])
y = np.array([2,1])
print(softmax_cross_entropy(Z,y))
```

若目標向量是one-hot型式:

```
def softmax(Z):
    A = np.exp(Z-np.max(Z,axis=1,keepdims=True))
    return A/np.sum(A,axis=1,keepdims=True)

def softmax_cross_entropy_one_hot(Z,y):
    F = softmax(Z)
    loss = -np.sum(y*np.log(F),axis=1)
    return np.mean(loss)
```

舉例:

```
Z = np.array([[2,25,13],[54,3,11]])
y = np.array([[0,0,1],[0,1,0]])
print(softmax_cross_entropy_one_hot(Z,y))
```

3.6.6 softmax回歸的梯度計算

- 目標: 求解使交叉熵損失$\mathcal{L}(W)$最小的W
- 方法 : 一樣是梯度下降法
- 需計算$\mathcal{L}(W)$關於W的梯度(關於W_{jk}的偏導數)

1.交叉商損失關於加權和的梯度

推導:**p.3-97**()
程式:

```
def grad_softmax_crossentropy(Z,y):
    F = softmax(Z)
    I_i = np.zeros_like(Z)
    I_i[np.arange(len(Z)),y] = 1
    return (F - I_i)/Z.shape[0]
def grad_softmax_cross_entropy(Z,y):
    m = len(Z)
    F = softmax(Z)
    F[range(m),y] -=1
    return F/m
```

舉例:

```
Z = np.array([[2,25,13],[54,3,11]])
y = np.array([2,1])
print(grad_softmax_cross_entropy(Z,y))
```

數值梯度函數的程式(用以確認分析梯度是正確的)

```
def loss_f():
    return softmax_cross_entropy(Z,y)

import util
Z = Z.astype(float)#注意:必須將整數陣列換成float型態
print("num_grad",util.numerical_gradient(loss_f,[Z]))
```

2.交叉熵損失關於權值參數的梯度

推導: **p.3-99**

程式:

X表示 資料特徵矩陣

y表示 目標特徵值向量

reg表示 正則化參數

```
def gradient_softmax(W,X,y,reg):
    m = len(X)
    Z = np.dot(X,W)
    I_i = np.zeros_like(Z)
    I_i[np.arrange(len(Z)),y] = 1
    F = softmax(Z)
    #F = np.exp(Z)/np.exp(Z).sum(axis=1,keepdoms=True)
    grad = (1/m)*np.dot(X.T,F - I_i) #Z.shape[0]
    grad = grad +2*reg*W
    return grad

def loss_softmax(W,X,y,reg):
    m = len(X)
    Z = np.dot(X,W)
    Z_i_y_i = Z[np.arrange(len(Z)),y]
    negtive_log_prob = - Z_i_y_i + np.log(np.sum(np.exp(Z),axis=1))
    loss = np.mean(negtive_log_prob)+reg*np.sum(W*W)
    return loss
```

測試一下:

```
X = np.array([[2,3],[4,5]])
y = np.array([2,1])
W = np.array([[0.1,0.2,0.3],[0.4,0.2,0.8]])

reg = 0.2

print(gradient_softmax(W,X,y,reg))
print(loss_softmax(W,X,y,reg))
```

若用one-hot表示 code在**p.3-102**

3.6.7 softmax回歸的梯度下降法實現 p.3-103

```
def gradient_descent_softmax(x,X,y,reg=0.0,alpha=0.01,iterations=100,gamma=0.8,eps
    X = np.hstack((np.ones((X.shape[0],1),dtype=X.dtype),X)) #增加一列特徵 "1"
    v= np.zeros_like(w)
    #losses = []
    w_history=[]
    for i in range(0,iterations):
        gradient = gradient_softmax(w,X,y,reg)
        if np.max(np.abs(gradient))<epsilon:
            print("gradient is small enough!")
            print("iterated num is: ",i)
            break

        w = w - (alpha*gradient)
        #v = gamma*v+alpha*gradientz
        #w= w-v
        #losses.append(loss)
        w_history.append(w)
    return w_history
```

3.6.8 spiral 資料集的softmax回歸模型

對三分類資料及spiral訓練一個softmax回歸模型:

```
X_spiral,y_spiral = gen_spiral_dataset()
X = X_spiral
y = y_spiral
alpha = 1e-0
iteration = 200
reg = 1e-3

w = np.zeros([X.shape[1]+1,len(np.unique(y))])
w_history = gradient_descent_softmax(w,X,y,reg,alpha,iterations)
w = w_history[-1]
print("w: ",w)
loss_history = compute_loss_history(w_history,X,y,reg)
print(loss_history[:-1:len(loss_history)//10])
plt.plot(loss_history,color='r')
```

計算訓練模型在一批資料(X,y)上的預測準確性

```
def getAccuracy(w,X,y):
    X = np.hstack((np.ones((X.shape[0],1)dtype=X.dtype),X)) #增加一列特徵"1"
    probs = softmax(np.dot(X,w))
    predicts = np.argmax(probs,axis=1)
    accuracy = sum(predicts ==y)/(float(len(y)))
    return accuracy
```

使用

```
getAccuracy(w,X_spiral,y_spiral)
```

繪製softmax模型的分類邊界

```
#plot the resulting classifier
h = 0.02
x_min, x_max = X[:,0].min()-1,X[:,0].max()+1
y_min, y_max = X[:,1].min()-1,X[:,1].max()+1
xx,yy = np.meshgrid(np.arrange(x_main,x_max,h), np.arrange(y_min,y_max,h))

z=np.dot(np.c_[np.ones(xx.size),xx.ravel(),yy.ravel()],w)
Z = np.argmax(Z,axis=1)
Z = Z.reshape(xx.shape)
fig = plt.figure()
plt.contourf(xx,yy,Z,camp=plt.cm.Spectral,alpha=0.3)
plt.scatter(X[:,0],X[:,1],c=y,s=40,cmap=plt.cm.Spectral)
plt.xlim(xx.min(),xx.max())
plt.ylim(yy.min(),yy.max())
#fig.savefig('spiral_linear.png')
```

3.7 批次梯度下降法和隨機梯度下降法

3.7.1 MNIST手寫數字集

1. MNIST手寫數字集是一些手寫數字的圖形，每幅圖都有一個手寫數字（0,1,2．．．,9 共10種數字）。
2. 訓練集中共有784（28 * 28）個像素點，每個像素點的值介於0~255之間。
3. 60000張訓練影像，10000張測試影像。

```
import numpy as np
import pandas as pd
from keras.utils import np_utils
np.random.seed(10)

# 匯入資料
from keras.datasets import mnist
(x_train_image,y_train_label),(x_test_image,y_test_label)=mnist.load_data()
print('train data= ',len(x_train_image))
print('test data=', len(x_test_image))

# train data=  60000
# test data= 10000
```

用以上程式便可匯入MNIST，然而通常為了避免梯度爆炸，並提升收斂速度，我們通常會將每個像素點的值除以255.0，使其變為介於0~1之間的浮點數。

3.7.2 用部份訓練樣本訓練邏輯回歸模型

MNIST的訓練集中有60000個樣本，若全部使用會耗費大量運算資源和時間，因此可以改成僅使用部分資料進行訓練，方法如下：

```
subset_size = 500  # 選擇子集大小
trainset_subset = trainset[:subset_size]   # 創建子集
# 創建新的dataloader
batch_size = 32
trainloader_subset = torch.utils.data.DataLoader(trainset_subset, batch_size=batch
```

3.7.3 批次梯度下降法

隨機抽取少量樣本對樣本進行梯度更新，一般作法如下：
1. 對原本的訓練集樣本重新排序
2. 對重新排序過的訓練集，從頭開始，按照順序取少量樣本批次計算模型函數的損失並更新模型的參數
3. 多次重覆1. 2. 兩步（1. 2. 兩步稱為一個epoch）

3.7.4 隨機梯度下降法

批次梯度下降法每次只取1個樣本即為隨機梯度下降法

範例

以MNIST進行手寫辨識並只取500個樣本訓練，採用批次梯度下降法，程式如下：

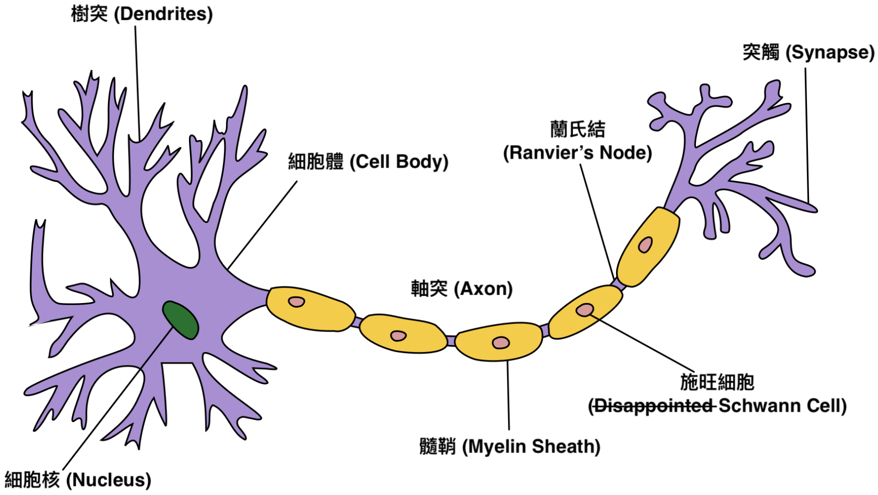
```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import matplotlib.pyplot as plt
import numpy as np

# 資料預處理和加載 MNIST 訓練數據集
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,)

# 下載完整的 MNIST 訓練數據集
trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, tr

# 要使用的子集大小（前500個樣本）
subset_size = 500

# 創建 MNIST 訓練子集
indices = np.random.choice(len(trainset), subset_size, replace=False)
trainset_subset = torch.utils.data.Subset(trainset, indices)

# 設定批次大小
batch_size = 32

# 創建 DataLoader
trainloader_subset = torch.utils.data.DataLoader(trainset_subset, batch_size=batch

# CNN 模型定義
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3)
        self.conv2 = nn.Conv2d(32, 64, 3)
        self.fc1 = nn.Linear(64 * 5 * 5, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2)
        x = x.view(x.size(0), -1)  # 攤平特徵
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# 初始化模型、損失函數和優化器
net = Net()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

# 訓練模型
num_epochs = 10
history = {'train_acc': [], 'train_loss': []}

for epoch in range(num_epochs):
    net.train()  # 設定為訓練模式
    running_loss = 0.0
    correct = 0
    total = 0

    for i, data in enumerate(trainloader_subset, 0):
        inputs, labels = data
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += labels.size(0)
        correct += predicted.eq(labels).sum().item()

    train_acc = 100. * correct / total
    train_loss = running_loss / len(trainloader_subset)

    # 將訓練準確度和損失加入 history 字典中
    history['train_acc'].append(train_acc)
```

```
    history['train_acc'].append(train_acc)
    history['train_loss'].append(train_loss)

    print(f"Epoch {epoch+1}, Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.

# 繪製學習曲線
plt.plot(range(1, num_epochs+1), history['train_acc'], label='Train Accuracy')
plt.plot(range(1, num_epochs+1), history['train_loss'], label='Train Loss')
plt.xlabel('Epoch')
plt.legend()
plt.title('Training Curve')
plt.show()
```

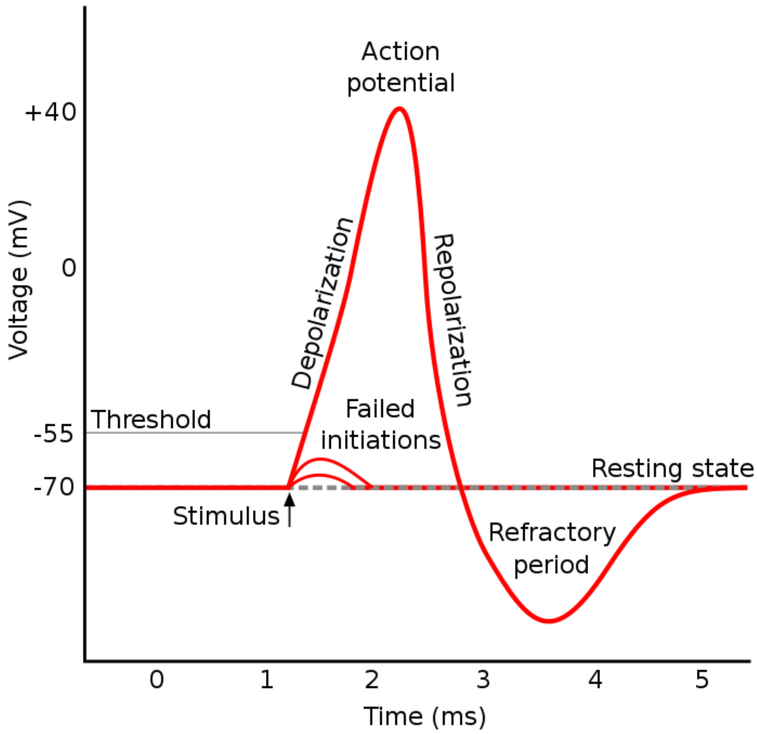
4.1 ~ 4.1.3

生物上的神經網路

以下是一個神經元的示意圖：
人類的神經網路是由非常多的神經元連接而成，一個神經元有許多樹突和一條軸突，樹突負責接受其他神經元傳出的訊號，軸突負責傳出訊號。



神經元會整合輸入的訊號，一旦細胞膜電位超過閾值(-55mV)，則神經元會產生動作電位(+40mV)，由軸突傳遞到下一個神經元的樹突。動作電位的產生，遵循「全有全無定律」，超過閾值電位會瞬間提升，反之則會回到靜止電位(-70mV)。



機器學習中的神經元

感知機

在機器學習中，科學家試著模擬出人腦中的神經網路，以達到學習的目的。因此，我們首先需要造出一個類似於神經元的東西。人造的神經元有許多類型，其中一種最基本的神經元是「**感知機**」。感知機是一個函數，計算公式如下

$$f_{\mathbf{w},b}(\mathbf{x}) = sign_b(\sum_j w_j x_j)$$

其中$\mathbf{w}$是權重，$\mathbf{x}$是神經元輸入的值，b是閾值。$sign_b()$是步階函數，表達式如下

$$sign_b(z) = \begin{cases} 1, & z \geq b \\ 0, & else \end{cases}$$

感知機整合了輸入的訊號，並用步階函數模擬人類神經元達到閾值就產生動作電位的效果。

神經元

在機器學習中，神經元有許多種，但差別只在於啟動函數。線性回歸、邏輯回歸、softmax和感知機都是神經元的一種。神經元是一個函數，它將多個輸入經過加權和，在經過一個啟動函數，輸出值。神經元的表達是如下

$$a = g(\sum_j w_j x_j)$$

其中，a是神經元的輸出，g是啟動函數。啟動函數有多種選擇，感知機即是啟動函數為步階函數的神經元。啟動函數，也可選擇tanh、ReLU、LeakReLU、simoid、softmax。

啟動函數

1. SIGMOID

函數：$f(x) = \frac{1}{1+e^{-x}}$
適用情況：
- 明確的預測，即非常接近1或0

缺點：

- 輸出值接近1或0時，梯度趨近於0，所以和sigmoid神經元連接的神經元，權重更新緩慢。若是神經網路太多sigmoid神經元達飽和，會讓神經網路無法反向傳播。
- 不以中心點為零，會導致下一個神經元產生偏置偏移，會使得收斂速度變慢。
- 計算成本高昂，因為exp()較其他函數計算成本高。

2. TANH

函數：$f(x) = tanh(x)$
優點：

- 中心點為零
- 在二分類問題，一般將tanh作為隱藏層，sigmoid作為輸出層。

缺點：

- 依然有梯度消失的問題。

3. ReLU

函數：$f(x) = max(0, x)$
優點：

- 當輸入為正時，導數為1，一定程度上改善了梯度消失問題，加速梯度下降的收斂速度。
- 計算速度快得多。ReLU 函數中只存在線性關係，因此它的計算速度比sigmoid 和tanh 更快。
- 被認為具有生物學合理性（Biological Plausibility），比如單側抑制、寬興奮邊界。

缺點：

- Dead ReLU 問題。當輸入為負時，ReLU 完全失效，在正向傳播過程中，這不是問題。有些區域很敏感，有些則不敏感。但是在反向傳播過程中，如果輸入負數，則梯度將完全為零。
- 不以零為中心，給後一層的神經網絡引入偏置偏移，會影響梯度下降的效率。

4. LeakReLU

函數：

$$f(x) = \begin{cases} x, & x > 0 \\ \gamma x, & else \end{cases}$$

優點：

- 解決ReLU函數中的梯度消失問題

5. SOFTMAX

函數：$S_i = \frac{e^{x_i}}{\sum\limits_j e^{x_j}}$

softmax神經元較特別，它沒有經過加權，是一個輸出多個值得神經元。
優點：

- Softmax訓練的深度特徵，會把整個超空間或者超球，按照分類個數進行劃分，保證類別是可分的，這一點對多分類任務如MNIST和ImageNet非常合適，因為測試類別必定在訓練類別中。

缺點：

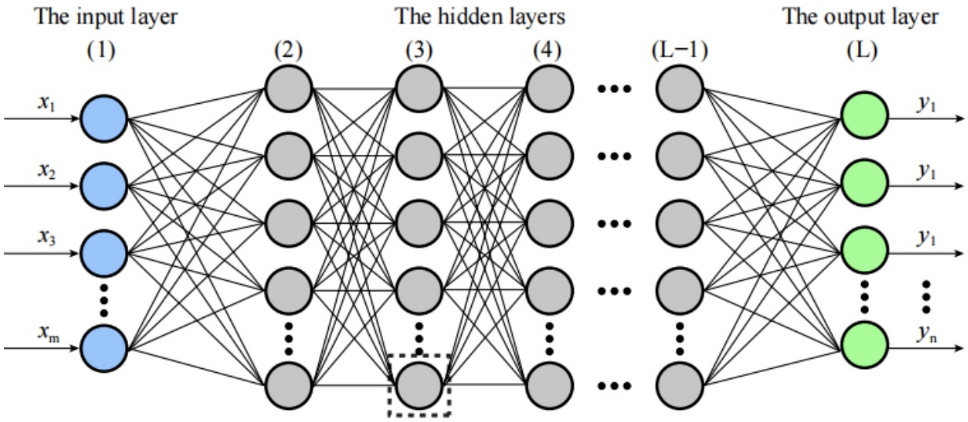
- 在零點不可微
- 負輸入的梯度為零，這意味著對於該區域的激活，權重不會在反向傳播期間更新，因此會產生永不激活的死亡神經元。
- Softmax並不要求類內緊湊和類間分離，這一點非常不適合人臉識別任務，因為訓練集的1W人數，相對測試集整個世界70億人類來說，非常微不足道，而我們不可能拿到所有人的訓練樣本，更過分的是，一般我們還要求訓練集和測試集不重疊

參考資料1 (https://zhuanlan.zhihu.com/p/364620596)、**參考資料2 (https://blog.csdn.net/xiaosongshine/article/details/88826715)**

前饋神經網路

下圖是一個前饋神經網路。前饋神經網路由輸入層、隱含層和輸出層組成。每層皆是由一列神經元組成，資訊傳遞方向，是由一層神經元傳遞至下一層神經元，不會往回傳遞。

且同層神經元沒有聯繫。



神經網路的層數稱為神經網路的深度，較深的神經網路，稱之為「**深度神經網路**」。基於深度神經網路的機器學習稱之為「**深度學習**」。

前向傳播

神經網路一層一層計算，最終輸出值的過程，稱之為「**前向傳播**」。
我們設$\mathbf{W}^{[l]}$表示第l層每個神經元的權重，$\mathbf{b}^{[l]}$為第l層每個神經元的偏置，$g^{[l]}$為第l層的啟動函數，$\mathbf{a}^{[l]}$為第l層每個神經元的輸出。前向傳播計算公式如下

$$\mathbf{a}^{[l]} = g^{[l]}(\mathbf{a}^{[l-1]}\mathbf{W}^{[l]} + \mathbf{b}^{[l]})$$

其中

$\mathbf{a}^{[l]} = \begin{bmatrix} a_1^{[l]} & a_2^{[l]} & \dots & a_p^{[l]} \end{bmatrix}$，$a_i^{[l]}$表示第$l$層第$i$個神經元的輸出

$\mathbf{W}^{[l]} = \begin{bmatrix} W_{11}^{[l]} & W_{12}^{[l]} & \dots & W_{1q}^{[l]} \\ W_{21}^{[l]} & W_{22}^{[l]} & \dots & W_{2q}^{[l]} \\ & \dots & & \\ W_{p1}^{[l]} & W_{p2}^{[l]} & \dots & W_{pq}^{[l]} \end{bmatrix}$，$W_{ij}^{[l]}$表示第$l$層第$j$個神經元，對第$(l-1)$層

$\mathbf{b}^{[l]} = \begin{bmatrix} b_1^{[l]} & b_2^{[l]} & \dots & b_q^{[l]} \end{bmatrix}$，$b_i^{[l]}$表示第$l$層第$i$個神經元的偏置

4.1.4～4.1.7

1.多個樣本的正向計算

- 前面已經舉過單個樣本的正向計算，現在把它推到多個樣本

多個樣本(m個樣本$\mathbf{x}^{(i)}$)的資料特徵可以組成一個矩陣X:

$$\mathbf{X}=\begin{bmatrix}\mathbf{x}^{(1)}\\\mathbf{x}^{(2)}\\\vdots\\\mathbf{x}^{(m)}\end{bmatrix}$$

每個樣本所對應的層輸出向量 $\mathbf{z}^{(1)[l]}$、$\mathbf{a}^{(1)[l]}$的矩陣$\mathbf{Z}^{(l)}$、$\mathbf{A}^{(l)}$可以表示成

$$\mathbf{Z}=\begin{bmatrix}\mathbf{z}^{(1)[l]}\\\mathbf{z}^{(2)[l]}\\\vdots\\\mathbf{z}^{(m)[l]}\end{bmatrix}$$

$$\mathbf{A}=\begin{bmatrix}\mathbf{a}^{(1)[l]}\\\mathbf{a}^{(2)[l]}\\\vdots\\\mathbf{a}^{(m)[l]}\end{bmatrix}$$

其中，$\mathbf{z}^{(1)[l]}$、$\mathbf{a}^{(1)[l]}$分別是第i個樣本的第l層的加權和、啟動值，他們分別作為矩陣$\mathbf{Z}^{(l)}$、$\mathbf{A}^{(l)}$的第i行，把他們展開來寫:

$$\mathbf{Z}=\begin{bmatrix}\mathbf{z}^{(1)[l]}\\\mathbf{z}^{(2)[l]}\\\vdots\\\mathbf{z}^{(m)[l]}\end{bmatrix}=\begin{bmatrix}\mathbf{a}^{(1)[l-1]}\mathbf{W}^{[l]}+\mathbf{b}^{[l]}\\\mathbf{a}^{(2)[l-1]}\mathbf{W}^{[l]}+\mathbf{b}^{[l]}\\\vdots\\\mathbf{a}^{(m)[l-1]}\mathbf{W}^{[l]}+\mathbf{b}^{[l]}\end{bmatrix}$$

可以將上式簡化成:

$$\mathbf{Z}^{[l]}=\mathbf{A}^{[l-1]}\mathbf{W}^{[l]}+\mathbf{b}^{[l]}$$

同樣的$\mathbf{A}^{(l)}$是$\mathbf{Z}^{(l)}$的啟動值:

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}^{(1)[l]} \\ \mathbf{a}^{(2)[l]} \\ \vdots \\ \mathbf{a}^{(m)[l]} \end{bmatrix} = \begin{bmatrix} \mathbf{a}^{(1)[l-1]}\mathbf{W}^{[l]} + \mathbf{b}^{[l]} \\ \mathbf{a}^{(2)[l-1]}\mathbf{W}^{[l]} + \mathbf{b}^{[l]} \\ \vdots \\ \mathbf{a}^{(m)[l-1]}\mathbf{W}^{[l]} + \mathbf{b}^{[l]} \end{bmatrix}$$

可以簡化成:

$$\mathbf{A}^{[l]} = \mathbf{g}^{[l]}(\mathbf{Z}^{[l]})$$

2.損失函數

- 損失函數:與真實值進行誤差評估(也可稱損失或者代價)

1.均方差損失函數

- 使用在一般的回歸問題
- 將所有預測值和真實值的差距平方取平均值作為誤差
- 公式:
  真實值:$F = (f^{(1)}, f^{(2)}, \cdots, f^{(m)})^T$

  預測值:$Y = (y^{(1)}, y^{(2)}, \cdots, y^{(m)})^T$

  均方差損失如下

$$L(F, Y) = \frac{1}{m}||f^{(i)} - y^{(i)}||_2^2 = \frac{1}{m}\sum_{i=1}^{m}||f^{(i)} - y^{(i)}||_2^2$$

  為了讓求導梯度更好看，會將上式除以2，亦即

$$L(F, Y) = \frac{1}{2m}||f^{(i)} - y^{(i)}||_2^2 = \frac{1}{2m}\sum_{i=1}^{m}||f^{(i)} - y^{(i)}||_2^2$$

2.二分類交叉熵損失函數

- 使用在二分類問題
- 公式:

  真實標籤:$F = (f^{(1)}, f^{(2)},\ \cdots\ ,\ f^{(m)})^T$

  預測機率:$Y = (y^{(1)}, y^{(2)}\ ,\ \cdots\ ,\ y^{(m)})^T$

$$L(f,y) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} log\,(f^{(i)}) + (1-y^{(i)})\ log\,(1-f^{(i)})]$$

$$L(\mathbf{f},\mathbf{y}) = -\frac{1}{m}\sum_{i=1}^{m}\mathbf{y}\ log\,\mathbf{f} + (1-\mathbf{y})\ log\,(1-\mathbf{f})$$

$P.S.$為了避免 $\mathbf{f}$ 或 $1-\mathbf{f}$ 出現很小的值，我們會適時的加讓一個很小的ϵ來避免 log 的值出現異常

3.多分類交叉熵損失函數

- 使用在多分類問題
- 公式:
  $f_c^{(i)}$表示第i個樣本屬於c類別的機率

  $y_c^{(i)}$用 1 or 0 表示第i個樣本是否屬於類別c (即用 **one-hot**)

  根據softmax回歸，多分類交叉熵損失函數的公式如下:

$$L(\mathbf{f},\mathbf{y}) = \frac{1}{m}\sum_{i=1}^{m}L_i(\mathbf{f}^{(i)},\mathbf{y}^{(i)})$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\mathbf{y}^{(i)}\cdot log\,(\mathbf{f}^{(i)})$$

- 舉例

  對於三分類問題，即$C=3$，某個樣本的$\mathbf{f}^{(i)}$和$\mathbf{y}^{(i)}$的值分別如下:

$$\mathbf{f}^{(i)}=\begin{bmatrix} f_1^{(i)} & f_2^{(i)} & f_3^{(i)} \end{bmatrix}=\begin{bmatrix} 0.3 & 0.5 & 0.2 \end{bmatrix}$$

$$\mathbf{y}^{(i)}=\begin{bmatrix} y_1^{(i)} & y_2^{(i)} & y_3^{(i)} \end{bmatrix}=\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

  則交叉熵損失如下:

$$-(0\times log(0.3)+0\times log(0.5)+1\times log(0.2))=-log(0.2)$$

  可以看出交叉熵損失只取決於真實的類別所對應的那一項(1那一項)

  我們也可以加入正則向避免參數過大

$$L(\mathbf{f},\mathbf{y})=\frac{1}{m}\sum_{i=1}^{m}L_i(\mathbf{f}^{(i)},\mathbf{y}^{(i)})+\lambda\sum_{i=1}^{L}||\mathbf{W}^{[l]}||_2^2$$

3.神經網路的訓練與實作

- 設神經元的數目為$n^{(l)}$，前一層的輸出值的數目是$n^{(l-1)}$，那麼該層的神經元泉質矩陣$W^{(l)}$是一個$n^{(l-1)}\times n^{(l)}$ 的矩陣

```
W1=np.random.randn(n_1_1,n_1)*0.01
```

1.製作一個兩層的神經網路，進行參數的初始化並return

```
def initialize_parameters(nx, nh, no):#只有兩層
    #nx:輸入的特徵數
    #nh:中間層的神經元數
    #no:輸出層的神經元數
    np.random.seed(2)  # 固定的種子，使每次運行這段程式時隨機數的值都是相同的
    W1 = np.random.randn(nx, nh) * 0.01
    b1 = np.zeros((1, nh))
    W2 = np.random.randn(nh, no) * 0.01
    b2 = np.zeros((1, no))

    #檢驗是否符合，否的話停止執行
    assert (W1.shape == (nx, nh))
    assert (b1.shape == (1, nh))
    assert (W2.shape == (nh, no))
    assert (b2.shape == (1, no))

    parameters = {"W1": W1, "b1": b1, "W2": W2, "b2": b2}
    return parameters

nx, nh, no = 2, 4, 3

parameters = initialize_parameters(nx, nh, no)
print("W1 = " + str(parameters["W1"]))
print("b1 = " + str(parameters["b1"]))
print("W2 = " + str(parameters["W2"]))
print("b2 = " + str(parameters["b2"]))
```

```
#print
W1 = [[-0.00416758 -0.00056267 -0.02136196  0.01640271]
 [-0.01793436 -0.00841747  0.00502881 -0.01245288]]
b1 = [[0. 0. 0. 0.]]
W2 = [[-1.05795222e-02 -9.09007615e-03  5.51454045e-03]
 [ 2.29220801e-02  4.15393930e-04 -1.11792545e-02]
 [ 5.39058321e-03 -5.96159700e-03 -1.91304965e-04]
 [ 1.17500122e-02 -7.47870949e-03  9.02525097e-05]]
b2 = [[0. 0. 0.]]
```

2.進行正向計算

- (第一層是進行$tanh(x)$，第二層是$sigmoid(x)$而這邊進行到將第二層加權過後的$\mathbf{Z}^{(2)}$輸出，沒有進行$sigmoid(x)$)

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def forward_propagation(X, parameters):
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]
    Z1 = np.dot(X, W1) + b1
    # Z1 的形狀：(3, 2) （2, 4）+(1, 4) => (3, 4)
    A1 = np.tanh(Z1)
    Z2 = np.dot(A1, W2) + b2
    # Z2 的形狀：(3, 4) (4, 3) + (1, 3) => (3, 3)
    # A2 = sigmoid(Z2)，第二個神經元，這邊選擇不做


    assert (Z2.shape == (X.shape[0], 3))
    return Z2

X = np.array([[1., 2.], [3., 4.], [5., 6.]])  # 每一行對應於一個樣本
Z2 = forward_propagation(X, parameters)
print("Z2=",Z2)
```

```
#print
Z2= [[-1.36253581e-04  4.87491807e-04 -2.47960226e-05]
 [-1.64985210e-04  1.01574088e-03 -5.99877659e-05]
 [-1.96135525e-04  1.54048069e-03 -9.36558871e-05]]
```

3.計算交叉熵

```
def softmax(Z):
    exp_Z = np.exp(Z - np.max(Z, axis=1, keepdims=True))
#減去最大的指數項，讓最大值的指數向=0
    return exp_Z / np.sum(exp_Z, axis=1, keepdims=True)
```

```
def softmax_cross_entropy(Z, Y, onehot=False):
    m = len(Z)
    F = softmax(Z)
    if onehot:
        loss = -np.sum(Y * np.log(F)) / m
    else:
        y.flatten()
        log_Fy = -np.log(F[range(m), y])
        loss = np.sum(log_Fy) / m
    return loss
```

```
def softmax_cross_entropy_reg(Z, Y, parameters, onehot=False, reg=1e-3):
    W1=parameter[0]
    W2=parameter[2]
    loss = softmax_cross_entropy(Z, Y, onehot)
    #損失
    reg_term = reg * (np.sum(W1**2) + np.sum(W2**2))
    #正則化向
    L = loss + reg_term
    assert isinstance(L, float)
    #檢查是否為浮點數
    return L
```

```
y = np.array([2, 0, 1])  # 每一行對應於一個樣本
loss = softmax_cross_entropy_reg(Z2, y, parameters)
print(loss)
```

```
#print
1.098427770814438
```

- 包起來讓我們只輸入資料和目標值就可以計算交叉熵

```
def compute_loss_reg(f, loss, X, Y, parameters, reg=1e-3):
    Z2 = f(X, parameters)
    return loss(Z2, Y, parameters, reg)

reg = 1e-3
L = compute_loss_reg(forward_propagation, softmax_cross_entropy_reg,
X, y, parameters, reg)
print(L)
```

4.製作一個丟入f和parameters就能返回的權重的函數

- 這在2.4章節實作過，但他預設是使用lambda函數，所以需要適時調整

```
def numerical_gradient(f, params, eps=1e-6):
    numerical_grads = []  # 儲存數值梯度的列表

    for x in params:
        grad = np.zeros(x.shape)  # 初始化梯度為零陣列

        # 創建用於遍歷 x 的迭代器，設定返回多維索引和可讀寫操作的標誌
        it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])

        # 簡單來說就是對每個x去算他的梯度

        # 遍歷每個元素
        while not it.finished:
            idx = it.multi_index  # 當前元素的多維索引
            old_value = x[idx]    # 儲存原始值

            # 對該元素進行微小變化，計算函數在新值下的變化
            x[idx] = old_value + eps
            fx_plus = f()  # 計算 f(x + eps)

            x[idx] = old_value - eps
            fx_minus = f()  # 計算 f(x - eps)

            # 根據數值變化計算該元素的數值偏導數
            grad[idx] = (fx_plus - fx_minus) / (2 * eps)

            x[idx] = old_value  # 恢復原始值
            it.iternext()  # 移動到下一個元素

        numerical_grads.append(grad)  # 將該參數的數值梯度加入列表

    return numerical_grads  # 返回數值梯度列表

#計算權重值
def f():
    return compute_loss_reg (forward_propagation, softmax_cross_entropy_reg, X, y,

num_grads = numerical_gradient (f, parameters)
print(num_grads[0])
print(num_grads[3])
#在這邊應該要用
#print(num_grads["W1"])我不清楚他單純是忘記前面用dict還是想表達使用list或numpy的狀況...
#print(num_grads["W2"])
```

5.簡單的梯度下降優化器

```
def max_abs(grads):#取最大的梯度
    return max(np.max(np.abs(grad)) for grad in grads)

def gradient_descent_ANN(f, X, y, parameters, reg=0.0,
alpha=0.01, iterations=100, gamma=0.8, epsilon=1e-8):
    losses = []

    for i in range(iterations):
        loss = f()  # 計算當前損失
        grads = numerical_gradient(f, parameters)  # 計算梯度

        if max_abs(grads) < epsilon:
            print("Gradient is small enough!")
            print("Number of iterations:", i)
            break

        for param, grad in zip(parameters, grads):
        #zip:把將 parameters和grads 中的對應元素分別配對
            param -= alpha * grad  # 更新模型參數

        losses.append(loss)  # 儲存當前損失值

    return parameters, losses

```

4_2~4_2_4

- **3.6.4~3.6.8**
  - **3.6.4**多分類交叉熵損失
    - **one-hot encoding**
  - **3.6.5**透過加權和計算交叉熵損失
  - **3.6.6 softmax**回歸的梯度計算
    - **1.**交叉商損失關於加權和的梯度
    - **2.**交叉熵損失關於權值參數的梯度
  - **3.6.7 softmax**回歸的梯度下降法實現 **p.3-103**
  - **3.6.8 spiral** 資料集的**softmax**回歸模型
- **3.7** 批次梯度下降法和隨機梯度下降法
  - **3.7.1 MNIST**手寫數字集
  - **3.7.2** 用部份訓練樣本訓練邏輯回歸模型
  - **3.7.3** 批次梯度下降法
  - **3.7.4** 隨機梯度下降法
  - 範例
- **4.1～4.1.3**
  - 生物上的神經網路
  - 機器學習中的神經元
    - 感知機
    - 神經元
    - 啟動函數
  - 前饋神經網路
    - 前向傳播
- **4.1.4～4.1.7**
  - **1.**多個樣本的正向計算
  - **2.**損失函數
    - **1.**均方差損失函數
    - **2.**二分類交叉熵損失函數
    - **3.**多分類交叉熵損失函數
  - **3.**神經網路的訓練與實作
    - **1.**製作一個兩層的神經網路，進行參數的初始化並**return**
    - **2.**進行正向計算

        - 3.計算交叉熵
        - 4.製作一個丟入f和parameters就能返回的權重的函數
        - 5.簡單的梯度下降優化器
- 4 2~4 2 4
    - 4.2 反向求導
        - 補充:正向求導
    - 4.2.1 正向計算反向求導
    - 4.2.2 計算圖
    - 4.2.3 損失函數關於輸出的梯度
        - 1. 二分類交叉熵損失函數關於輸出的梯度
        - 2. 均方差損失函數關於輸出的梯度
        - 3. 多分類交叉熵損失函數關於輸出的梯度
    - 4.2.4 2層神經網路的反向求導
        - 1. 單樣本的反向求導
        - 2. 反向求導的多樣本向量化表示
- 4.2.5～4.2.6

4.2 反向求導

補充:正向求導

- 對每一個參數進行微小的擾動(f′(x)=$lim_{h->0}\frac{f(x+h)-f(x)}{h}$)，以計算整個神經網路的損失
- 缺點: 當規模較大時(層數多、每層神經元多)計算的負擔太大
- eg: 2層神經網路(mnist為例)
    - sample_size=28*28pixels， batch=500， 中間層100神經元， output_layer:10神經元:
    - 參數個數:
        - 書上: 784x100+100+100*10+10=79510
        - 書上說的2層神經網路，指的是中間層+輸出層。中間層的權重W是一個784*100的矩陣，輸出值有100個。輸出層的權重是100*10的矩陣，輸出值有10個。所以總參數是
784*100+100+100*10+10=79510

- 更新一次須進行的運算次數:2*70510次 正向計算
- 僅2層就這樣，當層數變多後此方法效率過低，不可行

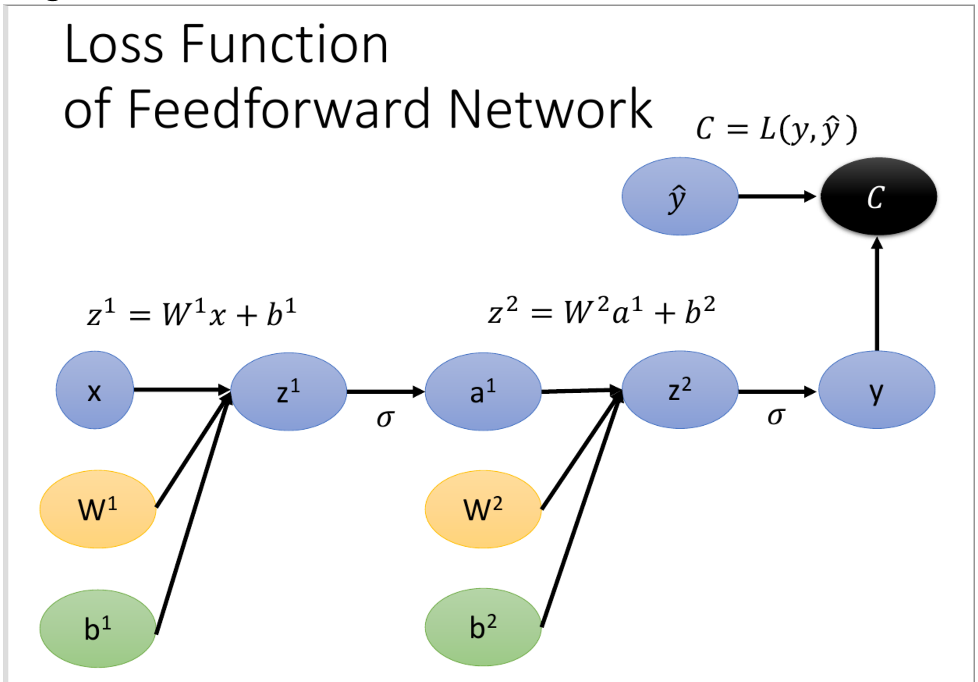
> 正向求導須對每一個參數分別進行，運算負荷過大

4.2.1 正向計算反向求導

- 將每一層視為一個function
- 正向計算(求值):
  - x -->g(x)–>f(g(x))–>k(h(g(x)))=f(x)
- 反向計算(求導):
  - f'(h) = k'(h) --> f'(g) = k'(h)h'(g) --> f'(x)=k'(h)h'(g)g'(x)
  - 對不同層進行計算時，不用一再的重複計算，效率較高

4.2.2 計算圖

把變數和函數的關係用流程圖表示
node:變數(node內可以保存其他變數)
edge: 運算動作



> — https://samaelchen.github.io/deep_learning_step2/ 2023_9_16

> 方便直覺的表示計算流程 👍

現在多數套件背後都會實做計算圖，儲存變數，方便計算

4.2.3 損失函數關於輸出的梯度

結論先說: 損失關於輸出(Z)的梯度 $\frac{1}{m}(F-Y)$

1. 二分類交叉熵損失函數關於輸出的梯度

根據二分類交叉熵損失的知識，$L(f,y) = -(y\log(f) + (1-y)\log(1-f))$ 關於 f 的導數為

$$\frac{\partial L}{\partial f} = -(\frac{y}{f} - \frac{(1-y)}{(1-f)}) = \frac{f-y}{f(1-f)}$$

$f=\sigma(z)$ 關於 z 的導數為

$$\frac{\partial f}{\partial z} = \sigma(z)(1-\sigma(z)) = f(1-f)$$

因此，$L(f,y)$ 關於 z 的導數為 $f-y$。

對於二分類問題，多個樣本的交叉熵是單一樣本的交叉熵的平均值，公式如下。

$$L(F,Y) = \frac{1}{m}\sum_{i=1}^{m} L_i\,(y^{(i)}, f^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(f^{(i)}) + (1-y^{(i)})\log(1-f^{(i)})\right]$$

因此，交叉熵損失 $L(\boldsymbol{F},\boldsymbol{Y})$ 關於 $\boldsymbol{z}$ 的梯度為

$$\frac{\partial L}{\partial \boldsymbol{F}} = \frac{1}{m}\frac{\boldsymbol{F}-\boldsymbol{Y}}{\boldsymbol{F}(1-\boldsymbol{F})}$$

$$\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{Z}} = \sigma(\boldsymbol{Z})\big(1-\sigma(\boldsymbol{Z})\big) = \boldsymbol{F}(1-\boldsymbol{F})$$

$$\frac{\partial L}{\partial \boldsymbol{Z}} = \frac{\partial L}{\partial \boldsymbol{f}}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} = \frac{1}{m}(\boldsymbol{F}-\boldsymbol{Y})$$

$$\boldsymbol{F} = \begin{bmatrix} f^{(1)} \\ \vdots \\ f^{(i)} \\ \vdots \\ f^{(m)} \end{bmatrix}, \quad \boldsymbol{Y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(i)} \\ \vdots \\ y^{(m)} \end{bmatrix}, \quad \frac{\partial \mathcal{L}}{\partial \boldsymbol{Z}} = \frac{1}{m}\begin{bmatrix} f^{(1)} - y^{(1)} \\ \vdots \\ f^{(i)} - y^{(i)} \\ \vdots \\ f^{(m)} - y^{(m)} \end{bmatrix}$$

注意：因為每個樣本都...

2. 均方差損失函數關於輸出的梯度

多樣本(batch)–>輸出向量 和 目標向量的 歐幾里德距離的平方(均方差)作為誤差

3. 多分類交叉熵損失函數關於輸出的梯度

類似二分類交叉熵

4.2.4 2層神經網路的反向求導

都是數學推導，詳見書上 p.4-48~4-54

1. 單樣本的反向求導

2. 反向求導的多樣本向量化表示

4.2.5 ~ 4.2.6
