Machine Learning HW #7 answer

1.

(i)

$q_1 = x_1 w_1 + x_2 w_2 = 2 - 1 = 1$

$q_2 = x_3 w_1 + x_4 w_2 = 1 - 1 = 0$

$h_1(q_1) = ReLU(q_1) = \max(q_1, 0) = 1$

$h_2(q_2) = ReLU(q_2) = \max(q_2, 0) = 0$

$z_1 = h_1 w_5 + h_2 w_6 = 1 + 0 = 1$

$z_2 = h_1 w_8 + h_2 w_7 = 1 + 0 = 1$

$y_1 = z_1 = 1$

$y_2 = z_2 = 1$

(ii)

$$\mathrm{J} = \frac{1}{2}\sum_{i=1}^{2}(y_i - d_i)^2$$

$$\frac{\partial J}{\partial w_1} = \frac{\partial J_1}{\partial w_1} + \frac{\partial J_2}{\partial w_1}$$

$$\frac{\partial J_1}{\partial w_1} => \frac{\partial J_1}{\partial y_1}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial h_1}\frac{\partial h_1}{\partial q_1}\frac{\partial q_1}{\partial w_1} \quad (Chain\ \ Rule)$$

$$=> (y_1 - d_1) \times 1 \times w_5 \times ReLU'(q_1) \times x_1$$

$$= (1 - 1) \times 1 \times 1 \times 1 \times 1 = 0$$

$$\frac{\partial J_2}{\partial w_1} => \frac{\partial J_2}{\partial y_2}\frac{\partial y_2}{\partial z_2}\frac{\partial z_2}{\partial h_1}\frac{\partial h_1}{\partial q_1}\frac{\partial q_1}{\partial w_1} \quad (Chain\ \ Rule)$$

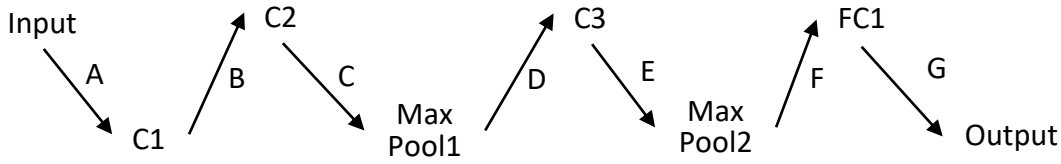
$$=> (y_2 - d_2) \times 1 \times w_8 \times ReLU'(q_1) \times x_1$$

$$= (1 - 0) \times 1 \times 1 \times 1 \times 1 = 1$$

$$\frac{\partial J}{\partial w_1} = 0 + 1 = 1$$

$$\Delta w_1 = \eta \frac{\partial J}{\partial w_1} = 0.1 \times 1 = 0.1$$

2.



A:

Connection: $(1023 \times 61) \times 16 \times (3 \times 3) = 62403 \times 16 \times 9 = 8986032$

Weights: $16 \times (3 \times 3) = 144$

B:

Connection: $(1021 \times 59) \times 16 \times (3 \times 3 \times 16) = 60239 \times 16 \times 144$
$= 138790656$

Weights: $16 \times \big(16 \times (3 \times 3)\big) = 2304$

C:

Max pool no connection & weight

D:

Connection: $(253 \times 12) \times 16 \times (3 \times 3 \times 16) = 3036 \times 16 \times 144 = 6994994$

Weights: $16 \times \big(16 \times (3 \times 3)\big) = 2304$

E:

Max pool no connection & weight

F:

Connection: $128 \times 16 \times (63 \times 3) = 128 \times 16 \times 189 = 387072$

Weights: $128 \times (16 \times (63 \times 3)) = 287072$

G:

Connection: $128 \times 2 = 256$

Weights: $128 \times 2 = 256$

number of trainable connection:

$8986032 + 138790656 + 6994994 + 387072 + 256 = 155159010$

number of trainable weights:

$144 + 2304 + 2304 + 287072 + 256 = 292080$

3.

(i)

The network has low training errors and high validation errors, there are high probability is caused by overfitting. We can decrease the parameter of 1st Full Connected Layer to improve it.

(ii)

Yes. The model of CNN is too complicated. We can add dropout layers to reduce the dependence on high-weight nodes and thus avoid overfitting.

(iii)

No. The CNN here is overfitting. Changing the loss function cross entropy to MSE cannot solve this situation.

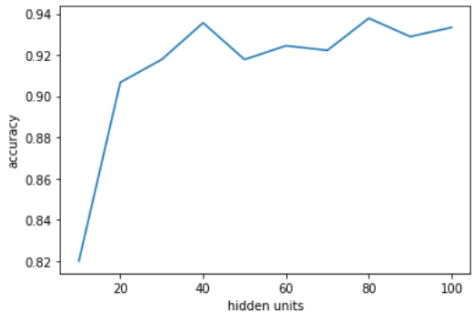
4.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
import numpy as np

iris_dataset = datasets.load_iris()

df_X = iris_dataset.data[:,:4]
df_y = iris_dataset.target
```

```
NB_CLASSES = 3
X_train, X_test, y_train, y_test = train_test_split(df_X, df_y, test_size=0.3)

# Preprocess The X Data By Scaling
sc = StandardScaler(with_mean=True, with_std=True)
sc.fit(X_train)
std_X_train = sc.transform(X_train)
std_X_test = sc.transform(X_test)

encode_y_train = np_utils.to_categorical(y_train, NB_CLASSES)
encode_y_test = np_utils.to_categorical(y_test, NB_CLASSES)
```

```
mean = []
units = range(10, 101, 10)
for unit in units:
    model = Sequential()
    model.add(Dense(units=unit, activation='relu', input_dim=4))
    model.add(Dense(units=3, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    accuracy=[]
    for i in range(10):
        model.fit(std_X_train, encode_y_train, epochs=10, batch_size=32, verbose=0)
        loss, acc = model.evaluate(std_X_test, encode_y_test, verbose=0)
        accuracy.append(acc)

    mean.append(np.mean(accuracy))
    print(f'unit: {unit} average accuracy: {np.mean(accuracy) * 100}')

plt.xlabel('hidden units')
plt.ylabel('accuracy')
plt.plot(units, mean)
plt.show()
print(f'total average accuracy: {np.mean(mean) * 100}')
```

```
unit: 10 average accuracy: 82.00000077486038
unit: 20 average accuracy: 90.66666722297668
unit: 30 average accuracy: 91.77777826786041
unit: 40 average accuracy: 93.55555593967438
unit: 50 average accuracy: 91.77777826786041
unit: 60 average accuracy: 92.44444489479065
unit: 70 average accuracy: 92.2222226858139
unit: 80 average accuracy: 93.77777814865112
unit: 90 average accuracy: 92.88888931274414
unit: 100 average accuracy: 93.33333373069763
```




```
total average accuracy: 91.44444492459296
```

5.

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten
from keras.utils import np_utils
from keras.datasets import mnist
```

```
NB_CLASSES = 10
(X_train, y_train), (X_test, y_test) = mnist.load_data()
y_train = np_utils.to_categorical(y_train, NB_CLASSES)
y_test = np_utils.to_categorical(y_test, NB_CLASSES)
model = Sequential()
model.add(Conv2D(filters=6, kernel_size=5, padding='valid', activation='tanh', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2,2), strides=2, padding='valid'))
model.add(Conv2D(filters=16, kernel_size=5, padding='valid', activation='tanh'))
model.add(MaxPooling2D(pool_size=(2,2), strides=2, padding='valid'))
model.add(Flatten())
model.add(Dense(120, activation='tanh'))
model.add(Dense(84, activation='tanh'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=32, epochs=20, verbose=1)
```

```
Epoch 1/20
1875/1875 [==============================] - 31s 16ms/step - loss: 0.1828 - accuracy: 0.9456
Epoch 2/20
1875/1875 [==============================] - 30s 16ms/step - loss: 0.0764 - accuracy: 0.9763
Epoch 3/20
1875/1875 [==============================] - 31s 17ms/step - loss: 0.0583 - accuracy: 0.9821
Epoch 4/20
1875/1875 [==============================] - 32s 17ms/step - loss: 0.0472 - accuracy: 0.9851
Epoch 5/20
1875/1875 [==============================] - 31s 16ms/step - loss: 0.0419 - accuracy: 0.9866
Epoch 6/20
1875/1875 [==============================] - 32s 17ms/step - loss: 0.0384 - accuracy: 0.9875
Epoch 7/20
1875/1875 [==============================] - 33s 18ms/step - loss: 0.0342 - accuracy: 0.9892
Epoch 8/20
1875/1875 [==============================] - 32s 17ms/step - loss: 0.0282 - accuracy: 0.9907
Epoch 9/20
1875/1875 [==============================] - 31s 17ms/step - loss: 0.0276 - accuracy: 0.9909
Epoch 10/20
1875/1875 [==============================] - 31s 17ms/step - loss: 0.0288 - accuracy: 0.9908
Epoch 11/20
1875/1875 [==============================] - 32s 17ms/step - loss: 0.0219 - accuracy: 0.9925
Epoch 12/20
1875/1875 [==============================] - 32s 17ms/step - loss: 0.0218 - accuracy: 0.9924
Epoch 13/20
1875/1875 [==============================] - 33s 18ms/step - loss: 0.0199 - accuracy: 0.9935
Epoch 14/20
1875/1875 [==============================] - 32s 17ms/step - loss: 0.0199 - accuracy: 0.9930
Epoch 15/20
1875/1875 [==============================] - 32s 17ms/step - loss: 0.0187 - accuracy: 0.9938
Epoch 16/20
1875/1875 [==============================] - 32s 17ms/step - loss: 0.0235 - accuracy: 0.9918
Epoch 17/20
1875/1875 [==============================] - 32s 17ms/step - loss: 0.0178 - accuracy: 0.9941
Epoch 18/20
1875/1875 [==============================] - 31s 17ms/step - loss: 0.0227 - accuracy: 0.9924
Epoch 19/20
1875/1875 [==============================] - 32s 17ms/step - loss: 0.0234 - accuracy: 0.9923
Epoch 20/20
1875/1875 [==============================] - 32s 17ms/step - loss: 0.0177 - accuracy: 0.9942
<keras.callbacks.History at 0x7f88ddc71490>
```

```
model.evaluate(X_test, y_test)
```

```
313/313 [==============================] - 3s 8ms/step - loss: 0.0455 - accuracy: 0.9870
[0.04545322805643082, 0.9869999885559082]
```