Homework 1. Student ID: 2019280373, Name: 费杰

1.1

1.1. $\begin{cases} x_1^2 + x_2^2 - 1 \to \min \\ x_1 + x_2 - 1 = 0 \\ 2x_2 - x_1 \le 0 \end{cases}$

$L(x_1, x_2) = \lambda_0(x_1^2 + x_2^2 - 1) + \lambda_1(x_1 + x_2 - 1) + \lambda_2(2x_2 - x_1)$

$$\begin{cases} \frac{\partial L}{\partial x_1} = 2\lambda_0 x_1 + \lambda_1 - \lambda_2 = 0 \\ \frac{\partial L}{\partial x_2} = 2\lambda_0 x_2 + \lambda_1 + 2\lambda_2 = 0 \\ x_1 + x_2 - 1 = 0 \\ \lambda_2(2x_2 - x_1) = 0 \end{cases}$$

$\lambda_0 = 0$ ↙ ; ↘ $\lambda_0 \neq 0$, let $\lambda_0 = 1/2$

$\begin{cases} \lambda_1 = \lambda_2 \\ \lambda_1 = -2\lambda_2 \\ x_1 + x_2 - 1 = 0 \\ \lambda_2(2x_2 - x_1) = 0 \end{cases} \Leftrightarrow \begin{cases} \lambda_1 = 0 \\ \lambda_2 = 0 \\ x_1 = 1 + x_2 \end{cases}$ ∅ all λ can't be 0

$\begin{cases} x_1 = \lambda_2 - \lambda_1 \\ x_2 = -\lambda_1 - 2\lambda_2 \\ -\lambda_2 - 2\lambda_1 - 1 = 0 \\ \lambda_2 \cdot (-5\lambda_2 - \lambda_1) = 0 \end{cases} \Leftrightarrow \begin{cases} \lambda_2 = 0 \\ \lambda_1 = 0 \\ \lambda_1 = -1/2 \end{cases}$ ∅

$\begin{cases} \lambda \neq 0 \\ \lambda_1 = -5\lambda_2 = -5/9 \\ \lambda_2 = 1/9 \\ x_1 = 2/3 \\ x_2 = 1/3 \end{cases}$ (1)

Check whether (1) is min:

1) $\lambda_0 = 1 \geq 0$

2) $\lambda_2 = 1/9 \geq 0$

3) $\frac{\partial^2 L}{\partial x_1^2} = 2\lambda_0 = 2 > 0$

4) $\frac{\partial^2 L}{\partial x_2^2} = 2\lambda_0 = 2 > 0$

⇓

(1) is min.

Answer: $x_1 = 2/3$

$x_2 = 1/3$.

1.2

Let E = the average number of tosses to get $H, \underbrace{T, \ldots, T}_{k}$.

Let E_H = the average number of tosses to get $H, \underbrace{T, \ldots, T}_{k}$, if the last toss was H.

Let E_T = the -11- if the last toss was T.

Let ~~E_{H,T^n}~~ E_{H,T^n} = the average number of tosses to get $H, \underbrace{T, \ldots, T}_{k}$ if the last tosses were $H, \underbrace{T, \ldots, T}_{n}$ ($E_{H,T\ldots T}$).

Since $P(H) = P(T) = 0.5$, we can say:

(0) $E = (E_T + E_H)/2$;

(1) $E_H = ((1+E_H) + E_{HT})/2$, because if the second toss is H, then we need to toss from the same situation, but the total number of tosses is 1 more; if the second toss is T, then the number of tosses in this case equals to E_{HT}.

(2) $E_T = ((1+E_H) + (1+E_T))/2$, the next toss leads to the game, where average number of tosses is 1 higher.

(3) $E_{HT} = (E_{HTT} + (2+E_H))/2$, same as before, we either start from the beginning, but with 2 more tosses in total or continue the sequence needed.

$E_{HT} = 2E_H - 1 - E_H = E_H - 1$ (from (1))

$E_T = 2 + E_H$ (from (2))

$E_{HTT} = 2E_{HT} - 2 - E_H = E_H - 4$ (from (3))

Lets prove that $E_{H\underbrace{T\ldots T}_{n}} = E_H - 2^{n+1} + n + 2$ by induction. (4)

Base: $E_{HT} = E_H - 2^{1+1} + 1 + 2 = E_H - 1$

$E_{HTT} = E_H - 2^{2+1} + 2 + 2 = E_H - 4$

Let it be correct for $m = n$, the for $m = n+1$:

$E_{H\underbrace{T\ldots T}_{n+1}}$: $E_{H\underbrace{T\ldots T}_{n}} = (E_{H\underbrace{T\ldots T}_{n+1}} + (n+1+E_H))/2$

$$E_{H\underbrace{T\ldots T}_{n+1}} = 2E_{H\underbrace{T\ldots T}_{n}} - E_H - n - 1$$

$$E_{H\underbrace{T\ldots T}_{n+1}} = 2(E_H - 2^{n+1} + n + 2) - E_H - n - 1$$

$$E_{H\underbrace{T\ldots T}_{n+1}} = E_H - 2^{n+2} + n + 3 \quad \square$$

Which proves induction hypothesis.

For the $E_{H\underbrace{T\ldots T}_{n-1}}$ we can either toss T or H: (here $n = k$)

$E_{H\underbrace{T\ldots T}_{n-1}} = ((n+1) + (n + E_H))/2$, because if toss T → get the needed sequence.

$$2E_{HT\underbrace{\cdots T}_{n-1}} = 2n+1+E_H$$

$2(E_H - 2^n + n + 1) = 2n+1+E_H$, from (4)

$E_H = 2^{n+1} - 1$

$E_T = 2 + E_H = 2^{n+1} + 1$, from previous eq.

$E = (E_H + E_T)/2 = 2^{n+1}$

Put $k = n$, get Answer: expected number of tosses is 2^{k+1}.

2. Form a Lagrange function:

$$L = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}(\xi_i + \hat{\xi}_i) - \sum_{i=1}^{N}\lambda_i(\varepsilon + \xi_i + w^T x_i + b - y_i) -$$

$$- \sum_{i=1}^{N}\lambda_i^*(\varepsilon + \hat{\xi}_i - w^T x_i - b + y_i) - \sum_{i=1}^{N}(\beta_i\xi_i + \beta_i^*\hat{\xi}_i) \quad (0)$$

The derivatives of L for the variables $w, b, \xi_i, \hat{\xi}_i$ equal 0:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{N}(\lambda_i - \lambda_i^*)x_i = 0 \iff w = \sum_{i=1}^{N}(\lambda_i - \lambda_i^*)x_i \quad (1)$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{N}(\lambda_i^* - \lambda_i) \quad (2)$$

$$\frac{\partial L}{\partial \xi_i} = C - \lambda_i - \beta_i \iff \beta_i = C - \lambda_i \quad (3)$$

$$\frac{\partial L}{\partial \hat{\xi}_i} = C - \lambda_i^* - \beta_i^* \iff \beta_i^* = C - \lambda_i^* \quad (4)$$

Put (1), (2), (3), (4) into (0):

$$\frac{1}{2}\sum_{i,j=1}^{N}(\lambda_i - \lambda_i^*)^2(x_i, x_j) + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) - \varepsilon\sum_{i=1}^{N}(\lambda_i + \lambda_i^*) -$$

$$- \sum_{i=1}^{N}(\lambda_i\xi_i + \lambda_i^*\hat{\xi}_i) - \overset{=0}{\sum_{i=1}^{N}(\lambda_i b - \lambda_i^* b)} + \sum_{i=1}^{N} y_i(\lambda_i - \lambda_i^*) -$$

$$- \overset{=0}{\sum_{i=1}^{N}(\lambda_i w^T x_i - \lambda_i^* w^T x_i)} - \sum_{i=1}^{N}(C\xi_i - \lambda_i\xi_i + C\hat{\xi}_i - \lambda_i^*\hat{\xi}_i) =$$

$$= \frac{1}{2}\sum_{i,j=1}^{N}(\lambda_i - \lambda_i^*)(x_i, x_j) - \varepsilon\sum_{i=1}^{N}(\lambda_i + \lambda_i^*) + \sum_{i=1}^{N} y_i(\lambda_i - \lambda_i^*) \quad (4)$$

Dual problem: maximize (4),

~~where $\lambda_i, \lambda_i^* \in$~~ sorry

s.t. $\sum_{i=1}^{N}(\lambda_i^* - \lambda_i) = 0$

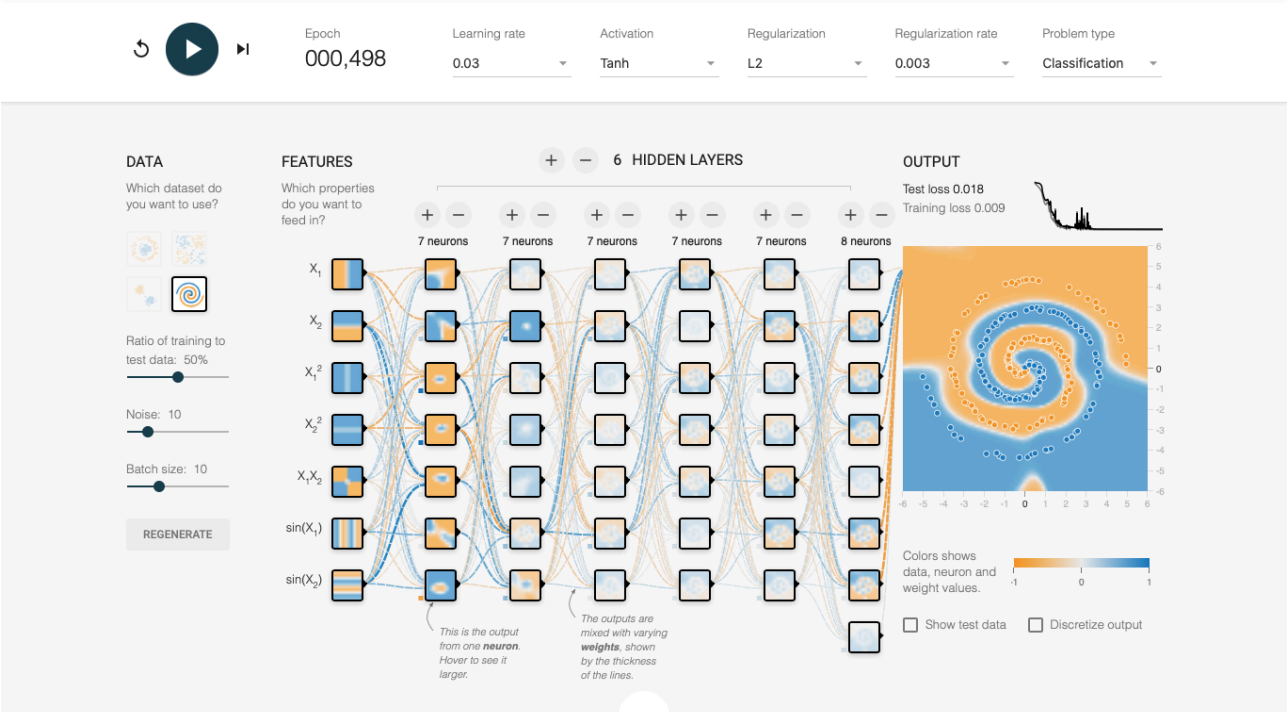
Ratio of Training to test data is 50%, Noise is 10, Batch size is 10.
The number of epochs is the one when the next epoch does not provide significant train or training loss decrease.
The learning rate must not be very small, so we can not only stay in the local extremum part, but to get out of it. Also it must not be too big, because the result of next output may differ too much.from the previous one, we can just jump to very random positions on the graph.
For the next experiment, the activation function is tested on the learning rate 0.03, Regularization L1, number of hidden layers is 6, 7 neurons on each. The number of epochs is 400, so training loss and test loss stabilise, and the epoch with the best test loss was taken.

Activation	Training loss	Test loss
ReLU	0,001	0,05
Tanh	0,001	0,02
Sigmoid	0,5	0,5
Linear	0,475	0,472

ReLU and Tanh are the best in this case, and the last showed the lowest loss. So we choose this one to find the other suitable options. Finding the best regularization:

Regularization for Tanh	Test loss
None	0,019
L1	0,02
L2	0,018

L2 showed the best result, but the difference is not that big.
For the regularization rate, big numbers won't show any good results, the best test loss was shown on the regularization rate of 0.003.
Usually more layers and more neurons mean better loss.
If the amount of neurons in the layer is too big (in this case >7, because the data generated by 7 functions), some of them (1-2 in this case) may have almost no weight at all and will not participate in the final result.
For the number of layers: the bigger is better.
The best configuration I got is shown on the screenshot



```
import numpy as np
import tensorflow as tf
from sklearn.datasets import load_svmlight_file

LAMBDA = 30
N_ITER = 10
EPS = 0.001
W_MULT = 0.005


def PrintVal(session, x_train, y_train, x_test, y_test, w, iter_num):
    print('iter', iter_num)
    print('log-likelihood:', session.run(loglikelyhood, feed_dict={x: x_train, y: y_train})[0][0])
    print('train acc:', session.run(outpp, feed_dict={x: x_train, y: y_train}))
    print('test acc:', session.run(outpp, feed_dict={x: x_test, y: y_test}))
    print('L2-norm:', np.linalg.norm(session.run(w)))

#log-likelihood including L2-norm
def LogLikeliHood(x, y, w):
    return (-tf.matmul(tf.matmul(tf.transpose(w), tf.transpose(x)), y)
            + tf.reduce_sum(tf.log(1 + tf.exp(tf.matmul(x, w))))
            + LAMBDA * tf.norm(w) / 2.0)

# iteration of irls (O(N + dim^3))


def Outp(x, y, w):
    return tf.reduce_mean(
        tf.cast(tf.equal(
            tf.squeeze(y), tf.cast(
                tf.argmax(tf.exp(
                    tf.matmul(x, w) *
                    tf.constant(np.array([0, 1]), dtype=tf.float32)) /
                          (tf.exp(tf.matmul(x, w))+ 1), axis=1), tf.float32)), tf.float32))
```

```
# from pres lect 3
def DelW(x, y, w):
    mul = tf.sigmoid(tf.matmul(x, w))
    S, U, V = tf.svd(tf.matmul(tf.transpose(x), mul * (1 - mul) * x) + LAMBDA * tf.eye(dim))
    S = tf.expand_dims(S, 1)
    InvS_p = tf.where(tf.not_equal(S, 0), 1 / S, tf.zeros_like(S))
    InvXRX_p = tf.matmul(V, InvS_p * tf.transpose(U))
    return tf.matmul(InvXRX_p, tf.matmul(tf.transpose(x), mul - y) + LAMBDA * w)

def NewW(w, wd):
    return w.assign(w - wd)



x_train, y_train = load_svmlight_file('a9a', n_features=123, dtype=np.float32)
x_test, y_test = load_svmlight_file('a9a.t', n_features=123, dtype=np.float32)
num_train = x_train.shape[0]
num_test = x_test.shape[0]
x_train = np.hstack((np.ones((num_train, 1)), x_train.toarray()))
x_test = np.hstack((np.ones((num_test, 1)), x_test.toarray()))
y_train = np.where(y_train != -1, 1, 0)
y_test = np.where(y_test != -1, 1, 0)
y_train = y_train.reshape((num_train, 1))
y_test = y_test.reshape((num_test, 1))
dim = x_train.shape[1]
x = tf.placeholder(dtype=tf.float32, shape=(None, dim), name="x")
y = tf.placeholder(dtype=tf.float32, shape=(None, 1), name="y")
w = tf.Variable(tf.fill((dim, 1), W_MULT), name='w')
wd = DelW(x, y, w)
loglikelyhood = LogLikeliHood(x, y, w)
outpp = Outp(x, y, w)
new_w = NewW(w, wd)
session = tf.Session(config=tf.ConfigProto())
session.run(tf.global_variables_initializer())
for iter_num in range(N_ITER):
    PrintVal(session, x_train, y_train, x_test, y_test, w, iter_num)
    if np.linalg.norm(session.run(wd, feed_dict={x: x_train, y: y_train})) < EPS:
        break
    session.run(new_w, feed_dict={x: x_train, y: y_train})
```