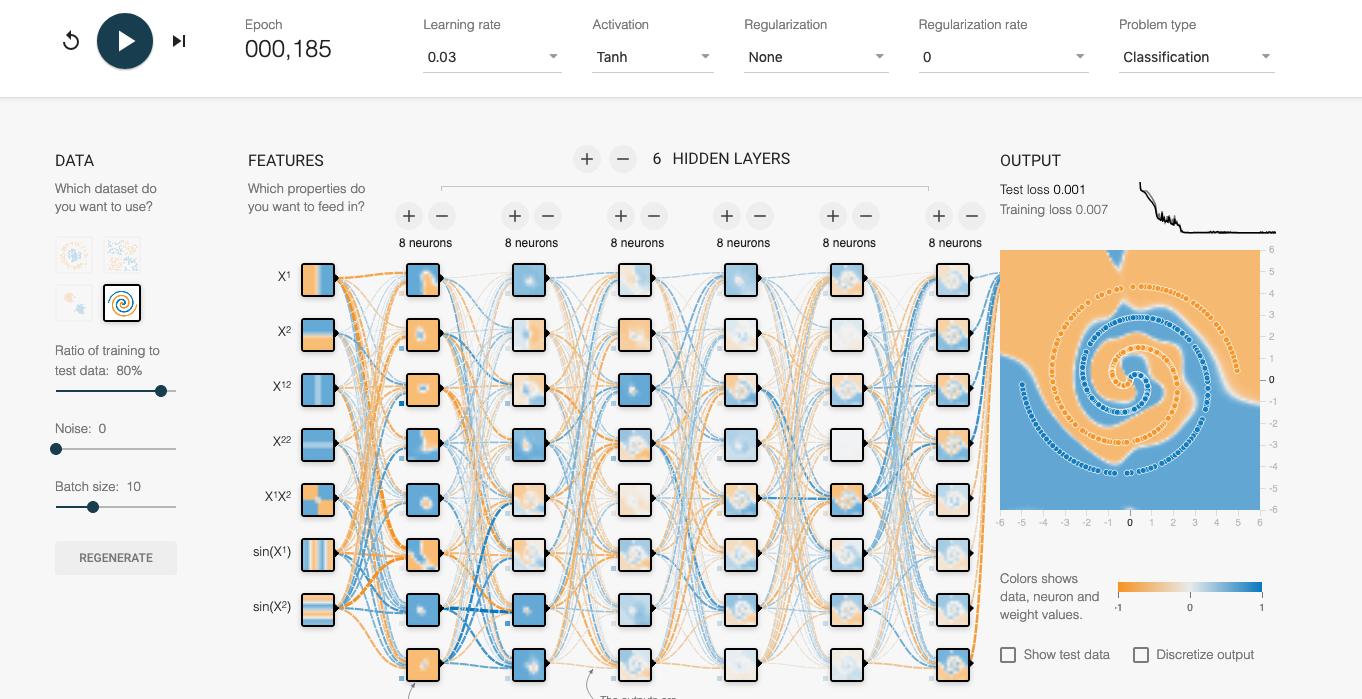


3. Deep Neural Networks



Tweaking with the Neural Network playground. I have found that the more neurons, hidden layers and features I have added, the training loss decreased each time. Also increasing training test ratio to 80% decreased the test loss. I have tried Tanh and Relu as it is good for keeping the non-linearity. It would be really hard to train with given data with linear.

4 IRLS for Logistic Regression

Code

In [46]:

**import numpy as np import tensorflow as tf**

**from sklearn.datasets import** load\_svmlight\_file

In [47]: **def** shaping\_the\_data(x\_training, x\_testing, y\_training, y\_testing):

n\_training = x\_training.shape[0] n\_testing = x\_testing.shape[0]

x\_training = np.hstack((np.ones((n\_training, 1)), x\_training.toarray())) x\_testing = np.hstack((np.ones((n\_testing, 1)), x\_testing.toarray()))

y\_training = y\_training.reshape((n\_training, 1)) y\_testing = y\_testing.reshape((n\_testing, 1))

y\_training = np.where(y\_training == -1, 0, 1)

y\_testing = np.where(y\_testing == -1, 0, 1)

**return** x\_training, x\_testing, y\_training, y\_testing

**def** log\_likelihood(w, x, y, L2\_lamda=0):

res = tf.matmul(tf.matmul(tf.transpose(w), tf.transpose(x)), y)

- tf.reduce\_sum(tf.log(1 + tf.exp(tf.matmul(x, w)))) res += -0.5 \* L2\_lamda \* tf.norm(w)

**return** -res[0][0]

**def** soft\_max\_func(x, w):

y = tf.constant(np.array([0., 1.]), dtype=tf.float32)

proba = tf.exp(tf.matmul(x, w) \* y) / (1 + tf.exp(tf.matmul(x, w)))

**return** proba

**def** score(x, y, w):

p = soft\_max\_func(x, w)

y\_pred = tf.cast(tf.argmax(p, axis=1), tf.float32) y = tf.squeeze(y)

acc = tf.reduce\_mean(tf.cast(tf.equal(y, y\_pred), tf.float32))

**return** acc

**def** optimize(w, w\_update):

**return** w.assign(w - w\_update)

**def** update(w, x, y, L2\_lamda=0):

mul = tf.sigmoid(tf.matmul(x, w)) R\_flat = mul \* (1 - mul)

dim = x.shape.as\_list()[1] L2\_reg\_term = L2\_lamda \* tf.eye(dim)

xRx = tf.matmul(tf.transpose(x), R\_flat \* x) + L2\_reg\_term S, U, V = tf.svd(xRx, full\_matrices=**True**, compute\_uv=**True**) S = tf.expand\_dims(S, 1)

S\_pinv = tf.where(tf.not\_equal(S, 0), 1 / S, tf.zeros\_like(S)) xRx\_pinv = tf.matmul(V, S\_pinv \* tf.transpose(U))

w\_update = tf.matmul(xRx\_pinv, tf.matmul(tf.transpose(x), mul - y) + L2\_lam da \* w)

**return** w\_update

In [48]: **if** name

== " main ":

x\_training, y\_training = load\_svmlight\_file("a9a", n\_features=123, dtype=n p.float32)

X\_test, y\_test = load\_svmlight\_file("a9a.t", n\_features=123, dtype=np.float

32)

x\_training, X\_test, y\_training, y\_test =shaping\_the\_data(x\_training, X\_tes

t, y\_training, y\_test) L2\_lamda = 30

N, dim = x\_training.shape

X = tf.placeholder(dtype=tf.float32, shape=(**None**, 124), name="x") y = tf.placeholder(dtype=tf.float32, shape=(**None**, 1), name="y")

w = tf.Variable(0.01 \* tf.ones((dim, 1), dtype=tf.float32), name="w") w\_update = update(w, X, y, L2\_lamda)

loss = log\_likelihood(w, X, y, L2\_lamda) acc = score(X, y, w)

optimize\_op = optimize(w, w\_update)

**se**)

config = tf.ConfigProto(allow\_soft\_placement=**True**, log\_device\_placement=**Fal**

session = tf.Session(config=config) session.run(tf.global\_variables\_initializer())

max\_iter = 100

**for** i **in** range(1, max\_iter): print("iteration: **{}**".format(i)) print("**\n**")

print("log likelihood: **{}**".format(session.run(loss, feed\_dict={X: x\_tra ining, y: y\_training})))

train\_acc = session.run(acc, feed\_dict={X: x\_training, y: y\_training}) test\_acc = session.run(acc, feed\_dict={X: X\_test, y: y\_test})

print("test data accuracy score: **{}**,**\n**training data accuracy score:

**{}**".format(test\_acc, train\_acc))

L2\_norm\_w = np.linalg.norm(session.run(w)) print("L2-norm of |w|2: **{}**".format(L2\_norm\_w)) print("**\n**==============================================================")

deri\_w = np.linalg.norm(session.run(w\_update, feed\_dict={X: x\_training, y: y\_training}))

g})

**if** deri\_w < 0.001:

**break**

w\_new = session.run(optimize\_op, feed\_dict={X: x\_training, y: y\_trainin

print("End of Iteration.")

**Results**

iteration: 1

log likelihood: -1169.1973876953125

test data accuracy score: 0.23622627556324005, training data accuracy score: 0.24080955982208252

L2-norm of |w|2: 0.11135528236627579

==============================================================

iteration: 2

log likelihood: -906.0789794921875

test data accuracy score: 0.8458325862884521, training data accuracy score: 0.8442308306694031

L2-norm of |w|2: 2.0898401737213135

==============================================================

iteration: 3

log likelihood: -1882.44189453125

test data accuracy score: 0.8495792746543884, training data accuracy score: 0.8468105792999268

L2-norm of |w|2: 3.069146156311035

==============================================================

iteration: 4

log likelihood: -2360.68408203125

test data accuracy score: 0.8506848216056824, training data accuracy score: 0.847179114818573

L2-norm of |w|2: 3.6801035404205322

==============================================================

iteration: 5

log likelihood: -2470.489501953125

test data accuracy score: 0.8509305119514465, training data accuracy score: 0.8476091027259827

L2-norm of |w|2: 3.954674482345581

==============================================================

iteration: 6

log likelihood: -2475.976318359375

test data accuracy score: 0.8511762022972107, training data accuracy score: 0.8476705551147461

L2-norm of |w|2: 4.0110764503479

==============================================================

iteration: 7

log likelihood: -2475.8369140625

test data accuracy score: 0.8511762022972107, training data accuracy score: 0.8476705551147461

L2-norm of |w|2: 4.013559341430664

==============================================================

End of Iteration.