Devi_Prasad_Happy_Monk

April 30, 2021

[135]:
```
import numpy as np
from scipy.special import softmax
from math import log2
from sklearn.metrics import f1_score, accuracy_score
from sklearn import datasets
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
```

[136]:
```
# build model without using keras api

class Model():

    def __init__(self, input_shape, output_shape, batch_size, epochs):

        self.learning_rate = 0.01
        self.output = None
        self.input_shape = input_shape
        self.output_shape = output_shape
        self.batch_size = batch_size
        self.epochs = epochs
        self.history = {"loss":[], "val_loss":[], "acc":[], "val_acc":[], "f1":
 ↪[], "val_f1":[], "K":[]}


        self.W1 = np.random.normal(size=(input_shape[1], 1))
        self.b1 = np.random.normal(size=(1, 1))

        self.W2 = np.random.normal(size=(1, output_shape[1]))
        self.b2 = np.random.normal(size=(1, output_shape[1]))

        self.K = np.random.normal(size=(2))
        self.init_K = self.K

        self.cache = None
```

```
    def activation(self, z):
        return np.add(self.K[0], np.dot(self.K[1], z))

    def cross_entropy(self, p, a):
        return (-1/len(p)) * np.sum(np.log(p) * a)

    def feed_forward(self, input):
        Z1 = np.dot(input, self.W1) + self.b1
        A1 = self.activation(Z1)
        Z2 = np.dot(A1, self.W2) + self.b2
        A2 = softmax(Z2.reshape(-1, 1), axis=0).reshape(1, 3)

        self.output = A2

        self.cache = {"Z1":Z1, "A1":A1, "Z2":Z2, "A2":A2,}

        return self.output

    def backPropagate(self, X, y):
        m = y.shape[0]

        y = y.reshape(1, 3)
        dZ2 = self.cache['A2'] - y
        dW2 = dZ2 * self.cache['A1']
        db2 = (1 / m) * np.sum(dZ2, axis = 1,)

        dA1 = np.dot(dZ2, self.W2.T)
        dZ1 = np.dot(dA1, self.K[1])
        dW1 = dZ1 * X.T
        dW1 = dW1.T

        db1 = (1 / m) * np.sum(dZ1, axis = 1)

        dK = np.array([1/m * np.sum(dA1, axis=1), 1/m * np.sum((np.dot(dA1,␣
↪self.cache['Z1'])), axis=1,)])

        self.W1 = self.W1 - self.learning_rate * dW1
        self.b1 = self.b1 - self.learning_rate * db1
        self.W2 = self.W2 - self.learning_rate * dW2
        self.b2 = self.b2 - self.learning_rate * db2
        self.K  = self.K.reshape(-1, 1) - self.learning_rate * dK
        self.K = self.K.reshape(1, -1)[0]

        self.history['K'].append(self.K)
```

```
  def f1_score(self, y_true, y_pred):

    y_true = np.argmax(y_true, axis=-1)
    y_pred = np.argmax(y_pred, axis=-1)
    return f1_score(y_true, y_pred, average='macro')



  def accuracy(self, y_true, y_pred):

    y_true = np.argmax(y_true, axis=-1)
    y_pred = np.argmax(y_pred, axis=-1)
    return accuracy_score(y_true, y_pred)



  def train(self, X_train, y_train, X_test, y_test):


      for epoch in range(self.epochs):

          for train_x, train_y in zip(X_train, y_train):

            # train data forward, loss, backward
            out = self.feed_forward(train_x)[0]
            self.backPropagate(train_x, train_y)


          train_pred = []
          for train_x, train_y in zip(X_train, y_train):

            out = self.feed_forward(train_x)[0]
            train_pred.append(out)


          test_pred = []
          for test_x, test_y in zip(X_test, y_test):

            out = self.feed_forward(test_x)[0]
            test_pred.append(out)


          # calculate train loss
          loss = self.cross_entropy(train_pred, y_train)
          self.history['loss'].append(loss)
          # calculate test loss
          loss = self.cross_entropy(test_pred, y_test)
```

```
            self.history['val_loss'].append(loss)
            # calculate train f1 score
            self.history['f1'].append(self.f1_score(y_train, train_pred))
            # calculate test f1 score
            self.history['val_f1'].append(self.f1_score(y_test, test_pred))
            # calculate train acc
            self.history['acc'].append(self.accuracy(y_train, train_pred))
            # calculate test acc
            self.history['val_acc'].append(self.accuracy(y_test, test_pred))


        return self.history, self.init_K, self.K
```

[136]:

[137]:
```
iris = datasets.load_iris()
X = iris.data  # we only take the first two features.
y = iris.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
↪random_state=42)

# scale and center
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


ohe = OneHotEncoder()
y_train_ohe = ohe.fit_transform(y_train.reshape(-1, 1)).toarray()
y_test_ohe = ohe.transform(y_test.reshape(-1, 1)).toarray()

EPOCHS = 1000

model = Model(input_shape=(1, 4), output_shape=(1, 3), batch_size=3,␣
↪epochs=EPOCHS)

HISTORY, initial_activation_parameters, final_activation_papameters = model.
↪train(X_train=X_train, y_train=y_train_ohe, X_test=X_test,␣
↪y_test=y_test_ohe,)
```

0.1 Activation parameters values

[138]:
```
initial_activation_parameters
```

[138]:
```
array([0.86647402, 1.21202402])
```

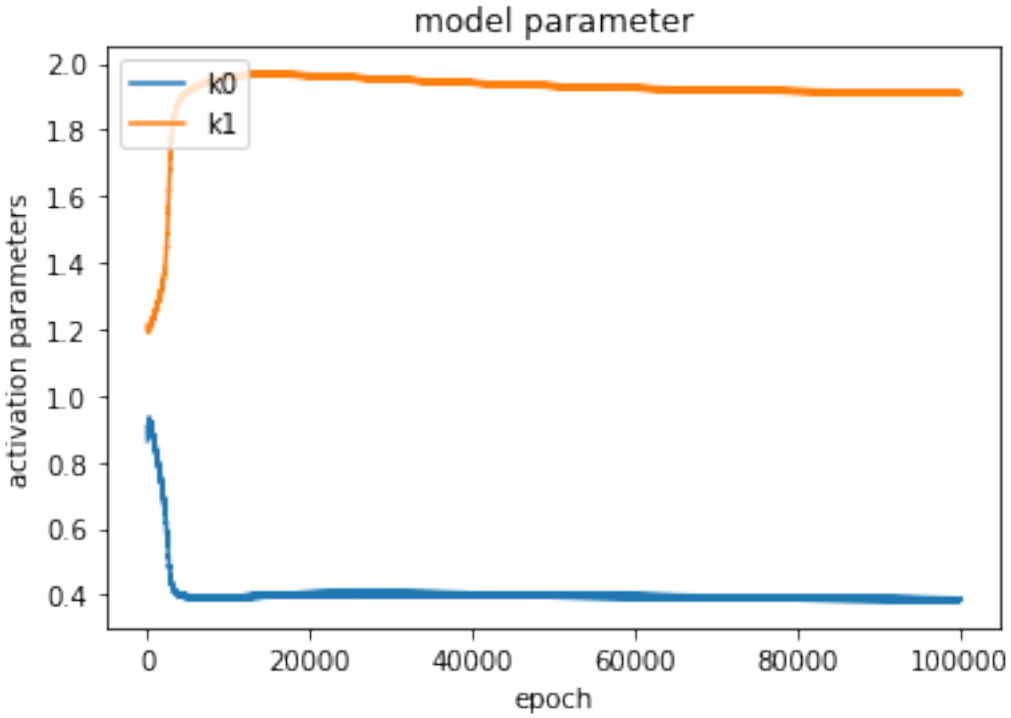
[139]:
```
final_activation_papameters
```

[139]:
```
array([0.38737734, 1.91209644])
```

1 Model parameter update

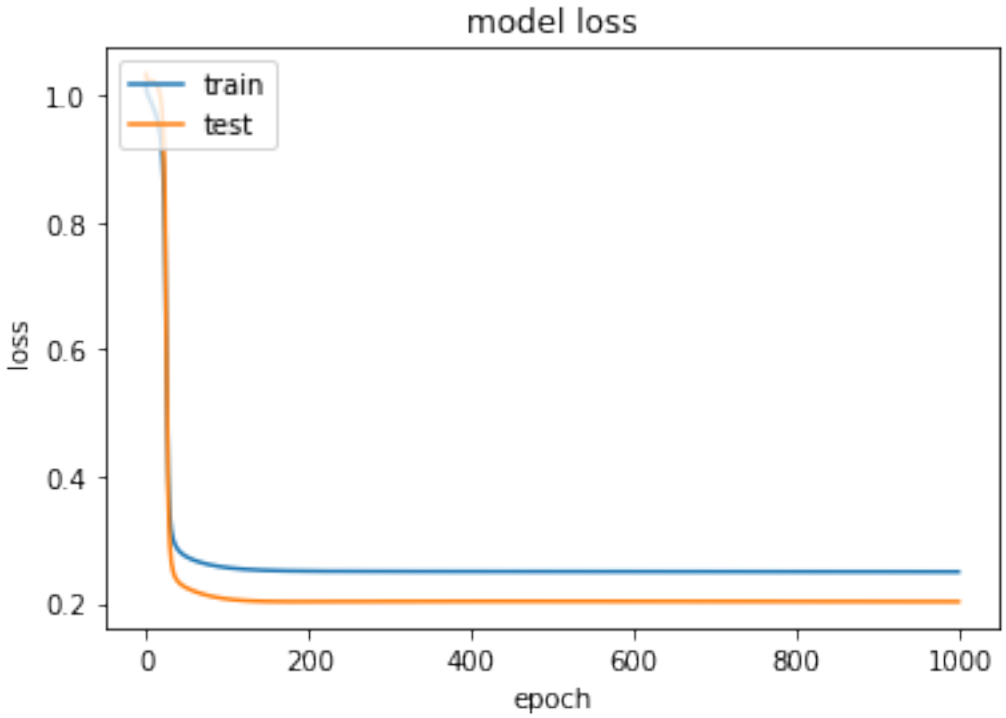
[140]:
```
k0 = []
k1 = []

for item in HISTORY['K']:
  k0.append(item[0])
  k1.append(item[1])
```

[141]:
```
plt.plot(k0)
plt.plot(k1)
plt.title('model parameter')
plt.ylabel('activation parameters')
plt.xlabel('epoch')
plt.legend(['k0', 'k1'], loc='upper left')
plt.show()
```



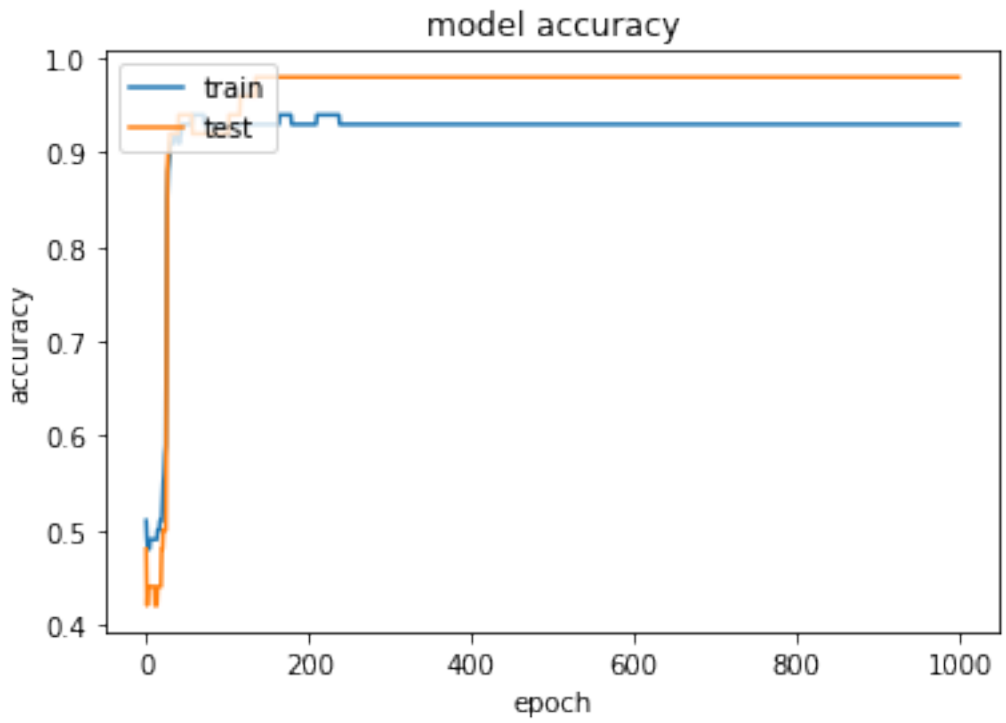
1.1 Model loss

[142]:
```
plt.plot(HISTORY['loss'])
plt.plot(HISTORY['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



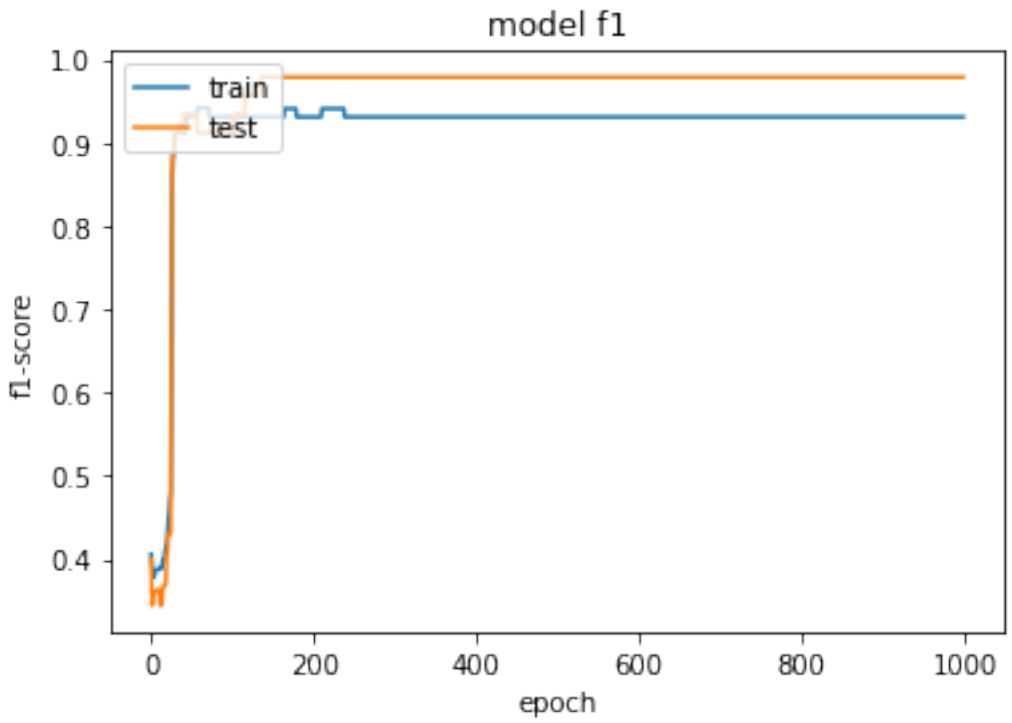
1.2 Model accuracy

[143]:
```
plt.plot(HISTORY['acc'])
plt.plot(HISTORY['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



1.3 Model f1 score

[144]:
```
plt.plot(HISTORY['f1'])
plt.plot(HISTORY['val_f1'])
plt.title('model f1')
plt.ylabel('f1-score')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



2 Github URL

https://github.com/green93/HMDevi-Prasad/blob/main/Monk_test.ipynb

3 Parameter initialization

W1,b1, W2, b2 ,k0,k1 are initialized from normal distribution

[ ]: