hw8

December 4, 2024

1 Homework 8 - Ishaan Sathaye

1.1 Part A: Concepts and Derivations

In the last homework, you created a Neural Network with the following properties: - Three sets of weights and biases for the initial layer, $\overrightarrow{w_1}$, $\overrightarrow{w_2}$, $\overrightarrow{w_3}$; c_1, c_2, c_3 - One weight set and bias for the hidden layer, $\vec{v} = v_1, v_2, v_3$, d - A Sigmoid activation function,

$$g(u) = \frac{1}{1 + e^{-u}}$$

- A loss function of squared-error on the predicted probabilities of Class 1, compared to y values of 0 or 1.

1.1.1 Question 1: Softplus

Consider instead using the Softplus activation function for the first layer only:

$$g(u) = log(1 + e^u)$$

Derive the back propagation gradient for v_1 and for $w_1 1$ with this activation function.

- first layer: $g(u) = log(1 + e^u)$
- second layer: $g(u) = \frac{1}{1+e^{-u}}$
- Gradient for v_1:
  - $\frac{\partial L}{\partial v_1} = \frac{\partial L}{\partial \hat{p_i}} \frac{\partial \hat{p_i}}{\partial u} \frac{\partial u}{\partial v_1}$
  - $\frac{\partial L}{\partial \hat{p_i}} = \frac{2}{n}(\hat{p_i} - y_i)$
  - $\frac{\partial \hat{p_i}}{\partial u} = \hat{p_i}(1 - \hat{p_i})$
  - $\frac{\partial u}{\partial v_1} = x_{1i},$ where $u = v_1 x_{1i} + v_2 x_{2i} + v_3 x_{3i} + d$
  - $\frac{\partial L}{\partial v_1} = \frac{2}{n}(\hat{p_i} - y_i)\hat{p_i}(1 - \hat{p_i})x_{1i}$
- Gradient for $w_1 1$:
  - $\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial \hat{p_i}} \frac{\partial \hat{p_i}}{\partial u} \frac{\partial u}{\partial x_{1i}} \frac{\partial x_{1i}}{\partial z_1} \frac{\partial z_1}{\partial w_{11}}$
  - $\frac{\partial u}{\partial x_{1i}} = v_1$
  - $\frac{\partial x_{1i}}{\partial z_1} = \frac{1}{1+e^{-z_1}},$ since $x_{1i} = g(z_1)$
  - $\frac{\partial z_1}{\partial w_{11}} = x_{1i}$
  - $\frac{\partial L}{\partial w_{11}} = \frac{2}{n} \sum_{i=1}^{n} (\hat{p_i} - y_i)\hat{p_i}(1 - \hat{p_i}) v_1 \frac{1}{1+e^{-z_1}} x_{1i}$

1.1.2 Question 2: ReLu

Consider instead using the ReLu activation function for the first layer only:

$$g(u) = u \text{ if } u \geq 0; 0 \text{ otherwise}$$

Derive the back propagation gradient for v_1 and for $w_1 1$ with this activation function.

- first layer: $g(u) = u$ if $u \geq 0; 0$ otherwise
- second layer: $g(u) = \frac{1}{1+e^{-u}}$
- Gradient for v_1:
  - same as above
  - $\frac{\partial L}{\partial v_1} = \frac{2}{n}(\hat{p}_i - y_i)\hat{p}_i(1-\hat{p}_i)x_{1i}$
- Gradient for $w_1 1$:
  - same as above
  - $x_{1i} = z_1$ if $z_1 \geq 0; 0$ otherwise
  - $\frac{\partial x_{1i}}{\partial z_1} = 1$ if $z_1 \geq 0; 0$ otherwise
  - $\frac{\partial z_1}{\partial w_{11}} = x_{1i}$
  - $\frac{\partial L}{\partial w_{11}} = \frac{2}{n}\sum_{i=1}^{n}(\hat{p}_i - y_i)\hat{p}_i(1-\hat{p}_i)v_1 x_1 \cdot 1$ if $z_1 \geq 0; 0$ otherwise

1.1.3 Question 3: SVC Loss

Consider instead the **loss** function

$$L = \frac{1}{n}\sum_{i=1}^{n} max(0, 1 - y_i \cdot \hat{y}_i)$$

Derive the back propagation gradient for v_1 and for $w_1 1$ with this loss function, with ReLu activation on the first layer.

- Gradient for v_1:
  - $\frac{\partial L}{\partial v_1} = \frac{\partial L}{\partial \hat{p}_i}\frac{\partial \hat{p}_i}{\partial u}\frac{\partial u}{\partial v_1}$
  - $\frac{\partial L}{\partial \hat{p}_i} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial max(0,1-y_i \cdot \hat{y}_i)}{\partial \hat{p}_i}$
  - $\frac{\partial \hat{p}_i}{\partial u} = \hat{p}_i(1-\hat{p}_i)$
  - $\frac{\partial u}{\partial v_1} = x_{1i},$ where $u = v_1 x_{1i} + v_2 x_{2i} + v_3 x_{3i} + d$
  - $\frac{\partial L}{\partial v_1} = \frac{-1}{n}\sum_{i=1}^{n}\frac{\partial max(0,1-y_i \cdot \hat{y}_i)}{\partial \hat{p}_i}\hat{p}_i(1-\hat{p}_i)x_{1i}$
- Gradient for $w_1 1$:
  - $\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial \hat{p}_i}\frac{\partial \hat{p}_i}{\partial u}\frac{\partial u}{\partial x_{1i}}\frac{\partial x_{1i}}{\partial z_1}\frac{\partial z_1}{\partial w_{11}}$
  - $\frac{\partial u}{\partial x_{1i}} = v_1$
  - $\frac{\partial x_{1i}}{\partial z_1} = 1$ if $z_1 \geq 0; 0$ otherwise
  - $\frac{\partial z_1}{\partial w_{11}} = x_{1i}$
  - $\frac{\partial L}{\partial w_{11}} = \frac{-1}{n}\sum_{i=1}^{n}\frac{\partial max(0,1-y_i \cdot \hat{y}_i)}{\partial \hat{p}_i}\hat{p}_i(1-\hat{p}_i)v_1 \cdot 1$ if $z_1 \geq 0; 0$ otherwise

1.2 Part B: Computer Implementation

1.2.1 Question 1: NN with Squared-Error Loss

Update your neural network function so that it uses a Sigmoid activation function for the first layer by default, but includes an **optional argument** allowing for either ReLu or SoftPlus activation instead. (Continue to use Sigmoid for the final layer activation.)

Use your function to fit an Indica vs Sativa model on a 80% training set from the cannabis data.

[32]:
```python
import pandas as pd

cannabis = pd.read_csv('../hw3/cannabis_full.csv')
cannabis_hybrid = cannabis[cannabis['Type'] == 'hybrid']

# remove hybrid strains
cannabis = cannabis[cannabis['Type'] != 'hybrid']

# remove effects and flavors
cannabis = cannabis.drop(columns=['Effects', 'Flavor'])

# map type to 0 or 1
cannabis['Type'] = cannabis['Type'].map({'indica': 0, 'sativa': 1})

# drop null values
cannabis = cannabis.dropna()

from sklearn.linear_model import LogisticRegression

effect_cols = cannabis.columns[cannabis.columns.get_loc("Creative"):cannabis.
 ↪columns.get_loc("Mouth")+1]
flavors_cols = cannabis.columns[cannabis.columns.get_loc("Earthy"):cannabis.
 ↪columns.get_loc("Pear")+1]

# Logistic Regression using only effect predictors
m1 = LogisticRegression()
m1.fit(cannabis[effect_cols], cannabis['Type'])

# Logistic Regression using only flavor predictors
m2 = LogisticRegression()
m2.fit(cannabis[flavors_cols], cannabis['Type'])

# Logistic Regression using only rating predictor
m3 = LogisticRegression()
m3.fit(cannabis[['Rating']], cannabis['Type'])

import numpy as np

w1 = np.concatenate([m1.coef_[0], np.zeros(len(flavors_cols) + 1)])
w2 = np.concatenate([np.zeros(len(effect_cols)), m2.coef_[0], np.zeros(1)])
w3 = np.concatenate([np.zeros(len(effect_cols) + len(flavors_cols)), m3.
 ↪coef_[0]])

c1 = m1.intercept_[0]
```

```
c2 = m2.intercept_[0]
c3 = m3.intercept_[0]
```

[33]:
```
def loss(pred, target):
    return (pred - target) ** 2

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

def sigmoid_der(x):
    return sigmoid(x) * (1 - sigmoid(x))

def relu(x):
    return max(0, x)

def relu_der(x):
    return 1 if x > 0 else 0

def softplus(x):
    return np.log(1 + np.exp(x))

def softplus_der(x):
    return np.exp(x) / (1 + np.exp(x))

def check_stopping_condition_all(v, d, w1, w2, w3, c1, c2, c3, threshold=1e-4):
    return (np.linalg.norm(v) < threshold and np.abs(d) < threshold and
        np.linalg.norm(w1) < threshold and np.linalg.norm(w2) < threshold
        and np.linalg.norm(w3) < threshold and np.abs(c1) < threshold and
        np.abs(c2) < threshold and np.abs(c3) < threshold)
```

[34]:
```
def gd_perceptron_all(train_pred, train_target, w1, w2, w3, c1, c2, c3,␣
 ↪learning_rate=0.01, epochs=500,
                       activation='sigmoid'):
    # initialize v and d
    v = np.array([1/3, 1/3, 1/3])
    d = 0

    # set the activation function
    if activation == 'sigmoid':
        act_func = sigmoid
        act_der = sigmoid_der
    elif activation == 'relu':
        act_func = relu
        act_der = relu_der
    elif activation == 'softplus':
        act_func = softplus
        act_der = softplus_der
```

```
for epoch in range(1, epochs+1):

    total_loss = 0

    for preds, target in zip(train_pred, train_target):
        preds = np.array(preds)

        # forward pass
        z1 = np.dot(preds, w1) + c1
        z2 = np.dot(preds, w2) + c2
        z3 = np.dot(preds, w3) + c3
        u1 = act_func(z1)
        u2 = act_func(z2)
        u3 = act_func(z3)
        u = np.array([u1, u2, u3])

        # hidden layer
        hidden_layer_input = np.dot(u, v) + d

        output = act_func(hidden_layer_input)

        # loss
        current_loss = loss(output, target)
        total_loss += current_loss

        # backpropagation
        output_error = output - target
        deriv = act_der(hidden_layer_input)

        # gradients for v and d
        grad_v = output_error * deriv * u
        grad_d = output_error * deriv

        grad_u = output_error * deriv * v
        grad_z1 = grad_u[0] * act_der(z1)
        grad_z2 = grad_u[1] * act_der(z2)
        grad_z3 = grad_u[2] * act_der(z3)
        grad_w1 = grad_z1 * preds
        grad_w2 = grad_z2 * preds
        grad_w3 = grad_z3 * preds
        grad_c1 = grad_z1
        grad_c2 = grad_z2
        grad_c3 = grad_z3

        w1 -= learning_rate * grad_w1
        w2 -= learning_rate * grad_w2
```

```
            w3 -= learning_rate * grad_w3
            c1 -= learning_rate * grad_c1
            c2 -= learning_rate * grad_c2
            c3 -= learning_rate * grad_c3
            v -= learning_rate * grad_v
            d -= learning_rate * grad_d

        if check_stopping_condition_all(v, d, w1, w2, w3, c1, c2, c3, 1e-4):
            break

    return w1, w2, w3, c1, c2, c3, v, d
```

[35]:
```
def predict_with_fitted_model(preds, w1, w2, w3, c1, c2, c3, v, d,␣
 ↪activation='sigmoid'):
    preds = np.array(preds)
    if activation == 'sigmoid':
        act_func = sigmoid
    elif activation == 'relu':
        act_func = relu
    elif activation == 'softplus':
        act_func = softplus

    z1 = np.dot(preds, w1) + c1
    z2 = np.dot(preds, w2) + c2
    z3 = np.dot(preds, w3) + c3
    u1 = act_func(z1)
    u2 = act_func(z2)
    u3 = act_func(z3)
    u = np.array([u1, u2, u3])
    # hidden layer
    hidden_layer_input = np.dot(u, v) + d
    output = act_func(hidden_layer_input)

    return output * 100
```

[36]:
```
# split data into training and testing

def split_data(X, y, test_size=0.2):
    n = X.shape[0]
    indices = list(range(n))
    split = int(test_size * n)
    train_indices = indices[split:]
    test_indices = indices[:split]
    X_train = X[train_indices]
    y_train = y[train_indices]
    X_test = X[test_indices]
    y_test = y[test_indices]
```

```python
    return X_train, y_train, X_test, y_test

cannabis_preds = cannabis[effect_cols.tolist() + flavors_cols.tolist() +
 ↪['Rating']].values
cannabis_target = cannabis['Type'].values
X_train, y_train, X_test, y_test = split_data(cannabis_preds, cannabis_target)

# fit the model
sig_w1, sig_w2, sig_w3, sig_c1, sig_c2, sig_c3, sig_v, sig_d =
 ↪gd_perceptron_all(X_train, y_train, w1, w2, w3, c1, c2, c3,
 ↪activation='sigmoid')
relu_w1, relu_w2, relu_w3, relu_c1, relu_c2, relu_c3, relu_v, relu_d =
 ↪gd_perceptron_all(X_train, y_train, w1, w2, w3, c1, c2, c3,
 ↪activation='relu')
softplus_w1, softplus_w2, softplus_w3, softplus_c1, softplus_c2, softplus_c3,
 ↪softplus_v, softplus_d = gd_perceptron_all(X_train, y_train, w1, w2, w3, c1,
 ↪c2, c3, activation='softplus')

# print values
print('Sigmoid Activation')
print('w1:', sig_w1)
print('w2:', sig_w2)
print('w3:', sig_w3)
print('c1:', sig_c1)
print('c2:', sig_c2)
print('c3:', sig_c3)
print('v:', sig_v)
print('d:', sig_d)
print()
print('ReLU Activation')
print('w1:', relu_w1)
print('w2:', relu_w2)
print('w3:', relu_w3)
print('c1:', relu_c1)
print('c2:', relu_c2)
print('c3:', relu_c3)
print('v:', relu_v)
print('d:', relu_d)
print()
print('Softplus Activation')
print('w1:', softplus_w1)
print('w2:', softplus_w2)
print('w3:', softplus_w3)
print('c1:', softplus_c1)
print('c2:', softplus_c2)
print('c3:', softplus_c3)
print('v:', softplus_v)
```

```python
print('d:', softplus_d)
```

```
Sigmoid Activation
w1: [ 0.075477   -2.12873377 -0.32514469  0.69564431 -0.88976412 -1.01296195
  0.27093805  0.13004627  0.10504993  0.31907568 -0.26513987 -0.22533783
 -1.95733903  0.          0.         -0.02699409  0.14580948 -0.05015092
 -0.59886813 -0.01454399  0.55447122 -0.67386983  0.34830609 -0.69616151
  0.00856163  0.09599129 -0.50840394  0.0845784  -0.40932511 -0.05446706
 -0.74440592 -0.78302727 -0.22730609 -0.18885883 -1.09468843  0.77302618
  0.09184274 -0.61724418 -0.1576483   0.32280642 -0.04405735 -0.69858147
 -0.51081936  0.10842524  0.95211691  0.0071222  -1.00626418 -1.09134218
 -0.41260983 -0.44863226 -0.81963786  0.01921836  0.45686051  0.59322104
  0.59589368 -0.27178954  0.11282168  0.11282168 -0.99624553  0.6602568
  0.02693892  0.45096323  0.37325641 -0.02779038  0.13262084]
w2: [-0.04841506 -1.54246462 -0.22043579 -0.84623993 -0.25187253  1.05957999
  0.13577786 -0.08677695 -0.24954619  0.41355016 -0.59747757  0.78475009
 -0.08509388  0.          0.          0.30090535 -0.2252802   0.73289866
  0.04626022  0.48341069 -0.42191005 -0.30113617  0.22997438 -0.52399858
  0.76569092  0.42592472  0.09090763 -0.44694365 -1.04716844 -1.31850534
  0.49724204  0.39060808 -0.29111156 -0.69148459 -0.66631269 -0.61943846
  0.46121787  0.65112334 -0.17098352  0.38778931  0.07845703 -0.2711626
 -1.32811121  0.2214002   0.52386042  0.36121437  0.43273977 -1.06434024
 -0.20948544 -0.57463214 -0.87067664 -0.61470267 -0.01630856 -0.44507834
 -0.64987735  0.06823951 -0.11854253 -0.11854253 -1.82046372  1.07956126
  0.59106468  0.46772804  0.29606295 -0.36565086  0.16856926]
w3: [-0.11366578  3.76324173 -0.11485786 -0.07668415  0.20787411 -0.24282371
 -0.25131471  0.06126664 -0.15936454 -0.42312391  0.21277345 -0.25553937
 -1.26675875  0.          0.         -0.0516171  -0.11865998 -0.30611271
  0.23177556  0.12494118 -0.40348297  0.38869446 -0.26173634  0.42477008
  0.02216513  0.51030638  0.15255343 -0.06681047  0.11409022  0.40104259
  0.10833526 -0.00696799  0.19473787 -0.5413486   0.04705075 -0.3738115
  0.60686447  0.09425342  0.11539596 -0.06294225  0.04323501  0.10795049
 -0.23809831 -0.06256848  0.12415922 -0.08754593  0.23200713  0.04981783
  0.24819462  0.29231266  0.00849426  0.09340355 -0.23857715 -0.68979169
 -0.76784011  0.11095347 -0.01910976 -0.01910976 -0.68266748 -0.47250931
 -0.12405014 -1.47991256  0.06260833 -0.00568889 -0.1866995 ]
c1: 0.36816783580494766
c2: -0.005427756049399452
c3: -0.17987481769247732
v: [ 4.81054983  3.43805604 -0.90254418]
d: -4.273618721086577

ReLU Activation
w1: [ 0.075477   -2.12873377 -0.32514469  0.69564431 -0.88976412 -1.01296195
  0.27093805  0.13004627  0.10504993  0.31907568 -0.26513987 -0.22533783
 -1.95733903  0.          0.         -0.02699409  0.14580948 -0.05015092
 -0.59886813 -0.01454399  0.55447122 -0.67386983  0.34830609 -0.69616151
```

```
  0.00856163  0.09599129 -0.50840394  0.0845784  -0.40932511 -0.05446706
 -0.74440592 -0.78302727 -0.22730609 -0.18885883 -1.09468843  0.77302618
  0.09184274 -0.61724418 -0.1576483   0.32280642 -0.04405735 -0.69858147
 -0.51081936  0.10842524  0.95211691  0.0071222  -1.00626418 -1.09134218
 -0.41260983 -0.44863226 -0.81963786  0.01921836  0.45686051  0.59322104
  0.59589368 -0.27178954  0.11282168  0.11282168 -0.99624553  0.6602568
  0.02693892  0.45096323  0.37325641 -0.02779038  0.13262084]
w2: [-0.04841506 -1.54246462 -0.22043579 -0.84623993 -0.25187253  1.05957999
  0.13577786 -0.08677695 -0.24954619  0.41355016 -0.59747757  0.78475009
 -0.08509388  0.          0.          0.30090535 -0.2252802   0.73289866
  0.04626022  0.48341069 -0.42191005 -0.30113617  0.22997438 -0.52399858
  0.76569092  0.42592472  0.09090763 -0.44694365 -1.04716844 -1.31850534
  0.49724204  0.39060808 -0.29111156 -0.69148459 -0.66631269 -0.61943846
  0.46121787  0.65112334 -0.17098352  0.38778931  0.07845703 -0.2711626
 -1.32811121  0.2214002   0.52386042  0.36121437  0.43273977 -1.06434024
 -0.20948544 -0.57463214 -0.87067664 -0.61470267 -0.01630856 -0.44507834
 -0.64987735  0.06823951 -0.11854253 -0.11854253 -1.82046372  1.07956126
  0.59106468  0.46772804  0.29606295 -0.36565086  0.16856926]
w3: [-0.11366578  3.76324173 -0.11485786 -0.07668415  0.20787411 -0.24282371
 -0.25131471  0.06126664 -0.15936454 -0.42312391  0.21277345 -0.25553937
 -1.26675875  0.          0.         -0.0516171  -0.11865998 -0.30611271
  0.23177556  0.12494118 -0.40348297  0.38869446 -0.26173634  0.42477008
  0.02216513  0.51030638  0.15255343 -0.06681047  0.11409022  0.40104259
  0.10833526 -0.00696799  0.19473787 -0.5413486   0.04705075 -0.3738115
  0.60686447  0.09425342  0.11539596 -0.06294225  0.04323501  0.10795049
 -0.23809831 -0.06256848  0.12415922 -0.08754593  0.23200713  0.04981783
  0.24819462  0.29231266  0.00849426  0.09340355 -0.23857715 -0.68979169
 -0.76784011  0.11095347 -0.01910976 -0.01910976 -0.68266748 -0.47250931
 -0.12405014 -1.47991256  0.06260833 -0.00568889 -0.1866995 ]
c1: 1.232214891556679
c2: 0.334930963047911
c3: 0.3487275127347523
v: [0.2647654  0.21956922 0.52782718]
d: -0.1035158872160213

Softplus Activation
w1: [ 0.075477   -2.12873377 -0.32514469  0.69564431 -0.88976412 -1.01296195
  0.27093805  0.13004627  0.10504993  0.31907568 -0.26513987 -0.22533783
 -1.95733903  0.          0.         -0.02699409  0.14580948 -0.05015092
 -0.59886813 -0.01454399  0.55447122 -0.67386983  0.34830609 -0.69616151
  0.00856163  0.09599129 -0.50840394  0.0845784  -0.40932511 -0.05446706
 -0.74440592 -0.78302727 -0.22730609 -0.18885883 -1.09468843  0.77302618
  0.09184274 -0.61724418 -0.1576483   0.32280642 -0.04405735 -0.69858147
 -0.51081936  0.10842524  0.95211691  0.0071222  -1.00626418 -1.09134218
 -0.41260983 -0.44863226 -0.81963786  0.01921836  0.45686051  0.59322104
  0.59589368 -0.27178954  0.11282168  0.11282168 -0.99624553  0.6602568
  0.02693892  0.45096323  0.37325641 -0.02779038  0.13262084]
w2: [-0.04841506 -1.54246462 -0.22043579 -0.84623993 -0.25187253  1.05957999
```

```
 0.13577786 -0.08677695 -0.24954619  0.41355016 -0.59747757  0.78475009
-0.08509388  0.          0.          0.30090535 -0.2252802   0.73289866
 0.04626022  0.48341069 -0.42191005 -0.30113617  0.22997438 -0.52399858
 0.76569092  0.42592472  0.09090763 -0.44694365 -1.04716844 -1.31850534
 0.49724204  0.39060808 -0.29111156 -0.69148459 -0.66631269 -0.61943846
 0.46121787  0.65112334 -0.17098352  0.38778931  0.07845703 -0.2711626
-1.32811121  0.2214002   0.52386042  0.36121437  0.43273977 -1.06434024
-0.20948544 -0.57463214 -0.87067664 -0.61470267 -0.01630856 -0.44507834
-0.64987735  0.06823951 -0.11854253 -0.11854253 -1.82046372  1.07956126
 0.59106468  0.46772804  0.29606295 -0.36565086  0.16856926]
w3: [-0.11366578  3.76324173 -0.11485786 -0.07668415  0.20787411 -0.24282371
 -0.25131471  0.06126664 -0.15936454 -0.42312391  0.21277345 -0.25553937
 -1.26675875  0.          0.         -0.0516171  -0.11865998 -0.30611271
  0.23177556  0.12494118 -0.40348297  0.38869446 -0.26173634  0.42477008
  0.02216513  0.51030638  0.15255343 -0.06681047  0.11409022  0.40104259
  0.10833526 -0.00696799  0.19473787 -0.5413486   0.04705075 -0.3738115
  0.60686447  0.09425342  0.11539596 -0.06294225  0.04323501  0.10795049
 -0.23809831 -0.06256848  0.12415922 -0.08754593  0.23200713  0.04981783
  0.24819462  0.29231266  0.00849426  0.09340355 -0.23857715 -0.68979169
 -0.76784011  0.11095347 -0.01910976 -0.01910976 -0.68266748 -0.47250931
 -0.12405014 -1.47991256  0.06260833 -0.00568889 -0.1866995 ]
c1: 0.7904571485287881
c2: 0.28213461196870143
c3: 0.04788863967355196
v: [1.32598995 1.0049609  1.20583514]
d: -3.767131507951776
```

1.2.2 Question 2: NN with SVC Loss

Make a new neural network fitting function, but with SVC loss instead. You only need to implement this with ReLu activation for the first layer, and Sigmoid for the second.

(Hint: remember that we need to represent the y-values as 1 or -1 to use this loss function.)

Use your function to fit an Indica vs Sativa model on a 80% training set from the cannabis data.

[37]:
```python
def relu(x):
    return np.maximum(0, x)

# hinge loss (SVC loss)
def hinge_loss(pred, target):
    return np.maximum(0, 1 - target * pred)

# derivative of hinge loss with respect to the prediction
def hinge_loss_derivative(pred, target):
    return-target if pred < 1 else 0

def check_stopping_condition(v, d, threshold=1e-4):
    return np.linalg.norm(v) < threshold and np.abs(d) < threshold
```

[38]:
```python
def fit_perceptron_svc(train_pred, train_target, w1, w2, w3, c1, c2, c3,␣
↪learning_rate=0.01, epochs=1000):
    # Initialize v and d
    v = np.array([1/3, 1/3, 1/3])
    d = 0

    # convert target to -1 and 1
    train_target = np.where(train_target == 0, -1, 1)

    for epoch in range(1, epochs+1):
        total_loss = 0

        for preds, target in zip(train_pred, train_target):
            preds = np.array(preds)
            z1 = np.dot(preds, w1) + c1
            z2 = np.dot(preds, w2) + c2
            z3 = np.dot(preds, w3) + c3
            u1 = relu(z1)
            u2 = relu(z2)
            u3 = relu(z3)
            u = np.array([u1, u2, u3])

            # hidden layer
            hidden_layer_input = np.dot(u, v) + d
            output = sigmoid(hidden_layer_input)

            # compute loss
            current_loss = hinge_loss(output, target)
            total_loss += current_loss

            # backpropagation
            output_error = hinge_loss_derivative(output, target)
            deriv = sigmoid_der(output)

            # gradients for v and d
            grad_v = output_error * deriv * u
            grad_d = output_error * deriv

            # gradients for w's and c's
            grad_u = output_error * deriv * v
            grad_z1 = grad_u[0] * (z1 > 0)
            grad_z2 = grad_u[1] * (z2 > 0)
            grad_z3 = grad_u[2] * (z3 > 0)
            grad_w1 = grad_z1 * preds
            grad_w2 = grad_z2 * preds
            grad_w3 = grad_z3 * preds
            grad_c1 = grad_z1
```

```
            grad_c2 = grad_z2
            grad_c3 = grad_z3

            # update params
            w1 -= learning_rate * grad_w1
            w2 -= learning_rate * grad_w2
            w3 -= learning_rate * grad_w3
            c1 -= learning_rate * grad_c1
            c2 -= learning_rate * grad_c2
            c3 -= learning_rate * grad_c3
            v -= learning_rate * grad_v
            d -= learning_rate * grad_d

        if check_stopping_condition(v, d, 1e-4):
            break

    return w1, w2, w3, c1, c2, c3, v, d
```

[39]:
```
# ignore warnings
import warnings
warnings.filterwarnings('ignore')
def predict_svc(preds, w1, w2, w3, c1, c2, c3, v, d):
    preds = np.array(preds)
    z1 = np.dot(preds, w1) + c1
    z2 = np.dot(preds, w2) + c2
    z3 = np.dot(preds, w3) + c3
    u1 = relu(z1)
    u2 = relu(z2)
    u3 = relu(z3)
    u = np.array([u1, u2, u3])
    # Hidden layer output
    hidden_layer_input = np.dot(u, v) + d
    output = sigmoid(hidden_layer_input)

    return output * 100

svc_w1, svc_w2, svc_w3, svc_c1, svc_c2, svc_c3, svc_v, svc_d =␣
 ↪fit_perceptron_svc(X_train, y_train, w1, w2, w3, c1, c2, c3)

# print values
print('SVC')
print('w1:', svc_w1)
print('w2:', svc_w2)
print('w3:', svc_w3)
print('c1:', svc_c1)
print('c2:', svc_c2)
print('c3:', svc_c3)
```

```python
print('v:', svc_v)
print('d:', svc_d)
```

```
SVC
w1: [ 1.74907501e+00 -9.84223499e+00 -3.61494796e+01 -1.21407108e+00
 -3.10341934e+01 -3.24834144e+01 -3.93728573e-01  1.09853916e+01
 -5.24278386e+00  1.75691949e+01 -1.39809669e+01  2.88459984e+01
 -4.11026028e+01  0.00000000e+00  0.00000000e+00  4.49056398e+00
  2.30507834e+01  2.94350882e+01 -1.10735864e+00 -1.56819797e-02
  3.43280771e+01 -8.27868388e+00  2.40920096e-01 -1.81014108e+01
  7.49706117e-02  9.07279533e+00  7.62553980e+00 -7.22878567e+00
 -2.36213700e+01  2.14067039e+01  4.88739285e+00 -1.10841861e+01
 -7.12001212e+00 -2.22544112e+00 -1.24756225e+01  7.21840982e-01
  3.17679381e-01 -8.24662967e-01  4.08685870e+00  5.85358956e-01
 -4.22253044e-02 -3.35964245e+00 -1.15396746e+00  1.66269341e+00
  2.07392698e+00 -7.08799301e+00  7.81826734e+00 -1.24722762e+01
 -7.00857351e+00  7.94785908e+00 -8.19637857e-01 -5.01335387e-01
  6.59929770e+00  1.28767226e+01  2.90600998e+00 -3.52178610e+00
  9.41207311e+00  9.41207311e+00 -1.64497882e+00  8.88650080e+00
  2.69389195e-02  8.00829151e-01  1.92000435e+00 -2.77903816e-02
  4.66881235e-01]
w2: [ -1.34442846   1.65453904 -11.38326412  -9.44318385  -9.35623261
  19.97854496   3.15286943  -5.4813523   -5.18656955  16.39597146
   5.93815888  21.5210856   -5.61626348   0.           0.
  -2.34963887   3.84708479  27.40563018  -8.96980644   0.48341069
  -6.79665012   9.19354431  12.78829127   2.03089389   0.80030359
   0.48046162   5.37132927  -6.41070978  -3.62539792 -11.16287951
   5.7860305   -7.44808755   0.46835727  -2.32603301   1.23477461
  -6.60879617   0.8057851    0.38331621   8.4338799    0.51107019
   0.07845703  -0.2711626  -18.03689174   3.6694631    1.09562671
   7.79046518   0.15340551   0.83674706  -2.28915367   4.88869763
  -0.87067664  -0.6194929   -4.35335406  -0.44507834  -0.68271934
  -1.86168329  -9.13134098  -9.13134098  -7.80982142   1.32456532
   0.59106468  -5.8114758    0.29526807  -0.36565086  -4.23562549]
w3: [ 4.44669840e+00  4.78108984e+01 -1.47395206e+01  1.57774460e+00
 -2.62381878e+01 -8.00752181e+00  7.81698432e-01 -3.39653316e+00
 -7.92589022e+00 -3.59363490e+00 -1.28527800e+01  1.45908096e+01
 -5.02534893e+01  0.00000000e+00  0.00000000e+00 -5.55199148e+00
  8.64226314e+00  1.57285624e+00 -2.76007920e+01  1.12693911e-01
  1.22642017e+01 -1.05139851e+01  5.98900306e+00 -8.90803379e+00
  3.04625100e-01 -6.34831927e+00  1.43272210e+01 -7.30717672e+00
 -1.18871029e+01 -3.52733511e+00  1.00758271e+01 -1.16694439e+01
 -6.35377924e+00 -4.59962916e+00 -2.14721714e+01  1.50681690e+00
  2.31130988e+00  1.70931878e+01  2.07444160e+01  7.68521655e-01
  5.18047284e+00 -1.77258478e+01 -6.46734591e+00  8.08560354e+00
  3.19079812e+00  2.17145670e+01  4.07147181e+00 -2.14602879e+01
 -1.22805180e+01  2.30204038e-01  1.23541951e+01  1.33993341e+01
```

```
 -1.18525614e+01  7.13587166e+00  2.97288833e+01 -1.55468013e+00
  6.00129884e+00  6.00129884e+00 -1.00320206e+01  3.90412247e+01
 -1.15834248e-01 -2.84971533e+01  4.61932704e+00 -1.89184280e-02
  3.57768791e+00]
c1: -2.432809733936798
c2: -0.6418640611156522
c3: -0.6274317492814332
v: [85.26455589 57.05702403 93.66643475]
d: -1130.568568673382
```

1.2.3 Question 3: Comparison

Which of the four Neural Networks (squared loss sigmoid, relu, or softplus; and SVC loss) performed the best on the remaining 20% test set?

[40]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score

# sigmoid activation
y_pred = [predict_with_fitted_model(pred, sig_w1, sig_w2, sig_w3, sig_c1,␣
 ↪sig_c2, sig_c3, sig_v, sig_d) for pred in X_test]
y_pred = [1 if pred > 50 else 0 for pred in y_pred]
print('Sigmoid Activation')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Precision:', precision_score(y_test, y_pred))
print('Recall:', recall_score(y_test, y_pred))
print('F1 Score:', f1_score(y_test, y_pred))
print()

# relu activation
y_pred = [predict_with_fitted_model(pred, relu_w1, relu_w2, relu_w3, relu_c1,␣
 ↪relu_c2, relu_c3, relu_v, relu_d, activation='relu') for pred in X_test]
y_pred = [1 if pred > 50 else 0 for pred in y_pred]
print('ReLU Activation')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Precision:', precision_score(y_test, y_pred))
print('Recall:', recall_score(y_test, y_pred))
print('F1 Score:', f1_score(y_test, y_pred))
print()

# softplus activation
y_pred = [predict_with_fitted_model(pred, softplus_w1, softplus_w2,␣
 ↪softplus_w3, softplus_c1, softplus_c2, softplus_c3, softplus_v, softplus_d,␣
 ↪activation='softplus') for pred in X_test]
y_pred = [1 if pred > 50 else 0 for pred in y_pred]
print('Softplus Activation')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Precision:', precision_score(y_test, y_pred))
```

```
print('Recall:', recall_score(y_test, y_pred))
print('F1 Score:', f1_score(y_test, y_pred))
print()

# svc
y_pred = [predict_svc(pred, svc_w1, svc_w2, svc_w3, svc_c1, svc_c2, svc_c3,␣
 ↪svc_v, svc_d) for pred in X_test]
y_pred = [1 if pred > 50 else 0 for pred in y_pred]
print('SVC')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Precision:', precision_score(y_test, y_pred))
print('Recall:', recall_score(y_test, y_pred))
print('F1 Score:', f1_score(y_test, y_pred))
```

```
Sigmoid Activation
Accuracy: 0.7354260089686099
Precision: 0.6341463414634146
Recall: 0.37142857142857144
F1 Score: 0.46846846846846846

ReLU Activation
Accuracy: 0.8295964125560538
Precision: 0.6951219512195121
Recall: 0.8142857142857143
F1 Score: 0.75

Softplus Activation
Accuracy: 0.8340807174887892
Precision: 0.7037037037037037
Recall: 0.8142857142857143
F1 Score: 0.7549668874172185

SVC
Accuracy: 0.852017937219731
Precision: 0.7761194029850746
Recall: 0.7428571428571429
F1 Score: 0.759124087591241
```

The model with the sigmoid activation function and SVC loss performed the best on the test set with an accuracy of 0.85 and an F1 score of 0.76. The models with the ReLU and Softplus activation functions had better accuracy scores however their F1 scores were just on par with the SVC loss model. THe model that did the worst was the model with the squared loss and sigmoid activation function.

1.3 Part C: Concepts

1.3.1 Question 1: interpret the weights

Investigate the fitted weights in the first layer of one of your networks from B1 and B2: First look at the scores, z_1, z_2, z_3 and see if they correlate to anything related to the data. If they do - for example, if z_1 seems to capture a meaningful data measurement - interpret some of the corresponding weights.

- Taking a look at the weights for the first layer of the model with the sigmoid activation function and squared loss, there does not seem to be any distinct relationship with the data. However the higher z values do indicate more influential features for the final predictions. But from the weights there is no single feature that stands out as being more important than the others. The variation in the z values means that the relationship is more complex that just relying on a single feature.

1.3.2 Question 2: Multiclass model

Suppose you had been asked to instead implement an NN with three outputs: P(Indica), P(Sativa), P(Hybrid).

Explain what parts of you code would need to change, and what parts could stay the same.

- Parts that need to change:
  - Output layer would need to have 3 nodes instead of 1 since we are now returning 3 values
  - Also the activation function would need to changed to a softmax function since we are now dealing with multiple classes. This would also need to be normalized to sum to 1 as they are probabilities.
  - The loss function would also need to be adjusted since we are now dealing with multiple classes. The loss function would need to be a cross-entropy loss function. This works well for multi-class classification problems and is the negative log likelihood of the true labels given the predicted probabilities.
  - Finally the target variable would need to be one-hot encoded to represent the three classes. So instead of binary values we could have an array of 1 or 0 to indicate which class the observation belongs to.
- Parts that could stay the same:
  - General neural network structure would remain the same since we would still have the input, then hidden layer, and then output layer. There would be the same number of layers, but just with more nodes in the output layer.
  - The weights would still be the same to initialize the model. Also the input and hidden layers would stay the same since they are not dependent on the number of classes.