Lab6-The_Perceptron

February 6, 2025

1 Imports

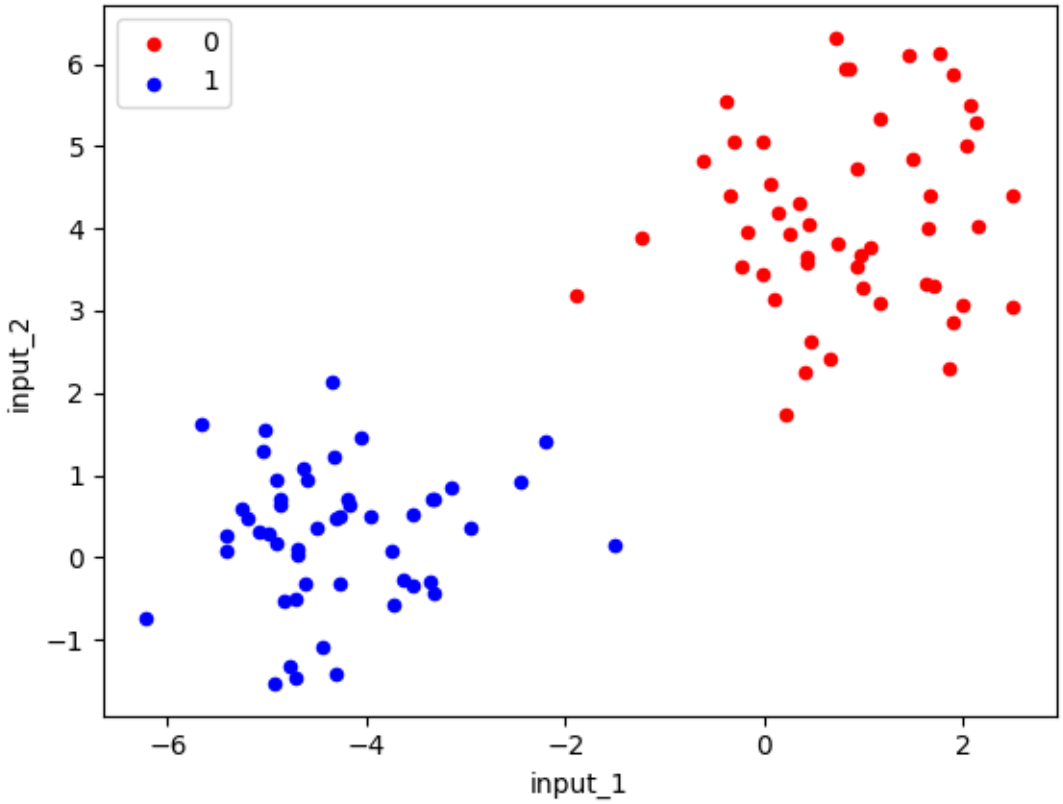
[5]:
```
import numpy as np
from sklearn.datasets import make_blobs
import pandas as pd
from pandas import DataFrame
%matplotlib inline
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
```

2 Dataset

[6]:
```
def prep_data():
    X, labels = make_blobs(n_samples=100, centers=2, n_features=2,␣
 ↪cluster_std=1, random_state=3)
    # labels = np.where(labels == 0, -1, 1)
    #df = DataFrame(dict(constant=np.repeat(1, len(X)), input_1=X[:,0],␣
 ↪input_2=X[:,1], label=labels))
    df = DataFrame(dict(input_1=X[:,0], input_2=X[:,1], label=labels))
    print(df.head())
    colors = {1:'blue', 0:'red'}
    fig, ax = plt.subplots()
    grouped = df.groupby('label')
    for key, group in grouped:
        group.plot(ax=ax, kind='scatter', x='input_1', y='input_2', label=key,␣
 ↪color=colors[key])
    plt.show()
    #y = np.where(labels == 0, -1, 1)
    y = labels
    # sepal length and petal length # from sklearn.datasets.samples_generator␣
 ↪import make_blobs
    X = df.iloc[0:100, [0,1]].values
    return X, y
X, y = prep_data()
```

```
    input_1   input_2  label
0 -3.755938  0.067538      1
```

```
1 -2.948434  0.365538      1
2 -3.367091 -0.303440      1
3  1.890244  2.869420      0
4 -4.770407 -1.330004      1
```



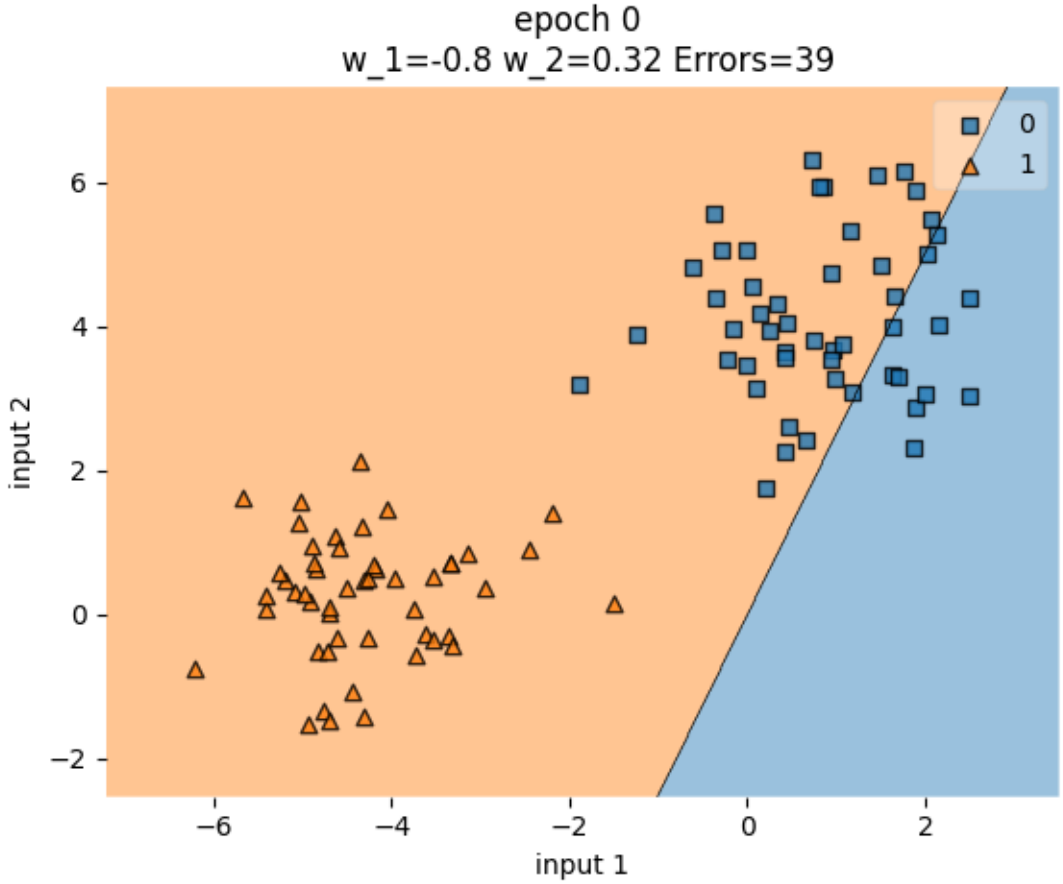
3 The Perceptron Implementation

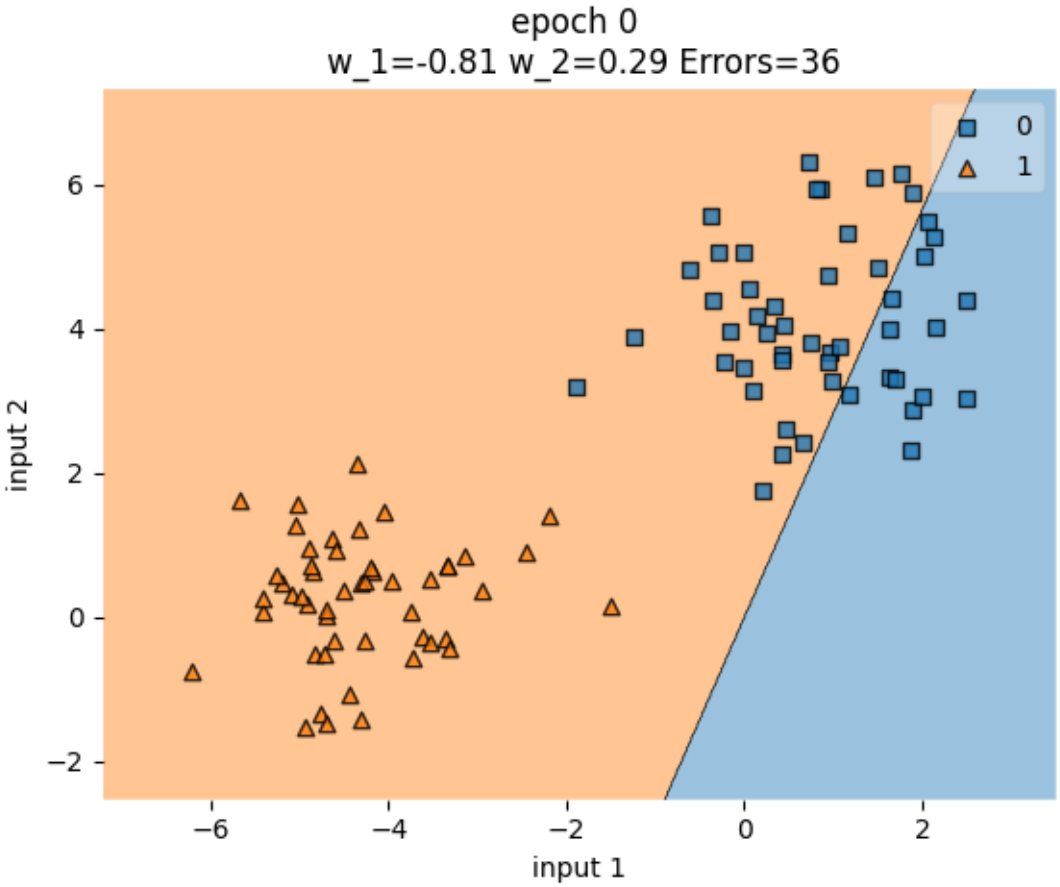
[7]:
```python
class Perceptron(object):
# """Perceptron class

# Usage:
# ppn = Perceptron(epochs=10, lr=0.01)
# ppn.train(X, y)
# ppn.plot_delta()

# Author: Uzair Ahmad
# """
    def __init__(self, lr=0.001, epochs=50):
        self.lr = lr
        self.epochs = epochs
```

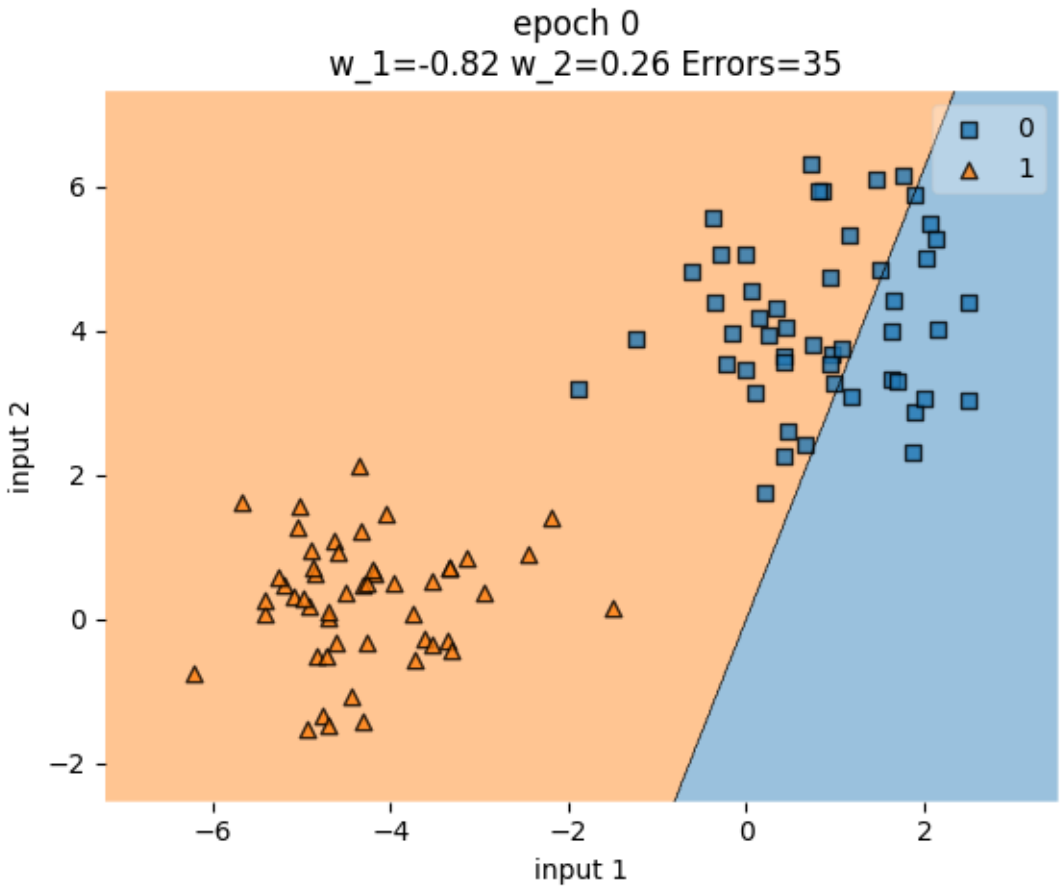
```
    def predict(self, X):
        preds = np.dot(X, self.w_)
        return np.where(preds >= 0, 1, 0)

    def train(self, X, y):
        # Initialize weights
        self.w_ = np.random.randn(2)
        self.errors = []
        for epoch in range(self.epochs):
            # Calculate classification errors
            self.errors.append(np.sum(np.abs(y - self.predict(X))))
            self.plot_dr(X, y, title='epoch {0} \n w_1={1} w_2={2} Errors={3}'.
↪format(epoch, np.round(self.w_[0],2), np.round(self.w_[1],2),
                                                                                ␣
↪     int(np.sum(np.abs(y - self.predict(X))))))
            for xi, yi in zip(X, y):
                output = self.predict(xi)
                # calculate update
                update = self.lr * (yi - output) * xi
                # Update w
                self.w_ +=  update
                if np.all(update != 0):
                  self.plot_dr(X, y, title='epoch {0} \n w_1={1} w_2={2}␣
↪Errors={3}'.format(epoch, np.round(self.w_[0],2), np.round(self.w_[1],2),
                                                                                ␣
↪     int(np.sum(np.abs(y - self.predict(X))))))
        return self

    def plot_dr(self, X, y, title):
        plot_decision_regions(X, y, clf=self)
        plt.title(title)
        plt.xlabel('input 1')
        plt.ylabel('input 2')
        plt.show()

    def plot_delta(self):
      plt.plot(range(1, len(self.errors)+1), self.errors, marker='o')
      plt.title('Errors / Epoch')
      plt.xlabel('Epochs')
      plt.ylabel('Error Count')
      plt.show()
```

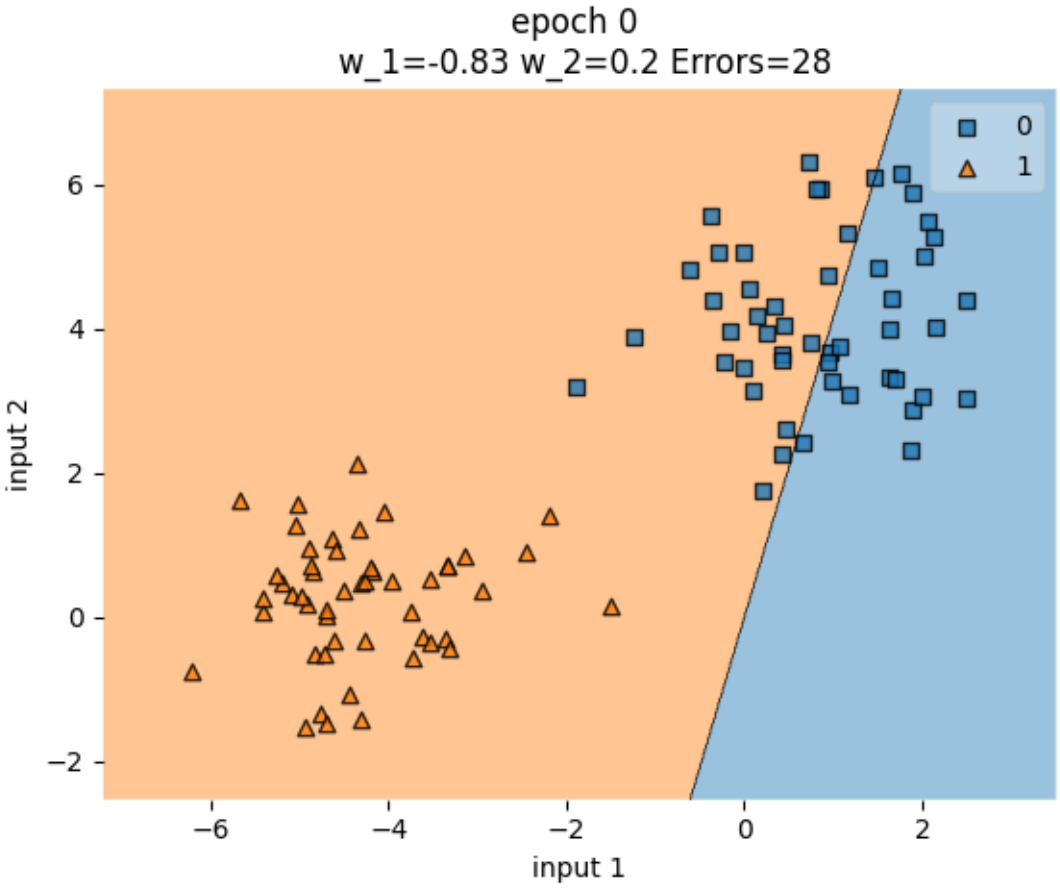
[8]:
```
np.random.seed(1000)
ppn = Perceptron(epochs=10, lr=0.01)
ppn.train(X, y)
ppn.plot_delta()
```

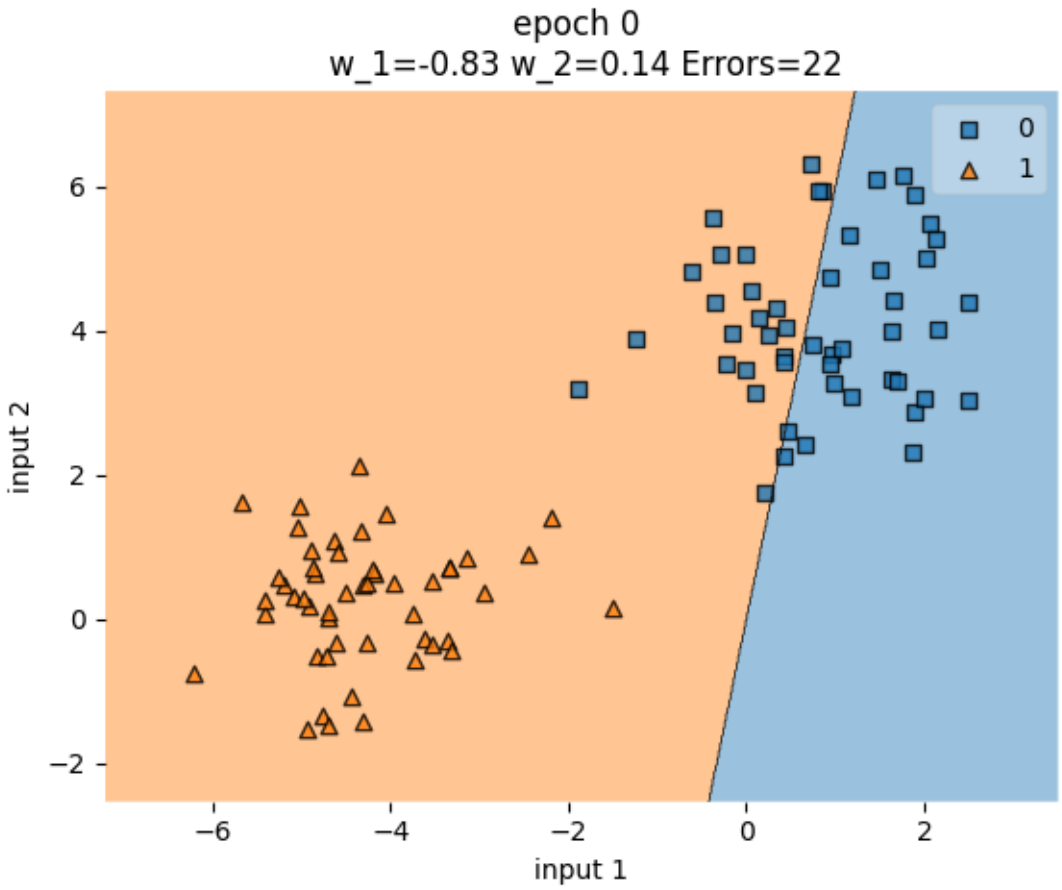
```
W= np.asarray([-0.68, -0.4])
```

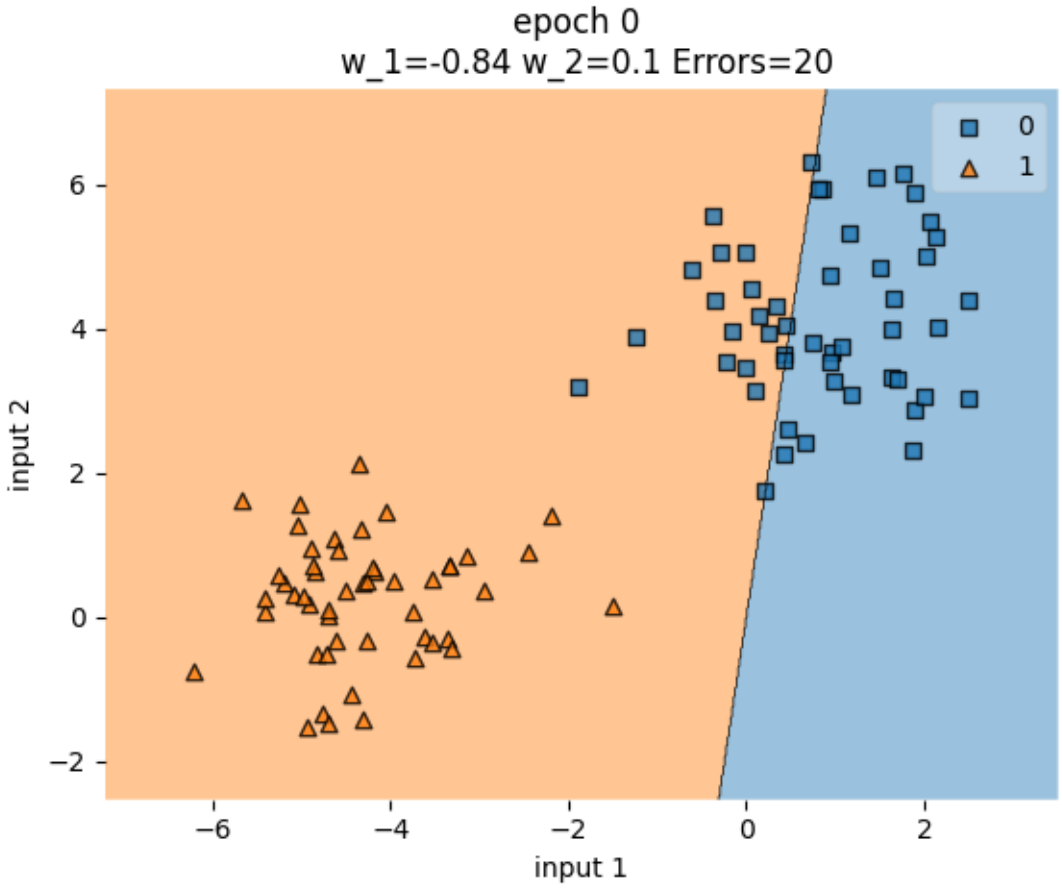


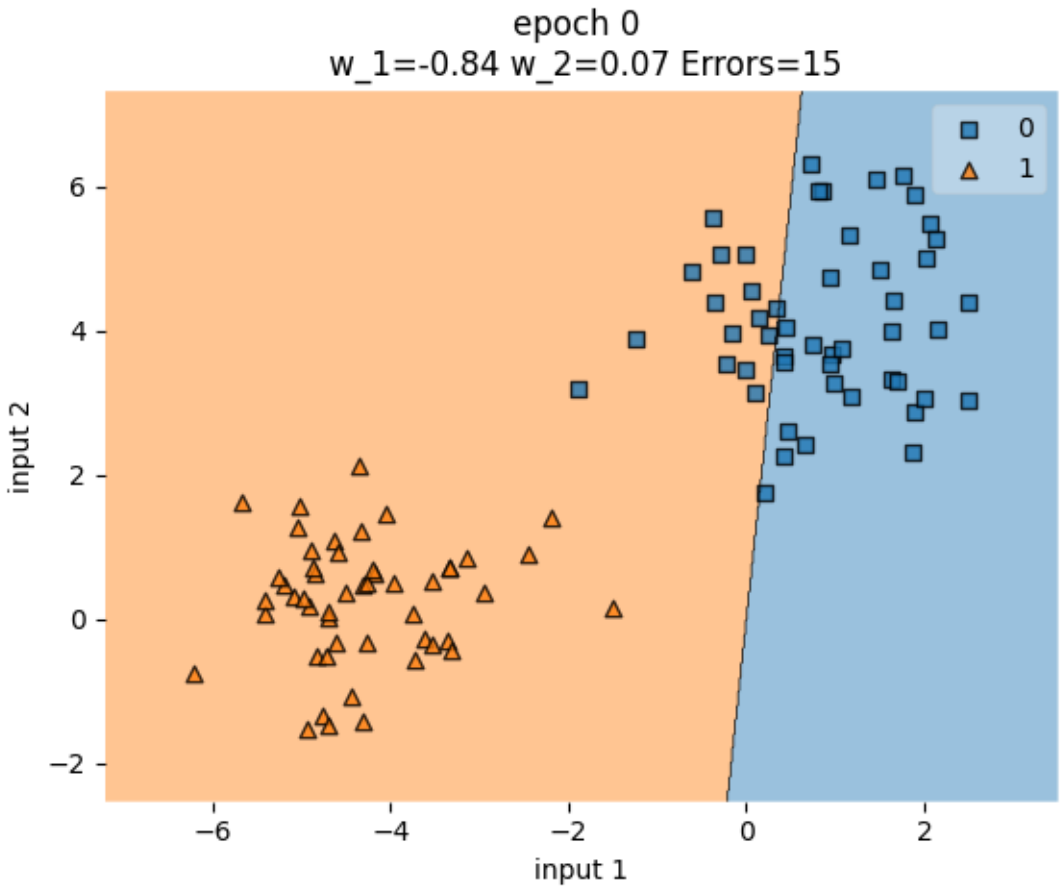

epoch 0
w_1=-0.81 w_2=0.29 Errors=36
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

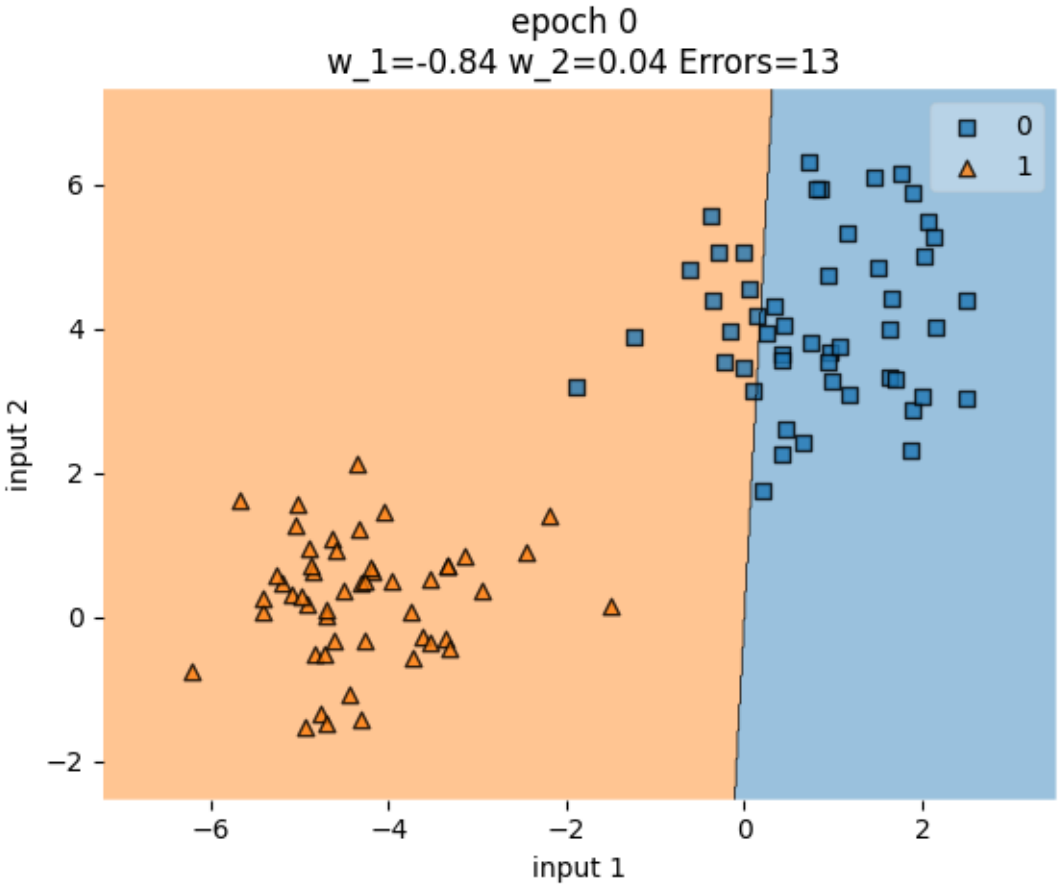

epoch 0
w_1=-0.82 w_2=0.26 Errors=35
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

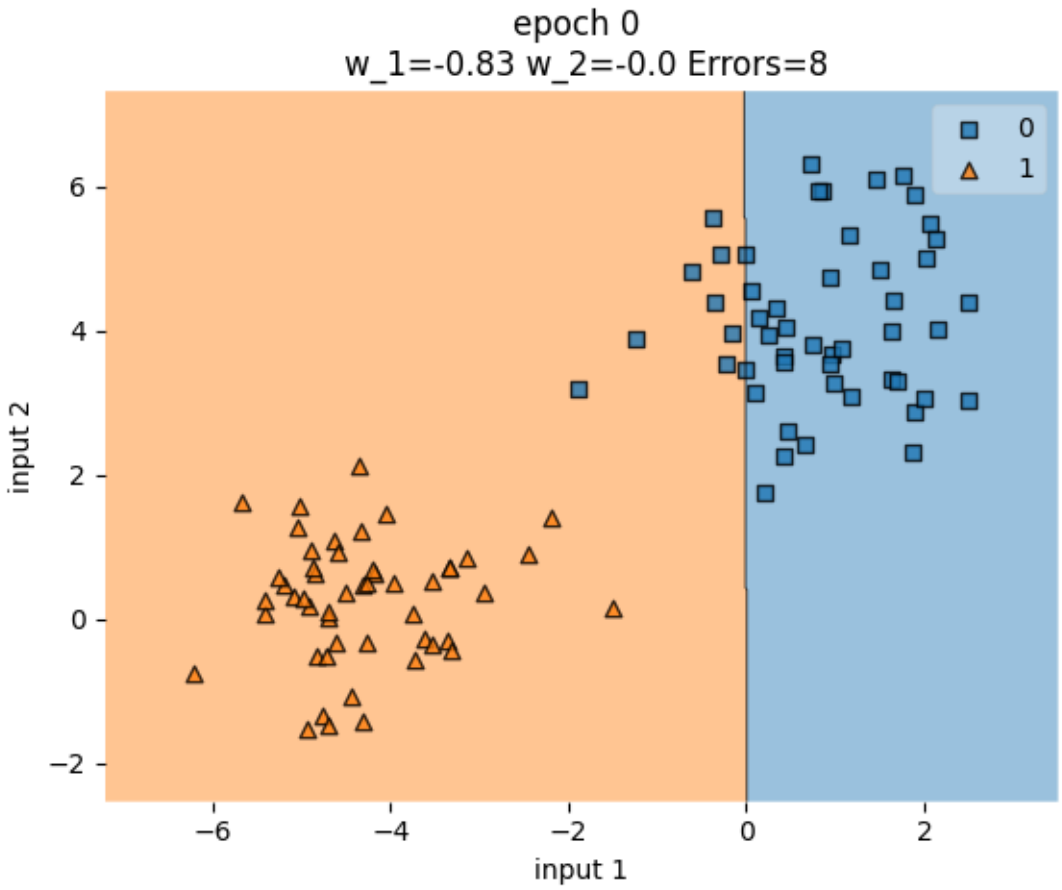

epoch 0
w_1=-0.83 w_2=0.2 Errors=28
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

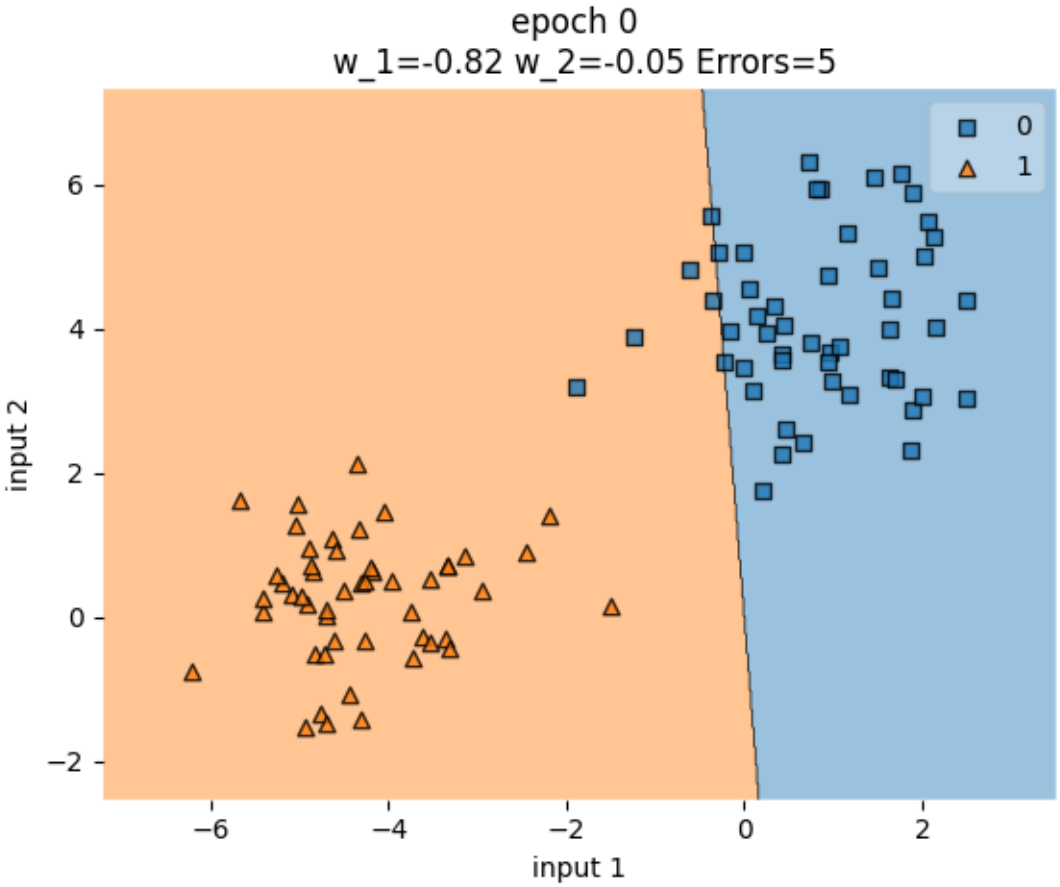

epoch 0
w_1=-0.83 w_2=0.14 Errors=22
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

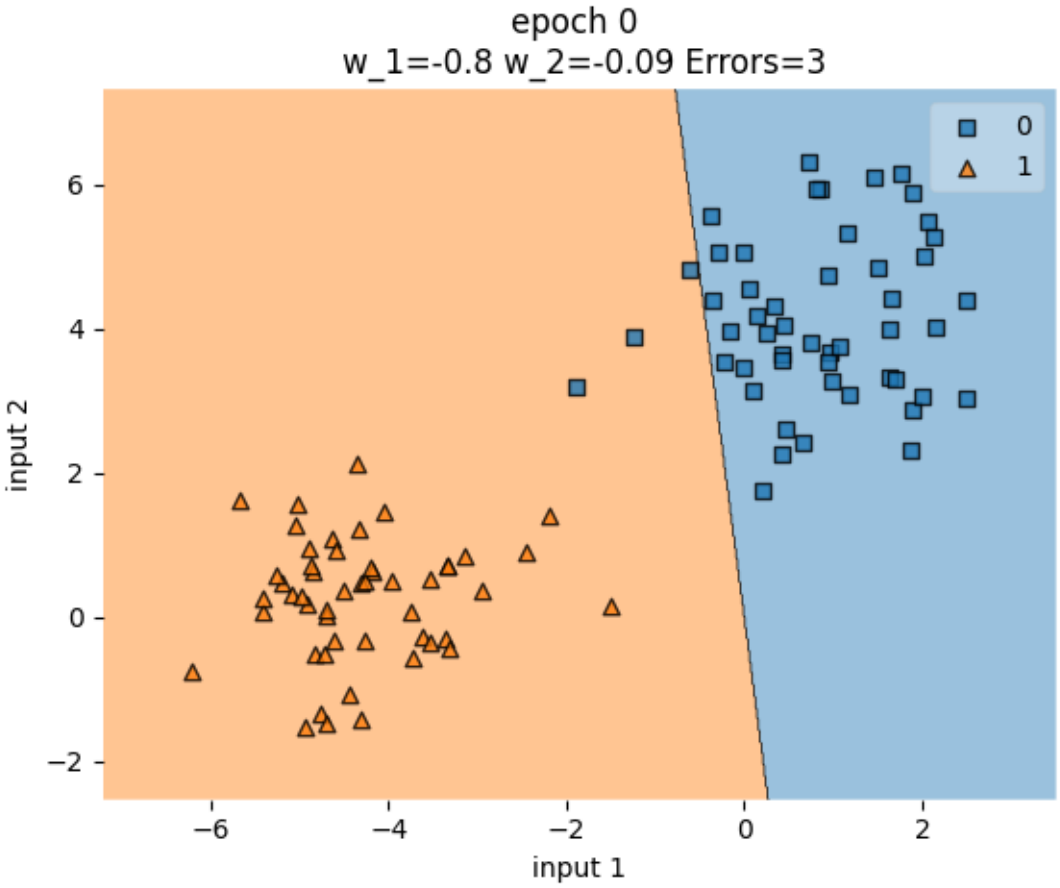

epoch 0
w_1=-0.84 w_2=0.1 Errors=20
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

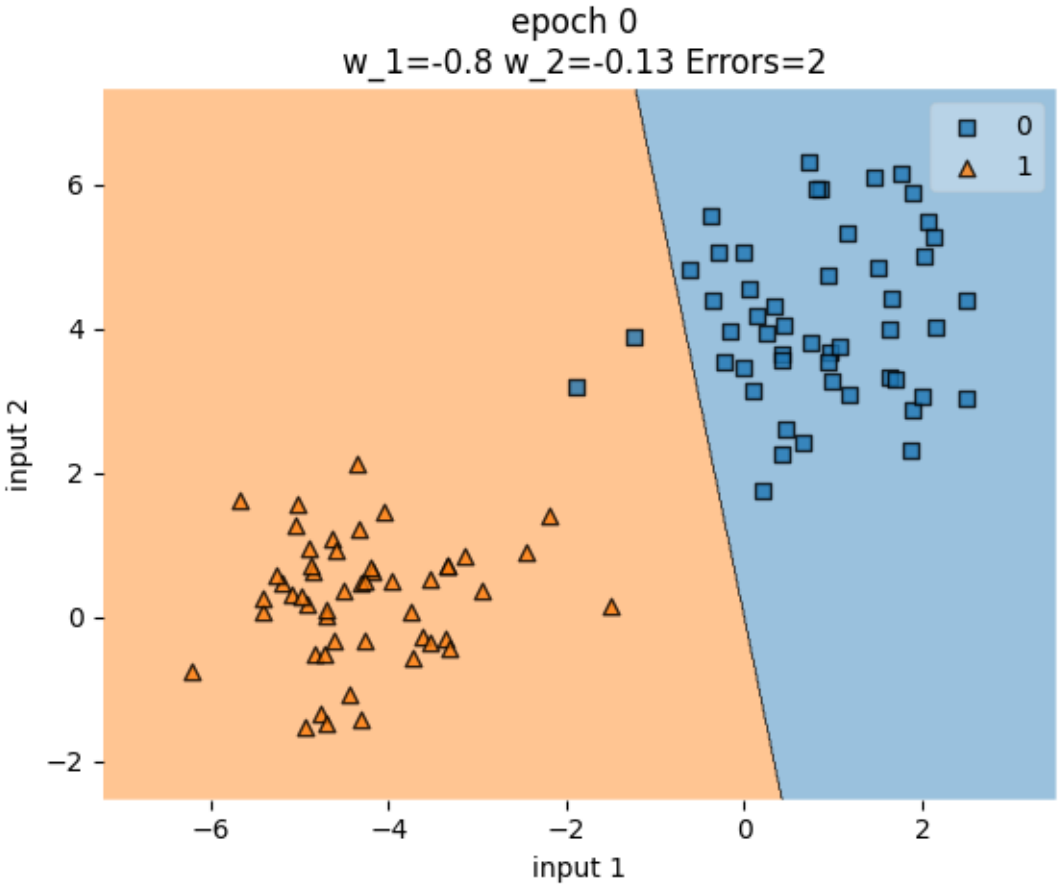

epoch 0
w_1=-0.84 w_2=0.07 Errors=15
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

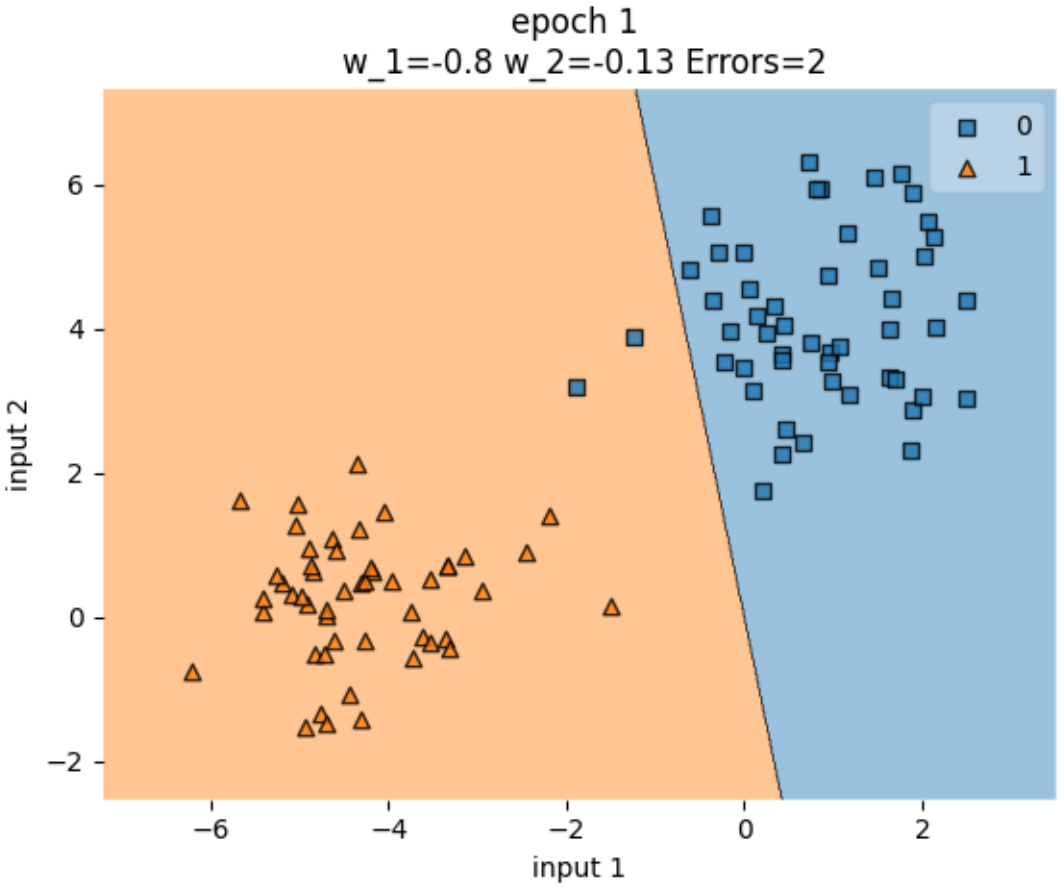

epoch 0
w_1=-0.84 w_2=0.04 Errors=13
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

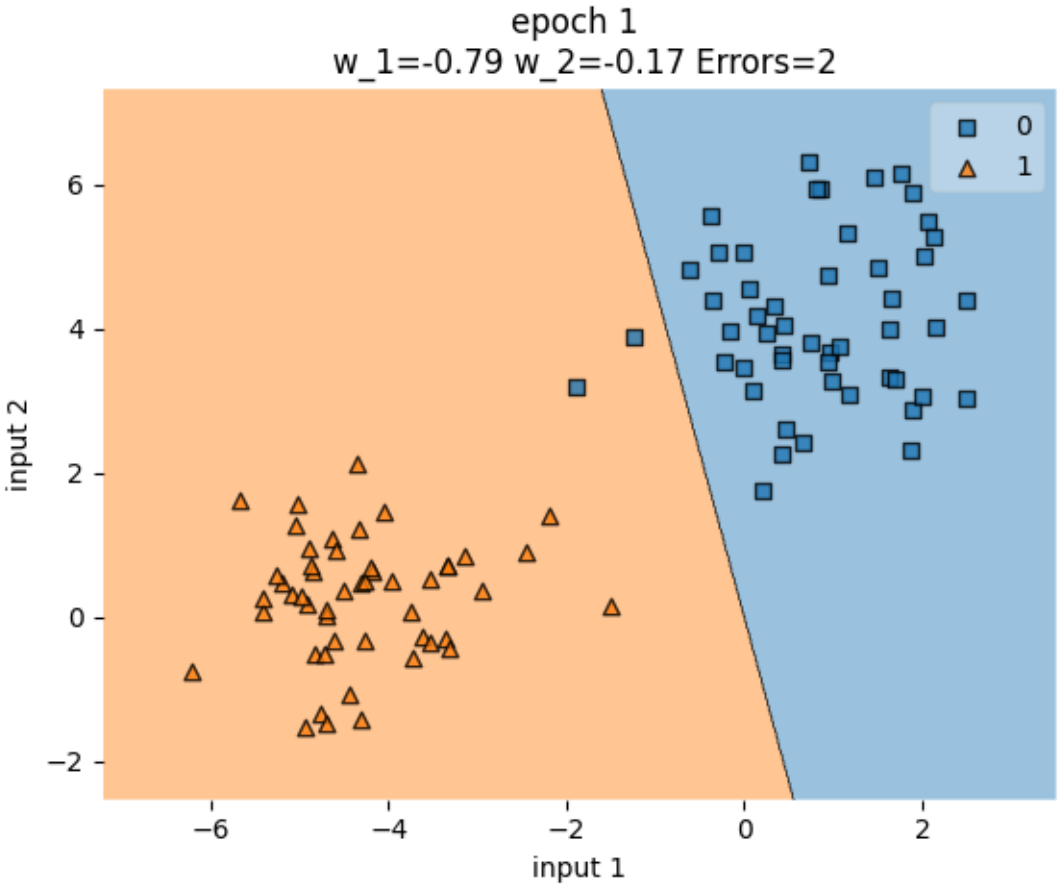

epoch 0
w_1=-0.83 w_2=-0.0 Errors=8
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

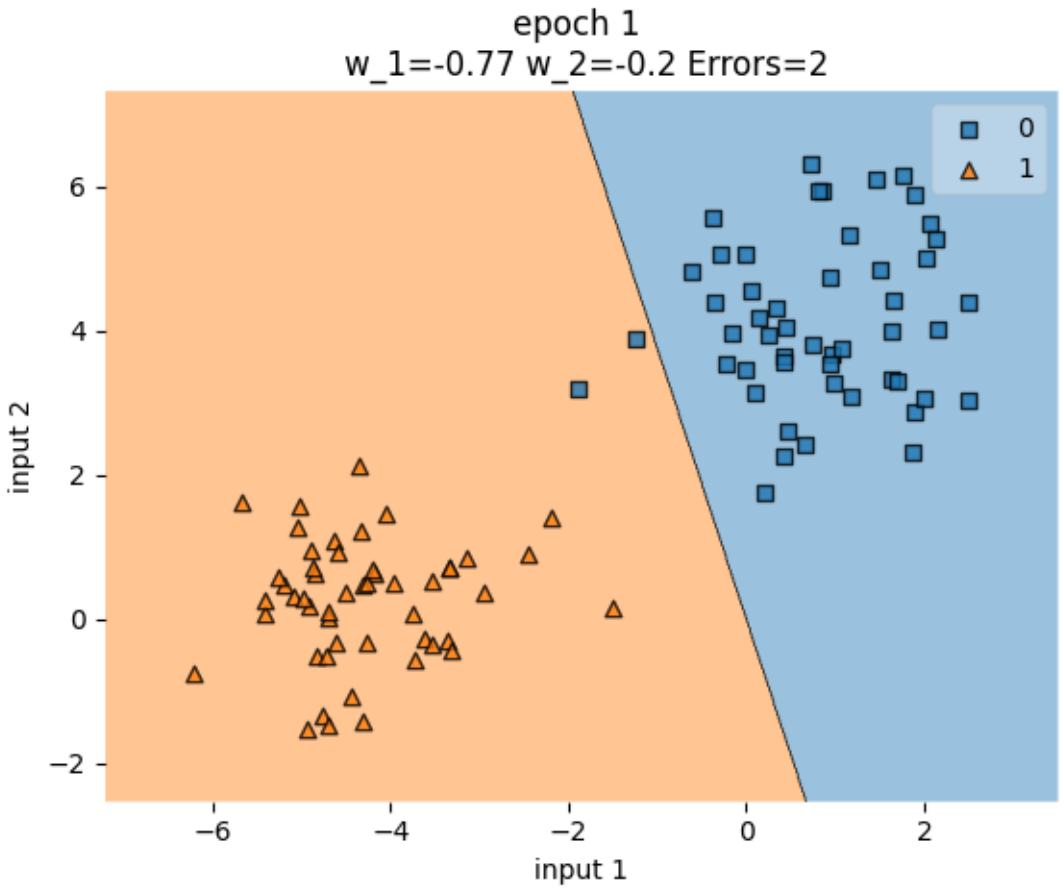

epoch 0
w_1=-0.82 w_2=-0.05 Errors=5
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

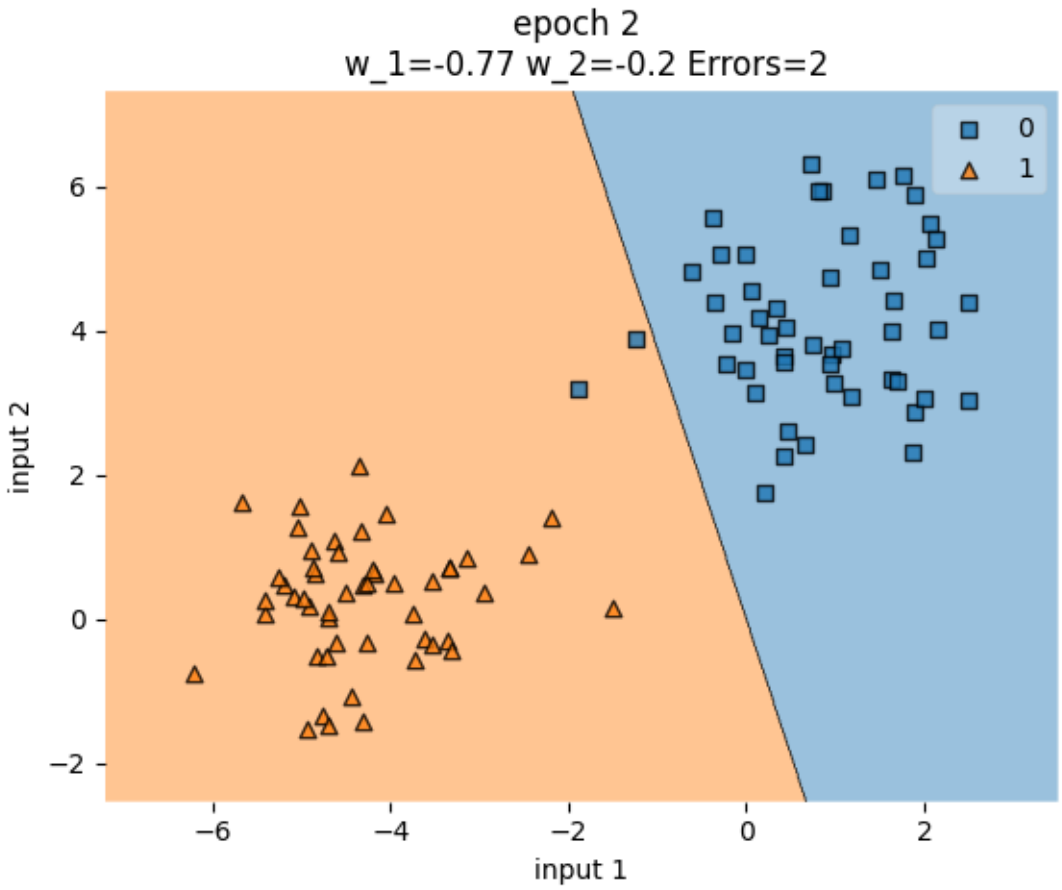

epoch 0
w_1=-0.8 w_2=-0.09 Errors=3
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

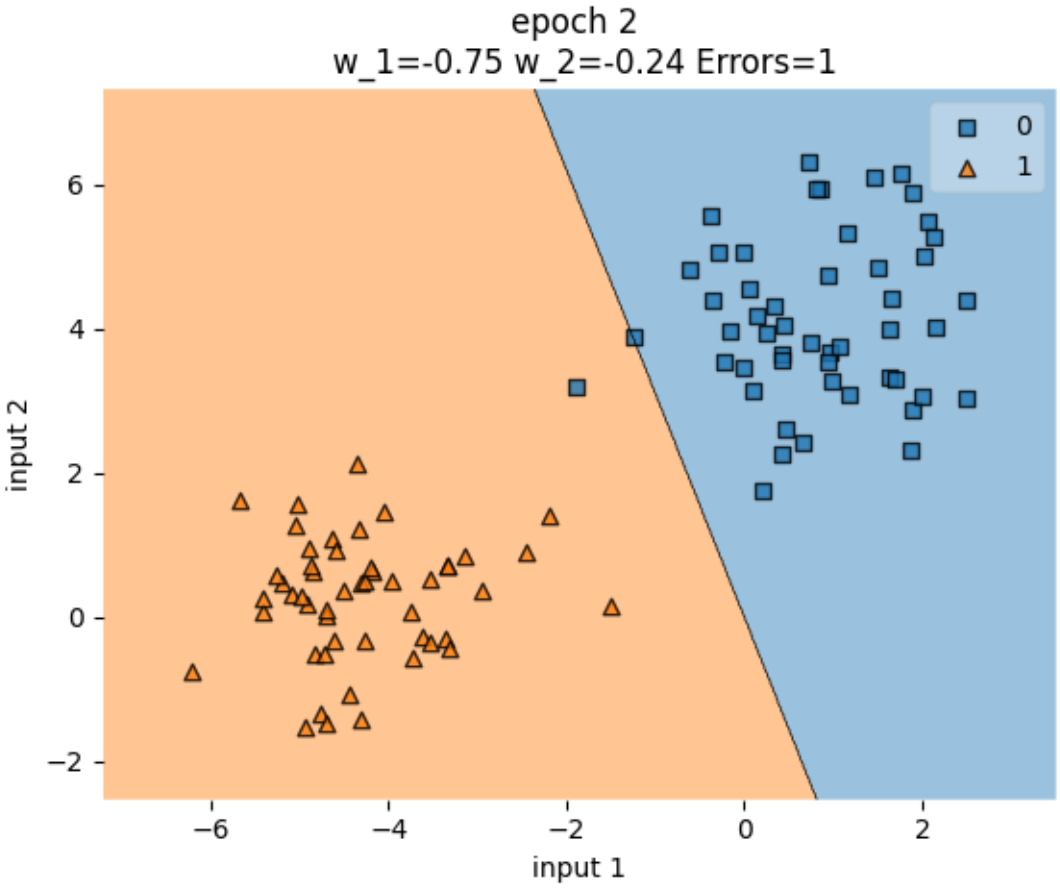

epoch 0
w_1=-0.8 w_2=-0.13 Errors=2
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

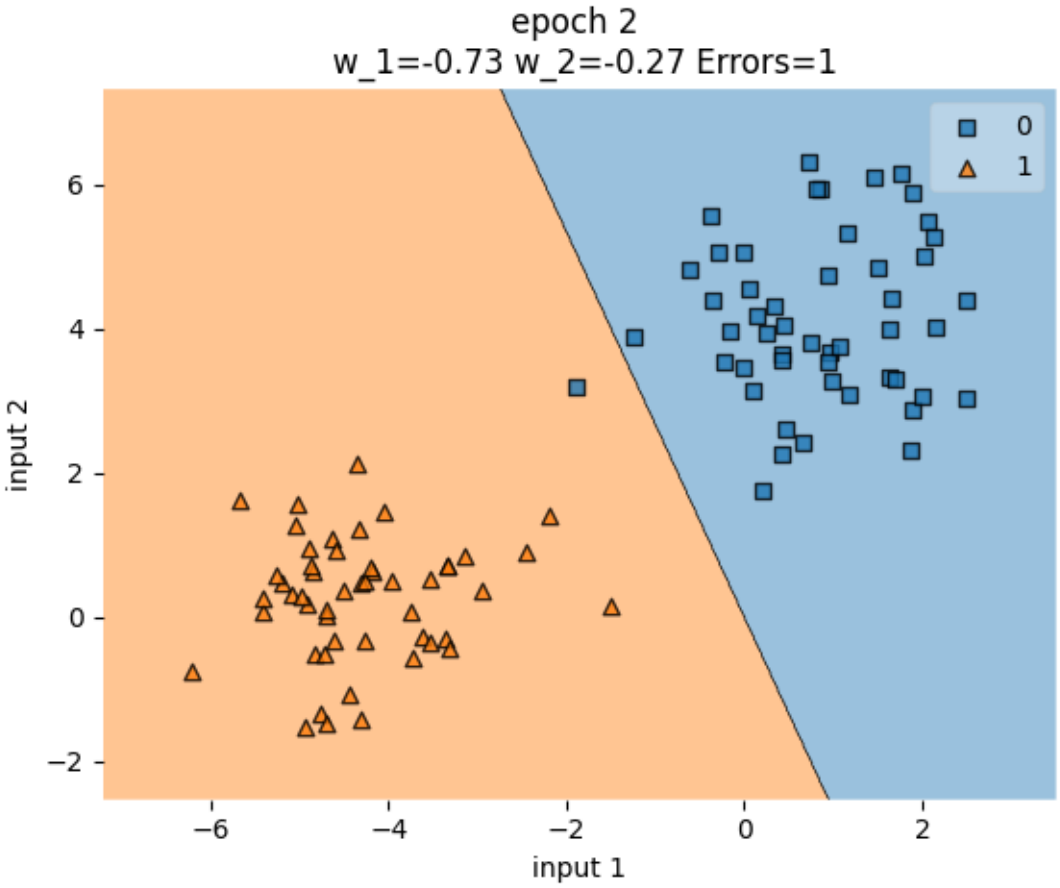

epoch 1
w_1=-0.8 w_2=-0.13 Errors=2
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

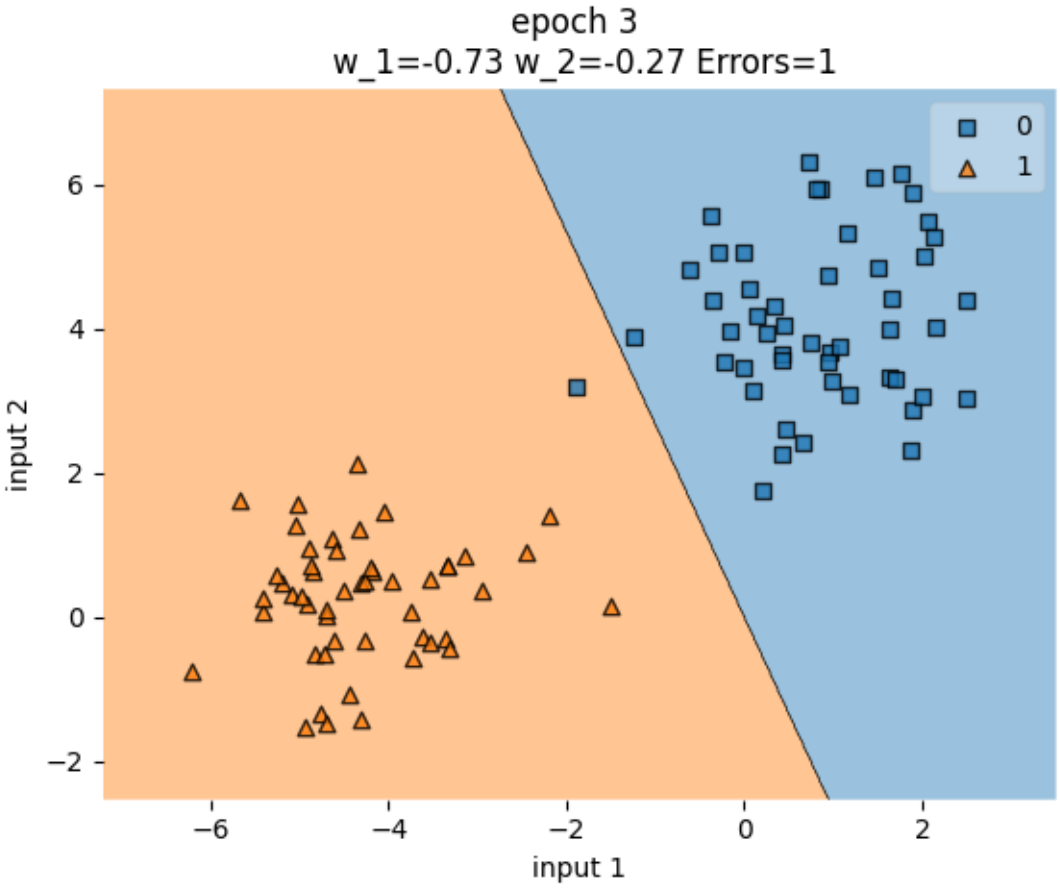

epoch 1
w_1=-0.79 w_2=-0.17 Errors=2
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

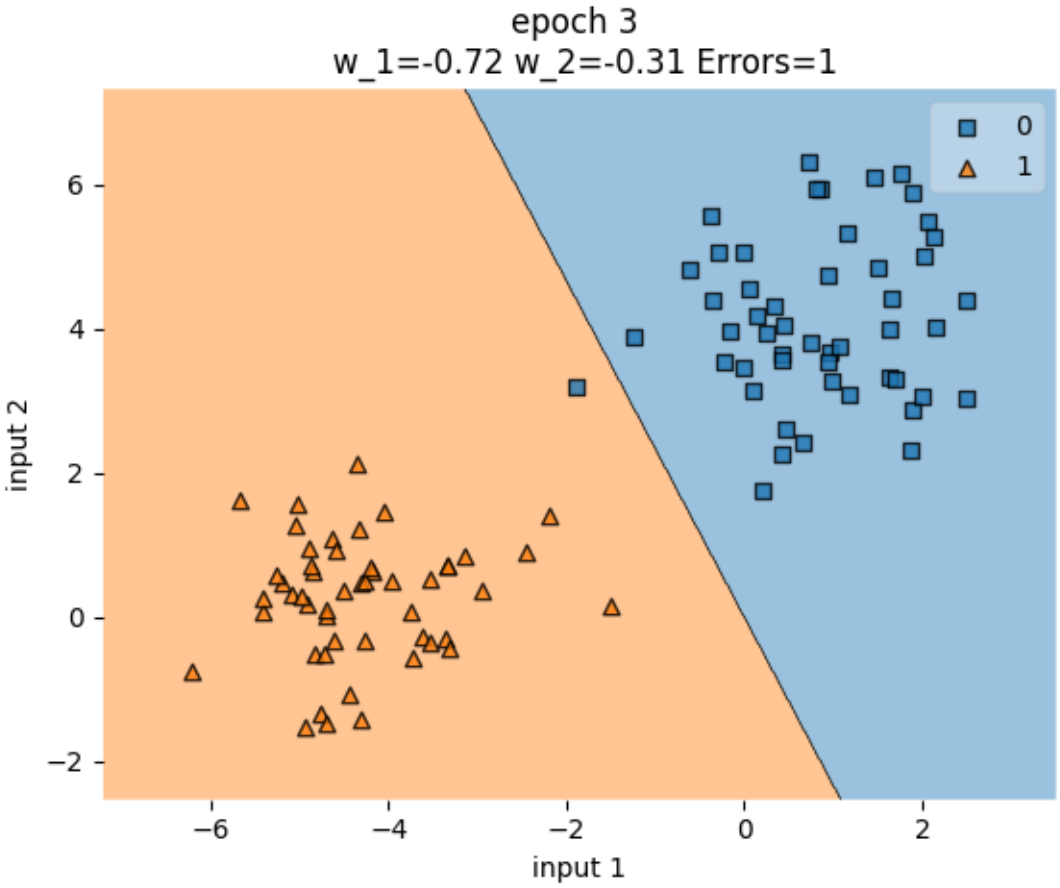

epoch 1
w_1=-0.77 w_2=-0.2 Errors=2
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

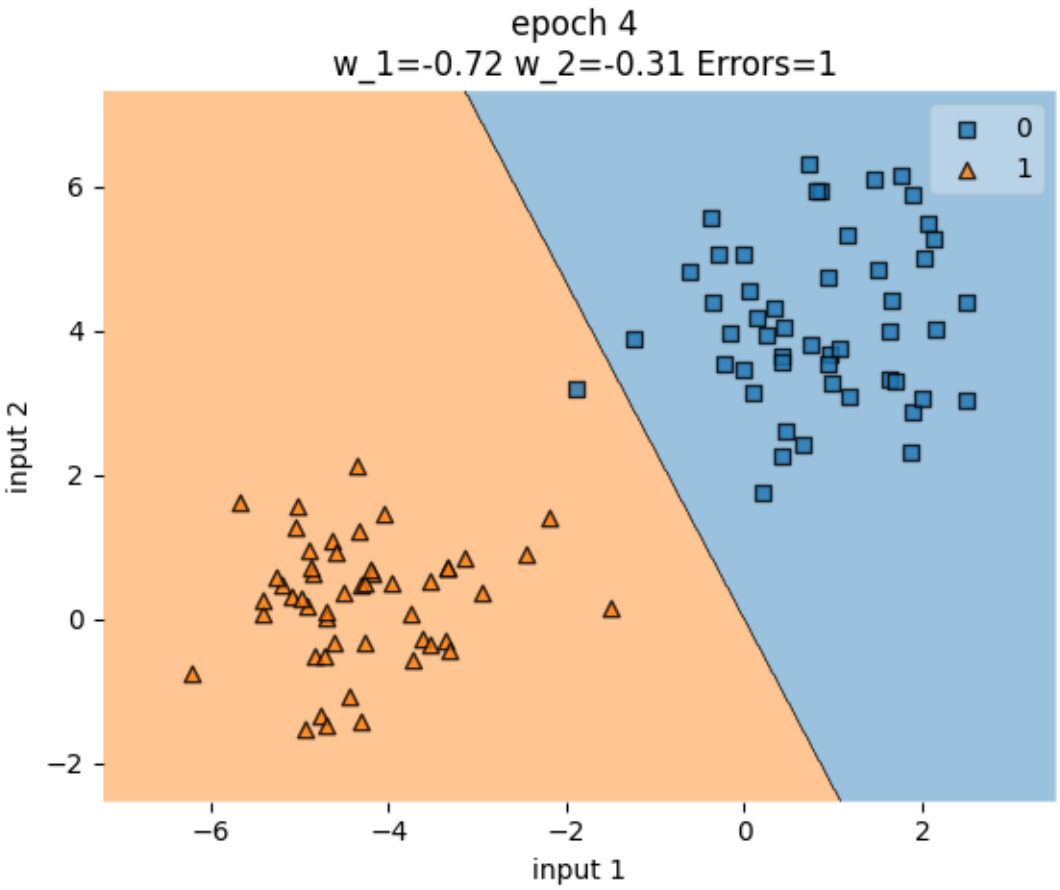

epoch 2
w_1=-0.77 w_2=-0.2 Errors=2
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

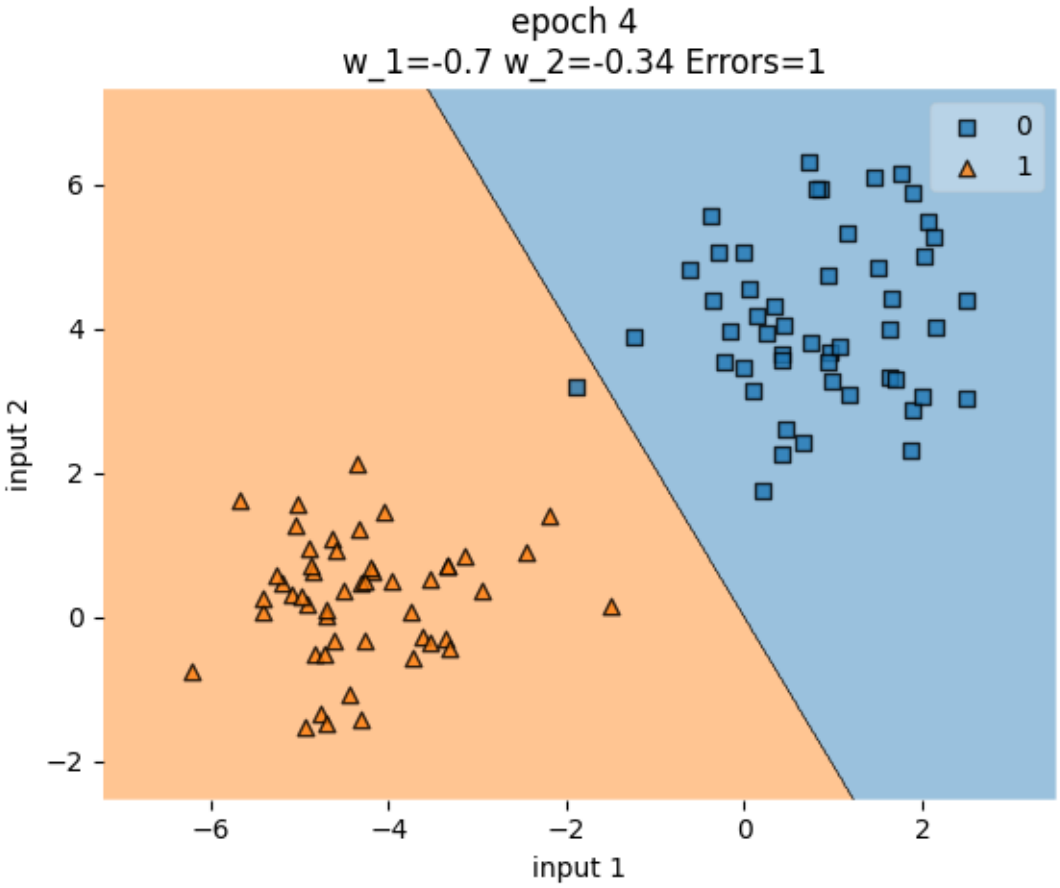

epoch 2
w_1=-0.75 w_2=-0.24 Errors=1
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

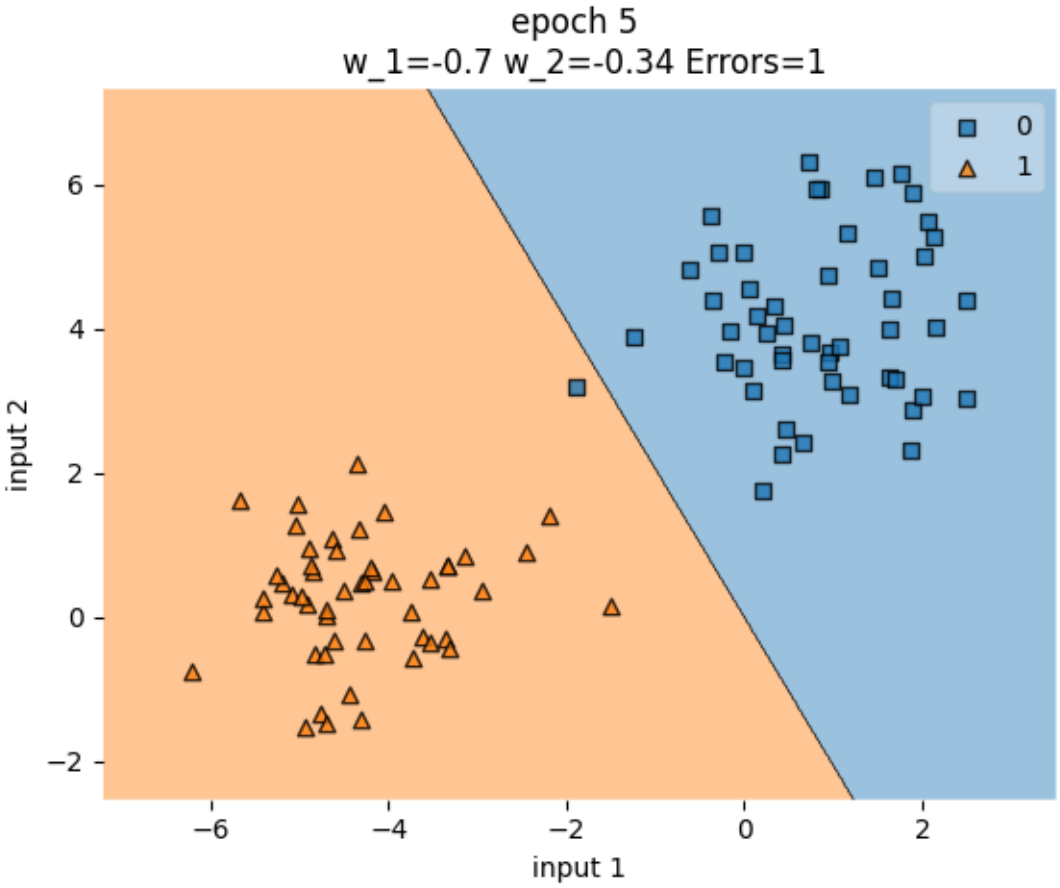

epoch 2
w_1=-0.73 w_2=-0.27 Errors=1
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

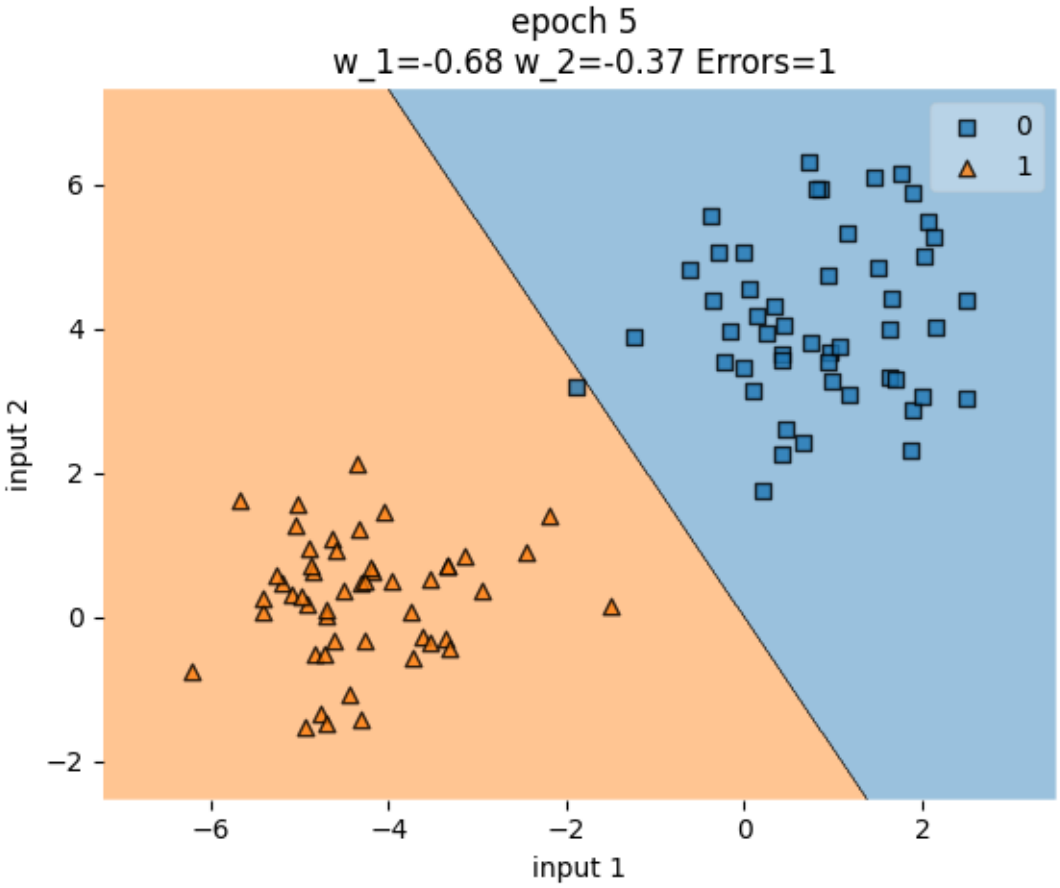

epoch 3
w_1=-0.73 w_2=-0.27 Errors=1
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

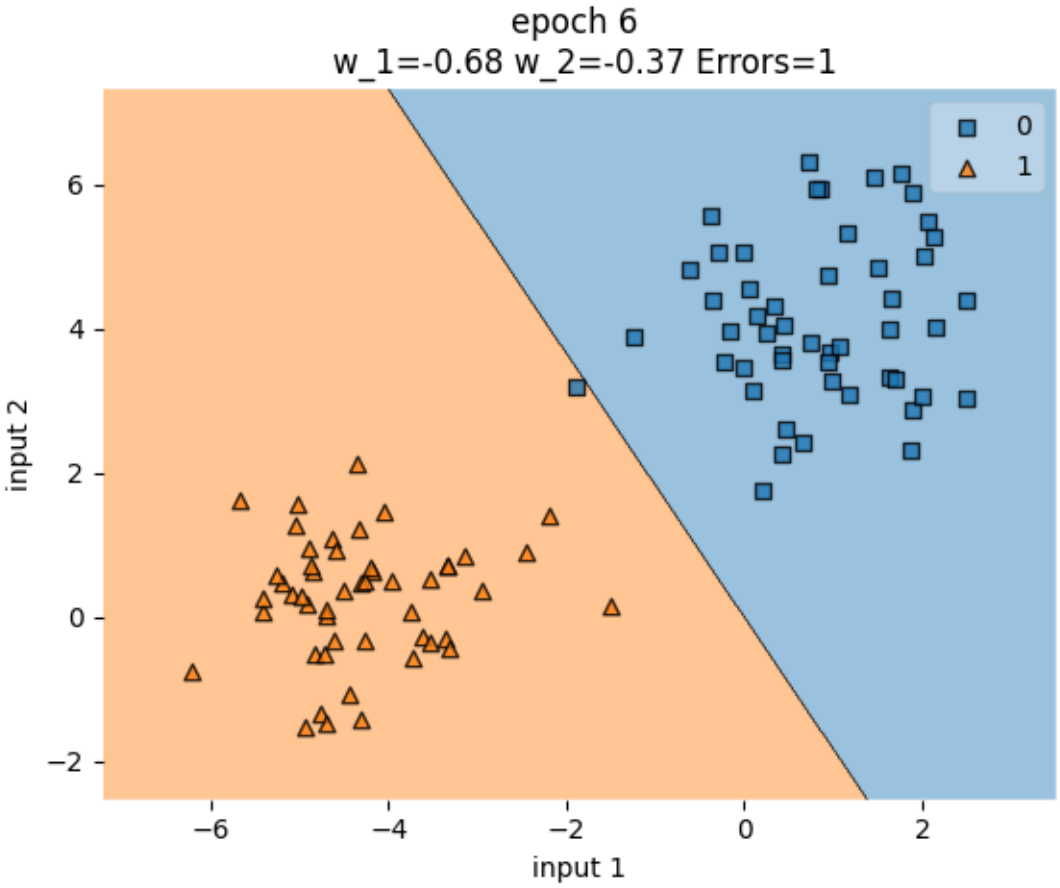

epoch 3
w_1=-0.72 w_2=-0.31 Errors=1
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

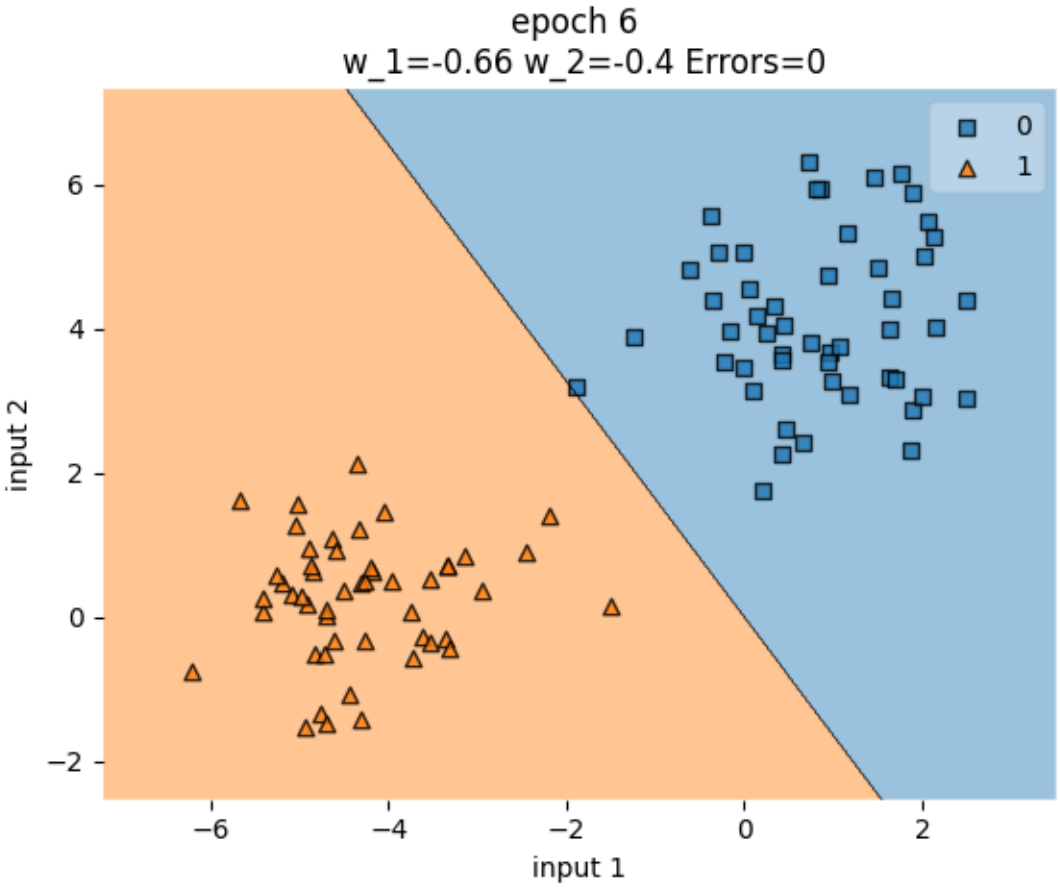

epoch 4
w_1=-0.72 w_2=-0.31 Errors=1
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

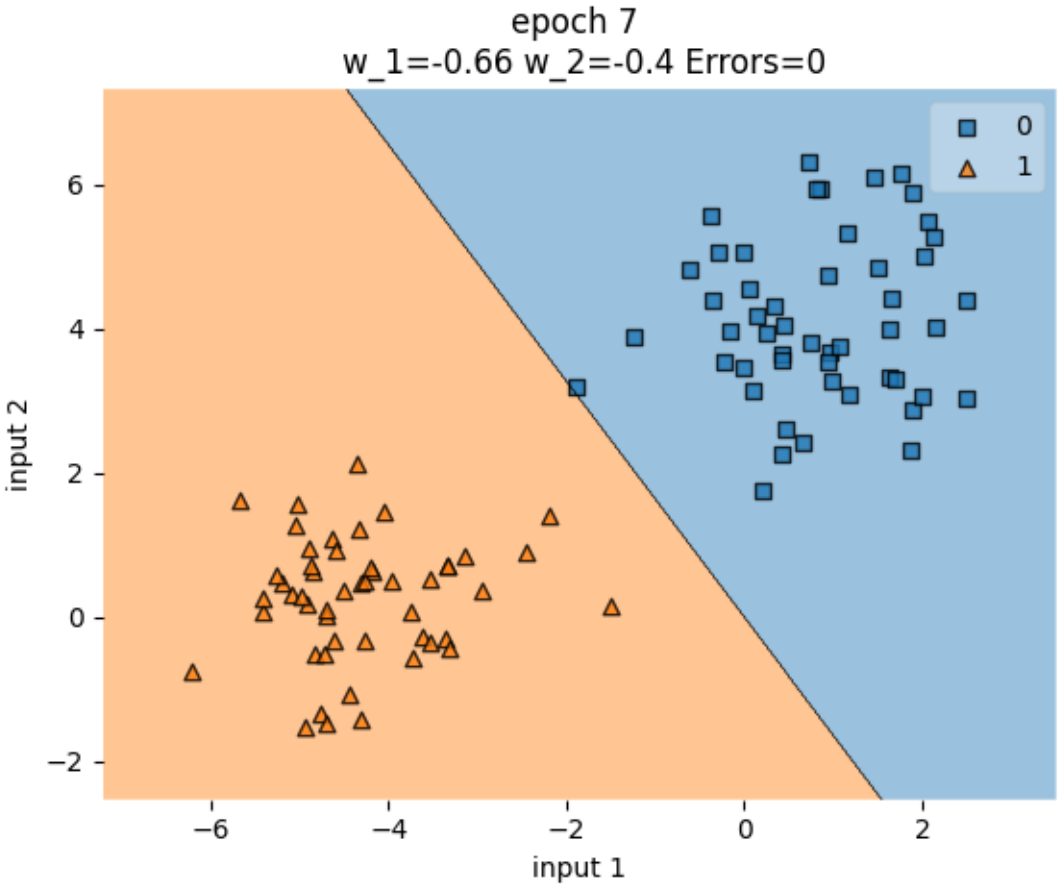

epoch 4
w_1=-0.7 w_2=-0.34 Errors=1
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

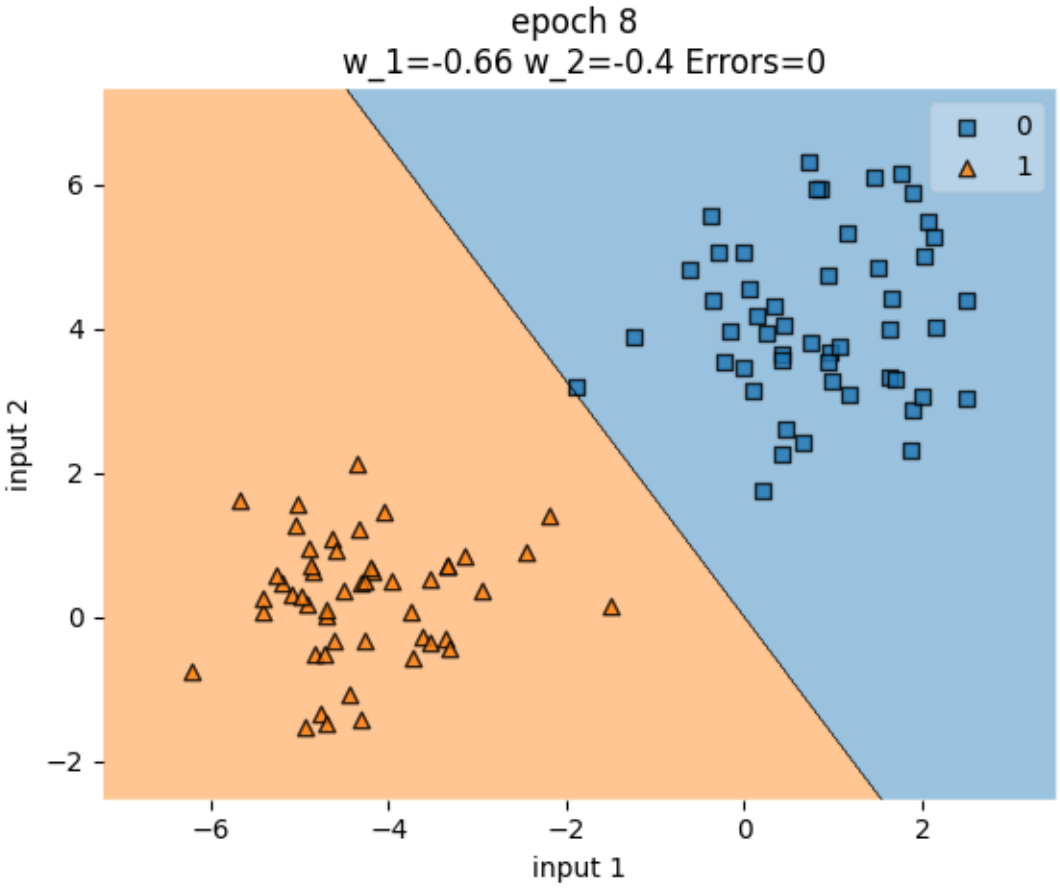

epoch 5
w_1=-0.7 w_2=-0.34 Errors=1
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1

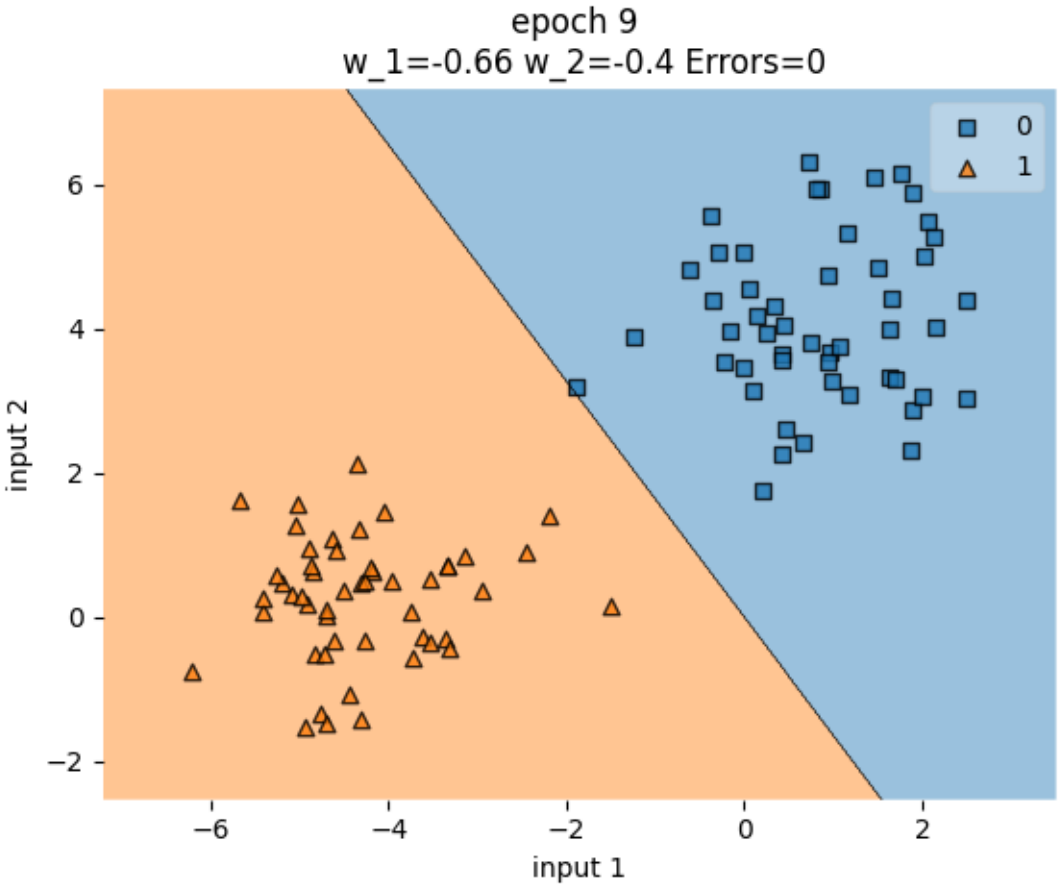

epoch 5
w_1=-0.68 w_2=-0.37 Errors=1
0
1
input 2
input 1
6
4
2
0
−2
−6
−4
−2
0
2

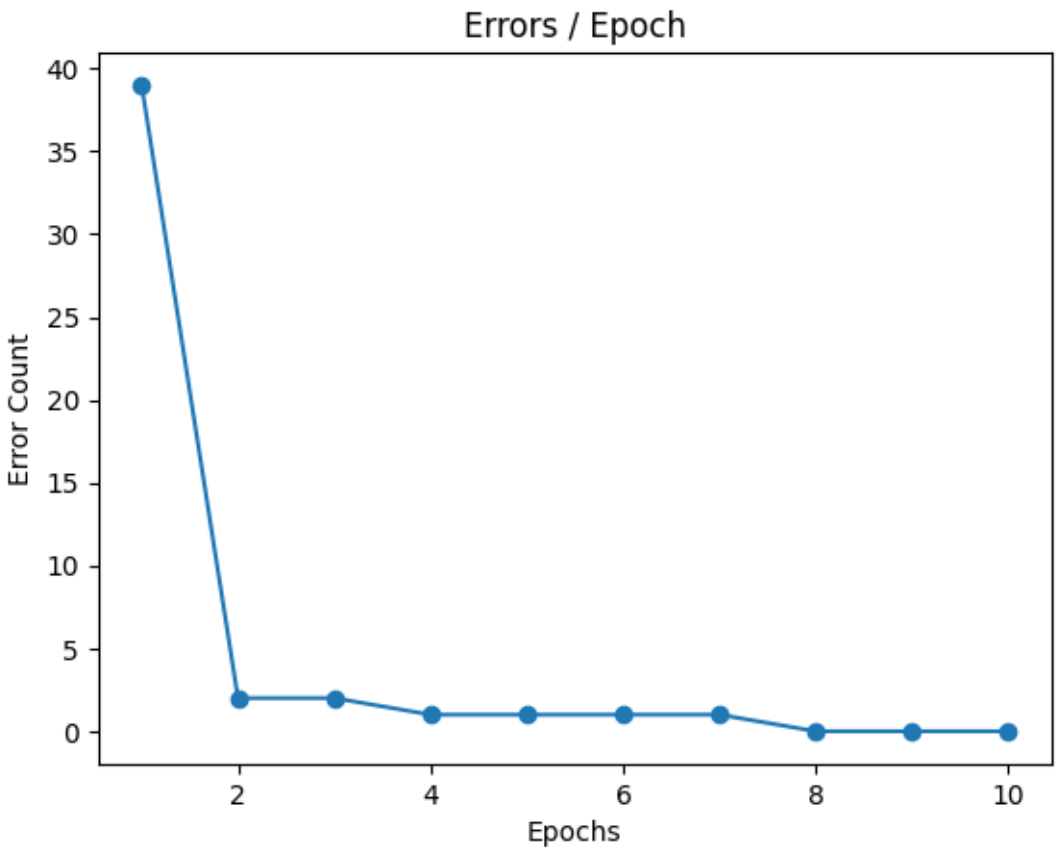

epoch 6
w_1=-0.68 w_2=-0.37 Errors=1
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1


epoch 6
w_1=-0.66 w_2=-0.4 Errors=0
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1


epoch 7
w_1=-0.66 w_2=-0.4 Errors=0
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1


epoch 8
w_1=-0.66 w_2=-0.4 Errors=0
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1


epoch 9
w_1=-0.66 w_2=-0.4 Errors=0
0
1
6
4
2
0
−2
input 2
−6
−4
−2
0
2
input 1



[9]:
```
W= np.asarray([-0.68, -0.4])
magnitude_W = np.sqrt(np.dot(W, W))
l = []
for xi, yi in zip(X, y):
    magnitude_xi = np.sqrt(np.dot(xi, xi))
    theta = round(np.arccos(np.dot(W, xi)/(magnitude_W*magnitude_xi)), 2)
    l.append((xi, round(np.degrees(theta)), round(np.cos(theta),2), round(np.
 ↪dot(W, xi),2), yi))
l.sort(key=lambda r:r[1])
l
# np.arccos(np.dot(W, np.sort(X.T))/np.linalg.norm(W)*np.linalg.norm(X))
```

[9]:
```
[(array([-4.31126623, -1.40590907]), 13, 0.98, 3.49, 1),
 (array([-4.70716987, -1.45908253]), 13, 0.97, 3.78, 1),
 (array([-4.93524002, -1.52755815]), 13, 0.97, 3.97, 1),
 (array([-4.77040684, -1.33000372]), 15, 0.97, 3.78, 1),
 (array([-4.44314571, -1.08149453]), 17, 0.96, 3.45, 1),
 (array([-3.73612426, -0.56615694]), 22, 0.93, 2.77, 1),
 (array([-6.21229878, -0.74422301]), 23, 0.92, 4.52, 1),
 (array([-3.31907313, -0.43187992]), 23, 0.92, 2.43, 1),
```

```
(array([-4.83031611, -0.52091273]), 24, 0.91, 3.49, 1),
(array([-4.71248737, -0.51397454]), 24, 0.91, 3.41, 1),
(array([-3.36709068, -0.30343965]), 25, 0.9, 2.41, 1),
(array([-3.53407542, -0.34382131]), 25, 0.91, 2.54, 1),
(array([-3.62319202, -0.26181256]), 26, 0.9, 2.57, 1),
(array([-4.61279528, -0.32371816]), 26, 0.9, 3.27, 1),
(array([-4.27714282, -0.31063568]), 26, 0.9, 3.03, 1),
(array([-4.69855315,  0.02748044]), 31, 0.86, 3.18, 1),
(array([-3.75593847,  0.06753833]), 32, 0.85, 2.53, 1),
(array([-5.41754185,  0.07576566]), 32, 0.85, 3.65, 1),
(array([-4.69303421,  0.0937087 ]), 32, 0.85, 3.15, 1),
(array([-4.9073387 ,  0.18160872]), 33, 0.84, 3.26, 1),
(array([-5.40280714,  0.27616754]), 33, 0.84, 3.56, 1),
(array([-5.08839992,  0.30496731]), 34, 0.83, 3.34, 1),
(array([-4.97781991,  0.28929763]), 34, 0.83, 3.27, 1),
(array([-4.50348991,  0.36768248]), 35, 0.82, 2.92, 1),
(array([-1.50579275,  0.14533308]), 36, 0.81, 0.97, 1),
(array([-5.2022645 ,  0.48714553]), 36, 0.81, 3.34, 1),
(array([-4.30966728,  0.46472013]), 37, 0.8, 2.74, 1),
(array([-5.25249072,  0.58742208]), 37, 0.8, 3.34, 1),
(array([-4.25934618,  0.4926206 ]), 37, 0.8, 2.7, 1),
(array([-2.94843418,  0.3655385 ]), 38, 0.79, 1.86, 1),
(array([-4.8601179 ,  0.64298718]), 38, 0.79, 3.05, 1),
(array([-3.95615496,  0.50358727]), 38, 0.79, 2.49, 1),
(array([-4.17946106,  0.64118782]), 39, 0.78, 2.59, 1),
(array([-3.53684324,  0.52961248]), 39, 0.78, 2.19, 1),
(array([-4.8718659 ,  0.70296658]), 39, 0.78, 3.03, 1),
(array([-4.20053294,  0.70008089]), 40, 0.76, 2.58, 1),
(array([-4.89793102,  0.94801784]), 41, 0.75, 2.95, 1),
(array([-3.34606132,  0.70867101]), 42, 0.74, 1.99, 1),
(array([-3.33038627,  0.70280143]), 42, 0.74, 1.98, 1),
(array([-4.59810324,  0.94120976]), 42, 0.75, 2.75, 1),
(array([-4.63916601,  1.07583611]), 44, 0.72, 2.72, 1),
(array([-5.04421371,  1.28823603]), 45, 0.71, 2.91, 1),
(array([-5.66856329,  1.62517906]), 46, 0.69, 3.2, 1),
(array([-3.1429531 ,  0.84829628]), 46, 0.7, 1.8, 1),
(array([-4.32252525,  1.21364047]), 46, 0.69, 2.45, 1),
(array([-5.01614507,  1.56154456]), 48, 0.67, 2.79, 1),
(array([-4.0538974 ,  1.45816862]), 50, 0.64, 2.17, 1),
(array([-2.45248779,  0.91060438]), 51, 0.63, 1.3, 1),
(array([-4.34999532,  2.12582891]), 57, 0.55, 2.11, 1),
(array([-2.19341554,  1.41161044]), 63, 0.45, 0.93, 1),
(array([-1.8997797 ,  3.19111595]), 90, 0.0, 0.02, 0),
(array([-1.23229972,  3.89519459]), 103, -0.23, -0.72, 0),
(array([-0.60732739,  4.8096319 ]), 113, -0.4, -1.51, 0),
(array([-0.35471096,  4.40151964]), 116, -0.43, -1.52, 0),
(array([-0.37980545,  5.55461936]), 116, -0.44, -1.96, 0),
```

```
(array([-0.22816528,  3.53653954]), 117, -0.45, -1.26, 0),
(array([-0.2979067 ,  5.04757883]), 117, -0.45, -1.82, 0),
(array([-0.16908848,  3.95730655]), 118, -0.47, -1.47, 0),
(array([-0.00897282,  5.0622949 ]), 120, -0.5, -2.02, 0),
(array([-0.00782709,  3.44996325]), 120, -0.5, -1.37, 0),
(array([0.05596187, 4.54025169]), 121, -0.51, -1.85, 0),
(array([0.09216603, 3.13908069]), 122, -0.53, -1.32, 0),
(array([0.14207575, 4.19267027]), 123, -0.54, -1.77, 0),
(array([0.2471217 , 3.93292573]), 124, -0.56, -1.74, 0),
(array([0.34123054, 4.31357332]), 125, -0.57, -1.96, 0),
(array([0.71575198, 6.32110579]), 127, -0.6, -3.02, 0),
(array([0.42487931, 3.64653908]), 127, -0.6, -1.75, 0),
(array([0.44124963, 4.05350212]), 127, -0.6, -1.92, 0),
(array([0.21219196, 1.74387328]), 127, -0.6, -0.84, 0),
(array([0.41930841, 3.57436207]), 127, -0.6, -1.71, 0),
(array([0.81854337, 5.93760148]), 128, -0.62, -2.93, 0),
(array([0.8614512 , 5.93258376]), 129, -0.63, -2.96, 0),
(array([0.4705981 , 2.61647914]), 131, -0.65, -1.37, 0),
(array([0.41307295, 2.24848441]), 131, -0.65, -1.18, 0),
(array([0.73856985, 3.80819747]), 131, -0.66, -2.03, 0),
(array([0.93621711, 4.72744197]), 132, -0.67, -2.53, 0),
(array([1.16102127, 5.33083874]), 133, -0.68, -2.92, 0),
(array([1.45390466, 6.10193491]), 134, -0.7, -3.43, 0),
(array([0.97213988, 3.68573842]), 135, -0.71, -2.14, 0),
(array([0.93321657, 3.53595578]), 135, -0.71, -2.05, 0),
(array([0.65968729, 2.41981542]), 136, -0.72, -1.42, 0),
(array([1.76101432, 6.13906724]), 136, -0.72, -3.65, 0),
(array([1.06599169, 3.75827904]), 136, -0.72, -2.23, 0),
(array([0.99185358, 3.27429903]), 138, -0.74, -1.98, 0),
(array([1.4997464 , 4.83917285]), 138, -0.74, -2.96, 0),
(array([1.89727609, 5.87252952]), 139, -0.75, -3.64, 0),
(array([2.06410556, 5.49669427]), 141, -0.78, -3.6, 0),
(array([1.16890375, 3.09876118]), 141, -0.78, -2.03, 0),
(array([1.65912133, 4.41204316]), 141, -0.78, -2.89, 0),
(array([2.12414555, 5.28234711]), 142, -0.79, -3.56, 0),
(array([1.64120302, 4.00244309]), 143, -0.8, -2.72, 0),
(array([2.02914149, 5.01575429]), 143, -0.8, -3.39, 0),
(array([1.63003514, 3.32504418]), 147, -0.84, -2.44, 0),
(array([2.13993601, 4.03104222]), 148, -0.85, -3.07, 0),
(array([1.69502965, 3.30751928]), 148, -0.85, -2.48, 0),
(array([2.50210641, 4.39967272]), 150, -0.87, -3.46, 0),
(array([1.89024377, 2.86941982]), 154, -0.9, -2.43, 0),
(array([1.99832549, 3.06188882]), 154, -0.9, -2.58, 0),
(array([1.86179146, 2.30206692]), 159, -0.94, -2.19, 0),
(array([2.50350118, 3.04465577]), 160, -0.94, -2.92, 0)]
```

[10]:
```
W = np.asarray([-0.68, -0.4])
magnitude_W = np.sqrt(np.dot(W, W))
X_ = np.asarray([[0, 2.0], [-2.0, 0.0]])
l = []
for xi in X_:
    print(xi)
    magnitude_xi = np.sqrt(np.dot(xi, xi))
    theta = round(np.arccos(np.dot(W, xi)/(magnitude_W*magnitude_xi)), 2)
    yi = 1 if np.dot(W, xi) >= 0 else 0
    l.append((xi, round(np.degrees(theta)), round(np.cos(theta),2), round(np.
 ↪dot(W, xi),2), yi))
l.sort(key=lambda r:r[1])
l
```

```
[0. 2.]
[-2.  0.]
```

[10]:
```
[(array([-2.,  0.]), 30, 0.86, 1.36, 1), (array([0., 2.]), 120, -0.5, -0.8, 0)]
```