Numerical Methods and Machine Learning for Image Processing

Week 6, Class 2: Basic Neural Nets, Part 1b

November 13, 2021

Damon M. Chandler and Yi Zhang

Last time: Basic Neural Nets, Part 1a

- **Gradient descent**
- **Perceptron model**
  1. Background and history of neural networks
  2. Perceptron model math

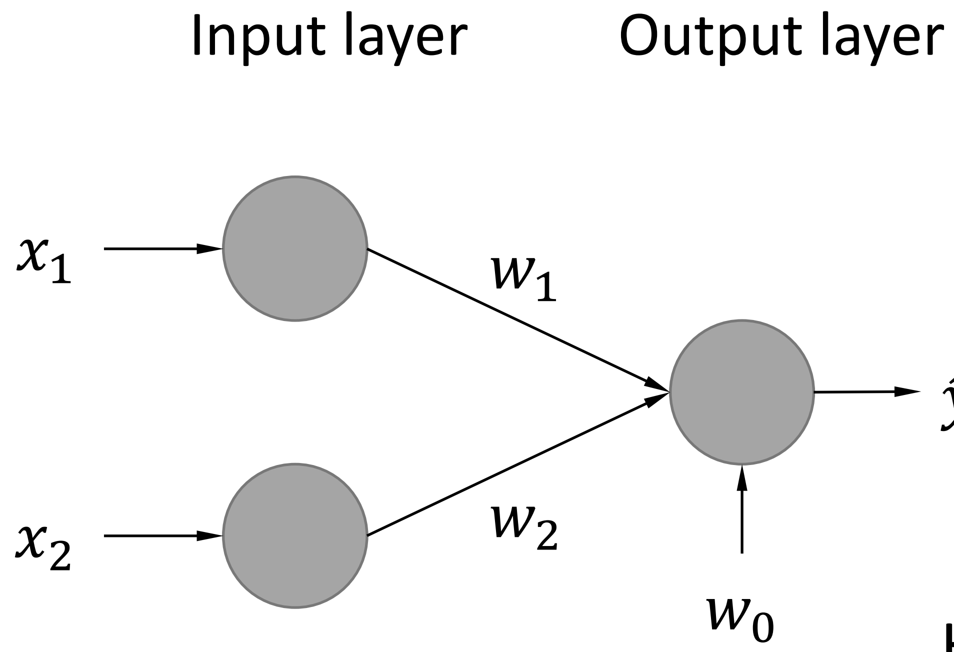
Today: Basic Neural Nets, Part 1b

- **Perceptron model**
  1. Perceptron code
- **Multilayer Perceptron model**
  1. MLP math
  2. MLP code

Today: Basic Neural Nets, Part 1b

- **Perceptron model**
  1. Perceptron code
- **Multilayer Perceptron model**
  1. MLP math
  2. MLP code

Perceptron (1958)



$$\hat{y} = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

How to optimize the weights?

1. Compute error: $E = \frac{1}{2}(y - \hat{y})^2$

2. Adjust w_0, w_1, and w_2 based on the error gradient.

$$E = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}\left(y - \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}\right)^2$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_i} = -(y - \hat{y}) \frac{\partial \hat{y}}{\partial w_i} = -(y - \hat{y}) x_i \hat{y}(1 - \hat{y}) \qquad (x_0 = 1)$$

$$\frac{\partial E}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}\left\{\frac{1}{2}(y - \hat{y})^2\right\} = 2\frac{1}{2}(y - \hat{y}) \cdot (-1) = -(y - \hat{y})$$

$$\frac{\partial \hat{y}}{\partial w_0} = \frac{1 e^{-(w_0 + w_1 x_1 + w_2 x_2)}}{(1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)})^2} = \hat{y}(1 - \hat{y})$$

$$\frac{\partial \hat{y}}{\partial w_1} = \frac{x_1 e^{-(w_0 + w_1 x_1 + w_2 x_2)}}{(1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)})^2} = x_1 \hat{y}(1 - \hat{y})$$

$$\frac{\partial \hat{y}}{\partial w_2} = \frac{x_2 e^{-(w_0 + w_1 x_1 + w_2 x_2)}}{(1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)})^2} = x_2 \hat{y}(1 - \hat{y})$$

(Showing how the derivatives on the previous slide were computed.)

$$E = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}\left(y - \frac{1}{1 + e^{-(w_0+w_1x_1+w_2x_2)}}\right)^2$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_i} = -(y - \hat{y})\frac{\partial \hat{y}}{\partial w_i}$$

$$\frac{\partial \hat{y}}{\partial w_i} = \frac{\partial}{\partial w_i}\left\{\frac{1}{1 + e^{-(w_0+w_1x_1+w_2x_2)}}\right\}$$

Note: $\frac{d}{dw}\left(\frac{1}{1 + \beta e^{-\alpha w}}\right) = \frac{\alpha\beta e^{-\alpha w}}{(1 + \beta e^{-\alpha w})^2}$

$$\frac{\partial \hat{y}}{\partial w_0} = \frac{1e^{-(w_0+w_1x_1+w_2x_2)}}{(1 + e^{-(w_0+w_1x_1+w_2x_2)})^2}$$

Note: $\frac{a}{(1+a)^2} = \frac{1}{1+a}\left(1 - \frac{1}{1+a}\right)$

$$= \frac{1}{1 + e^{-(w_0+w_1x_1+w_2x_2)}}\left(1 - \frac{1}{1 + e^{-(w_0+w_1x_1+w_2x_2)}}\right) = \hat{y}(1 - \hat{y})$$

$$\frac{\partial \hat{y}}{\partial w_1} = \frac{x_1 e^{-(w_0+w_1x_1+w_2x_2)}}{(1 + e^{-(w_0+w_1x_1+w_2x_2)})^2} = x_1\hat{y}(1 - \hat{y})$$

$$\frac{\partial \hat{y}}{\partial w_2} = \frac{x_2 e^{-(w_0+w_1x_1+w_2x_2)}}{(1 + e^{-(w_0+w_1x_1+w_2x_2)})^2} = x_2\hat{y}(1 - \hat{y})$$

$$\frac{\partial E}{\partial w_i} = -(y - \hat{y})x_i\hat{y}(1 - \hat{y}) \qquad (x_0 = 1)$$

<u>How to change the weights (*update rule*):</u>

$$w_i = w_i - \eta\frac{\partial E}{\partial w_i} = w_i + \eta(y - \hat{y})x_i\hat{y}(1 - \hat{y})$$

```
    for X_row, y_row in zip(X, y):
       hat_y_row = neural_response(X_row, w)

       err = y_row - hat_y_row
       sse += err**2

       delta = err*hat_y_row*(1.0 - hat_y_row)

       w[0] = w[0] + lrate*delta
       for i in range(len(X_row)):
         w[i+1] = w[i+1] + lrate*delta*X_row[i]
```

X

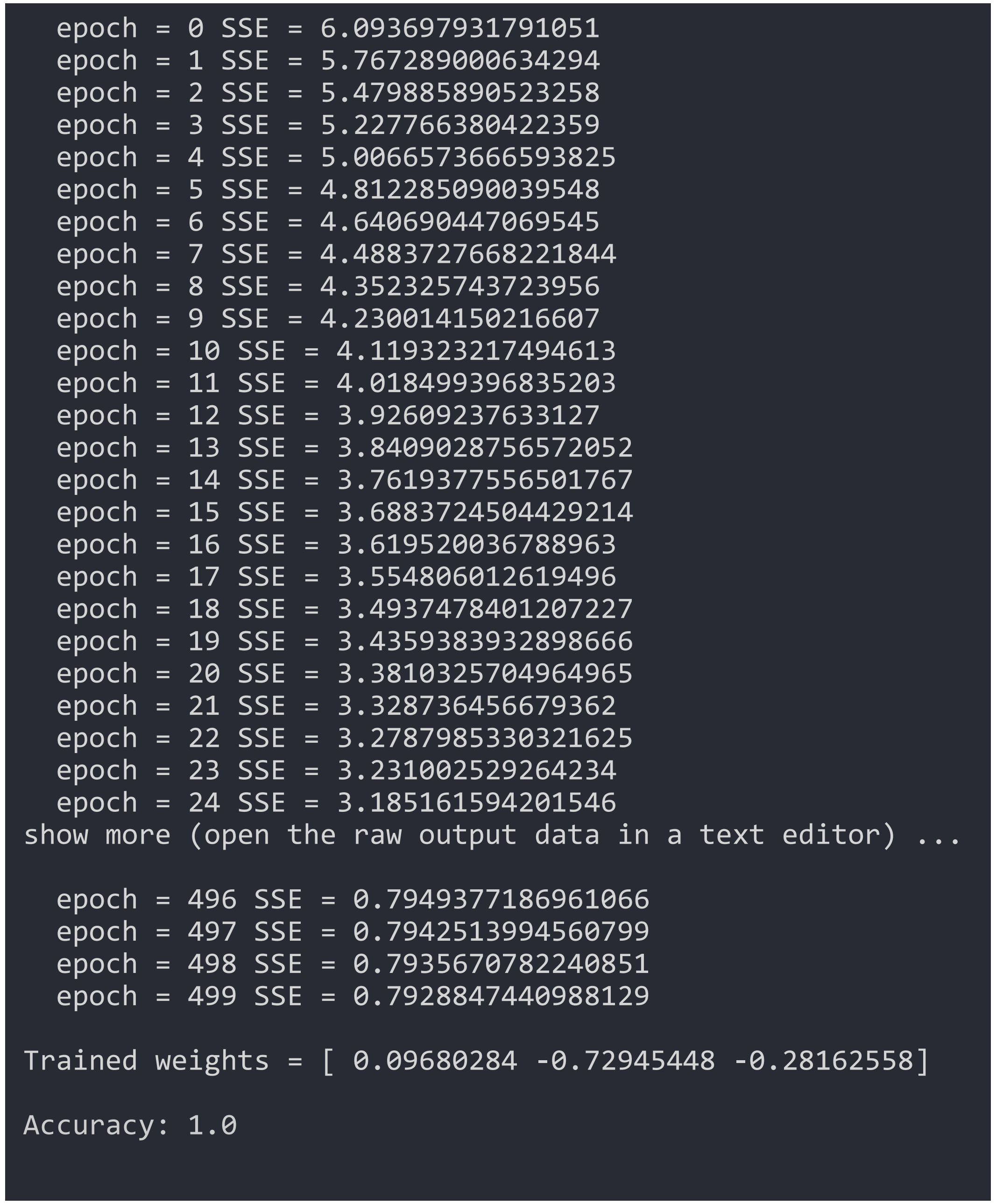
X[:,0]	X[:,1]	X[:,2]	X[:,3]	y
wav_var	wav_skw	wav_krt	pix_ent	class
0.964	5.616	2.214	-0.125	0
0.259	5.010	-5.039	-6.386	1
0.331	4.573	2.057	-0.190	0
-0.531	-0.097	-0.218	1.043	1
-3.137	0.422	2.623	-0.064	1
-7.042	9.200	0.259	-4.683	1
3.184	7.232	-1.071	-2.591	0
-1.119	3.336	-1.346	-1.957	1
-0.234	3.241	-3.067	-2.778	1
-1.279	-2.409	4.574	0.476	1
-2.410	3.743	-0.402	-1.295	1
-0.394	-0.021	-0.066	-0.447	1
-2.380	-1.440	1.127	0.161	1
3.776	7.178	-1.520	0.401	0

Perceptron model on banknote (tiny subset) data (2 features only)

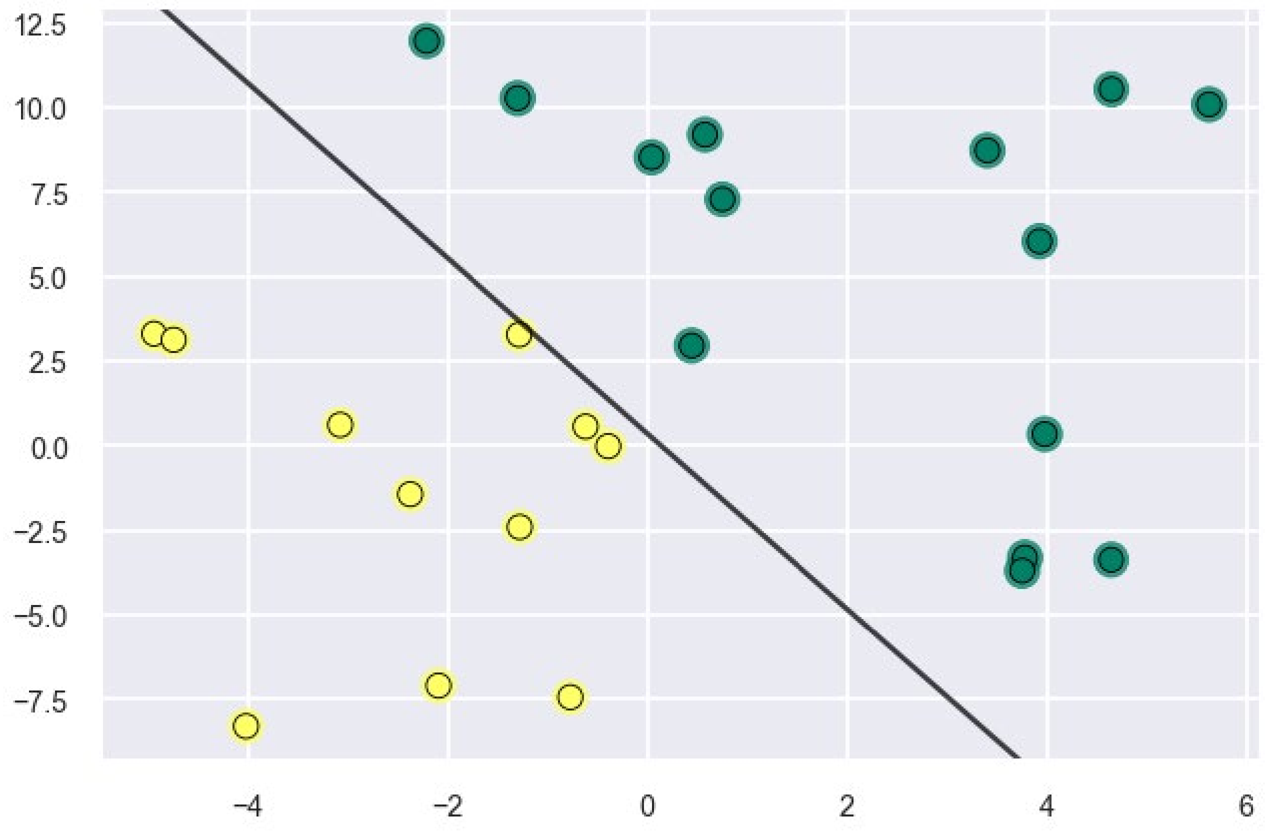
```python
from sklearn.metrics import accuracy_score

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt


#%% PERCEPTRON CODE

# computes the neuron's output
def neural_response(x, w):
  res = w[0] # bias weight (w0)
  for i in range(len(x)):
    res += w[i+1]*x[i] # weighted input sum (w1*x1 + w2*x2 + ...)

  #return 1 if res >= 0 else 0
  return 1 / (1 + np.exp(-res))


# adjusts the weights via training
def train_neuron(X, y, lrate, num_epochs):
  w = np.zeros((X.shape[1]+1))

  for epoch in range(num_epochs):
    sse = 0.0

    for X_row, y_row in zip(X, y):
      hat_y_row = neural_response(X_row, w)

      err = y_row - hat_y_row
      sse += err**2

      delta = err*hat_y_row*(1.0 - hat_y_row)

      w[0] = w[0] + lrate*delta
      for i in range(len(X_row)):
        w[i+1] = w[i+1] + lrate*delta*X_row[i]

    print("  epoch =", epoch, "SSE =", sse)

  return w


#%% LOAD THE DATA

data = pd.read_excel("banknote.xlsx", sheet_name="tiny")
print(data.shape)
data.head()
```

```python
features = data.drop(["class"], axis=1)
X = np.array(features)
y = np.array(data["class"])


#%% TRAINING

lrate = 0.001
num_epochs = 500

w = train_neuron(X, y, lrate, num_epochs)
print("Trained weights =", w)


#%% SHOW THE RESULTS

hat_y = np.zeros(y.shape)
for r in range(X.shape[0]):
  res = neural_response(X[r], w)
  hat_y[r] = 1 if res >= 0.5 else 0

print("Accuracy: ", accuracy_score(y_true=y, y_pred=hat_y))


#%% PLOT THE DECISION CUT LINE

plt.scatter(X[:, 0], X[:, 1], c=hat_y, s=100,
  cmap='summer', linewidths=0.5, alpha=0.75)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50,
  cmap='summer', edgecolors='black', linewidths=0.5)

ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

xs = np.linspace(-6, 6, 2)
ys = -(w[1]*xs + w[0])/w[2]
plt.plot(xs, ys, color="black", alpha=0.75)

ax.set_xlim(xlim)
ax.set_ylim(ylim)
plt.show()
```

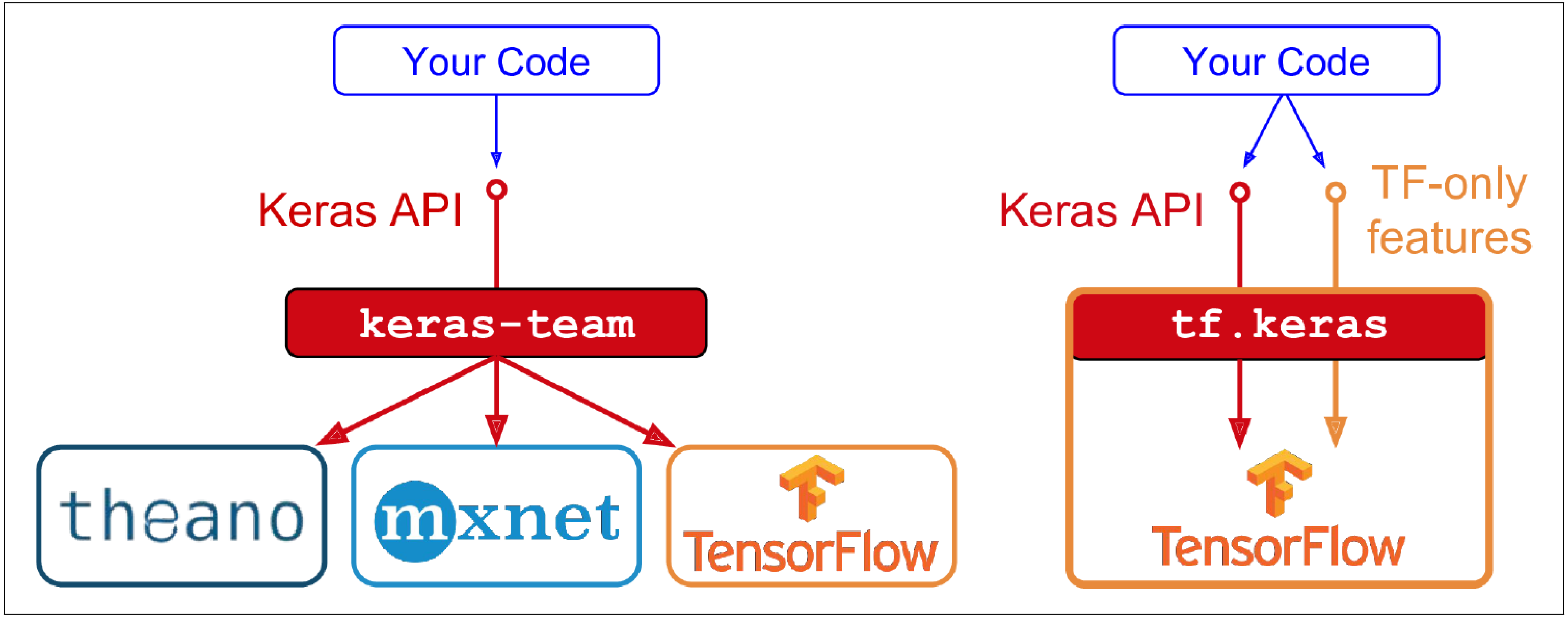
Perceptron model on banknote (tiny subset) data (2 features only)



```
  epoch = 0 SSE = 6.093697931791051
  epoch = 1 SSE = 5.767289000634294
  epoch = 2 SSE = 5.479885890523258
  epoch = 3 SSE = 5.227766380422359
  epoch = 4 SSE = 5.0066573666593825
  epoch = 5 SSE = 4.812285090039548
  epoch = 6 SSE = 4.640690447069545
  epoch = 7 SSE = 4.4883727668221844
  epoch = 8 SSE = 4.352325743723956
  epoch = 9 SSE = 4.230014150216607
  epoch = 10 SSE = 4.119323217494613
  epoch = 11 SSE = 4.018499396835203
  epoch = 12 SSE = 3.92609237633127
  epoch = 13 SSE = 3.8409028756572052
  epoch = 14 SSE = 3.7619377556501767
  epoch = 15 SSE = 3.6883724504429214
  epoch = 16 SSE = 3.619520036788963
  epoch = 17 SSE = 3.554806012619496
  epoch = 18 SSE = 3.4937478401207227
  epoch = 19 SSE = 3.4359383932898666
  epoch = 20 SSE = 3.3810325704964965
  epoch = 21 SSE = 3.328736456679362
  epoch = 22 SSE = 3.2787985330321625
  epoch = 23 SSE = 3.231002529264234
  epoch = 24 SSE = 3.185161594201546
show more (open the raw output data in a text editor) ...

  epoch = 496 SSE = 0.7949377186961066
  epoch = 497 SSE = 0.7942513994560799
  epoch = 498 SSE = 0.7935670782240851
  epoch = 499 SSE = 0.7928847440988129

Trained weights = [ 0.09680284 -0.72945448 -0.28162558]

Accuracy: 1.0
```



Keras and Tensorflow



How to install TensorFlow and Keras

1. Create a new Anaconda environment with Keras and TensorFlow:

```
conda create --name tfkeras_gpu keras-gpu tensorflow-gpu
```

2. Switch to the new environment:

```
conda activate tfkeras_gpu
```

3. Ensure that it is installed properly:

```
import tensorflow as tf
from tensorflow import keras
print(tf.__version__)
print(keras.__version__)
```

4. Install other usual packages:

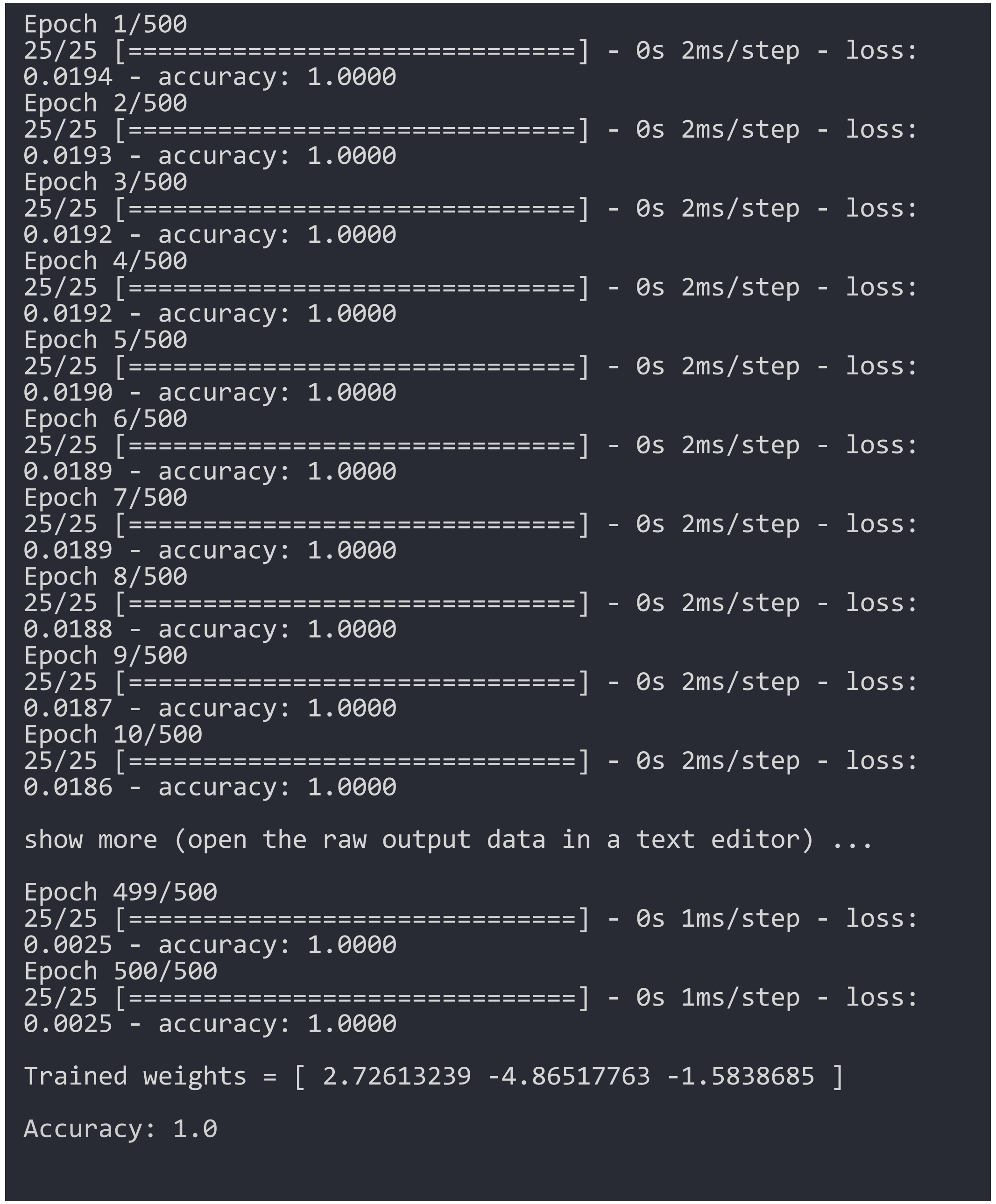
```
conda install matplotlib pandas scikit-learn ipython ipykernel
conda install -c conda-forge opencv
```

Perceptron model on banknote (tiny subset) data (2 features only)

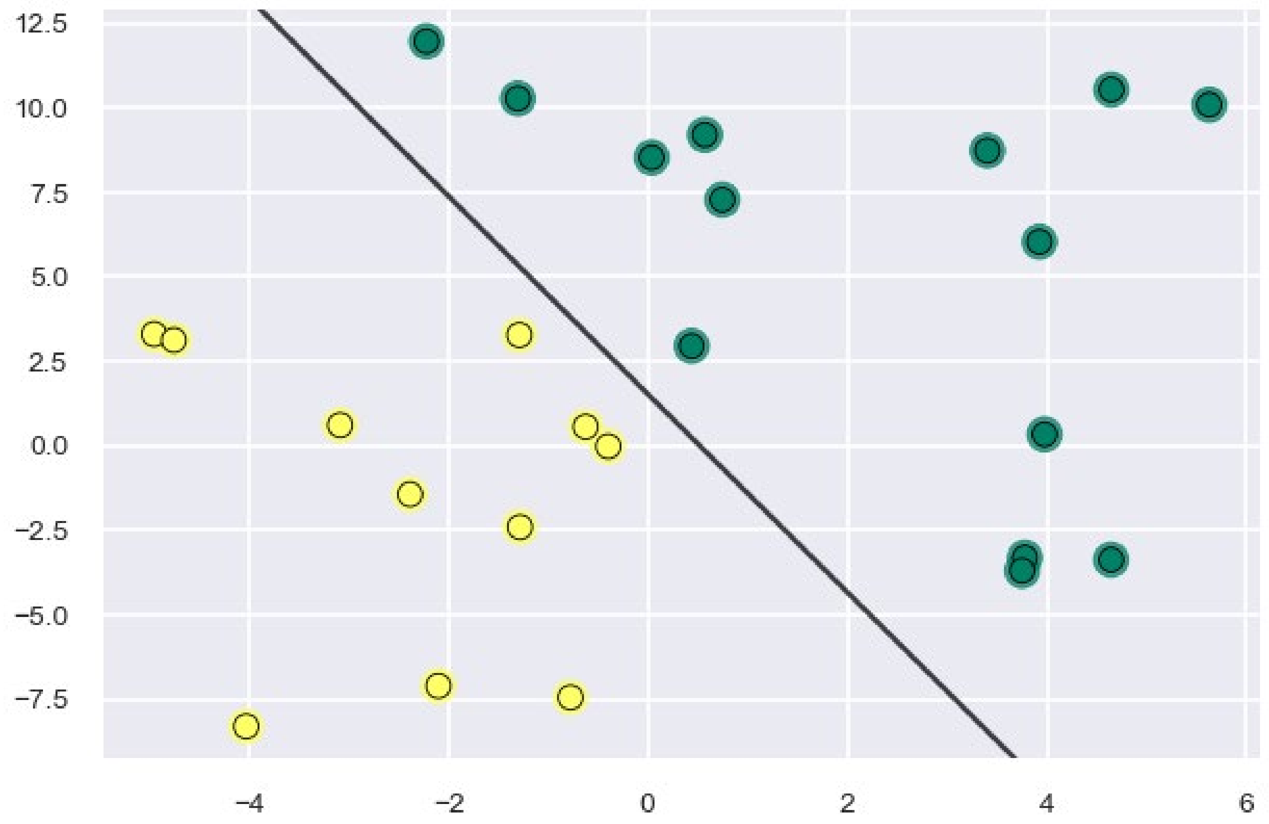
```python
from sklearn.metrics import accuracy_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


#%% LOAD THE DATA
data = pd.read_excel("banknote.xlsx", sheet_name="tiny")
print(data.shape)
data.head()

features = data.drop(["class"], axis=1)
X = np.array(features)
y = np.array(data["class"])


#%%  PERCEPTRON
# define the model
model = Sequential()
model.add(Dense(1, input_shape=(2,), activation="sigmoid"))

# compile the keras model
model.compile(loss='binary_crossentropy',
  optimizer='adam', metrics=['accuracy'])

# show the model info
model.summary()


#%% TRAINING
# fit the model to the dataset
model.fit(X, y, epochs=500, batch_size=1)


#%% SHOW THE RESULTS
hat_y = model.predict_classes(X)
print("Accuracy: ", accuracy_score(y_true=y, y_pred=hat_y))
```

```python
print("")

print("")
print("Classification Report")
print(classification_report(y, hat_y))

cm = confusion_matrix(y, hat_y)
sns.set(font_scale=0.75)
sns.heatmap(cm.T, square=True, annot=True, fmt='d',
  cbar=False, linewidths=0.5)
plt.xlabel('GT label')
plt.ylabel('Predicted label')

plt.scatter(X[:, 0], X[:, 1], c=hat_y, s=100,
  cmap='summer', linewidths=0.5, alpha=0.75)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50,
  cmap='summer', edgecolors='black', linewidths=0.5)

ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

weights = model.get_weights()
w = np.empty((3,))
w[0] = weights[1][0]
w[1] = weights[0][0][0]
w[2] = weights[0][1][0]

xs = np.linspace(-6, 6, 2)
ys = -(w[1]*xs + w[0])/w[2]
plt.plot(xs, ys, color="black", alpha=0.75)

ax.set_xlim(xlim)
ax.set_ylim(ylim)
plt.show()
```

Perceptron model on banknote (tiny subset) data (2 features only)

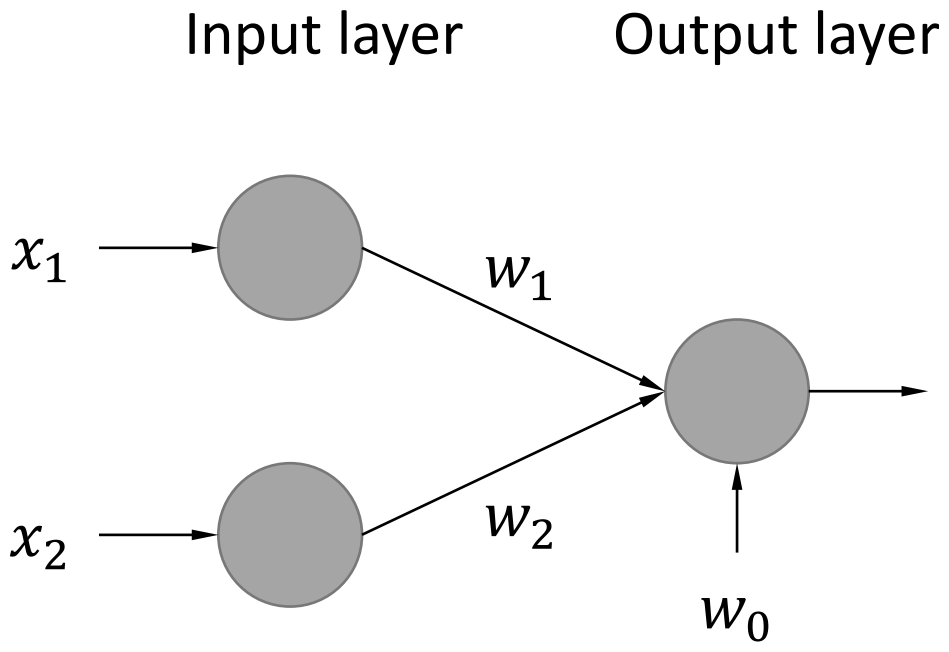


```
Epoch 1/500
25/25 [==============================] - 0s 2ms/step - loss:
0.0194 - accuracy: 1.0000
Epoch 2/500
25/25 [==============================] - 0s 2ms/step - loss:
0.0193 - accuracy: 1.0000
Epoch 3/500
25/25 [==============================] - 0s 2ms/step - loss:
0.0192 - accuracy: 1.0000
Epoch 4/500
25/25 [==============================] - 0s 2ms/step - loss:
0.0192 - accuracy: 1.0000
Epoch 5/500
25/25 [==============================] - 0s 2ms/step - loss:
0.0190 - accuracy: 1.0000
Epoch 6/500
25/25 [==============================] - 0s 2ms/step - loss:
0.0189 - accuracy: 1.0000
Epoch 7/500
25/25 [==============================] - 0s 2ms/step - loss:
0.0189 - accuracy: 1.0000
Epoch 8/500
25/25 [==============================] - 0s 2ms/step - loss:
0.0188 - accuracy: 1.0000
Epoch 9/500
25/25 [==============================] - 0s 2ms/step - loss:
0.0187 - accuracy: 1.0000
Epoch 10/500
25/25 [==============================] - 0s 2ms/step - loss:
0.0186 - accuracy: 1.0000

show more (open the raw output data in a text editor) ...

Epoch 499/500
25/25 [==============================] - 0s 1ms/step - loss:
0.0025 - accuracy: 1.0000
Epoch 500/500
25/25 [==============================] - 0s 1ms/step - loss:
0.0025 - accuracy: 1.0000

Trained weights = [ 2.72613239 -4.86517763 -1.5838685 ]

Accuracy: 1.0
```



Today: Basic Neural Nets, Part 1b

- **Perceptron model**
  1. Perceptron code
- **Multilayer Perceptron model**
  1. MLP math
  2. MLP code



$$\hat{y} = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

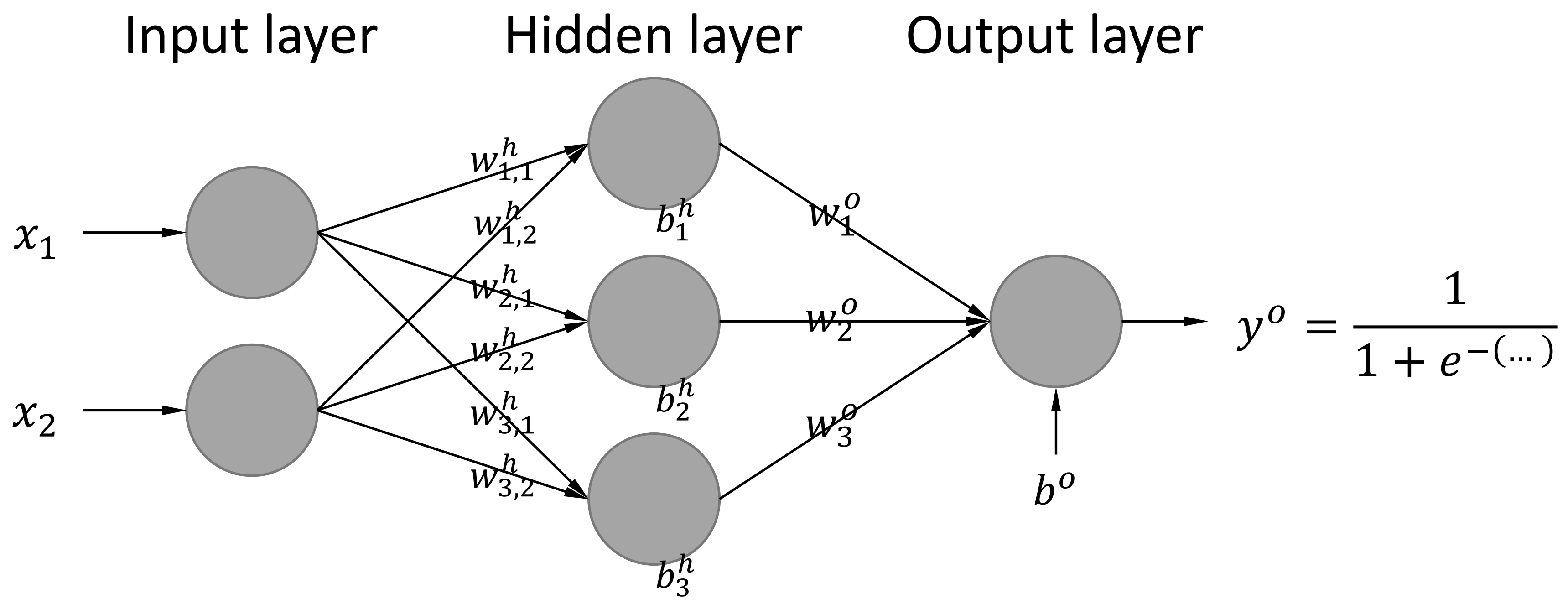
1. Compute error:

$$E = \frac{1}{2}(y - \hat{y})^2$$

2. Adjust w_0, w_1, and w_2 based on the error gradient.

Multi-layer Perceptron (MLP)

MLP with a single output



$$y^o = \frac{1}{1+e^{-\left(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h\right)}}$$

$$y_1^h = \frac{1}{1+e^{-\left(b_1^h + w_{1,1}^h x_1 + w_{1,2}^h x_2\right)}}$$

$$y_2^h = \frac{1}{1+e^{-\left(b_2^h + w_{2,1}^h x_1 + w_{2,2}^h x_2\right)}}$$

$$y_3^h = \frac{1}{1+e^{-\left(b_3^h + w_{3,1}^h x_1 + w_{3,2}^h x_2\right)}}$$

MLP with a single output

i = input node index
j = hidden node index

$$E = \frac{1}{2}(y - y^o)^2 = \frac{1}{2}\left(y - \frac{1}{1 + e^{-\left(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h\right)}}\right)^2$$

$$\frac{\partial E}{\partial w_j^o} = \frac{\partial E}{\partial y^o}\frac{\partial y^o}{\partial w_j^o} = \ ?$$

$$\frac{\partial E}{\partial y^o} = \frac{\partial}{\partial y^o}\left\{\frac{1}{2}(y - y^o)^2\right\} = 2\,\frac{1}{2}(y - y^o)\cdot(-1) = -(y - y^o)$$

$$\frac{\partial y^o}{\partial w_1^o} = \frac{\partial}{\partial w_1^o}\left(\frac{1}{1 + e^{-\left(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h\right)}}\right) = \frac{y_1^h e^{-\left(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h\right)}}{\left(1 + e^{-\left(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h\right)}\right)^2} = y_1^h y^o (1 - y^o)$$

Note: $$\frac{d}{dw}\left(\frac{1}{1 + \beta e^{-\alpha w}}\right) = \frac{\alpha\beta e^{-\alpha w}}{(1 + \beta e^{-\alpha w})^2}$$

Note: $$\frac{a}{(1 + a)^2} = \frac{1}{1 + a}\left(1 - \frac{1}{1 + a}\right)$$

MLP with a single output

i = input node index
j = hidden node index

$$E = \frac{1}{2}(y - y^o)^2 = \frac{1}{2}\left(y - \frac{1}{1 + e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}}\right)^2$$

$$\frac{\partial E}{\partial w_j^o} = \frac{\partial E}{\partial y^o}\frac{\partial y^o}{\partial w_j^o} = -(y - y^o)\frac{\partial y^o}{\partial w_j^o} = -(y - y^o) y_j^h y^o (1 - y^o)$$

$$\frac{\partial E}{\partial y^o} = \frac{\partial}{\partial y^o}\left\{\frac{1}{2}(y - y^o)^2\right\} = 2\frac{1}{2}(y - y^o)\cdot(-1) = -(y - y^o)$$

$$\frac{\partial y^o}{\partial w_1^o} = \frac{y_1^h e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}}{\left(1 + e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}\right)^2} = y_1^h y^o (1 - y^o)$$

$$\frac{\partial y^o}{\partial w_2^o} = \frac{y_2^h e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}}{\left(1 + e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}\right)^2} = y_2^h y^o (1 - y^o)$$

$$\frac{\partial y^o}{\partial w_3^o} = \frac{y_3^h e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}}{\left(1 + e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}\right)^2} = y_3^h y^o (1 - y^o)$$

MLP with a single output

i = input node index
j = hidden node index

$$E = \frac{1}{2}(y - y^o)^2 = \frac{1}{2}\left(y - \frac{1}{1 + e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}}\right)^2$$

$$\frac{\partial E}{\partial b^o} = \frac{\partial E}{\partial y^o}\frac{\partial y^o}{\partial b^o} = -(y - y^o)\frac{\partial y^o}{\partial b^o} = -(y - y^o)1y^o(1 - y^o)$$

$$\frac{\partial E}{\partial y^o} = \frac{\partial}{\partial y^o}\left\{\frac{1}{2}(y - y^o)^2\right\} = 2\frac{1}{2}(y - y^o)\cdot(-1) = -(y - y^o)$$

$$\frac{\partial y^o}{\partial b^o} = \frac{1e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}}{\left(1 + e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}\right)^2} = 1y^o(1 - y^o)$$

MLP with a single output

$$\frac{\partial E}{\partial w_j^o} = -(y - y^o) y_j^h y^o (1 - y^o) \qquad \frac{\partial E}{\partial b^o} = -(y - y^o) 1 y^o (1 - y^o)$$

How to change the weights (*update rule*):

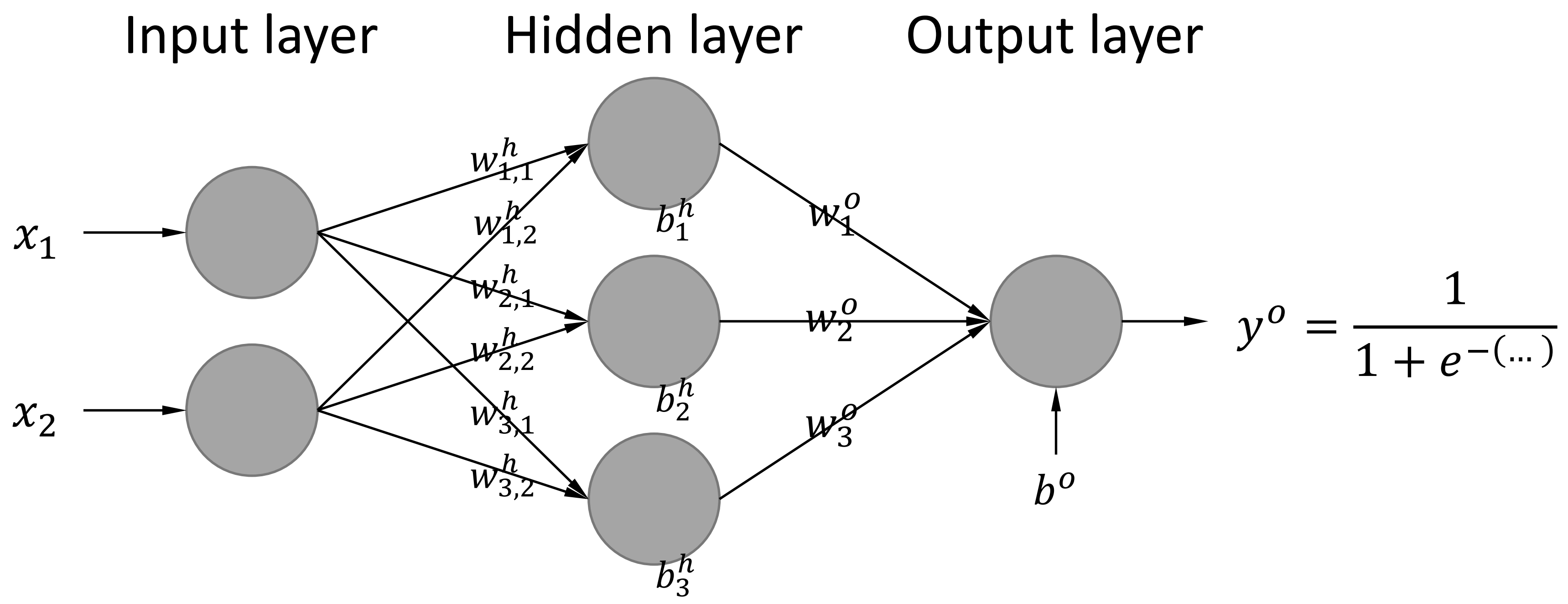
$$w_j^o = w_j^o - \eta \frac{\partial E}{\partial w_j^o} = w_j^o + \eta (\boldsymbol{y} - \boldsymbol{y^o}) y_j^h \boldsymbol{y^o} (\boldsymbol{1} - \boldsymbol{y^o})$$

$$b^o = b^o - \eta \frac{\partial E}{\partial b^o} = b^o + \eta (\boldsymbol{y} - \boldsymbol{y^o}) \boldsymbol{y^o} (\boldsymbol{1} - \boldsymbol{y^o})$$

- lrate is η
- tgt is y
- out is y^o
- err is $(y - y^o)$
- hid[h_idx] is y_j^h
- wo[h_idx] is w_j^o
- bo is b^o
- i_idx is input index i
- h_idx is input index j

```
err = tgt - out
delta_out = out * (1.0 - out) * err

wo[h_idx] += lrate * hid[h_idx] * delta_out
bo += lrate * delta_out
```

MLP with a single output



$$y^o = \frac{1}{1 + e^{-\left(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h\right)}}$$

$$y_1^h = \frac{1}{1 + e^{-\left(b_1^h + w_{1,1}^h x_1 + w_{1,2}^h x_2\right)}}$$

$$y_2^h = \frac{1}{1 + e^{-\left(b_2^h + w_{2,1}^h x_1 + w_{2,2}^h x_2\right)}}$$

$$y_3^h = \frac{1}{1 + e^{-\left(b_3^h + w_{3,1}^h x_1 + w_{3,2}^h x_2\right)}}$$

MLP with a single output

i = input node index
j = hidden node index

$$E = \frac{1}{2}(y - y^o)^2 = \frac{1}{2}\left(y - \frac{1}{1 + e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}}\right)^2$$

$$y_j^h = \frac{1}{1 + e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}$$

$$\frac{\partial E}{\partial w_{j,i}^h} = \frac{\partial E}{\partial y^o}\frac{\partial y^o}{\partial w_{j,i}^h} = \frac{\partial E}{\partial y^o}\frac{\partial y^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial w_{j,i}^h}$$

$$\frac{\partial E}{\partial y^o} = \frac{\partial}{\partial y^o}\left\{\frac{1}{2}(y - y^o)^2\right\} = 2\frac{1}{2}(y - y^o)\cdot(-1) = -(y - y^o)$$

$$\frac{\partial y^o}{\partial y_j^h} = \frac{w_j^o e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}}{\left(1 + e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}\right)^2} = w_j^o y^o(1 - y^o)$$

$$\frac{\partial y_j^h}{\partial w_{j,1}^h} = \frac{x_1 e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}{\left(1 + e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}\right)^2} = x_1 y_j^h\left(1 - y_j^h\right)$$

$$\frac{\partial y_j^h}{\partial w_{j,2}^h} = \frac{x_2 e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}{\left(1 + e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}\right)^2} = x_2 y_j^h\left(1 - y_j^h\right)$$

MLP with a single output

i = input node index
j = hidden node index

$$E = \frac{1}{2}(y - y^o)^2 = \frac{1}{2}\left(y - \frac{1}{1 + e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}}\right)^2$$

$$\frac{\partial E}{\partial w_{j,i}^h} = \frac{\partial E}{\partial y^o}\frac{\partial y^o}{\partial w_{j,i}^h} = \frac{\partial E}{\partial y^o}\frac{\partial y^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial w_{j,i}^h} = -(y - y^o) w_j^o y^o (1 - y^o) x_i y_j^h (1 - y_j^h)$$

$$\frac{\partial E}{\partial y^o} = \frac{\partial}{\partial y^o}\left\{\frac{1}{2}(y - y^o)^2\right\} = 2\frac{1}{2}(y - y^o)\cdot(-1) = -(y - y^o)$$

$$\frac{\partial y^o}{\partial y_j^h} = \frac{w_j^o e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}}{\left(1 + e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}\right)^2} = w_j^o y^o (1 - y^o)$$

$$\frac{\partial y_j^h}{\partial w_{j,1}^h} = \frac{x_1 e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}{\left(1 + e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}\right)^2} = x_1 y_j^h (1 - y_j^h)$$

$$\frac{\partial y_j^h}{\partial w_{j,2}^h} = \frac{x_2 e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}{\left(1 + e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}\right)^2} = x_2 y_j^h (1 - y_j^h)$$

MLP with a single output

i = input node index
j = hidden node index

$$E = \frac{1}{2}(y - y^o)^2 = \frac{1}{2}\left(y - \frac{1}{1 + e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}}\right)^2$$

$$y_j^h = \frac{1}{1 + e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}$$

$$\frac{\partial E}{\partial b_j^h} = \frac{\partial E}{\partial y^o}\frac{\partial y^o}{\partial b_j^h} = \frac{\partial E}{\partial y^o}\frac{\partial y^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial b_j^h}$$

$$\frac{\partial E}{\partial y^o} = \frac{\partial}{\partial y^o}\left\{\frac{1}{2}(y - y^o)^2\right\} = 2\frac{1}{2}(y - y^o)\cdot(-1) = -(y - y^o)$$

$$\frac{\partial y^o}{\partial y_j^h} = \frac{w_j^o e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}}{\left(1 + e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}\right)^2} = w_j^o y^o (1 - y^o)$$

$$\frac{\partial y_j^h}{\partial b_j^h} = \frac{1 e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}{\left(1 + e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}\right)^2} = 1 y_j^h\left(1 - y_j^h\right)$$

MLP with a single output

i = input node index
j = hidden node index

$$E = \frac{1}{2}(y - y^o)^2 = \frac{1}{2}\left(y - \frac{1}{1 + e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}}\right)^2$$

$$\frac{\partial E}{\partial b_j^h} = \frac{\partial E}{\partial y^o}\frac{\partial y^o}{\partial b_j^h} = \frac{\partial E}{\partial y^o}\frac{\partial y^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial b_j^h} = -(y - y^o) w_j^o y^o (1 - y^o) 1 y_j^h (1 - y_j^h)$$

$$\frac{\partial E}{\partial y^o} = \frac{\partial}{\partial y^o}\left\{\frac{1}{2}(y - y^o)^2\right\} = 2\frac{1}{2}(y - y^o) \cdot (-1) = -(y - y^o)$$

$$\frac{\partial y^o}{\partial y_j^h} = \frac{w_j^o e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}}{\left(1 + e^{-(b^o + w_1^o y_1^h + w_2^o y_2^h + w_3^o y_3^h)}\right)^2} = w_j^o y^o (1 - y^o)$$

$$\frac{\partial y_j^h}{\partial b_j^h} = \frac{1 e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}{\left(1 + e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}\right)^2} = 1 y_j^h (1 - y_j^h)$$

MLP with a single output

$$\frac{\partial E}{\partial w^h_{j,i}} = -(y - y^o) w^o_j y^o (1 - y^o) x_i y^h_j \left(1 - y^h_j\right) \qquad \frac{\partial E}{\partial b^h_i} = -(y - y^o) w^o_j y^o (1 - y^o) 1 y^h_j \left(1 - y^h_j\right)$$

How to change the weights (*update rule*):

$$w^h_{j,i} = w^h_{j,i} - \eta \frac{\partial E}{\partial w^h_{j,i}} = w^h_{j,i} + \eta \boldsymbol{(y - y^o) w^o_j y^o (1 - y^o)} x_i \boldsymbol{y^h_j \left(1 - y^h_j\right)}$$

$$b^h_j = b^h_j - \eta \frac{\partial E}{\partial b^h_j} = b^h_j + \eta \boldsymbol{(y - y^o) w^o_j y^o (1 - y^o)} 1 \boldsymbol{y^h_j \left(1 - y^h_j\right)}$$

- `lrate` is η
- `tgt` is y
- `out` is y^o
- `err` is $(y - y^o)$
- `hid[h_idx]` is y^h_j
- `wo[h_idx]` is w^o_j
- `bo` is b^o
- `i_idx` is input index i
- `h_idx` is input index j

```
err = tgt - out
delta_out = out * (1.0 - out) * err
err_prop[h_idx] = delta_out * wo[h_idx]
delta_hid[h_idx] = hid[h_idx] * (1.0 - hid[h_idx]) * err_prop[h_idx]

wh[h_idx, i_idx] += lrate * inp[i_idx] * delta_hid[h_idx]
bh[h_idx] += lrate * delta_hid[h_idx]
```

DIY MLP code for banknote data (2 features only)

```python
from sklearn.metrics import accuracy_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


#%% LOAD THE DATA
data = pd.read_excel("banknote.xlsx", sheet_name="mini",
  header=1, usecols="C,D,G")

features = data.drop(["class"], axis=1)
X = np.array(features)
y = np.array(data["class"])

print(data.shape)
print(X)
print(y)
data.head()


#%% MLP CODE

def neural_response(inp, wh, bh, wo, bo):
  num_inp_nodes = wh.shape[1]
  num_hid_nodes = wh.shape[0]
  num_out_nodes = wo.shape[0]

  hid = np.zeros((num_hid_nodes))
  out = np.zeros((num_out_nodes))

  # compute the hidden layer activation
  for h_idx in range(num_hid_nodes):
    val = 0
    for i_idx in range(num_inp_nodes):
      val += inp[i_idx] * wh[h_idx, i_idx]
    val += bh[h_idx]

    hid[h_idx] = 1.0 / (1.0 + np.exp(-val))

  # compute the output layer activation
  for o_idx in range(num_out_nodes):
    val = 0
    for h_idx in range(num_hid_nodes):
      val += hid[h_idx] * wo[o_idx, h_idx]
    val += bo[o_idx]

    out[o_idx] = 1.0 / (1.0 + np.exp(-val))

  # return the output
  return (hid, out)
```

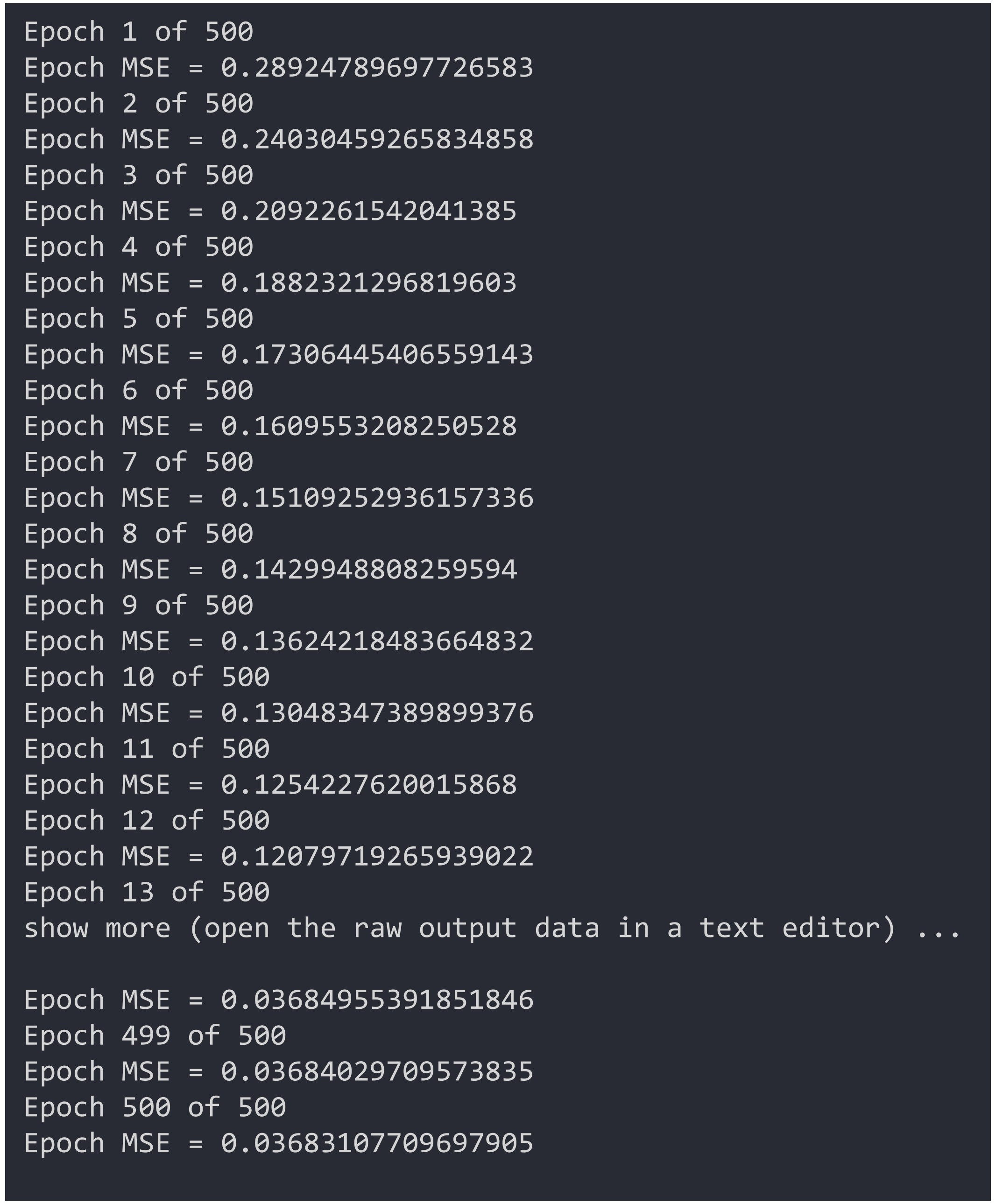
```python
def update_network(hid, out, err, wh, bh, wo, bo):
  num_inp_nodes = wh.shape[1]
  num_hid_nodes = wh.shape[0]
  num_out_nodes = wo.shape[0]

  err_prop = np.zeros((num_hid_nodes))
  delta_hid = np.zeros((num_hid_nodes))
  delta_out = np.zeros((num_out_nodes))

  # compute the output layer deltas
  for o_idx in range(num_out_nodes):
    delta_out[o_idx] = out[o_idx]*(1.0 - out[o_idx]) * err[o_idx]

  # compute the error to propagate back to the hidden layer
  for h_idx in range(num_hid_nodes):
    val = 0
    for o_idx in range(num_out_nodes):
      val += (delta_out[o_idx] * wo[o_idx, h_idx])
    err_prop[h_idx] = val

  # compute the hidden layer deltas
  for h_idx in range(num_hid_nodes):
    delta_hid[h_idx] = hid[h_idx]*(1.0 - hid[h_idx]) * err_prop[h_idx]

  # adjust wo
  for h_idx in range(num_hid_nodes):
    for o_idx in range(num_out_nodes):
      wo[o_idx, h_idx] += lrate * (hid[h_idx] * delta_out[o_idx])

  # adjust bo
  for o_idx in range(num_out_nodes):
    bo[o_idx] += lrate * delta_out[o_idx]

  # adjust wh
  for i_idx in range(num_inp_nodes):
    for h_idx in range(num_hid_nodes):
      wh[h_idx, i_idx] += lrate * (inp[i_idx] * delta_hid[h_idx])

  # adjust bh
  for h_idx in range(num_hid_nodes):
    bh[h_idx] += lrate * delta_hid[h_idx]
```

DIY MLP code for banknote data (2 features only)

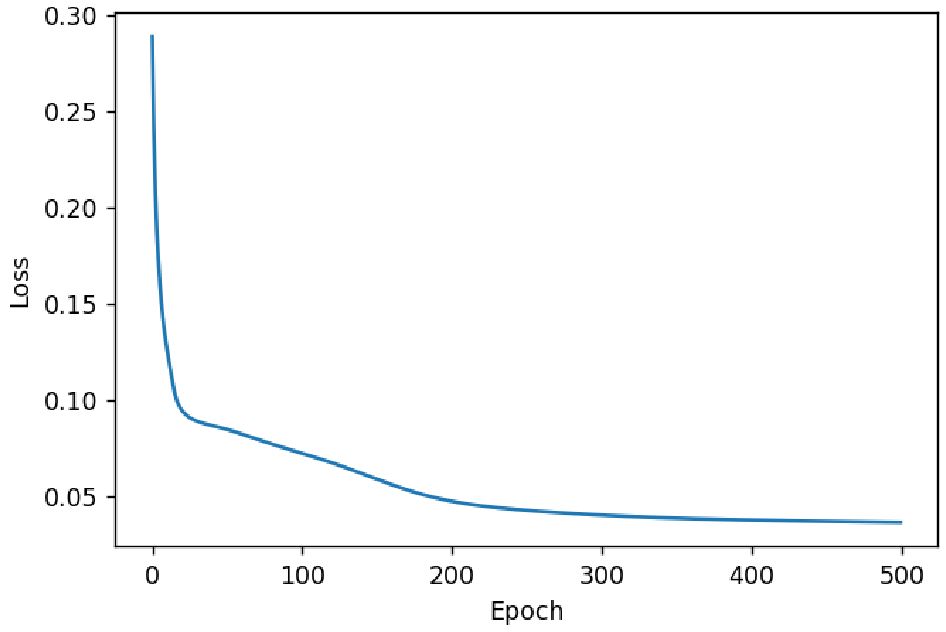
```python
#%% TRAINING
NUM_INP_NODES = 2
NUM_HID_NODES = 3
NUM_OUT_NODES = 1

lrate = 0.15
num_epochs = 500

# randomize the (input to) hidden weights
wh = 2*np.random.rand(NUM_HID_NODES, NUM_INP_NODES) - 1
# randomize the (hidden to) output weights
wo = 2*np.random.rand(NUM_OUT_NODES, NUM_HID_NODES) - 1
# initialize the bias weights
bh = np.zeros((NUM_HID_NODES))
bo = np.zeros((NUM_OUT_NODES))

# for error tracking and propagation
err = np.zeros((NUM_OUT_NODES))
mses = np.zeros((num_epochs))

for epoch in range(num_epochs):
  print("Epoch", epoch+1, "of", num_epochs)

  # for each input/target pattern combo...
  epoch_mses = list()
  for inp, tgt in zip(X, y):

    # compute the MLP's response(s)
    hid, out = neural_response(inp, wh, bh, wo, bo)

    # compute the o/p node's MSE
    err = tgt - out
    epoch_mses.append((err**2).mean())

    # update the weights
    update_network(hid, out, err, wh, bh, wo, bo)

  # show/store the MSE for all patterns
  epoch_mse = np.mean(epoch_mses)
  print("Epoch MSE =", epoch_mse)
  mses[epoch] = epoch_mse
```

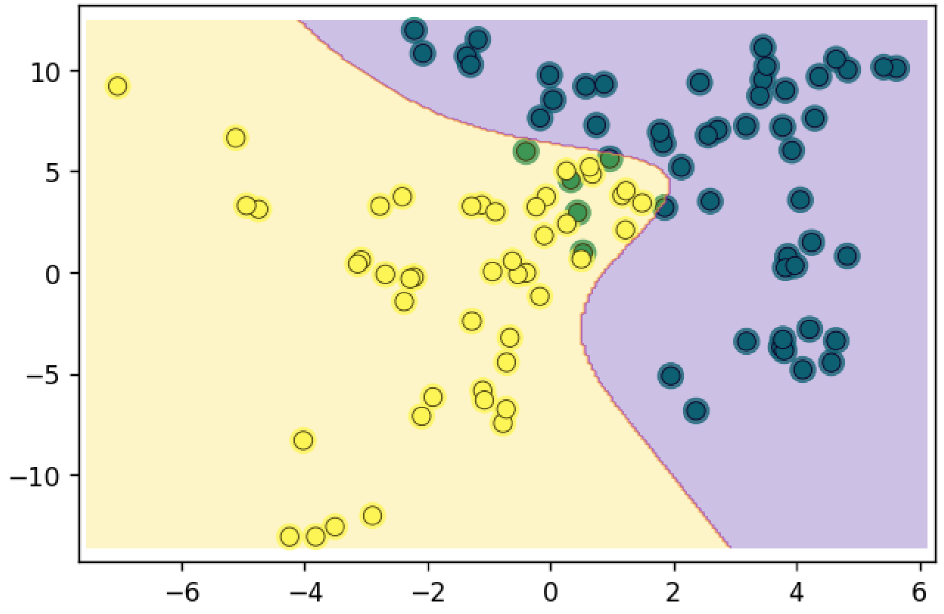
```python
plt.plot(mses)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()


#%% PLOT THE DECISION SURFACE

plt.scatter(X[:, 0], X[:, 1], c=y, s=100,
  cmap='summer', linewidths=0.5, alpha=0.75)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50,
  cmap='summer', edgecolors='black', linewidths=0.5)

ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

x_min = X[:, 0].min() - 0.5
x_max = X[:, 0].max() + 0.5
y_min = X[:, 1].min() - 0.5
y_max = X[:, 1].max() + 0.5
xx, yy = np.meshgrid(
  np.linspace(x_min, x_max, 250),
  np.linspace(y_min, y_max, 250)
  )
xy_list = np.c_[xx.ravel(), yy.ravel()]

Z = np.zeros((len(xy_list)))
for idx in range(len(xy_list)):
  _, out = neural_response(xy_list[idx], wh, bh, wo, bo)
  Z[idx] = 1 if out >= 0.5 else 0

Z = Z.reshape(xx.shape)
ax.contourf(xx, yy, Z, cmap="plasma", alpha=0.25)

ax.set_xlim(xlim)
ax.set_ylim(ylim)
plt.show()
```

DIY MLP code for banknote data (2 features only)



```
Epoch 1 of 500
Epoch MSE = 0.28924789697726583
Epoch 2 of 500
Epoch MSE = 0.24030459265834858
Epoch 3 of 500
Epoch MSE = 0.2092261542041385
Epoch 4 of 500
Epoch MSE = 0.1882321296819603
Epoch 5 of 500
Epoch MSE = 0.17306445406559143
Epoch 6 of 500
Epoch MSE = 0.1609553208250528
Epoch 7 of 500
Epoch MSE = 0.15109252936157336
Epoch 8 of 500
Epoch MSE = 0.1429948808259594
Epoch 9 of 500
Epoch MSE = 0.13624218483664832
Epoch 10 of 500
Epoch MSE = 0.13048347389899376
Epoch 11 of 500
Epoch MSE = 0.1254227620015868
Epoch 12 of 500
Epoch MSE = 0.12079719265939022
Epoch 13 of 500
show more (open the raw output data in a text editor) ...

Epoch MSE = 0.03684955391851846
Epoch 499 of 500
Epoch MSE = 0.03684029709573835
Epoch 500 of 500
Epoch MSE = 0.03683107709697905
```



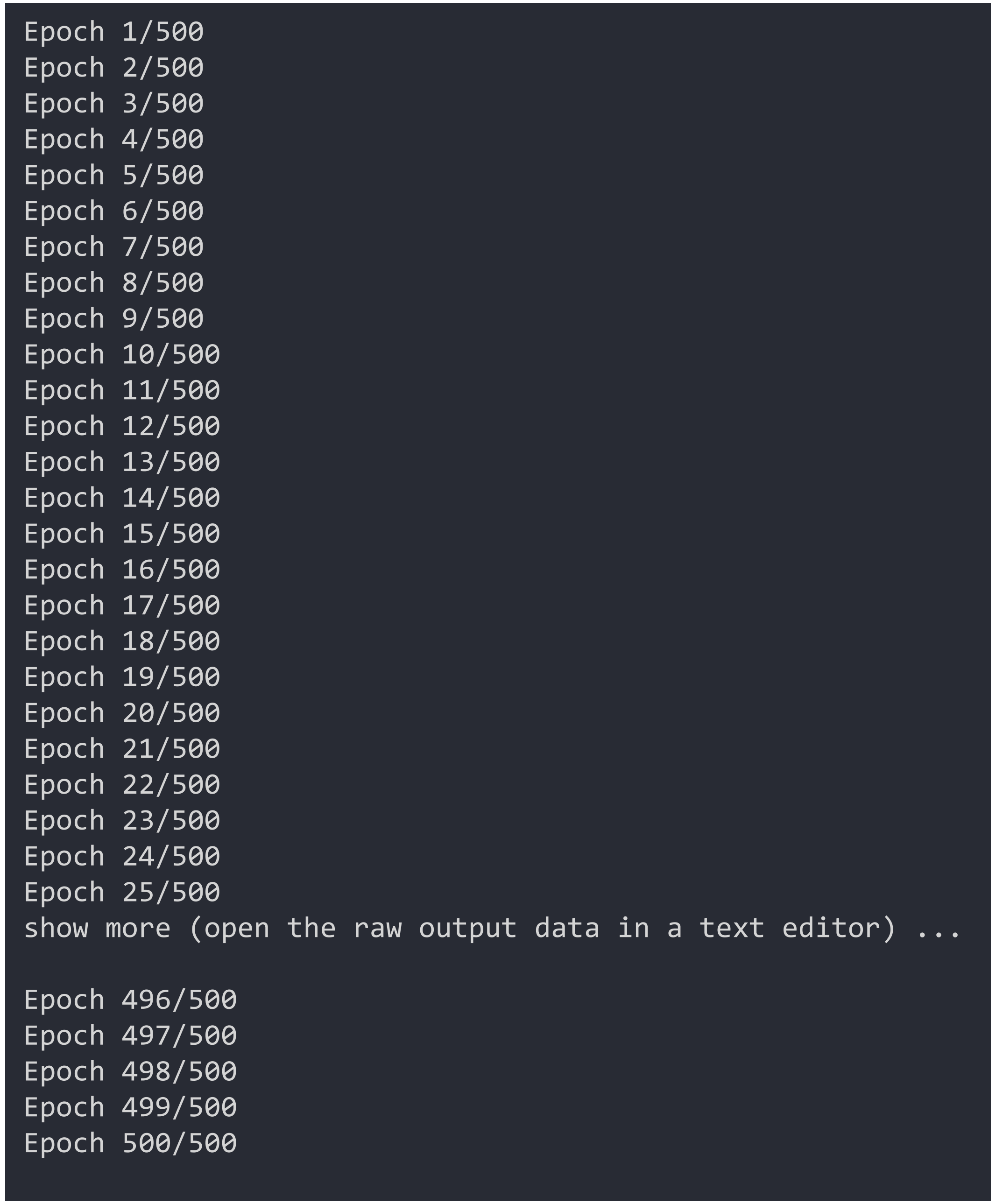


Keras MLP code for banknote data (2 features only)

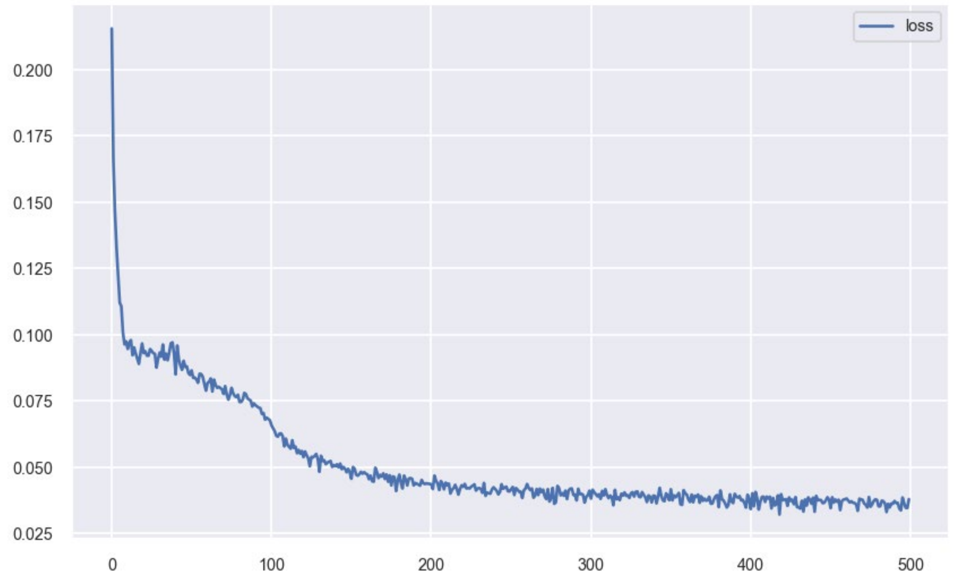
```python
from sklearn.metrics import accuracy_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


#%% LOAD THE DATA
data = pd.read_excel("banknote.xlsx", sheet_name="mini",
  header=1, usecols="C,D,G")

features = data.drop(["class"], axis=1)
X = np.array(features)
y = np.array(data["class"])

print(data.shape)
print(X)
print(y)
data.head()


#%% MLP CODE
NUM_INP_NODES = 2
NUM_HID_NODES = 3
NUM_OUT_NODES = 1

model = Sequential()
hidden_layer = Dense(NUM_HID_NODES,
  input_shape=(NUM_INP_NODES,),
  activation="sigmoid")
output_layer = Dense(NUM_OUT_NODES,
  activation="sigmoid")
model.add(hidden_layer)
model.add(output_layer)

# model.compile(loss="mean_squared_error",
#   optimizer=tf.keras.optimizers.Adam(learning_rate=0.05))
model.compile(loss="mean_squared_error",
  optimizer=tf.keras.optimizers.SGD(learning_rate=0.1))

model.summary()
```

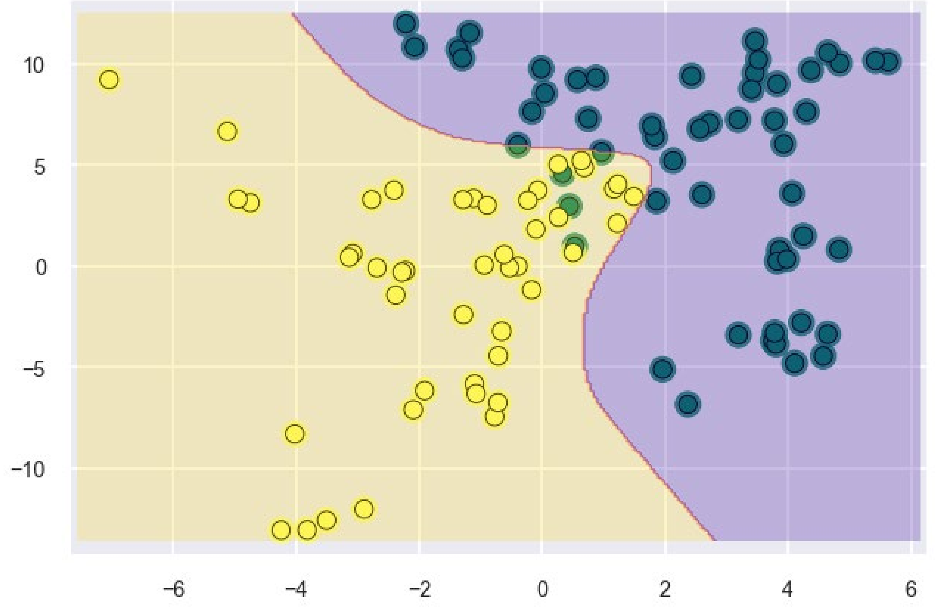
```python
#%% TRAINING

num_epochs = 500
history = model.fit(X, y, epochs=num_epochs, batch_size=1,
  verbose="0")

pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.show()


#%% PLOT THE DECISION SURFACE

plt.scatter(X[:, 0], X[:, 1], c=y, s=100,
  cmap='summer', linewidths=0.5, alpha=0.75)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50,
  cmap='summer', edgecolors='black', linewidths=0.5)

ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

x_min = X[:, 0].min() - 0.5
x_max = X[:, 0].max() + 0.5
y_min = X[:, 1].min() - 0.5
y_max = X[:, 1].max() + 0.5
xx, yy = np.meshgrid(
  np.linspace(x_min, x_max, 250),
  np.linspace(y_min, y_max, 250)
  )
xy_list = np.c_[xx.ravel(), yy.ravel()]

Z = (model.predict(xy_list) > 0.5).astype("int32")
Z = Z.reshape(xx.shape)
ax.contourf(xx, yy, Z, cmap="plasma", alpha=0.25)

ax.set_xlim(xlim)
ax.set_ylim(ylim)
plt.show()
```

Keras MLP code for banknote data (2 features only)



```
Epoch 1/500
Epoch 2/500
Epoch 3/500
Epoch 4/500
Epoch 5/500
Epoch 6/500
Epoch 7/500
Epoch 8/500
Epoch 9/500
Epoch 10/500
Epoch 11/500
Epoch 12/500
Epoch 13/500
Epoch 14/500
Epoch 15/500
Epoch 16/500
Epoch 17/500
Epoch 18/500
Epoch 19/500
Epoch 20/500
Epoch 21/500
Epoch 22/500
Epoch 23/500
Epoch 24/500
Epoch 25/500
show more (open the raw output data in a text editor) ...

Epoch 496/500
Epoch 497/500
Epoch 498/500
Epoch 499/500
Epoch 500/500
```


loss


10
5
0
−5
−10
−6
−4
−2
0
2
4
6