Psychoinformatics - Week 11 (Exercises)

by 黃靖娉 (r12944057@ntu.edu.tw)

In [ ]:
```
%config IPCompleter.greedy=True
%matplotlib inline

from numpy import *
from matplotlib.pyplot import *
from IPython.display import *
import warnings
warnings.simplefilter('ignore', DeprecationWarning)
from sklearn import *
import pandas as pd
import matplotlib.pyplot as plt

iris = datasets.load_iris()
X=iris.data; Y=iris.target
```

1 Performance Tuning of a Neural Net (8 points)

1.0 Baseline Performance

SVM can reach an classifcation accuracy ~ 8x% correct for the HARD Iris problem.

In [ ]:
```
sss=model_selection.StratifiedShuffleSplit(n_splits=3,test_size=0.1) # (45+45+45) v
model=svm.SVC(C=10)
acc=[]
for train_index, test_index in sss.split(X, Y): # 3-fold cross-validation
    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]
    model.fit(X_train[:,0:2],Y_train) #training
    acc.append(model.predict(X_test[:,0:2])==Y_test) # testing
print(np.mean(acc))
```

```
0.7555555555555555
```

1.1 Tuning your ANN (4 points)

Tune your model hyperparameters (# of layers, # of units in each layer, activation function, optimizer, epochs, batch_size, etc.) to see if you can push your ANN performance up to ~9x% correct for the HARD iris problem.

In [ ]:
```
import keras; print(keras.__version__)
import tensorflow; print(tensorflow.__version__)

from keras.models import Sequential, clone_model
from keras.layers import Dense, Dropout
from keras.utils import to_categorical

from keras.optimizers import Adam
from keras.callbacks import EarlyStopping

import matplotlib.pyplot as plt
```

```
WARNING:tensorflow:From c:\Users\ngjin\AppData\Local\Programs\Python\Python39\lib
\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_e
ntropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy
instead.

2.15.0
2.15.0
```

Original Model

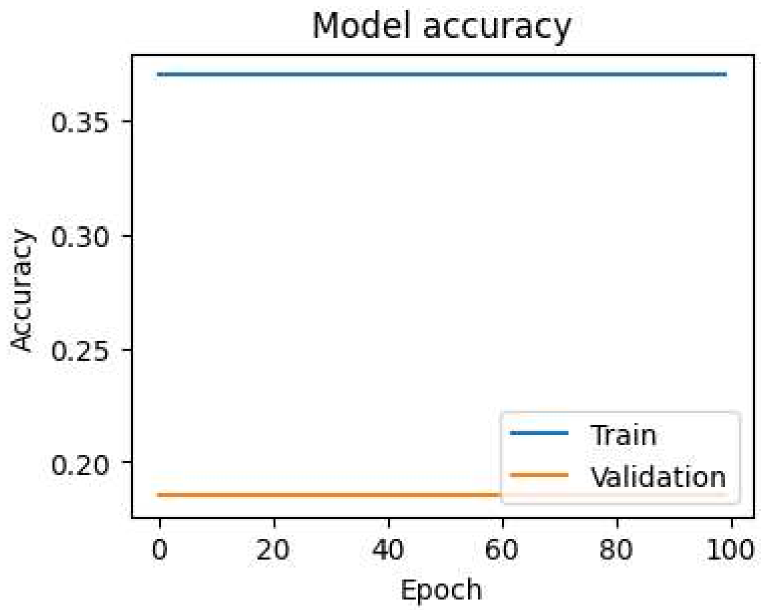
ANN_000

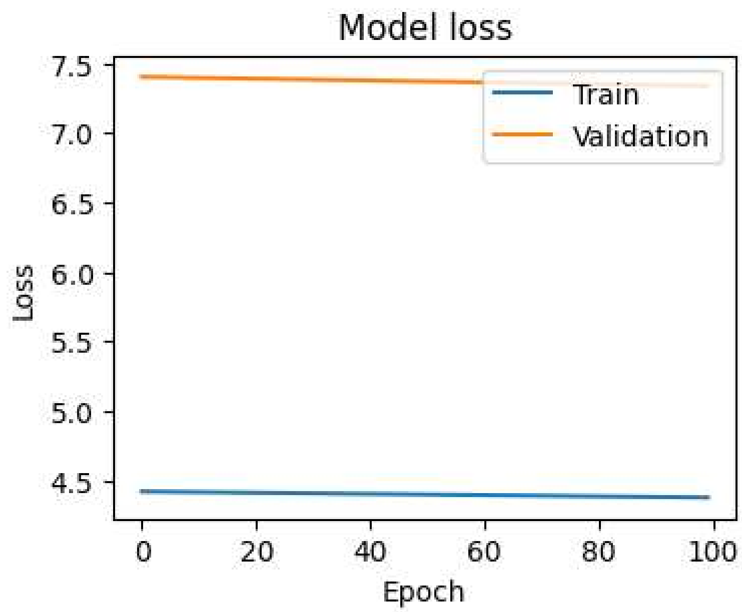
In [ ]:

```python
model = Sequential()

model.add(Dense(units=3, activation='relu'))
model.add(Dense(units=3, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy',optimizer='Adadelta',metrics=['accura

# Early Stopping Callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10)

acc = []
for train_index, test_index in sss.split(X, Y):
    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]


    train_history = model.fit(X_train[:,0:2], to_categorical(Y_train),
                              epochs=100, batch_size=10, verbose=0,
                              validation_split=0.2, callbacks=[early_stopping])  #

    acc.append(np.mean(np.argmax(model.predict(X_test[:,0:2]), axis=1) == Y_test))

print("testing acc:", np.mean(acc))

# Plot training & validation accuracy values
plt.figure(1, figsize=(4, 3))
plt.plot(train_history.history['accuracy'])  # Changed 'acc' to 'accuracy'
plt.plot(train_history.history['val_accuracy'])  # Changed 'val_acc' to 'val_accura
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()

# Plot training & validation loss values
plt.figure(1, figsize=(4, 3))
plt.plot(train_history.history['loss'])
plt.plot(train_history.history['val_loss'])

plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()

# Plot the decision boundary
h = .02  # step size in the mesh
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5  # x-axis range
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5  # y-axis range
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))  # cre
```

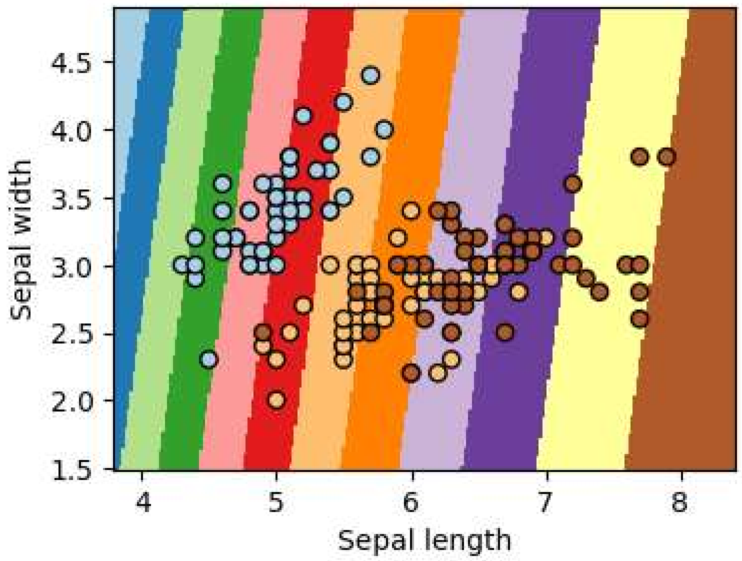
```
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])[:, 0]  # predict on the grid
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)  # plot decision boundary
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired)  # plot data
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.show()
```

```
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
testing acc: 0.3333333333333333
```






```
1235/1235 [==============================] - 1s 1ms/step
```



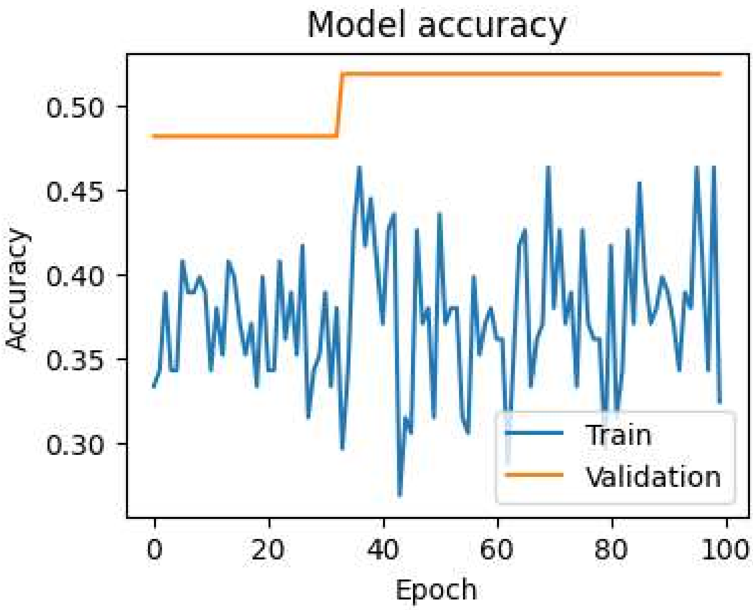
ANN_001

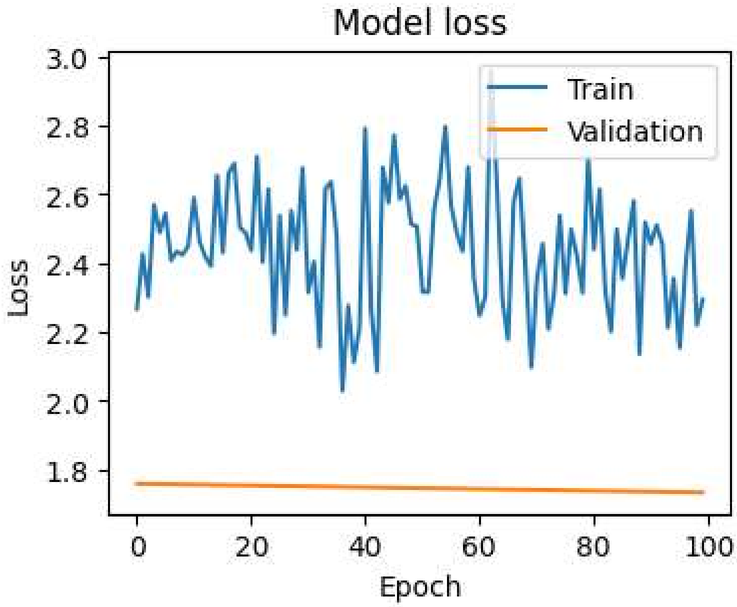
Try changing the units to improve the accuracy of data.

```
In [ ]: model = Sequential()

model.add(Dense(units=8, activation='relu', kernel_regularizer='l2'))  # More neurc
model.add(Dropout(0.2))                                                # Dropout fc
model.add(Dense(units=3, activation='softmax'))                        # Output lay

# Compile the model
model.compile(loss='categorical_crossentropy',optimizer='Adadelta',metrics=['accura

# model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001), metrics=

# Early Stopping Callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10)

acc = []
for train_index, test_index in sss.split(X, Y):
    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]


    train_history = model.fit(X_train[:,0:2], to_categorical(Y_train),
                              epochs=100, batch_size=10, verbose=0,
                              validation_split=0.2, callbacks=[early_stopping])  #

    acc.append(np.mean(np.argmax(model.predict(X_test[:,0:2]), axis=1) == Y_test))


print("testing acc:", np.mean(acc))

# Plot training & validation accuracy values
plt.figure(1, figsize=(4, 3))
plt.plot(train_history.history['accuracy'])  # Changed 'acc' to 'accuracy'
plt.plot(train_history.history['val_accuracy'])  # Changed 'val_acc' to 'val_accurc
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()
```

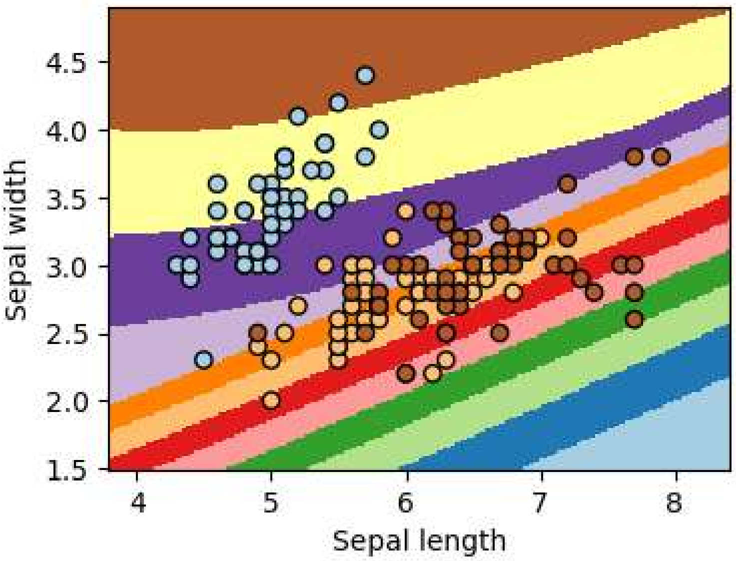
```python
# Plot training & validation loss values
plt.figure(1, figsize=(4, 3))
plt.plot(train_history.history['loss'])
plt.plot(train_history.history['val_loss'])

plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()

# Plot the decision boundary
h = .02  # step size in the mesh
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5  # x-axis range
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5  # y-axis range
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))  # cre
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])[:, 0]  # predict on the grid
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)  # plot decision boundary
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired)  # plot data
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.show()
```

```
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 26ms/step
testing acc: 0.37777777777777777
```






```
1235/1235 [==============================] - 1s 1ms/step
```



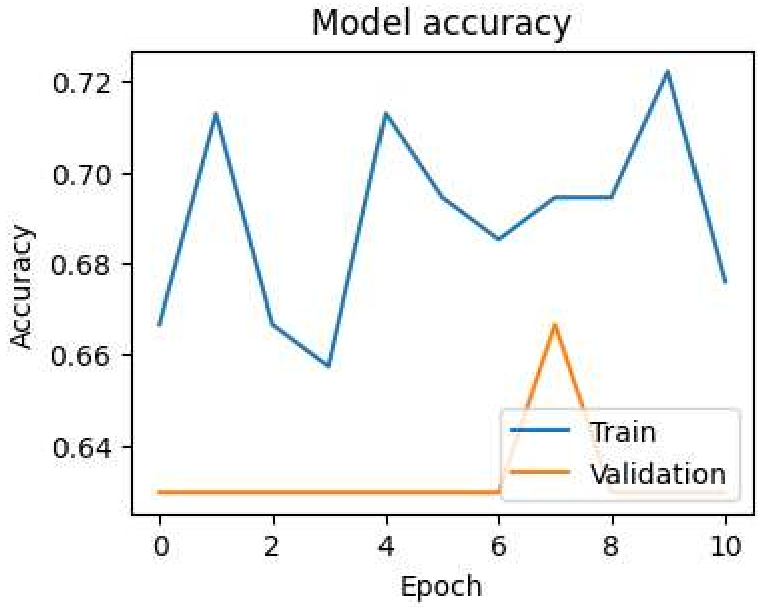
ANN_002

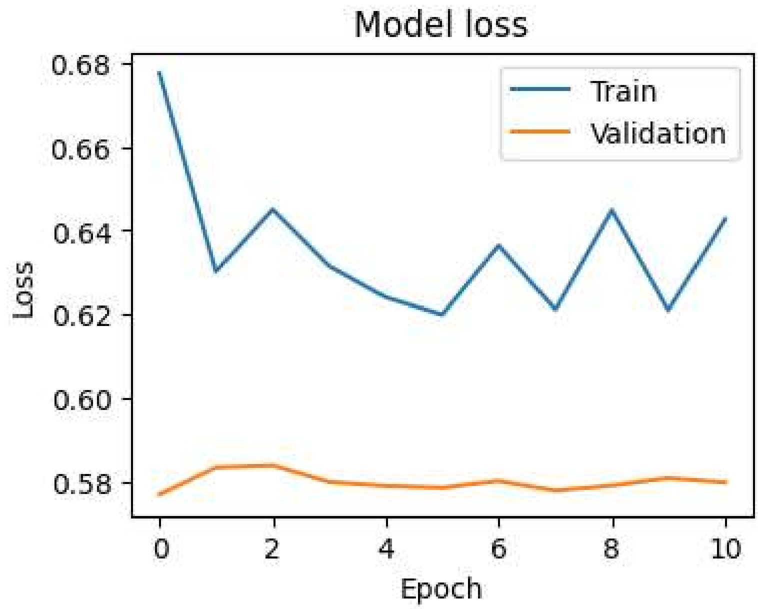
Try Adam Optimizer. Optimizers like Adam which often perform better than simple SGD due to adaptive learning rates.

In [ ]:
```
model = Sequential()

model.add(Dense(units=8, activation='relu', kernel_regularizer='l2'))  # More neuro
model.add(Dropout(0.2))                                                # Dropout fo
model.add(Dense(units=3, activation='softmax'))                        # Output lay

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001), metrics=['

# Early Stopping Callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10)

acc = []
for train_index, test_index in sss.split(X, Y):
    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]
```

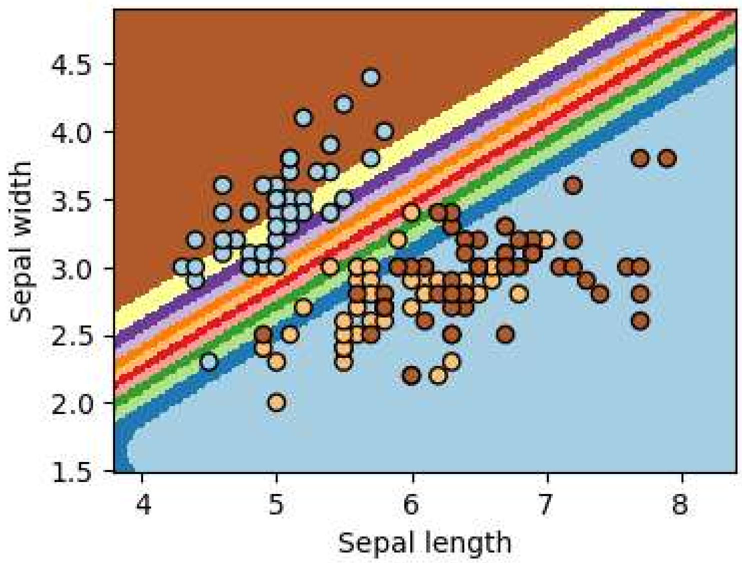
```
    train_history = model.fit(X_train[:,0:2], to_categorical(Y_train),
                              epochs=100, batch_size=10, verbose=0,
                              validation_split=0.2, callbacks=[early_stopping])  #

    acc.append(np.mean(np.argmax(model.predict(X_test[:,0:2]), axis=1) == Y_test))


print("testing acc:", np.mean(acc))

# Plot training & validation accuracy values
plt.figure(1, figsize=(4, 3))
plt.plot(train_history.history['accuracy'])  # Changed 'acc' to 'accuracy'
plt.plot(train_history.history['val_accuracy'])  # Changed 'val_acc' to 'val_accura
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()

# Plot training & validation loss values
plt.figure(1, figsize=(4, 3))
plt.plot(train_history.history['loss'])
plt.plot(train_history.history['val_loss'])

plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()

# Plot the decision boundary
h = .02  # step size in the mesh
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5  # x-axis range
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5  # y-axis range
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))  # cre
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])[:, 0]  # predict on the grid
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)  # plot decision boundary
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired)  # plot data
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.show()
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or
use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 21ms/step
testing acc: 0.6222222222222222
```






```
1235/1235 [==============================] - 1s 1ms/step
```



In [ ]:

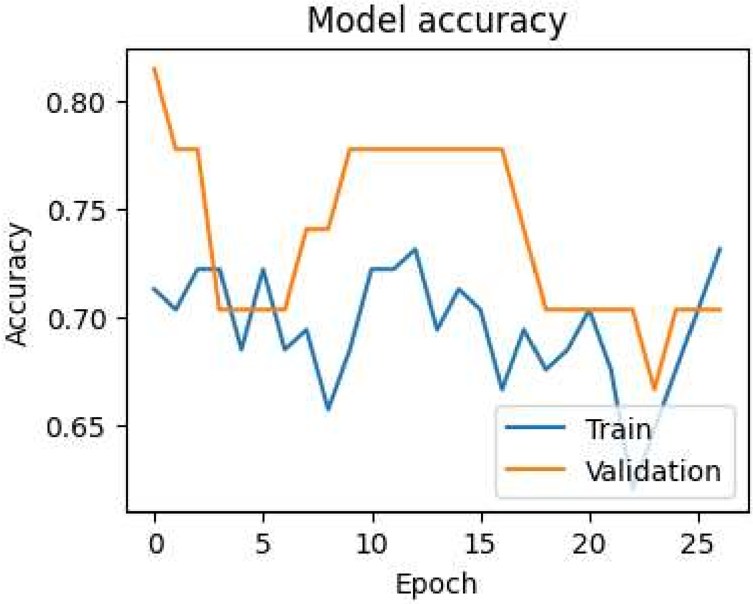
ANN_003

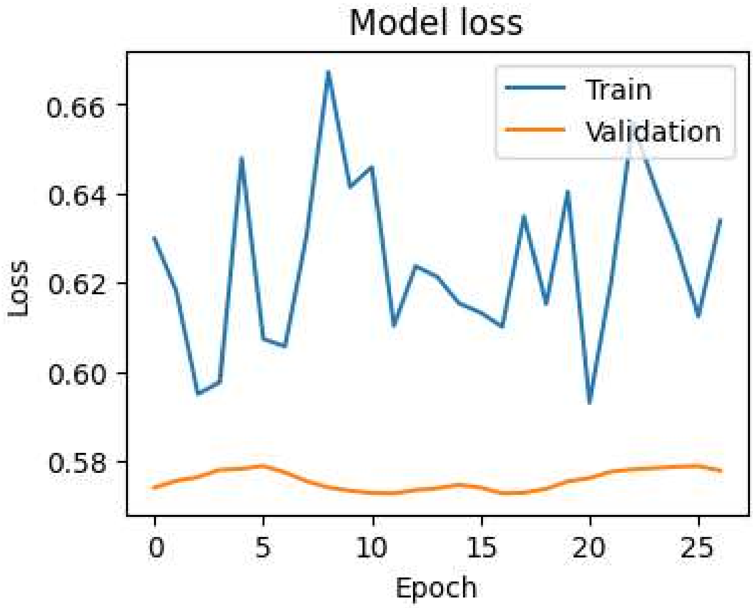
Try tuning with each variables(epoch and batch_size) and adding more layers.(Best ANN Model)

In [ ]:
```
model = Sequential()

model.add(Dense(units=8, activation='relu', kernel_regularizer='l2'))  # More neuro
model.add(Dropout(0.2))                                                # Dropout fo
model.add(Dense(units=8, activation='relu', kernel_regularizer='l2'))  # Additional
model.add(Dense(units=3, activation='softmax'))                        # Output lay

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001), metrics=['

# Early Stopping Callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10)

acc = []
for train_index, test_index in sss.split(X, Y):
    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]


    train_history = model.fit(X_train[:,0:2], to_categorical(Y_train),
                              epochs=300, batch_size=20, verbose=0,
                              validation_split=0.2, callbacks=[early_stopping])  #

    acc.append(np.mean(np.argmax(model.predict(X_test[:,0:2]), axis=1) == Y_test))


print("testing acc:", np.mean(acc))

# Plot training & validation accuracy values
plt.figure(1, figsize=(4, 3))
plt.plot(train_history.history['accuracy'])  # Changed 'acc' to 'accuracy'
plt.plot(train_history.history['val_accuracy'])  # Changed 'val_acc' to 'val_accura
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()

# Plot training & validation loss values
plt.figure(1, figsize=(4, 3))
plt.plot(train_history.history['loss'])
plt.plot(train_history.history['val_loss'])

plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()

# Plot the decision boundary
h = .02  # step size in the mesh
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5  # x-axis range
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5  # y-axis range
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))  # cre
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])[:, 0]  # predict on the grid
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)  # plot decision boundary
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired)  # plot data
```

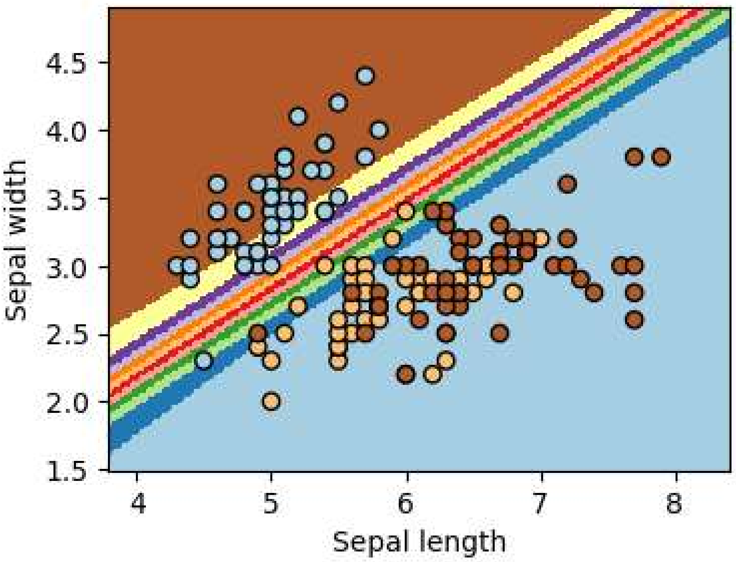
```
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.show()
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1/1 [==============================] - 0s 51ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
testing acc: 0.8000000000000002
```






```
1235/1235 [==============================] - 1s 1ms/step
```



1.2 Is your (deep) network better than SVM? Why or why not? (4 points)

My best ANN Model has 80% testing accuracy comparing to the original model which is 33% accuracy, but my best ANN model doesn't reach 90%.

Which first to compare with SVM which is 0.75555 , and my best ANN model 0.8000000000000002, it doesn't have a large gap between them. Still ANN model wins the SVM model.

It's maybe because of ANNs generally better for complex problems with large amounts of data, especially where the relationships in the data are non-linear and intricate.

While SVMs often preferred for tasks where the data is well-structured and the problem is more about finding a clear margin of separation between classes. They are particularly useful when the dataset is not too large, and computational resources are limited. So ANN is more suitable using in this situation in my opinion.