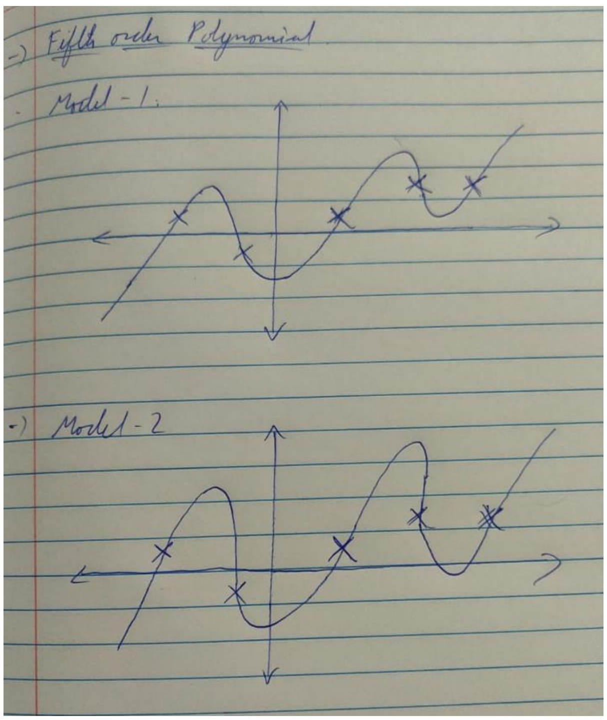
Assignment- 4

Hruday Vishal Kanna Anand

1006874517

Exercise: Draw some points on a 2D plane to which a fifth-order polynomial can fit. Show one candidate fifth order polynomial that will fit these points. Now show an alternate fifth order polynomial that will fit the same points. Comment on the relative coefficients of the two polynomials (how can you do this from a sketch?).



We can see that both the models fit the data points and are both fifth order polynomials but model 1 has smaller slopes than model 2 which implies that the coefficients of the first model will be smaller than the second model.

We can also say that the first model is a better fit as it is less prone to overfitting whereas the second model is prone to overfitting the dataset

Exercise: Contrast this exercise with the motivating example from lecture I on polynomial regression.

In this exercise we can see even by keeping the DOF's constant the fit and generalization of the model depends on its weights- how large they may be. Larger weights may overfit the dataset while smaller weights may be more generalized. Whereas in the polynomial regression we increases the DOF's step by step in order to minimize the possibility of the model to overfit the data.

Exercise: Why are we only proposing augmentation of the training dataset?

Augmentation produces synthetic data that may have a bias imparted by the creator of the synthetic data. While training this may be acceptable as we are trying to train our model with as many possibilities possible but while we validate the model we want to validate it with proper data and not make any assumptions on our data set.

Exercise: Does a bottleneck layer require nonlinearities? Justify your answer

The bottleneck layer doesn't need Non linearities as the main function of this layer is to reduce the number of DOF's in the system and the data from layer A will be transferred to layer B through this bottle neck layer. For this purpose a linear function can be used in the bottle neck layer as it will work fine for this purpose.

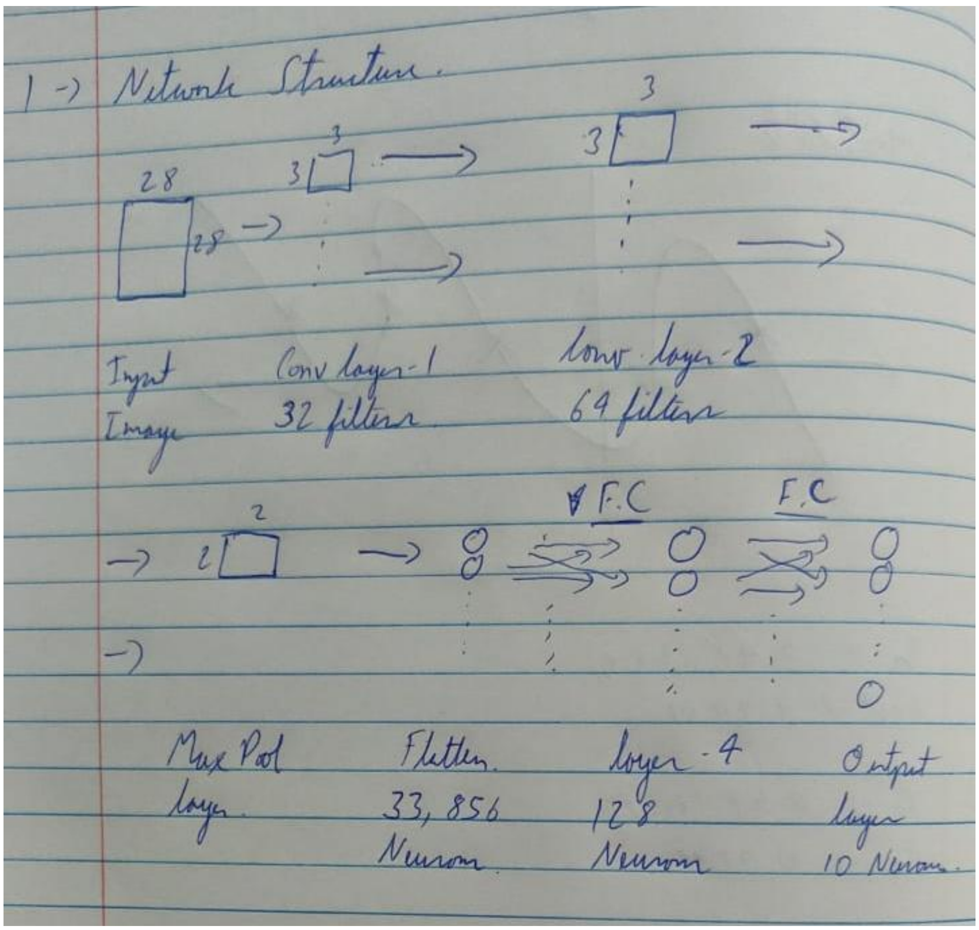
Exercise: Suppose we knew that model M1 with N1 DOFs fit the first mode of operation, and model M2 with N2 DOFs fit the second mode of operation. Describe how these three solutions {M,N}, {M1,N1} and {M2,N2} relate to each other in terms of suitability to solving the problem, lower number of DOFs, etc. Consider a practical case: suppose our dataset expresses a relationship that has (for some inputs) a fourth order polynomial nonlinear relationship, and (for other inputs) a linear relationship. How would {M,N}, {M1,N1}, {M2,N2} look for this case?

M2 and M1 working together will give us a better output than a single model M as the DOF required by M (N) will be larger than N1 and N2 which in turn will be more prone to overfitting than models M1 and M2.

In the practical case we can say the minimum DOF needed for M1 to fit a 4^{th} order polynomial is 5 – 4 weights and a bias and the minimum DOF for M2 to fit a linear polynomial is 2 – 1 weight and a bias. Whereas a model M to fit to fit both the 4^{th} order polynomial and the linear polynomial will be 6 or greater and also depends on the interaction of these two modes of operation with each other and may requires many DOF's to truly represent their relationship and properly fit both modes. Hence can be prone to overfitting as we increase the DOF of model M.

Question- Dropout

1. Network structure



CNN Dropout

```
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense

# Commonly used modules
import numpy as np
import os
import sys

# Images, plots, display, and visualization
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import cv2
import IPython
from six.moves import urllib

print(tf.__version__)
```

```
2.4.1
```

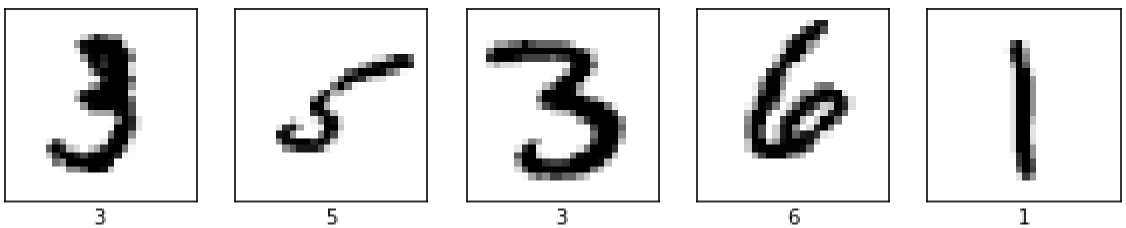
```
(train_images, train_labels), (test_images, test_labels) = keras.datasets.mnist.load_data()
# reshape images to specify that it's a single channel
print(test_labels.shape)
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1)
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
(10000,)
```

```
def preprocess_images(imgs): # should work for both a single image and multiple images
    sample_img = imgs if len(imgs.shape) == 2 else imgs[0]
    assert sample_img.shape in [(28, 28, 1), (28, 28)], sample_img.shape # make sure images are 28x28 and single-channel (gr
    return imgs / 255.0

train_images = preprocess_images(train_images)
test_images = preprocess_images(test_images)


plt.figure(figsize=(10,2))
for i in range(10,15):
    plt.subplot(1,5,i-9)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i].reshape(28, 28), cmap=plt.cm.binary)
    plt.xlabel(train_labels[i])
```



2. Annotate the sizes of the mathematical objects within; inputs, outputs, filters, etc.

The input to the neural network are 28 X 28 images- the train set has 60000 images and test set has 10000 images.

The output vector is a 10 element one hot vector that maps the output to numbers ranging from 0-9

The model-

> 1 layer- 32 convolution filters of size 3 x 3

> 2 layer- 64 convolution filters of size 3 x 3 x 32

> 3 layer- max-pool filter of size 2 x 2

> 4 layer- fully conected layer with 128 neurons connnected with max pool layer

> Output layer- 10 neurons fully connected with layer 4

The ouputs of each layer and the weights of fully connected layers-

> 1 layer-26 x 26 x 32

> 2 layer- 24 x 24 x 64

> 3 layer- 23 x 23 x 64

> 4 layer- 128 outputs from 33,856 inputs- which gives us 4,333,568 weights and 128 biases

> Output layer- 10 outputs from 128 inputs- which gives us 1,280 weights and 10 biases

```
model = keras.Sequential()
# 32 convolution filters used each of size 3x3
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
# 64 convolution filters used each of size 3x3
model.add(Conv2D(64, (3, 3), activation='relu'))
# choose the best features via pooling
model.add(MaxPooling2D(pool_size=(2, 2)))
# randomly turn neurons on and off to improve convergence
model.add(Dropout(0.25))
# flatten since too many dimensions, we only want a classification output
model.add(Flatten())
# fully connected to get all relevant data
model.add(Dense(128, activation='relu'))
# one more dropout
model.add(Dropout(0.5))
# output a softmax to squash the matrix into output probabilities
model.add(Dense(10, activation='softmax'))
```

3. How is dropout applied?

Dropout is applied to the previous layer where the arrgument represents the chance with which a neurons activation will be 0.

eg- 0.5 means that there is a 50% chance of dropout for each neuron in the layer before

```
model.compile(optimizer=tf.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
history = model.fit(train_images, train_labels, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 145s 77ms/step - loss: 0.3605 - accuracy: 0.8879
Epoch 2/5
1875/1875 [==============================] - 146s 78ms/step - loss: 0.0787 - accuracy: 0.9760
Epoch 3/5
1875/1875 [==============================] - 147s 78ms/step - loss: 0.0587 - accuracy: 0.9820
Epoch 4/5
1875/1875 [==============================] - 148s 79ms/step - loss: 0.0477 - accuracy: 0.9855
Epoch 5/5
1875/1875 [==============================] - 150s 80ms/step - loss: 0.0415 - accuracy: 0.9864
```

```
print(test_images.shape)
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

```
(10000, 28, 28, 1)
313/313 [==============================] - 6s 20ms/step - loss: 0.0308 - accuracy: 0.9903
Test accuracy: 0.9902999997138977
```

4. What is the performance with Dropout enabled?

There is very high accurancy for both the training set and the test set while using drop out. The varience between both the training accuracy and the test accuracy is also quite low

Removing Dropout

```
model2 = keras.Sequential()
# 32 convolution filters used each of size 3x3
model2.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
# 64 convolution filters used each of size 3x3
model2.add(Conv2D(64, (3, 3), activation='relu'))
# choose the best features via pooling
model2.add(MaxPooling2D(pool_size=(2, 2)))
# flatten since too many dimensions, we only want a classification output
model2.add(Flatten())
# fully connected to get all relevant data
model2.add(Dense(128, activation='relu'))
# output a softmax to squash the matrix into output probabilities
model2.add(Dense(10, activation='softmax'))
```

```
model2.compile(optimizer=tf.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
history = model2.fit(train_images, train_labels, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 145s 77ms/step - loss: 0.2481 - accuracy: 0.9223
Epoch 2/5
1875/1875 [==============================] - 144s 77ms/step - loss: 0.0365 - accuracy: 0.9887
Epoch 3/5
1875/1875 [==============================] - 145s 77ms/step - loss: 0.0215 - accuracy: 0.9927
Epoch 4/5
1875/1875 [==============================] - 144s 77ms/step - loss: 0.0130 - accuracy: 0.9958
Epoch 5/5
1875/1875 [==============================] - 145s 77ms/step - loss: 0.0111 - accuracy: 0.9962
```

```
print(test_images.shape)
test_loss, test_acc = model2.evaluate(test_images, test_labels)
```

```
print('Test accuracy:', test_acc)
```

```
(10000, 28, 28, 1)
313/313 [==============================] - 6s 20ms/step - loss: 0.0421 - accuracy: 0.9894
Test accuracy: 0.9894000291824341
```

5. Now disable it. What is the performance?

Disabling dropout we can notice the accuracy of the training set increased but there is a slighty larger gap in accuracy between the training set and the test set. This may imply a case of slight overfitting to the Training set.

Increasing the probability of Dropout

```
model3 = keras.Sequential()
# 32 convolution filters used each of size 3x3
model3.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
# 64 convolution filters used each of size 3x3
model3.add(Conv2D(64, (3, 3), activation='relu'))
# choose the best features via pooling
model3.add(MaxPooling2D(pool_size=(2, 2)))
# randomly turn neurons on and off to improve convergence
model3.add(Dropout(0.5))
# flatten since too many dimensions, we only want a classification output
model3.add(Flatten())
# fully connected to get all relevant data
model3.add(Dense(128, activation='relu'))
# one more dropout
model3.add(Dropout(0.7))
# output a softmax to squash the matrix into output probabilities
model3.add(Dense(10, activation='softmax'))
```

```
model3.compile(optimizer=tf.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
```

```
              metrics=['accuracy'])
```

```
history = model3.fit(train_images, train_labels, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 151s 80ms/step - loss: 0.5483 - accuracy: 0.8233
Epoch 2/5
1875/1875 [==============================] - 150s 80ms/step - loss: 0.1664 - accuracy: 0.9509
Epoch 3/5
1875/1875 [==============================] - 150s 80ms/step - loss: 0.1287 - accuracy: 0.9623
Epoch 4/5
1875/1875 [==============================] - 149s 80ms/step - loss: 0.1028 - accuracy: 0.9681
Epoch 5/5
1875/1875 [==============================] - 150s 80ms/step - loss: 0.0955 - accuracy: 0.9712
```

```
print(test_images.shape)
test_loss, test_acc = model3.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

```
(10000, 28, 28, 1)
313/313 [==============================] - 6s 20ms/step - loss: 0.0294 - accuracy: 0.9899
Test accuracy: 0.9898999929428101
```

Using a lower probability of Dropout

```
model4 = keras.Sequential()
# 32 convolution filters used each of size 3x3
model4.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
# 64 convolution filters used each of size 3x3
model4.add(Conv2D(64, (3, 3), activation='relu'))
# choose the best features via pooling
model4.add(MaxPooling2D(pool_size=(2, 2)))
# randomly turn neurons on and off to improve convergence
model4.add(Dropout(0.1))
# flatten since too many dimensions, we only want a classification output
```

```
model4.add(Flatten())
# fully connected to get all relevant data
model4.add(Dense(128, activation='relu'))
# one more dropout
model4.add(Dropout(0.1))
# output a softmax to squash the matrix into output probabilities
model4.add(Dense(10, activation='softmax'))


model4.compile(optimizer=tf.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


history = model4.fit(train_images, train_labels, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 152s 81ms/step - loss: 0.0123 - accuracy: 0.9959
Epoch 2/5
1875/1875 [==============================] - 151s 81ms/step - loss: 0.0087 - accuracy: 0.9972
Epoch 3/5
1875/1875 [==============================] - 152s 81ms/step - loss: 0.0088 - accuracy: 0.9972
Epoch 4/5
1875/1875 [==============================] - 152s 81ms/step - loss: 0.0072 - accuracy: 0.9976
Epoch 5/5
1875/1875 [==============================] - 153s 81ms/step - loss: 0.0077 - accuracy: 0.9973
```

```
print(test_images.shape)
test_loss, test_acc = model4.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

```
(10000, 28, 28, 1)
313/313 [==============================] - 6s 20ms/step - loss: 0.0452 - accuracy: 0.9917
Test accuracy: 0.9916999936103821
```

6 and 7

Altering the dropout in model 3 we can see that increasing drop out to a higher extent- Lowers the DOF of the model drastically while training as most of the neurons get turned off and the much of the useful data is lost. This leads to a lower accuracy in both the training and test set. We can also see a larger gap between the accuracy of the test and training set.

Altering the dropout in model 4 we can see that decreasing the dropout closer to 0 brings us closer to a model that doesnt have drop out but also gives us the benfits of generalization of dropout. The accuracy of the training set is close to the model with no dropout. The gap in the accuracy of the test set and the train set is also quite low; it is comparable to the first model.

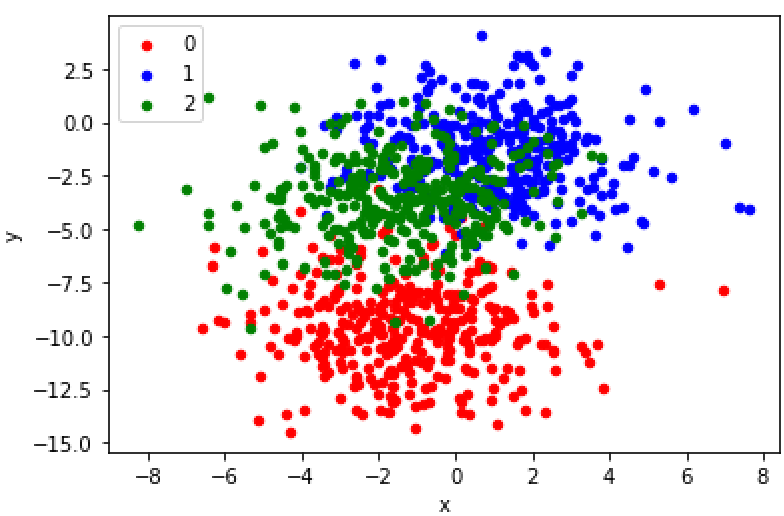
We can see that there is no definate value of dropot that will give us the best model for our problem. This is why this is a hyperparameter of the model and has to be empirically choosen by us.

```
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense
from sklearn.datasets import make_blobs
from sklearn.metrics import accuracy_score
from itertools import product
from numpy.linalg import norm

# Commonly used modules
import numpy as np
import os
import sys

# Images, plots, display, and visualization
import matplotlib.pyplot as plt
from pandas import DataFrame
```

```
# generate 2d classification dataset
X, y = make_blobs(n_samples=1000, centers=3, n_features=2, cluster_std=2, random_state=2)
# scatter plot, dots colored by class value
df = DataFrame(dict(x=X[:,0], y=X[:,1], label=y))
colors = {0:'red', 1:'blue', 2:'green'}
fig, ax = plt.subplots()
grouped = df.groupby('label')
for key, group in grouped:
    group.plot(ax=ax, kind='scatter', x='x', y='y', label=key, color=colors[key])
plt.show()
```




```
# one hot encode output variable
y1 = keras.utils.to_categorical(y)
```

```
# split into train and test
```

```
n_train = 100
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y1[:n_train], y1[n_train:]
print(trainX.shape, testX.shape)
```

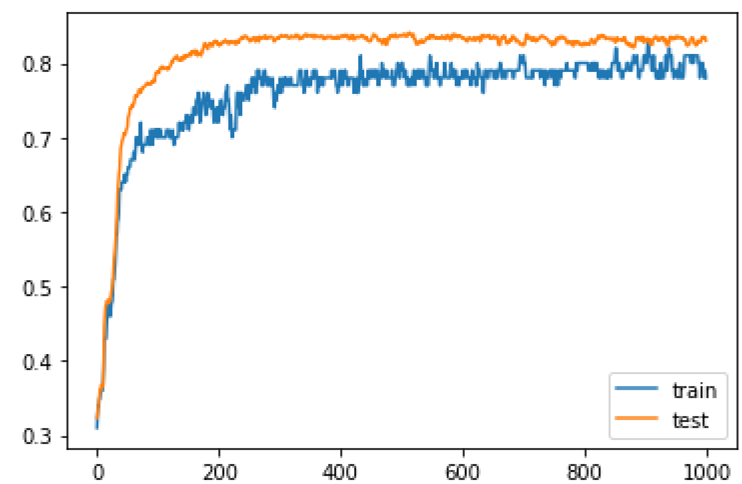
```
(100, 2) (900, 2)
```

```
# making the model
model = keras.Sequential()
model.add(Dense(25, input_dim=2, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])


# fit model
history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=1000, verbose=0)


# evaluate the model
_, train_acc = model.evaluate(trainX, trainy, verbose=0)
_, test_acc = model.evaluate(testX, testy, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
# learning curves of model accuracy
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.legend()
plt.show()
```

```
Train: 0.780, Test: 0.830
```



Horizontal Voting Ensemble-

```
# create directory for models
os.makedirs('model')
# fit model
n_epochs, n_save_after = 1000, 950
```

```
for i in range(n_epochs):
  # fit model for a single epoch
  model.fit(trainX, trainy, epochs=1, verbose=0)
  # check if we should save the model
  if i >= n_save_after:
    model.save('models/model_' + str(i) + '.h5')


# load models from file
def load_all_models(n_start, n_end):
  all_models = list()
  for epoch in range(n_start, n_end):
    # define filename for this ensemble
    filename = 'models/model_' + str(epoch) + '.h5'
    # load model from file
    model = keras.models.load_model(filename)
    # add to list of members
    all_models.append(model)
    print('>loaded %s' % filename)
  return all_models


# make an ensemble prediction for multi-class classification
def ensemble_predictions(members, testX):
  # make predictions
  yhats = [model.predict(testX) for model in members]
  yhats = np.array(yhats)
  # sum across ensemble members
  summed = np.sum(yhats, axis=0)
  # argmax across classes
  result = np.argmax(summed, axis=1)
  return result


# evaluate a specific number of members in an ensemble
def evaluate_n_members(members, n_members, testX, testy):
  # select a subset of members
  subset = members[:n_members]
  # make prediction
  yhat = ensemble_predictions(subset, testX)
  # calculate accuracy
  return accuracy_score(testy, yhat)


# load models in order
members = load_all_models(950, 1000)
print('Loaded %d models' % len(members))
# reverse loaded models so we build the ensemble with the last models first
members = list(reversed(members))
```

SHOW HIDDEN OUTPUT

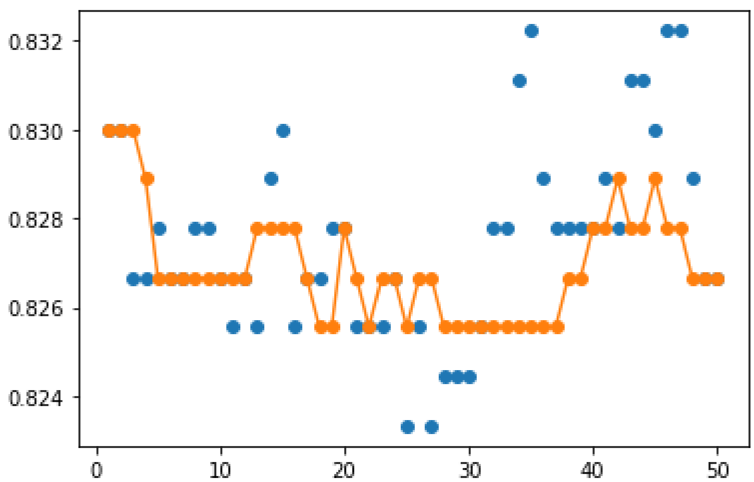
```
trainy, testy = y[:n_train], y[n_train:]
```

```
trainy, testy = y[:n_train], y[n_train:]
# evaluate different numbers of ensembles on hold out set
single_scores, ensemble_scores = list(), list()
for i in range(1, len(members)+1):
  # evaluate model with i members
  ensemble_score = evaluate_n_members(members, i, testX, testy)
  # evaluate the i'th model standalone
  testy_enc = keras.utils.to_categorical(testy)
  _, single_score = members[i-1].evaluate(testX, testy_enc, verbose=0)
  # summarize this step
  print('> %d: single=%.3f, ensemble=%.3f' % (i, single_score, ensemble_score))
  ensemble_scores.append(ensemble_score)
  single_scores.append(single_score)
```

SHOW HIDDEN OUTPUT

```
# summarize average accuracy of a single final model
print('Accuracy %.3f (%.3f)' % (np.mean(single_scores), np.std(single_scores)))
# plot score vs number of ensemble members
x_axis = [i for i in range(1, len(members)+1)]
plt.plot(x_axis, single_scores, marker='o', linestyle='None')
plt.plot(x_axis, ensemble_scores, marker='o')
plt.show()
```

```
Accuracy 0.828 (0.002)
```



we can see that the ensemble accuracy is quite stable and has a lower varience than the single models.

Average Ensemble

```
# fit model on dataset
def fit_model(trainX, trainy):
  trainy_enc = keras.utils.to_categorical(trainy)
  # define model
```

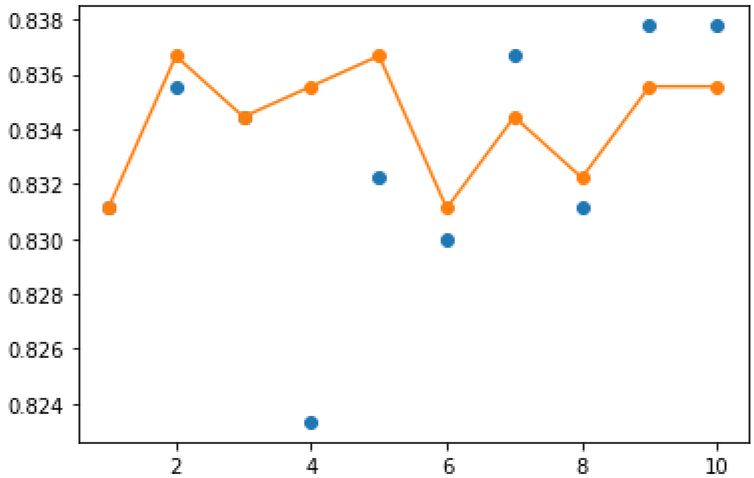
```
  model = keras.Sequential()
  model.add(Dense(25, input_dim=2, activation='relu'))
  model.add(Dense(3, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  # fit model
  model.fit(trainX, trainy_enc, epochs=500, verbose=0)
  return model


n_members = 10
members = [fit_model(trainX, trainy) for _ in range(n_members)]
# evaluate different numbers of ensembles on hold out set
single_scores, ensemble_scores = list(), list()
for i in range(1, len(members)+1):
  # evaluate model with i members
  ensemble_score = evaluate_n_members(members, i, testX, testy)
  # evaluate the i'th model standalone
  testy_enc = keras.utils.to_categorical(testy)
  _, single_score = members[i-1].evaluate(testX, testy_enc, verbose=0)
  # summarize this step
  print('> %d: single=%.3f, ensemble=%.3f' % (i, single_score, ensemble_score))
  ensemble_scores.append(ensemble_score)
  single_scores.append(single_score)
```

```
> 1: single=0.831, ensemble=0.831
> 2: single=0.836, ensemble=0.837
> 3: single=0.834, ensemble=0.834
> 4: single=0.823, ensemble=0.836
> 5: single=0.832, ensemble=0.837
> 6: single=0.830, ensemble=0.831
> 7: single=0.837, ensemble=0.834
> 8: single=0.831, ensemble=0.832
> 9: single=0.838, ensemble=0.836
> 10: single=0.838, ensemble=0.836
```

```
# summarize average accuracy of a single final model
print('Accuracy %.3f (%.3f)' % (np.mean(single_scores), np.std(single_scores)))
# plot score vs number of ensemble members
x_axis = [i for i in range(1, len(members)+1)]
plt.plot(x_axis, single_scores, marker='o', linestyle='None')
plt.plot(x_axis, ensemble_scores, marker='o')
plt.show()
```

```
Accuracy 0.833 (0.004)
```



We can see using the average ensemble we get an accuracy that is neither too high nor too low there by reducing our models varience.

Grid Search Weighted Average Ensemble

```
# make an ensemble prediction for multi-class classification
def ensemble_predictions(members, weights, testX):
  # make predictions
  yhats = [model.predict(testX) for model in members]
  yhats = np.array(yhats)
  # weighted sum across ensemble members
  summed = np.tensordot(yhats, weights, axes=((0),(0)))
  # argmax across classes
  result = np.argmax(summed, axis=1)
  return result


# evaluate a specific number of members in an ensemble
def evaluate_ensemble(members, weights, testX, testy):
  # make prediction
  yhat = ensemble_predictions(members, weights, testX)
  # calculate accuracy
  return accuracy_score(testy, yhat)


# normalize a vector to have unit norm
def normalize(weights):
  # calculate l1 vector norm
  result = norm(weights, 1)
  # check for a vector of all zeros
  if result == 0.0:
    return weights
  # return normalized vector (unit norm)
  return weights / result
```

```
# grid search weights
def grid_search(members, testX, testy):
  # define weights to consider
  w = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
  best_score, best_weights = 0.0, None
  # iterate all possible combinations (cartesian product)
  for weights in product(w, repeat=len(members)):
    # skip if all weights are equal
    if len(set(weights)) == 1:
      continue
    # hack, normalize weight vector
    weights = normalize(weights)
    # evaluate weights
    score = evaluate_ensemble(members, weights, testX, testy)
    if score > best_score:
      best_score, best_weights = score, weights
      print('>%s %.3f' % (best_weights, best_score))
  return list(best_weights)


# fit all models
n_members = 3
members = [fit_model(trainX, trainy) for _ in range(n_members)]
# evaluate each single model on the test set
testy_enc =  keras.utils.to_categorical(testy)
for i in range(n_members):
  _, test_acc = members[i].evaluate(testX, testy_enc, verbose=0)
  print('Model %d: %.3f' % (i+1, test_acc))
# evaluate averaging ensemble (equal weights)
weights = [1.0/n_members for _ in range(n_members)]
score = evaluate_ensemble(members, weights, testX, testy)
print('Equal Weights Score: %.3f' % score)
# grid search weights
weights = grid_search(members, testX, testy)
score = evaluate_ensemble(members, weights, testX, testy)
print('Grid Search Weights: %s, Score: %.3f' % (weights, score))
```

```
Model 1: 0.833
Model 2: 0.838
Model 3: 0.832
Equal Weights Score: 0.837
>[0. 0. 1.] 0.832
>[0. 1. 0.] 0.838
>[0.   0.75 0.25] 0.840
>[0.         0.88888889 0.11111111] 0.841
Grid Search Weights: [0.0, 0.888888888888889, 0.11111111111111112], Score: 0.841
```

Weighted Average MLP Ensemble

```
# global optimization to find coefficients for weighted ensemble on blobs problem
from sklearn.datasets import make_blobs
from sklearn.metrics import accuracy_score
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense
from matplotlib import pyplot
from numpy import mean
from numpy import std
from numpy import array
from numpy import argmax
from numpy import tensordot
from numpy.linalg import norm
from scipy.optimize import differential_evolution


# fit model on dataset
def fit_model(trainX, trainy):
  trainy_enc = to_categorical(trainy)
  # define model
  model = Sequential()
  model.add(Dense(25, input_dim=2, activation='relu'))
  model.add(Dense(3, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  # fit model
  model.fit(trainX, trainy_enc, epochs=500, verbose=0)
  return model


# make an ensemble prediction for multi-class classification
def ensemble_predictions(members, weights, testX):
  # make predictions
  yhats = [model.predict(testX) for model in members]
  yhats = array(yhats)
  # weighted sum across ensemble members
  summed = tensordot(yhats, weights, axes=((0),(0)))
  # argmax across classes
  result = argmax(summed, axis=1)
  return result


# evaluate a specific number of members in an ensemble
def evaluate_ensemble(members, weights, testX, testy):
  # make prediction
  yhat = ensemble_predictions(members, weights, testX)
  # calculate accuracy
  return accuracy_score(testy, yhat)
```

```
    acc = accuracy_score(testy, yhat)


# normalize a vector to have unit norm
def normalize(weights):
  # calculate l1 vector norm
  result = norm(weights, 1)
  # check for a vector of all zeros
  if result == 0.0:
    return weights
  # return normalized vector (unit norm)
  return weights / result


# loss function for optimization process, designed to be minimized
def loss_function(weights, members, testX, testy):
  # normalize weights
  normalized = normalize(weights)
  # calculate error rate
  return 1.0 - evaluate_ensemble(members, normalized, testX, testy)


# generate 2d classification dataset
X, y = make_blobs(n_samples=1100, centers=3, n_features=2, cluster_std=2, random_state=2)
# split into train and test
n_train = 100
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y[:n_train], y[n_train:]
print(trainX.shape, testX.shape)
# fit all models
n_members = 3
members = [fit_model(trainX, trainy) for _ in range(n_members)]
# evaluate each single model on the test set
testy_enc = to_categorical(testy)
for i in range(n_members):
  _, test_acc = members[i].evaluate(testX, testy_enc, verbose=0)
  print('Model %d: %.3f' % (i+1, test_acc))
# evaluate averaging ensemble (equal weights)
weights = [1.0/n_members for _ in range(n_members)]
score = evaluate_ensemble(members, weights, testX, testy)
print('Equal Weights Score: %.3f' % score)
# define bounds on each weight
bound_w = [(0.0, 1.0)  for _ in range(n_members)]
# arguments to the loss function
search_arg = (members, testX, testy)
# global optimization of ensemble weights
result = differential_evolution(loss_function, bound_w, search_arg, maxiter=1000, tol=1e-7)
# get the chosen weights
weights = normalize(result['x'])
print('Optimized Weights: %s' % weights)
# evaluate chosen weights
score = evaluate_ensemble(members, weights, testX, testy)
print('Optimized Weights Score: %.3f' % score)
```

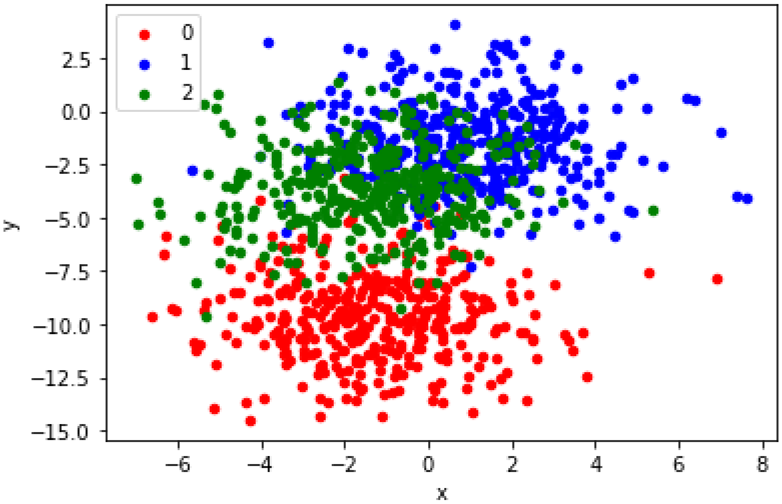
```
(100, 2) (1000, 2)
Model 1: 0.808
Model 2: 0.810
Model 3: 0.811
Equal Weights Score: 0.811
Optimized Weights: [0.0652782  0.35024905 0.58447275]
Optimized Weights Score: 0.816
```

Stacked Ensemble

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import sklearn
from sklearn.datasets import make_blobs
from sklearn.linear_model import LogisticRegression
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense
from matplotlib import pyplot
from os import makedirs
from keras.layers.merge import concatenate
from numpy import argmax
from pandas import DataFrame
```

```
# fit model on dataset
def fit_model(trainX, trainy):
  # define model
  model = Sequential()
  model.add(Dense(25, input_dim=2, activation='relu'))
  model.add(Dense(3, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  # fit model
  model.fit(trainX, trainy, epochs=500, verbose=0)
  return model
```

```
# generate 2d classification dataset
X, y = make_blobs(n_samples=1100, centers=3, n_features=2, cluster_std=2, random_state=2)
df = DataFrame(dict(x=X[:,0], y=X[:,1], label=y))
colors = {0:'red', 1:'blue', 2:'green'}
fig, ax = pyplot.subplots()
grouped = df.groupby('label')
for key, group in grouped:
    group.plot(ax=ax, kind='scatter', x='x', y='y', label=key, color=colors[key])
pyplot.show()
# one hot encode output variable
y1 = to_categorical(y)
# split into train and test
n_train = 100
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y1[:n_train], y1[n_train:]
print(trainX.shape, testX.shape)
```




```
# create directory for models
#makedirs('models')
# fit and save models
n_members = 5
for i in range(n_members):
  # fit model
  model = fit_model(trainX, trainy)
  # save model
  filename = 'models/model_' + str(i + 1) + '.h5'
  model.save(filename)
  print('>Saved %s' % filename)
```

```
>Saved models/model_1.h5
>Saved models/model_2.h5
>Saved models/model_3.h5
>Saved models/model_4.h5
>Saved models/model_5.h5
```

```
# load models from file
def load_all_models(n_models):
  all_models = list()
  for i in range(n_models):
    # define filename for this ensemble
    filename = 'models/model_' + str(i + 1) + '.h5'
    # load model from file
    model = keras.models.load_model(filename)
    # add to list of members
    all_models.append(model)
    print('>loaded %s' % filename)
  return all_models
```

```
# create stacked model input dataset as outputs from the ensemble
def stacked_dataset(members, inputX):
  stackX = None
  for model in members:
    # make prediction
```

```
        yhat = model.predict(inputX, verbose=0)
        # stack predictions into [rows, members, probabilities]
        if stackX is None:
          stackX = yhat
        else:
          stackX = np.dstack((stackX, yhat))
      # flatten predictions to [rows, members x probabilities]
      stackX = stackX.reshape((stackX.shape[0], stackX.shape[1]*stackX.shape[2]))
      return stackX


    # fit a model based on the outputs from the ensemble members
    def fit_stacked_model(members, inputX, inputy):
      # create dataset using ensemble
      stackedX = stacked_dataset(members, inputX)
      # fit standalone model
      model = LogisticRegression()
      model.fit(stackedX, inputy)
      return model


    # make a prediction with the stacked model
    def stacked_prediction(members, model, inputX):
      # create dataset using ensemble
      stackedX = stacked_dataset(members, inputX)
      # make a prediction
      yhat = model.predict(stackedX)
      return yhat


    trainy, testy = y[:n_train], y[n_train:]
    # load all models
    n_members = 5
    members = load_all_models(n_members)
    print('Loaded %d models' % len(members))
    # evaluate standalone models on test dataset
    for model in members:
      testy_enc = to_categorical(testy)
      _, acc = model.evaluate(testX, testy_enc, verbose=0)
      print('Model Accuracy: %.3f' % acc)
    # fit stacked model using the ensemble
    model = fit_stacked_model(members, testX, testy)
    # evaluate model on test set
    yhat = stacked_prediction(members, model, testX)
    acc = sklearn.metrics.accuracy_score(testy, yhat)
    print('Stacked Test Accuracy: %.3f' % acc)
```

```
>loaded models/model_1.h5
>loaded models/model_2.h5
>loaded models/model_3.h5
>loaded models/model_4.h5
>loaded models/model_5.h5
Loaded 5 models
```

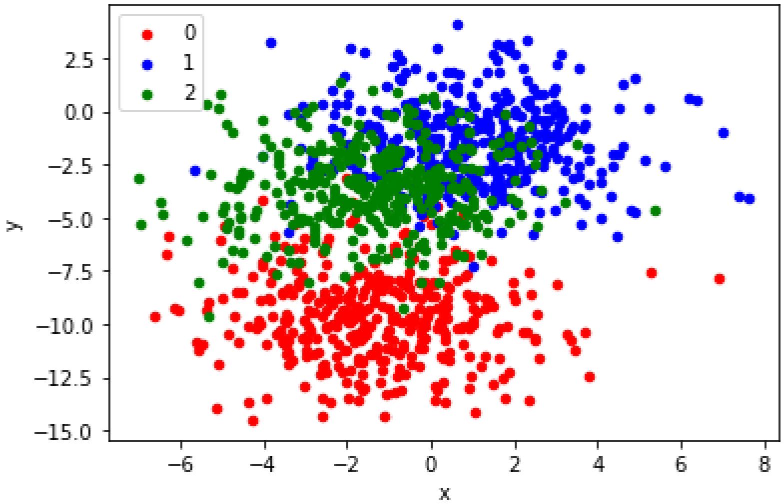
```
Model Accuracy: 0.814
Model Accuracy: 0.806
Model Accuracy: 0.808
Model Accuracy: 0.809
Model Accuracy: 0.806
Stacked Test Accuracy: 0.833
```

Integrated Stacked Ensemble

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import sklearn
from sklearn.datasets import make_blobs
from sklearn.linear_model import LogisticRegression
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense
from matplotlib import pyplot
from os import makedirs
from keras.layers.merge import concatenate
from numpy import argmax
from pandas import DataFrame
```

```
# fit model on dataset
def fit_model(trainX, trainy):
  # define model
  model = Sequential()
  model.add(Dense(25, input_dim=2, activation='relu'))
  model.add(Dense(3, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  # fit model
  model.fit(trainX, trainy, epochs=500, verbose=0)
  return model
```

```
# generate 2d classification dataset
X, y = make_blobs(n_samples=1100, centers=3, n_features=2, cluster_std=2, random_state=2)
df = DataFrame(dict(x=X[:,0], y=X[:,1], label=y))
colors = {0:'red', 1:'blue', 2:'green'}
fig, ax = pyplot.subplots()
grouped = df.groupby('label')
for key, group in grouped:
    group.plot(ax=ax, kind='scatter', x='x', y='y', label=key, color=colors[key])
pyplot.show()
# one hot encode output variable
y1 = to_categorical(y)
# split into train and test
n_train = 100
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y1[:n_train], y1[n_train:]
print(trainX.shape, testX.shape)
```




```
# create directory for models
#makedirs('models')
# fit and save models
n_members = 5
for i in range(n_members):
  # fit model
  model = fit_model(trainX, trainy)
  # save model
  filename = 'models/model_' + str(i + 1) + '.h5'
  model.save(filename)
  print('>Saved %s' % filename)

    >Saved models/model_1.h5
    >Saved models/model_2.h5
    >Saved models/model_3.h5
    >Saved models/model_4.h5
    >Saved models/model_5.h5
```

```
# load models from file
def load_all_models(n_models):
  all_models = list()
  for i in range(n_models):
    # define filename for this ensemble
    filename = 'models/model_' + str(i + 1) + '.h5'
    # load model from file
    model = keras.models.load_model(filename)
    # add to list of members
    all_models.append(model)
    print('>loaded %s' % filename)
  return all_models
```

```
# define stacked model from multiple member input models
def define_stacked_model(members):
  # update all layers in all models to not be trainable
  for i in range(len(members)):
    model = members[i]
```

```
    for layer in model.layers:
      # make not trainable
      layer.trainable = False
      # rename to avoid 'unique layer name' issue
      layer._name = 'ensemble_' + str(i+1) + '_' + layer.name
  # define multi-headed input
  ensemble_visible = [model.input for model in members]
  # concatenate merge output from each model
  ensemble_outputs = [model.output for model in members]
  merge = concatenate(ensemble_outputs)
  hidden = Dense(10, activation='relu')(merge)
  output = Dense(3, activation='softmax')(hidden)
  model = keras.Model(inputs=ensemble_visible, outputs=output)
  # plot graph of ensemble
  keras.utils.plot_model(model, show_shapes=True, to_file='model_graph.png')
  # compile
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  return model


# fit a stacked model
def fit_stacked_model(model, inputX, inputy):
  # prepare input data
  X = [inputX for _ in range(len(model.input))]
  # encode output data
  inputy_enc = to_categorical(inputy)
  # fit model
  model.fit(X, inputy_enc, epochs=300, verbose=0)


# make a prediction with a stacked model
def predict_stacked_model(model, inputX):
  # prepare input data
  X = [inputX for _ in range(len(model.input))]
  # make prediction
  return model.predict(X, verbose=0)


trainy, testy = y[:n_train], y[n_train:]
# load all models
n_members = 5
members = load_all_models(n_members)
print('Loaded %d models' % len(members))
# define ensemble model
stacked_model = define_stacked_model(members)
# fit stacked model on test dataset
fit_stacked_model(stacked_model, testX, testy)
# make predictions and evaluate
yhat = predict_stacked_model(stacked_model, testX)
yhat = argmax(yhat, axis=1)
acc = sklearn.metrics.accuracy_score(testy, yhat)
print('Integrated Stacked Test Accuracy: %.3f' % acc)
```

```
>loaded models/model_1.h5
>loaded models/model_2.h5
>loaded models/model_3.h5
>loaded models/model_4.h5
>loaded models/model_5.h5
Loaded 5 models
Integrated Stacked Test Accuracy: 0.828
```