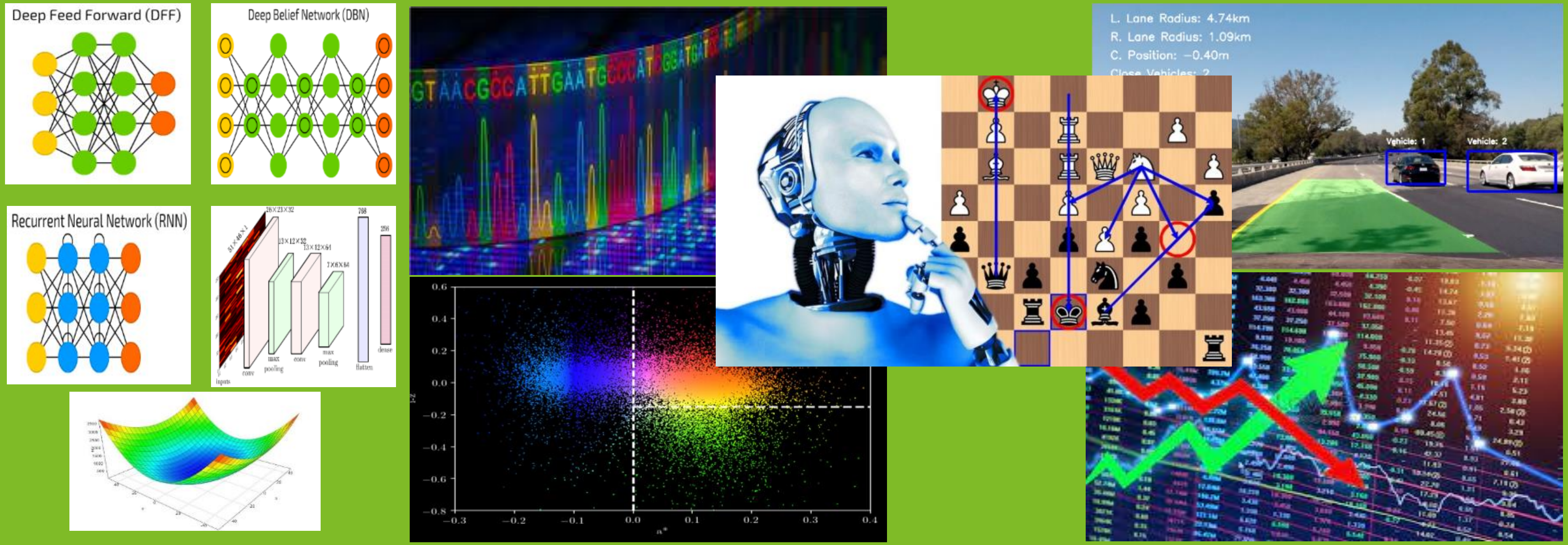
Deep Learning CSI_7_DEL



Tutorial 9: Restricted Boltzmann Machine (RBM)

London South Bank University
EST 1892

Import the data augmentation, plotting and preprocessing libraries:

```
# Import numpy
import numpy as np
#Data augmentation library
from scipy.ndimage import convolve
#Dataset repository
from sklearn import datasets
#Normalisation
from sklearn.preprocessing import minmax_scale
#Matplotlib
import matplotlib.pyplot as plt
plt.style.use("classic")
```

Import modelling related libraries

```
#Pareto's principle library 80/20
from sklearn.model_selection import train_test_split
#Import Metrics
from sklearn import metrics
#RBM model
from sklearn.neural_network import BernoulliRBM
#MLP Model Neural Net to classify the mnist data
from sklearn.neural_network import MLPClassifier
#Pipeline is used to train RBM and then feed input into neural nets.
from sklearn.pipeline import Pipeline
```

Data Augmentation Function

```
def augment_dataset(X, Y):
    #Data augmentation produces 5 time larger dataset by moving the original
    #image by 1px to the left, right, down and up
    vectors_direction = [
        [[0, 1, 0], [0, 0, 0], [0, 0, 0]],
        [[0, 0, 0], [1, 0, 0], [0, 0, 0]],
        [[0, 0, 0], [0, 0, 1], [0, 0, 0]],
        [[0, 0, 0], [0, 0, 0], [0, 1, 0]],
    ]
    def shift_pos(x, w):
        return convolve(x.reshape((8, 8)), mode="constant", weights=w).ravel()

    X = np.concatenate(
        [X] + [np.apply_along_axis(shift, 1, X, vector) for vector in direction_vectors]
    )
    Y = np.concatenate([Y for _ in range(5)], axis=0)
    return X, Y
```

Apply data augmentation, splitting and normalisation

```
#import mnist dataset
X, y = datasets.load_digits(return_X_y=True)
#Convert dataset to 32bit float
X = np.asarray(X, "float32")
#Increase the dataset samples
X, Y = nudge_dataset(X, y)
#Normalise dataset to (0-1)
X = minmax_scale(X, feature_range=(0, 1))
#Split dataset into 80/20
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)
```

Create a pipeline for feeding RBM into a logistic regression model.

```
#Create neural MLP 100 and 10 hidden nuerons
neural_net = MLPClassifier(activation='relu', hidden_layer_sizes=(100, 10), random_state=1,verbose=True)
#Create RBM model verbose set to true to show the learning
rbm = BernoulliRBM(verbose=True)
#Combine the two models in a pipeline
rbm_features_classifier = Pipeline(steps=[("rbm", rbm), ("logistic", neural_net)])
#Update the learning rate for SGD
rbm.learning_rate = 0.06
#The number of iterations
rbm.n_iter = 15
#The number of hidden units
rbm.n_components = 100
```

Fit the model for RBM and Neural Nets
Notice how RBM starts first and then Neural nets

```
#Fit the model
rbm_features_classifier.fit(X_train, Y_train)
```

```
[BernoulliRBM] Iteration 1, pseudo-likelihood = -26.07, time = 0.14s
[BernoulliRBM] Iteration 2, pseudo-likelihood = -23.92, time = 0.25s
[BernoulliRBM] Iteration 3, pseudo-likelihood = -22.65, time = 0.27s
[BernoulliRBM] Iteration 4, pseudo-likelihood = -21.27, time = 0.24s
[BernoulliRBM] Iteration 5, pseudo-likelihood = -22.02, time = 0.24s
[BernoulliRBM] Iteration 6, pseudo-likelihood = -21.37, time = 0.26s
[BernoulliRBM] Iteration 7, pseudo-likelihood = -21.18, time = 0.26s
[BernoulliRBM] Iteration 8, pseudo-likelihood = -20.48, time = 0.26s
[BernoulliRBM] Iteration 9, pseudo-likelihood = -20.46, time = 0.24s
[BernoulliRBM] Iteration 10, pseudo-likelihood = -19.88, time = 0.24s
[BernoulliRBM] Iteration 11, pseudo-likelihood = -19.97, time = 0.25s
[BernoulliRBM] Iteration 12, pseudo-likelihood = -19.80, time = 0.23s
[BernoulliRBM] Iteration 13, pseudo-likelihood = -20.14, time = 0.24s
[BernoulliRBM] Iteration 14, pseudo-likelihood = -19.42, time = 0.26s
[BernoulliRBM] Iteration 15, pseudo-likelihood = -19.25, time = 0.24s
Iteration 1, loss = 2.26432864
Iteration 2, loss = 1.95778037
Iteration 3, loss = 1.61730595
Iteration 4, loss = 1.27504843
```

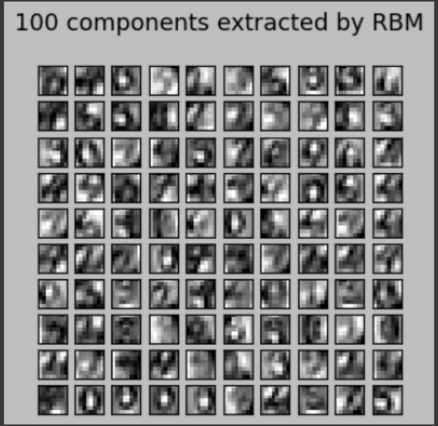
Check the accuracy of the model

```
#Check the prediction
Y_pred = rbm_features_classifier.predict(X_test)
print(
    "Neural networks using RBM features:\n%s\n"
    % (metrics.classification_report(Y_test, Y_pred))
)
```

```
Neural Networks using RBM features:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       199
           1       0.97      0.98      0.98       183
           2       0.97      0.96      0.96       183
           3       0.98      0.90      0.94       187
           4       0.97      0.97      0.97       199
           5       0.96      0.96      0.96       182
           6       0.98      0.99      0.98       165
           7       0.96      0.96      0.96       158
           8       0.88      0.93      0.91       163
           9       0.91      0.94      0.92       178

    accuracy                           0.96      1797
   macro avg       0.96      0.96      0.96      1797
weighted avg       0.96      0.96      0.96      1797
```

Visualise the 100 extract components.

```
#Plot the extracted components
plt.figure(figsize=(4.2, 4))
for i, comp in enumerate(rbm.components_):
    plt.subplot(10, 10, i + 1)
    plt.imshow(comp.reshape((8, 8)), cmap=plt.cm.gray_r, interpolation="nearest")
    plt.xticks(())
    plt.yticks(())
plt.suptitle("100 components extracted by RBM", fontsize=16)
plt.subplots_adjust(0.08, 0.02, 0.92, 0.85, 0.08, 0.23)

plt.show()
```



Tasks

- Change the number of hidden neurons in RBM to 50 the 150.
- Change the iterations for RBM to 20.
- Do you notice any improvements in the accuracy?
- Visualise the newly created 50 and 150 components and see if they correlate with your model.
- Read the paper available under this week and discuss how RBM was used to created recommender system.
- Try to apply RBM to Fashion MNIST in your own time.
  https://www.kaggle.com/datasets/zalando-research/fashionmnist

End of tutorial