IMDB Sentiment Analysis using Deep Learning

Prepare Dataset

```
from keras.datasets import imdb
import numpy as np
from keras import models
from keras.layers import Dense
import matplotlib.pylab as plt
from keras.layers import Dropout
```

```
((XT,YT),(Xt,Yt)) = imdb.load_data(num_words=10000)
```

```
len(XT)
```

```
25000
```

```
word_idx = imdb.get_word_index()
```

```
print(word_idx.items())
```

```
dict_items([('fawn', 34701), ('tsukino', 52006), ('nunnery', 52007), ('sonja'
```

```
idx_word = dict([val,key] for (key,val) in word_idx.items())
```

```
print(idx_word)
```

```
{34701: 'fawn', 52006: 'tsukino', 52007: 'nunnery', 16816: 'sonja', 63951: 'v
```

```
actual_review = ' '.join(idx_word.get(idx-3,'#') for idx in XT[0])
```

```
print(actual_review)
```

```
# this film was just brilliant casting location scenery story direction every
```

```
def vectorize_sentences(sentences,dim=10000):
    output = np.zeros((len(sentences),dim))
    for i,idx in enumerate(sentences):
        output[i,idx] = 1
    return output
```

```
X_train = vectorize_sentences(XT)
X_test = vectorize_sentences(Xt)
```

```
X_train.shape
```

```
(25000, 10000)
```

```
X_test.shape
```

```
(25000, 10000)
```

```
Y_train = np.asarray(YT).astype('float32')
```

```
Y_test = np.asarray(Yt).astype('float32')
```

Build Neural Network and create model

```
model = models.Sequential()
model.add(Dense(16,input_shape=(10000,),activation='relu'))
```

```
model.add(Dense(16,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(8,activation='relu'))
model.add(Dense(4,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
```

```
model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_9 (Dense)              (None, 16)                160016

 dense_10 (Dense)             (None, 16)                272

 dropout_1 (Dropout)          (None, 16)                0

 dense_11 (Dense)             (None, 8)                 136

 dense_12 (Dense)             (None, 4)                 36

 dense_13 (Dense)             (None, 1)                 5

=================================================================
Total params: 160,465
```

```
Trainable params: 160,465
Non-trainable params: 0
_________________________________________________________________
```

Training and Validation

```
X_val = X_train[:5000]
X_train_new = X_train[5000:]
```

```
Y_val = Y_train[:5000]
Y_train_new = Y_train[5000:]
```

```
hist = model.fit(X_train_new,Y_train_new,epochs=4,batch_size=512,valida
```

```
Epoch 1/4
40/40 [==============================] - 2s 39ms/step - loss: 0.5564 - accura
Epoch 2/4
40/40 [==============================] - 2s 49ms/step - loss: 0.3298 - accura
Epoch 3/4
40/40 [==============================] - 2s 57ms/step - loss: 0.2394 - accura
Epoch 4/4
40/40 [==============================] - 1s 32ms/step - loss: 0.1884 - accura
```

```
h = hist.history
```

```
plt.plot(h['val_loss'],label='Validation Loss')
plt.plot(h['loss'],label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
plt.plot(h['val_accuracy'],label='Validation Accuracy')
```

```
plt.plot(h['accuracy'],label='Training Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
model.evaluate(X_test,Y_test)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.3221 - accur
[0.3221316337585449, 0.8828799724578857]
```

```
model.predict(X_test)
```

```
array([[0.06379473],
       [0.9954054 ],
       [0.8130638 ],
       ...,
       [0.0535534 ],
       [0.04301476],
       [0.514066  ]], dtype=float32)
```

MNIST Classifier using Deep learning

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
import tensorflow as tf
from keras.datasets import mnist
from keras.models import Sequential

from keras.layers.core import Dense, Dropout, Activation
from keras.utils import np_utils


(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
x_train, x_test = x_train / 255.0, x_test / 255.0


model = Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])


predictions = model(x_train[:1]).numpy()
predictions
```

```
array([[ 0.4220651 ,  0.29447064,  0.04016034, -0.47560254,  0.03954495,
         0.45632064,  0.00710693, -0.87317663,  0.77282095,  0.13149685]],
      dtype=float32)
```

```
tf.nn.softmax(predictions).numpy()
```

```
array([[0.12837866, 0.11300021, 0.08762614, 0.05231675, 0.08757223,
        0.13285252, 0.08477715, 0.03515415, 0.18231574, 0.0960065 ]],
      dtype=float32)
```

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tru


loss_fn(y_train[:1], predictions).numpy()
```

```
2.0185156
```

```
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])


model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2982 - acc
Epoch 2/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1436 - acc
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1094 - acc
Epoch 4/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0904 - acc
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0754 - acc
<keras.callbacks.History at 0x7fbafd452c50>
```

```
probability_model = tf.keras.Sequential([
  model,
```

```
  tf.keras.layers.Softmax()
])

probability_model(x_test[:5])
```

```
<tf.Tensor: shape=(5, 10), dtype=float32, numpy=
array([[1.17722628e-07, 1.66647071e-10, 1.26441500e-05, 9.76931420e-04,
        8.29481189e-12, 5.10077234e-06, 4.58808555e-13, 9.98998702e-01,
        1.13544763e-06, 5.43147962e-06],
       [1.13564219e-07, 3.86933243e-05, 9.99946475e-01, 1.03060775e-05,
        2.07761300e-16, 1.15454463e-06, 9.60845114e-07, 6.19574596e-13,
        2.21347136e-06, 1.79281401e-14],
       [2.36739902e-06, 9.96731758e-01, 2.37533241e-04, 3.11529730e-05,
        4.80432209e-05, 4.48161118e-05, 4.09462227e-05, 1.17580697e-03,
        1.68568140e-03, 1.96545579e-06],
       [9.99839664e-01, 2.31955621e-10, 6.92707326e-05, 6.16646503e-08,
        4.50022611e-07, 9.49757305e-06, 1.55394753e-06, 7.79129696e-05,
        8.50616004e-08, 1.46252705e-06],
       [5.72809313e-06, 1.41032253e-09, 2.57200281e-05, 2.28989791e-07,
        9.95049894e-01, 1.59290885e-05, 6.14766213e-06, 1.64833997e-04,
        5.34693281e-05, 4.67795460e-03]], dtype=float32)>
```

✓ 0s completed at 1:13 PM