

Tushar Goyal, IIT MANDI, CSE

## Sentiment Analysis using Neural Networks

---

So here we used IMDB dataset, and are gonna do binary classification on movie reviews. We are going to import the dataset from Keras, we can download directly from the internet as well.

### Data Preparation :

```

Classifying Movie Reviews
Binary Classification on IMDB Dataset
Inputs: 50,000 Reviews
Output: Positive and Negative

Sentiment Analysis
Analysing a given set of words to predict the sentiment in the paragraph.

IMDB Large Movie Dataset

- The dataset has a huge number of 50,000 reviews
- All of these reviews are in English, polarised labelled reviews

Load the IMDB Dataset With Keras

- Keras provides access to the IMDB dataset built-in.
- The keras.datasets.imdb.load_data() allows you to load the dataset in a format that is ready for use in neural network and deep learning models.
- The words have been replaced by integers that indicate the absolute popularity of the word in the dataset. The sentences in each review are therefore comprised of a sequence of integers.



In [11]: from keras.datasets import imdb

Using TensorFlow backend.

Data Preperation

In [12]: ((Xt,Yt) , (Xt,Yt)) = imdb.load_data(num_words = 10000)

In [13]: import numpy as np
print(len(Xt))
print(len(Yt))
print(Xt[0])

20000
[0, 24, 22, 16, 43, 338, 673, 1632, 1285, 65, 458, 4469, 66, 2941, 4, 173, 26, 220, 5, 25, 189, 43, 838, 112, 59, 679, 2, 6, 25, 489, 284, 5, 158, 4, 172, 112, 187, 2, 235, 285, 39, 4, 172, 4535, 1111, 17, 546, 28, 12, 447, 4, 192, 59, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 189, 5, 4, 2223, 5244, 16, 489, 66, 3787, 31, 4, 139, 12, 16, 38, 619, 5, 25, 124, 31, 39, 135, 44, 25, 1415, 35, 6, 22, 12, 215, 26, 77, 52, 5, 14, 467, 16, 82, 2, 8, 4, 187, 117, 5952, 15, 256, 4, 1, 8, 594, 7480, 18, 4, 228, 22, 21, 134, 476, 26, 489, 5, 144, 38, 5535, 18, 51, 38, 28, 224, 82, 25, 184, 4, 226, 85, 16, 36, 1334, 88, 12, 16, 283, 5, 18, 4472, 113, 183, 32, 15, 16, 5343, 19, 178, 32]
```

Half of the reviews go in the training set and half of the reviews go in the test set.

- Every review is represented as a list of numbers, it does this according to a mapping which we say vocab
- Vocab gave by Keras, we can see using `imdb.get_word_index()` , it's given as word to index mapping, we make a reverse lookup map from this to convert review given as list of numbers to actual text.

```

[7] word_idx = imdb.get_word_index()

Downloading data from https://s3.amazonaws.com/text-datasets/imdb_word_index.json
1646592/1641221 [-----] - 0s 0un/step

[8] print(word_idx.items())

dict_items([('fawn', 14701), ('tsukino', 52006), ('nunnery', 52007), ('sonja', 16016), ('vani', 63951), ('woods', 1400), ('spiders', 16115), ('ha

[9] idx_word = dict([value,key] for (key,value) in word_idx.items())

[10] print(idx_word.items())

dict_items([(34701, 'fawn'), (52006, 'tsukino'), (52007, 'nunnery'), (16016, 'sonja'), (63951, 'vani'), (1400, 'woods'), (16115, 'spiders'), (234

actual_review = ' '.join([idx_word.get(idx-3, '')])
```

Our next step is , **To feed the review into MLP(neural network)** , then we should do vectorization of the review,we can't feed variable length reviews into classifier.

- Our vocab size is 10,000 so we make sure every review is vector of len 10k [0...0110010..10] of 0's and 1'st
- So like lets say review is "Movie was amazing" so Movie ki jo mapping hogi vocab me vector me us position ko 1 krdenge , like say Movie is mapped to 113 then 113 is 1 in vector , similarly for all words

```
# Next Step - Vectorize the Data
# Vocab size - 10,000 we will make every sentence is represented by vector of len 10k [0010100..1....1..0..1..]

def vectorize_sentences(sentences,dim = 10000):
    outputs = np.zeros((len(sentences),10000))

    for i,idx in enumerate(sentences):

        outputs[i,idx] = 1

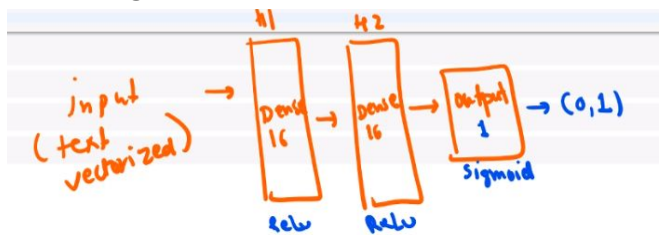
    return outputs

X_train = vectorize_sentences(XT)
X_test = vectorize_sentences(Xt)
print(X_train.shape)
print(X_train[0].shape)
X_train[0]

(25000, 10000)
(10000,)

array([0., 1., 1., ..., 0., 0., 0.])
```

## # Defining Model Architecture



### Define Your Model Architecture

- Use fully connected/Dense Layers with ReLU Activation
- 2 Hidden Layers with 16 Units each
- 1 Output layer with 1 unit(Sigmoid function)

```
from keras import models
from keras.layers import Dense

#define the model
model = models.Sequential()
model.add(Dense(16,activation = 'relu',input_shape = (10000,)))
model.add(Dense(16,activation = 'relu'))
model.add(Dense(1,activation = 'sigmoid'))

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

model.summary()
```

```
Model: "sequential_1"
Layer (type) Output Shape Param #
-----
dense_1 (Dense) (None, 16) 160016
dense_2 (Dense) (None, 16) 272
dense_3 (Dense) (None, 1) 17
-----
Total params: 160,305
Trainable params: 160,305
Non-trainable params: 0
```

## # TRAINING

## Training and Validation

Out of 25000 training points ,lets pick first 5000 for validation and 20k for training

```
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 = 15, batch_size = 512 , validation_data=(x_val,y_val) )
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/15: 20000/20000 [.....] - 2s 81ms/step - loss: 0.4683 - accuracy: 0.8932 - val_loss: 0.3445 - val_accuracy: 0.8728
Epoch 2/15: 20000/20000 [.....] - 1s 81ms/step - loss: 0.2881 - accuracy: 0.9183 - val_loss: 0.3290 - val_accuracy: 0.8648
Epoch 3/15: 20000/20000 [.....] - 1s 83ms/step - loss: 0.1987 - accuracy: 0.9298 - val_loss: 0.2783 - val_accuracy: 0.8888
Epoch 4/15: 20000/20000 [.....] - 1s 64ms/step - loss: 0.1618 - accuracy: 0.9452 - val_loss: 0.2916 - val_accuracy: 0.8842
Epoch 5/15: 20000/20000 [.....] - 1s 64ms/step - loss: 0.1344 - accuracy: 0.9534 - val_loss: 0.2872 - val_accuracy: 0.8912
Epoch 6/15: 20000/20000 [.....] - 1s 67ms/step - loss: 0.1165 - accuracy: 0.9613 - val_loss: 0.3228 - val_accuracy: 0.8808
Epoch 7/15: 20000/20000 [.....] - 1s 66ms/step - loss: 0.8993 - accuracy: 0.9671 - val_loss: 0.3382 - val_accuracy: 0.8822
Epoch 8/15: 20000/20000 [.....] - 1s 65ms/step - loss: 0.6858 - accuracy: 0.9736 - val_loss: 0.3668 - val_accuracy: 0.8752
Epoch 9/15: 20000/20000 [.....] - 1s 66ms/step - loss: 0.8739 - accuracy: 0.9768 - val_loss: 0.3788 - val_accuracy: 0.8838
Epoch 10/15: 20000/20000 [.....] - 1s 65ms/step - loss: 0.8844 - accuracy: 0.9884 - val_loss: 0.3938 - val_accuracy: 0.8812
Epoch 11/15: 20000/20000 [.....] - 1s 67ms/step - loss: 0.8543 - accuracy: 0.9837 - val_loss: 0.4218 - val_accuracy: 0.8788
Epoch 12/15: 20000/20000 [.....] - 1s 65ms/step - loss: 0.8488 - accuracy: 0.9851 - val_loss: 0.4520 - val_accuracy: 0.8774
Epoch 13/15: 20000/20000 [.....] - 1s 64ms/step - loss: 0.8281 - accuracy: 0.9902 - val_loss: 0.4879 - val_accuracy: 0.8728
Epoch 14/15: 20000/20000 [.....] - 1s 66ms/step - loss: 0.8333 - accuracy: 0.9908 - val_loss: 0.5159 - val_accuracy: 0.8722
Epoch 15/15: 20000/20000 [.....] - 1s 66ms/step - loss: 0.8267 - accuracy: 0.9930 - val_loss: 0.5454 - val_accuracy: 0.8738
```

## Visualising error and accuracy

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



## #Accuracy metrics →

```
[21]: model.evaluate(X_train ,Y_train)[1]

25000/25000 [=====] - 2s 99us/step

0.9729599952697754

[22]: model.evaluate(X_test,Y_test)[1]

25000/25000 [=====] - 2s 96us/step

0.853879988193512

[23]: model.predict(X_test)

array([[0.01978706],
       [0.99999917],
       [0.28758934],
       ...,
       [0.05373098],
       [0.0148973 ],
       [0.8695117 ]], dtype=float32)
```