Sequence Classification of IMDB Reviews

Using RNNs, LSTMs and GRUs

Karthik Chaganti, 50169441, CSE 701 Project

Intention

On a given dataset,

➔ **Implement**
Notable Deep Learning Neural Models

➔ **Compare**
Their performances

➔ **Benchmark**
Their Losses, Speeds and ultimately their Accuracies

Objective

- To Classify the Large Movie Review Dataset's reviews as either Good or Bad
- This is analogous to classifying Sequences of Text a.k.a. Sequence Classification
- To use Word Embedding
- To use Deep Learning Techniques on top of the Word Embedding to train and then validate the prediction of classification

Sequence Classification

- One of the best ways to classify text is to represent it in the form of Sequences. But why?
  - Text makes sense only with context.
  - To preserve context, sequential representation is necessary.
  - To achieve this, the text is numerically represented so that they can be mapped together with ease in order to preserve context.

- Word Embedding
  - Classification is clustering. So we need to cluster similar words together while training by using Word Embedding.
  - Words are represented as Real-valued Vectors in High Dimensional Space.
  - Similarity between the original word and others(by meaning) maps to closeness in vector space.
  - Depending on the length of the dimension, more and more similar words can be clustered together!
  - Along with the sequential representation and Embedding, we train the model to classify them

The Large Movie Review Dataset from ai.stanford.edu

- Contains highly polarized (extremely Good / Bad) reviews
- 60% of reviews for Training
- Rest 40% for Testing
- top_K terms only can be used if needed as the words are numerically represented!
- Padding can be done to maintain constant max size of each review

In each review:

- Each word is represented as a number in order of the frequencies in the whole dataset
- One review represents one sequence
- During training, all the words in the Seqs. are mapped onto an embedding layer for clustering of similar words

Note

Neutral Reviews are eliminated for ease of use!

Polarized reviews help in faster convergence of the models during training

Keras & Theano

- Keras is a minimalist Python library for deep learning that can run on top of Theano or TensorFlow.
- Theano is a python library that is analogous to TensorFlow
- Keras can be set to run only on GPU or CPU or both
- CUDA is used to fit the models to run on GPU
- Used Nvidia Geforce GTX 960M with 600 Cores (on my Laptop)
- Also tried Nvidia Kepler GRID K520 on Amazon AWS EC2 Compute (g2.2x)

Models

- RNN
- RNN with Dropout
- LSTM
- LSTM with Dropout
- GRU
- GRU with Dropout

The MODEL

The Base model for all the implementations fairly remain the same as below:

- Top 10000 terms are only loaded
- All reviews are truncated and padded to a max length of 600 words
- Word Embed layer is created with vector dimension of 32
- Next layer is the RNN or LSTM layer with a default of 100 neurons selected for all types as the dataset is constant for all
- Next layer is the Dense layer in which each and every neuron is connected to the other as this is a binary classification problem
- Sigmoid activation is used in this stage
- log loss function is used as Loss Function called Binary Cross Entropy in keras
- Batch size is selected as 64 here by trial and error
- Model is fitted and then compared with labels
- Dropout is added when necessary

```
Using Theano backend.
Using gpu device 0: GeForce GTX 960M (CNMeM is enabled with initial size: 80.0% of memory, cuDNN 5005)
```

In [4]:
```
# load the dataset.
# Keep the top k highest frequency terms for ease of use.
top_k = 10000
(X_train, y_train),(X_test,y_test) = imdb.load_data(nb_words = top_k)
#print(X_train)
```

In [5]:
```
# truncate the excessive length reviews to a length of 600 words
max_rev_length = 600
X_train = sequence.pad_sequences(X_train, maxlen = max_rev_length)
X_test = sequence.pad_sequences(X_test, maxlen = max_rev_length)
```

In [6]:
```
# Design the model
# Using embedding layer, we can convert the number representation of words
# to real-valued vector space a.k.a. word embedding

# Create an embed vector of length 32
embed_length = 32
# Use the sequential Model. For bigger modesls, functional api can be used
model = Sequential()
# Add the embedding layer to represent the words
model.add(Embedding(top_k,embed_length,input_length = max_rev_length))
# add the RNN layer containing 100 mem units
model.add(SimpleRNN(100))
# Dense layer connects each input neuron to the output neurons.
# It deals with density of connections of neurons
# Here we are using connections on each neuron to every other output neuron
# Activation Function: Sigmoid
model.add(Dense(1,activation = 'sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
# train the model with no.of. epochs
model.fit(X_train,y_train,nb_epoch = 3, batch_size=64)
# Evaluate Model
scores = model.evaluate(X_test,y_test,verbose = 0)
print("Accuracy:  %.2f%%" % (scores[1]*100))
```

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
embedding_1 (Embedding)          (None, 600, 32)       320000      embedding_input_1[0][0]
____________________________________________________________________________________________________
simplernn_1 (SimpleRNN)          (None, 100)           13300       embedding_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 1)             101         simplernn_1[0][0]
====================================================================================================
Total params: 333401
____________________________________________________________________________________________________
None
Epoch 1/3
25000/25000 [==============================] - 29s - loss: 0.6814 - acc: 0.5467
Epoch 2/3
25000/25000 [==============================] - 29s - loss: 0.6088 - acc: 0.6654
Epoch 3/3
```

Code Snapshot

Observations

Model	No.of Epochs	Max Loss (1st Epoch)	Min Loss (Final Epoch)	Total Time per Epoch	Accuracy of Classification
RNN	3	0.6814	0.5854	30s	64.01%
RNN With Dropout (0.4)	4	0.7163	0.6967	28s	50.92%
LSTM	3	0.4660	0.1889	169s	86.67%
LSTM With Dropout (0.2)	5	0.6829	0.4002	161s	81.18%
GRU	3	0.4572	0.2045	135s	86.97%
GRU With Dropout ()	5	0.6679	0.3996	130s	82.06%