

Deep Learning

Objectives

Objectives

Keras is a model level library, providing high-level building blocks for developing deep learning models. It does not handle itself low-level operations such as convolutions Instead, it relies on a specialized, well-optimized library to do so, serving as the "backend engine" of Keras. Keras being modular, several different backend engines can be plugged seamlessly into Keras.

The objective of this lab is to use CNTK as the backend for Keras and implement sentiment analysis from movie reviews using Deep Learning.

At the end of this lab, you will know:

* About the IMDB sentiment analysis problem for natural language processing
* How to use word embedding for natural language problems
* How to train a recurrent convolutional network on the IMDB sentiment classification task

Problem

The IMDB dataset contains 25,000 highly polar moving reviews (good or bad) for training and the same amount again for testing. The problem is to determine whether a given moving review has a positive or negative sentiment.

The dataset is also known as the Large Movie Review Dataset and can be found at

http://ai.stanford.edu/~amaas/data/sentiment/

CNTK Backend

If you have run Keras at least once, you will find the Keras configuration file at:

~/.keras/keras.json

If it isn't there, you can create it. The default configuration file looks like this:

{

"image\_data\_format": "channels\_last",

"epsilon": 1e-07,

"floatx": "float32",

"backend": "tensorflow"

}

Simply change the field backend to "cntk", and Keras will use the new configuration next time you run any Keras code.

**keras.json details**

* image\_data\_format: string, either "channels\_last" or "channels\_first". It specifies which data format convention Keras will follow. (keras.backend.image\_data\_format() returns it.)
* For 2D data (e.g. image), "channels\_last" assumes (rows, cols, channels) while "channels\_first" assumes (channels, rows, cols).
* epsilon: float, a numeric fuzzing constant used to avoid dividing by zero in some operations.
* floatx: string, "float16", "float32", or "float64". Default float precision.
* backend: string, "tensorflow", "theano", or "cntk".

Loading IMDB Data

The train\_model() function loads the imdb dataset and builds a deep learning model. 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.

Reviews have been preprocessed, and each review is encoded as a [sequence](https://keras.io/preprocessing/sequence/) of word indexes (integers). For convenience, words are indexed by overall frequency in the dataset, so that for instance the integer "3" encodes the 3rd most frequent word in the data.

Parameters

* max\_features: sets the vocabulary size. If we are only interested in the first 5,000 most used words in the dataset, vocabulary size will be 5,000.
* maxlen: maxlen is used to truncate or pad the dataset to a length of maxlen for each observation using the sequence.pad\_sequences() function.
* batch\_size: number of samples that going to be propagated through the network.
* epochs: an arbitrary cutoff, generally defined as "one pass over the entire dataset”
* embedding\_dims: dimension of the dense embedding
* filters: number output of filters in the convolution
* kernel\_size: length of the 1D convolution window
* hidden\_dims: used for the dimensionality of the output space of Dense layer

|  |
| --- |
| def train\_model():  #set parameters:  max\_features = 5000  maxlen = 400  batch\_size = 32  embedding\_dims = 50  filters = 250  kernel\_size = 3  hidden\_dims = 250  epochs = 2  print('Loading data...')  (x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=max\_features)  print(len(x\_train), 'train sequences')  print(len(x\_test), 'test sequences')  print('Pad sequences (samples x time)')  x\_train = sequence.pad\_sequences(x\_train, maxlen=maxlen)  x\_test = sequence.pad\_sequences(x\_test, maxlen=maxlen)  print('x\_train shape:', x\_train.shape)  print('x\_test shape:', x\_test.shape)  # build the model  print('Building model...')  model = Sequential()  # we start off with an efficient embedding layer which maps  # our vocab indices into embedding\_dims dimensions  model.add(Embedding(max\_features,  embedding\_dims,  input\_length=maxlen))  model.add(Dropout(0.2))    # we add a Convolution1D, which will learn filters  # word group filters of size filter\_length:  model.add(Conv1D(filters,  kernel\_size,  padding='valid',  activation='relu',  strides=1))  # we use max pooling:  model.add(GlobalMaxPooling1D())  # We add a vanilla hidden layer:  model.add(Dense(hidden\_dims))  model.add(Dropout(0.2))  model.add(Activation('relu'))    # We project onto a single unit output layer, and squash it with a sigmoid:  model.add(Dense(1))  model.add(Activation('sigmoid'))  model.compile(loss='binary\_crossentropy',  optimizer='adam',  metrics=['accuracy'])  model.fit(x\_train, y\_train,  batch\_size=batch\_size,  epochs=epochs,  validation\_data=(x\_test, y\_test))  return model |

Running train\_model() displays the following messages when loading data.

>>> model = train\_model()

Loading data...  
(25000, 'train sequences')  
(25000, 'test sequences')  
Pad sequences (samples x time)  
('x\_train shape:', (25000, 400))  
('x\_test shape:', (25000, 400))  
Building model...  
Train on 25000 samples, validate on 25000 samples  
Epoch 1/2  
25000/25000 [==============================] - 117s - loss: 0.4091 - acc: 0.7978 - val\_loss: 0.2758 - val\_acc: 0.8838  
Epoch 2/2  
25000/25000 [==============================] - 120s - loss: 0.2281 - acc: 0.9082 - val\_loss: 0.2716 - val\_acc: 0.8882

Model

In this lab, we build a sequential model which is a linear stack of layers. The model consists of the following types of layers:

1. Embedding layer that transforms words into their corresponding word embeddings. The weights of the Embedding layer are of the shape (vocabulary\_size, embedding\_dimension).
2. Dropout layer that includes regularization, which aims to reduce the complexity of the model with the goal to prevent overfitting.
3. ID Convolution layer. This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs.
4. Activation layer that applies an activation function to a layer.
5. Global max pooling layer

Sentiment Prediction

The sentiment prediction predict\_review() takes model and review\_text as arguments and performs sentiment prediction. The function first converts the review\_text to a vector using get\_vectors\_From\_text

|  |
| --- |
| def predict\_review(model,review\_text):  '''  Predict the sentiment of the review text.  @param  model: SequentialModel which we trained the data on.  review\_text: Review text to predict  @returns  sentiment score on the review text.  '''  # convert the review text into vector  x\_predict = get\_vectors\_from\_text([review\_text])[0]  # reshape the x\_predict  x\_predict = np.reshape(x\_predict,(1,len(x\_predict)))  # predict on the model  return model.predict(x\_predict)[0][0] |

The sentiment score for a sample text (‘I loved the movie’ and ‘I hated the movie’) can be obtained as shown below:

>>> parser = argparse.ArgumentParser()  
>>> parser.add\_argument("--test\_review", default='i loved the movie.')  
\_StoreAction(option\_strings=['--test\_review'], dest='test\_review', nargs=None, const=None, default='i loved the movie.', type=None, choices=None, help=None, metavar=None)  
>>> args = parser.parse\_args()  
>>> review\_text = args.test\_review  
>>> model = train\_model()  
Loading data...  
(25000, 'train sequences')  
(25000, 'test sequences')  
Pad sequences (samples x time)  
('x\_train shape:', (25000, 400))  
('x\_test shape:', (25000, 400))  
Building model...  
Train on 25000 samples, validate on 25000 samples  
Epoch ½  
25000/25000 [==============================] - 121s - loss: 0.4091 - acc: 0.7978 - val\_loss: 0.2758 - val\_acc: 0.8838  
Epoch 2/2  
25000/25000 [==============================] - 125s - loss: 0.2281 - acc: 0.9082 - val\_loss: 0.2716 - val\_acc: 0.8882  
>>> print predict\_review(model, review\_text)  
0.985743  
>>> print predict\_review(model, 'i hated the movie')  
0.268919