Keras Embeddings

Subash Gandyer
Data Scientist, HealthChain

Embeddings

A word embedding is a class of approaches for representing words and documents using a dense vector representation.

Traditional bag-of-word model encoding schemes have large sparse vectors.
Sparsity

In an embedding, words are represented by dense vectors.

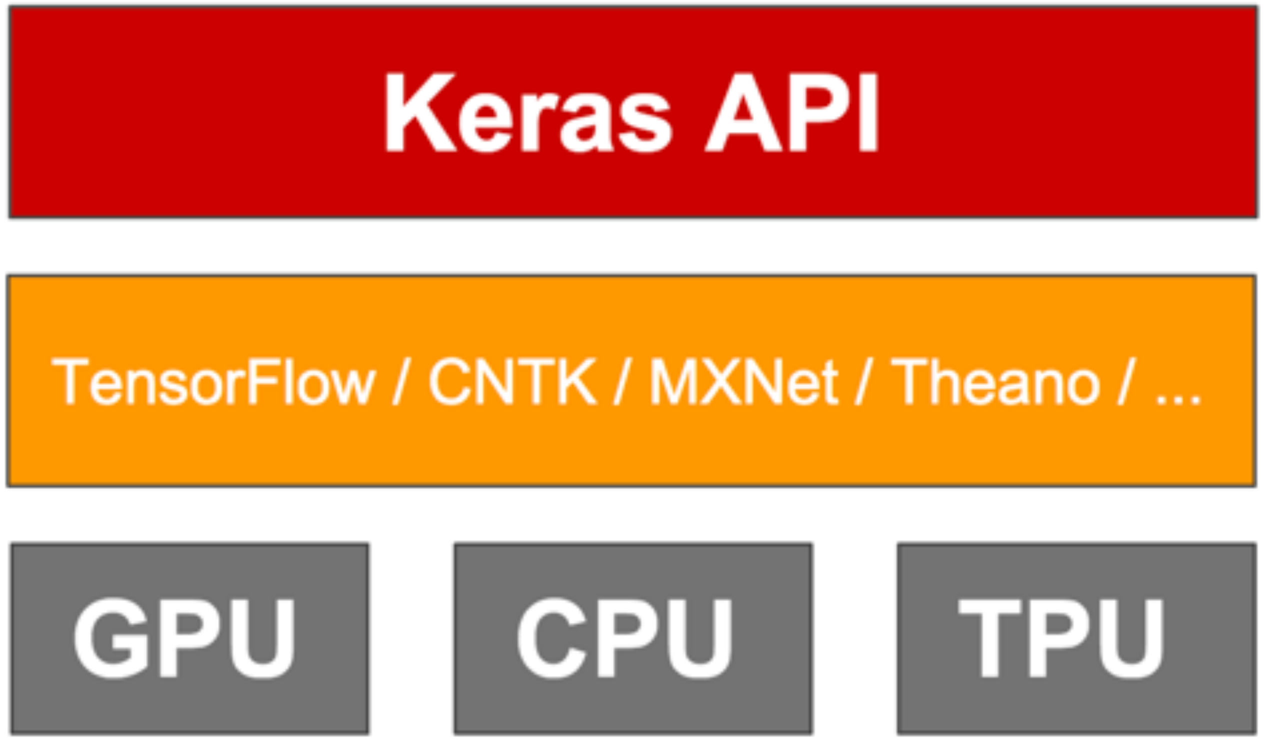
The position of a word and words that surround the word is learned.

The position of a word in the learned vector space is referred to as its embedding.

Examples of methods of learning word embeddings:

- Word2Vec
- GloVe

Keras



3 API Styles

The Sequential Model

- Dead simple
- Only for single-input, single-output, sequential layer stacks
- Good for 70+% of use cases

The functional API

- Like playing with Lego bricks
- Multi-input, multi-output, arbitrary static graph topologies
- Good for 95% of use cases

Model subclassing

- Maximum flexibility
- Larger potential error surface

Sequential API

```
import keras
from keras import layers

model = keras.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(10,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.fit(x, y, epochs=10, batch_size=32)
```

Functional API

```python
import keras
from keras import layers

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```

Model SubClassing

```
import keras
from keras import layers

class MyModel(keras.Model):

    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = layers.Dense(20, activation='relu')
        self.dense2 = layers.Dense(20, activation='relu')
        self.dense3 = layers.Dense(10, activation='softmax')

    def call(self, inputs):
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.fit(x, y, epochs=10, batch_size=32)
```

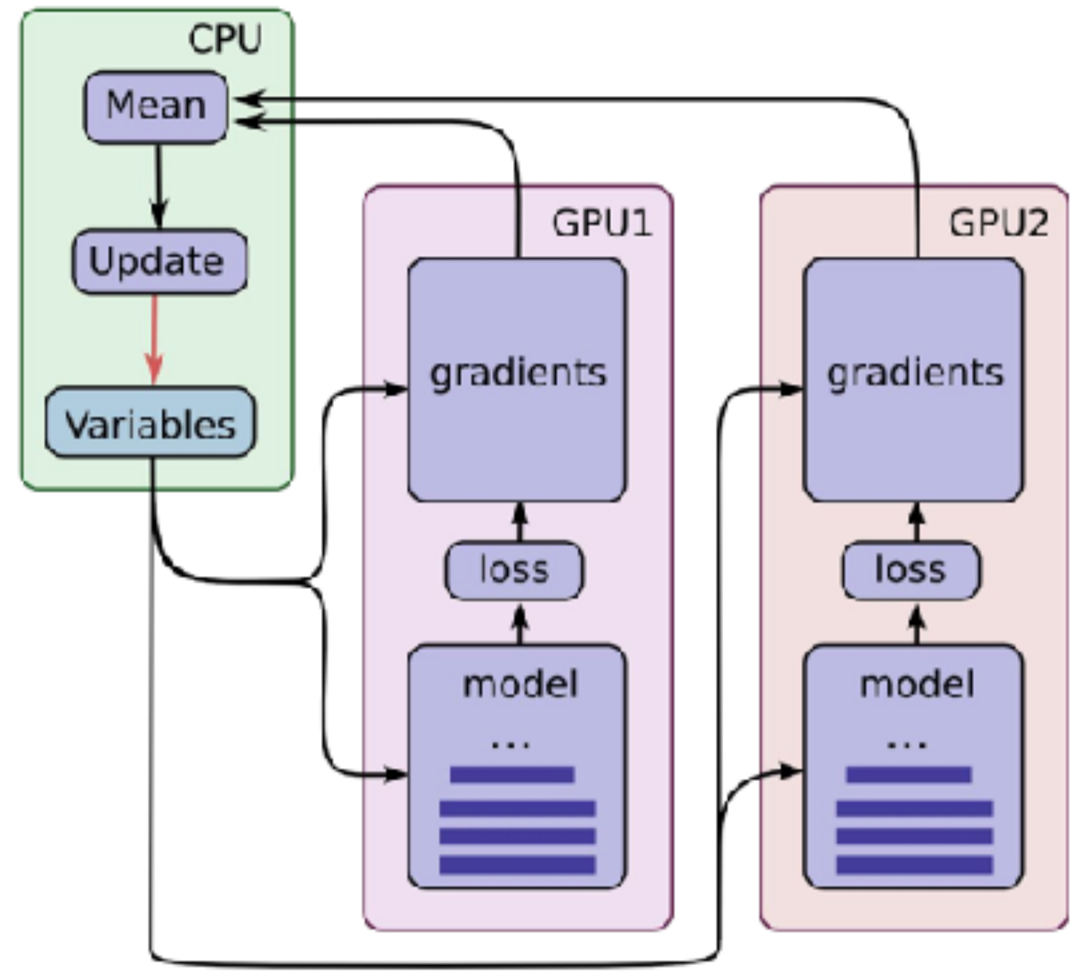
Parallel Computing

```
import tensorflow as tf
from keras.applications import Xception
from keras.utils import multi_gpu_model

# Instantiate the base model
# (here, we do it on CPU, which is optional).
with tf.device('/cpu:0'):
    model = Xception(weights=None,
                     input_shape=(height, width, 3
                     classes=num_classes)

# Replicates the model on 8 GPUs.
# This assumes that your machine has 8 available GPUs.
parallel_model = multi_gpu_model(model, gpus=8)
parallel_model.compile(loss='categorical_crossentropy',
                       optimizer='rmsprop')

# This `fit` call will be distributed on 8 GPUs.
# Since the batch size is 256, each GPU will process 32 samples.
parallel_model.fit(x, y, epochs=20, batch_size=256)
```



Ways of using embeddings

- Learn a word embedding that can be saved and used in another model.
- Learn the embedding along with the model.
- Load a pre-trained word embedding model (transfer learning).

Keras Embedding layer

Embedding layer is initialized with random weights and these are learned

3 arguments
1. **input_dim** : vocabulary size of the dataset
2. **output_dim** : Embedding dimension
3. **input_length** : Length of input sequences

```
e = Embedding(100, 32, input_length=50)
```

Text Classification Example

Text Classification —> Documents with labels (Sentiment)

```
# define corpus
docs = ['Well done!',
        'Good work',
        'Great effort',
        'nice work',
        'Excellent!',
        'Weak',
        'Poor effort!',
        'not good',
        'poor work',
        'Could have done better.']

# define class labels
labels = array([1,1,1,1,1,0,0,0,0,0])
```

Steps

- 1. Integer encode every document (One-hot encoding, Counts, TF-IDF)
- 2. Pad sequences to make every input same length
- 3. Define the model
- 4. Compile the model
- 5. Fit the model
- 6. Evaluate the model

Let’s build it

Keras-OwnEmbedding.ipynb

Keras - Glove Embeddings

- Idea: Download **GloVe** embeddings and seed the Keras ***Embedding*** layer with weights from the pre-trained embedding for the words in the training dataset

Steps

- 1. Download Glove embeddings
- 2. Integer encode the documents
- 3. Pad sequences to make every input same length
- 4. Load GloVe embeddings into memory
- 5. Create Embedding matrix for the training dataset
- 6. Define Embedding layer
- 7. Define the model
- 8. Compile the model
- 9. Fit the model
- 10. Evaluate the model

Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl/1.0/.
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip

glove.6B.50d.txt

the -0.038194 -0.24487 0.72812 -0.39961 0.083172 0.043953 -0.39141 0.3344 -0.57545 0.087459 0.28787 -0.06731 0.30906 -0.26384 -0.13231 -0.20757 0.33395 -0.33848 -0.31743 -0.48336 0.1464 -0.37304 0.34577 0.052041 0.44946 -0.46971 0.02628 -0.54155 -0.15518 -0.14107 -0.039722 0.28277 0.14393 0.23464 -0.31021 0.086173 0.20397 0.52624 0.17164 -0.082378 -0.71787 -0.41531 0.20335 -0.12763 0.41367 0.55187 0.57908 -0.33477 -0.36559 -0.54857 -0.062892 0.26584 0.30205 0.99775 -0.80481 -3.0243 0.01254 -0.36942 2.2167 0.72201 -0.24978 0.92136 0.034514 0.46745 1.1079 -0.19358 -0.074575 0.23353 -0.052062 -0.22044 0.057162 -0.15806 -0.30798 -0.41625 0.37972 0.15006 -0.53212 -0.2055 -1.2526 0.071624 0.70565 0.49744 -0.42063 0.26148 -1.538 -0.30223 -0.073438 -0.28312 0.37104 -0.25217 0.016215 -0.017099 -0.38984 0.87424 -0.72569 -0.51058 -0.52028 -0.1459 0.8278 0.27062

Integer encode documents

Map words to integers as well as integers to words.

Tokenizer class fits on the training data, converts text to sequences consistently by calling the *texts_to_sequences()* method on the *Tokenizer* class

Provides access to the dictionary mapping of words to integers in a *word_index* attribute.

```
t = Tokenizer()
t.fit_on_texts(docs)
vocab_size = len(t.word_index) + 1

encoded_docs = t.texts_to_sequences(docs)
print(encoded_docs)
```

Load embedding to memory

```
embeddings_index = dict()
f = open('glove.6B.100d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print(f'Loaded {len(embeddings_index)} word vectors.')
```

Create Embedding matrix

Create a matrix of one embedding for each word in the training dataset.
1. Enumerate all unique words in the *Tokenizer.word_index*
2. Locate the embedding weight vector from the loaded GloVe embeddings

```
embedding_matrix = zeros((vocab_size, 100))
for word, i in t.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

Embedding Layer

```
e = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=4,
trainable=False)
```

Let’s build it

Keras-GloveEmbedding.ipynb