Text Classification with Neural Networks

Basic Architectures

Train embedding layer from scratch

```
# integer encode the documents
vocab_size = 50

# pad documents to a max length of 4 words
max_length = 4

# define the model
model = Sequential()
model.add(Embedding(vocab_size, 8, input_length=max_length))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 4, 8)              400
_________________________________________________________________
flatten_1 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
```

← **Param# = vocab_size * embedding_dimension**

Load pre-trained embeddings

```
# load the whole embedding into memory
embeddings_index = dict()
f = open('glove.6B.100d.txt')
…….
```

```
# define model
model = Sequential()
e = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=4, trainable=False)
model.add(e)
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Freeze or unfreeze for fine-tuning

Embedding layer + CNN

```
# define CNN model
model = Sequential()
model.add(embedding_layer)
model.add(Conv1D(filters=128, kernel_size=5, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Embedding layer + RNN

```
model = keras.Sequential()
# Add an Embedding layer expecting input vocab of size 1000, and
# output embedding dimension of size 64.
model.add(layers.Embedding(input_dim=1000, output_dim=64))

model.add(layers.SimpleRNN(128))

model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

RNN evolution

- **RNN** - Sequential, fail to memorize long range information.
- **LSTM** - uses a cell state to convey the long range information but with increased number of parameters.
- **CNN** - multpile short kernels on few word windows, difficult to model long sequences.

- **Attention** - allows the neural network to focus its attention on particular past inputs and ignore the others.
- **Transformers** – parallel process (not sequential), self-attention.