EDAN95
Applied Machine Learning
http://cs.lth.se/edan95/
Lecture 9: Autoencoders and Generative Learning

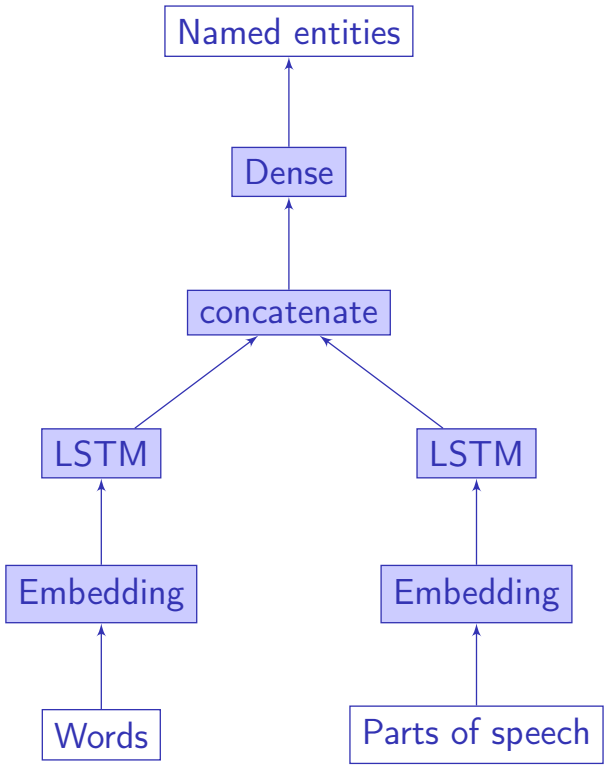
Pierre Nugues

Lund University
Pierre.Nugues@cs.lth.se
http://cs.lth.se/pierre_nugues/

December 3, 2018

The Functional Model

So far, we have used the `Sequential` model to build networks
These models correspond to pipelines with one input and one output



To build graphs, we need to use the functional model.

Comparing the Models

Sequential:

```
seq_model = Sequential()
seq_model.add(layers.Dense(32, activation='relu',
  input_shape=(64,)))
seq_model.add(layers.Dense(32, activation='relu'))
seq_model.add(layers.Dense(10, activation='softmax'))
```

Functional:

```
input_tensor = Input(shape=(64,))
x = layers.Dense(32, activation='relu')(input_tensor)
x = layers.Dense(32, activation='relu')(x)
output_tensor = layers.Dense(10, activation='softmax')(x)
model = Model(input_tensor, output_tensor)
```

From Chollet, page 237

Building a Multi Input Model

To build a multi input, we need the functional model and at a certain point, merge the branches with `layers.concatenate()` function
We will now build a NER tagger that uses two inputs: the words and parts of speech

The Word Branch

```
text_vocabulary_size = len(word_set) + 2
text_input = Input(shape=(None,), dtype='int32', name='text')
embedded_text = layers.Embedding(text_vocabulary_size,
                                 64, mask_zero=True)(text_input)
encoded_text = layers.LSTM(32,
                           return_sequences=True)(embedded_text)
```

The Part-of-Speech Branch

```
pos_vocabulary_size = len(pos_set) + 2
pos_input = Input(shape=(None,),
                  dtype='int32',
                  name='pos')
embedded_pos = layers.Embedding(pos_vocabulary_size,
                                32, mask_zero=True)(pos_input)
encoded_pos = layers.LSTM(16,
                  return_sequences=True)(embedded_pos)
```

Merging and Common Part

```
concatenated = layers.concatenate(
            [encoded_text, encoded_pos],axis=-1)
ner_vocabulary_size = len(ner_set) + 2
ner = layers.Dense(ner_vocabulary_size,
                   activation='softmax')(concatenated)

model = Model([text_input, pos_input], ner)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['acc'])
```

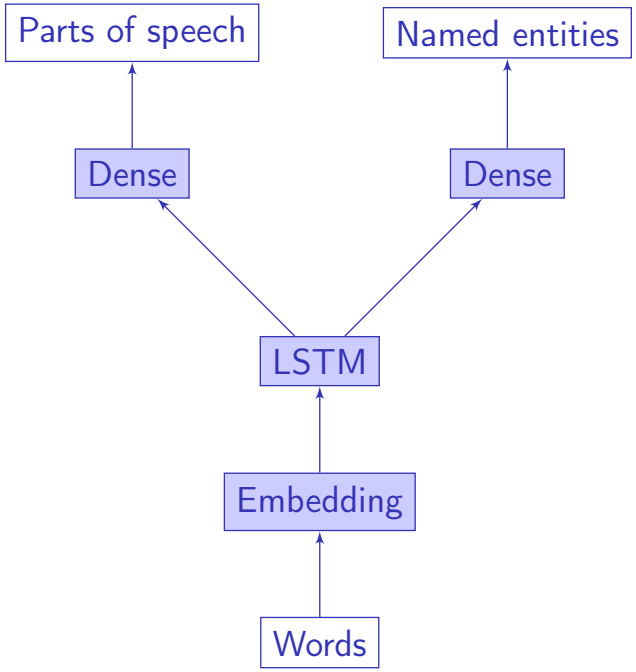
Code Example

The NER tagger with two inputs: the words and parts of speech and we will compare it to a sequential model

Jupyter Notebooks: `5.2-multiinput.ipynb` and `5.3-monoinput.ipynb`

Multiple Outputs

It is also possible to build a model with multiple outputs, for instance the word as input to predict the parts of speech and the named entities.



The Word Input

```
text_vocabulary_size = len(word_set) + 2
text_input = Input(shape=(None,), dtype='int32', name='text')
embedded_text = layers.Embedding(text_vocabulary_size,
                                 64, mask_zero=True)(text_input)
encoded_text = layers.LSTM(32,
                           return_sequences=True)(embedded_text)
```

The POS output

```
pos_vocabulary_size = len(pos_set) + 2
pos_output = layers.Dense(pos_vocabulary_size,
                          activation='softmax',
                   name='pos')(encoded_text)
```

The NER Output

```
ner_vocabulary_size = len(ner_set) + 2
ner_output = layers.Dense(ner_vocabulary_size,
                          activation='softmax',
                          name='ner')(encoded_text)
```

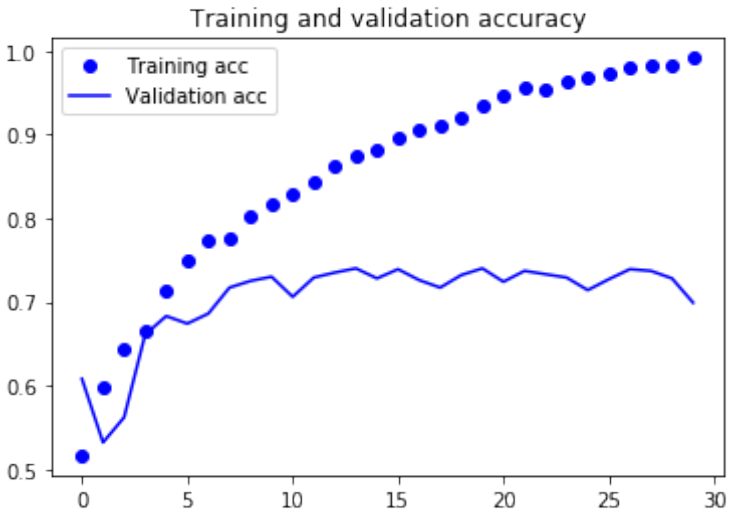
The Model

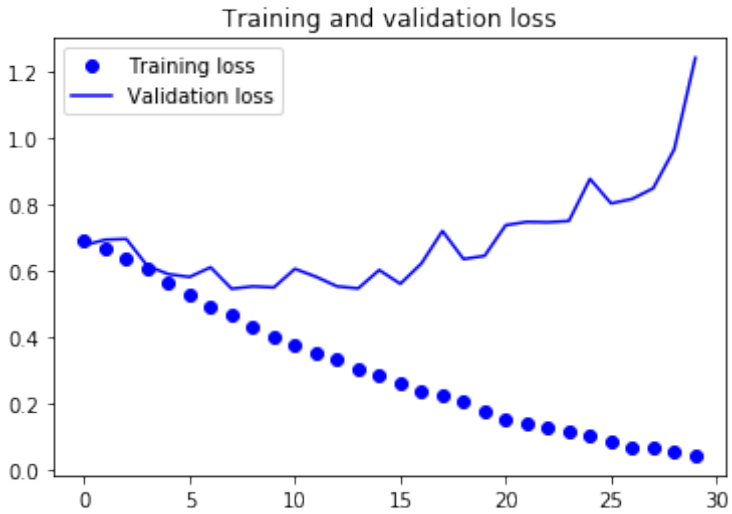
```
model = Model(text_input, [pos_output, ner_output])
model.compile(optimizer='rmsprop',
              loss=['categorical_crossentropy',
                    'categorical_crossentropy'],
              metrics=['acc'])

model.fit(X_words_idx,
          {'pos':Y_pos_idx_cat, 'ner':Y_ner_idx_cat},
          epochs=3, batch_size=128)
```

It is possible to build mode complex models, provided that they have the form of a directed acyclic graph. See the book.

Monitoring Training

We have seen different shapes of validation accuracies and loss:





15 epochs seem the optimal number and it is probably useless to run more. Keras provided callbacks for this.

Two Callbacks

1. `keras.callbacks.EarlyStopping` to stop training when validation scores do not improve;
2. `keras.callbacks.ModelCheckpoint` to save models

```
callbacks_list = [
    keras.callbacks.EarlyStopping(
        monitor='acc',
        patience=1,),
    keras.callbacks.ModelCheckpoint(
        filepath='my_model.h5',
        monitor='val_loss',
        save_best_only=True,)
 ]
```

From Chollet, page 250

Including the Callbacks

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
model.fit(x, y,
          epochs=10,
          batch_size=32,
          callbacks=callbacks_list,
          validation_data=(x_val, y_val))
```

You can also write your own callbacks, see Chollet, page 251-252

Tensorboard

Tensorboard is a visualization tool
You include it with a callback

```
callbacks = [
    keras.callbacks.TensorBoard(
        log_dir='tb_log_folder',
        histogram_freq=1
) ]
```

Demonstration

Tensorboard

Generative Learning

???

???