Module IV

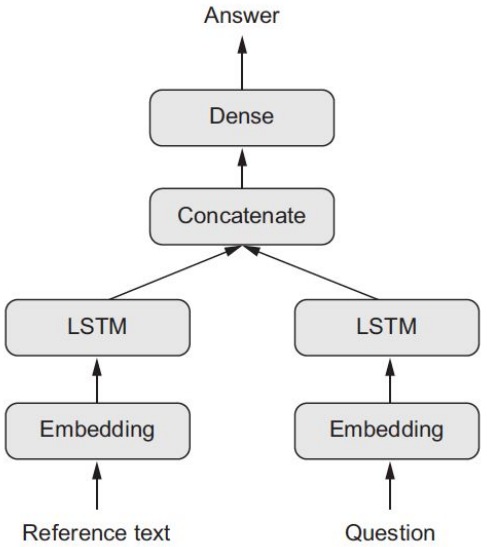
Deep Learning Practices

Multi-input and Multi-output Models

Multi-input Models

- Functional API can be used to build models with multiple inputs
- At some point we merge their different input branches
  - Using `keras.layers.add` or `keras.layers.concatenate`

- An example: A question-answering model
  - A typical question-answering model has two inputs
    - a natural-language question and
    - a text snippet (such as a news article) providing information to be used for answering the question
  - The model must then produce an answer:
    - In the simplest possible setup, this is a one-word answer obtained via a softmax over some predefined vocabulary



Why can/may we not merge the two inputs to one and submit?

Multi-input Models

```
from keras import Input

text_vocabulary_size = 10000
question_vocabulary_size = 10000
answer_vocabulary_size = 500

text_input = Input(shape=(None,), dtype='int32', name='text')

embedded_text = layers.Embedding(
    64, text_vocabulary_size)(text_input)

encoded_text = layers.LSTM(32)(embedded_text)

question_input = Input(shape=(None,),
                       dtype='int32',
                       name='question')

embedded_question = layers.Embedding(
    32, question_vocabulary_size)(question_input)
encoded_question = layers.LSTM(16)(embedded_question)

concatenated = layers.concatenate([encoded_text, encoded_question],
                                  axis=-1)

answer = layers.Dense(answer_vocabulary_size,
                      activation='softmax')(concatenated)

model = Model([text_input, question_input], answer)

model.fit([text, question], answers, epochs=10, batch_size=128)
```

The text input is a variable-length sequence of integers. Note that you can optionally name the inputs.

Embeds the inputs into a sequence of vectors of size 64

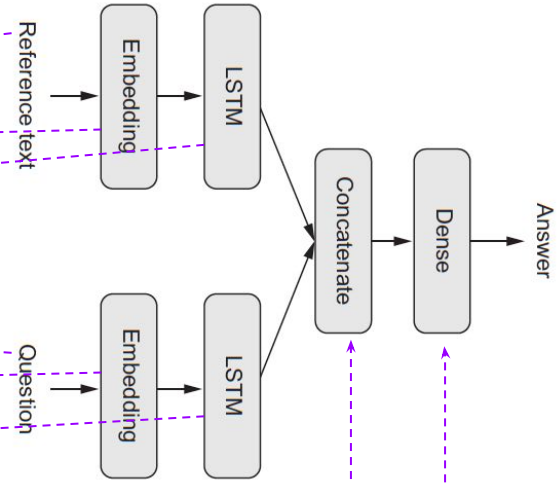
Encodes the vectors in a single vector via an LSTM

Same process (with different layer instances) for the question

Concatenates the encoded question and encoded text

Adds a softmax classifier on top

At model instantiation, you specify and the output.



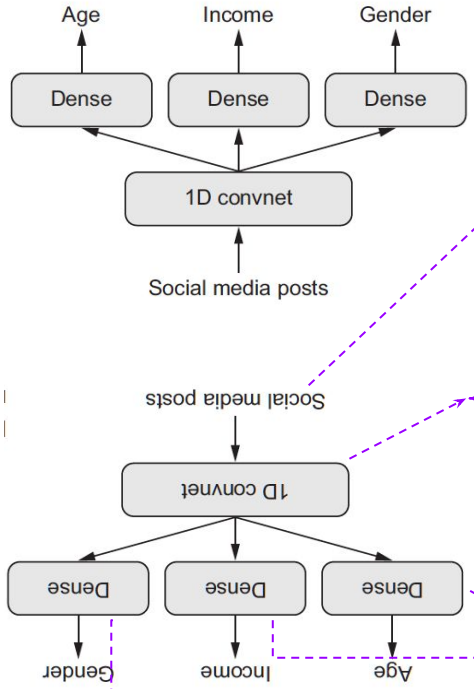
Applications of 1D CNNs

- A 1D CNN is very effective when
  - you expect to derive interesting features from shorter (fixed-length) segments of the overall data set and
  - where the location of the feature within the segment is not of high relevance
- This applies well to the analysis of time sequences of sensor or signal data over a fixed-length period
  - For example, audio signals
- Another application is NLP
  - Although here LSTM networks are more promising

```
"? this film was just brilliant casting location scenery story direction everyone's really suited
the part they played and you could just imagine being there robert ? is an amazing actor and now t
he same being director ? father came from the same scottish island as myself so i loved the fact t
here was a real connection with this film the witty remarks throughout the film were great it was
just brilliant so much that i bought the film as soon as it was released for ? and would recommend
it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and
you know what they say if you cry at a film it must have been good and this definitely was also ?
to the two little boy's that played the ? of norman and paul they were just brilliant children are
often left out of the ? list i think because the stars that play them all grown up are such a big
profile for the whole film but these children are amazing and should be praised for what they have
done don't you think the whole story was so lovely because it was true and was someone's life afte
r all that was shared with us all"
```

What are some other applications of 1D CNNs?

Multi-output Models




```
posts_input = Input(shape=(None,), dtype='int32', name='posts')
embedded_posts = layers.Embedding(256, vocabulary_size)(posts_input)
x = layers.Conv1D(128, 5, activation='relu')(embedded_posts)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation='relu')(x
```

Note that the output layers are given names.

```
age_prediction = layers.Dense(1, name='age
income_prediction = layers.Dense(num_income_groups,
                                 activation='softmax',
                                 name='income')(x)
gender_prediction = layers.Dense(1, activation='sigmoid', name='gender')(x)
```

```
model = Model(posts_input,[age_prediction, income_prediction, gender_prediction])
```

Multi-output Models

```
model.compile(optimizer='rmsprop', loss = ['mse', 'categorical_crossentropy', 'binary_crossentropy'])
```

age income gender

The problem of Imbalanced Loss Contributions

Model will be optimized preferentially for the task with the largest individual loss, at the expense of the other tasks

What are some ways we can remedy this?


```
model.compile(optimizer = 'rmsprop', loss = ['mse', 'categorical_crossentropy', 'binary_crossentropy']
                                    , loss_weights = [0.25, 1., 10.] )
```

```
model.fit(posts, [age_targets, income_targets, gender_targets], epochs = 10, batch_size = 64)
```

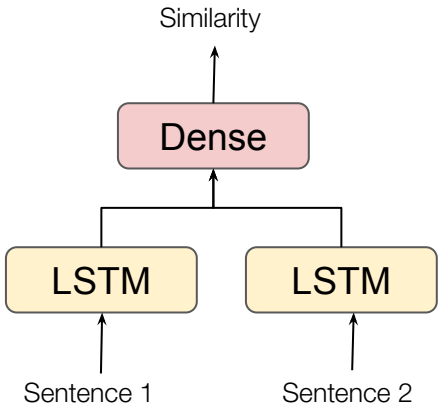
How will model.fit look like if we have multi-input & multi-output type problem?

Layer Weight Sharing (The *Siamese* LSTM)

Sentence Similarity Problem

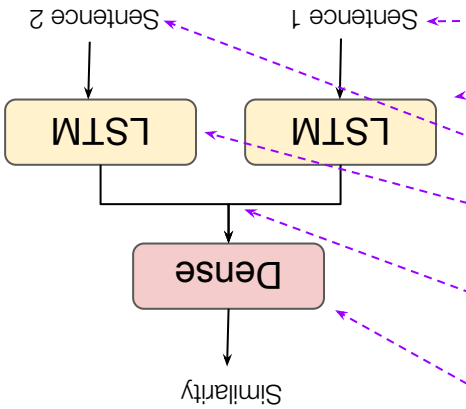
Example: Assess the semantic similarity between two sentences

- The model has two inputs
  - the two sentences to compare
- It outputs a score between 0 and 1
  - where 0 means unrelated sentences and 1 means sentences that are either identical or reformulations of each other
- Where can this be useful (application)?
  - For duplicating natural-language queries in a dialog system
- A unique characteristic of this problem:
  - the two input sentences are interchangeable
  - because semantic similarity is a symmetrical relationship:
    - similarity of A to B is identical to similarity of B to A



Does it make sense to learn two independent models for processing each input sentence?

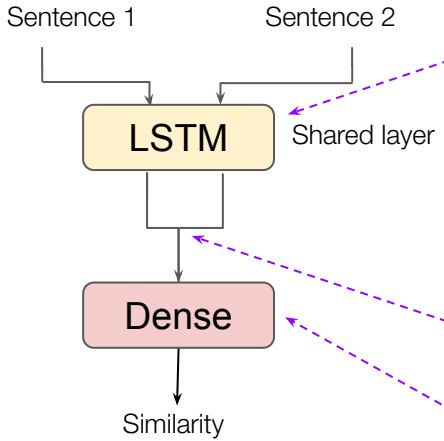
No Weight Sharing




```
from keras import layers
from keras import Input
from keras.models import Model

left_input = Input(shape=(None, 128))
left_output = layers.LSTM(32)(left_input)

right_input = Input(shape=(None, 128))
right_output = layers.LSTM(32)(right_input)

merged = layers.concatenate([left_output, right_output], axis=-1)

predictions = layers.Dense(1, activation='sigmoid')(merged)

model = Model([left_input, right_input], predictions)
model.fit([left_data, right_data], targets)
```

Layer Weight Sharing (Siamese LSTM)

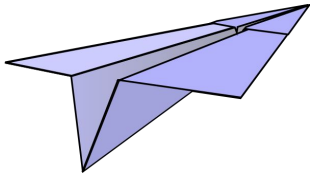
When you train such a model, the weights of the LSTM layer are updated based on both inputs




```
from keras import layers
from keras import Input
from keras.models import Model

lstm = layers.LSTM(32)
left_input = Input(shape=(None, 128))
left_output = lstm(left_input)
right_input = Input(shape=(None, 128))
right_output = lstm(right_input)

merged = layers.concatenate([left_output, right_output], axis=-1)

predictions = layers.Dense(1, activation='sigmoid')(merged)

model = Model([left_input, right_input], predictions)
model.fit([left_data, right_data], targets)
```

Keras Callbacks

Without Callbacks



With Callbacks

Launching **`train.fit()`** on a large dataset for tens of epochs - past the initial impulse, you don't have any control over its trajectory or its landing spot

Model Checkpointing

So far..

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

model.save( ‘model-file-name.h5’ )
```

Model Checkpointing is saving weights of the model during training

```
a = keras.callbacks.ModelCheckpoint( filepath='my_model.h5', monitor='val_loss', save_best_only=True )

model.fit(x, y, epochs=10, batch_size=32, callbacks=[a], validation_data=(x_val, y_val))
```

What happens if we monitor **‘loss’** instead of **‘val_loss’**?

How to download models from Google Colab?

```
from google.colab import files
files.download("model.h5")
```

Early Stopping

So far..

```
model.fit(x, y, epochs=10000, batch_size=32, …)
```

Early Stopping is interrupting training once a target metric being monitored has stopped improving for a fixed number of epochs

```
b = keras.callbacks.EarlyStopping( monitor = 'val_acc', patience = 100 )

model.fit(x, y, epochs=10, batch_size=32, callbacks=[b], validation_data=(x_val, y_val))
```

What happens if we choose **'acc'** instead of **'val_acc'**?

Reduce Learning Rate on Plateau

- Use this callback to reduce the learning rate when the validation loss has stopped improving
- Reducing or increasing the learning rate in case of a loss plateau is is an effective strategy to get out of local minima during training

```
keras.callbacks.ReduceLROnPlateau( monitor = 'val_loss', factor = 0.1, patience = 10 )
```

Divide the learning rate by 10

Do it when ‘val_loss’ does not improve for 10 epochs

Writing Custom Callbacks

`on_epoch_begin`	Called at the start of every epoch
`on_epoch_end`	Called at the end of every epoch
`on_batch_begin`	Called right before processing each batch
`on_batch_end`	Called right after processing each batch
`on_train_begin`	Called at the start of training
`on_train_end`	Called at the end of training

Example: You have a large dataset and would like to plot 'predictions' vs 'true' values after each epoch to observe the progress

Solution: Overwrite the 'on_epoch_end' method with all that you want to do

DEMO

395 lines (395 sloc) | 96.6 KB

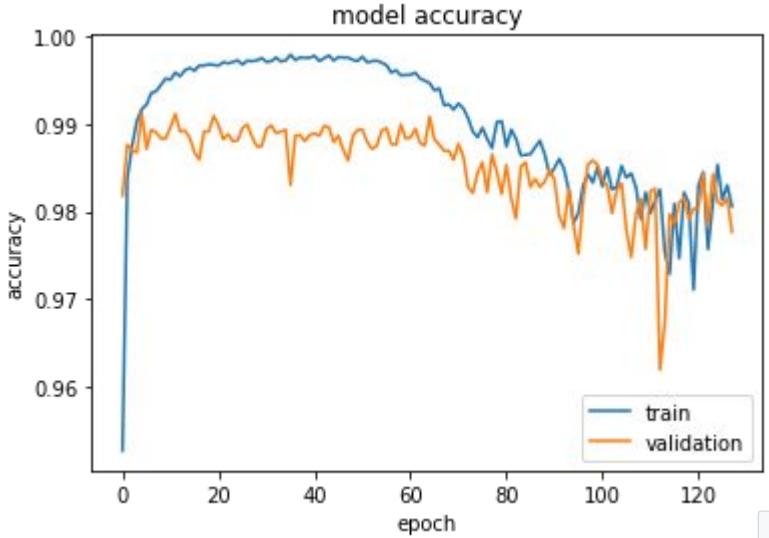
Early Stopping and Model Checkpointing - Pima Indians Diabetes Classification with Callbacks

In [1]:

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import numpy as np
np.random.seed(7)

print('')
print('Load pima indians dataset..')
dataset = np.loadtxt("https://raw.githubusercontent.com/badriadhikari/2019-Spring-DL/master/course
_content/module1_intro2ML/pima-indians-diabetes.csv", delimiter=",")
print(dataset.shape)

print('')
print('Preview first 5 rows and all columns..')
np.set_printoptions(precision = 4, suppress = True)
print(dataset[0:5, :])
```

When Training Accuracy Increase, Then Decrease?

DEMO



785 lines (785 sloc) | 117 KB

Raw Blame History

Open in Colab

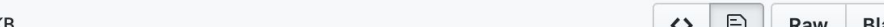
When Training Loss Decreases and Then Increases - MNIST Digit Classification with adam & rmsprop

In [1]:
```
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.layers import *
from keras.models import *
import matplotlib.pyplot as plt

( train_images, train_labels ), ( valid_images, valid_labels ) = mnist.load_data()

print('')
```

99.8% Accuracy on MNIST using ‘Callbacks’

DEMO

694 lines (694 sloc) | 56.3 KB



Early Stopping, Model Checkpointing & Reduce LR on Plateau - MNIST Digit Classification with Callbacks

In [1]:
```
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.layers import *
from keras.models import *
import matplotlib.pyplot as plt

( train_images, train_labels ), ( valid_images, valid_labels ) = mnist.load_data()
```

Display Shape Information in the Graph of Layers

DEMO

- A network is ‘directed acyclic graph’ of layers
  - Neural networks in Keras are allowed to be arbitrary directed acyclic graphs of layers
  - It’s impossible for a tensor x to become the input of one of the layers that generated x
- The only processing loops that are allowed are those internal to recurrent layers