__: Same Text __:Same Code

Koay Chin Yang: awd_lstm.py

```
#!/usr/bin/env python
# coding: utf-8

# # Main Code

# In[ ]:


from fastai import *








from fastai.text import *
from fastai.callbacks.tracker import EarlyStoppingCallback
from fastai.callbacks.tracker import SaveModelCallback
from fastai.callbacks.tracker import ReduceLROnPlateauCallback
import pandas as pd


# ## Preparing the data

# First let's load the dataset we are going to study. The dataset

has been curated by DBPedia

and contains a total of 45,000 company descriptions.
```

Fast.ai lesson3-imdb.py

```
#!/usr/bin/env python
# coding: utf-8

# # IMDB

# In[ ]:


get_ipython().run_line_magic('reload_ext', 'autoreload')
get_ipython().run_line_magic('autoreload', '2')
get_ipython().run_line_magic('matplotlib', 'inline')


# In[ ]:


from fastai.text import *


# ## Preparing the data

# First let's download the dataset we are going to study. The[dataset](http://ai.stanford.edu/~amaas/data/sentiment/) has been curated by Andrew Maas et al.
and contains a total of 100,000 reviews on IMDB. 25,000 of them are labelled as positive and negative for training, another 25,000 are labelled for testing (in both cases they are highly polarized). The remaning 50,000 is an additional unlabelled data (but we will find a use for it nonetheless).
#
# We'll begin with a sample we've prepared for you, so that things run quickly before going over the full dataset.
```

___: Same Text ___:Same Code

```
# In[ ]:
path =
untar_data(URLs.IMDB_SAMPLE)
path.ls()


# It only contains one csv file,
let's have a look at it.
```

```
# In[ ]:


df_lm =
pd.read_csv('company.csv')
df_lm.head()


# In[ ]:


from sklearn.model_selection
import train_test_split

lm_train, lm_valid =
train_test_split(df_lm,

test_size = 0.10,

random_state = 7)

print("Number of rows in train: "
+ str(len(lm_train)) + ", valid:
" + str(len(lm_valid)))
```

```
# In[ ]:


df =
pd.read_csv(path/'texts.csv')
df.head()


# In[ ]:


df['text'][1]


# It contains one line per
review, with the label
('negative' or 'positive'), the
text and a flag to determine if
it should be part of the
validation set or the training
set. If we ignore this flag, we
can create a DataBunch containing
this data in one line of code:

# In[ ]:


data_lm =
TextDataBunch.from_csv(path,
'texts.csv')


# By executing this line a
process was launched that took a
bit of time. Let's dig a bit into
it. Images could be fed (almost)
directly into a model because
they're just a big array of pixel
values that are floats between 0
and 1. A text is composed of
words, and we can't apply
mathematical functions to them
directly. We first have to
convert them to numbers. This is
done in two differents steps:
tokenization and
numericalization. A
```

: Same Text   :Same Code

```
`TextDataBunch` does all of that
behind the scenes for you.
#
# Before we delve into the
explanations, let's take the time
to save the things that were
calculated.

# In[ ]:


data_lm.save()


# Next time we launch this
notebook, we can skip the cell
above that took a bit of time
(and that will take a lot more
when you get to the full dataset)
and load those results like this:

# In[ ]:


data = TextDataBunch.load(path)


# ### Tokenization

# The first step of processing we
make texts go through is to split
the raw sentences into words, or
more exactly tokens. The easiest
way to do this would be to split
the string on spaces, but we can
be smarter:
#
# - we need to take care of
punctuation
# - some words are contractions
of two different words, like
isn't or don't
# - we may need to clean some
parts of our texts, if there's
HTML code for instance
#
# To see what the tokenizer had
done behind the scenes, let's
have a look at a few texts in a
batch.

# In[ ]:


data =
TextClasDataBunch.load(path)
data.show_batch()
```

: Same Text :Same Code

```
# The texts are truncated at 100
tokens for more readability. We
can see that it did more than
just split on space and
punctuation symbols:
# - the "'s" are grouped together
in one token
# - the contractions are
separated like his: "did", "n't"
# - content has been cleaned for
any HTML symbol and lower cased
# - there are several special
tokens (all those that begin by
xx), to replace unkown tokens
(see below) or to introduce
different text fields (here we
only have one).

# ### Numericalization

# Once we have extracted tokens
from our texts, we convert to
integers by creating a list of
all the words used. We only keep
the ones that appear at list
twice with a maximum vocabulary
size of 60,000 (by default) and
replace the ones that don't make
the cut by the unknown token
`UNK`.
#
# The correspondance from ids
tokens is stored in the `vocab`
attribute of our datasets, in a
dictionary called `itos` (for int
to string).

# In[ ]:


data.vocab.itos[:10]


# And if we look at what a what's
in our datasets, we'll see the
tokenized text as a
representation:

# In[ ]:


data.train_ds[0][0]


# But the underlying data is all
numbers

# In[ ]:
```

__: Same Text __:Same Code

```
data.train_ds[0][0].data[:10]
```

```
# ### With the data block API

# We can use the data block API with NLP and have a lot more flexibility than what the default factory methods offer. In the previous example for instance, the data was randomly split between train and validation instead of reading the third column of the csv.
#
# With the data block API though, we have to manually call the tokenize and numericalize steps. This allows more flexibility, and if you're not using the defaults from fastai, the variaous arguments to pass will appear in the step they're revelant, so it'll be more readable.

# In[ ]:


data = (TextList.from_csv(path, 'texts.csv', cols='text')
                .split_from_df(col=2)
                .label_from_df(cols=0)
                .databunch())
```

```
# ## Language Model

# In[ ]:


path = Path("/content")
```

```
# ## Language model

# Note that language models can use a lot of GPU, so you may need to decrease batchsize here.

# In[ ]:


bs=48


# Now let's grab the full dataset for what follows.

# In[ ]:


path = untar_data(URLs.IMDB)
path.ls()
```

__: Same Text __:Same Code

```
# In[ ]:


(path/'train').ls()
```

The reviews are in a training and test set following an imagenet structure. The only difference is that there is an `unsup` folder on top of `train` and `test` that contains the unlabelled data.
#

We're not going to train a model that classifies companies from scratch. Like in computer vision, we'll use a model pretrained on a bigger dataset (a cleaned subset of wikipedia called wikitext-103).

We're not going to train a model that classifies the reviews from scratch. Like in computer vision, we'll use a model pretrained on a bigger dataset (a cleaned subset of wikipeia called [wikitext-103](https://einstein.ai/research/blog/the-wikitext-long-term-dependency-language-modeling-dataset)). That model has been trained to guess what the next word, its input being all the previous words. It has a recurrent structure and a hidden state that is updated each time it sees a new word. This hidden state thus contains information about the sentence up to that point.
#

That model has been trained to guess what the next word, its input being all the previous words. It has a recurrent structure and a hidden state that is updated each time it sees a new word. This hidden state thus contains information about the sentence up to that point.
#

We are going to use that 'knowledge' of the English language to build our classifier, but first, like for computer vision, we need to fine-tune the pretrained model to our particular dataset. Because the English of the company descriptions isn't the same as the English of wikipedia, we'll need to adjust the parameters of our model by a little bit. Plus there might be some words that would be extremely common in the reviews dataset but would be barely present in wikipedia, and therefore might not be part of the vocabulary the model was trained on.
#

We are going to use that 'knowledge' of the English language to build our classifier, but first, like for computer vision, we need to fine-tune the pretrained model to our particular dataset. Because the English of the reviex lefts by people on IMDB isn't the same as the English of wikipedia, we'll need to adjust a little bit the parameters of our model. Plus there might be some words extremely common in that dataset that were barely present in wikipedia, and therefore might no be part of the vocabulary the model was trained on.

As we can use it to fine-tune our model. Let's create our data

This is where the unlabelled data is going to be useful to us, as we can use it to fine-tune our model. Let's create our data

: Same Text :Same Code

object with the data block API
(next line takes a few minutes).

```
# In[ ]:
```

object with the data block API
(next line takes a few minutes).

```
# In[ ]:


data_lm =
(TextList.from_folder(path)
            #Inputs: all the text
files in path
             .filter_by_folder(inc
lude=['train', 'test', 'unsup'])
            #We may have other
temp folders that contain text
files so we only keep what's in
train and test
             .random_split_by_pct(
0.1)
            #We randomly split and
keep 10% (10,000 reviews) for
validation
             .label_for_lm()
            #We want to do a
language model so we label
accordingly
             .databunch(bs=bs))
data_lm.save('tmp_lm')


# We have to use a special kind
of `TextDataBunch` for the
language model, that ignores the
labels (that's why we put 0
everywhere), will shuffle the
texts at each epoch before
concatenating them all together
(only for training, we don't
shuffle for the validation set)
and will send batches that read
that text in order with targets
that are the next word in the
sentence.
#
# The line before being a bit
long, we want to load quickly the
final ids by using the following
cell.

# In[ ]:
```

```
data_lm_small =
TextLMDataBunch.from_df(path,
lm_train, lm_valid,
```

```
data_lm =
TextLMDataBunch.load(path,
'tmp_lm', bs=bs)


# In[ ]:
```

__: Same Text __:Same Code

```
bs=16, text_cols=['text'],

max_vocab=60000, min_freq=2)
data_lm_small.save('data_lm_small
')
print(f"Language model vocab
size:
{len(data_lm_small.vocab.itos)}."
)
print("data_lm saved to: " +
str(path))
```

```
# We can then put this in a
learner object very easily with a
model loaded with the pretrained
weights. They'll be downloaded
the first time you'll execute the
following line.


# In[ ]:


lm_learner =
language_model_learner(data_lm_sm
all, AWD_LSTM, drop_mult=0.3)


# In[ ]:


lm_learner.lr_find()


# In[ ]:


lm_learner.recorder.plot(skip_end
=15)


# In[ ]:

lm_learner.fit_one_cycle(1, 1e-2,
moms=(0.8,0.7))


# In[ ]:


lm_learner.save('fit_head')
```

```
data_lm.show_batch()
```

```
# We can then put this in a
learner object very easily with a
model loaded with the pretrained
weights. They'll be downloaded
the first time you'll execute the
following line and stored in
`~/.fastai/models/` (or elsewhere
if you specified different paths
in your config file).


# In[ ]:


learn =
language_model_learner(data_lm,
pretrained_model=URLs.WT103_1,
drop_mult=0.3)


# In[ ]:


learn.lr_find()


# In[ ]:


learn.recorder.plot(skip_end=15)


# In[ ]:

learn.fit_one_cycle(1, 1e-2,
moms=(0.8,0.7))


# In[ ]:


learn.save('fit_head')
```

__: Same Text __:Same Code

```
# In[ ]:

lm_learner.load('fit_head');
```

```
# In[ ]:

learn.load('fit_head');
```

```
# To complete the fine-tuning, we can then unfeeze and launch a new training.

# In[ ]:

lm_learner.unfreeze()


# In[ ]:


lm_learner.fit_one_cycle(5, 1e-3, moms=(0.8,0.7))


# In[ ]:


lm_learner.save('fine_tuned')


# In[ ]:


lm_learner.load('fine_tuned');
```

```
# To complete the fine-tuning, we can then unfeeze and launch a new training.

# In[ ]:

learn.unfreeze()


# In[ ]:


learn.fit_one_cycle(10, 1e-3, moms=(0.8,0.7))


# In[ ]:


learn.save('fine_tuned')


# How good is our model? Well let's try to see what it predicts after a few given words.

# In[ ]:


learn.load('fine_tuned');


# In[ ]:

TEXT = "I liked this movie because"
N_WORDS = 40
N_SENTENCES = 2


# In[ ]:


print("\n".join(learn.predict(TEXT, N_WORDS, temperature=0.75) for _ in range(N_SENTENCES)))


# We have to save the model but also it's encoder, the part that's responsible for creating and updating the hidden state. For the next part, we don't care
```

__: Same Text __:Same Code

```
# In[ ]:

lm_learner.save_encoder('fine_tuned_enc')

# ## Classifier

# In[ ]:

from fastai import *
from fastai.text import *
from fastai.callbacks.tracker import EarlyStoppingCallback
from fastai.callbacks.tracker import SaveModelCallback
from fastai.callbacks.tracker import ReduceLROnPlateauCallback
import pandas as pd

# In[ ]:

path = Path("/content")
# In[ ]:


df = pd.read_csv('texts.csv')
df = df.dropna()
df.to_csv('texts.csv')
df.head()

# In[ ]:


for i in range(len(df['rm'])):
    cat = df['rm'][i]
    cat = cat.replace('[','').replace(']','').replace('\'','').split(', ')
    df['rm'][i] = cat


# In[ ]:


from sklearn.model_selection import train_test_split

train_df, test_df = train_test_split(df, test_size = 0.05, random_state = 7)
train_df, valid_df = train_test_split(train_df, test_size = 0.05, random_state = 7)
```

```
about the part that tries to guess the next word.
# In[ ]:

learn.save_encoder('fine_tuned_enc')

# ## Classifier

# Now, we'll create a new data object that only grabs the labelled data and keeps those labels. Again, this line takes a bit of time.

# In[ ]:


path = untar_data(URLs.IMDB)


# In[ ]:


data_clas = (TextList.from_folder(path, vocab=data_lm.vocab)
             #grab all the text files in path
             .split_by_folder(valid='test')
             #split by train and valid folder (that only keeps 'train' and 'test' so no need to filter)
             .label_from_folder(classes=['neg', 'pos'])
             #label them all with their folders
             .databunch(bs=bs))

data_clas.save('tmp_clas')
```

: Same Text :Same Code

```
# In[ ]:


data_clas =
TextClasDataBunch.from_df(path,
train_df, valid_df, test_df,
bs=16,

max_vocab = 60000, min_freq = 2,

vocab =
data_lm_small.train_ds.vocab,

text_cols = ['text'],

label_cols ='rm')
data_clas.save('data_clas.pkl')


# In[ ]:

data_clas = load_data(path,
'data_clas.pkl', bs=16)


# In[ ]:

data_clas.show_batch()


# We can then create a model to
classify those company
descriptions and load the encoder
we saved before.

# In[ ]:

learn =
text_classifier_learner(data_clas
, AWD_LSTM, drop_mult=0.5)
learn.load_encoder('fine_tuned_en
c')


# In[ ]:


learn.lr_find()


# In[ ]:


learn.recorder.plot()
```

```
# In[ ]:


data_clas =
TextClasDataBunch.load(path,
'tmp_clas', bs=bs)


# In[ ]:

data_clas.show_batch()


# We can then create a model to
classify those reviews and load
the encoder we saved before.

# In[ ]:

learn =
text_classifier_learner(data_clas
, drop_mult=0.5)
learn.load_encoder('fine_tuned_en
c')
learn.freeze()


# In[ ]:


learn.lr_find()


# In[ ]:


learn.recorder.plot()


# In[ ]:
```

: Same Text   :Same Code

```
# In[ ]:
learn.fit_one_cycle(1, 1e-1, moms=(0.8,0.7))

# In[ ]:

learn.save('first')

# In[ ]:

learn.load('first');

# In[ ]:

learn.freeze_to(-2)
learn.fit_one_cycle(1, slice(5e-2/(2.6**4),5e-2), moms=(0.8,0.7))

# In[ ]:

learn.save('second')

# In[ ]:

learn.load('second');

# In[ ]:

learn.freeze_to(-3)
learn.fit_one_cycle(1, slice(2.5e-2/(2.6**4),2.5e-2), moms=(0.8,0.7))

# In[ ]:

learn.save('third')

# In[ ]:

learn.load('third');

# In[ ]:

learn.unfreeze()
```

```
# In[ ]:
learn.fit_one_cycle(1, 2e-2, moms=(0.8,0.7))

# In[ ]:

learn.save('first')

# In[ ]:

learn.load('first');

# In[ ]:

learn.freeze_to(-2)
learn.fit_one_cycle(1, slice(1e-2/(2.6**4),1e-2), moms=(0.8,0.7))

# In[ ]:

learn.save('second')

# In[ ]:

learn.load('second');

# In[ ]:

learn.freeze_to(-3)
learn.fit_one_cycle(1, slice(5e-3/(2.6**4),5e-3), moms=(0.8,0.7))

# In[ ]:

learn.save('third')

# In[ ]:

learn.load('third');
# In[ ]:

learn.unfreeze()
```

__: Same Text __:Same Code

```
learn.fit_one_cycle(2, slice(1e-2/(2.6**4),1e-2), moms=(0.8,0.7))


# In[ ]:


learn.save('final')


# ## Test Set Performance


# In[ ]:


preds, y = learn.get_preds()


# In[ ]:


preds


# In[ ]:


y


# In[ ]:
```

```
learn.fit_one_cycle(2, slice(1e-3/(2.6**4),1e-3), moms=(0.8,0.7))


# In[ ]:


learn.predict("I really loved that movie, it was awesome!")


# In[ ]:
```