Week 8 Review Session

February 24, 2023

1 Week 8 Review Session

TL;DR: Using pre-trained deep neural networks on the Kaggle Cats vs Dogs challenge.

1.1 Why Python, all of a sudden?

- `R` – Really good for statistics, inference (confidence intervals, hypothesis testing), linear regression, tree-based methods, biostats, data augmentation (to some extent)
- `Python` – Really good for deep learning, but some other functions may be limited (e.g. getting CIs from linear regressions requires `statsmodels`, which is poorly documented compared to `R`).
- `R` good for business intelligence (visualizations and explanations), `Python` good for training large scale models where you only care about performance. If you want to explain stuff, `R` is really useful, especially when it comes to econometrics or biostats
- We can keep on using `R` but …
- While `R` has `Keras` and `tensorflow` (popular deep learning libraries), it does not have `Pytorch`, another super popular DL framework. The amount of resources on `Keras` + `R` is really scarce and it would be hard to find resources after you finish this course
- The techniques discussed in the guest lectures are mostly implemented in Python
- Picking up Python is probably a good idea if you want to use DL for business purposes (more resources etc)

Debating between R and Python is choosing between forks and spoons. Both are good. Both are good at different stuff. It is probably a good idea to learn how to use both and then pick the best tool for the job.

1.2 Deep Learning in Python

- The research community really likes Pytorch nowadays (easier to implement complicated network structures)
- But we will stick with Keras and Tensorflow (easier to use off-the-shelf models from 2016 or so)
- There were some non-negligible API changes to Tensorflow and Keras due to introduction of TF2.0. Please be aware if you are trying to dig deeper on these topics (e.g. `tf.python.keras` vs `tf.keras`. They are not always the same!)

1.3 Why Use Pretrained Deep Neural Networks?

- Saves time, money, space (A LOT of time, A LOT of money, and A LOT of space)
  - In Feb 2019, OpenAI released GPT-2. Training cost ~50,000. Has 1.5 billion parameters => 11 - 12 Gigs of VRAM (GPU memory). Trained on 8 million webpages => 50 Gigs of storage for dataset at minimum (to make things faster you probably need ~80 Gigs of RAM to store the data. Reading data off disks can be a bottleneck and 80G RAM is reasonable at industrial scale). You can probably do this as an enthusiast (an instance capable of this costs ~12 per hour)
  - Everything was reasonable until the large models attacked...
  - GPT-3 has 175 BILLION parameters. Requires 800 Gigs of DISK SPACE just to store the model.
  - The largest LaMDA (backbone of Google's Bard) has 135 BILLION parameters and is trained on 1.56 TRILLION words.
  - If you are not Google/Microsoft/Meta, you probably don't have that amount of money for training.
  - You need to trian more than once too: remember tuning?
- In many applications, you probably don't need to do everything from scratch
  - Word2vec contains openly available vector representations of words => sentiment analysis should be easier
  - Networks trained on ImageNet should already be good at differentiating objects
  - Object detection models should already be good in potential applications
- The results are good-ish with minimum effort (which makes them amazing prototypes!)

1.4 Loading Libraries and Datasets

There are some API changes so the top-rated code on Kaggled does not work out of the box. But that's fine. We can still do this. 1. Download the images from Kaggle 2. Extract the zip and store it in the same folder/directory this .ipynb is in. 3. Read the file names: each file name has the format "{cat | dog}.{index}".jpg, which gives us the label. 4. df: a dataframe storing the sample information. First column: name of the img. Second column: cat or dog? 5. A simple train_test_split works for now

[1]:
```
from tensorflow import keras
from keras.preprocessing.image import ImageDataGenerator as IDG
import pandas as pd
from sklearn.model_selection import train_test_split
import os
# We are not going to bother with data augmentation (additional transformations␣
 ↪on the images)
img_gen = IDG()
DATA_DIR = 'train'
# Shamelessly inspired Kaggle because I am lazy
filenames = os.listdir(DATA_DIR)
categories = [f.split('.')[0] for f in filenames] # Python shorthand
df = pd.DataFrame({
    'filename': filenames,
    'category': categories
```

```
})

train_df, validate_df = train_test_split(df, test_size=0.20, random_state=42)
```

1.5 Data Generators

Instead of storing the data on memory, you can store them on the disk and only load the images when needed.

Kind of not really needed for modern hardware (ResNet50 was introduced in 2015) but could be good practice if you are dealing with large-scale data.

In practice, if dataset $<$ 60 Gigs, it is not a terrible idea to store them entirely in your memory to save disk I/O time. SSD is 10x - 100x slower than memory.

[2]:
```
train_data_gen = img_gen.flow_from_dataframe(
    train_df,
    DATA_DIR,
    x_col='filename',
    y_col='category',
    target_size=(224, 224), # shape required by resnet50
    class_mode='categorical',
    batch_size=200
)

val_data_gen = img_gen.flow_from_dataframe(
    validate_df,
    DATA_DIR,
    x_col='filename',
    y_col='category',
    target_size=(224, 224),
    class_mode='categorical',
    batch_size=200
)
```

```
Found 20000 validated image filenames belonging to 2 classes.
Found 5000 validated image filenames belonging to 2 classes.
```

1.6 ResNet50

Using pretrained weights for ResNet50.

- ResNet50: a revolutionary (really not exaggerating here) neural network structure proposed in 2015 that "solved" imagenet.
- ImageNet: one of the most famous benchmarks in deep learning. Cateogrizing images according to the objects in them. Kind of like training robots to do Captcha (really, really, really, really loosely speaking)
  - It is a biased dataset (ethics of AI). See here. The model does not perform well for all peoples, anywhere in the world.

- Requires input images to have shape 224x224. Outputs a 1000-dimensional vector (probability that the image belongs to each of the 1000 classes in ImageNet)

[3]:
```
from tensorflow.keras.applications.resnet50 import ResNet50

resnet50 = ResNet50(weights='imagenet')
train_features = resnet50.predict(train_data_gen)
val_features = resnet50.predict(val_data_gen)
```

```
100/100 [==============================] - 21s 188ms/step
25/25 [==============================] - 5s 196ms/step
```

We now have a 1000-dimensional feature vector for each observation! Let's do logistic regression!

Don't laugh. The final layer of neural networks (even the deepest ones) is usually essentially doing logistic regression (if you train them to do classification)!

[4]:
```
print(train_features.shape, val_features.shape)
```

```
(20000, 1000) (5000, 1000)
```

[5]:
```
from sklearn.linear_model import LogisticRegression as LR
lr = LR()
lr.fit(train_features, train_df['category'])
val_pred = lr.predict(val_features)
print(val_pred)
print("There are {} mistakes".format(sum(val_pred != validate_df['category'])))
```

```
['dog' 'cat' 'cat' … 'cat' 'dog' 'cat']
There are 2448 mistakes
```

Doesn't seem to work well :(The features are not scaled well.

Let's try some other options & standardize the data first

[6]:
```
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.neighbors import KNeighborsClassifier as KNC
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
train_features = scaler.fit_transform(train_features)
val_features = scaler.transform(val_features)

rfc = RFC(min_samples_split = 0.01)
knc = KNC()

lr.fit(train_features, train_df['category'])
rfc.fit(train_features, train_df['category'])
knc.fit(train_features, train_df['category'])

lr_val_pred = lr.predict(val_features)
```

```
rfc_val_pred = rfc.predict(val_features)
knc_val_pred = knc.predict(val_features)

print("LR made {} mistakes".format(sum(lr_val_pred != validate_df['category'])))
print("Random Forests made {} mistakes".format(sum(rfc_val_pred !=␣
 ↪validate_df['category'])))
print("5-NN made {} mistakes".format(sum(knc_val_pred !=␣
 ↪validate_df['category'])))
```

```
C:\Users\boxia\anaconda3\envs\gputensorflow\lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
LR made 2442 mistakes
Random Forests made 2444 mistakes
5-NN made 2490 mistakes
```

The approach unfortunately does not work well for the challenge. As future steps, you can look into fine-tuning your ResNet50 for the Dogs vs Cats Challenge.

For R user (esp for your final project), a useful approach for converting words into vectors is Word2Vec. Here is an excellent tutorial. Some packages use fancy pre-trained language models as backends.