Lercio

May 10, 2023

1 Introduction

Pick up one of the available implementations of the Char-RNN (e.g. implement1, implement2, implement3, implement4, etc.) and train it on the dataset which contains about 6500 headlines from the Lercio satirical newspage, scraped by Michele Cafagna, past student of the ISPR course. The dataset is contained in a CSV file, one line per headlines. Be aware that the dataset can be a bit noisy (some errors due to encoding conversions) so you might need some preprocessing in order to prepare it for the task. Also, I am afraid the dataset is in Italian only as this is the language of the newspage. Try experimenting with different configurations of the CHAR-RNN, varying the number of layers. Since the dataset is quite small, keep the number of hidden neurons contained otherwise the net will overfit. Use the trained model (the best or the worst, your choice) to generate new headlines.
The softmax has a temperature parameter T that you can use to control the randomness of the output distribution (i.e. output logits are divided by T). Experiment with different values of T and comment the results.

I used the pytorch implementation of a char-rnn https://github.com/spro/char-rnn.pytorch

The code for this notebook as well as the train an generate script and the models are available on my GitHub https://github.com/matteotolloso/ispr

2 Data preprocessing

The dataset is composed by 6494 headlines, the mean length is 70 characters and the maximum length is 168 characters. I decided to pad all the headlines to the same length (168) with the special character "~" and to add at the begining the special character "^". This is because this implementation only supports fixed length chunks during the training and in order to give to the model the possibility to learn the semantic rules of an italian phrase it should observe one complete phrase as training example.

[ ]:
```
#load the dateset
import pandas as pd

#load the dataset, no header
df = pd.read_csv('dataset/lercio_headlines.csv', header=None)

#record with the maximum number of characters
max = df[0].str.len().max()
print(max)
```

```
# pad the headlines with special characters to make them all the same length
df[0] = df[0].str.pad(max, side='right', fillchar='~')

# insert a special character at the beginning of each headline
df[0] = "^" + df[0]

# create a txt file with the headlines
with open('dataset/lercio_padded.txt', 'w') as f:
    for line in df[0]:
        f.write(line)
```

```
168
```

3 Training and validation setup

The original experiments with this model have been made on the Shaekespeare dataset that is approximately 3 times bigger than the lercio dataset (without padding), since the original experiments were performed with 2000 epoch, i used 6000 epoch of training as baseline.

The other parameters that I played with in the training phase are the hidden size of the gru units and the number of layer, in addition to the temperature in the generation phase.

Grid search parameters:

- Unit type : GRU
- Epochs : 6000
- Chunks length : 169
- Batch size : 100
- Learning rate : 0.01
- Shuffle : True
- Hidden size : 100, 200, 300, 400
- Layers : 1, 2, 3, 4

It's difficult to systematically asses the quality of the generated headlines, in particular the semantic accuracy. Concerning the correctness of the single words, I prepared a benchmark using a dataset found on Github https://github.com/napolux/paroleitaliane. The "parole.txt" file contains a list of words in italian (almost complete in my opinion, around 1 milion words), including: compound words, names, surnames, cities and locations, verbs, adjectives, adverbs, etc. The idea is to use that file to calculate the percetage of correct words generate by the model with different hidden size, number of layers and epochs.

4 Syntactic test

[ ]:
```
# open the file "parole.txt" and read it creating a dictionary

real_words = {}
with open('./dataset/parole.txt', 'r') as f:
    for p in f:
        real_words[p.strip().lower()] = True

len(real_words)
```

```
[ ]: 952734
```

The following function computes the number of correct words in a given number of generated headlines, each one of a given length.

```
[ ]: from generate import generate

def percetage_correct_words(real_words, model_path, temperature, num_titles,␣
 ↪len_titles):
    total_words = 0
    wrong_words = 0
    for i in range(num_titles):
        title = generate(model_path, temperature, len_titles)
        # remove the ^ character
        title = title[1:]
        # remove the ~ characters
        title = title.replace('~', '')
        # for each word in the title
        for word in title.split():
            # if the word is not in the dictionary
            if word.lower() not in real_words:
                wrong_words += 1
            total_words += 1

    return 1 - wrong_words/total_words
```

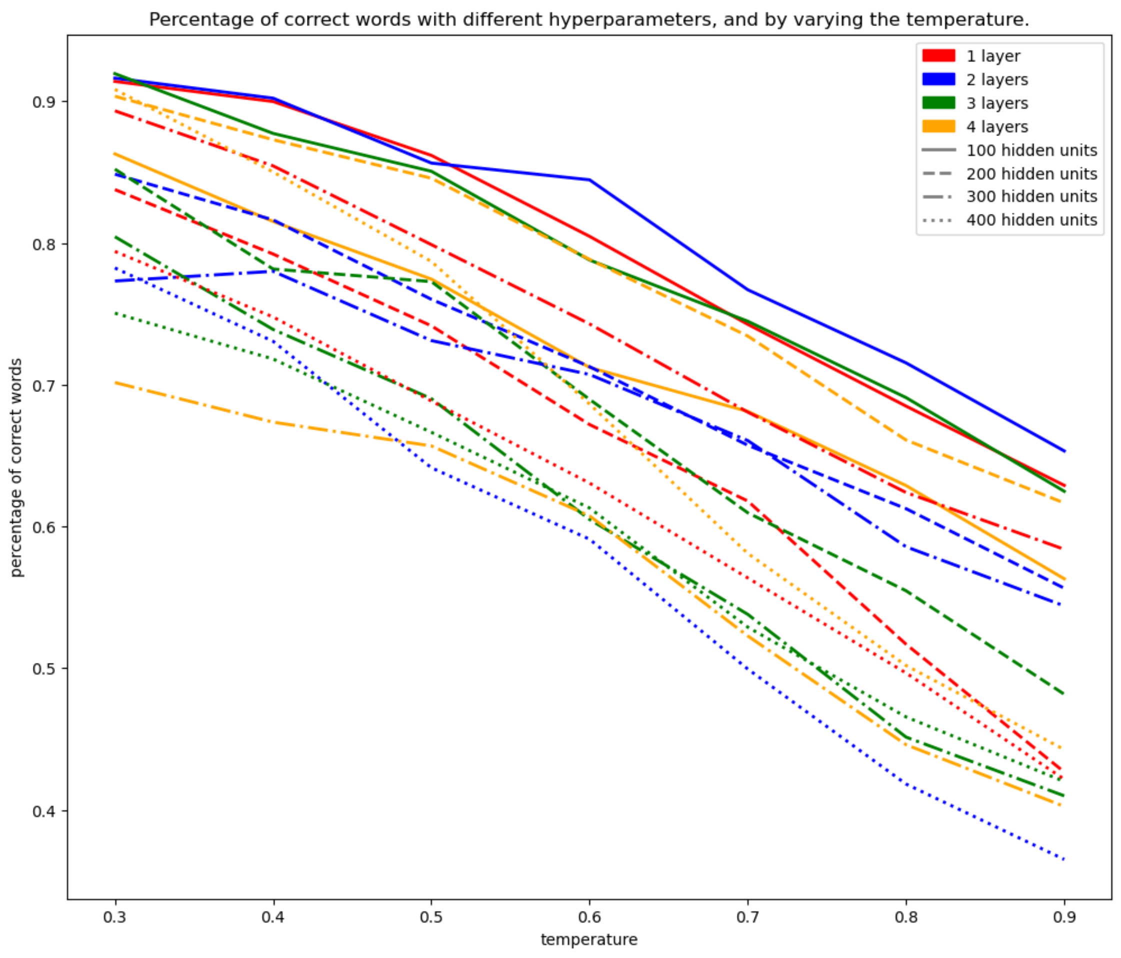
Now let's run some tests: for each hyperparameter configuration I tested different tempreatures in the generation phase and I computed the percentage of correct words in 100 headlines of 200 characters each.

```
[ ]: import numpy as np
import os

hidden_sizes = [100, 200, 300, 400]
layers = [1, 2, 3, 4]
temperatures = [0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

if (not os.path.exists('./results.npy')):
    results = np.ndarray((len(hidden_sizes), len(layers), len(temperatures)),␣
 ↪dtype=float)
    for k , hidden_size in enumerate(hidden_sizes):
        for i, layer in enumerate(layers):
            for j, temperature in enumerate(temperatures):
                results[k][i][j] = percetage_correct_words(
                    real_words=real_words,
                    model_path=f'./models/lercio_E6000_H{hidden_size}_L{layer}.
↪pt',
                    temperature=temperature,
                    num_titles=100,
                    len_titles=200
                )
    np.save('./results.npy', results)
```

Now i'm going to plot the results of the tests:

```
[ ]: import numpy as np
     import matplotlib.pyplot as plt
     temperatures = [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

     results = np.load('results.npy')

     # make the plot bigger
     plt.figure(figsize=(12, 10))

     plt.plot(results[0, 0, 1:], label='1 layer'  , linestyle="-", color='red',␣
      ↪linewidth=2)
     plt.plot(results[0, 1, 1:], label='2 layers' , linestyle="-", color='blue'  ,␣
      ↪linewidth=2)
     plt.plot(results[0, 2, 1:], label='3 layers' , linestyle="-", color='green',␣
      ↪linewidth=2)
     plt.plot(results[0, 3, 1:], label='4 layers' , linestyle="-", color='orange', ␣
      ↪linewidth=2)

     plt.plot(results[1, 0, 1:], linestyle="--", color='red',  linewidth=2)
     plt.plot(results[1, 1, 1:], linestyle="--", color='blue',  linewidth=2)
     plt.plot(results[1, 2, 1:], linestyle="--", color='green',  linewidth=2)
     plt.plot(results[1, 3, 1:], linestyle="--", color='orange',  linewidth=2)

     plt.plot(results[2, 0, 1:], linestyle="-.", color='red',  linewidth=2)
     plt.plot(results[2, 1, 1:], linestyle="-.", color='blue',  linewidth=2)
     plt.plot(results[2, 2, 1:], linestyle="-.", color='green',  linewidth=2)
     plt.plot(results[2, 3, 1:], linestyle="-.", color='orange',  linewidth=2)

     plt.plot(results[3, 0, 1:], linestyle=":", color='red',  linewidth=2)
     plt.plot(results[3, 1, 1:], linestyle=":", color='blue',  linewidth=2)
     plt.plot(results[3, 2, 1:], linestyle=":", color='green',  linewidth=2)
     plt.plot(results[3, 3, 1:], linestyle=":", color='orange',  linewidth=2)
     # personalizad legend
     from matplotlib.lines import Line2D
     from matplotlib.patches import Patch
```

```
custom_lines = [Patch( color="red", lw=1),
                Patch( color="blue", lw=1),
                Patch( color="green", lw=1),
                Patch( color="orange", lw=1),
                Line2D([0], [0], color="gray", lw=2, linestyle="-"),
                Line2D([0], [0], color="gray", lw=2, linestyle="--"),
                Line2D([0], [0], color="gray", lw=2, linestyle="-."),
                Line2D([0], [0], color="gray", lw=2, linestyle=":"),]

plt.legend(custom_lines, ['1 layer', '2 layers', '3 layers', '4 layers', '100␣
 ↪hidden units', '200 hidden units', '300 hidden units', '400 hidden units'])

plt.xlabel('temperature'),
plt.ylabel('percentage of correct words')
plt.xticks(np.arange(len(temperatures)), temperatures)
plt.title('''Percentage of correct words with different hyperparameters, and by␣
 ↪varying the temperature.''')
plt.show()
```



In this plot we can see the all the 16 models tested in a single picture. Combining different colors and line types, it's possible to compare all the hyperparameter combinations at different temperatures.

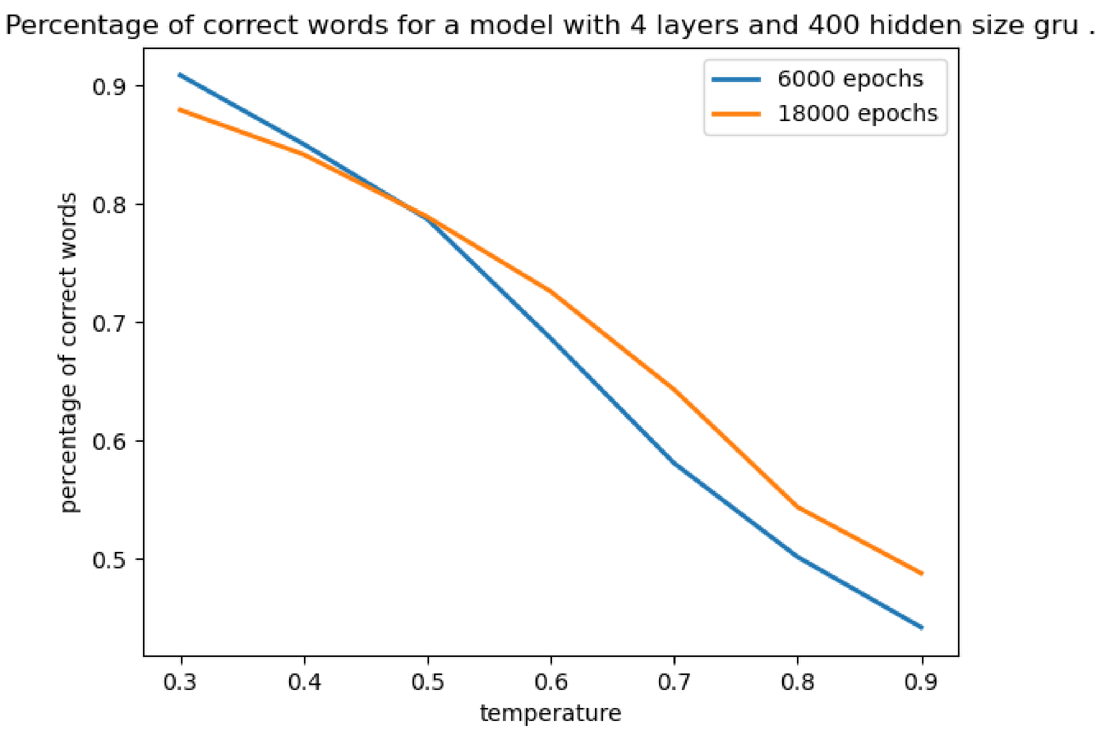
The two main conclusion we can draw are: - In all the models increasing the temperature decreases the percentage of correct words - The best models are the ones with less layers and/or less hidden size

There is a correlation between the complexity of the models and the number of wrong words generated, probably because complex models need more training data.

To test this hypothesis i tried to train the model with 4 layers and 400 gru hidden size for the triple of the epochs (18 000).

```
[ ]: big_results = np.ndarray(len(temperatures), dtype=float)

     for i, temperature in enumerate(temperatures):
         big_results[i] = percetage_correct_words(
             real_words=real_words,
             model_path=f'./models/lercio_E18000_H400_L4.pt',
             temperature=temperature,
             num_titles=100,
             len_titles=200
         )
```

```
[ ]: plt.plot(results[3, 3, 1:],  linewidth=2, label='6000 epochs')
     plt.plot(big_results,  linewidth=2, label='18000 epochs')
     plt.xlabel('temperature'),
     plt.ylabel('percentage of correct words')
     plt.xticks(np.arange(len(temperatures)), temperatures)
     plt.title('''Percentage of correct words for a model with 4 layers and 400␣
       ↪hidden size gru .''')
     plt.legend()
     plt.show()
```



The results are not as expected since a greater number of eopchs with a big model usually lead to overfitting the dataset, while in this case the percentage of correct italian words is comparable with the model trained for only 6000 epochs. Since i have not control on the regularization methods used by this implementation, i cannot make other conclusions.

5 Semantic tests

Some generated headlines of the best model based on the correct words metric, with different temperatures.

```
[ ]: print(generate(filename=f'./models/lercio_E6000_H100_L2.pt', temperature=0.2,␣
      ↪predict_len=200))
     print(generate(filename=f'./models/lercio_E6000_H100_L2.pt', temperature=0.3,␣
      ↪predict_len=200))
     print(generate(filename=f'./models/lercio_E6000_H100_L2.pt', temperature=0.4,␣
      ↪predict_len=200))
     print(generate(filename=f'./models/lercio_E6000_H100_L2.pt', temperature=0.5,␣
      ↪predict_len=200))
     print(generate(filename=f'./models/lercio_E6000_H100_L2.pt', temperature=0.6,␣
      ↪predict_len=200))
     print(generate(filename=f'./models/lercio_E6000_H100_L2.pt', temperature=0.7,␣
      ↪predict_len=200))
```

```python
print(generate(filename=f'./models/lercio_E6000_H100_L2.pt', temperature=0.8,␣
↪predict_len=200))
```

```
^Scoperto il primo contro le casa della sua bambina di Santare di San Salvini
alla moglie di contrario in un contratto di contratto il prossimo contro la
prossima di contrarmento~~~~~~~~~~~~~~~~~~~~~~~
^Sonda contro i finalmente contro la proprio contro la candidata su San Di Maio~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
^Si risparico se ne alle stato sulle per il cancro di piace di colpo per la
prossima di essere in casa di essere testa di appartante senza
possinda~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
^Mass in coscoprono sulla prima dei protestare in rifiultato di lavoro della
sindaco della compagne~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
^Scoperto con le consiglia in Garancia si fa topp sulla moglie il risvoltere a
al mondo di amici la conta di un castre di
miracori~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
^Femme in compagno compianto di sindaco il PD lo scop per una calcola di euro la
sciolta~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
^Matteo Renzi si al vincendo terrarcegreti con un paradini con il concerto~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Syntactically, we can confirm the results given by the test. The first two phrases have not wrong words while the other have always 1-2 wrong words. Semantically is difficult to find differences because all the headlines seems equally nosense. Anyway, with low temperature the models is able to better replicate series of words in a row from the dataset.

More in general, the words seem to be coherent with the previous one and the following one, but not with the ones at longer distances.

5.1 Prompt engineering

Let's now insert some information in the contex of the RNN, forcing the first words to be the same and then giving the model the "start sequence" token. The prompt is "Giorgia Meloni".

```python
[ ]: print(generate(filename=f'./models/lercio_E6000_H100_L2.pt', temperature=0.2,␣
      ↪predict_len=200, prompt='Giorgia Meloni ^'))
     print(generate(filename=f'./models/lercio_E6000_H100_L2.pt', temperature=0.3,␣
      ↪predict_len=200, prompt='Giorgia Meloni ^'))
     print(generate(filename=f'./models/lercio_E6000_H100_L2.pt', temperature=0.4,␣
      ↪predict_len=200, prompt='Giorgia Meloni ^'))
     print(generate(filename=f'./models/lercio_E6000_H100_L2.pt', temperature=0.5,␣
      ↪predict_len=200, prompt='Giorgia Meloni ^'))
     print(generate(filename=f'./models/lercio_E6000_H100_L2.pt', temperature=0.6,␣
      ↪predict_len=200, prompt='Giorgia Meloni ^'))
```

```python
print(generate(filename=f'./models/lercio_E6000_H100_L2.pt', temperature=0.7,␣
↪predict_len=200, prompt='Giorgia Meloni ^'))
print(generate(filename=f'./models/lercio_E6000_H100_L2.pt', temperature=0.8,␣
↪predict_len=200, prompt='Giorgia Meloni ^'))
```

```
Giorgia Meloni ^SI Stato contro la compagna della scuola~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Giorgia Meloni ^Manisti e in concertimoni di incinta la prossima di Santare di
Santare per la magliore~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Giorgia Meloni ^punti a Toninelli a parlanza in casso~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Giorgia Meloni ^vane scatta la moglie sulla persegra con le fine e il concorso
della Tossida~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Giorgia Meloni ^SI Maggia manda al PD~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Giorgia Meloni ^via Moramma di Mick Jappunta~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Giorgia Meloni ^Costituoboglio in galeanale~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^Med
```

The intresting result is that, as expected, some phrases have some terms related to the politics.