Università di Pisa - Master Big Data & Social Mining 2021

Deep Learning Project

Gianluca Guidi
Colab link

1. Project and dataset description

The dataset employed for this study is made of many artificial audio samples, where 30 different words are pronounced. These samples are in the form of a numpy array (i.e. (1187, 101, 40)) , and altogether sum up to 41849. Each file of the dataset is named after the word it represents. The objective of this project is performing a Speech Classification: the problem is approached by building deep learning models that take as input a sequence and return 30 different answers (probabilities of the input being one of the 30 words), and therefore we aim to develop a *multiclass classification*.

2. Preprocessing

In order to proceed with the construction of different deep learning models for the multiclass classification, a preprocessing is needed.

```
up : 2.84 %      [(1187, 101, 40),
right : 3.05 %    (1276, 101, 40),
one : 3.05 %      (1276, 101, 40),
down : 2.84 %     (1188, 101, 40),
four : 5.73 %     (2400, 101, 40),
no : 2.29 %       (957, 101, 40),
tree : 2.84 %     (1188, 101, 40),
happy : 3.54 %    (1481, 101, 40),
six : 3.55 %      (1485, 101, 40),
sheila : 3.5 %    (1463, 101, 40),
zero : 3.12 %     (1306, 101, 40),
off : 5.36 %      (2244, 101, 40),
marvel : 2.99 %   (1253, 101, 40),
seven : 3.37 %    (1411, 101, 40),
cat : 3.29 %      (1378, 101, 40),
eight : 2.66 %    (1113, 101, 40),
wow : 2.29 %      (957, 101, 40),
left : 3.55 %     (1485, 101, 40),
go : 2.29 %       (960, 101, 40),
yes : 2.97 %      (1244, 101, 40),
dog : 3.52 %      (1474, 101, 40),
bed : 3.24 %      (1356, 101, 40),
five : 2.61 %     (1092, 101, 40),
house : 5.69 %    (2382, 101, 40),
two : 2.16 %      (902, 101, 40),
three : 2.84 %    (1188, 101, 40),
bird : 3.22 %     (1346, 101, 40),
stop : 3.55 %     (1485, 101, 40),
nine : 2.73 %     (1144, 101, 40),
on : 5.32 %       (2228, 101, 40)]
```

To commence, being the 30 audio samples divided into 30 different files, they have to be concatenated. The 30 files representing the 30 words are shown on the table on the left, together with the number of samples each (both in numbers (shape) and in percentage on the total). As observed, words' audio samples cover from 2.16% to 5.69% of the total. After the concatenation of all these samples, we end up with a single numpy array of the shape (41849,101,40), that serves as our variable X. This numpy array, nevertheless, does not bear with it the labels of the single words (our variable Y). For this reason, a one-dimensional numpy array is created, with shape (41849,), and filled with integer values ranging from 0 to 29, to have a one to one correspondence (numbers are assigned to word's samples, see the figure on the right for the correspondences). We deal now with two numpy arrays, the first containing the audio samples (variable x), and the second our labels (variable y).

```
{0: 'off',
 1: 'five',
 2: 'zero',
 3: 'bed',
 4: 'dog',
 5: 'seven',
 6: 'happy',
 7: 'house',
 8: 'wow',
 9: 'go',
 10: 'tree',
 11: 'nine',
 12: 'stop',
 13: 'four',
 14: 'yes',
 15: 'on',
 16: 'six',
 17: 'three',
 18: 'cat',
 19: 'marvel',
 20: 'left',
 21: 'two',
 22: 'eight',
 23: 'no',
 24: 'up',
 25: 'one',
 26: 'right',
 27: 'bird',
 28: 'sheila',
 29: 'down'}
```

At this point, the dataset is divided into train and test sets with a proportion of 70%-30% respectively. For robustness purposes, the split is done in a stratified manner, in order to maintain the different labels' weights constant into the two new datasets (train and test). The dataset is also shuffled with the purpose of reducing variance and making sure that models remain general and overfit less. (I recall that the dataset was initially sorted by the labels). Subsequently, in order to proceed with a grid search for the model selection, a further split is done on the train set, dividing it into (x and y) train and (x and y) validation sets, with a proportion of 85%-15% respectively.
Finally, in order to ease and speed up both the training of our models and their convergence (that is, loss reaches local or global minima faster), our train sets are normalized to have mean 0 and standard deviation 1.
Our dataset is now ready and we proceed to deep learning models' building and selection.

3. Deep Learning Models

We first recall that, our task is to perform a *multiclass classification* and therefore, it is a classification task with 30 classes that assumes that each sample is assigned to one and only one label (mutually exclusive classes). For this reason, for all the models developed in the project, the output layer is activated through a Softmax function. The softmax activation function predicts only one class at a time, it in fact assigns a probability to each one of the 30 output neurons, with all probabilities summing up to 1. For what concerns the loss function, since we are forecasting probability distributions, it is used the cross entropy, for it measures the difference between probability distributions of a given set of events.
Three different models are hereafter developed: a feed-forward MultiLayer Perceptron, a Convolutional NN and a Recurrent NN.

3.1 Grid selection

For all of the developed models, a model selection is performed in order to choose for each model the hyperparameters that return the best results. The hyperparameters tested are:

- the learning rate (for all the models);
- the batch size (for all the models);
- hidden layer size (for the Multilayer Perceptron model);
- units size (for the Recurrent NN model);
- filter size (for the Convolutional NN 1D model).

3.2 Models and Results

3.2.1 Feedforward Models

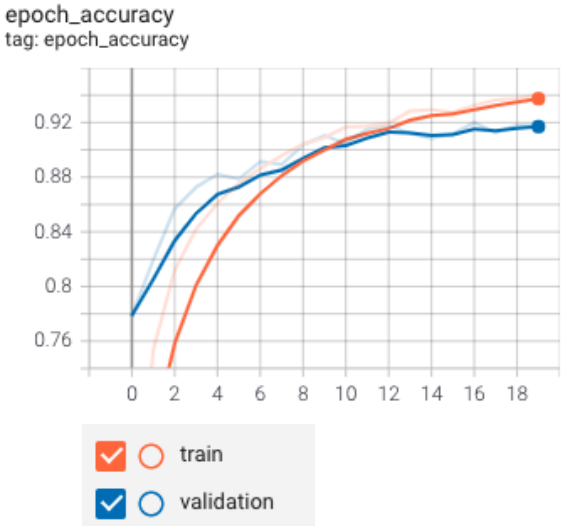
Deep learning literature suggests that sequential data, in addition to RNN and other models, can be also modeled with a dense MLP network. For this reason, two feedforward models, differing only on the optimization functions (SGD and ADAM), are developed. For both models, Sequential Keras API are a good fit for the job since sequential models are appropriate for models where each layer has exactly one input and one output tensor. In order to employ our dataset in these models, a reshape of the input data is necessary. It in fact is transformed from (101,40) to a linear shape of (101*40) for the input layer.
Two regularizations are also added to both models, the first is the dropout set to 0.2, in order to avoid and reduce the overfitting, the second is the early stopping, that allows the training to stop if its accuracy does not get better after 5 epochs (patience) in a row.
The loss function employed is the sparse categorical cross entropy since we have sparse labels, that is for each observation there is only one target class (0 to 29) and classes are mutually exclusive. Finally, since the task is a multiclass classification, the evaluation metrics utilized is the accuracy.
Both feedforward models have only one hidden layer, with the aim of being the simplest models possible, with a Relu activation function, to avoid the gradient vanishing issue.
The optimizer chosen differs, the first model employs the Adam while the second the SGD. Adam's weights update rule is invariant to the magnitude of the gradient, and this facilitates a lot when going through areas with small gradients. Unlike Adam, the SGD struggles to quickly move through these areas. Below, we observe the results of both best model configurations. For each model, the grid search tested 12 different configurations of hyperparameters:

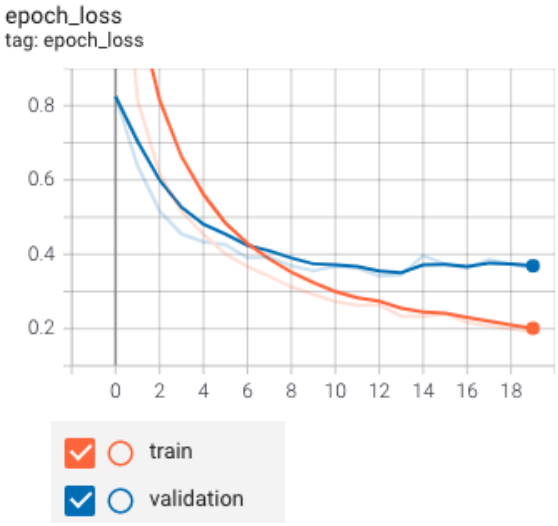
- Learning rate: [1e-2, 1e-3, 1e-4],
- Hidden sizes : [32, 64]
- Batch sizes : [16,32]

The best configuration for the Relu+Adam model is:

- Hidden size: 64,
- Learning rate: 0.001
- Batch size: 32

This configuration has an accuracy of 0.94 on the train set, and 0.91 on the validation set (graphs below).

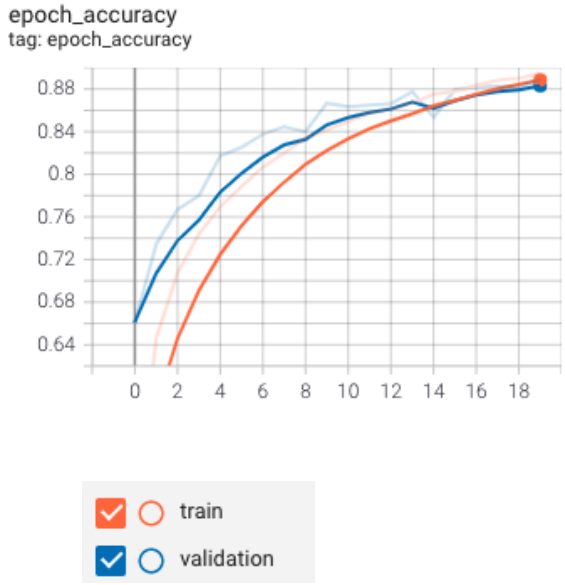


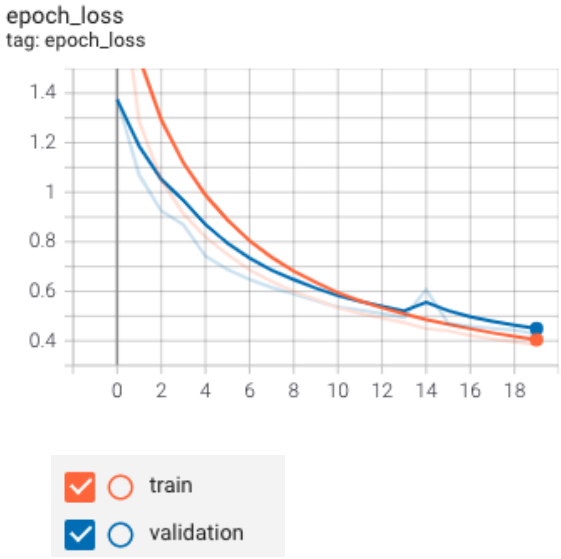


The model sees its validation set (in blue) performing better than the train set for almost half of the epochs tested. The reason behind this behaviour can lie on the fact that a number of neurons is randomly disabled (20% in this case, due to the dropout) during the training, while all of them are used during the validation, and so the train set has a lower number of neurons to use.
Finally, the evaluation of the model on the test set portrays a loss and accuracy of respectively **[0.339, 0.929].**
The best configuration for the Relu+SGD model is instead:

- hidden size: 64
- learning rate: 0.01
- batch size: 32

This configuration has an accuracy of 0.90 for the train , and 0.89 for the validation. This slightly lower performance can be justified with the premise given above.
The same behaviour as before can be observed in the graphs below.





Finally, the evaluation of the model on the test set portrays a loss and accuracy of respectively **[0.437, 0.888].**

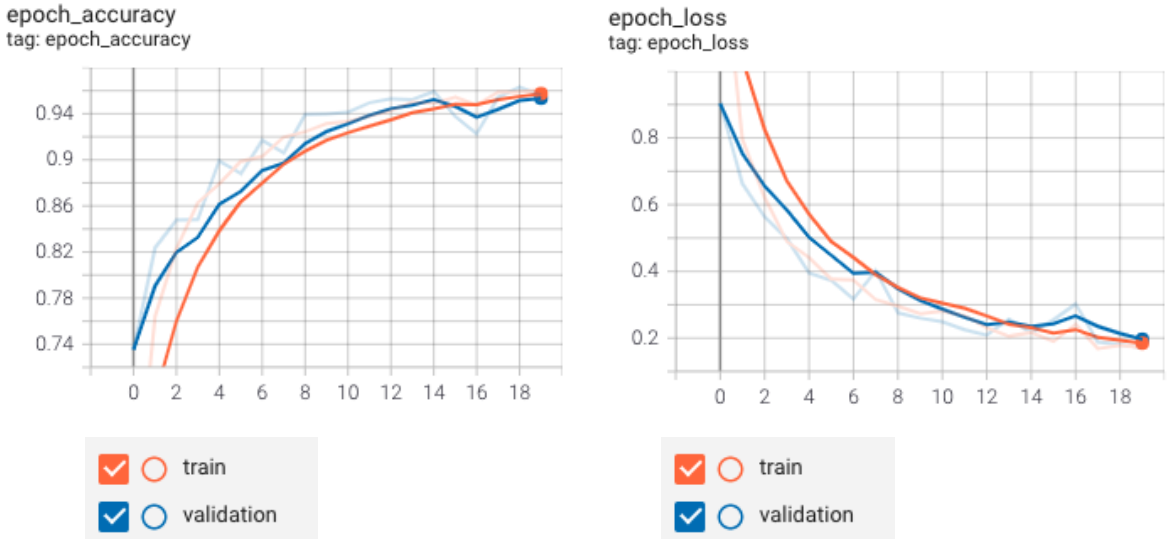
3.2.2 Convolutional Neural Network

Convolutional neural network of 1 dimension is another model that we expect to work well with long sequences such as audio samples (our case) or texts. From the literature, we know that there are two main properties of CNN that could improve speech recognition performance. The first one is that pooling at a local frequency region makes CNN more robust to little shifts that may arise with different speakers or speaking style variation (although our data is generated by an artificial algorithm). Secondly, sparse local connections of the convolutional layer need far less parameters in order to extract features at low level, and this helps to avoid overfitting. For the CNN 1D model, the grid search tested 12 different configurations of hyperparameters:

- Learning_rate: [1e-2, 1e-3, 1e-4],
- Filters: [32, 64]
- Batch sizes : [16,32]

The model that better performed is a CNN network with a conv1D layer with 64 filters and kernel size equal to 2, learning rate equal to 0.01, a Relu activation function and a global average pooling layer. While it is possible to add multiple blocks of convolutional and pooling layers, our experiments found that additional convolutional blocks do not result in further improvement. Therefore the best architecture adopts just one convolutional layer followed by one max-pooling layer and then the output layer.

Below in the graph it is shown accuracy and loss of such a model, of both train (in orange) and validation set (in blue). It emerges how their two performances almost coincide, with accuracies that nearly touch 96%.



Finally, the evaluation of the model on the test set portrays a loss and accuracy of respectively **[0.175, 0.957]**

3.2.3 Recurrent Neural Network

Finally, since RNN are adapt for tasks that involve sequential or temporal data, (such as speech recognition, automatic translation etc.), it is suitable to conclude our task with this model, that we expect to bring out notable results. Since RNN networks hold a vector of activation for each temporal step, they are extremely deep and hard to train due to the vanishing or exploding gradient issue.

When it came to choose between a RNN GRU or a RNN LSTM, the choice sticked to the first for three reasons: first, in order to tackle issues of memory in the short term, second, for its computational efficiency and less

complexity, and third, because we are dealing with samples and LSTM are to be preferred when sequences' length is greater and when there are much more data available.

The first issue to work on was related to the time consumed by the training of the model, as the number of units chosen was increased. Therefore, the number of units of the first layer is set equal to, or slightly higher than, the number of features of the numpy array vector.

Concerning the learning rate: if it is set too low, the parameters seem to be locked in a local optimum until a certain time when they were able to unlock, and the curve begins to grow gradually. Conversely, with a too high learning rate, the network seems to assume an unstable behavior. For this reason, after a series of experiments with "standard" learning rate (0.01, 0.001, 0.001), learning rate values between 0.003 and 0.007 are tested to obtain the most stable network possible without having to dramatically increase the epochs of training.

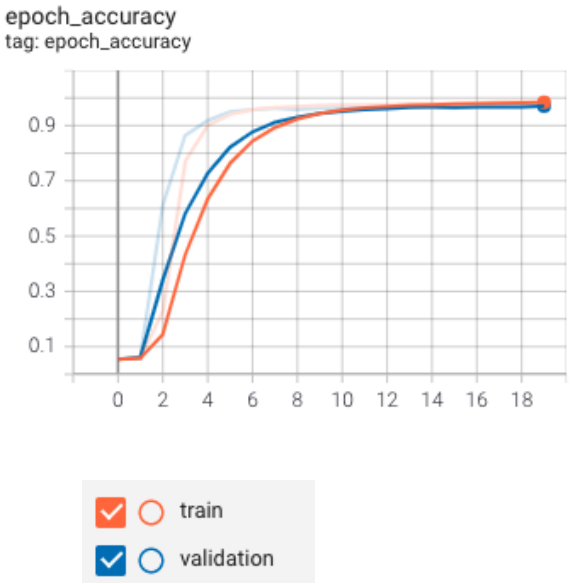
As for the batch size, sticking to small batches seems a good way to proceed, as Master and Luschi suggests in a paper (https://homl.info/smallbatch), according to which it is preferable to use small batches (from 2 to 32) as they tend to build better models in less training time.

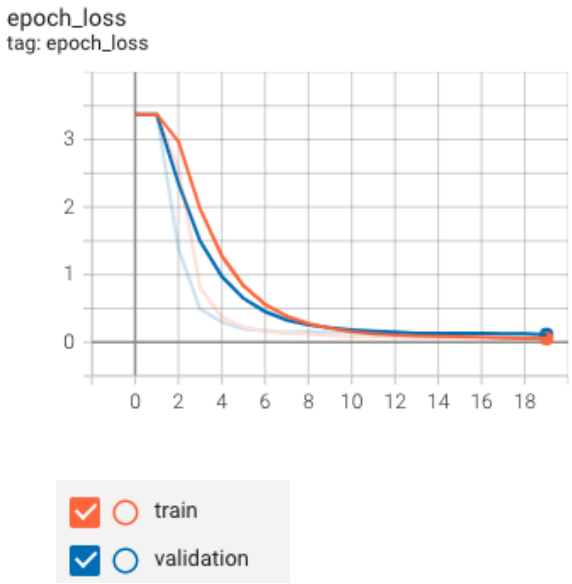
Our final model therefore tried to balance both the number of hyperparameters, the epochs required for training and the learning rate to obtain a model that was efficient without falling into overfitting.

Through the grid search we tested 12 different configurations of hyperparameters:

- Learning rate: [3e-3, 5e-3, 7e-3],
- Units: [16, 32]
- Batch sizes : [16,32]

The best model ended up being with two GRU layers of 48 (kept fixed) and 16 (chosen by the grid) units respectively and a learning rate of 0.003 with Adam optimizer. Below the graph portrays accuracy and loss of train (in orange) and validation set (in blue). Accuracy reaches 98% for the train and 96% for the validation.

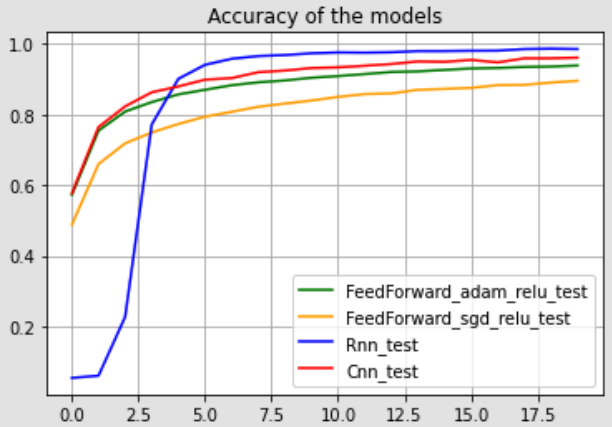




Finally, the evaluation of the model on the test set portrays a loss and accuracy of respectively **[0.104, 0.973],** setting this as the best model out of the four tested.

4. Conclusion and further research research speculation

To conclude, the graph below aims to compare the four models developed. It in fact plots the accuracy that the models reached once applied on the test set, through the 20 epochs.



As previously asserted, the model that showed to be the most accurate is the RNN- indeed, with the premises we made above each model, this result could be expected. In order to further develop this winning model, we may opt for a different and wider test and configurations of the hyperparameters, and by adding different layers, with the risk of making the model too complex and falling into overfitting. Nevertheless, the present research primarily aimed at testing how different deep learning models behave with speech classification, and a 97% accuracy on the test set was a satisfying result for the project aims and thus, further implementations may arise in a different research, primarily focused on the RNN network and on maximizing the accuracy.