ASSIGNMENT 1 (Pos Tagging)

Davide Mercanti (davide.mercanti@studio.unibo.it)
Riccardo Fava (riccardo.fava6@studio.unibo.it)
Luca Bompani (luca.bompani4@studio.unibo.it)

0.1 Abstract

In this work we have tried four different models to address the "Part-of-Speech (POS) tagging" task. Starting from a baseline we have applied different strategies to improve the performances of our architectures. To determine the best models we have tried to vary the different hyperparameters of models so that, through a comparison on the F1-metric, we have been able to select the two most promising models: one based on a bidirectional GRU layer, the other on a two-layers bidirectional LSTM. On them we have performed the evaluation on the test set and the error analysis.

0.2 General setting and data handling

We decided to rely on the Pytorch framework for the implementation of the neural network and to use the Pandas library for better handling of the data. The dataset is read into a pandas Dataframe and encoded using pre-trained Glove embedding. For handling the out of vocabulary terms (later, OOV) we decided to create a different random embedding for each of these words. We have also mapped all the words in the dataset to lowercase format before embedding them to avoid the majority of the OOV words. To emulate a real-world scenario, we have computed the embeddings and added them to the vocabulary at different time steps (train/validation and test). The labels are one hot encoded; this is done to avoid the superimposition of a fake order of our labels and to ease the working of the softmax layer.

0.3 More about the models used

We tried four different architectures: the first one, which we considered as our baseline, uses a Bidirectional-LSTM with a Dense layer with a softmax as output, the second one substitutes the LSTM with a Bidirectional-GRU. The last two models are a variation of our baseline doubling the Bidirectional-LSTM in the first case and the Dense layer in the other. All models have an embedding layer to convert the word data to a vector format, set to not be trainable.

0.4 Experiments

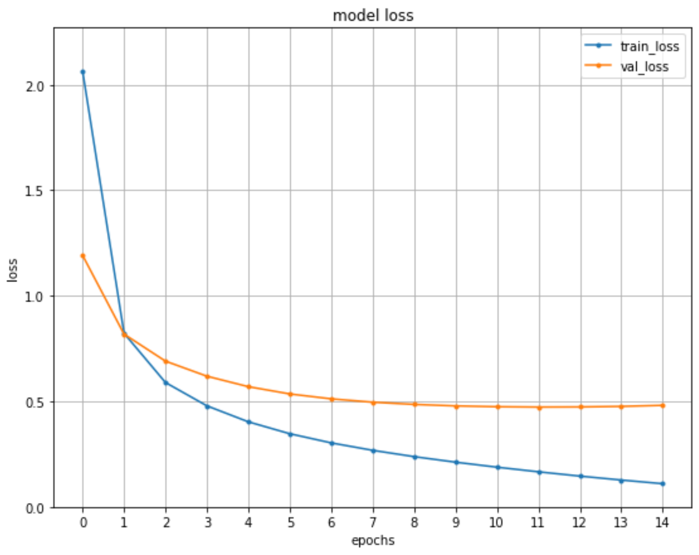
To optimize our solutions we have made different trials varying the hyperparameters obtained by tuning the different networks on their F1-score. The hyperparameters used are the Optimizer function, the size of the hidden layers of the networks and the learning rate. To tune our models we have first made a coarse optimization with large differences between the different learning rates. After finding a promising value we have fine-tuned it by varying it with smaller steps around the identified value. A similar strategy has been adopted to decide the hidden layer size while for the Optimizer we have realized that the ADAM optimizer always outperforms the SGD one.
Another test that we performed is to avoid the lowering in the dataset by treating words with different formats, with respect to the vocabulary, as OOV. This introduction allowed the network to better distinguish between common nouns and proper names but reduced the performances; so it was discarded. We have also noticed that the dataset is greatly unbalanced. To balance the network response we have decided to add weights to the loss; this has been shown to give a boost to the F1-score at the cost of reducing the accuracy. To increase the performances of the model we have added a scheduler for the learning rate. With this addition, we noticed that the model did not remain stuck in an oscillatory behaviour after some epochs of training. Last, to avoid overfitting, we have also introduced an early stopping procedure based on the loss of the validation set. For brevity, we present in the following table the effects of the various modifications on an exemplificative case, the Baseline architecture.

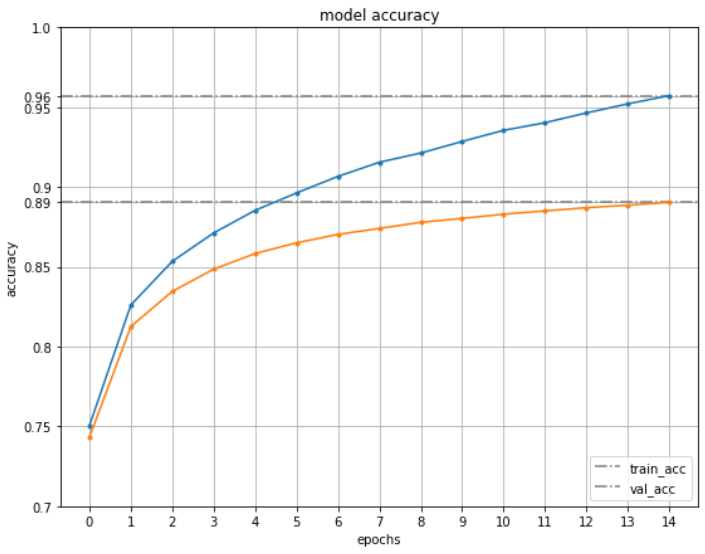
	Accuracy	F1-Score
Baseline	88.2%	0.708
Baseline + lowering	88.1%	0.714
Baseline + weights	85.6%	0.693
Baseline + lowering + weights	86.68%	0.730

0.5 Results

The following results have been obtained by training our neural networks for twenty epochs (or less, in the case of early stopping) on the best configuration of the hyperparameters on the **validation set**:

	Accuracy	F1-Score
Baseline	86.68%	0.73
Bi-Gru + FC	89.32%	0.77
Bi-LSTMx2 + FC	85.6%	0.76
Bi-LSTM + FCx2	86.57%	0.73





*Training and validation metrics for **Bi-Gru + FC**.*

We finally tried the models on the **test set** (we report here the two best ones):

	Accuracy	F1-Score
Bi-Gru + FC	90.13%	0.81
Bi-LSTMx2 + FC	89.27%	0.81

0.6 Error analysis

In order to analyze the prediction errors that our best models made, we firstly generated *classification reports* (in the scikit-learn format) and confusion matrices. From that we deduced that the 2 most common errors (for both models) are the misclassification between these classes: 1) *cardinal number* with *list item marker* and 2) *adverb* with *preposition or subordinating conjunction*. Other common errors are the wrong classification of adjectives with respect to adverbs and of singulars with respect to plurals.

For what concerns the classes that have low values in terms of f1-score and accuracy, we always observed a support in the training set ranging from extremely low to low:

Class	RBR	NNPS	PDT	LS
Support in the training set	86	95	9	10

Class: RBR=Comparative adverb, NNPS=Plural proper noun, PDT=Predeterminer, LS=List item marker

Nonetheless, given that a low support doesn't necessarly reflects into low metrics, we cannot exclude that our model's ability to learn from data could be better balanced among classes.

0.7 General conclusions

As mentioned, the architectures that have reached the best performances in terms of F1-score and accuracy are: the **Bi-Gru + FC** model and the **Bi-LSTMx2 + FC** model.
For what concerns the Bi-LSTMx2 + FC model, we can hypothesize that, due to the increased number of LSTM layers, the network is able to better extract contextual information from the sentence.
Instead, we can hypothesize that the Bi-Gru + FC model works slightly better with respect to the Bi-LSTMx2 + FC because of the restricted dimension of the dataset.

Possible **extensions** of this work could be:
1) The addition of learneable embeddings, in order to be able to fine-tune them to our domain.
2) Feeding the network with handcrafted features that we know are useful to better perform our task.
3) Trying with a broader spectrum of architectures and with variants of the ones we used. For example, it would be worth investigating the substitution of the dense layer(s) with time-distributed dense layer(s).