Sebastián Arbeláez Castaño

arbelaezsebastian16@gmail.com

Fake news classifier report

This project was based on an online example[1] of fake news classifier using PassiveAggressiveClassifier from sklearn library. Instead of using the sklearn classifier, a feed-forward neural networks was implemented to compare if it's possible obtain a higher accuracy value and to practice more in the use of neural networks. The increase was just of 0.7% from 92.7% accuracy to 93.4% but still it gives some insight of the adaptability of neural networks as a flexible tool for machine learning.

Index

Database structure....2

Database handling....2

Model architecture....3

Model tuning: 3

Results: 3

Hidden size: 5

Number of hidden layers: 5

Learning rate:....5

Number of epochs:....6

Mixed analysis:....6

Hidden size vs number hidden layers: 6

Number of hidden layers vs number of epochs: 6

Hidden size vs number of epochs: 7

Conclusion:....7

[1] https://data-flair.training/blogs/advanced-python-project-detecting-fake-news/

Database structure

The database that was used in this project is loaded from a csv which contained 6335 entries each with the following information:

- Identifier: a number to identify each entry – integer format.
- Title: the title of the news - string format.
- Text: the text of the news - string format.
- Label: the label of the news, only can be REAL or FAKE – string format.

Database handling

Since this database is already cleaned up there was no need to implement any test for outlier or other types of data cleansing. The sklearn PassiveAggressiveClassifier could get labels as an input for the prediction parameter, but the neural network works with number of classes to use the cross-entropy loss function. To do this a numpy array of values was created according to the following function:

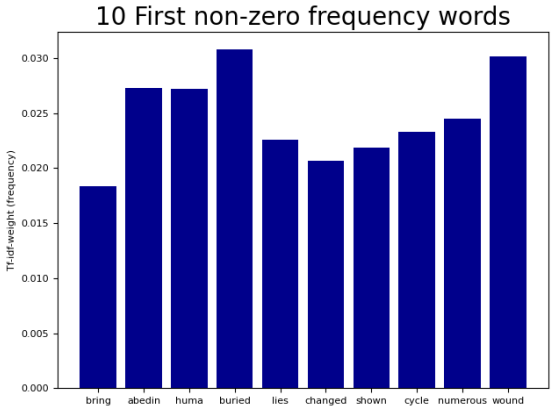
$$f(x) = \begin{cases} 1 \; if \; x = REAL \\ 0 \; if \; x = FAKE \end{cases}$$

Besides from the past transformation a tool from the online example was used that turns text into a vector of the size of the vocabulary in which each position of the array corresponds to one word. Each value corresponds to the relative frequency the word appears in the text. The tool used is the TfidfVectorizer object from sklearn library. An example is:

“Daniel Greenfield, a Shillman Journalism Fellow at the Freedom Center, is a New York writer focusing on radical Islam.

In the final……it lies buried in her emails with Huma Abedin. And it can bring her down like nothing else has” (1296 words)

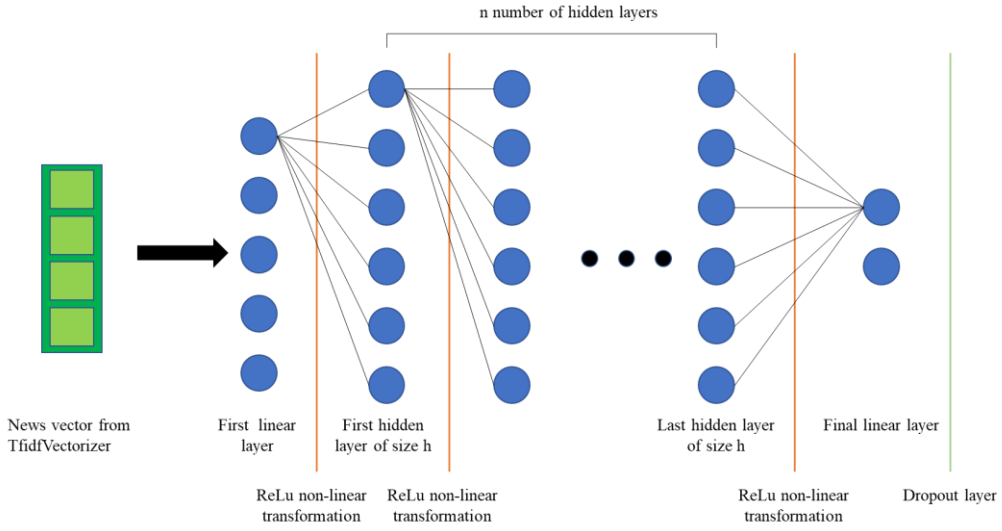
Here is an example of the first 10 nonzero values in the vector corresponding to the fragment of text:



This data is the one processed by the model.

Model architecture

The model is a feed-forward neural network with hidden size h, number of hidden layers n, input size of the first linear layer of vocabulary size and output size of a 2 since the news can only be REAL or FAKE. Between each layer there is a ReLu nonlinear transformation and after the last linear layer there is a dropout layer to help in the training step with a 0.5 dropout probability. The last linear layer works with as a SoftMax function in which the highest value of the two neurons classifies the news.



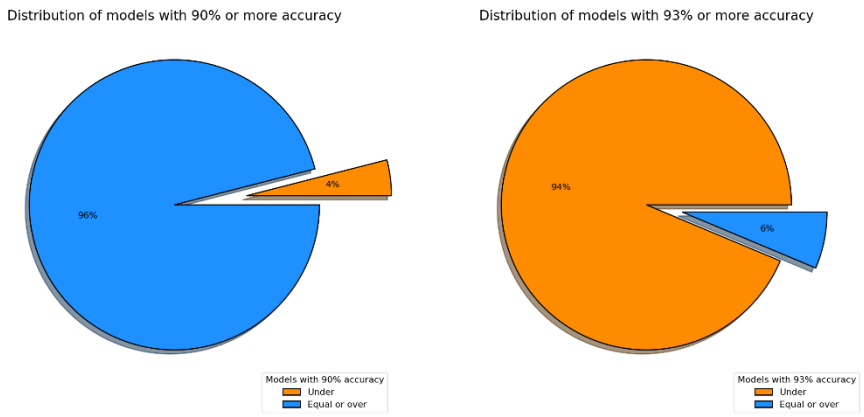
Model tuning:

To find out which values of hidden size (h), number of hidden layers (n) among other hyperparameters for the training phase should be used 300 different models where tried. Each testing different values of the hyperparameters. For the training step the data was divided randomly as 80% train, 20% test. The tuned hyperparameters were the following:

- Hidden size (h): size of the output of the first layer, input and output of the n hidden layers and the input of the final linear layer. The values were chosen randomly with a uniform distribution between 5 and 50.
- Number of hidden layers (n): number of hidden layers apart from the first and the last linear layers, these layers all have input and output size of hidden size. The values were chosen randomly with a multinomial up to 10 hidden layers.
- Number of epochs (num_epochs): number of times the data will be passed through the network during the training phase. The values were chosen randomly with a uniform distribution between 10 and 50 epochs.
- Learning rate (lr): learning rate of the optimizer. The optimizer being used is the adam optimizer, that gets the model parameters and applies stochastic gradient descent with the step size of lr accordingly to the loss function that is minimizing. In this case the loss is the cross-entropy loss from Pytorch. The values were chosen randomly with a uniform distribution between 0.0001 and 0.01.

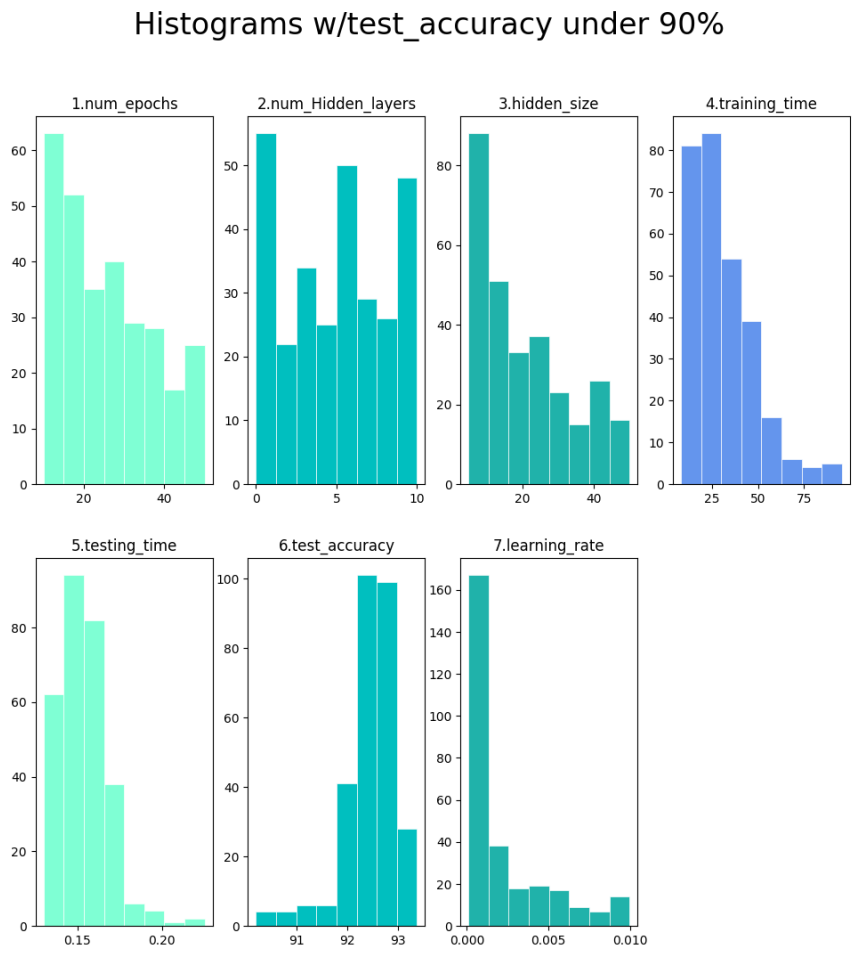
Results:

After 300 models trained, the data collected is the following:



96% of models trained got an accuracy of over 90% in the test set but only 6% got 93% of accuracy or better.

The values of the models which got 90% of better accuracy are the following:

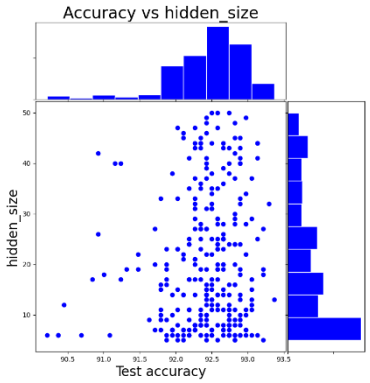


As we can see most of the models have a num_epochs of 0 to 20, even though there are others with more. This is due to an extensive number of epochs could get the model to overfit to the training set. Other interesting value is the

learning rate as most of the values range from 0.0001 to 0.00125, this is due to a big learning rate could lead to wrong stochastic gradient descend step, pushing the model off in the loss function local maximum the optimizer was trying to go. The hidden size has most of its values from 0 to 20, showing that the model doesn't need extensive bigger sizes to incorporate the information. Finally, the test accuracy is mostly between 92% and 93%, close to the sklearn tool at 92.7%.

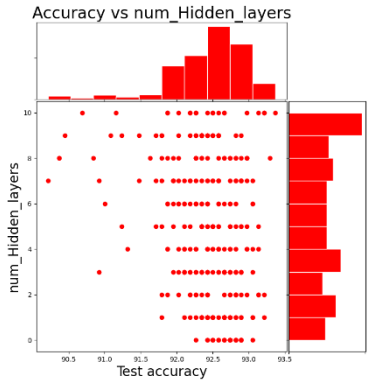
Following this line of thought on different variables against the accuracy we got the following graphs:

Hidden size:



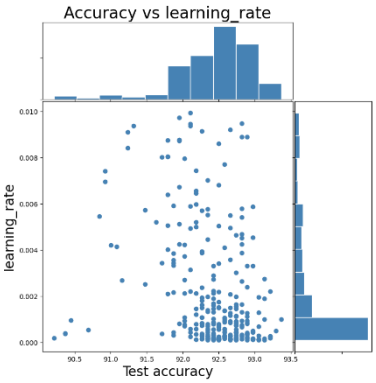
Comparing again the hidden size (number of neurons in the hidden layers) against the test accuracy we got these interesting results in which we can see most of the models with a relatively small hidden size between 5 and 10. Even though there are other factors among the number of hidden layers and the learning rate, both which have a strong influence in the model accuracy. There are still a lot of good models with a bigger hidden size but still a small size is easier to train, and bigger models tend to overfit more often. There will be further analysis comparing hidden size and other variables against its accuracy.

Number of hidden layers:

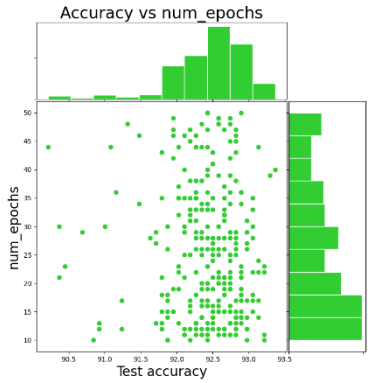


The number of hidden layers helps the model capture more abstract concepts as it's a parameter that affects the depth of the model. The more complex the model, the more complex the training. Comparing the results of the accuracy vs the number of hidden layers there is a variety of very good results in all different values. There is as big concentration of models from 8 to 10 number of hidden layers but there is also bigger variance since the more complex the harder to correctly tune it.

Learning rate:



The learning rate is the step size of change in the model parameters made by the optimizer. It can't be too big since the optimizer wound be able to follow correctly the stochastic gradient descent and it can't be too small since the model will take forever to converge to the optimum. It can be seen accurately in this case that most of the good models have a learning rate value between 0.0001 and 0.001. This shows us that even though there are good models with a bigger learning rate, the probability of achieving that level of accuracy decreases the bigger it gets.

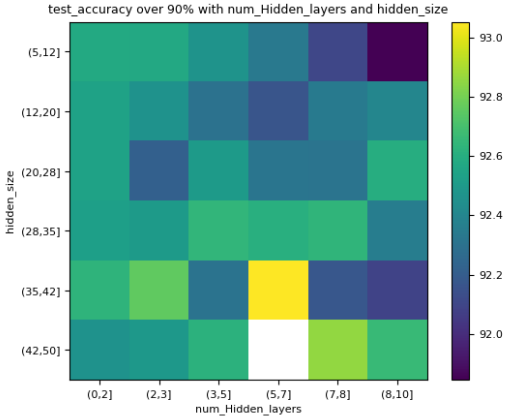


Number of epochs:

Number of times the data is passed through the network in the training step. A lower value might not be enough to train the network and an excessive number of times might make the model to overfit to the training data, creating a bigger error in the test set. Most of the values are within a small number of epochs (from 10 to 20). Even though there are models with very good performance from 35 to 45. The ideal number of epochs will be directly correlated to the model complexity, the bigger the number of parameters and the depth of the network the more epochs will be necessary for the model to converge to the optimal loss.

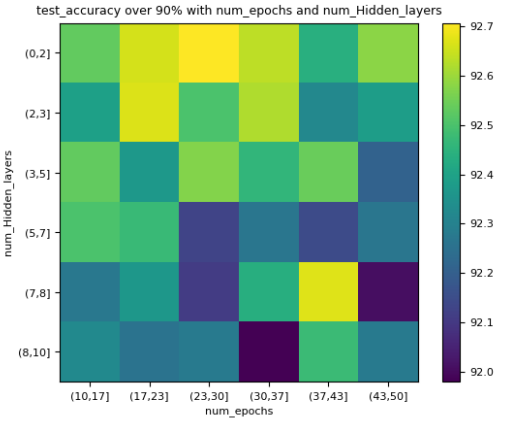
Mixed analysis:

Hidden size vs number hidden layers:



There are more yellow values with a bigger hidden size. This is seen mostly due to a more complex model can incorporate more information. Even though in the number of hidden layers the ideal values range from 5 to 7. Models with higher number of hidden layers decreases in accuracy due to overfitting.

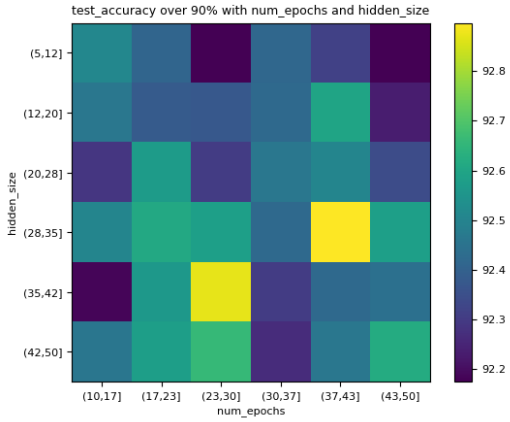
Number of hidden layers vs number of epochs:



It can be observer how models with lower number of hidden layers require a smaller number of epochs compared to bigger models requiring more epochs. This is due to the number of parameters, the more parameters needed to be calibrated the more passes are required.

Hidden size vs number of epochs:

Similarly, to the past section the more complex the model is the more passes of the data is needed to correctly calibrate the model. This is interesting since there are just 2 points of good accuracy. Even thought the less complex model requires less epochs this is not considering the number of hidden layers, a variable also directly impacting the number of epochs required.



Conclusion:

The ideal hyperparameters in this case is the model with 10 hidden layers and a hidden size of 13. To train this network was needed 40 epochs, enough times considering the number of parameters. This network has a significant depth but a small hidden size, making it the correct amount of complexity the model needs to predict the label of the news. Once the network is already trained the time required to estimate a value is small, giving it a good chance to compete against other methods like the previously mentioned sklearn library. Even though the model did better than the other library there are possibilities to improve. These improvements are mostly in the ranges of the tried hyperparameters like the learning rate which had values that didn't help at all to train the model or the loss function which can be trained to punish harder incorrect classified fake news. Telling that this neural network could be adapted to many different scenarios it gives insight in the strength of neural networks as classifiers and the adaptability it has to face a variety of challenges.

Bibliography

- https://data-flair.training/blogs/advanced-python-project-detecting-fake-news/
- Pytorch, sklearn among various other python libraries.
- Matplotlib gallery.