Adversarial POMCP

Alessandro Rodeghero
Lorenzo Tabarelli

Performance analysis and Plots

Introduction

In this report we're going to show the software performance over different starting configurations. The initial state of the environment can be set changing 7 parameters explained in the next chapter. The first set of tests has been executed using the default Neural Network (which aims to emulate the black-box model). At the end we propose a different NN model that guarantees better emulating performances.

Parameters explanation

The software performance and evolution depends strongly on its starting state. Here's the explanation of the configuration parameters:

TRAIN_SIZE	Initial number of points known, the base of the synthetic generated points.
NON	Number of nodes of each NN's hidden layer
NOL	NN's number of layers
NOE	Number of train epochs
LAMBDA	Length of shift step of a starting point to generate another one
MAX_RHO	Number of iteration, times we generate new points
K	Number of points to be shifted for each iteration

Neural Network

The neural network is used to emulate the black-box behavior. In the first iteration it's trained on the starting dataset. Following its gradient we generate synthetic points. After this generation process the augmented dataset is now composed by *starting_points+generated_points* and the NN is trained again on this new dataset. The expectation is that the accuracy over the test set does not get worse. In the best case scenario it should increase.
The test set has 300 labeled points and it is the same for every augmented train set.

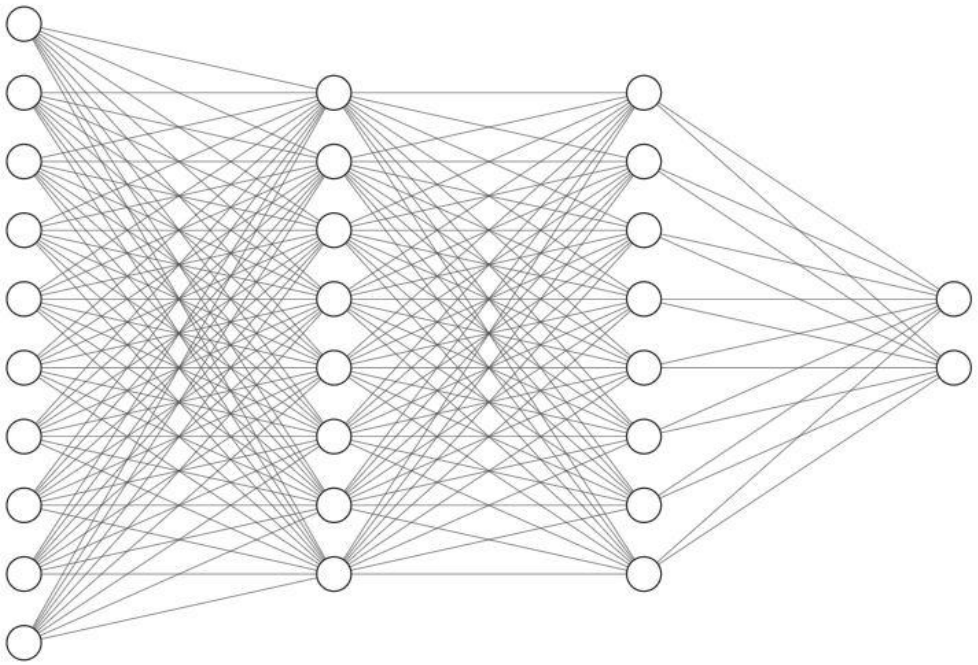


Fig.1: NN architecture with 81 input nodes, 2 hidden layers with 8 nodes each and 2 output nodes.

Test 1: Changing initial train size

The first result we propose is the analysis of the accuracy evolution changing the initial *train size*: 50, 190 and 310 starting points.

Configurations:

TRAIN_SIZE	NON	NOL	NOE	LAMBDA	MAX_RHO	K
50	8	2	800	0.5	20	40
190	8	2	800	0.5	20	40
310	8	2	800	0.5	20	40

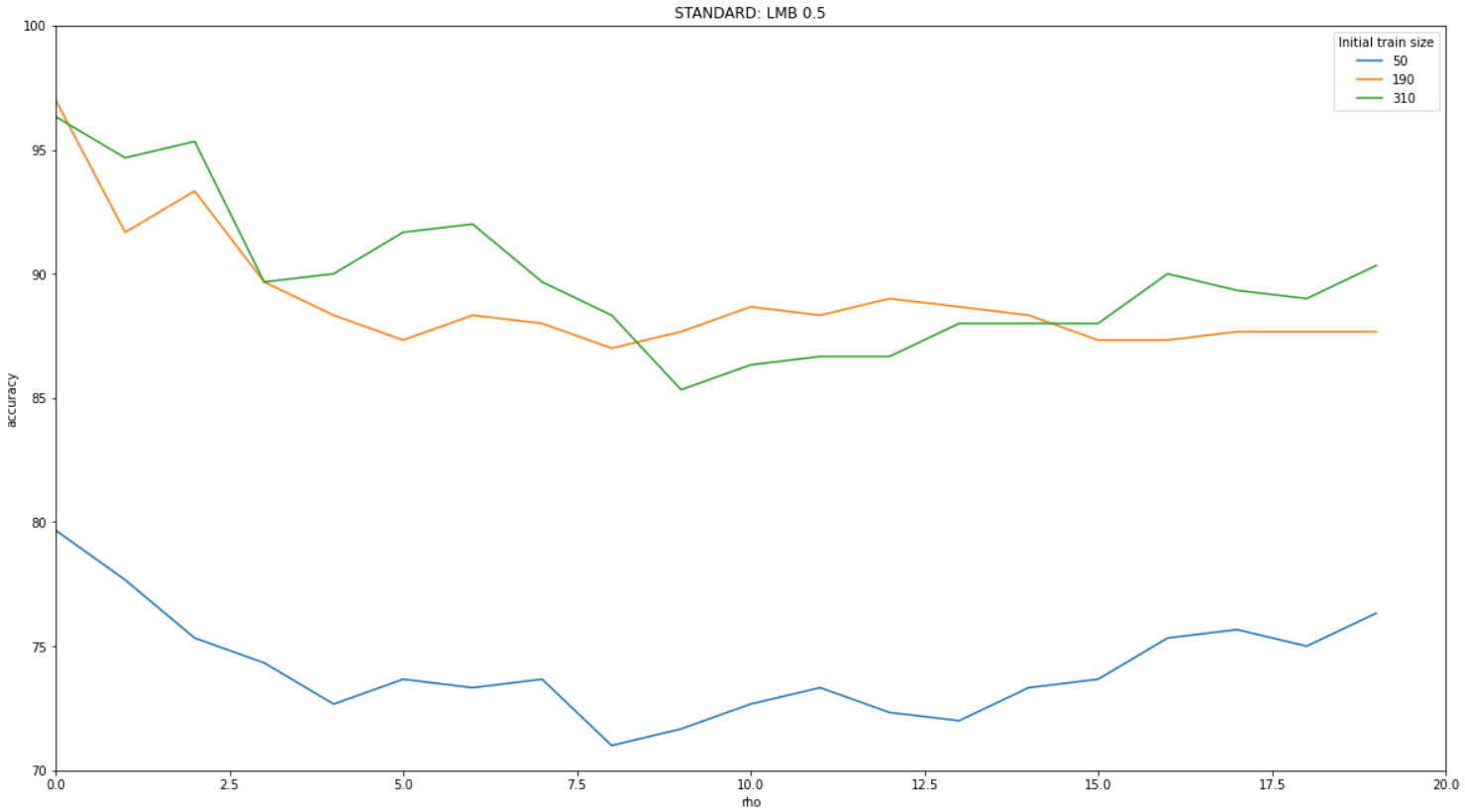


Fig.2: Comparison between different initial train set with lambda 0.5.

It can be noticed that the bigger is the starting dataset, the better is the accuracy of emulating the black-box.
We can also see that the behavior of the model is getting worse over the iterations.

Test 2: Changing Lambda value

For the second test the parameter that changes is the one regarding the shift step. We plot accuracies over time with the *Lambda* interval from 0.1 to 0.9.

Configurations:

TRAIN_SIZE	NON	NOL	NOE	LAMBDA	MAX_RHO	K
310	8	2	800	0.1	20	40
310	8	2	800	0.2	20	40
310	8	2	800	0.3	20	40
310	8	2	800	0.4	20	40
310	8	2	800	0.5	20	40
310	8	2	800	0.6	20	40
310	8	2	800	0.7	20	40
310	8	2	800	0.8	20	40
310	8	2	800	0.9	20	40

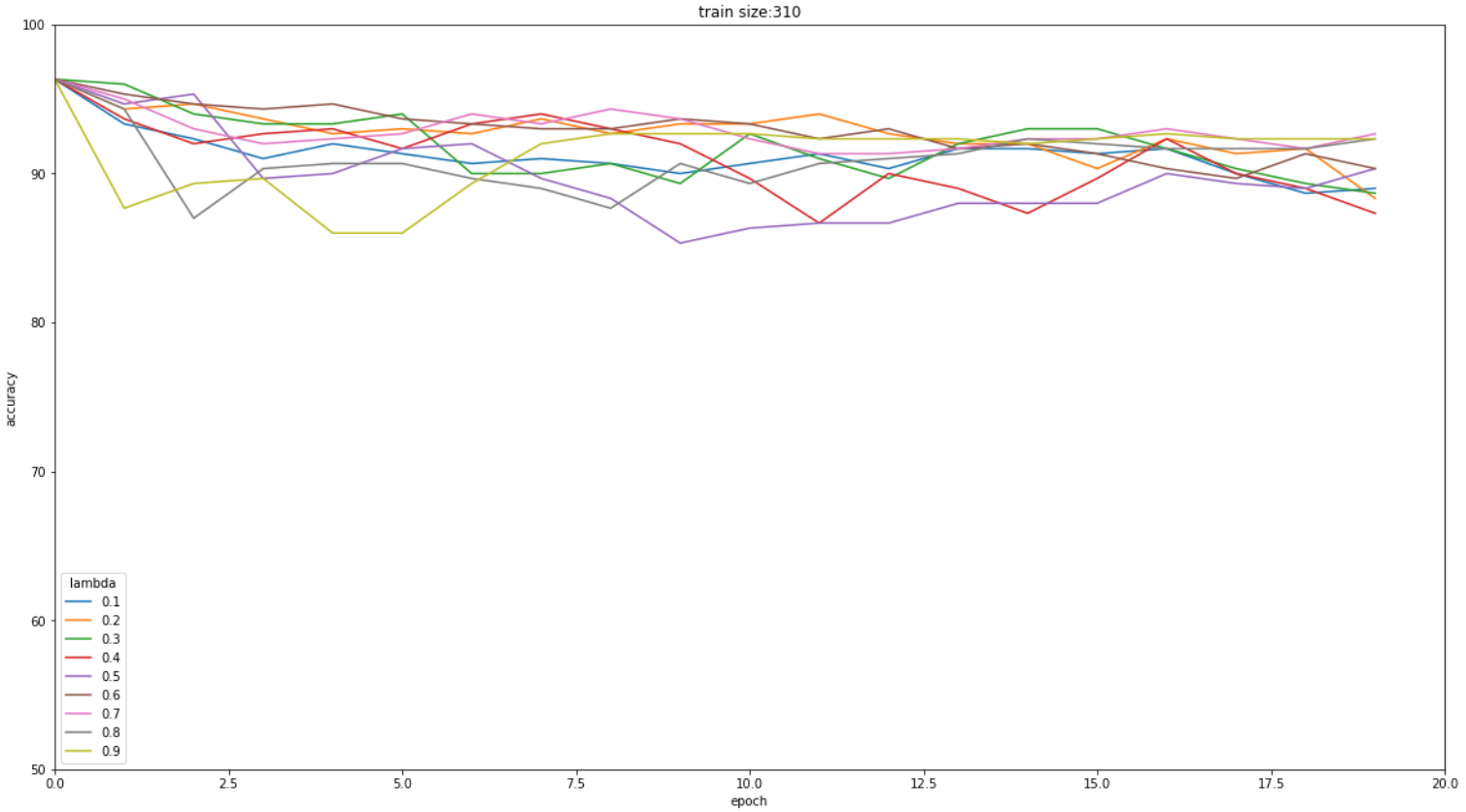


Fig.3: Comparison between different lambdas with initial train size 310.

The accuracies trend seems to be similar for all lambda values but let's take a better view of every single chart:

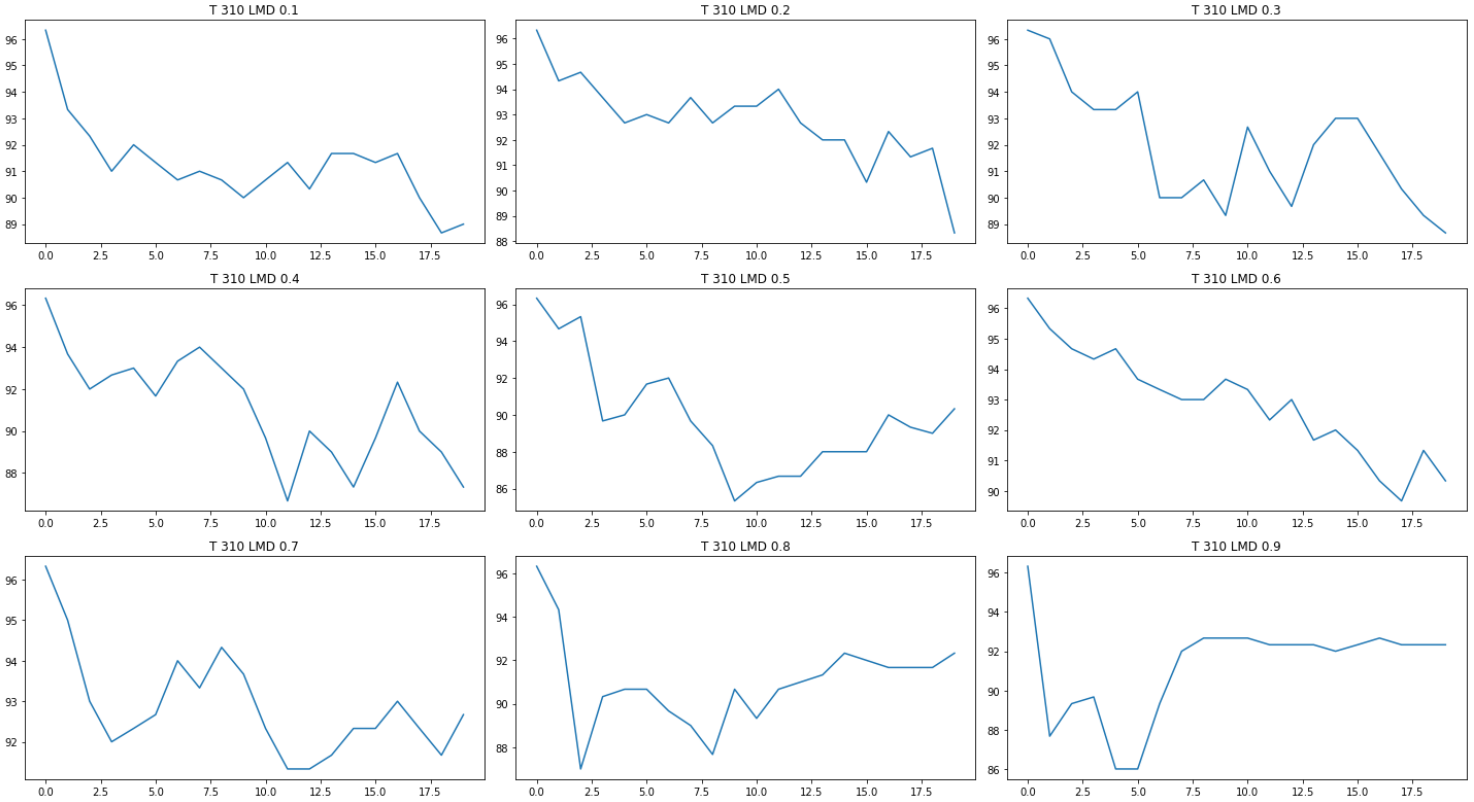


Fig.4: Comparison between different Lambda values (standard NN). Initial train size: 310.

We can see that the general trend is still descending. also changing the length of the shift step.

Test 3: Changing number of iterations

At this point we questioned if the descending trend is confirmed over a higher *number of iterations* (100). Our hope is to observe a recovery in accuracy values that outstands the initial one, the greatest in each test we have done.

Configurations:

TRAIN_SIZE	NON	NOL	NOE	LAMBDA	MAX_RHO	K
50	8	2	800	0.1	100	40
190	8	2	800	0.1	100	40
310	8	2	800	0.1	100	40

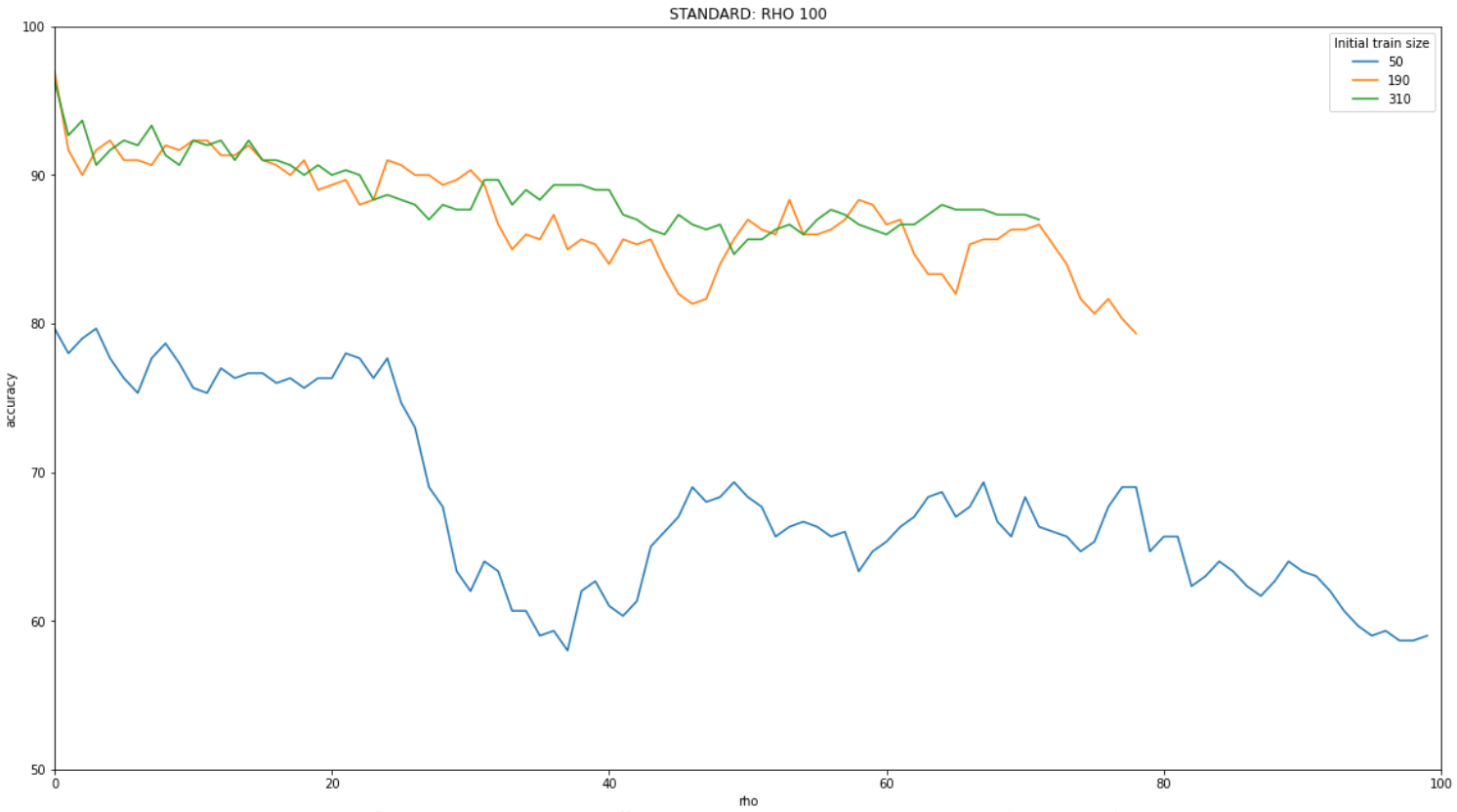


Fig.5: Comparison between different initial train size with lambda 0.1 and rho 100.

Also with a longer simulation process, the trend is still descending. We can observe some recoveries but the initial accuracy value is always the greatest.
Two tests have been forced to stop due to some errors in the synthetic points generation phase. The error consists in generating too small coordinates values for a point causing the process to fail.

Test 4: Changing NN architecture

In these tests we tried to use a different *NN* model, a more complex one instead of the previous. It also introduces a *dropout* of 20% on hidden layers.
Our expectation is to observe a better accuracy trend than before.
The first *NN* we propose is the following one:

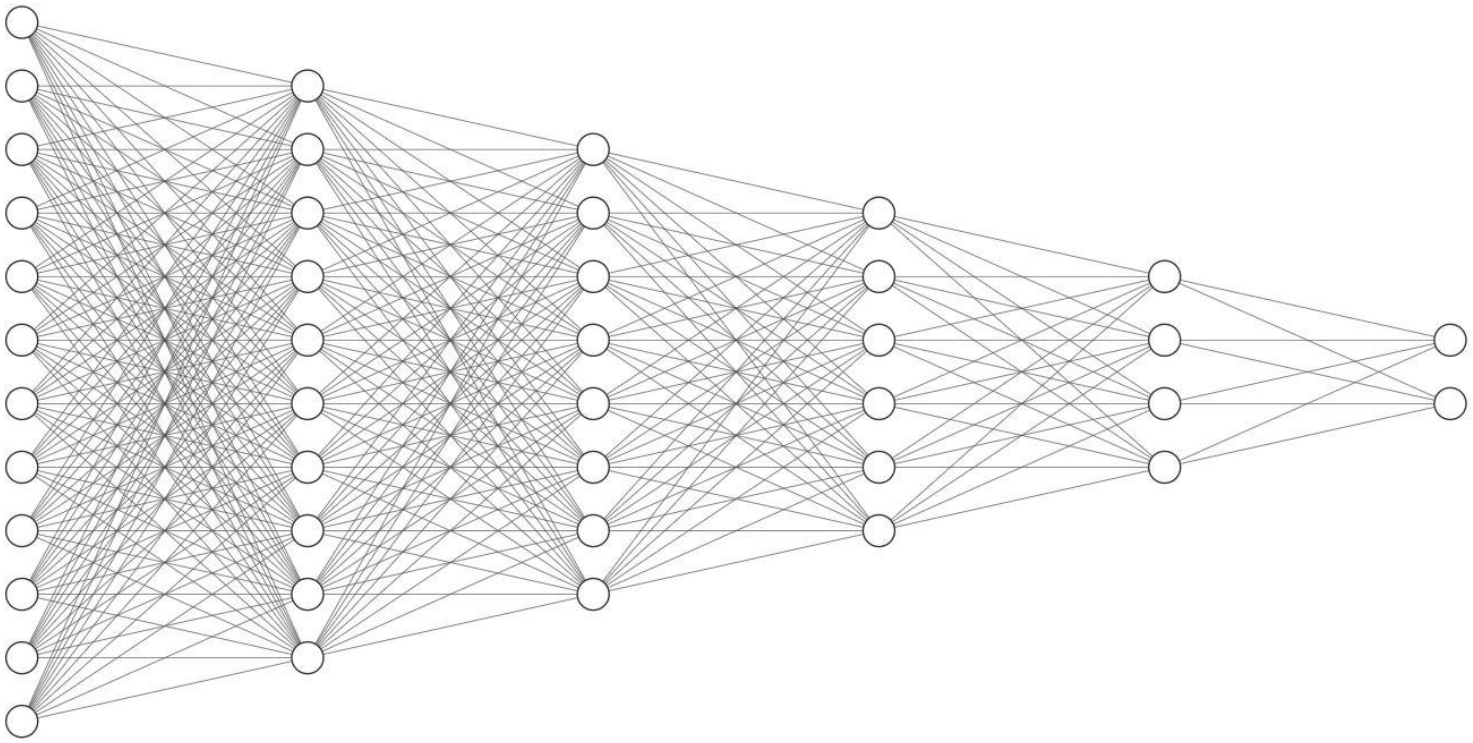


Fig.6: NN custom architecture with 81 input nodes, 4 hidden layers with 64-32-16-8 nodes and 2 output nodes.

The next chart presents the accuracy evolution using the new *NN,* changing the initial *train size*. The parameter *NN_ARC* defines the structure of hidden layers. The configurations are as following:

TRAIN_SIZE	NN_ARC	NOE	LAMBDA	MAX_RHO	K
50	64-32-16-8	800	0.5	20	40
190	64-32-16-8	800	0.5	20	40
310	64-32-16-8	800	0.5	20	40

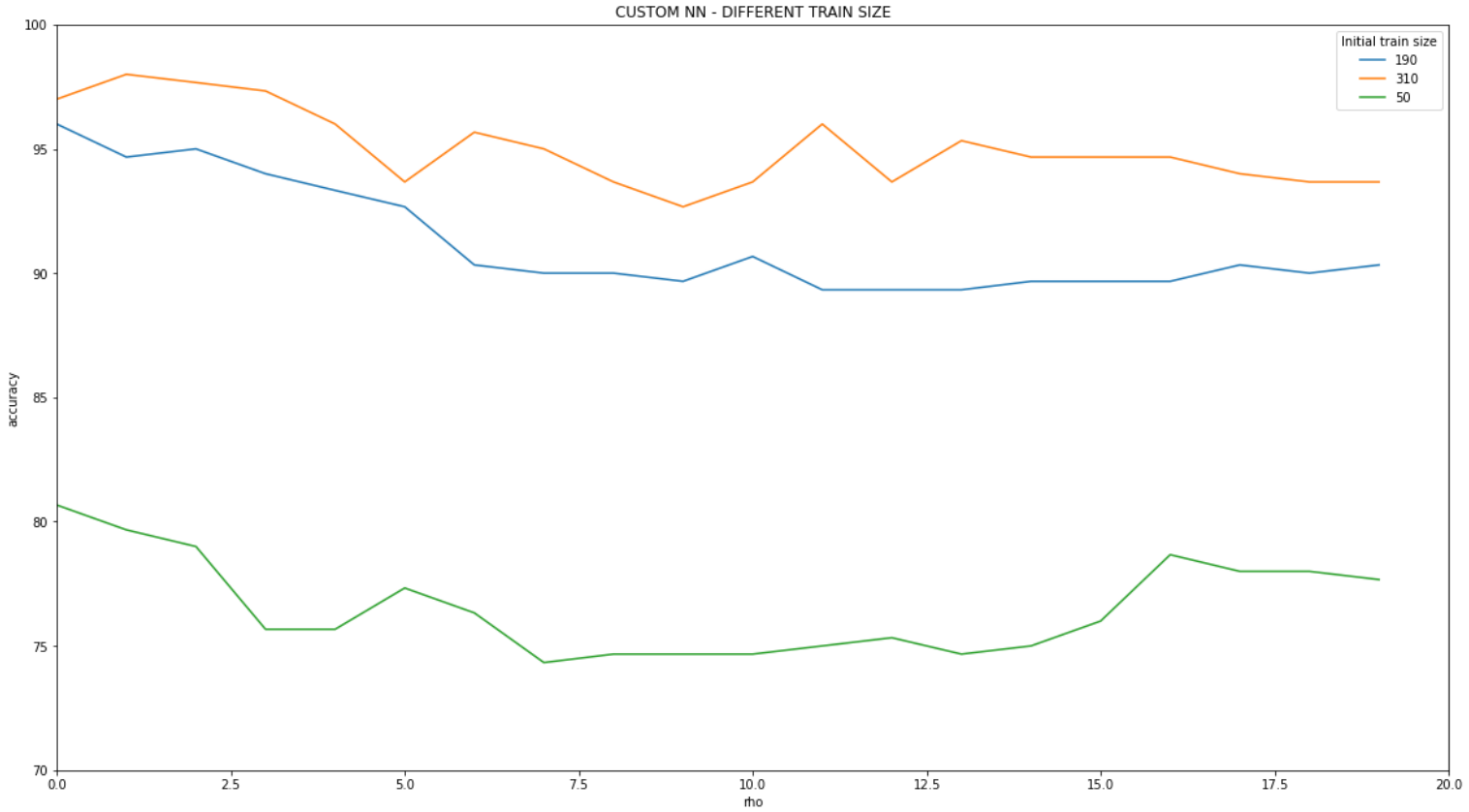


Fig.7: Comparison between different train size with custom nn architecture.

We can see that with a more complex *NN* the accuracy does not lose too much of its starting trend and tends to always be the same value in every initial train size case.
To confirm the benefits of a more complex model we tried with other two more complex architectures (64-32-32-32) and (128-96-64-48-32).
We now compare 4 different models over the same starting input of 190 points:

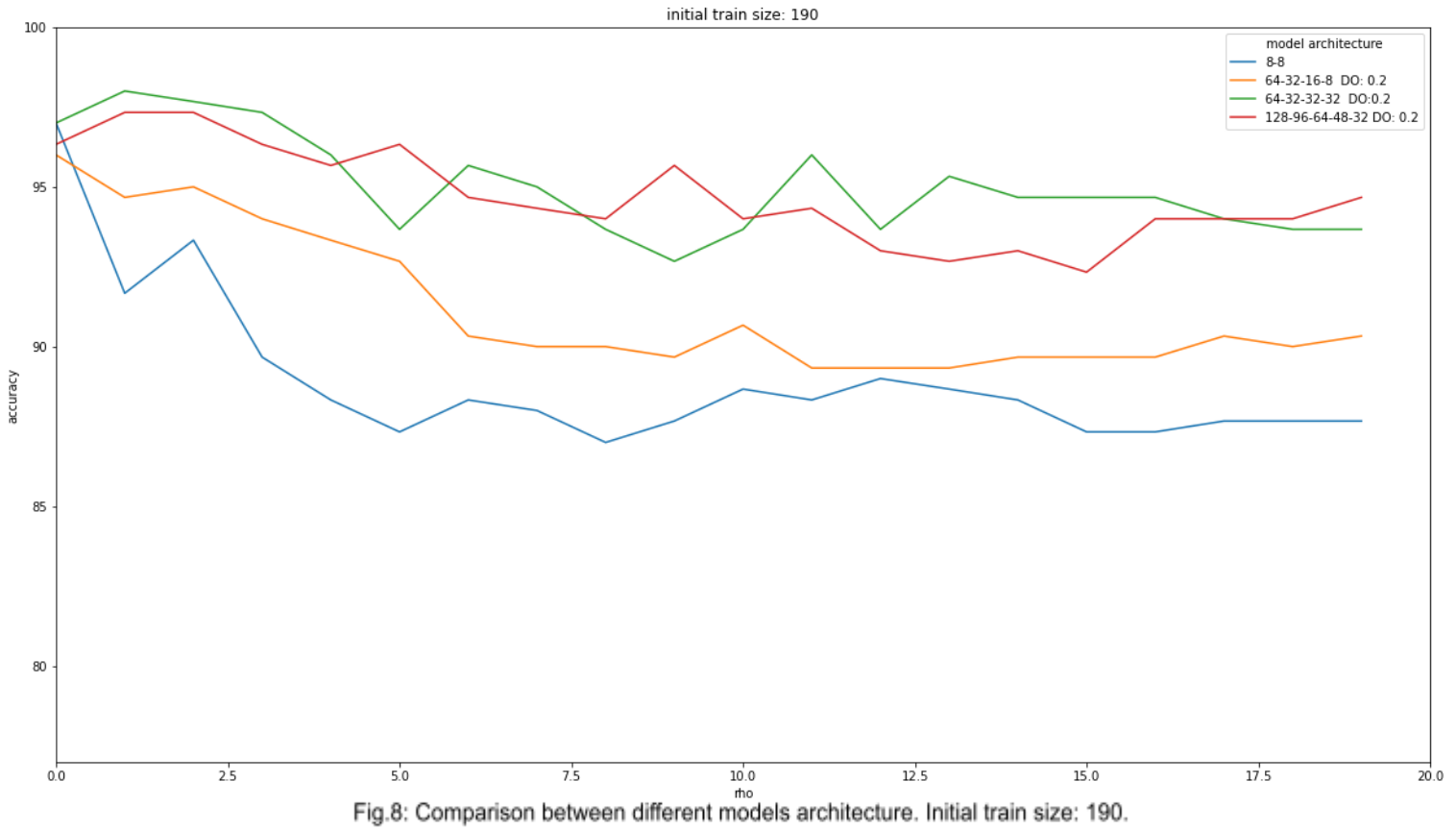


Fig.8: Comparison between different models architecture. Initial train size: 190.

The plot shows that a more *complex model* can guarantee a constant trend without accuracy loss. Comparing the two most complex models, we can see that the accuracy trend is similar. We can conclude that over a certain level of complexity the accuracy does not improve.
This last fact is confirmed by plotting the 310 initial train size chart: the two most complex models do have the same accuracy trend.

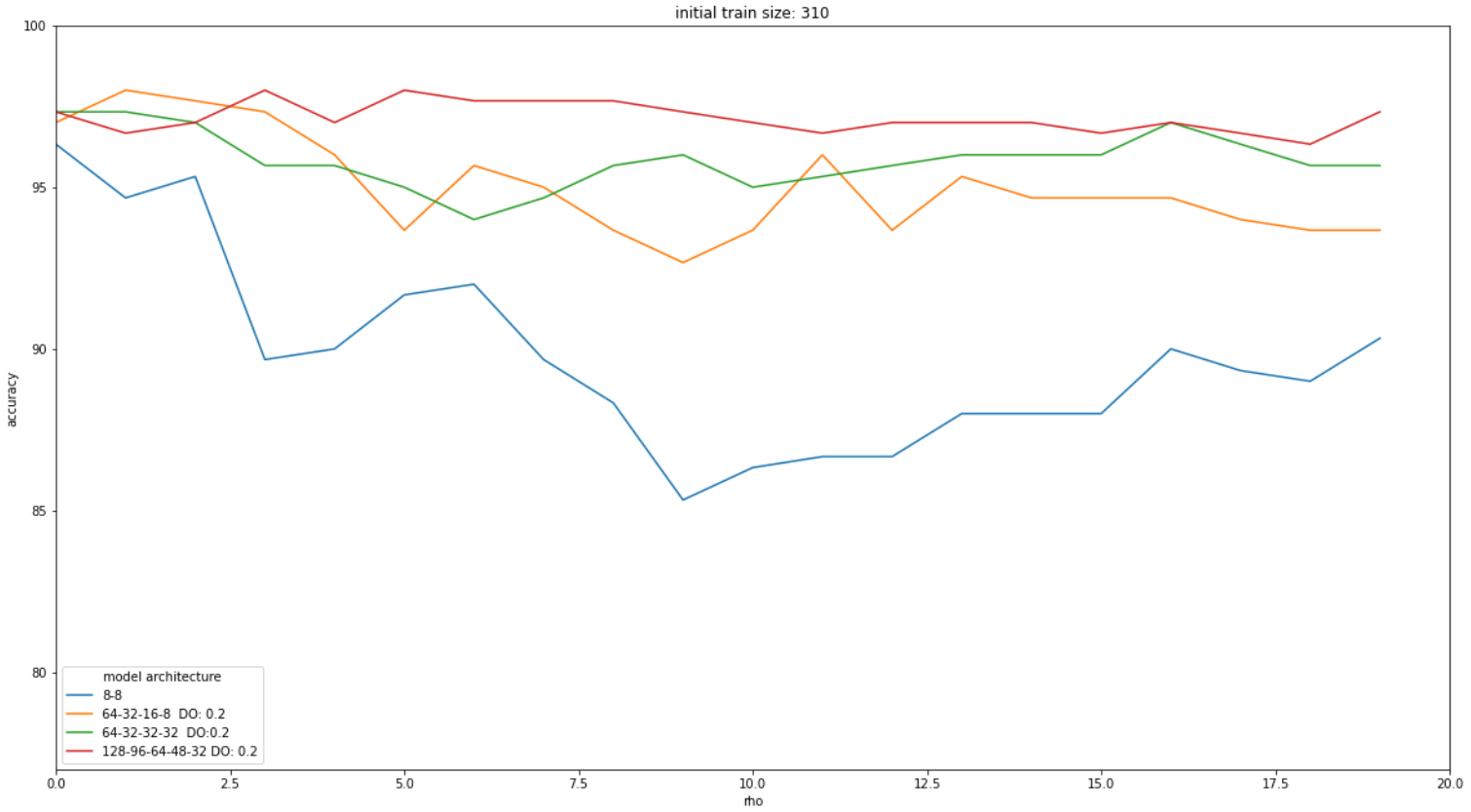


Fig.9: Comparison between different models architecture. Initial train size: 310.

Test 5.1: Changing number of training epochs

After all the tests done so far, we are now deadlocked: despite trying to change all the parameters relating to the points (initial dataset, displacement and so on) and trying to change the architecture of the model, no further improvements are observed. We now concentrate on different train-modes for the network.
We wonder if changing the number of training epochs compromises the accuracy value.
Here the tests we have done:

TRAIN_SIZE	NN_ARC	NOE	LAMBDA	MAX_RHO	K
310	64-32-32-32	5	0.5	20	40
310	64-32-32-32	25	0.5	20	40
310	64-32-32-32	50	0.5	20	40
310	64-32-32-32	100	0.5	20	40
310	64-32-32-32	200	0.5	20	40
310	64-32-32-32	400	0.5	20	40
310	64-32-32-32	800	0.5	20	40

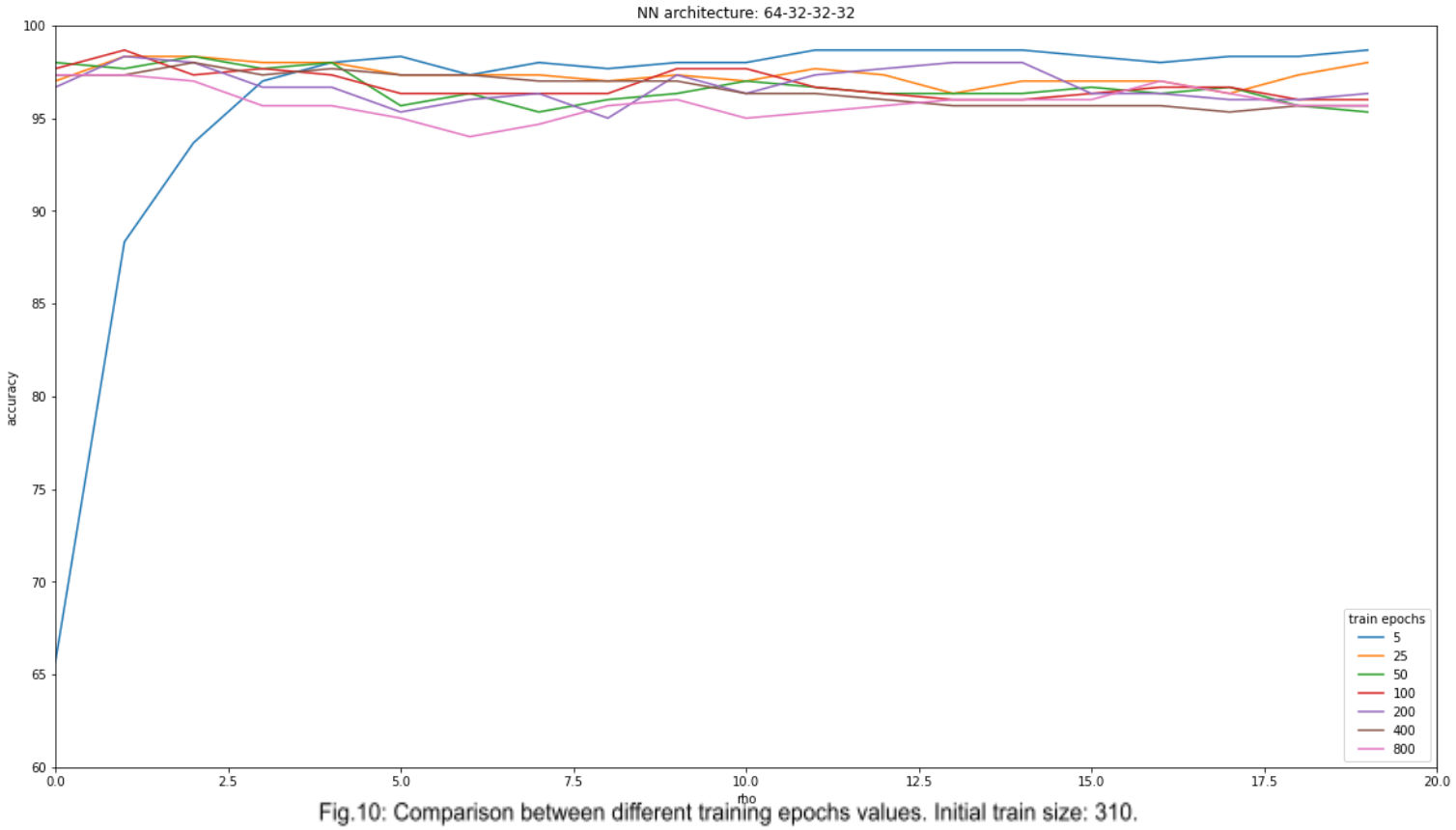


Fig.10: Comparison between different training epochs values. Initial train size: 310.

As we can see the number of training epochs doesn't affect the final accuracy value.
Despite the low number of training epochs that have bad initial accuracy, the trend realigns with the other tests that have more training epochs available.

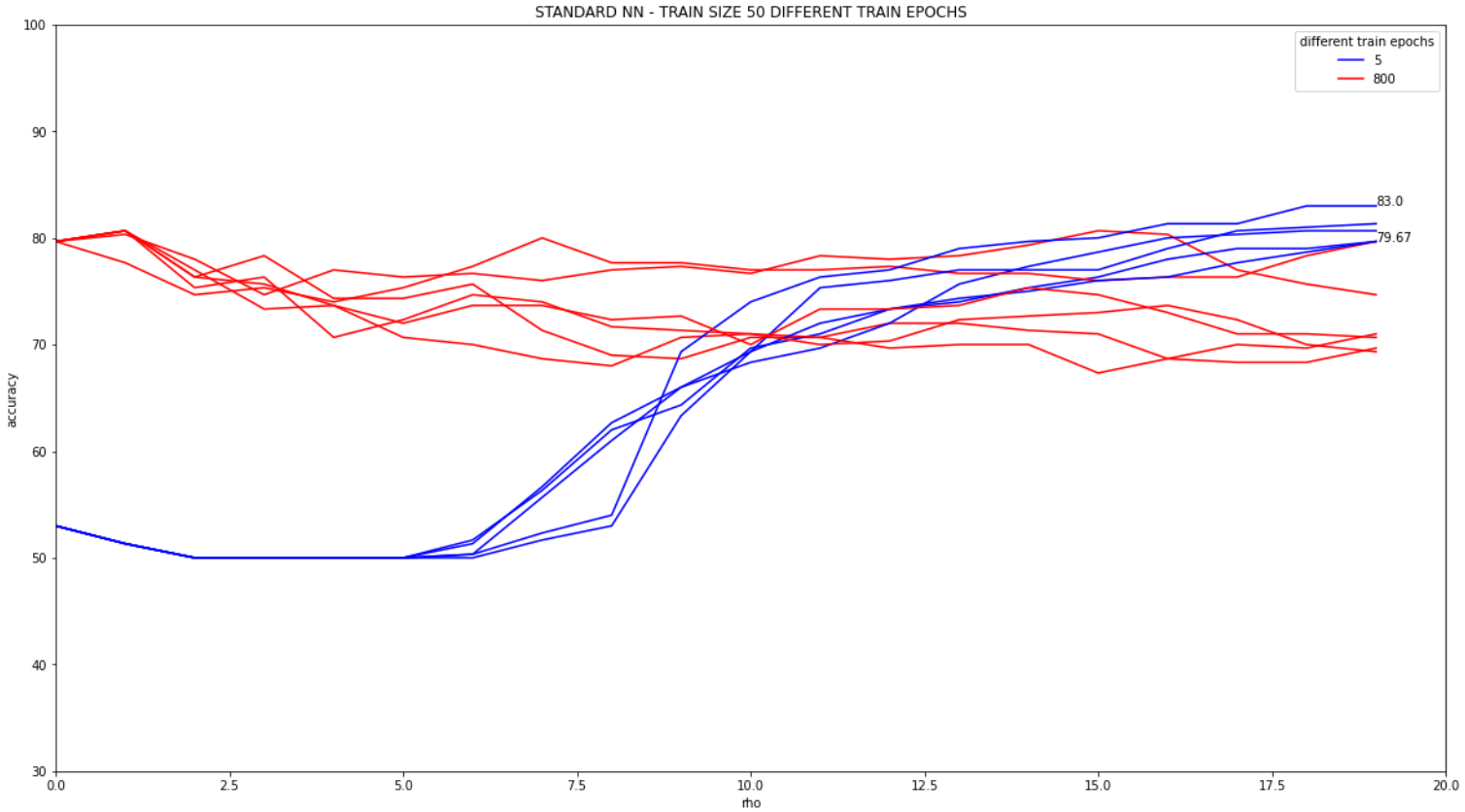
Test 5.2: More in detail

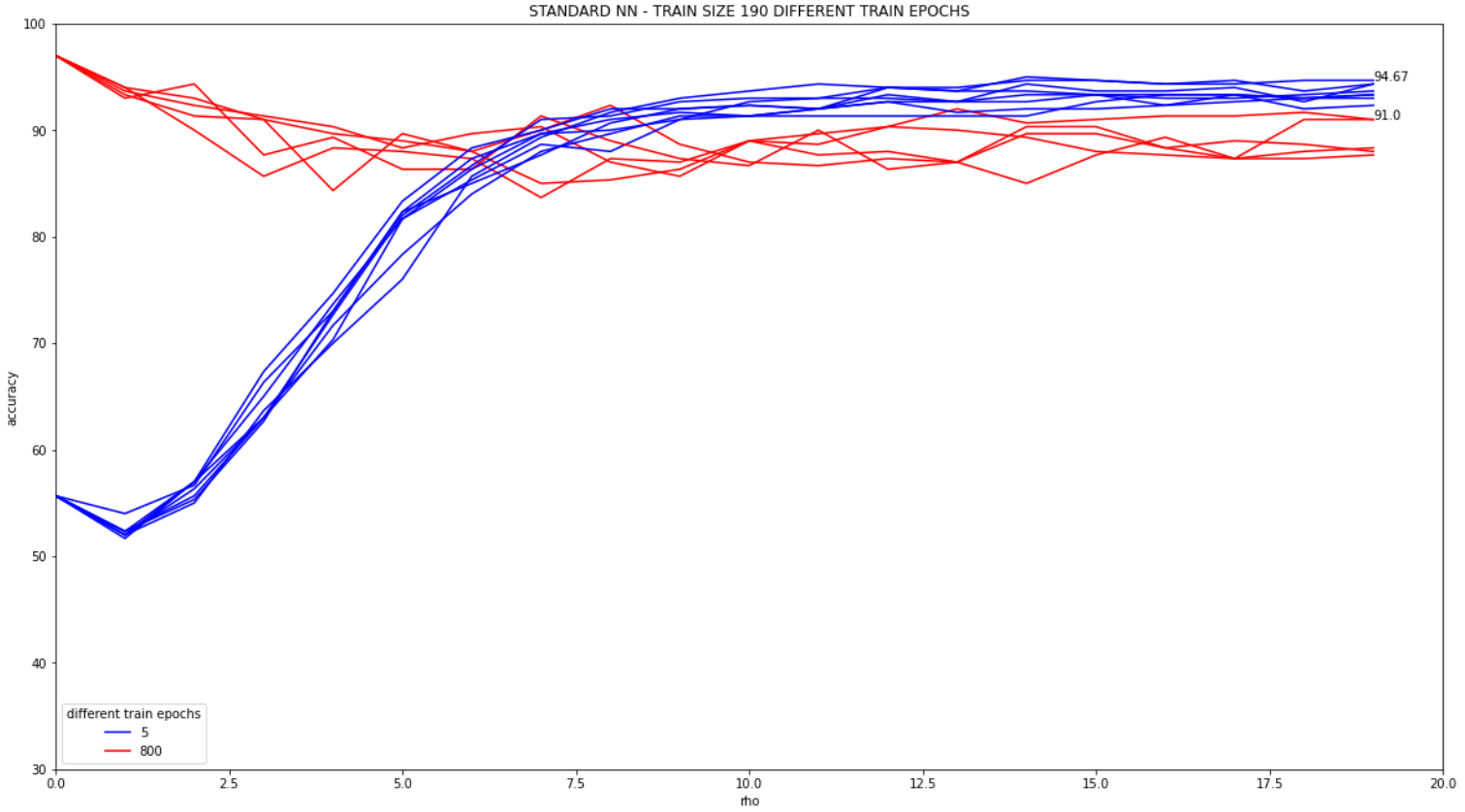
From test 5.1 we notice how the final accuracy seems not to be affected by the duration of the training. To confirm that the previous results are not due to coincidence, we run further tests to confirm this fact. We also notice that the final accuracy of the system increases. Let's take the two extremes as an example for comparisons: 800 training epochs and 5 training epochs.

Let us consider the following configurations which differ only in the number of training epochs. Note that we are using the standard network which turned out to be the worst in the tests reported above.

TRAIN_SIZE	NN_ARC	NOE	LAMBDA	MAX_RHO	K
50	8_8	5 / 800	0.5	20	40
190	8_8	5 / 800	0.5	20	40
310	8_8	5 / 800	0.5	20	40

For each configuration we repeat the tests several times. We are able to observe that the general trend reflects the previous results (so was not a coincidence).
We also report the maximum accuracy for each configuration next to the graph.





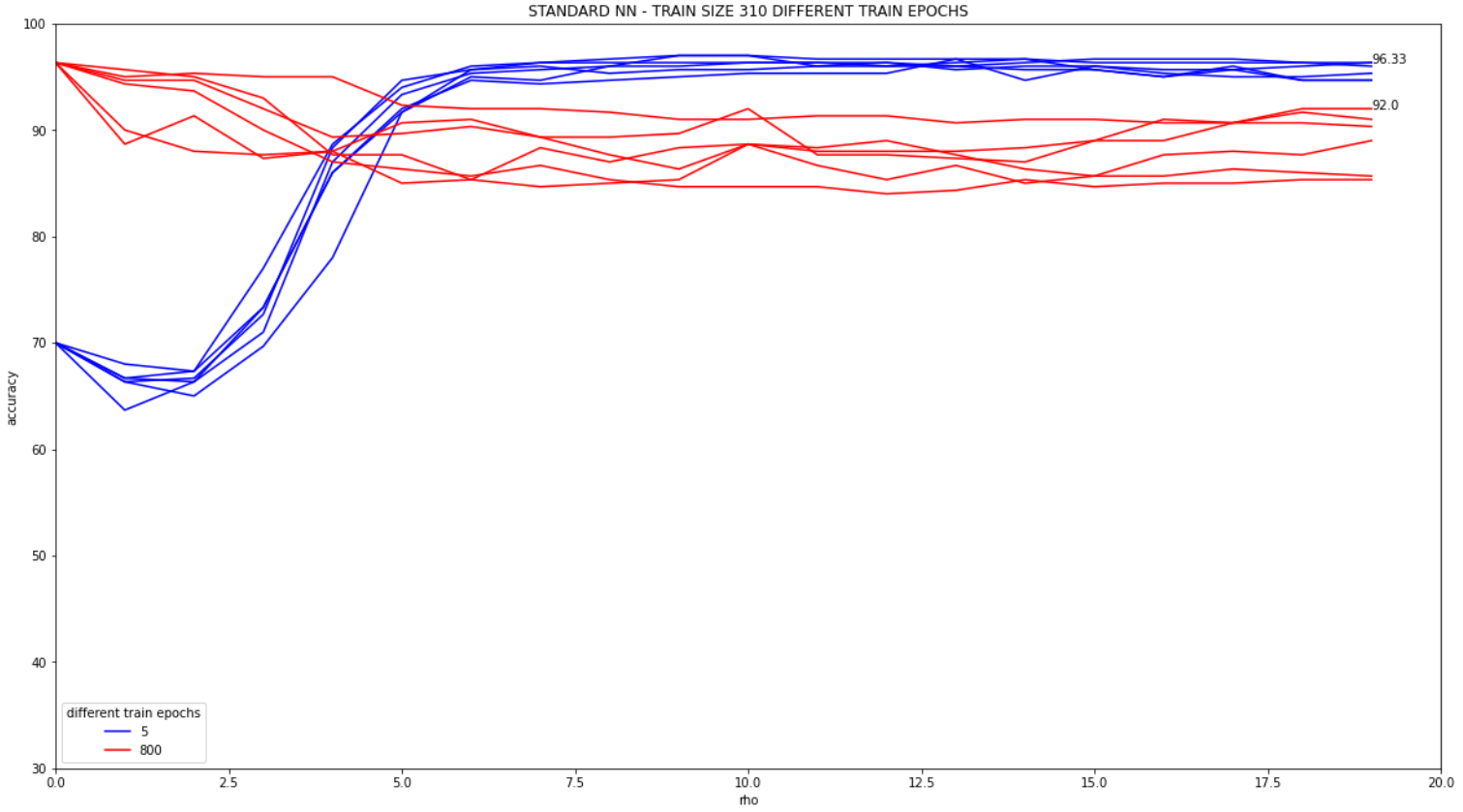


Fig.11: Comparison between different training epochs values (standard NN). Initial train size: 50 - 190 - 310.

Let's take a look at the plots. In every case in which the training epochs are set as 5 there is a low initial accuracy. On the other hand, after a few iterations, there is fast recovery in

performance that ultimately exceeds the accuracy obtained with 800 epochs. What is very important is that this behavior is confirmed in every test, thus discarding the possibility that it was a lucky result and finding out that less training epochs get a better final accuracy for each initial train size.

Another important observation is that the behavior is no longer constantly decreasing: in the previous tests the accuracy decreased during the iterations. With these new configurations, the system behaves in a totally opposite way: the trend is increasing even if it always tends to converge to the same range of values based on the initial training dataset.

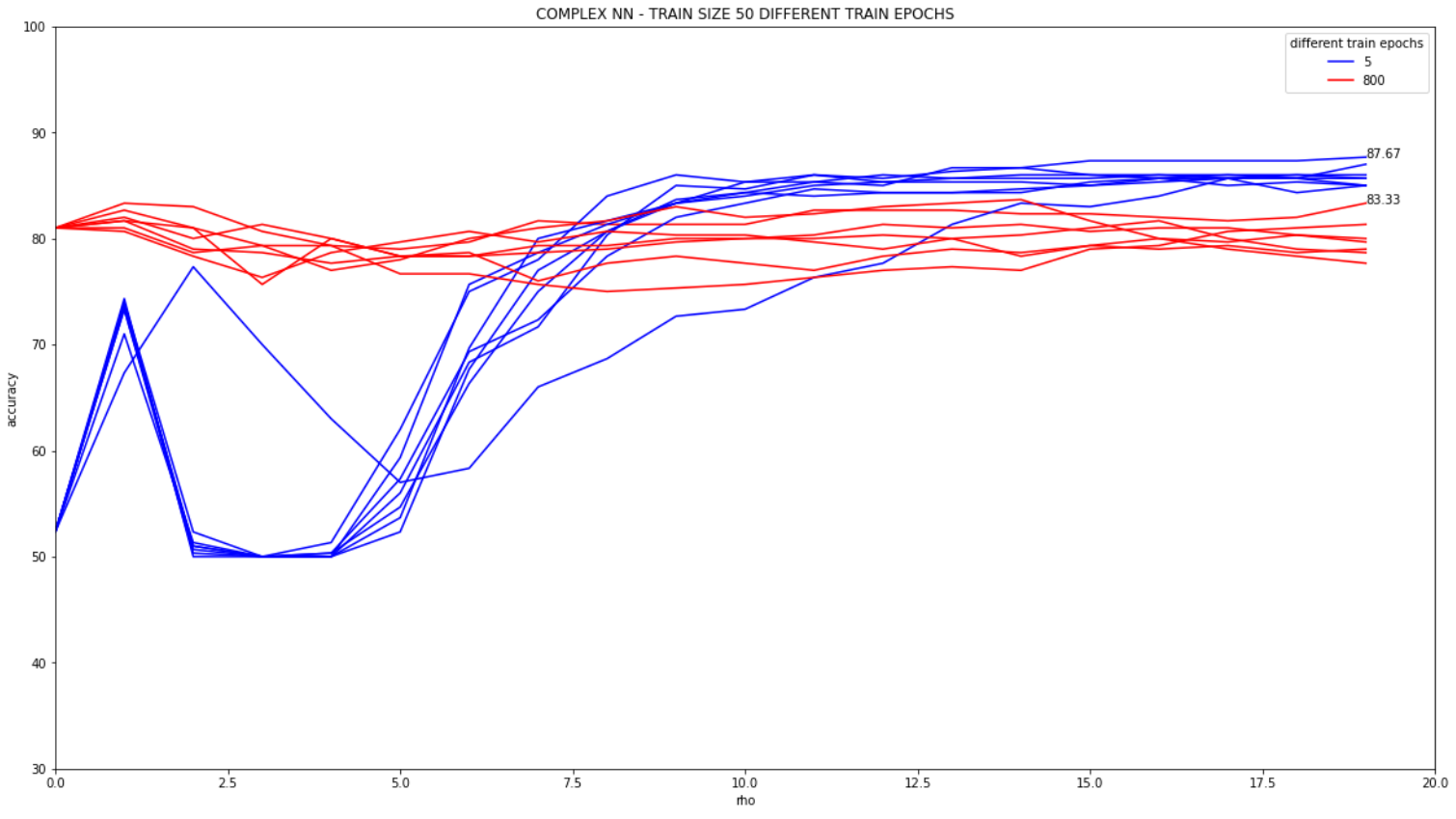
With these tests the fact of having the final accuracy dependent on the size of the initial training is also confirmed.

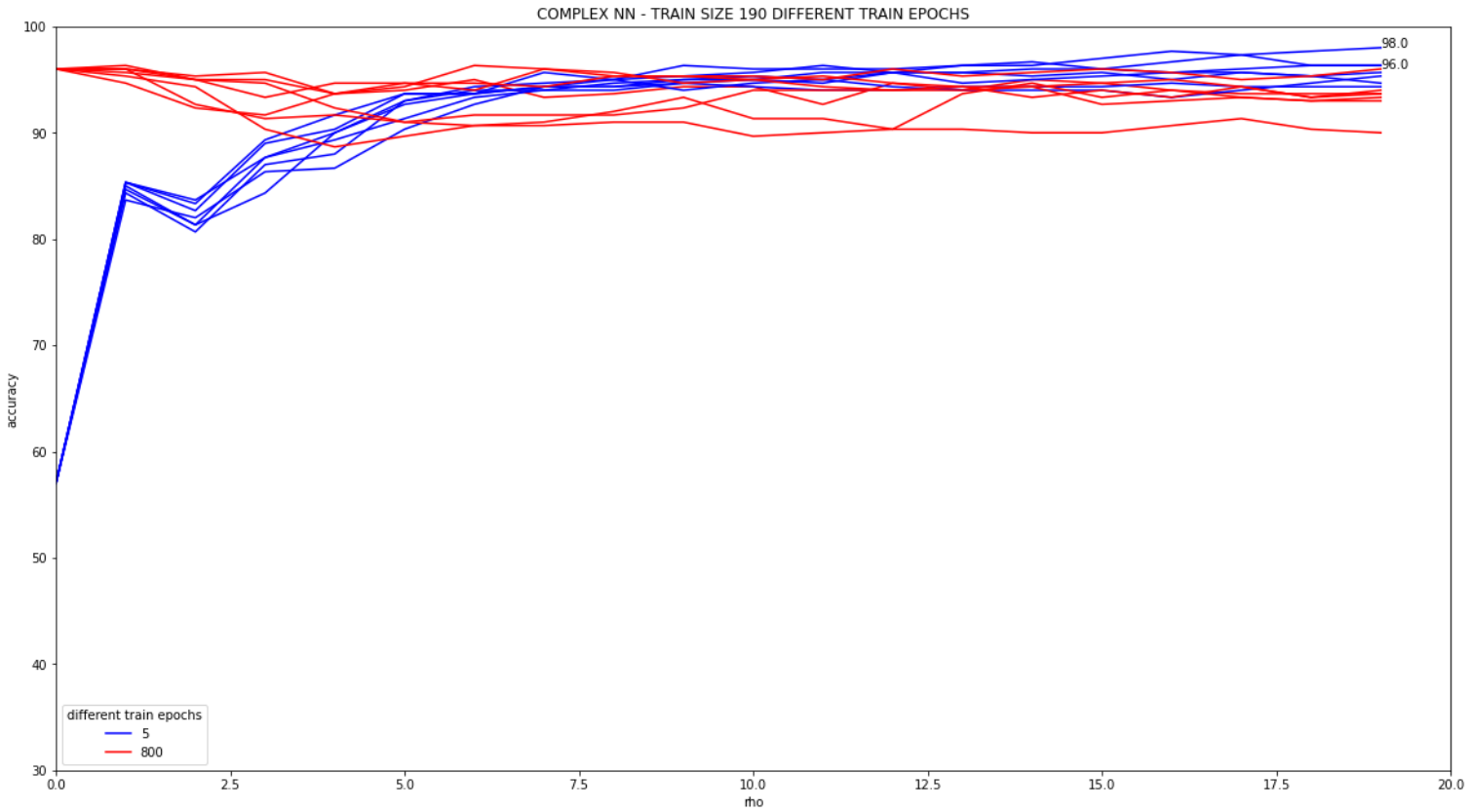
Test 5.3: Comparison between different architectures

At this point we compare a more complex network (64_32_32_32) against the standard network (8_8) to verify that the new behavior with a training of only 5 epochs was confirmed and was not dependent on the structure of the network itself.
First of all, we compared the system performance with the complex network by changing only the epoch parameter:

TRAIN_SIZE	NN_ARC	NOE	LAMBDA	MAX_RHO	K
50	64_32_32_32	5 / 800	0.5	20	40
190	64_32_32_32	5 / 800	0.5	20	40
310	64_32_32_32	5 / 800	0.5	20	40

Also in this case we repeat the tests several times to have more consistent results and avoid lucky cases:





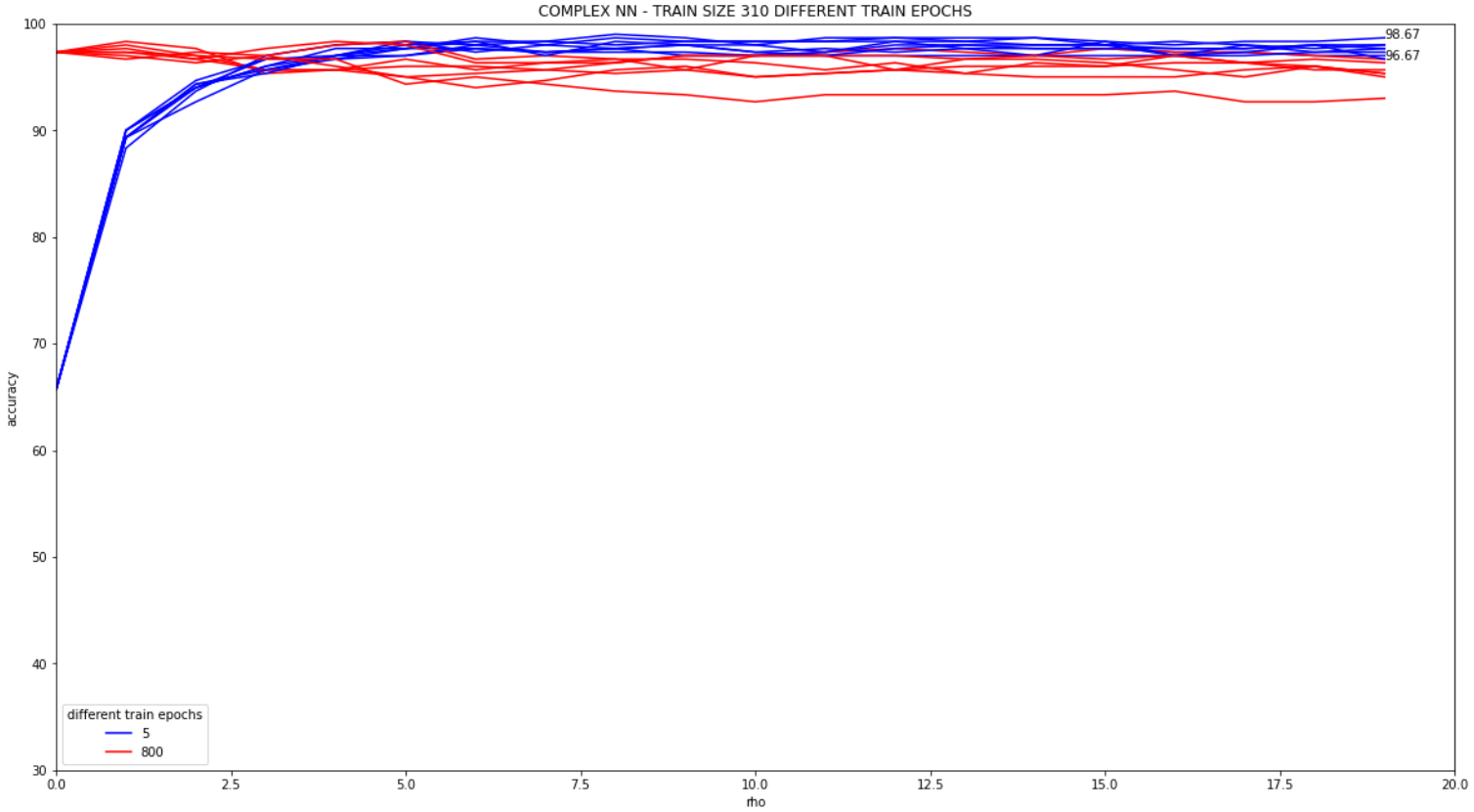
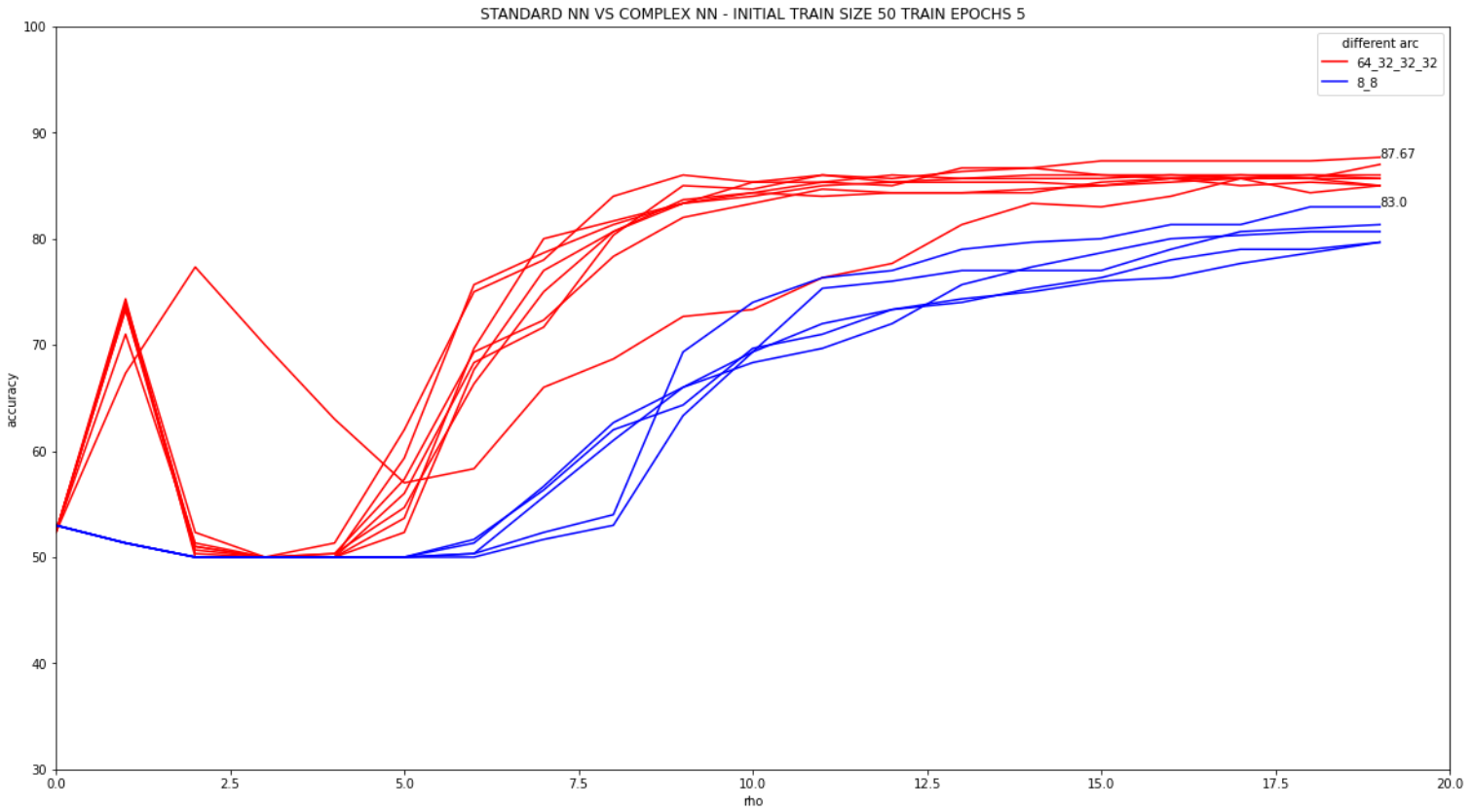
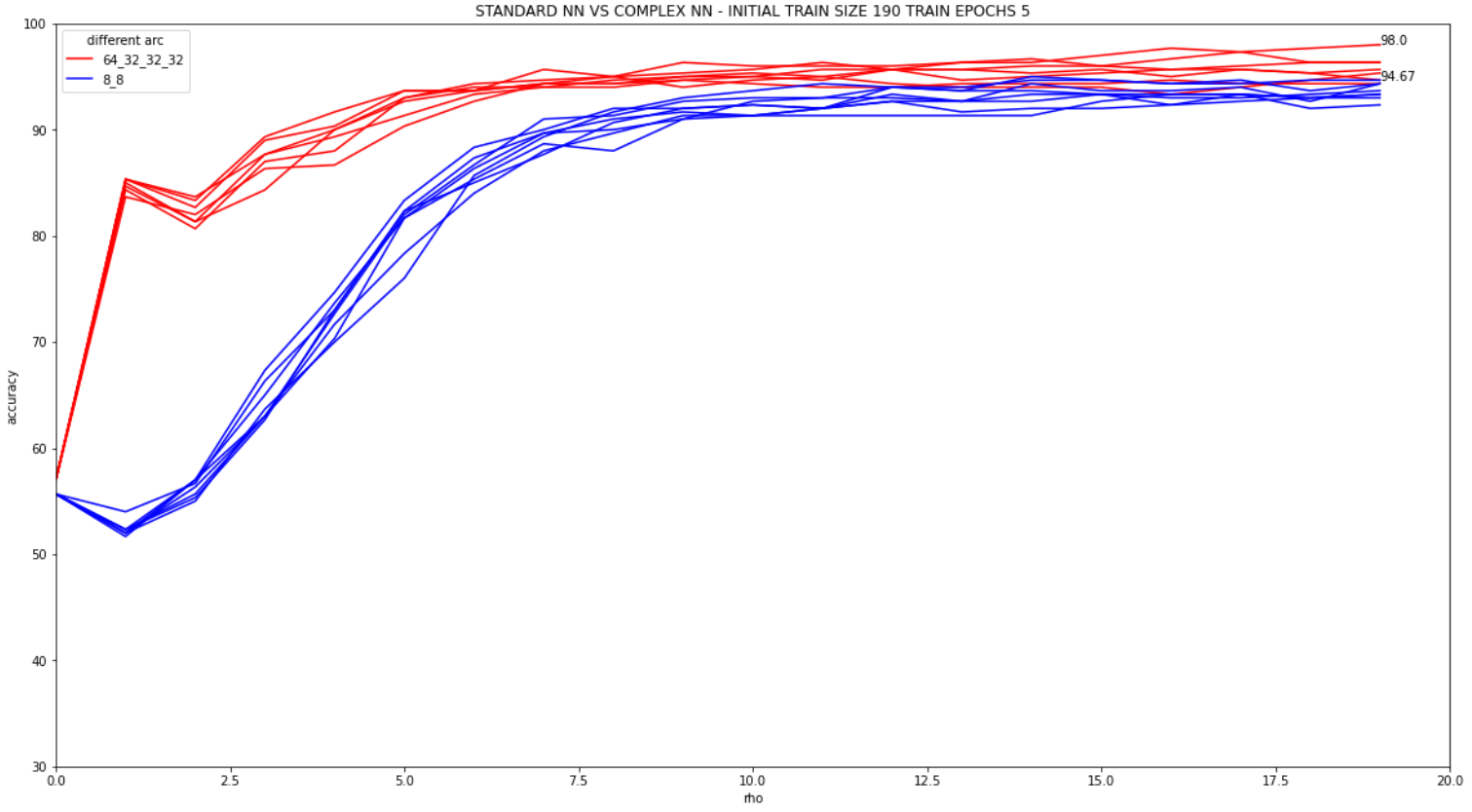


Fig.12: Comparison between different training epochs values (complex NN). Initial train size: 50 - 190 - 310.

The system's behavior is confirmed another time. Accuracy increases with the number of iterations and final accuracy is better by setting 5 epochs rather than 800.

We now compare the two architectures tested previously:





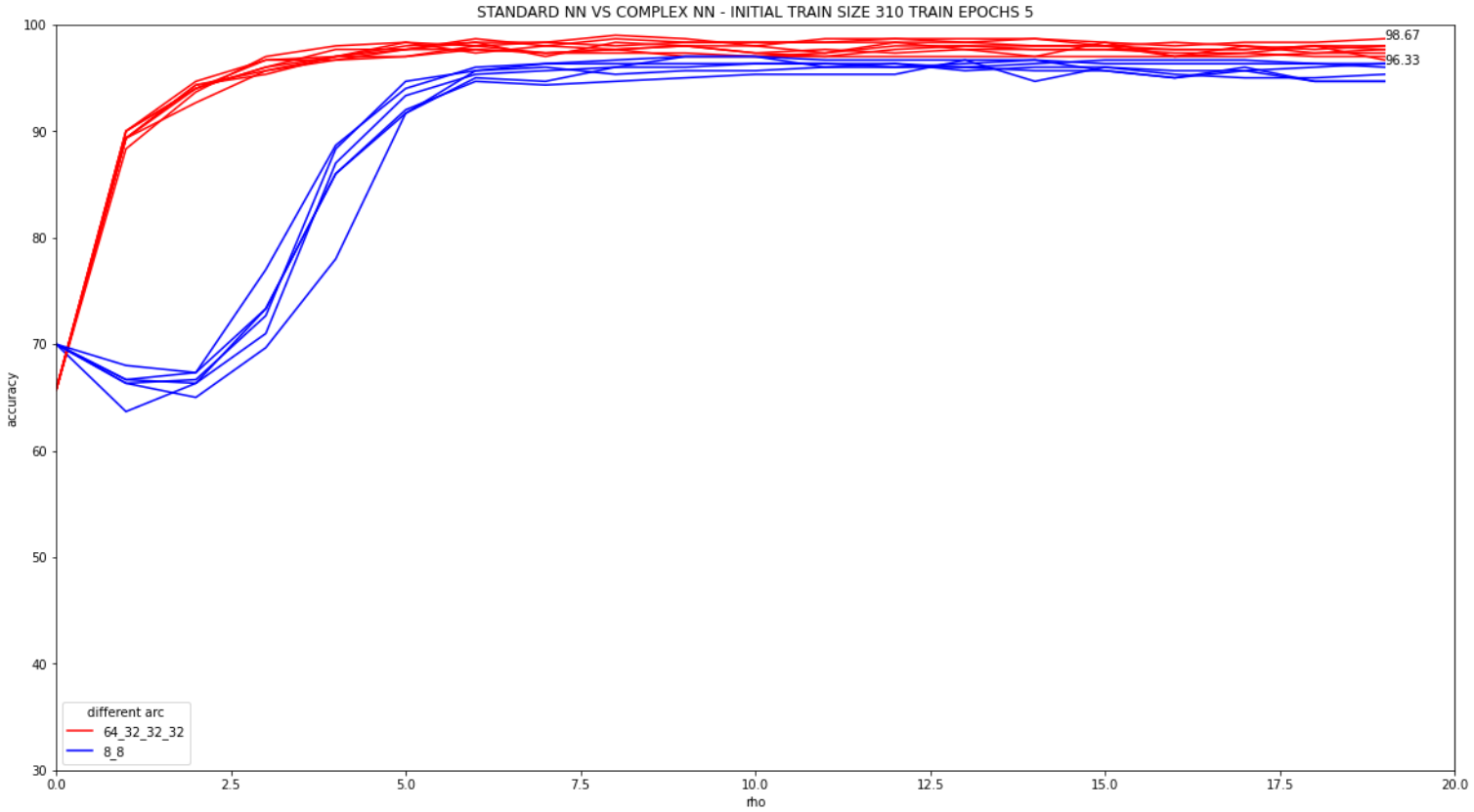


Fig.13: Comparison between different architectures (standard VS complex).
Initial train size: 50 - 190 - 310, Training epochs: 5.

Confirmed the fact that a more complex network is able to guarantee greater accuracy than a shallower network with fewer nodes. We can observe that the most complex network that we found guarantees at least 3% more accuracy than the initial one.

Observations using Jacobian Augmentation

After many tests we can now conclude with some considerations.
Testing the system with its default configuration (standard NN, 800 epochs and so on), we observe a descending accuracy trend, a scenario we want to avoid.
Our attempt to resolve this issue is to change this configuration and try new *NN* models that simulate the Black-Box.

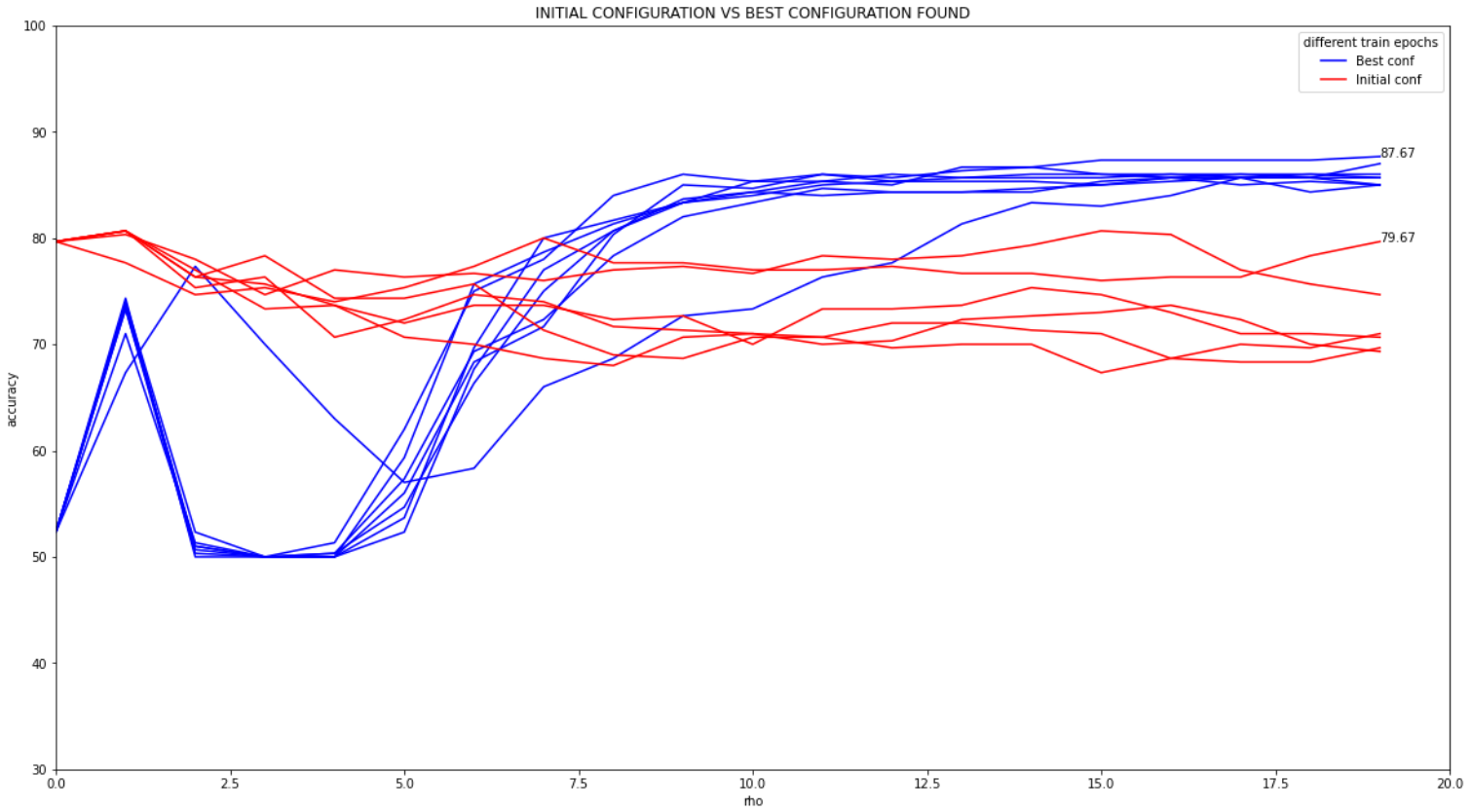
After some tests, we noticed that changing all the possible parameters regarding the points available does not improve the performances and so we moved in another direction focusing on the network.

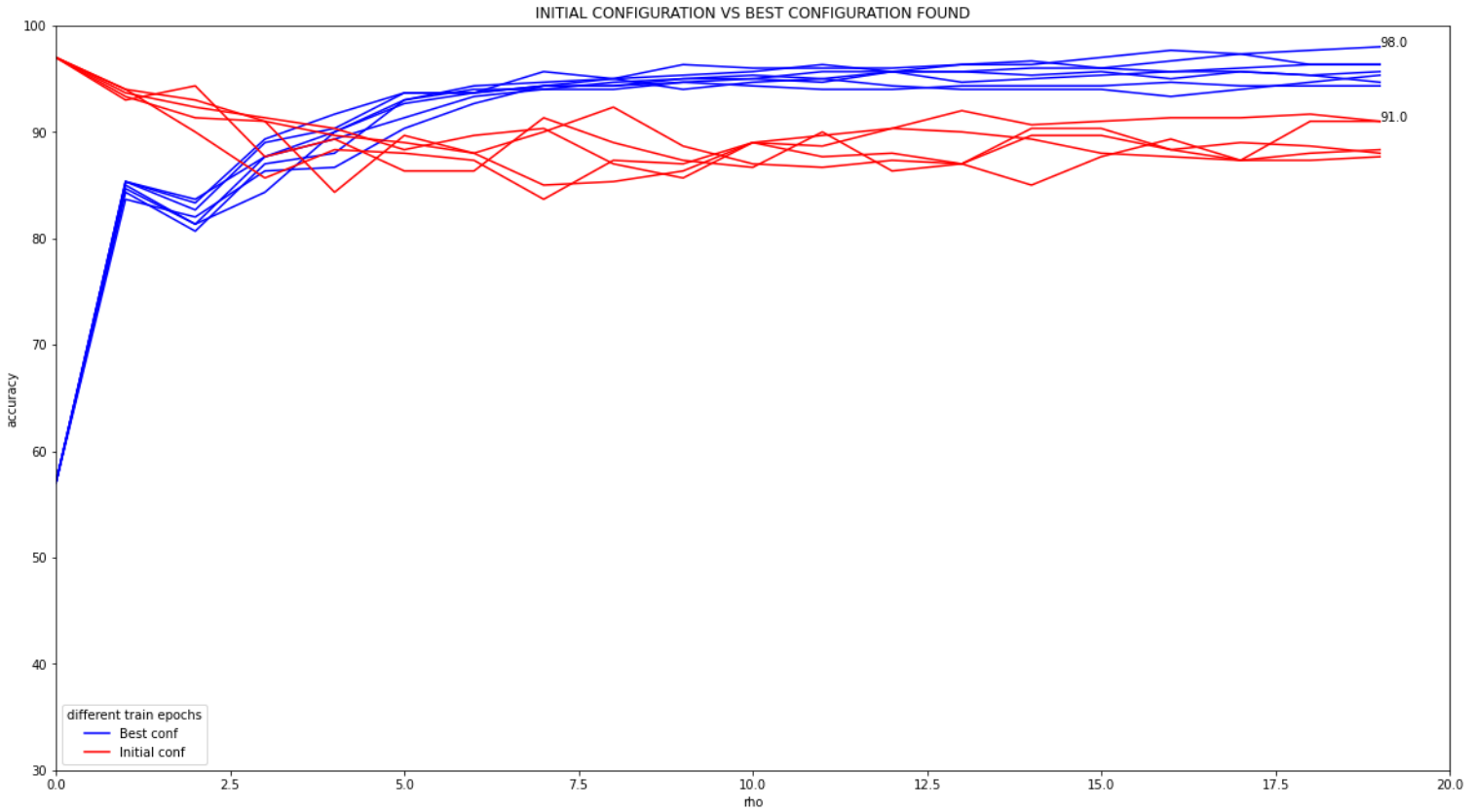
Finally we have encouraging results: using more complex *NN* models the accuracy trend does not decrease, changing the system behavior to be constant over iterations.
Observing the chart, in some phases, it's also possible to see an increment in accuracy values that are greater than the initial one, a fact that does not occur with the default *NN*.
Another fact we report is that even more complex models do not guarantee a better accuracy trend than the constant one: when the right complexity model is reached, it's useless to add larger layers to the *NN* model.

Failing to improve performance further, we focused on the training parameter, number of epochs. After many tests (100-50-25-5 epochs) we had the best performance setting 5 epochs for training.
We observed a noticeable change in the system behavior by adjusting the NOE parameter: there is no longer a constant or slightly downward trend. Accuracy grows over iterations and the final value is greater than setting 800 epochs as before.

The last comparison graphs we propose show the behavior of the initial system (descending trend accuracy) in comparison with the best configuration we have experienced.
By changing the architecture of the NN and adjusting its training parameters, we have increased the accuracy.

Concluding, to get good performances of the system, it's essential to have a ***good initial training set*** available, large enough for giving our simulating model the necessary information about the black-box behavior.
Another essential condition is to deploy a ***sufficiently complex NN model***: its complexity has to be comparable to the black-box one. To know the least complexity model needed has to be observed that the accuracy trend doesn't descend over iterations. Adjusting the ***training epochs*** parameter (NOE) allows to increase even more the overall system accuracy.


INITIAL CONFIGURATION VS BEST CONFIGURATION FOUND
different train epochs
Best conf
Initial conf
87.67
79.67
accuracy
rho
100
90
80
70
60
50
40
30
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


INITIAL CONFIGURATION VS BEST CONFIGURATION FOUND
different train epochs
Best conf
Initial conf
98.0
91.0
accuracy
rho
100
90
80
70
60
50
40
30
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

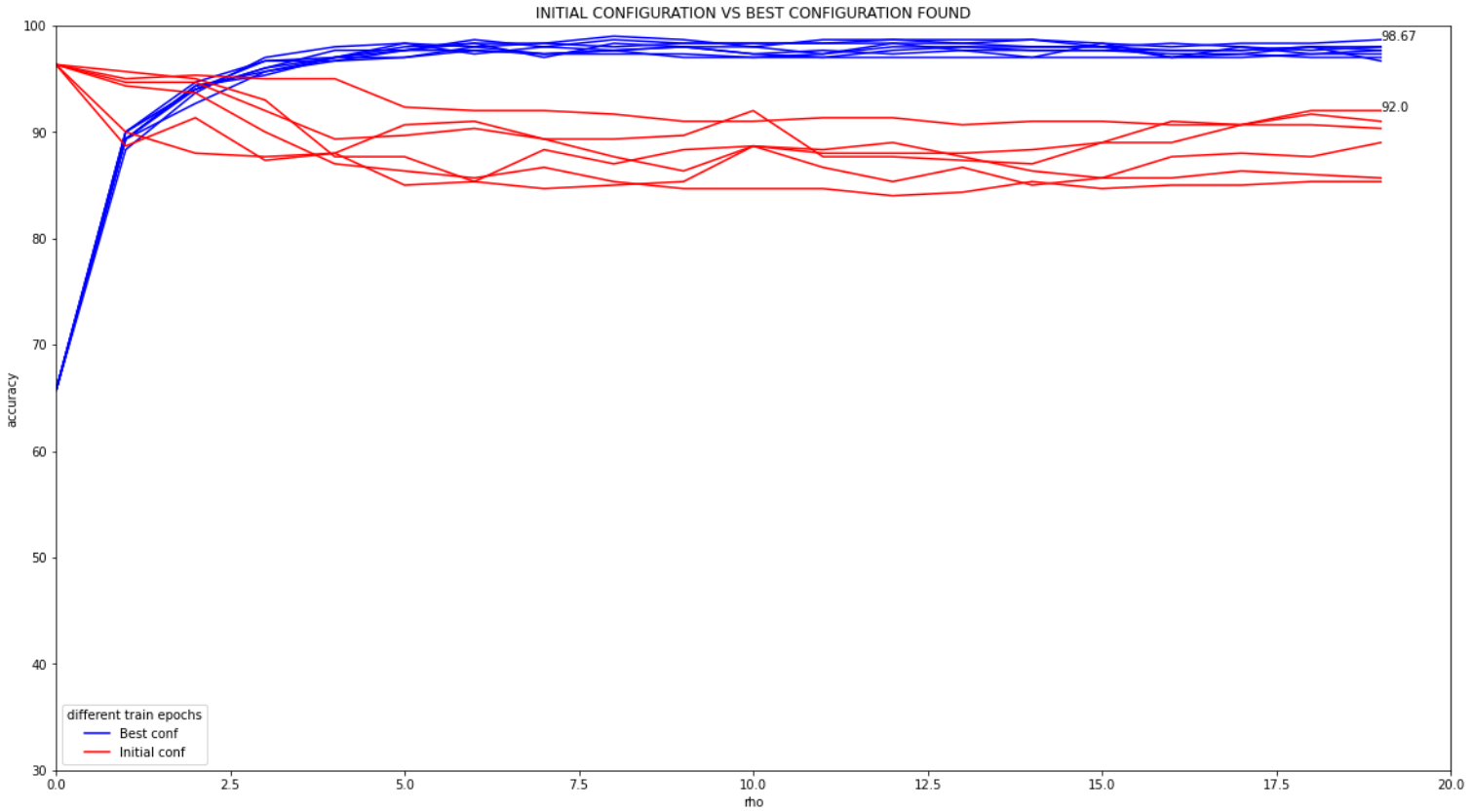


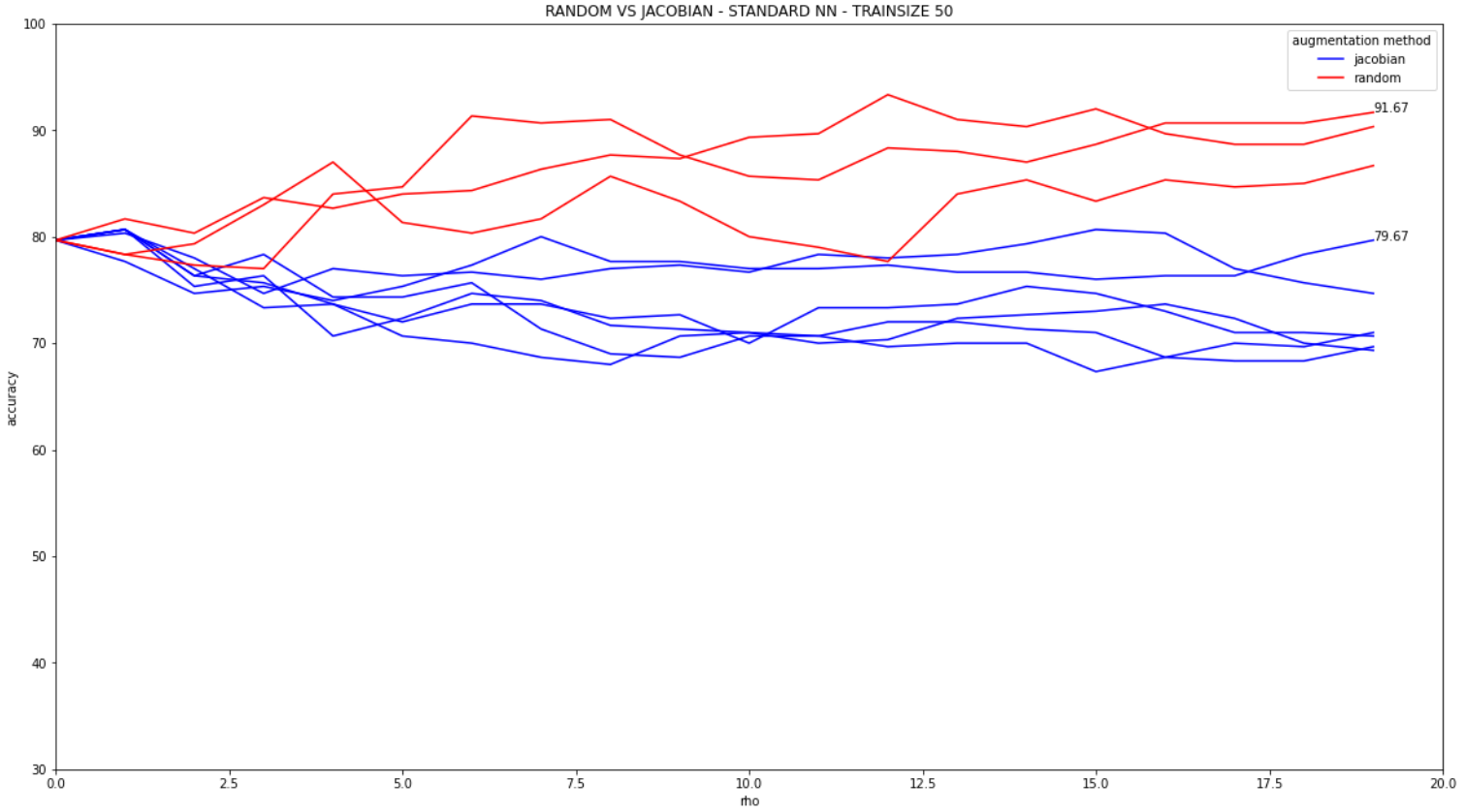
Fig.14: Comparison between initial configuration and the best configuration found. Initial train size: 50 - 190 - 310.

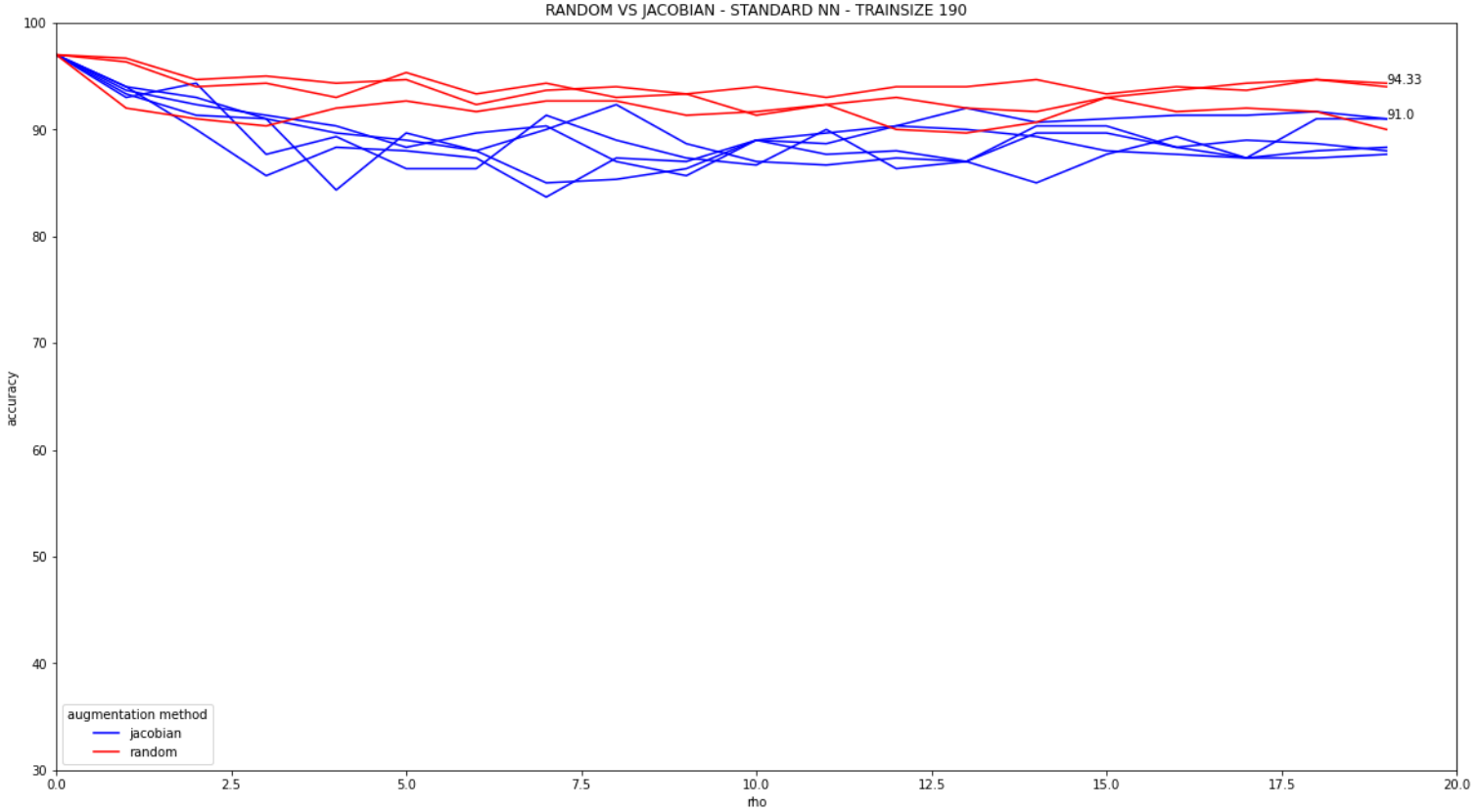
Test 6.1: Random augmentation vs Jacobian augmentation

After having optimized the system following the Jacobian Augmentation for crafting synthetic points, we want to compare this method against the Random one. Instead of generating points observing the gradient of the NN model, synthetic points are generated randomly. This test's aim is to evaluate the Jacobian method reliability and goodness.

In the first charts we show the accuracy evolution using the Jacobian method compared to the Random one, using the default NN model (2 hidden layers of 8 nodes each).
Systems configurations are the following:

TRAIN_SIZE	NN_ARC	NOE	LAMBDA	MAX_RHO	K	AUG. METHOD
50	8_8	800	0.5	20	40	jacobian / random
190	8_8	800	0.5	20	40	jacobian / random
310	8_8	800	0.5	20	40	jacobian / random


RANDOM VS JACOBIAN - STANDARD NN - TRAINSIZE 50
augmentation method
jacobian
random
91.67
79.67
accuracy
rho
100
90
80
70
60
50
40
30
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


RANDOM VS JACOBIAN - STANDARD NN - TRAINSIZE 190
augmentation method
jacobian
random
94.33
91.0
accuracy
rho
100
90
80
70
60
50
40
30
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

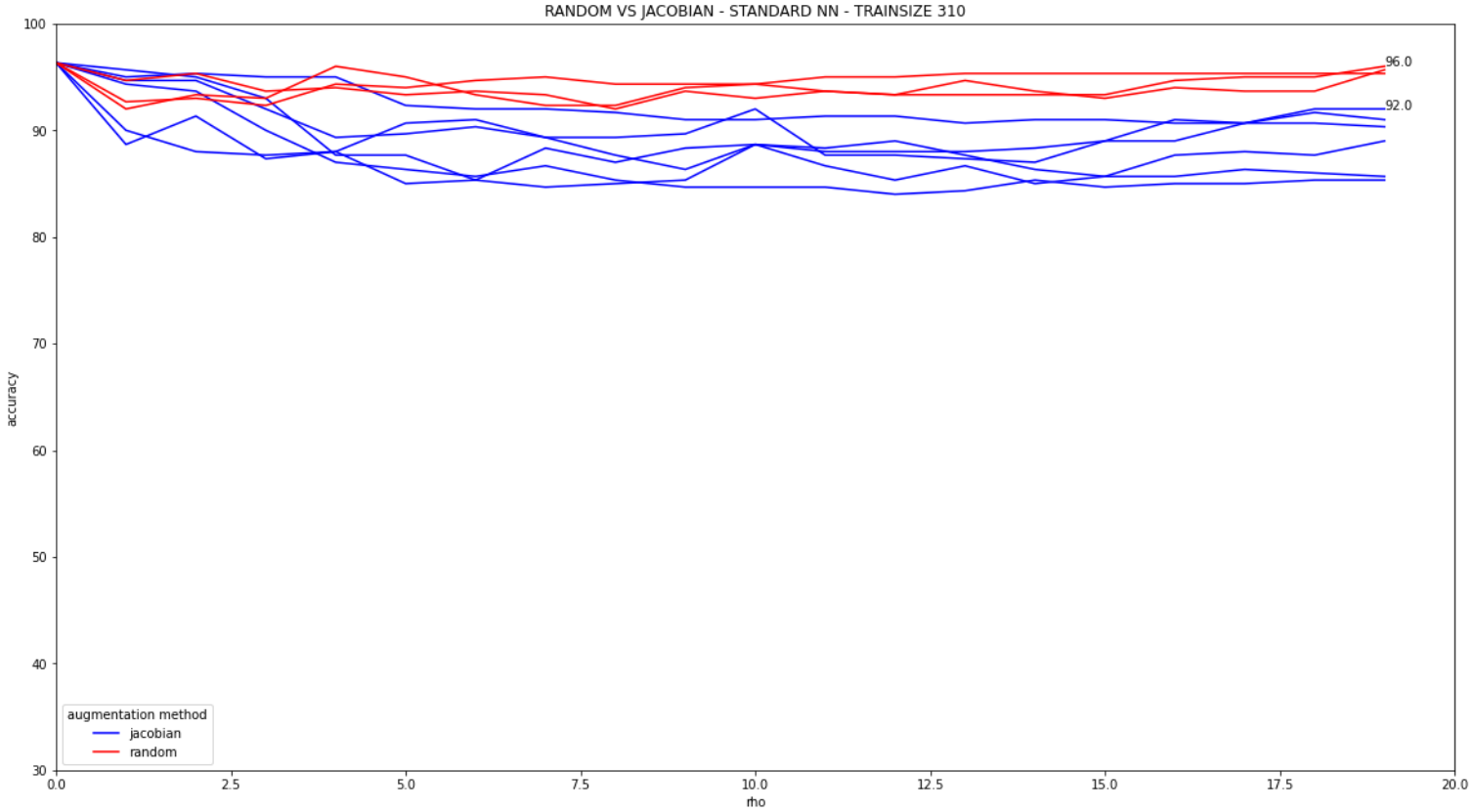


Fig.15: Comparison between Jacobian and Random augmentation (standard NN). Initial train size: 50 - 190 - 310.

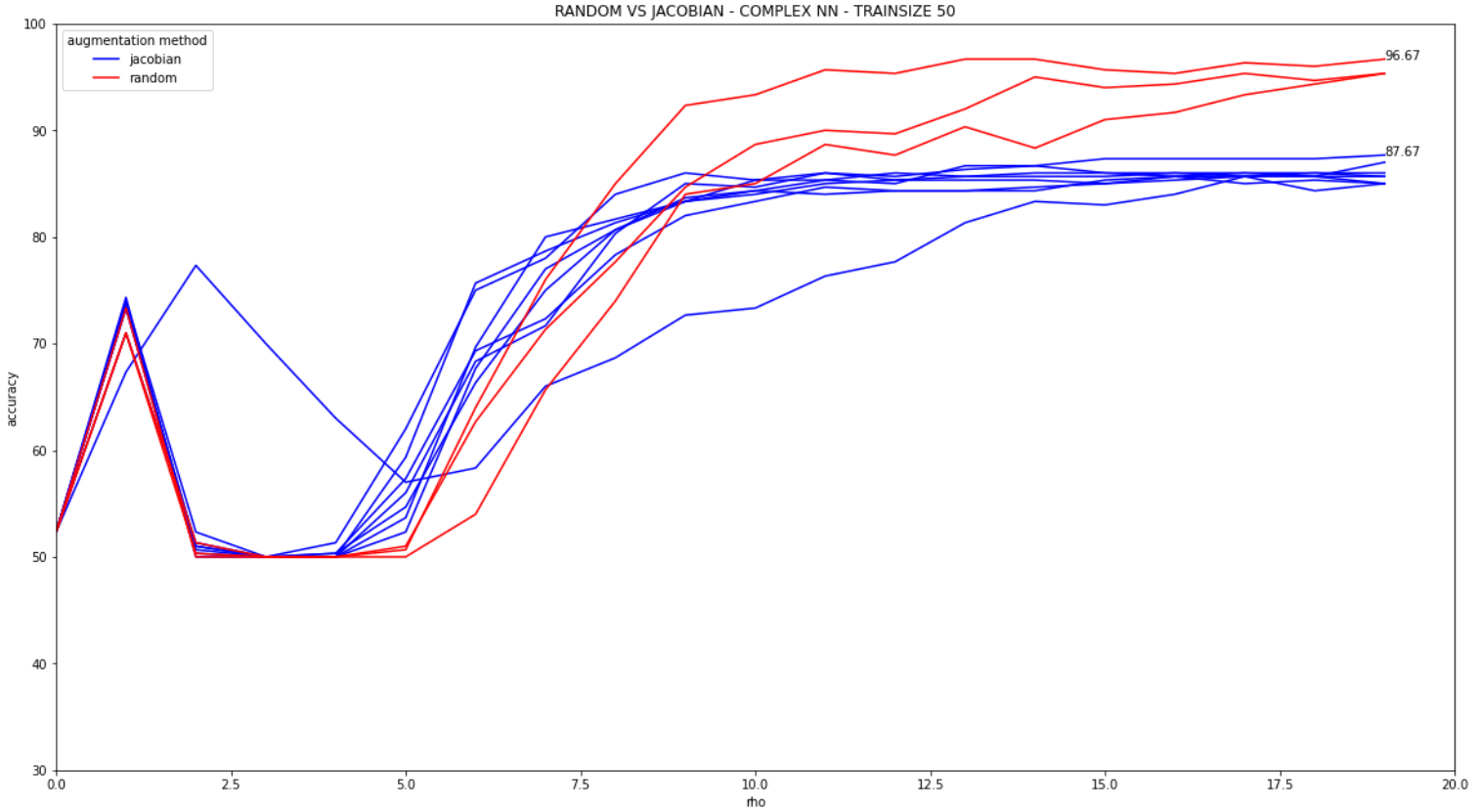
Observing the charts, with the default system configuration, the random method works better than the jacobian one.

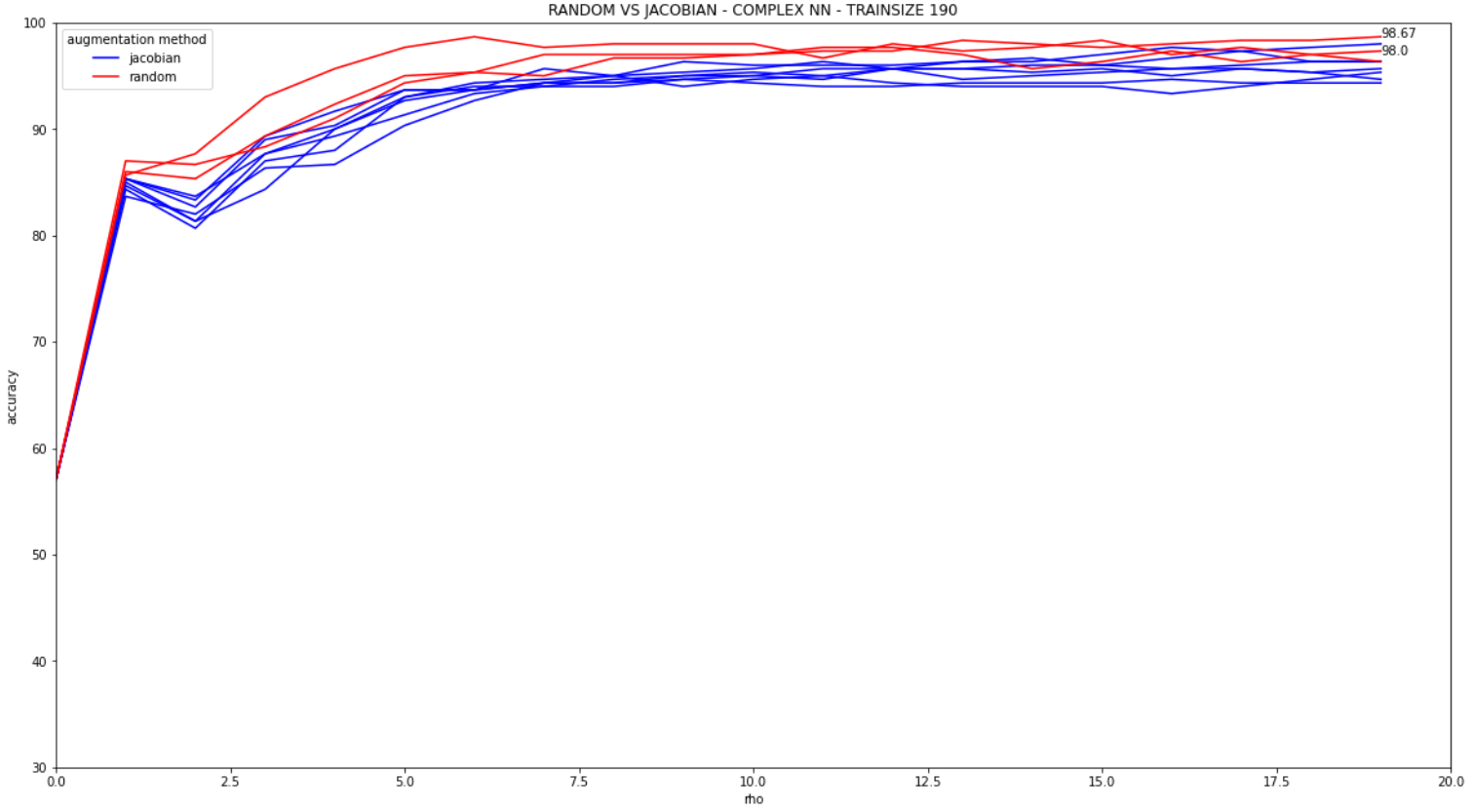
Now let's see if, with the complex NN, the random method behaves like with the standard NN.

Test 6.2: Rand vs Jacob with complex NN model

The last test in this report regards the comparison of the best system configuration using jacobian augmentation versus the same configuration using the random augmentation.
Systems configurations are the following:

TRAIN_SIZE	NN_ARC	NOE	LAMBDA	MAX_RHO	K	AUG. METHOD
50	64_32_32_32	5	0.5	20	40	jacobian / random
190	64_32_32_32	5	0.5	20	40	jacobian / random
310	64_32_32_32	5	0.5	20	40	jacobian / random


RANDOM VS JACOBIAN - COMPLEX NN - TRAINSIZE 50
augmentation method
jacobian
random
96.67
87.67
accuracy
rho
100
90
80
70
60
50
40
30
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


RANDOM VS JACOBIAN - COMPLEX NN - TRAINSIZE 190
augmentation method
jacobian
random
98.67
98.0
accuracy
rho
100
90
80
70
60
50
40
30
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

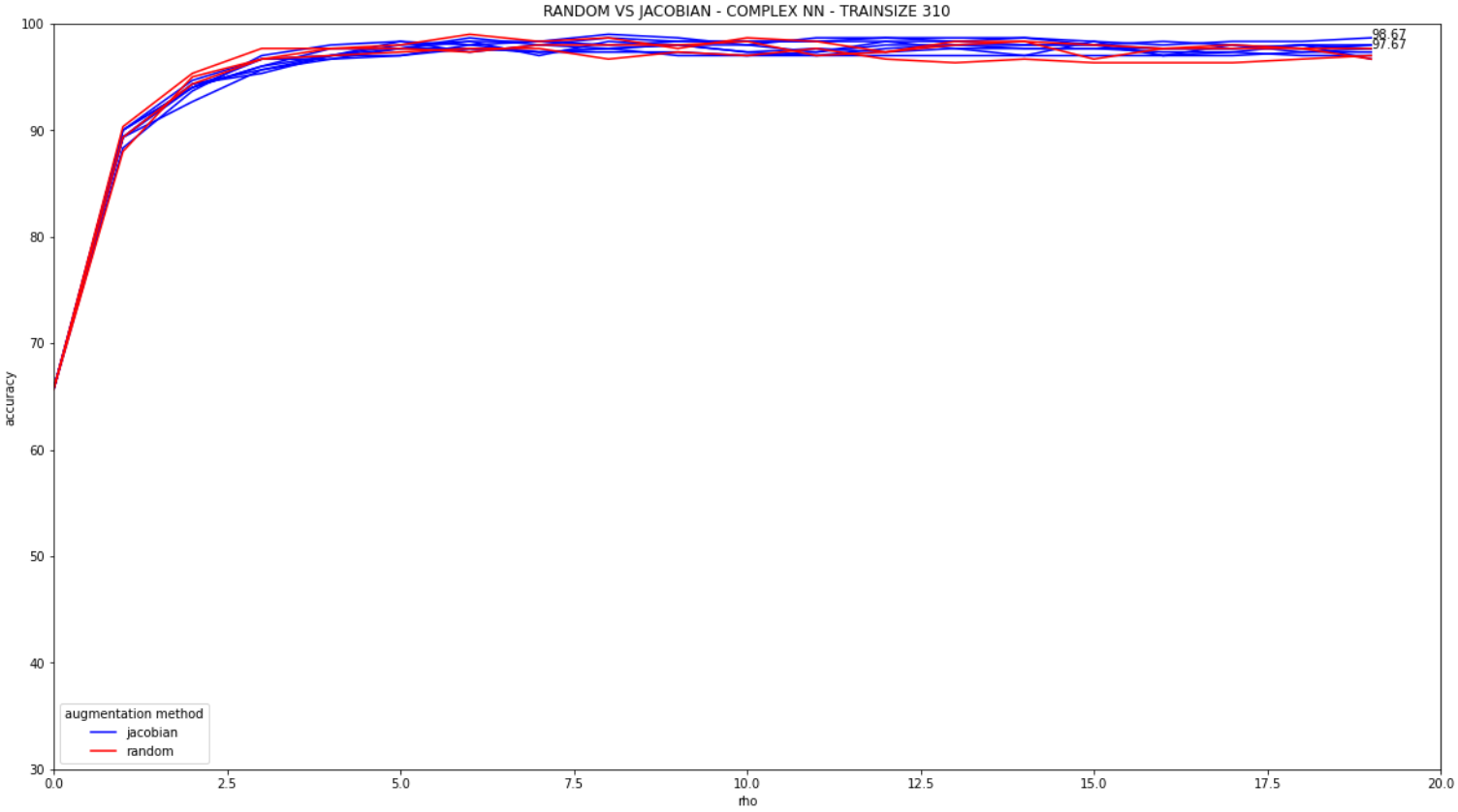


Fig.16: Comparison between Jacobian and Random augmentation (complex NN). Initial train size: 50 - 190 - 310.

The charts tell that the random augmentation works better than the jacobian one in every situation. The lower is the number of starting points, the higher is the difference in accuracy between the two systems.
We can observe that in the last case (310 starting points) the accuracy is quietly comparable. Having a low number of starting points and no training metrics the jacobian system is likely to shift points in a not useful manner causing the lower improving of accuracy.

Introducing Validation in training phase

We're pretty confident to say that overfitting is a problem of not improving performances using jacobian augmentation. A fact is that the lower are the number of training epochs, the higher are the system performances.

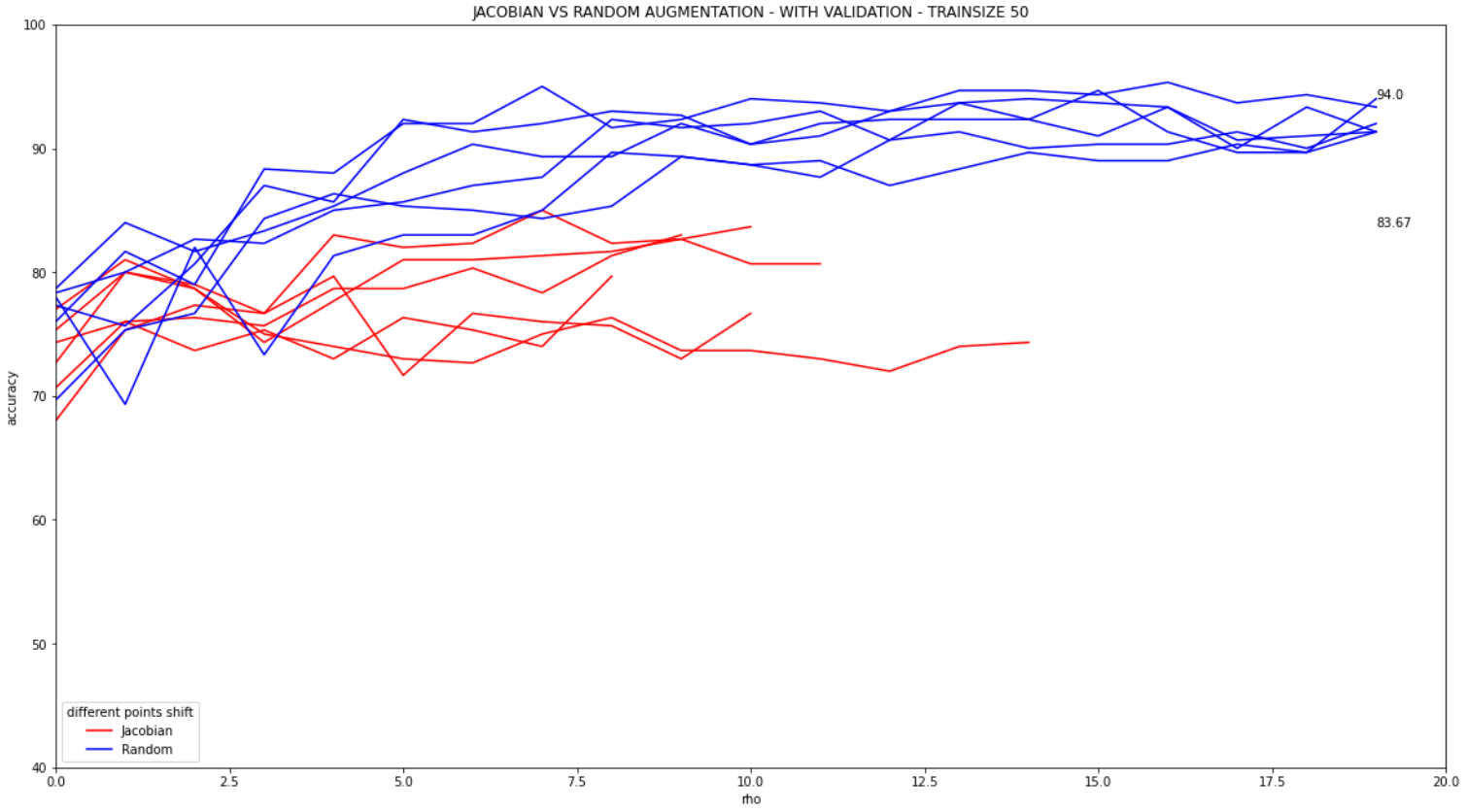
We now introduce the ***validation set*** in the training phase for guaranteeing a correct one. We set the training procedure to keep the best model during the train process, the one who has the ***lower validation loss.*** The train process max duration is set up to 800 epochs. Train-validation split is set to 70% on train-set.

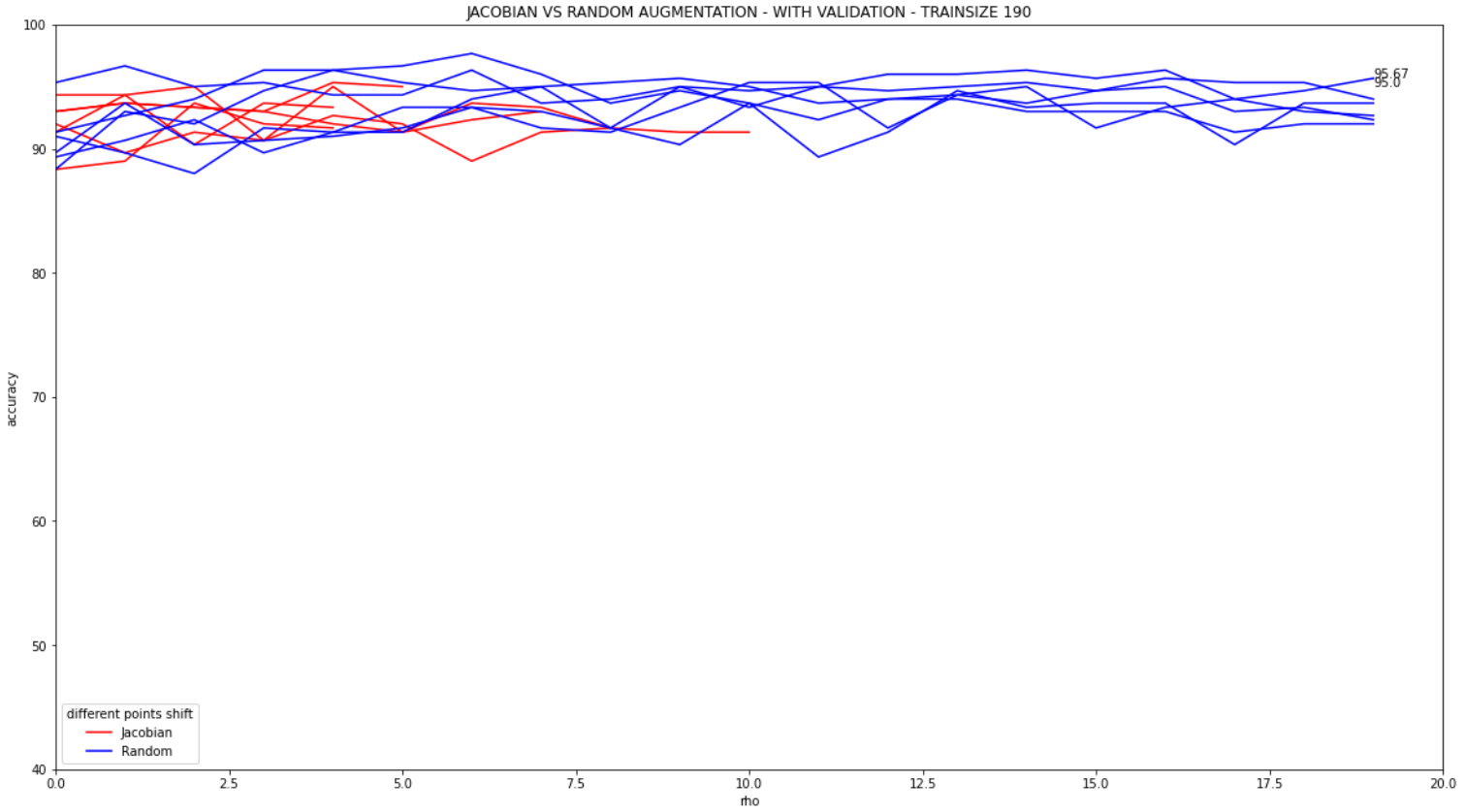
The introduction of the validation metrics should guarantee more precise gradient and thus more precise point shifting from the jacobian augmentation process.
Now all the right ingredients are set up: our prevision is to see observable performance improvements.

Test 7.1: Jacobian vs Random (with validation)

Let's take a look at the jacobian augmentation performances with the validation control available for the NN. We repeated the comparison test 6 times. Configurations are the following:

TRAIN_SIZE	NN_ARC	VAL_SPLIT	LAMBDA	MAX_RHO	K	AUG. METHOD
50	64_32_32_32	0.70	0.5	20	40	jacobian / random
190	64_32_32_32	0.70	0.5	20	40	jacobian / random
310	64_32_32_32	0.70	0.5	20	40	jacobian / random





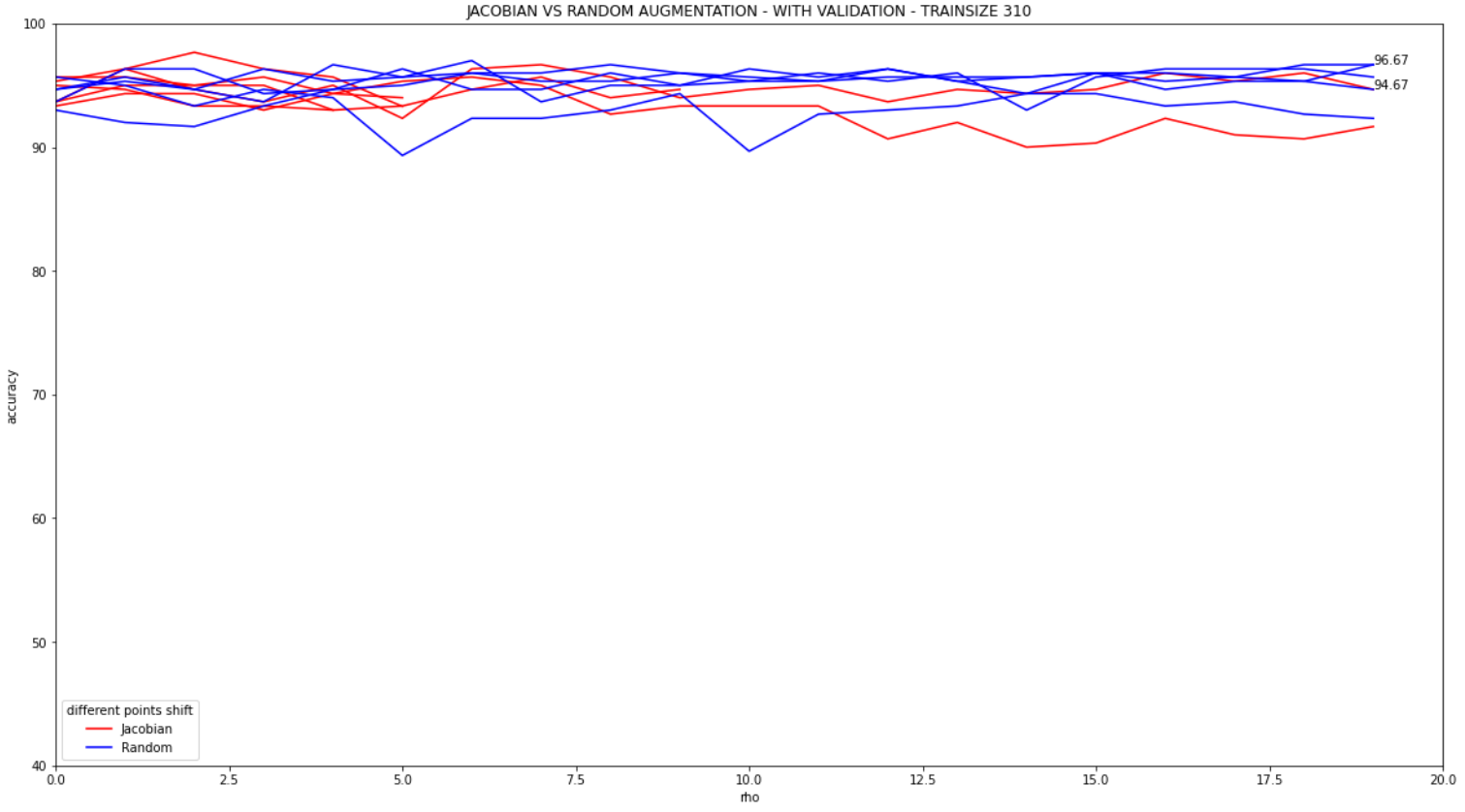
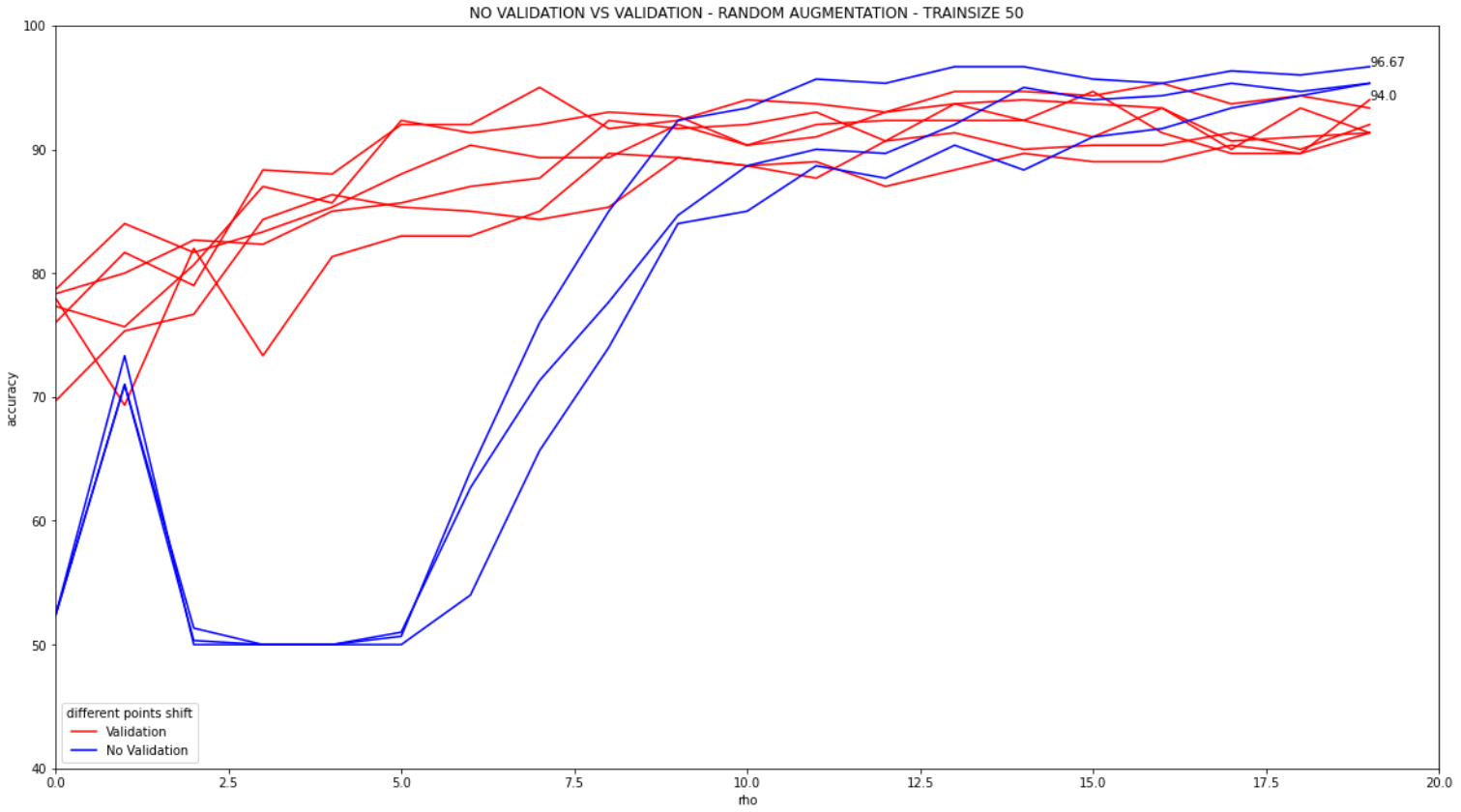


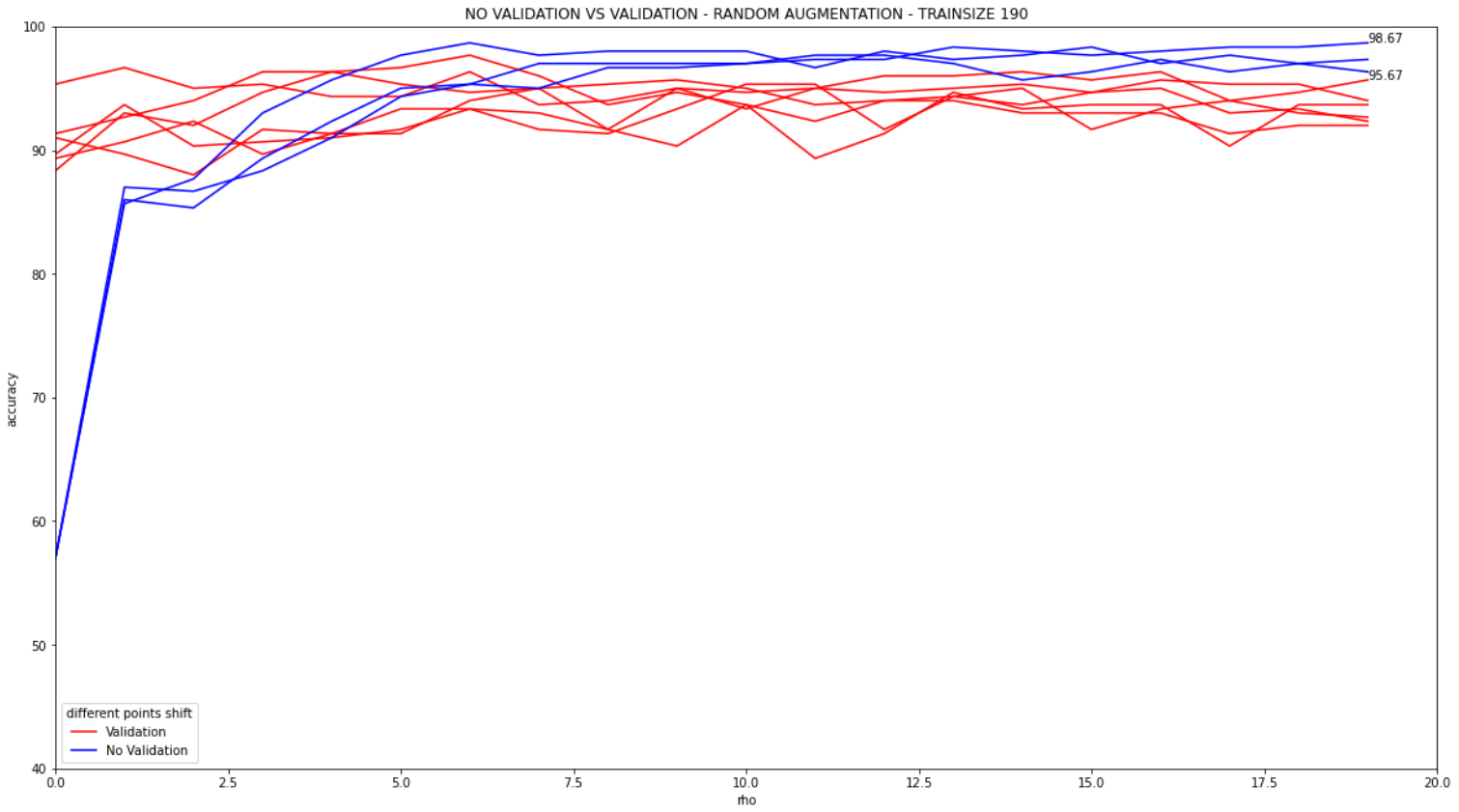
Fig.17: Comparison between Jacobian and Random augmentation (complex NN) with VALIDATION. Initial train size: 50 - 190 - 310.

The first thing to notice is that the introduction of validation causes an early stop for the system: the error occurs when a point is generated and one of its coordinates is too small causing it to be rounded to zero.
The second aspect is that, also with the validation, the jacobian system works worse than the random augmentation. Confirmed the aspect that more starting points guarantees a better final accuracy.

Test 7.2: Random aug. (validation) vs Random aug.

As we saw before the jacobian augmentation often occurs in errors without completing all the 20 iterations.
This problem is not observable following the random augmentation.
We now compare the random augmentation with the validation metric available in comparison with the random augmentation with no train metric controls.





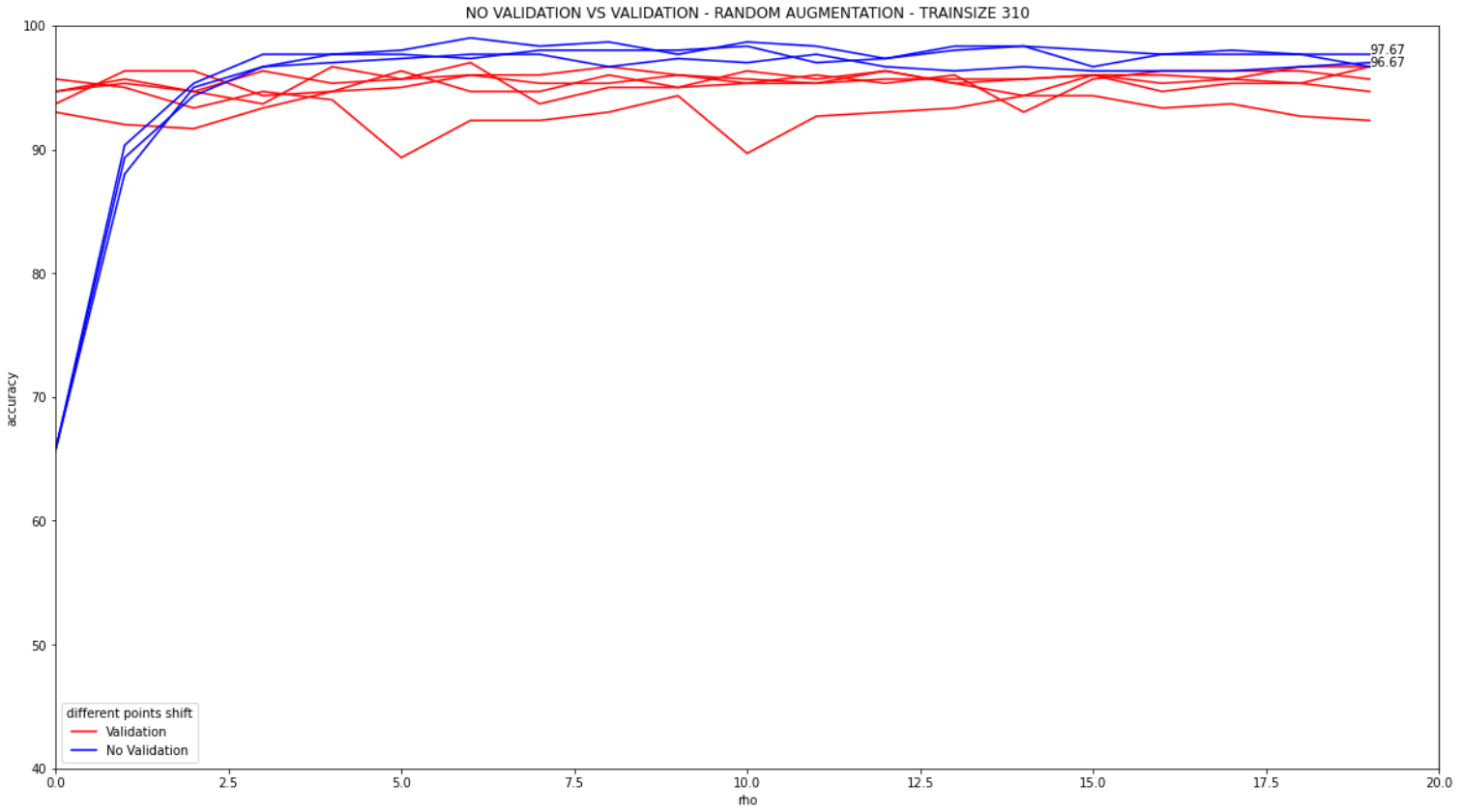


Fig.18: Comparison between Validation and No validation (complex NN with RANDOM augmentation). Initial train size: 50 - 190 - 310.

Final observations

We're now ending our report summing up with some observations regarding the Jacobian augmentation in comparison with the random one.

We observed from the charts that the Jacobian one has problems with useful points generation: following the NN gradient to shift points is not useful enough to increase a better explainability of the black-blox model. Following a random point generation rather than following the NN model gradient results to be more significant. The accuracy trend of random augmentation always covers the Jacobian one.

Secondly we introduce a **validation** set and newer training metrics. The NN model has available 800 epochs to be trained and, at the end of these iterations, the model with the lowest **validation_loss** is kept.

Results show that the introduction of validation is not enough for covering the performance divergence with the random augmentation.
Neither adding the validation to the random augmentation improves performance.