# Approximation a nonlinear function using a feedforward neural network, trained on noisy data.

## Approximated function

y = 0.5*cos(0.2*x.^2)+0.5;

---

## Through algorithm process

To the target function a random noise is added to simulate real-world imperfections. A neural network firstly with one and then with two hidden layers is trained to learn the pattern, and its predictions are compared to the original function to evaluate the quality of approximation.

```
ile_danych = 500;
ile_epok = 100;
a = 0;
b = 12;

x = a:0.05:b;
y = 0.5*cos(0.2.*x.^2)+0.5;

x_siec = a + (b-a)*rand(ile_danych,1);
y_siec = 0.5*cos(0.2.*x_siec.^2)+0.5 + 0.1*randn(ile_danych,1);

netconf = [12];
net = feedforwardnet(netconf);

net.trainParam.epochs = ile_epok;
% net.trainParam.max_fail = ile_epok;
net.trainParam.goal = 0;

net = train(net, x_siec', y_siec');

ypred = net(x);
```

---

## Analysis

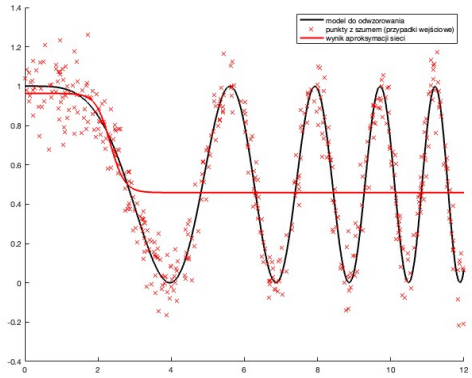
### Fixed training set and dynamic number of neurons and hidden layers (1/2 layers)

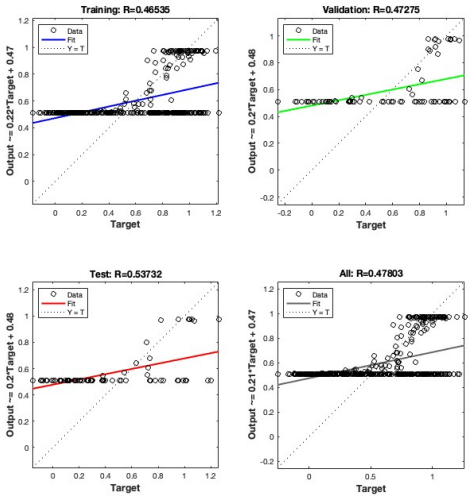
- number of training data = 500

## a) Test with the nurons count equals appropirately to proposed formula that count of nerons in hidden layer equals to square root of m multiplied by n, where m is count of input data (nuerons) and n is count of output data (nurons): k = sqrt( m * n )

- hidden layer = 1
- nuerons count in hidden layer = 1

| Training | Validation |
| --- | --- |



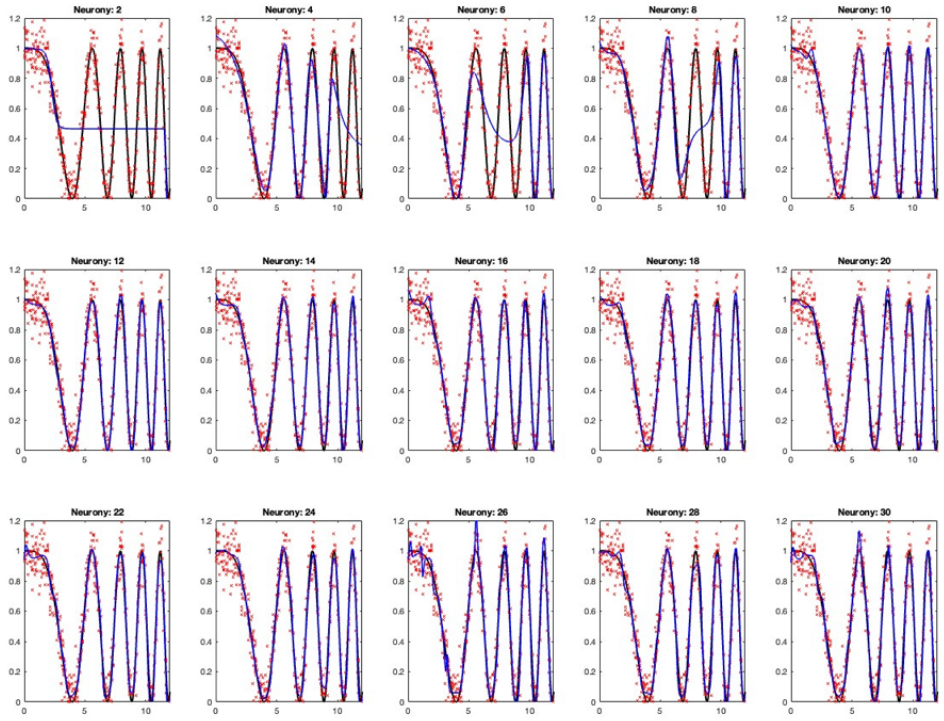



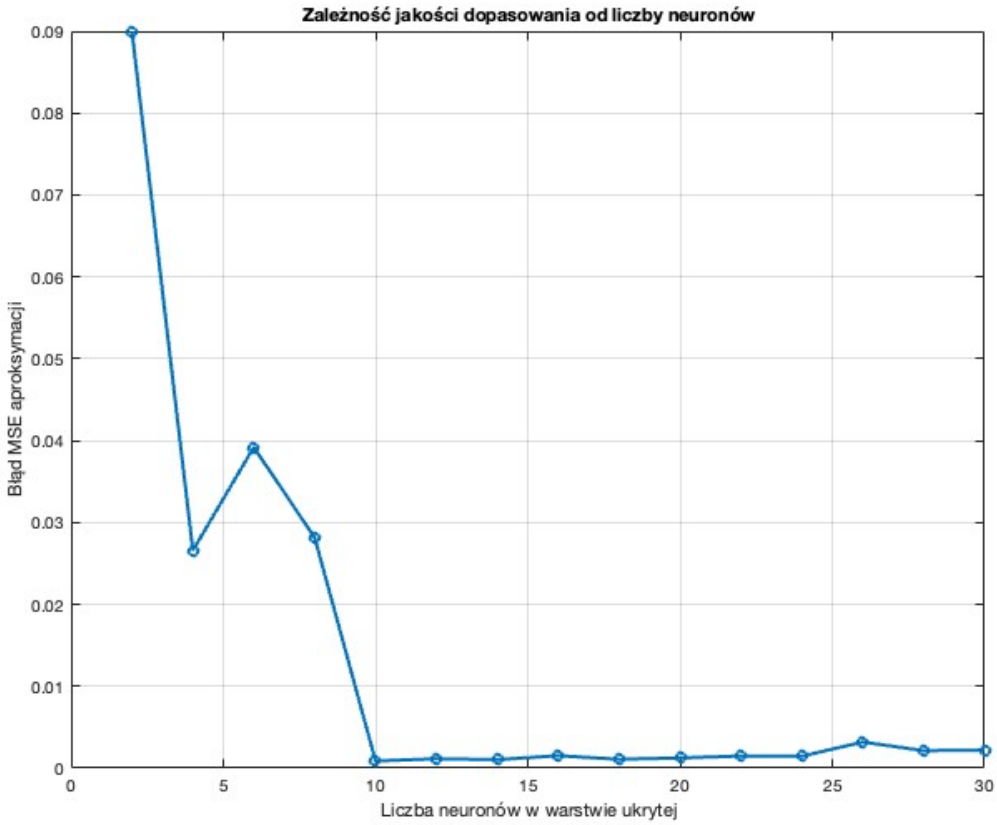

- mean square error: ~ 0.1 Summarizing, the above propozed formula about number of nurons in hidden layer is not good for not linear funcitons, line in our example.

The mean square error ispreety big in scale of our values thaht function can have, and also the regression plots show that model predict bad in every sets, so is not event good in training set of test set.
For values which starts from 0.5 and up, the model prediction is totaly bad.

## b) Test with various number of nurons in hidden layer

- hidden layer = 1
- nuerons count in hidden layer = from 2 to 30 by step 2




In range of small values for neurons count in hidden layer, the model is to general to our data points, so we can observer model underfitting.
In opposite situation, for range with bigger values of neurons count, the model is too

accurate for every single points od data, which makes its very sensitive for every noise is our data, and this is caled model overfitting.

Summarizing, abve plots figure of model predictions and the plot of MSE error clearly inlustrate the good range of neurons in hidden layer, to make model prediction appropraite to real values.

Count of min and max number od neurons in hidden layer to make good prediction:

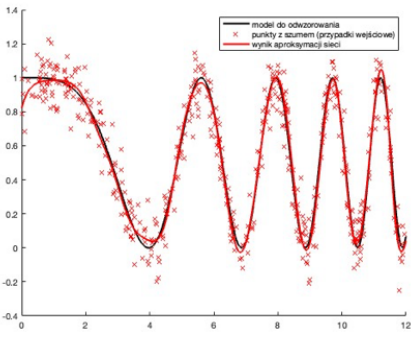
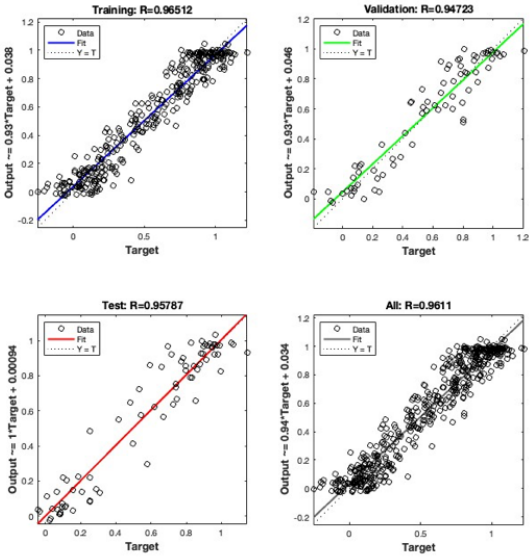
- min = ~ 10
- max = ~ 14

For this range of count of neurons in hidden layer, the MSE is pretty small ~ 0.001, so we can assume that this range in our case for single layer is appropraite to make model predictions satisfactory in compare with real values

### Pedictions for best neurons count in hidden layer

We can exactly see for what neurons number the model perform the best

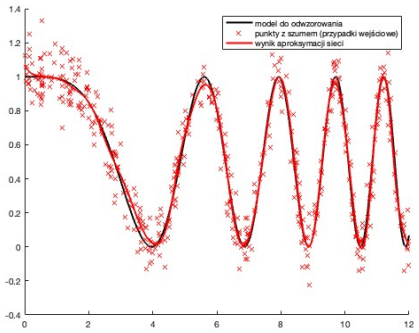
- 10 neurons
- MSE = ~ 0.009

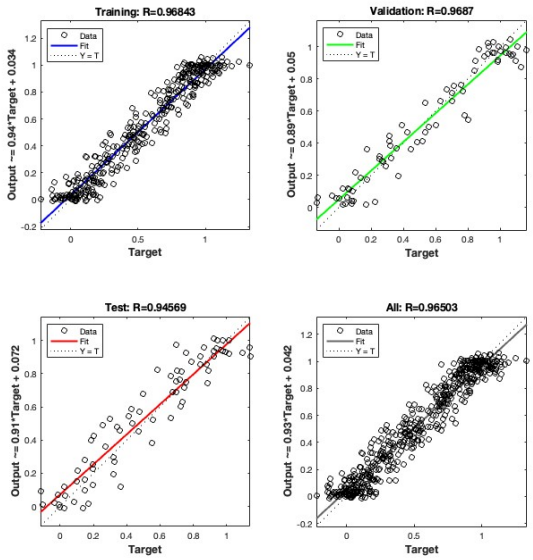
| Training | Validation |
| --- | --- |
|  |  |



- 12 neurons
- MSE = ~ 0.0002

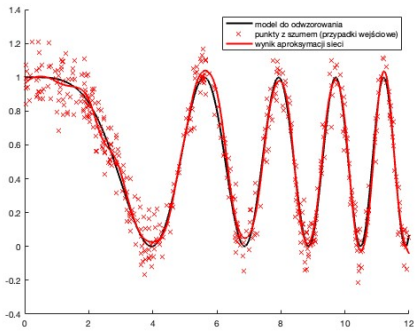
| Training | Validation |
| --- | --- |

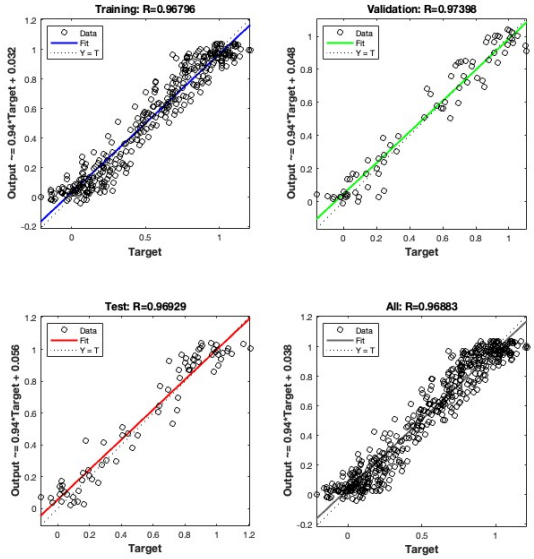



- 14 neurons
- MSE = ~ 0.0005

| Training | Validation |
|---|---|




Summarizing, from above possiblities of neruons (10,12,14) in single hiddne layer **the best one seems to be with 12 neurons**

We can doubt this choice, because the best Pearson's linear correlation coefficient (the highest) is with the 14 neurons in layer, but there the MSE error is much bigger than with 12 nurons. \

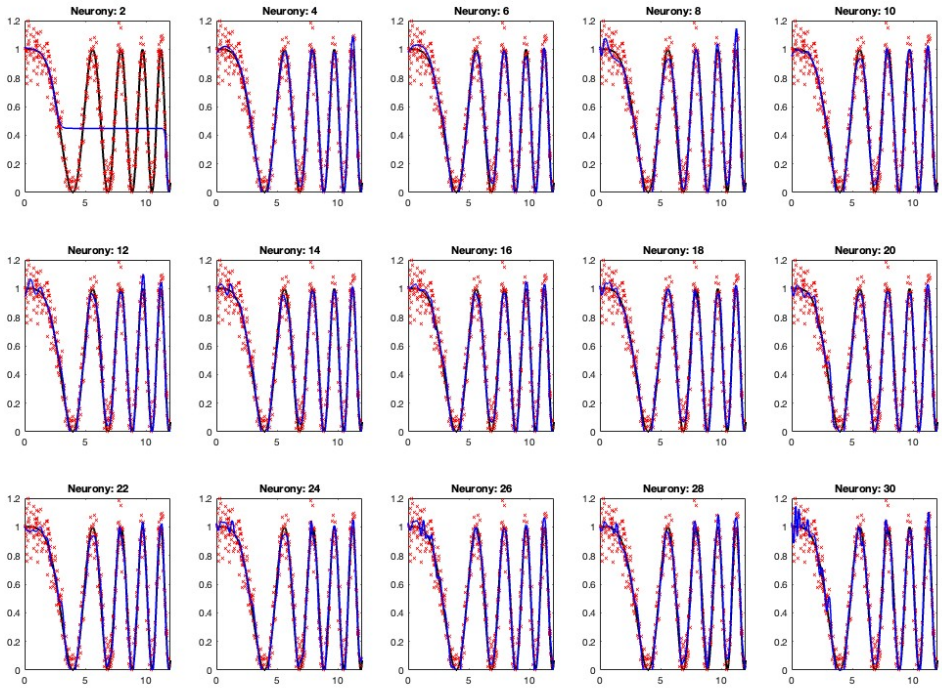
We prioritize the MSE Error value than R value, because the R value can be high and

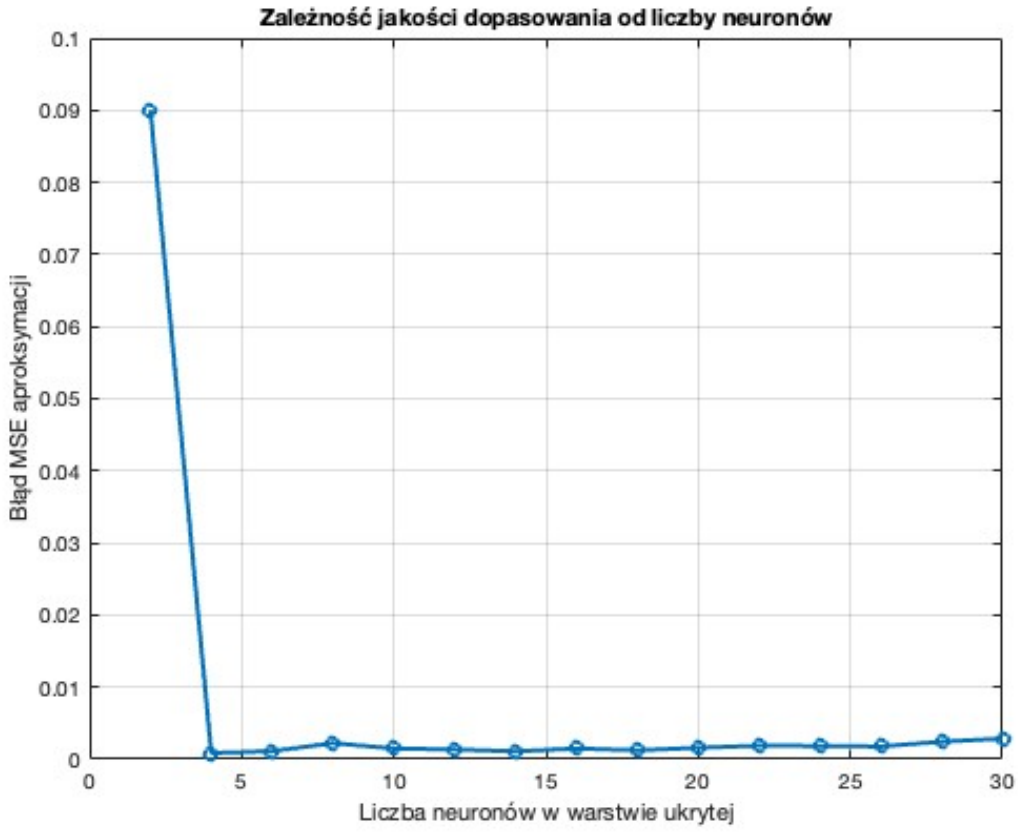
also the predict values can be far from the real ones, e.g. when the MSE error will be constant big, but also the tendancy about rising predicted values and rising in te same time real values will be maintained.

So with single hidden layer the best compromise between small and big amount of nuerons is the 12 neurons in hidden layer

## Can second layer addition improve the models predictions?

- hidden layer = 2
- nuerons count in hidden layer = from 2 to 30 by step 2

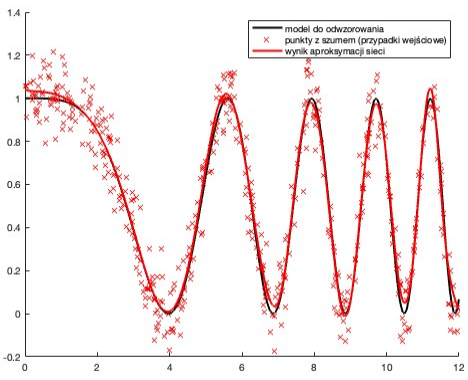
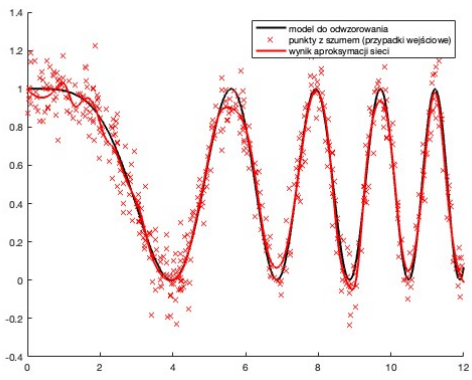





Summarizing, from above plots, the best seems to be ~ 5 and ~ 12 nuerons in hidden 2 layers, so lets check them

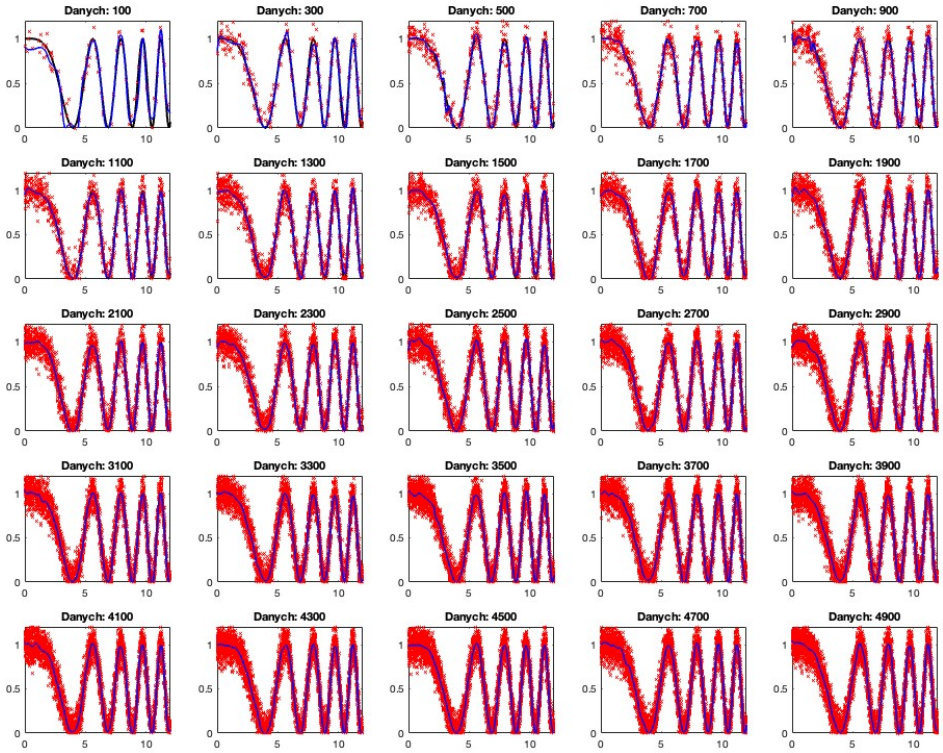
- 5 nerons, 2 layers : MSE = ~ 0.0006
- 12 neurons, 2 layers : MSE = ~ 0.002

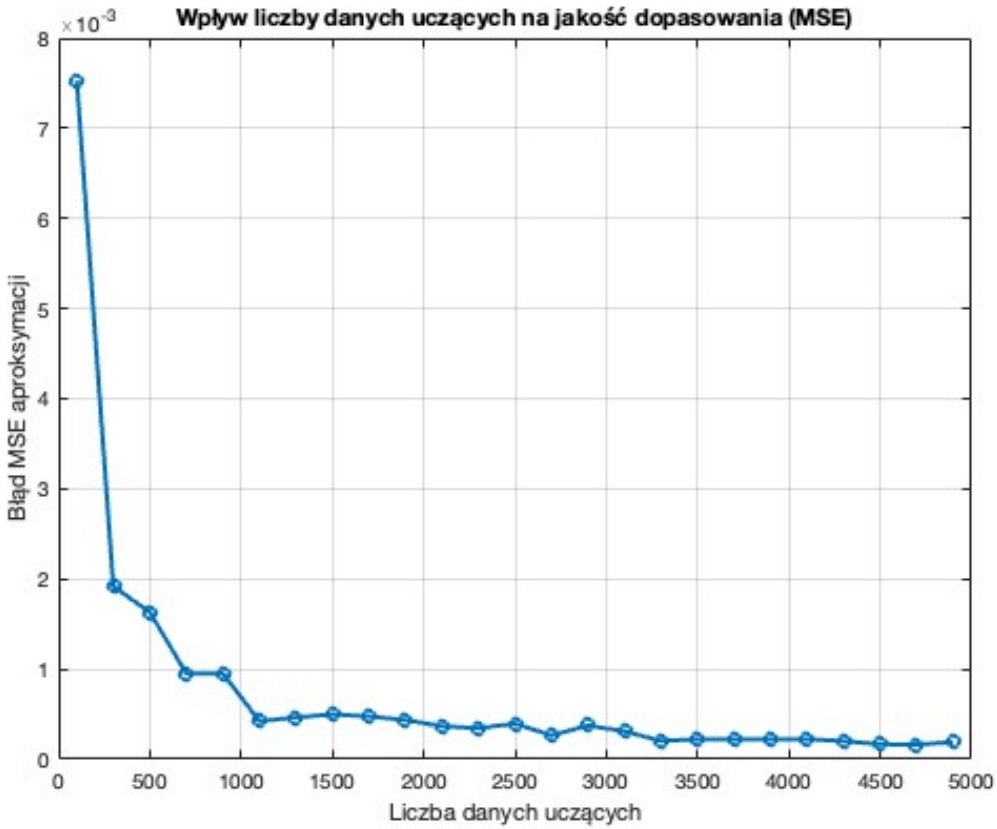
- 14 neurons
- MSE = ~ 0.0005

| Training | Training |
| --- | --- |
|  |  |

Summarizing, the second layer addition is not valuable for our no linear funciton value prediction, the best is 5 nerons for 2 layers, bit still it has worse results in MSE, so the MSE is bigger than in single layer.

## For the optimized number of neurons, lets manipulate the size of the training set.






So as we can see from above plots, the general tendence is that when the more points (input data) is, then the model predict better and better. \

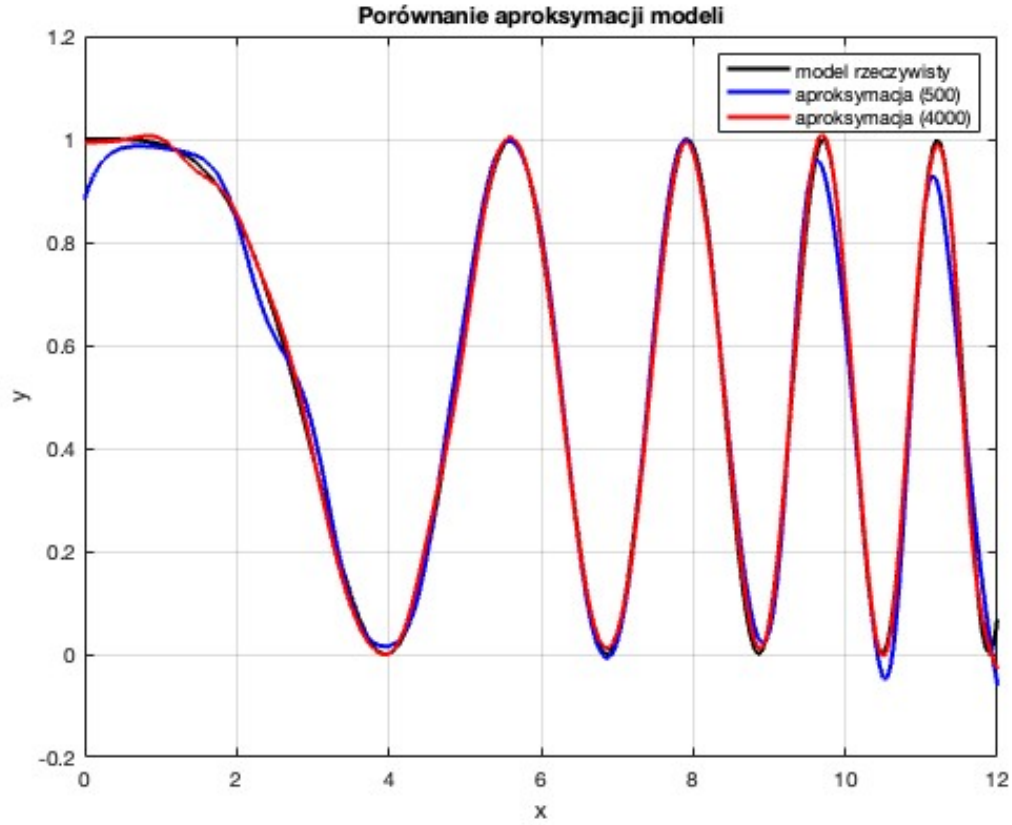
We can also see that we tendency line from about 1000 input data is const, especially from about 3500 counts of input data the model prediction is on the same good level. \

So in comparisson, lets choose 4000 input data and analyse this case

**Consequances of low number of data inputs**

These effects are also clearly visible in the case with a smaller number of samples ~ 100. The model tends to overfit the noisy data, shows poor generalization, and its predictions are less stable and less accurate due to the limited representativeness of the training set. So in case of low count of data inputs, there is model overfitting.

- input data = 4000
- layers = 1
- nerons in hidden layer = 12



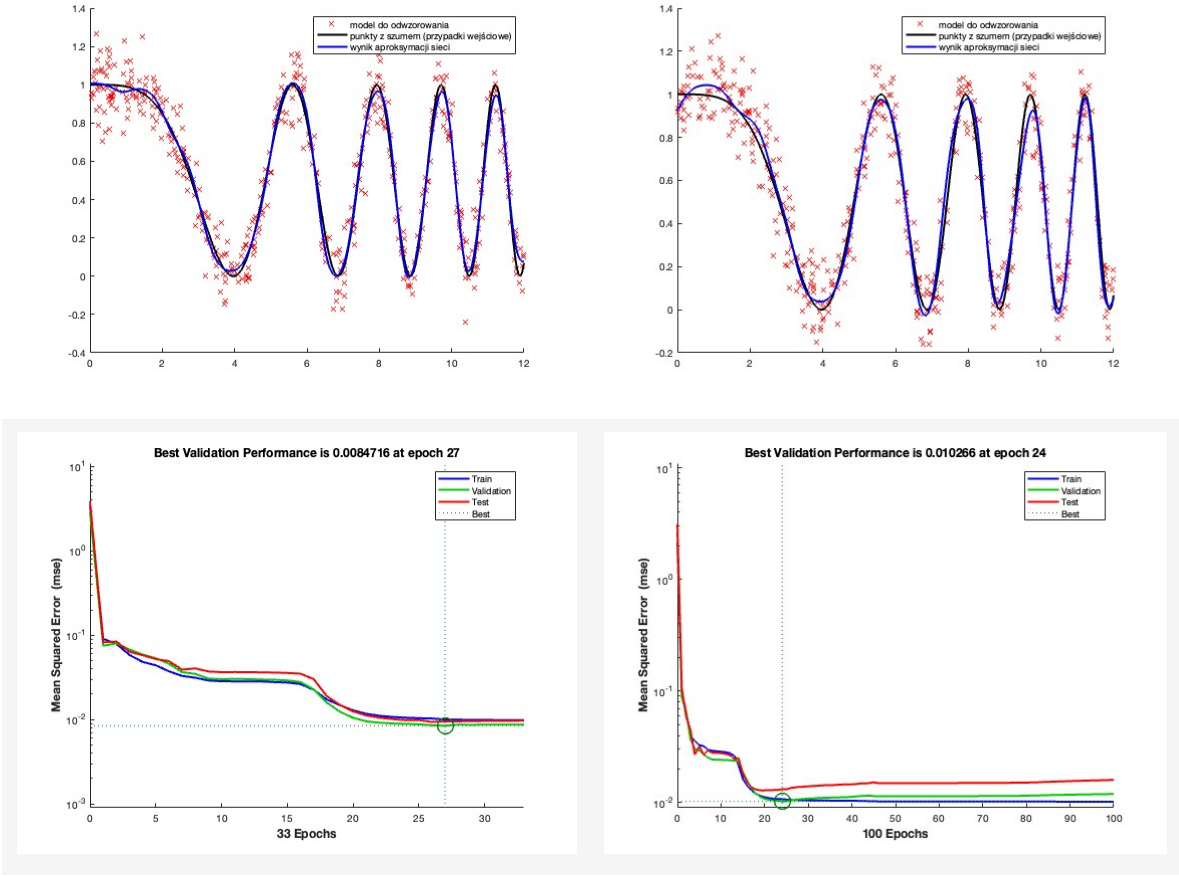

Model 1 (500 próbek): MSE = 0.00142 MAE = 0.02666 RMSE = 0.03774 $R^2$ = 0.98934

Model 2 (4000 próbek): MSE = 0.00020 MAE = 0.01020 RMSE = 0.01403 $R^2$ = 0.99853

Above example and metrics confirm our earlier prediction that with increasing input data, the model is able to make better predictions. This improvement is especially visible in metrics like $R^2$ and MAE, which reflect the model's ability to generalize well to the true underlying function, not just noisy training points. More input data helps the model avoid overfitting by providing a more diverse and representative sample of the target function. As a result, the model learns the actual patterns rather than memorizing noise or outliers. However, it’s also important to balance data size with model complexity — in very large datasets, overly complex models may still overfit noise if regularization or early stopping is not applied.

## Evaluation of the impact of training duration on the effect of overfitting

| Limited Training (up to 6 epoch since no improvment) | Not limited Training |
|---|---|




The two approaches illustrate how early stopping (limiting training when no improvement is observed) helps in preventing overfitting.

- Limited Training (Early Stopping):
  In this case, the training process stops automatically after 6 epochs without validation improvement. This helps the model generalize better, as seen in the smoother approximation curve and lower validation error (best at epoch 27, MSE ≈ 0.00847). The model does not try to fit the noise in the data, maintaining a good balance between bias and variance.
- Unrestricted Training:
  Training continues for all 100 epochs, regardless of validation performance. As shown in the graph, the model starts to overfit: while training error continues to decrease, validation and test errors start to increase after a point (best validation performance was actually earlier, at epoch 24 with MSE ≈ 0.01026). The approximation curve shows more unnecessary oscillations, indicating that the model has begun fitting the noise in the training data.

Conclusion:
Early stopping is an effective regularization technique to reduce overfitting. It halts training when further improvements on the validation set cease, ensuring the model retains generalization capability without fitting irrelevant patterns or noise.