Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

Отчёт

по лабораторной работе № 2
по дисциплине «Интеллектуальные технологии информационной безопасности»

Тема: «Применение однослойной нейронной сети с линейной функцией активации для прогнозирования временных рядов»

Вариант 4

Выполнил: Григорьев Е.Г.,
студент группы ИУ8-63

Проверил: Строганов И.С.,
преподаватель каф. ИУ8

г. Москва,
2022 г.

1. Цель работы

Изучить возможности однослойных НС в задачах прогнозирования временных рядов методом скользящего окна (авторегрессия)

2. Условия

Условия согласно варианту 3:

Функция:

$$x(t) = 0.5 \times \exp(0.5 \times \cos(0.5 \times t)) + \sin(0.5 \times t) \quad t \in [-5, 3]$$

Норма обучения $\eta = 0.115$

3. Ход работы

Рассмотрим прогноз функции из условия варианта по 20 равноотстоящим исходным значениям x. Размер окна будет равным 4ем, норма обучения $\eta = 0{,}01$. Вес смещения $w_0 = 0$

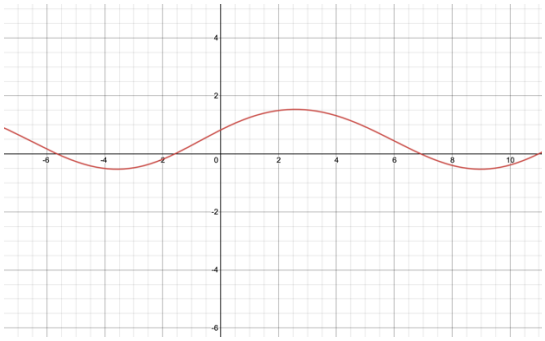

Рисунок 1 – Исходная функция: – $X(t)$

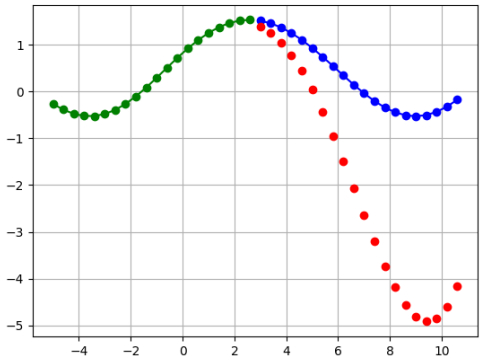

Рисунок 2 – Исходная функция и ее прогноз при M = 10, ($\varepsilon = 0.225$): — — $X(t)$, ∘∘∘ x

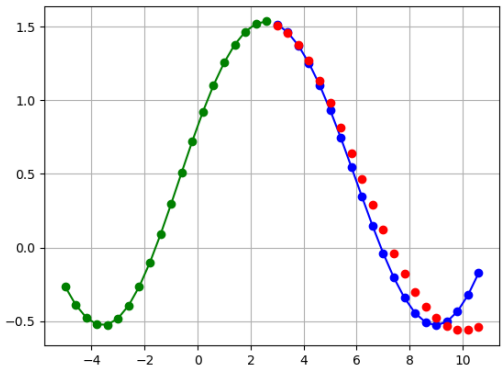

Рисунок 3 – Исходная функция и ее прогноз при M = 465, ($\varepsilon = 0.035$): — — $X(t)$, ∘∘∘ x

Вектор весовых коэффициентов при M = 465 равен

$w = [0, 0.305, -0.034, -0.248, -0.318, -0.231, 0.024, 0.450, 1.039,]$

Сравнивая результаты прогноза при различном количестве эпох, следует отметить, что его качество неудовлетворительно примерно до M = 200, а затем быстро улучшается, и при M = 465 прогнозные значения приблизительно совпадают с точными в пределах графического изображения, а затем при M > 500, нейронная сеть переобучается.

Для проверки корректности выбранных значений были построены графики зависимости среднеквадратичной ошибки от нормы обучения, размера окна и числа эпох.

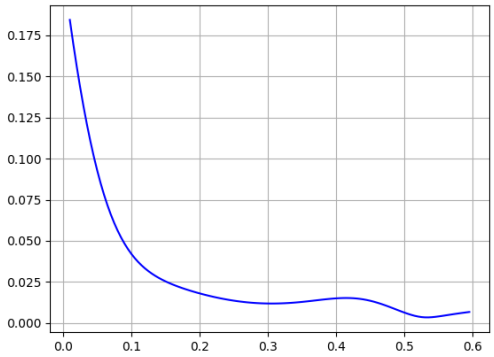

Рисунок 4 – Зависимость среднеквадратичной ошибки от нормы обучения (при числе эпох = 465 и размере окна = 8)

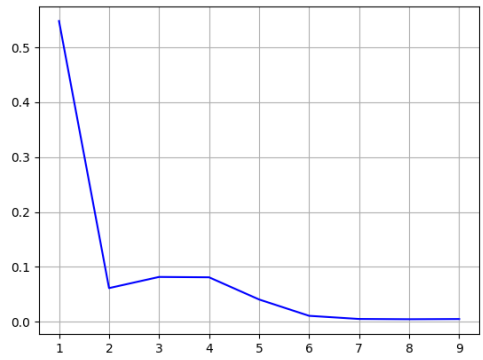

Рисунок 4 – Зависимость среднеквадратичной ошибки от размера окна (при числе эпох = 465 и норме обучения = 0.115)

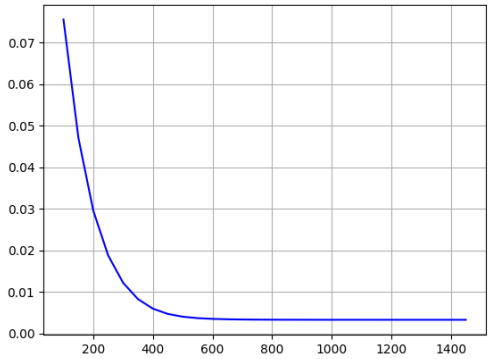

Рисунок 4 – Зависимость среднеквадратичной ошибки от числа эпох (при размере окна = 8 и норме обучения = 0.115)

4. Выводы

В ходе выполнения лабораторной работы, мною было исследованно функционирование простейшей нейронной сети (НС) на базе нейрона с нелинейной функцией активации. Также она была обучена по правилу Видроу-Хоффа. Результаты совпали с ожидаемыми, что говорит о корректности работы программы.

Приложение А. Исходный код программы

Файл main.go

```
package main

import (
   "errors"
   "fmt"
   "math"
   "strings"
)

func getStringsFormat(m []float64) string {
   strF := make([]string, 0, len(m))
   for _, n := range m {
      strF = append(strF, fmt.Sprintf("%f", n))
   }
   return strings.Join(strF, ", ")
}

func NewNeuron(M int, a, b, eta float64) *Neuron {
   windowSize := int(math.Abs(a) + math.Abs(b))
   n := 20
   epochs := make([]int, 0, M)
   epsilons := make([]float64, 0, M)
   return &Neuron{
      epochsNumber: M,
      a:            a,
      b:            b,
      eta:          eta,
      n:            n,
      windowSize:   windowSize,
      epochs:       epochs,
      epsilons:     epsilons,
   }
}

type Neuron struct {
   epochsNumber  int
   a             float64
   b             float64
   eta           float64
   learnFunction []float64
   mainFunction  []float64
   n             int
   windowSize    int
   Weights       []float64
   Epsilon       float64
   delta         float64
   epochs        []int
   epsilons      []float64
}

func standardError(xReal, xPred []float64) (float64, error) {
```

```
    if len(xReal) != len(xPred) {
        return 0., errors.New("dimensions of the vectors are not equal")
    }
    err := 0.0
    for i := 0; i < len(xReal); i++ {
        err += math.Pow(xReal[i]-xPred[i], 2)
    }
    err = math.Sqrt(err)
    return err, nil
}
func targetFunction(t float64) float64 {
    return 0.5*math.Exp(0.5*math.Cos(0.5*t)) + math.Sin(0.5*t)
}
func (n Neuron) net(x []float64) float64 {
    var net float64
    for i := 0; i < n.windowSize; i++ {
        net += n.Weights[i+1] * x[i]
    }
    return net
}

func (n Neuron) getT(a, b float64) []float64 {
    vectorT := make([]float64, 0, n.n)
    dt := (b - a) / float64(n.n)
    for i := 0; i < n.n; i++ {
        vectorT = append(vectorT, a+float64(i)*dt)
    }
    return vectorT
}

func (n Neuron) getX(a, b float64) []float64 {
    vectorX := make([]float64, 0, n.n)
    vectorT := n.getT(a, b)
    for _, t := range vectorT {
        vectorX = append(vectorX, targetFunction(t))
    }
    return vectorX
}

func (n *Neuron) TrainingMode() {
    n.learnFunction = make([]float64, n.n)
    n.mainFunction = n.getX(n.a, n.b)
    n.Weights = make([]float64, n.windowSize+1)
    for k := 0; k < n.epochsNumber; k++ {
        for q := 0; q < n.windowSize; q++ {
            n.learnFunction[q] = n.mainFunction[q]
        }
        for i := n.windowSize; i < n.n; i++ {
            n.learnFunction[i] = n.net(n.mainFunction[i-n.windowSize : i])
            n.delta = n.mainFunction[i] - n.learnFunction[i]

            for j := 0; j < n.windowSize; j++ {
                n.Weights[j+1] += n.eta * n.delta * n.mainFunction[i-n.windowSize+j]
            }
        }
        n.Epsilon, _ = standardError(n.mainFunction, n.learnFunction)
        n.epochs = append(n.epochs, k)
        n.epsilons = append(n.epsilons, n.Epsilon)

    }
}
```

```
func (n *Neuron) WorkingMode() {
   vecFT := n.getT(n.a, n.b)
   vecFX := n.getX(n.a, n.b)
   vectorT := n.getT(n.b, 2*n.b-n.a)
   vectorX := n.getX(n.b, 2*n.b-n.a)
   testFunction := make([]float64, 28)
   tF := make([]float64, n.n)
   for i := n.n - n.windowSize; i < n.n; i++ {
      testFunction[i-(n.n-n.windowSize)] = n.learnFunction[i]
   }
   for j := n.windowSize; j < len(vectorX)+n.windowSize; j++ {
      vectorTestFunction := make([]float64, n.windowSize)
      for k := 0; k < n.windowSize; k++ {
         vectorTestFunction[k] = testFunction[k+j-n.windowSize]
      }
      testFunction[j] = n.net(vectorTestFunction)
   }
   for i := n.windowSize; i < len(testFunction); i++ {
      tF[i-n.windowSize] = testFunction[i]
   }
   fmt.Println("vecFT:", getStringsFormat(vecFT))
   fmt.Println("vecFX:", getStringsFormat(vecFX))
   fmt.Println("vectorT:", getStringsFormat(vectorT))
   fmt.Println("vectorX:", getStringsFormat(vectorX))
   fmt.Println("tF:", getStringsFormat(tF))
}

func main() {
   obj := NewNeuron(10, -5, 3, 0.115)
   obj.TrainingMode()
   obj.WorkingMode()
   fmt.Println(obj.Epsilon)
   obj = NewNeuron(465, -5, 3, 0.115)
   obj.TrainingMode()
   obj.WorkingMode()
   fmt.Println(obj.Epsilon)
   fmt.Println(obj.Weights)
}
```

Файл main.py

```
from matplotlib import pyplot as plt


def plotting(vecFT, vecFX, vectorT, vectorX, TF):
    plt.plot(vecFT, vecFX, 'go-')
    plt.plot(vectorT, vectorX, 'bo-')
    plt.plot(vectorT, TF, 'ro')

    plt.grid(True)
    plt.show()


if __name__ == '__main__':
    plotting([-5.000000, -4.600000, -4.200000, -3.800000, -3.400000, -3.000000,
-2.600000, -2.200000, -1.800000,
              -1.400000, -1.000000, -0.600000, -0.200000, 0.200000, 0.600000,
1.000000, 1.400000, 1.800000, 2.200000,
              2.600000
              ], [-0.263504, -0.387370, -0.474751, -0.520928, -0.522860, -
0.479494, -0.392005, -0.263919, -0.101065,
                  0.088697, 0.295990, 0.510635, 0.722471, 0.922137, 1.101675,
```

```
1.254841, 1.377132, 1.465588, 1.518496,
                 1.535111
                 ],
             [3.000000, 3.400000, 3.800000, 4.200000, 4.600000, 5.000000,
5.400000, 5.800000, 6.200000, 6.600000,
              7.000000, 7.400000, 7.800000, 8.200000, 8.600000, 9.000000,
9.400000, 9.800000, 10.200000, 10.600000
              ], [1.515496, 1.460469, 1.371672, 1.251667, 1.104041, 0.933441,
0.745545, 0.546950, 0.344977, 0.147424,
                 -0.037728, -0.202641, -0.339961, -0.443176, -0.506964, -
0.527547, -0.503011, -0.433581, -0.321801,
                 -0.172561],
             [1.394411, 1.247662, 1.041866, 0.774167, 0.440738, 0.038949, -
0.427809, -0.945836, -1.492178, -2.064046,
              -2.642606, -3.206365, -3.730843, -4.189995, -4.557797, -4.808996,
-4.918434, -4.859722, -4.611325,
              -4.155739
              ])

    plotting([-5.000000, -4.600000, -4.200000, -3.800000, -3.400000, -3.000000,
-2.600000, -2.200000, -1.800000,
              -1.400000, -1.000000, -0.600000, -0.200000, 0.200000, 0.600000,
1.000000, 1.400000, 1.800000, 2.200000,
              2.600000
              ], [-0.263504, -0.387370, -0.474751, -0.520928, -0.522860, -
0.479494, -0.392005, -0.263919, -0.101065,
                 0.088697, 0.295990, 0.510635, 0.722471, 0.922137, 1.101675,
1.254841, 1.377132, 1.465588, 1.518496,
                 1.535111
                 ],
             [3.000000, 3.400000, 3.800000, 4.200000, 4.600000, 5.000000,
5.400000, 5.800000, 6.200000, 6.600000,
              7.000000, 7.400000, 7.800000, 8.200000, 8.600000, 9.000000,
9.400000, 9.800000, 10.200000, 10.600000
              ], [1.515496, 1.460469, 1.371672, 1.251667, 1.104041, 0.933441,
0.745545, 0.546950, 0.344977, 0.147424,
                 -0.037728, -0.202641, -0.339961, -0.443176, -0.506964, -
0.527547, -0.503011, -0.433581, -0.321801,
                 -0.172561],
             [1.509201, 1.458760, 1.377755, 1.268449, 1.134631, 0.981853,
0.816227, 0.642553, 0.464022, 0.288344,
              0.119955, -0.036844, -0.178203, -0.300573, -0.400987, -0.477626, -
0.529819, -0.556885, -0.559329,
              -0.538415
              ])
```