KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Intelektikos pagrindai 2020

Laboratorinio darbo nr. 2 ataskaita

Atliko:

IFF-7/2 gr. studentas

Justas Milišiūnas

2020-02-29

Dėstytojai:

lekt. Audrius Nečiūnas

doc. Agnė Paulauskaitė-Tarasevičienė

KAUNAS 2020

TURINYS

1. Dalis Nr. 1

1.1. Užduotis

Šiame darbe bus prognozuojamas saulės dėmių aktyvumas, išreikštas saulėje stebimų dėmių kiekiu tam tikrais kalendoriniais metais. Šis aktyvumas turi 11 metų ciklišKumą. Ataskaitoje bus pateiktas scenarijaus programos kodas su komentarais, sukurti grafikai, klausimų atsakymai su atliktų tyrimų rezultatais.

1.2. Programos kodas

```
class NeuralNetwork:
    def __init__(self, _inputs_count, _hidden_layers_count, _outputs_count, min_value=0,
                 max_value=0):
        self.inputs_count = _inputs_count
        self.hidden_layers_count = _hidden_layers_count
        self.outputs_count = _outputs_count
        self.min_value = min_value
        self.max_value = max_value

        self.network = []
        hidden_layer = [{'weights': list(2 * np.random.random(_inputs_count + 1) - 1)} for i in
                        range(_hidden_layers_count)]
        self.network.append(hidden_layer)
        output_layer = [{'weights': list(2 * np.random.random(_outputs_count + 1) - 1)} for i in
                        range(_outputs_count)]
        self.network.append(output_layer)

    def transfer(self, activation):
        return 1.0 / (1.0 + np.exp(-activation))

    def activate(self, weights, inputs):
        """
        Calcules weighted sum of inputs with given weights
        :param weights: Layer's weights
        :param inputs: Inputs
        :return: Weighted sum of inputs
        """
        activation = weights[-1]
        for i in range(len(weights) - 1):
            activation += weights[i] * inputs[i]

        return activation

    # Calculate the derivative of an neuron output
    def transfer_derivative(self, output):
        """
        Used formula: f(x) = 1 / (1 + e^(-x))
        :param output: x
        :return: calculated value
        """
        return output * (1.0 - output)

    def forward_propagate(self, row):
        """
        Goes through each network's layer and sums multiplied inputs by layer's weights + bias
        :param row: input data
        :return: calculated output
        """
        inputs = row
        for layer in self.network:
            new_inputs = []
            for neuron in layer:
```

```
            activation = self.activate(neuron['weights'], inputs)
            neuron['output'] = self.transfer(activation)
            new_inputs.append(neuron['output'])

        inputs = new_inputs
    return inputs

def backward_propagate_error(self, expected):
    """
    Calculates and saves each neuron's error
    :param expected: expected output
    """
    for i in reversed(range(len(self.network))):
        layer = self.network[i]
        errors = list()
        if i != len(self.network) - 1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in self.network[i + 1]:
                    error += (neuron['weights'][j] * neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])

        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * self.transfer_derivative(neuron['output'])

def update_weights(self, row, l_rate):
    """
    Updates each neuron's weight with it's saved error
    :param row: input data
    :param l_rate: learning rate
    """
    for i in range(len(self.network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in self.network[i - 1]]
        for neuron in self.network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] += l_rate * neuron['delta']

def train(self, dataset, l_rate, n_epoch, print_info=True):
    """
    Manages network's training by getting current prediction,
    calculating error, updating weights based on that error
    :param dataset: train dataset
    :param l_rate: learning rate
    :param n_epoch: number of epochs
    :param print_info: print epoch info
    """
    normalized_data = self.__normalize_data(dataset)

    for epoch in range(n_epoch):
        sum_error = 0
        for row in normalized_data:
            outputs = self.forward_propagate(row)
            # expected = [0 for i in range(n_outputs)]
            # expected[row[-1]] = 1
            expected = [row[-1]]
            sum_error += sum([(expected[i] - outputs[i]) ** 2 for i in range(len(expected))])
```

```
            self.backward_propagate_error(expected)
            self.update_weights(row, l_rate)

        if print_info:
            print(f">epoch={epoch}, lrate={l_rate}, error={sum_error}")

    def validate(self, dataset, print_info=False):
        """
        Validates neural network by measuring prediction accuracy
        :param dataset: Validation data set
        :param print_info: Print results to console
        :return: MSE (Mean-Square Error), MAD (Median Absolute Deviation), errors list, predictions
list
        """
        errors = 0
        errors_list = []
        expected_prediction = []
        for row in dataset:
            prediction = self.predict(row)
            expected_output = row[-1]

            errors += (expected_output - prediction) ** 2
            errors_list.append(expected_output - prediction)
            expected_prediction.append([expected_output, prediction])
            if print_info:
                print(f"Expected={expected_output} Prediction={prediction}")

        mse = errors / len(dataset)
        if print_info:
            print(f"MSE: {mse}")

        return mse, np.median(np.abs(errors_list)), errors_list, expected_prediction

    def predict(self, row):
        """
        Does prediction from given the input
        :param row: input
        :return: prediction result
        """
        normalized = self.__normalize_prediction(row)
        outputs = self.forward_propagate(normalized)
        denormalized = self.__denormalize_prediction(outputs)
        return denormalized[0]

    def __denormalize_data(self, dataset):
        min_value = self.min_value
        max_value = self.max_value

        n_cols = len(dataset[0])
        _data = np.array(dataset).flatten()

        _data = np.array([number * (max_value - min_value) + min_value for number in _data])
        _data = np.reshape(_data, (-1, n_cols))

        return list(_data)

    def __denormalize_prediction(self, prediction):
        return [number * (self.max_value - self.min_value) + self.min_value for number in prediction]

    def __normalize_data(self, dataset, low=0, high=1):
        n_cols = len(dataset[0])
        _data = np.array(dataset).flatten()
```

```
        min_number = self.min_value
        max_number = self.max_value

        _data = np.array([(number - min_number) / (max_number - min_number) * (high - low) + low
for number in _data])
        _data = np.reshape(_data, (-1, n_cols))

        return list(_data)

    def __normalize_prediction(self, row):
        return [(number - self.min_value) / (self.max_value - self.min_value) for number in row]
```

Neurinio tinklo klasė.

```
sunspots = read_sunspots('sunspot.txt')

data_min = min(np.array(sunspots)[:, 1])
data_max = max(np.array(sunspots)[:, 1])

dataset = prepare_data(sunspots, 2)

training_set = dataset[:200]
validation_set = dataset[200:]

n_inputs = len(training_set[0]) - 1
n_outputs = 1

network = NeuralNetwork(n_inputs, 1, n_outputs, data_min, data_max)
network.train(training_set, 0.01, 10000)

MSE, MAD, errors, predictions = network.validate(validation_set)
```

Duomenų nuskaitymas. Duomenų iššskirstymas į treniravimui, validavimui. Tinklo treniravimas ir validatacija.

1.3. Tyrimai

1. Nubrėžti saulės dėmių aktyvumo 1700-2014 metais diagramą:

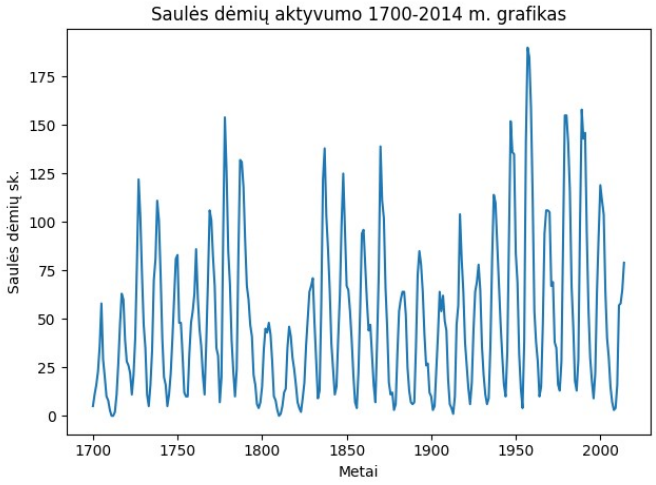

Diagrama 1: Saulės dėmių aktyvumas 1700-2014 metais

2. Nubrėžti įvesčių, išvesčių diagramą, kai įvestimis laikomi praeitų 2 metų duomenys:

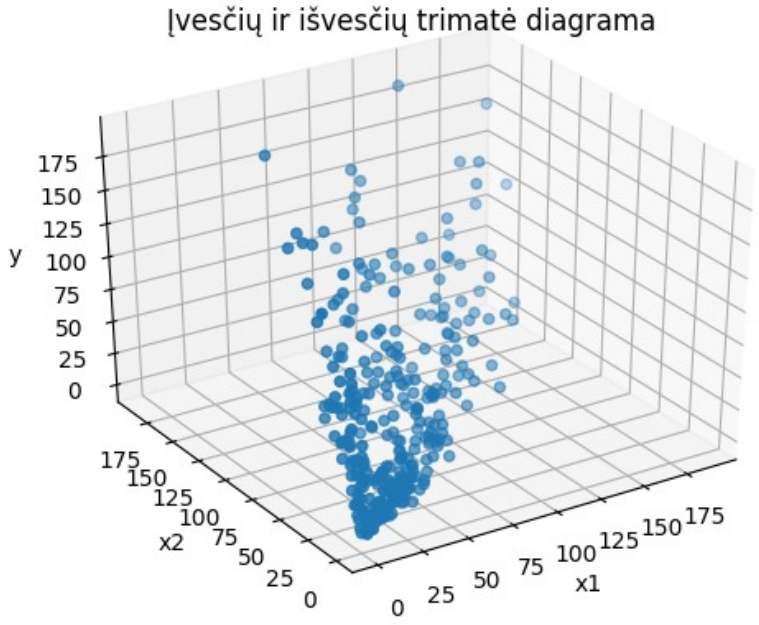

Diagrama 2: Įvestis ir išvestis

3. Kokia yra neurono svorio koeficientų w1, w2, b optimalių reikšmių parinkimo grafinė interpretacija?
Iš 2 diagramos matome, kad didžioji dalis duomenų pasiskirstę prie mažesnio kiekio dėmių reikšmių (Iki 50 dėmių). Geriausi svoriai dėl to būti tokie kurie mažai nutolę nuo 0.

4. Pavaizduoti gautas neurono svorio koeficientų reikšmes:
w1 = 3.19
w2 = -6.42
b1 = -0.27

5. Prognozuojamų reikšmių ir tikrųjų reikšmių palyginio grafikas naudojant treniravimo duomenis (Nuo 1700 iki 1900 metų):

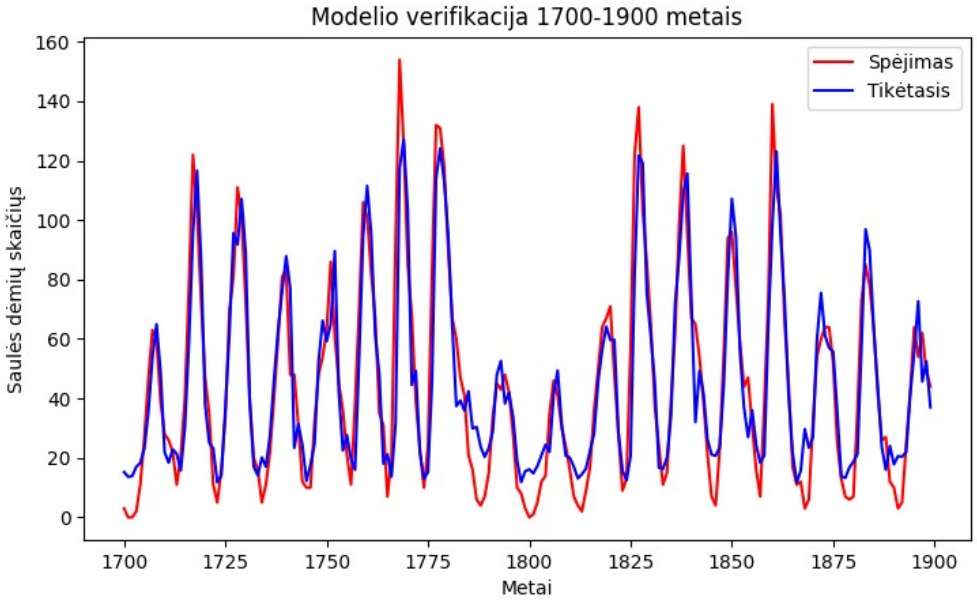

Diagrama 3: Prognozių palyginimas su tikrais duomenimis (1700-1900 metai)

6. Prognozuojamų reikšmių ir tikrųjų reikšmių palyginio grafikas naudojant visus duomenis (Nuo 1700 iki 2014 metų):

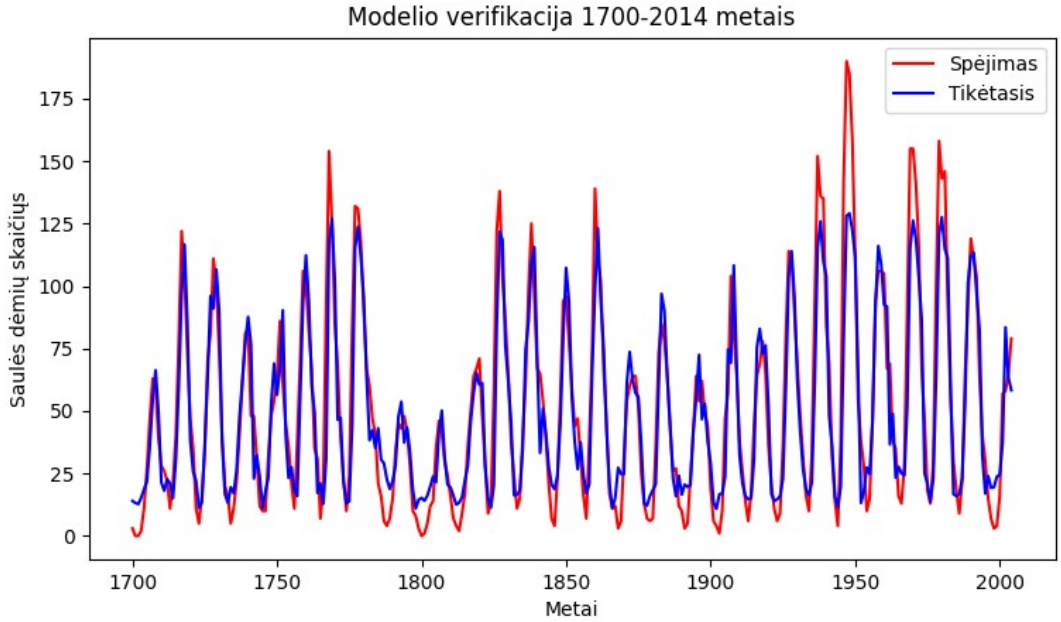

Diagrama 4: Prognozių ir tikrų duomenų palyginimas (1700-2014 metais)

7. Sukurti prognozės klaidos vektorių e. Nubrėžti prognozės klaidos grafiką:

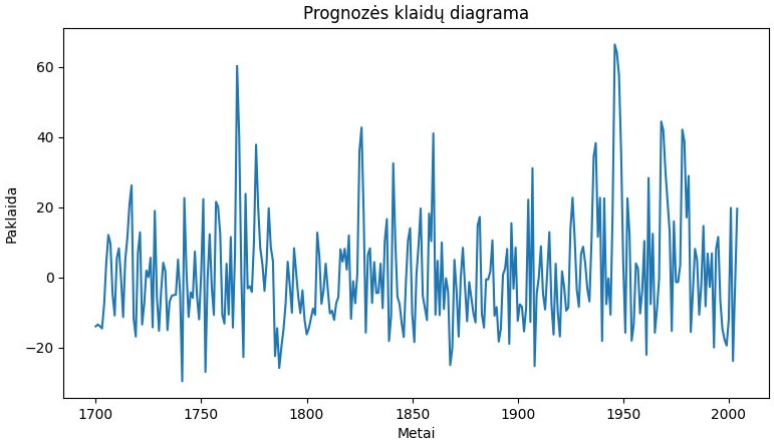

Diagrama 5: Klaidos vektoriaus e grafikas

8. Nubraižyti prognozės klaidų histogramą:

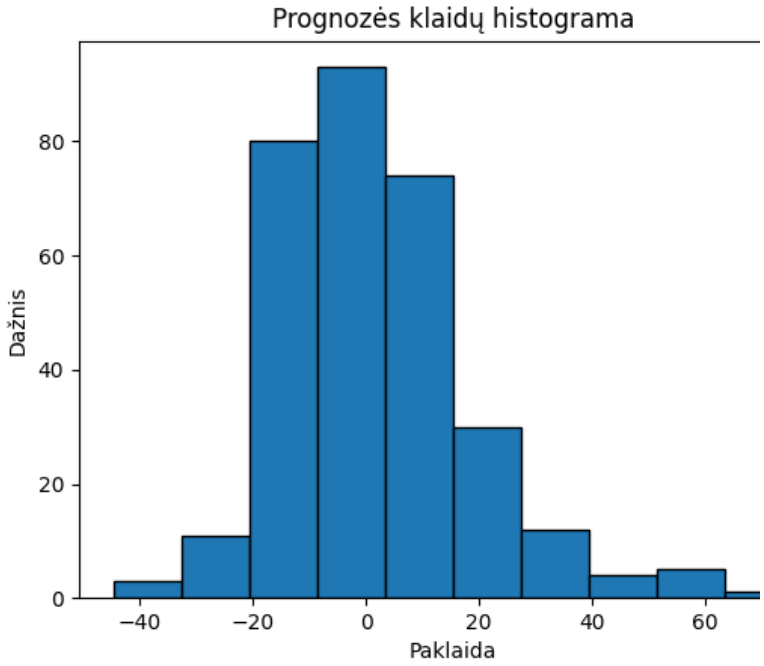

Diagrama 6: Prognozės klaidos

Diagrama 6: Prognozės klaidų histograma

9. Apskaičiuoti vidutinės kvadratinės prognozės klaidos reikšmę (ang. Mean-SquareError, MSE):
MSE = 289.32

10. Apskaičiuokite prognozės absoliutaus nuokrypio medianą (ang. Median Absolute Deviation):
MAD = 9.71

11. Palyginkite skirtumus tarp MSE ir MAD įverčių ir pakomentuokite:
Iš MAD reikšmės matome, kad dažniausiai spėjimas skiriasi tik per 10 dėmių. Iš MSE reikšmės galime spėti, kad yra keli spėjimai su labai dydeliu nuokrypiu nuo tikrosios reikšmės.

12. Ištirti epochų skaičiaus ir apmokymo reikšmės įtaką neuroninio tinklo apmokymui:
e – epochų skaičius
lr – apmokymo greitis

Kai e = 100, lr = 0.1:

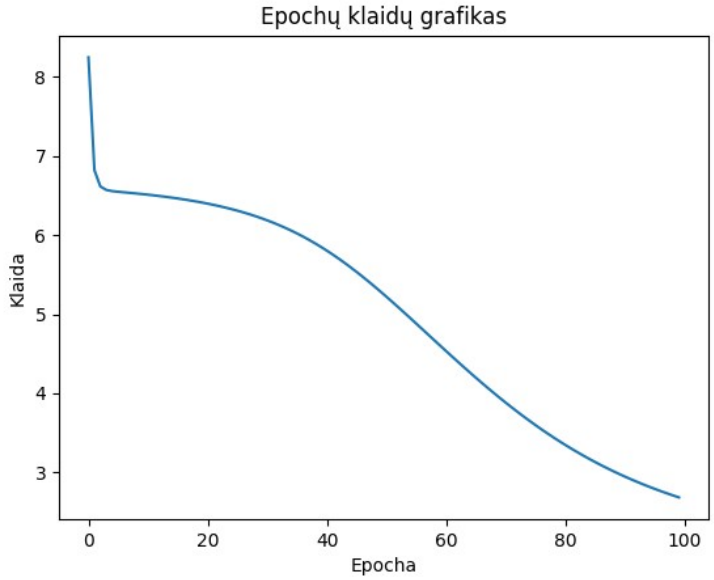

Diagrama 7: Epochų klaidos kai e = 100, lr = 0.1

MSE = 823.56, MAD = 20.2
Iš 7 diagramos matome, kad parinktas epochų skaičius yra per mažas, nes tinklas dar sparčiai tobulėja.

Kai e = 1000, lr = 0.1:

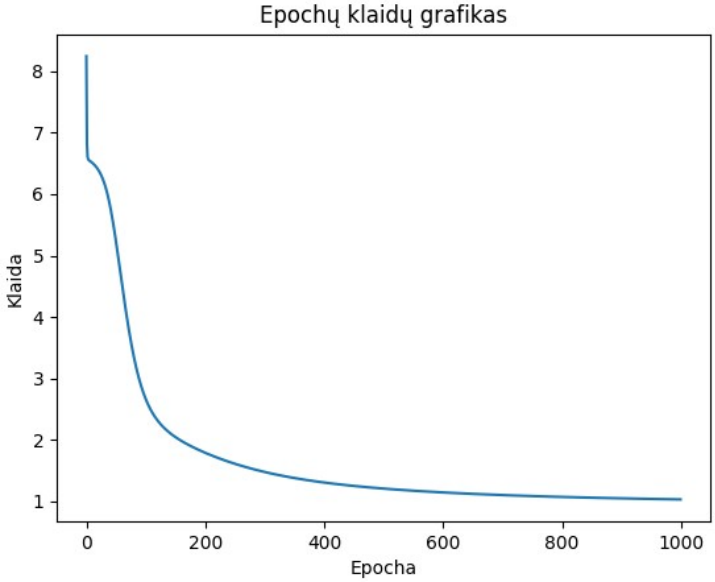

Diagrama 8: Epochų klaidos kai e = 1000, lr = 0.1

MSE = 290, MAD = 9.44

Iš 8 diagramos matome, kad tinklas kažkur nuo 800 epochos nustoja greitai tobulėti. Taip pat šįkart pasiektas daug didesnis prognozių tikslumas nei su 100 epochų.

Kai e = 50000, lr = 0.1:

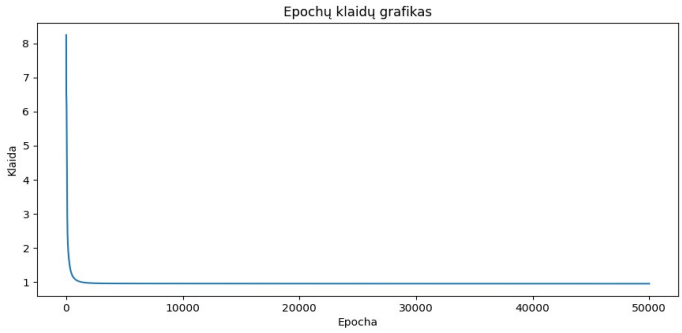

Diagrama 9: Epochų klaidos kai e = 50000, lr = 0.1

MSE: 292.6328454686611 MAD: 9.248102166282699

Iš šių rezultatų matome, kad keliant nuo 10000 iki 50000, kad MSE net pablogėjo, kol MAD nežymiai pagerėjo. Taip pat iš 9 diagramos matome, kad epochų skaičių toliau kelti nėra tikslo.

Kai e = 100, lr = 1:

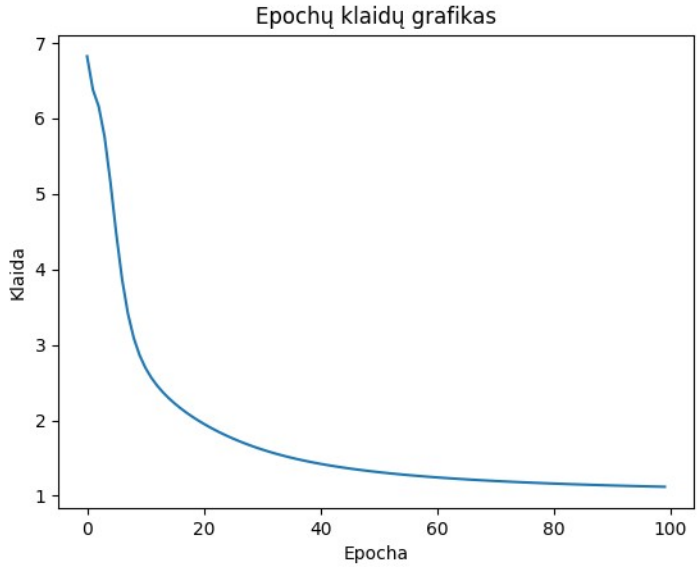

Diagrama 10: Epochų klaidos kai e = 100, lr = 1

MSE: 322.83779944999156 MAD: 9.375911060607624

Lyginant su rezultatais, kai e=100, lr=0.1, matome, kad esant mažam epochų skaičiui, didinant lr skirtumas yra labai didelis (MSE nukrito nuo 823 iki 322). Iš 10 diagramos matome, kad pradžioje tinklas labai greitai tobulėja.

Kai e=10000, lr=1:

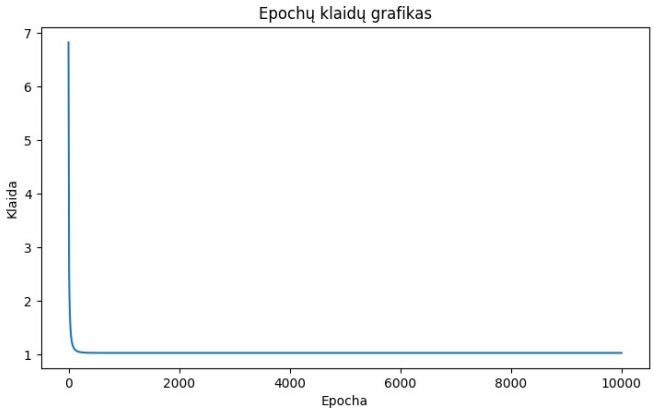

Diagrama 11: Klaidos kai e = 10000, lr = 1

MSE: 328.8569449559821 MAD: 9.203138673465375

Lyginant su rezultatais kai e=10000, lr=0.1, matome, kad prie didesnio epochų skaičio padidinus apmokymo greitį, rezultatai pradeda net blogėti (MSE pablogėjo nuo 290 iki 328). Pagal diagramą tinklo progresas smarkiai suletėja jau ties 300 epocha.

Tyrimo išvados:

- Prie mažo epochų skaičiaus (apie 100), apmokymo greičio didinimas labai pagerina rezultatus.
- Prie didesnio epochų skaičiaus (apie 1000) didelis apmokymo greitis blogina spėjimo tikslumą.
- Nustačius labai didelį epochų skaičius (pvz. 50000) prognozių paklaidos didėja. Tinklas persimoko.

13. Ištirti prognozių kokybę remiantis daugiau anskestiųjų duomenų:
Su 4 įvestimis:

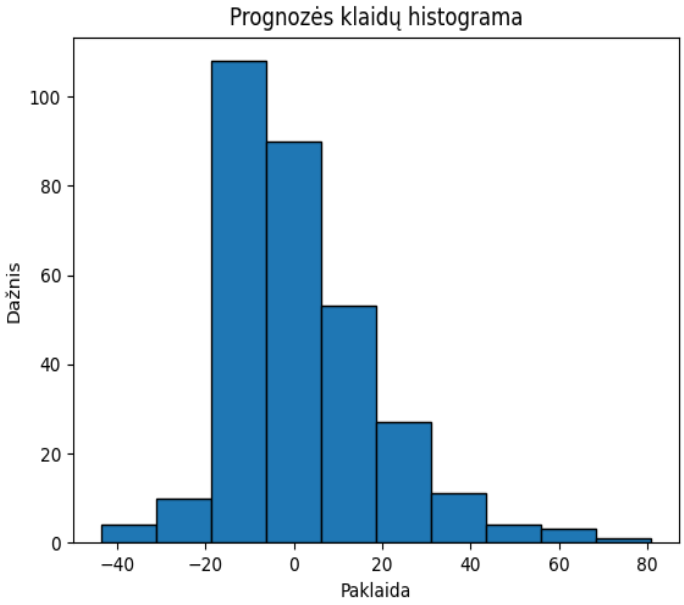

Diagrama 12: Prognozės klaidų histograma su 4 įvestimis

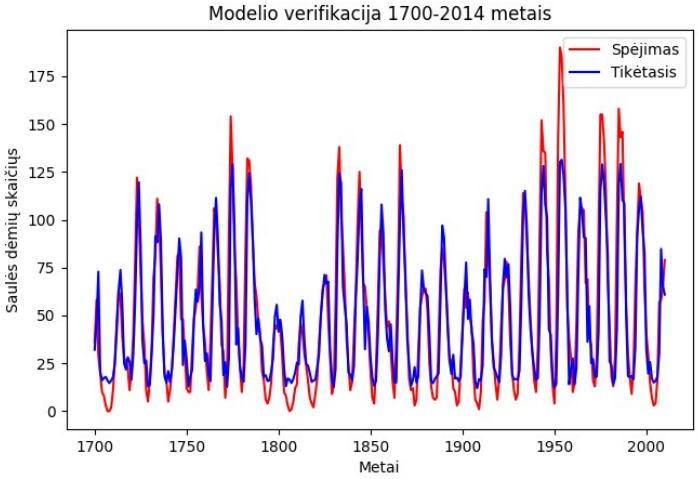

Diagrama 13: Prognozės palyginimas su tikrais duomenimis kai 4 įvestis

MSE: 291.33250974165816 MAD: 9.981920096048825

Kai 10 įvesčių:

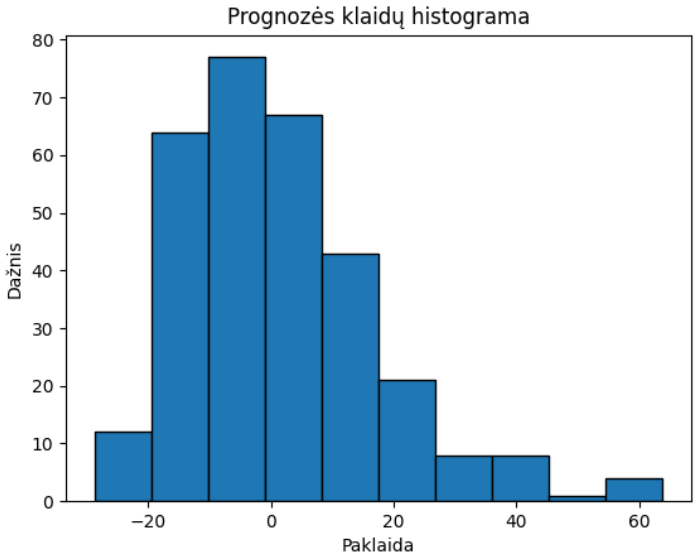

Diagrama 14: Prognozės klaidų histograma su 10 įvesčių

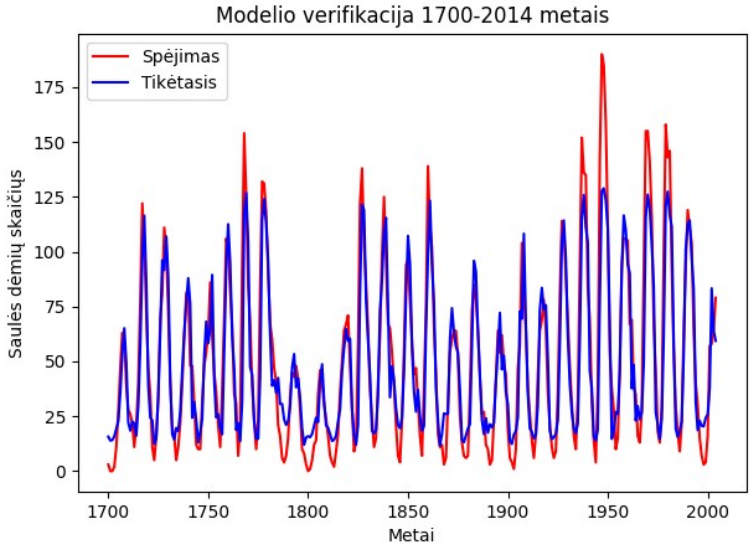

Diagrama 15: Prognozės palyginimas su tikrais duomenimis kai 10 įvesčių

MSE: 254.64148717508198 MAD: 9.77085054955596

Iš šių rezultatu matome, kad didinant įvesčių skaičių prognozės kokybė pradeda gerėti (2 įvestis MSE 290, su 10 įvesčių 254).

2. Dalis Nr. 2

2.1. Pradinio duomenų rinkinio aprašymas

Duomenų rinkinį sudaro 11 stulpelių. 54 tūksnt. Įrašu. Bus bandoma prognozuoti deimanto kainą. Stulpeliai:

- Id – įrašo numeracija
- Carat – deimanto karatai
- Cut (Fair, Good, Very Good, Premium, Ideal) – deimanto pjūvis
- Color (From J (worst) to D (best)) – deimanto spalva
- Clarity (I1 (worst), SI2, SI1, VS1, VS2, VVS2, VVS1, IF (best)) – deimanto skaidrumas
- Depth percentage – deimanto gylis padalintas iš deimanto pločio
- Table – deimanto viršaus plotis reliatyvus plačiausiai vietai
- Price – deimanto kaina
- Length – deimanto ilgis
- Width – deimanto plotis
- Depth – deimanto gylis

2.2. Duomenų rinkinio pertvarkymo aprašas

- Tikslinio stulpelio duomenys normalizuoti intervale [0; 1]
- Kategoriniai stulpelių reikšmės pakeistos skaitinėmis [0; n] ir normalizuotos intervale [0; 1]

2.3. DNT architektūra, įskaitant parametrų vertes (mokymosi greitis, aktyvavimo funkcija)

- Tinklas turi 9 įvestis (Carat, cut, color, clarity, depth percetange, table, length, width, depth) ir 1 išvestį (price).
- Pradinės svorių reikšmės bus nustatomos atsitiktinių būdų (Visų svorių suma beveik lygi 0).
- Tinklas susidaro iš 2 sluoksnių.
- Naudojama sigmoidės funkcija aktivacijai.
- Mokymosi greitis 0.5, 100 epochų.

2.4. 10 intervalų kryžminės patikros eksperimentų rezultatai

Kai apmokymo greitis 0.5, 100 epochų.

Intervalas	MSE	MAD
[0; 5394)	0.0028	0.021
[5394; 10788)	0.0029	0.025
[10788; 16182)	0.0026	0.021
[16182; 21576)	0.0028	0.02
[21576; 26970)	0.0029	0.024
[26970; 32364)	0.0028	0.02
[32364; 37758)	0.0028	0.024
[37758; 43152)	0.0029	0.022
[43152; 48546)	0.0032	0.021
[48546; 53940]	0.0026	0.022
Vidurkis	0.0028	0.022

2.5. Nurodyti priemones, kurių buvo imtasi siekiant pagerinti rezultatus

Mokymosi greitis pakeistas į 0.1. Epochų skaičius 200.

Intervalas	MSE	MAD
[0; 5394)	0.0027	0.021
[5394; 10788)	0.0026	0.023
[10788; 16182)	0.0026	0.020
[16182; 21576)	0.0028	0.02
[21576; 26970)	0.0026	0.022
[26970; 32364)	0.0026	0.021
[32364; 37758)	0.0025	0.021
[37758; 43152)	0.0028	0.021
[43152; 48546)	0.0029	0.021
[48546; 53940]	0.0025	0.021
Vidurkis	0.0026	0.021

Lyginant lenteles, gauname, kad sumažinus apmokymo greitį iki 0.1 ir padidinus epochų skaičių iki 200 gauname 7% tikslesnias prognozes (pagal MSE).

2.6. Išvados

1. Norint gauti kuo tikslesnias prognozes, reikia pabandyti pakaitalioti apmokymo greitį, epochų skaičių.
2. Didesnis apmokymo greitis pagerina apmokymą prie mažo epochų skaičio
3. Prie mažo apmokymo greičio apmokymui reikalingas didesnis epochų skaičius