Министерство образования Республики Беларусь

УО «Брестский государственный технический университет»

Кафедра ИИТ

**Лабораторные работы №1-5**

По дисциплине: “Методы и алгоритмы принятия решений”

Тема: “Нейронные сети”

**Вариант №9**

**Выполнил**: студент 2 курса группы ПО-7 Крупенков Михаил Дмитриевич

**Проверил:** Крощенко А.А.

Брест 2021

# Универсальная нейронная сеть

Для выполнения всех лабораторных работ в стенах Брестского Государственного Технического Университета была разработана универсальная динамическая нейронная сеть (universal dynamic neural network – uninn). Её особенность: вариативное количество слоев с разными функциями активации, количеством нейронов.

Ссылка на git: <https://github.com/FalseR20/miapr_po7/tree/main/reports/Krupenkov/universal/src>

## Код универсальной нейронной сети

### funsact.py

#### **import** numpy **as** np **def** sigmoid(s): *# sigmoid* **return** 1.0 / (1.0 + np.exp(-s)) **def** d\_sigmoid(y): *# sigmoid'* **return** y \* (1.0 - y) **def** linear(s): *# linear* **return** s **def** d\_linear(\_): *# linear'* **return** 1.0

### uninn.py

**import** os  
**import** pickle  
**import** numpy **as** np  
**import** funsact  
**from** numpy.typing **import** NDArray  
**from** typing **import** Optional, Callable  
  
  
**class** Layer:  
 *"""Слой нейросети"""* **def** \_\_init\_\_(  
 self,  
 lens: tuple[int, int],  
 f\_act: Callable = funsact.linear,  
 d\_f\_act: Callable = funsact.d\_linear,  
 w=**None**, t=**None**,  
 is\_need\_vectorize=**False** ):  
 *"""  
 - lens (количество нейронов этого и следующего слоя)  
 - функции активации  
 """* self.lens = lens  
 self.w = np.random.uniform(-0.5, 0.5, lens) **if** w **is None else** w  
 self.t = np.random.uniform(-0.5, 0.5, lens[1]) **if** t **is None else** t  
 **if** is\_need\_vectorize:  
 self.f\_act = np.vectorize(f\_act)  
 self.d\_f\_act = np.vectorize(d\_f\_act)  
 **else**:  
 self.f\_act = f\_act  
 self.d\_f\_act = d\_f\_act  
  
 **def** go(self, x: NDArray) -> NDArray:  
 *"""Прохождение слоя"""* self.x = x  
 self.s: NDArray = np.dot(x, self.w) - self.t  
 self.y: NDArray = self.f\_act(self.s)  
 **return** self.y  
  
 **def** back\_propagation(  
 self,  
 error: NDArray,  
 alpha: Optional[float],  
 is\_first\_layer=**False** ) -> Optional[NDArray]:  
 *"""Обратное распространение ошибки с изменением весов, порога"""* error\_later = (  
 np.dot(error \* self.d\_f\_act(self.y), self.w.transpose())  
 **if not** is\_first\_layer  
 **else None** )  
  
 **if not** alpha:  
 alpha = self.adaptive\_alpha(error)  
  
 gamma = alpha \* error \* self.d\_f\_act(self.y)  
 self.w -= np.dot(self.x.reshape(-1, 1), gamma.reshape(1, -1))  
 self.t += gamma  
  
 **return** error\_later  
  
 **def** adaptive\_alpha(self, error: NDArray) -> float:  
 **if not** hasattr(self, **"d\_f\_act\_0"**):  
 self.d\_f\_act\_0 = self.d\_f\_act(self.f\_act(0))  
 alpha = (  
 (error \*\* 2 \* self.d\_f\_act(self.y)).sum()  
 / self.d\_f\_act\_0  
 / (1 + (self.x \*\* 2).sum())  
 / ((error \* self.d\_f\_act(self.y)) \*\* 2).sum()  
 )  
 **return** alpha  
  
  
**class** NeuralNetwork:  
 **def** \_\_init\_\_(self, \*args: Layer) -> **None**:  
 *"""Создание нейросети с заданием массива слоев"""* self.layers = args  
 self.back\_propagation\_range = range(len(self.layers) - 1, 0, -1)  
  
 **def** go(self, x: NDArray) -> NDArray:  
 *"""Прохождение всех слоев нейросети"""* **for** layer **in** self.layers:  
 x = layer.go(x)  
 **return** x  
  
 **def** learn(  
 self,  
 x: NDArray[NDArray],  
 e: NDArray[NDArray],  
 alpha: Optional[float] = **None** ) -> NDArray:  
 *"""Обучение наборами обучающих выборок  
  
 - x: (n, len\_in) ... [[1 2] [2 3]]  
 - e: (n, len\_out) ....... [[3] [4]]  
 """* square\_error = np.zeros(self.layers[-1].lens[1])  
 **for** i **in** range(len(e)):  
 y: NDArray = self.go(x[i])  
 error: NDArray = y - e[i]  
 square\_error += error \*\* 2 / 2  
 **for** layer\_i **in** self.back\_propagation\_range:  
 error = self.layers[layer\_i].back\_propagation(error, alpha)  
 self.layers[0].back\_propagation(error, alpha, is\_first\_layer=**True**)  
 **return** square\_error / self.layers[-1].lens[1]  
  
 **def** prediction\_results\_table(self, x: NDArray[NDArray], e: NDArray[NDArray]) -> **None**:  
 *"""Красивый вывод прогона тестирующей выборки"""* print(  
 **" эталон выходное значение разница среднекв. ошибка"** )  
 y = self.go(x)  
 y = y.reshape(-1)  
 e = e.reshape(-1)  
 delta = y - e  
 square\_error = delta \*\* 2 / 2  
 **for** i **in** range(len(e)):  
 print(**f"{**e[i] **: 22}{**y[i] **: 25}{**delta[i] **: 25}{**square\_error[i] **: 25}"**)  
 print(**f"\n-- Final testing square error: {**np.average(square\_error) \* 2**} --"**)  
  
  
**def** save(nn: NeuralNetwork, filename=**None**) -> **None**:  
 ans = input(**"Желаете сохранить? (y/n): "**)  
 **if** ans **and** (ans[0] == **"y" or** ans[0] == **"н"**):  
 **if** filename **is None**:  
 filename = input(**"Имя файла (\*.nn): "**) + **".nn"** filename = **"nn\_files/"** + filename  
 **if not** os.path.exists(**"nn\_files"**):  
 os.mkdir(**"nn\_files"**)  
 **with** open(filename, **"wb"**) **as** file:  
 pickle.dump(nn, file)  
 print(**"Сохранено в"**, filename)  
 **else**:  
 print(**"Сохранение отклонено"**)  
  
  
**def** load(filename=**None**) -> NeuralNetwork:  
 **if** filename **is None**:  
 filename = input(**"Имя файла (\*.nn): "**) + **".nn"** filename = **"nn\_files/"** + filename  
 **if** os.path.exists(filename):  
 **with** open(filename, **"rb"**) **as** file:  
 new\_nn: NeuralNetwork = pickle.load(file)  
 **return** new\_nn  
 **else**:  
 **raise** FileNotFoundError  
  
  
**class** LayerSigmoid(Layer):  
 *"""Слой нейросети с сигмоидной функцией активации"""* **def** \_\_init\_\_(self, lens: tuple[int, int]):  
 *"""Слой нейросети с сигмоидной функцией активации"""* super().\_\_init\_\_(lens, f\_act=funsact.sigmoid, d\_f\_act=funsact.d\_sigmoid)  
  
 **def** back\_propagation(  
 self, error: NDArray,  
 alpha: Optional[float],  
 is\_first\_layer=**False** ) -> Optional[NDArray]:  
 *"""Обратное распространение ошибки с изменением весов, порога"""* error\_later = (  
 np.dot(error \* self.y \* (1 - self.y), self.w.transpose())  
 **if not** is\_first\_layer  
 **else None** )  
  
 **if not** alpha:  
 alpha = self.adaptive\_alpha(error)  
  
 gamma = alpha \* error \* self.y \* (1 - self.y)  
 self.w -= np.dot(self.x.reshape(-1, 1), gamma.reshape(1, -1))  
 self.t += gamma  
  
 **return** error\_later  
  
 **def** adaptive\_alpha(self, error) -> float:  
 alpha = (  
 4  
 \* (error \*\* 2 \* self.y \* (1 - self.y)).sum()  
 / (1 + (self.x \*\* 2).sum())  
 / np.square(error \* self.y \* (1 - self.y)).sum()  
 )  
 **return** alpha  
  
  
**class** LayerLinear(Layer):  
 *"""Слой нейросети с линейной функцией активации"""* **def** \_\_init\_\_(self, lens: tuple[int, int]):  
 *"""Слой нейросети с линейной функцией активации"""* super().\_\_init\_\_(lens, funsact.linear, funsact.d\_linear)  
  
 **def** back\_propagation(  
 self, error: NDArray,  
 alpha: Optional[float],  
 is\_first\_layer=**False** ) -> Optional[NDArray]:  
 *"""Обратное распространение ошибки с изменением весов, порога"""* error\_later = np.dot(error, self.w.transpose()) **if not** is\_first\_layer **else None  
  
 if not** alpha:  
 alpha = self.adaptive\_alpha(error)  
  
 gamma = alpha \* error  
 self.w -= np.dot(self.x.reshape(-1, 1), gamma.reshape(1, -1))  
 self.t += gamma  
  
 **return** error\_later  
  
 **def** adaptive\_alpha(self, error) -> float:  
 alpha = 1 / (1 + np.square(self.x).sum())  
 *# alpha = np.square(error).sum() / (1 + np.square(self.x).sum()) / error.sum()* **return** alpha  
  
  
**def** predict\_set(  
 begin: float,  
 lenght: float,  
 count: int,  
 step: float,  
 function: Optional[Callable] = **None**) -> tuple[NDArray[NDArray], NDArray[NDArray]]:  
 *"""Набор обучающей выборки типа:  
  
 x = [ [1 2 3] [2 3 4] [3 4 5] ]\n  
 e = [ [4] [5] [6] ]"""* base\_array = np.arange(count + lenght)  
 x = np.zeros(shape=(count, lenght))  
 **for** i **in** range(count):  
 x[i] = base\_array[i: lenght + i]  
 e = base\_array[lenght: lenght + count].reshape(-1, 1)  
  
 x = x \* step + begin  
 e = e \* step + begin  
 **if** function **is None**:  
 **return** x, e  
 **else**:  
 **return** function(x), function(e)  
  
  
**def** shuffle\_set(  
 x: NDArray[NDArray],  
 e: NDArray[NDArray]  
) -> tuple[NDArray[NDArray], NDArray[NDArray]]:  
 *"""Перемешивание набора обучающей выборки"""* randomize = np.arange(len(x))  
 np.random.shuffle(randomize)  
 x = x[randomize]  
 e = e[randomize]  
 **return** x, e

## Структура

Автор: Крупенков Михаил (см. фотокарточку)

# Однослойный персептрон.

# Линейная функция активации

### Цель работы:

Изучить обучение и функционирование линейной ИНС при решении задач прогнозирования. Изучить обучение и функционирование линейной ИНС с применением адаптивного шага.

### Постановка задачи:

Написать на любом ЯВУ программу моделирования прогнозирующей линейной ИНС. Для тестирования использовать функцию

Согласно варианту 9 в таблице:

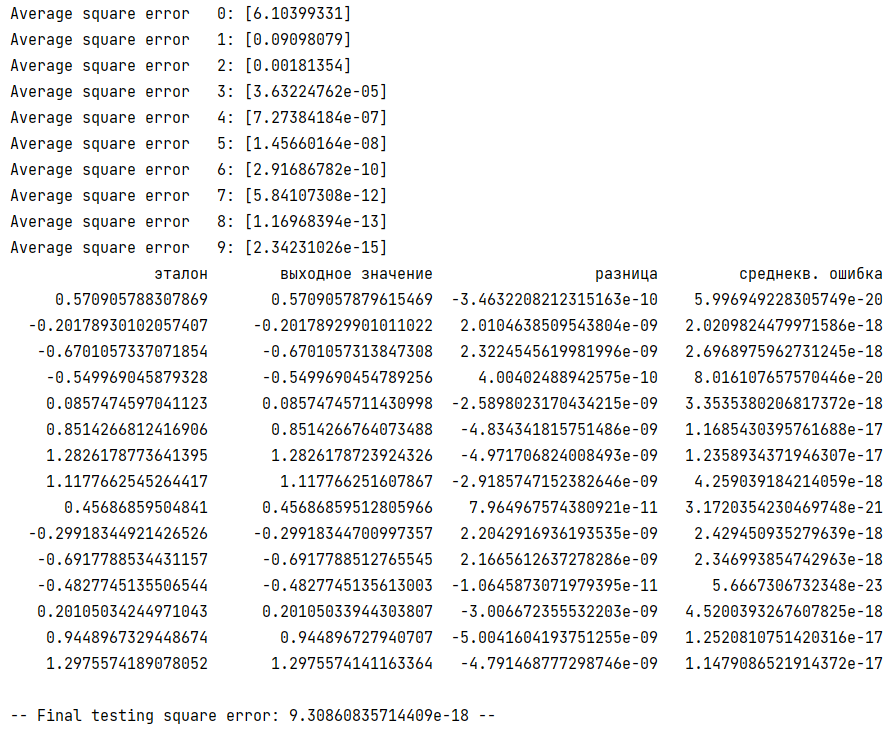
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варианта | a | b | d | Кол-во входов ИНС |
| 9 | 1 | 8 | 0.3 | 5 |

## Без адаптивного шага

### lab1u.py

**from** uninn **import** \*  
  
  
*# Функция по условию (лаб.1 вар.9)***def** function\_lab1\_9(x: float) -> float:  
 **return** np.sin(8 \* x) + 0.3  
  
  
**def** main():  
 layer = Layer(lens=(5, 1)) *# Создание слоя* nn = NeuralNetwork(layer) *# Создание нейронной сети с этим слоем* learn\_x, learn\_e = predict\_set(0, 5, 30, 0.1, function=function\_lab1\_9) *# Набор для предсказания функции* **for** t **in** range(10): *# Прогон набора \_ раз* square\_error = nn.learn(learn\_x, learn\_e, 0.08) *# Метод для обучения* print(**f"Average square error {**t **: 3}: {**square\_error**}"**) *# Вывод на каждой итерации средней ошибки* test\_x, test\_e = predict\_set(3, 5, 15, 0.1, function=function\_lab1\_9) *# Набор для тестирования* nn.prediction\_results\_table(test\_x, test\_e) *# Метод для красивого вывода***if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

## Результаты программы



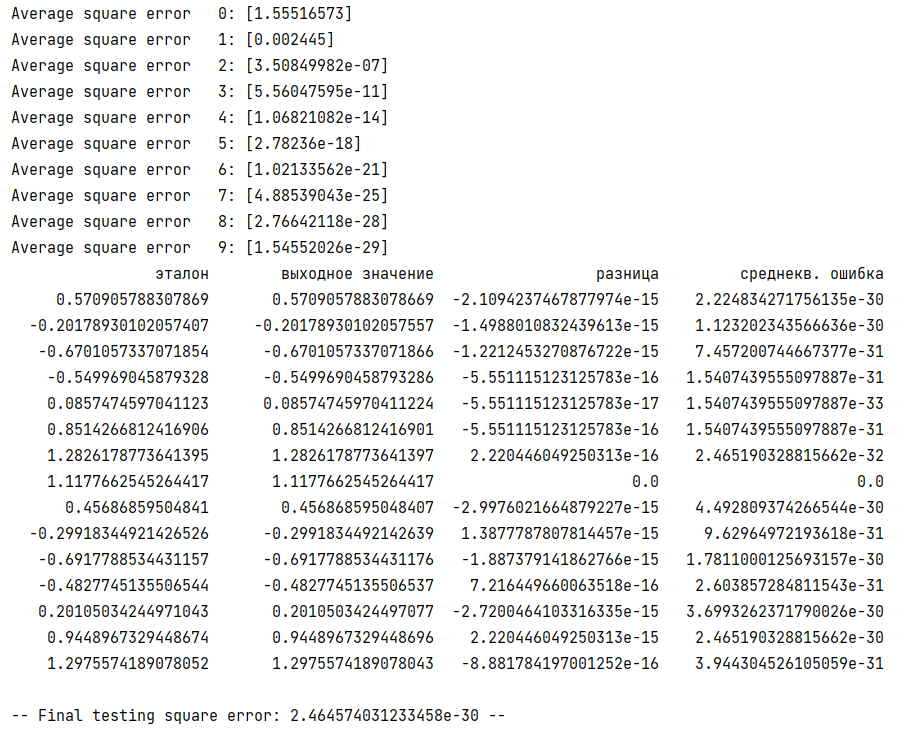
## С адаптивным шагом

Для того, чтобы использовать адаптивный шаг, нужно не вводить его в метод learn или вводить None.

### lab2u.py

**from** uninn **import** \*  
**from** lab1u **import** function\_lab1\_9  
  
  
**def** main():  
 layer = LayerLinear(lens=(5, 1))  
 nn = NeuralNetwork(layer)  
  
 learn\_x, learn\_e = predict\_set(0, 5, 30, 0.1, function=function\_lab1\_9)  
 **for** t **in** range(10):  
 square\_error = nn.learn(learn\_x, learn\_e)  
 print(**f"Average square error {**t **: 3}: {**square\_error**}"**)  
  
 test\_x, test\_e = predict\_set(3, 5, 15, 0.1, function=function\_lab1\_9)  
 nn.prediction\_results\_table(test\_x, test\_e)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

## Результаты программы



Благодаря адаптивному шагу нейронная сеть достигает лучших результатов за одинаковое количество повторений.

# Нелинейные ИНС в задачах прогнозирования

### Цель работы:

Изучить обучение и функционирование нелинейной ИНС при решении задач прогнозирования

### Постановка задачи:

Написать на любом ЯВУ программу моделирования прогнозирующей нелинейной ИНС. Для тестирования использовать функцию

.

Варианты заданий приведены в следующей таблице:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № варианта | a | b | с | d | Кол-во входов | Кол-во НЭ в скрытом слое |
| 9 | 0.1 | 0.3 | 0.08 | 0.3 | 10 | 4 |

.

Для прогнозирования использовать многослойную ИНС с одним скрытым слоем. В качестве функций активации для скрытого слоя использовать сигмоидную функцию, для выходного - линейную.

## Без адаптивного шага

### lab3u.py

**from** uninn **import** \*  
**import** time  
  
  
**def** function\_lab3\_9(x):  
 **return** 0.1 \* np.cos(0.3 \* x) + 0.08 \* np.sin(0.3 \* x)  
  
  
**def** main():  
 nn = NeuralNetwork(  
 Layer(lens=(10, 4), f\_act=funsact.sigmoid, d\_f\_act=funsact.d\_sigmoid),  
 Layer(lens=(4, 1))  
 )  
  
 learn\_x, learn\_e = predict\_set(0, 10, 30, 0.1, function=function\_lab3\_9)  
 test\_x, test\_e = predict\_set(3, 10, 15, 0.1, function=function\_lab3\_9)  
 alpha = 0.05  
 times = 30  
 sep = 1000  
 print(**f"- Learning {**times \* sep**} times -"**)  
  
 start\_time = time.time()  
  
 **for** thousand **in** range(times):  
 **for** \_ **in** range(sep - 1):  
 nn.learn(learn\_x, learn\_e, alpha)  
 error = nn.learn(learn\_x, learn\_e, alpha)[0]  
 print(**f"{**thousand + 1 **: 5d}x{**sep**} error: {**error **: .5e}"**)  
  
 print(**f"- Learning time: {**time.time() - start\_time**} seconds -"**)  
  
 nn.prediction\_results\_table(test\_x, test\_e)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

## Добавить смайлик &quot;booba&quot; и &quot;no booba&quot; | Форум социальной инженерии Lolz.guruРезультаты программы

## С адаптивным шагом

### lab4u.py

**from** uninn **import** \*  
**import** time  
**from** lab3u **import** function\_lab3\_9  
  
  
**def** main():  
 nn = NeuralNetwork(  
 LayerSigmoid(lens=(10, 4)),  
 LayerLinear(lens=(4, 1))  
 )  
  
 learn\_x, learn\_e = predict\_set(0, 10, 30, 0.1, function=function\_lab3\_9)  
 test\_x, test\_e = predict\_set(3, 10, 15, 0.1, function=function\_lab3\_9)  
 times = 30  
 sep = 1000  
 print(**f"- Learning {**times \* sep**} times -"**)  
  
 start\_time = time.time()  
  
 **for** thousand **in** range(times):  
 **for** \_ **in** range(sep - 1):  
 nn.learn(learn\_x, learn\_e)  
 error = nn.learn(learn\_x, learn\_e)[0]  
 print(**f"{**thousand + 1 **: 5d}x{**sep**} error: {**error **: .5e}"**)  
  
 print(**f"- Learning time: {**time.time() - start\_time**} seconds -"**)  
  
 nn.prediction\_results\_table(test\_x, test\_e)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

## Результаты программы

BOOBA

# Нелинейные ИНС в задачах распознавания образов

### Цель работы

Изучить обучение и функционирование нелинейной ИНС при решении задач распознавания образов.

### Постановка задачи

Написать на любом ЯВУ программу моделирования нелинейной ИНС для распознавания образов. Рекомендуется использовать сигмоидную функцию, но это не является обязательным. Количество НЭ в скрытом слое взять согласно варианту работы №3. Его можно варьировать, если сеть не обучается или некорректно функционирует.

Провести исследование полученной модели. При этом на вход сети необходимо подавать искаженные образы, в которых инвертированы некоторые биты. Критерий эффективности процесса распознавания - максимальное кодовое расстояние (количество искаженных битов) между исходным и поданным образом.

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Вектор 1 | Вектор 2 | Вектор 3 |
| 9 | 2 | 4 | 8 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | Данные вектора | | | | | | | | | | | | | | | | | | | |
| 2 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 4 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 8 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |

### lab5u.py