Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №3

По дисциплине «Методы решения задач в И С»

Тема: «Бинарная классификация с использованием кросс-энтропии (логистическая регрессия)»

Выполнил:

Студент 3 курса

Группы ИИ-26

Ковальчук А. И.

Проверил:

Андренко К. В.

Брест 2026

Цель работы: Изучить принципы бинарной классификации с вероятностной точки зрения. Реализовать однослойную нейронную сеть (логистический регрессор) с сигмоидной функцией активации и функцией потерь в виде бинарной кросс-энтропии (Binary Cross-Entropy, BCE). Провести сравнительный анализ скорости сходимости и качества классификации с моделями, использующими среднеквадратичную ошибку (ЛР №1) и адаптивный шаг обучения (ЛР №2).

Постановка задачи:

1. Модифицировать программу из ЛР №1:
   - Заменить пороговую функцию активации на сигмоидную.
   - Заменить функцию ошибки MSE на бинарную кросс-энтропию (BCE).
   - Реализовать вычисление градиента ŷ-exi для обновления весов.
2. Реализовать обучение с **фиксированным** шагом (взять значение α, показавшее наилучший результат в ЛР №1).
Запоминать суммарную ошибку Esp на каждой эпохе p.
3. Реализовать обучение с **адаптивным** шагом:
   - На каждой итерации (после подачи одного образа) вычислять t (формула (2.36)).
   - Использовать это $\alpha(t)$ для обновления весов и порога по градиенту BCE.
   - Сохранять значения Esp для построения графика.
4. Провести сравнительный анализ:
   - Построить на одном графике кривые сходимости Esp для четырёх конфигураций:
   - MSE + фиксированный шаг (данные из ЛР №1);
   - MSE + адаптивный шаг (данные из ЛР №2);
   - BCE + фиксированный шаг;
   - BCE + адаптивный шаг.
   - Зафиксировать число эпох, необходимое для достижения заданной точности Es≤Ee (взять Ee из ЛР №2, например 0.01).
5. Визуализировать результаты:
   - Отобразить точки обучающей выборки (разными цветами для классов 0 и 1).
   - Построить разделяющую линию, полученную после обучения модели с BCE (адаптивный шаг).
   - Для сравнения также показать линию из ЛР №1 (если она отличается). **Описать** сравнение двух линий.
6. Реализовать режим функционирования:
   - Пользователь вводит координаты x1,x2.
   - Сеть вычисляет вероятность ŷ и, применяя порог, определяет предсказанный класс.
   - Точка отображается на графике с разделяющей линией (добавляется маркером с подписью класса или вероятности).

7. Сравнить скорость сходимости BCE и MSE, объяснить различия с позиции теории. Оценить влияние адаптивного шага на обучение с BCE. Сравнить итоговые разделяющие линии, полученные разными методами, и объяснить возможные расхождения.
8. Написать вывод о применимости разных функций потерь для задачи бинарной классификации.

Вариант 7

x_1	x_2	e
4	6	0
-4	6	1
4	-6	1
-4	-6	1

Код программы:

```
import numpy as np


class DenseLayer:
    def __init__(self, units=1, activation='relu'):
        self.units = units
        self.activation = activation.lower()
        self.w = None
        self.b = None
        self.input = None
        self.z = None


    def init_weights(self, fan_in):
      if self.activation in ['relu', 'leaky_relu']:
        std = np.sqrt(2.0 / fan_in)
      else:
        std = np.sqrt(1.0 / fan_in)
```

```
  self.w = np.random.normal(0.0, std, (fan_in, self.units))
  self.b = np.zeros(self.units)

def forward(self, x):
    if self.w is None:
      self.init_weights(x.shape[-1])

    self.z = x @ self.w + self.b

    if self.activation == 'relu':
        return np.maximum(0, self.z)

    elif self.activation == 'leaky_relu':
        return np.maximum(0.01 * self.z, self.z)

    elif self.activation == 'sigmoid':
        return 1 / (1 + np.exp(-self.z))

    elif self.activation == 'tanh':
        return np.tanh(self.z)

    elif self.activation == 'softmax':
        exp_z = np.exp(self.z - np.max(self.z, axis=1, keepdims=True))
        return exp_z / np.sum(exp_z, axis=1, keepdims=True)

    elif self.activation == 'linear':
        return self.z

    else:
```

```
            raise ValueError(f"Неизвестная активация: {self.activation}")

    def derivative(self, a):
        if self.activation == 'relu':
            return (self.z > 0).astype(float)

        elif self.activation == 'leaky_relu':
            return (self.z > 0).astype(float) + 0.01 * (self.z <= 0).astype(float)

        elif self.activation == 'sigmoid':
            return a * (1 - a)

        elif self.activation == 'tanh':
            return 1 - a**2

        elif self.activation == 'linear':
            return np.ones_like(a)

        elif self.activation == 'softmax':
            return np.ones_like(a)

        return np.ones_like(a)

class Input:
    def __init__(self, shape=None):
        self.shape = shape

    def forward(self, x):
        if self.shape is not None:
            expected = self.shape if isinstance(self.shape, tuple) else (self.shape,)
```

```
            if x.shape[1:] != expected:
                x = x.reshape((x.shape[0],) + expected)
        return x


class Dropout:
  def __init__(self, rate=0.5):
    self.rate = rate
    self.mask = None
    self.training = True

  def forward(self, x):
    if not self.training:
      return x
    self.mask = (np.random.rand(*x.shape) > self.rate) / (1 - self.rate)
    return x * self.mask

class Sequential:
    def __init__(self, layers):
        self.layers = layers
        self.history = []
        self.loss_fn = None
        self.loss_deriv = None
        self.l1 = 0
        self.l2 = 0

    def compile(self, loss='mse', l1=0.0, l2=0.0):
      self.l1 = l1
      self.l2 = l2
      self.loss = loss.lower()
```

```
  if self.loss == 'mse':
    self.loss_fn = lambda y, p: np.mean((p - y) ** 2)
    self.loss_deriv = lambda y, p: (p - y)

  elif self.loss == 'mae':
    self.loss_fn = lambda y, p: np.mean(np.abs(p - y))
    self.loss_deriv = lambda y, p: np.sign(p - y)

  elif self.loss == 'binary_crossentropy':
    eps = 1e-8
    self.loss_fn = lambda y, p: -np.mean(
        y * np.log(p + eps) + (1 - y) * np.log(1 - p + eps)
    )
    self.loss_deriv = lambda y, p: (p - y)

  elif self.loss == 'categorical_crossentropy':
    eps = 1e-8
    self.loss_fn = lambda y, p: -np.mean(np.sum(y * np.log(p + eps), axis=1))
    self.loss_deriv = lambda y, p: (p - y)

def forward(self, x):
    out = x
    for layer in self.layers:
        out = layer.forward(out)
    return out

def fit(self, x_input, y_input, epochs=100, alpha=0.001, batch_size=32,
        clip_value=5.0, adaptive_alpha=False, Ee=1e-3):
    x_input = np.asarray(x_input, dtype=np.float32)
    y_input = np.asarray(y_input, dtype=np.float32).reshape(-1, 1)
```

```
if y_input.ndim == 1:
  y_input = y_input.reshape(-1, 1)

n_samples = x_input.shape[0]
self.history = []

for epoch in range(epochs):
    indices = np.random.permutation(n_samples)
    x_shuffled = x_input[indices]
    y_shuffled = y_input[indices]

    epoch_loss = 0

    for i in range(0, n_samples, batch_size):
        x = x_shuffled[i:i+batch_size]
        y = y_shuffled[i:i+batch_size]

        if adaptive_alpha:
          alpha_t = 1.0 / (1 + np.sum(x ** 2))
        else:
          alpha_t = alpha

        activations = [x]
        for layer in self.layers:
            if isinstance(layer, Dropout):
                layer.training = True
            activations.append(layer.forward(activations[-1]))

        pred = activations[-1]
```

```
loss = self.loss_fn(y, pred)

reg = 0
for layer in self.layers:
  if isinstance(layer, DenseLayer):
    reg += self.l2 * np.mean(layer.w ** 2) + self.l1 * np.mean(np.abs(layer.w))

loss += reg
epoch_loss += loss

# Обратное распространение
delta = self.loss_deriv(y, pred)
for l in range(len(self.layers) - 1, -1, -1):
    layer = self.layers[l]

    if isinstance(layer, Input):
        continue

    if isinstance(layer, Dropout):
        delta *= layer.mask
        continue

    a_prev = activations[l]
    da = layer.derivative(activations[l+1])
    if da is not None:
      delta *= da

    grad_w = (a_prev.T @ delta) / x.shape[0]
    grad_b = np.mean(delta, axis=0)
```

```
            if self.l2:
              grad_w += 2 * self.l2 * layer.w
            if self.l1:
              grad_w += self.l1 * np.sign(layer.w)

            grad_w = np.clip(grad_w, -clip_value, clip_value)
            grad_b = np.clip(grad_b, -clip_value, clip_value)

            layer.w -= alpha_t * grad_w
            layer.b -= alpha_t * grad_b

            delta = delta @ layer.w.T

        epoch_loss /= (n_samples // batch_size + 1)
        self.history.append(epoch_loss)

        if epoch % 10 == 0 or epoch == epochs - 1:
            print(f"Epoch {epoch:4d} | {self.loss} = {epoch_loss:.6f}")

        if epoch_loss <= Ee:
          print(f"Достигнут критерий остановки: {epoch_loss} <= {Ee}")
          break

def predict(self, X):
    X = np.asarray(X, dtype=np.float32)

    if X.ndim == 1:
        X = X.reshape(1,-1)
```

```
    for layer in self.layers:
        if isinstance(layer, Dropout):
            layer.training = False

    return self.forward(X)

def evaluate(self, X, Y):
    if Y.ndim == 1:
        Y = Y.reshape(-1, 1)

    preds = self.forward(X)

    loss = None
    if self.loss_fn is not None:
        loss = self.loss_fn(Y, preds)

    last_activation = getattr(self.layers[-1], "activation_name", None)

    if last_activation == "sigmoid":
        y_hat = (preds >= 0.5).astype(np.float32)
        acc = np.mean(y_hat == Y)
    else:
        acc = None

    return loss, acc
```

Графики обучения Es(p), где p – номер эпохи, для всех экспериментов на одних осях координат:

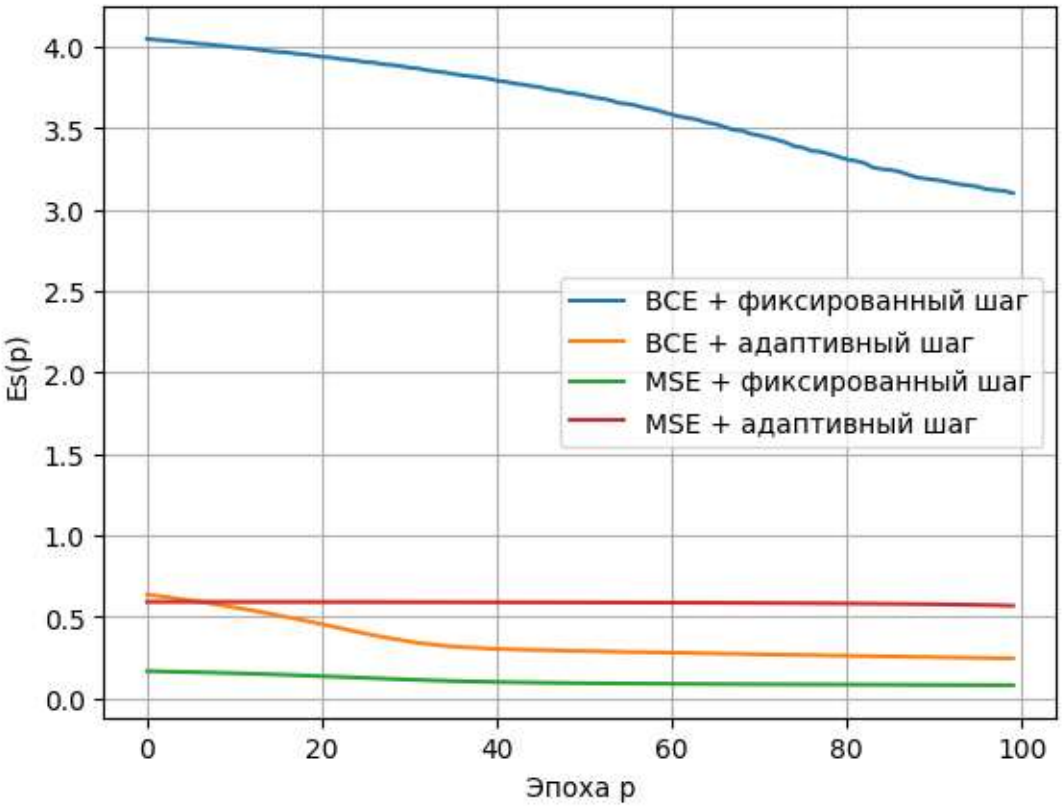


График разделяющей прямой для логистической регресии с адаптивным шагом:

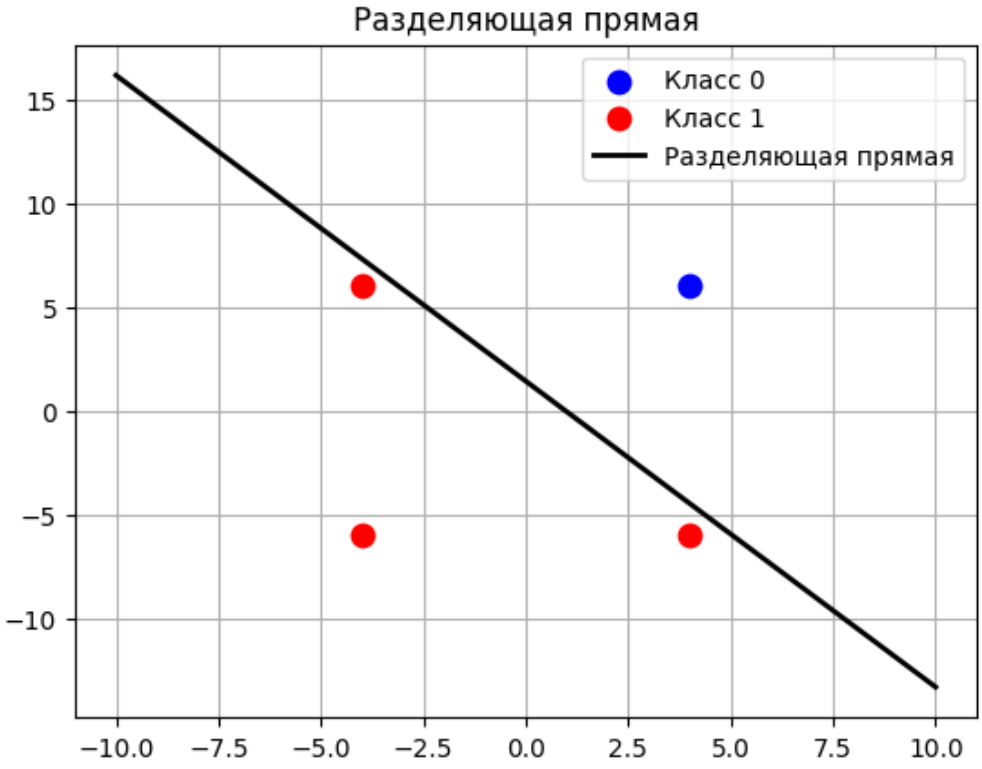
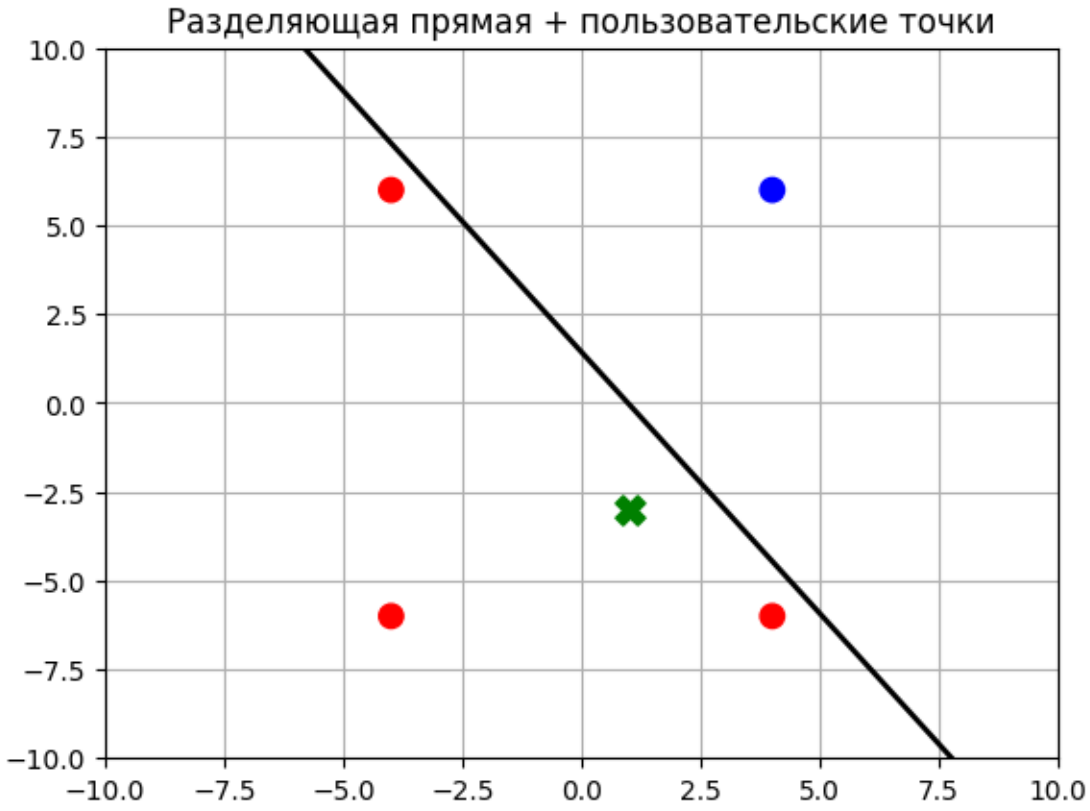


График с введёнными пользователем точками:



Вывод: В ходе выполнения данной лабораторной работы была изучена и реализована модель бинарной классификации на основе однослойной нейронной сети (логистического регрессора). Ключевым отличием от предыдущих работ стало применение сигмоидной функции активации в паре с функцией потерь "бинарная кросс-энтропия" (BCE), что позволило подойти к задаче классификации с вероятностной точки зрения. Были проведены эксперименты по сравнению скорости сходимости и качества классификации для моделей, использующих BCE и среднеквадратичную ошибку (MSE), как с фиксированным, так и с адаптивным шагом обучения.

Основные результаты и выводы по проделанной работе:

1. Превосходство BCE над MSE в задаче классификации.
2. Влияние адаптивного шага обучения. Адаптивный шаг обучения продемонстрировал свою эффективность для обеих функций потерь, но особенно в сочетании с BCE.