МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Комп’ютерний практикум №3**

з дисципліни «**Нейронні мережі**»

на тему: «РОЗРОБКА ПРОСТОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ЗАДАЧІ КЛАСИФІКАЦІЇ, ЗАСТОСОВУЮЧИ ГРАДІЄНТНИЙ СПУСК»

**Виконав:**

студент гр. БС-03

Затуловський Г. А.

**Перевірив:**

ас. каф. БМК Дюмін О.Д.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2023

# Варіант 6

**Практична частина:**

[6, 12, 18, 24, 30] Pima Indians Diabetes Database (діабет серед пімських індіанців, <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>)

1. Завантажити датасет відповідно до варіанту завдання. Застосувати до нього розроблену нейронну мережу. Зробити за необхідністю попередню обробку даних.
2. Порівняння з готовими рішеннями та проаналізувати результати роботи, розрахувавши метрики якості. Використати готову модель нейронної мережі з бібліотеки scikit-learn і порівняти її роботу з розробленою вручну мережею, наприклад для MLPClassifier:

...

*from sklearn.neural\_network import MLPClassifier from sklearn.metrics import classification\_report clf = MLPClassifier(hidden\_layer\_sizes=(n\_hidden,), max\_iter=1000, alpha=0.01, solver='sgd', verbose=10, random\_state=42, learning\_rate\_init=.01) clf.fit(X\_iris\_train, Y\_iris\_train.ravel())*

*# MLPClassifier з sklearn*

*Y\_iris\_pred\_sklearn = clf.predict(X\_iris\_test)*

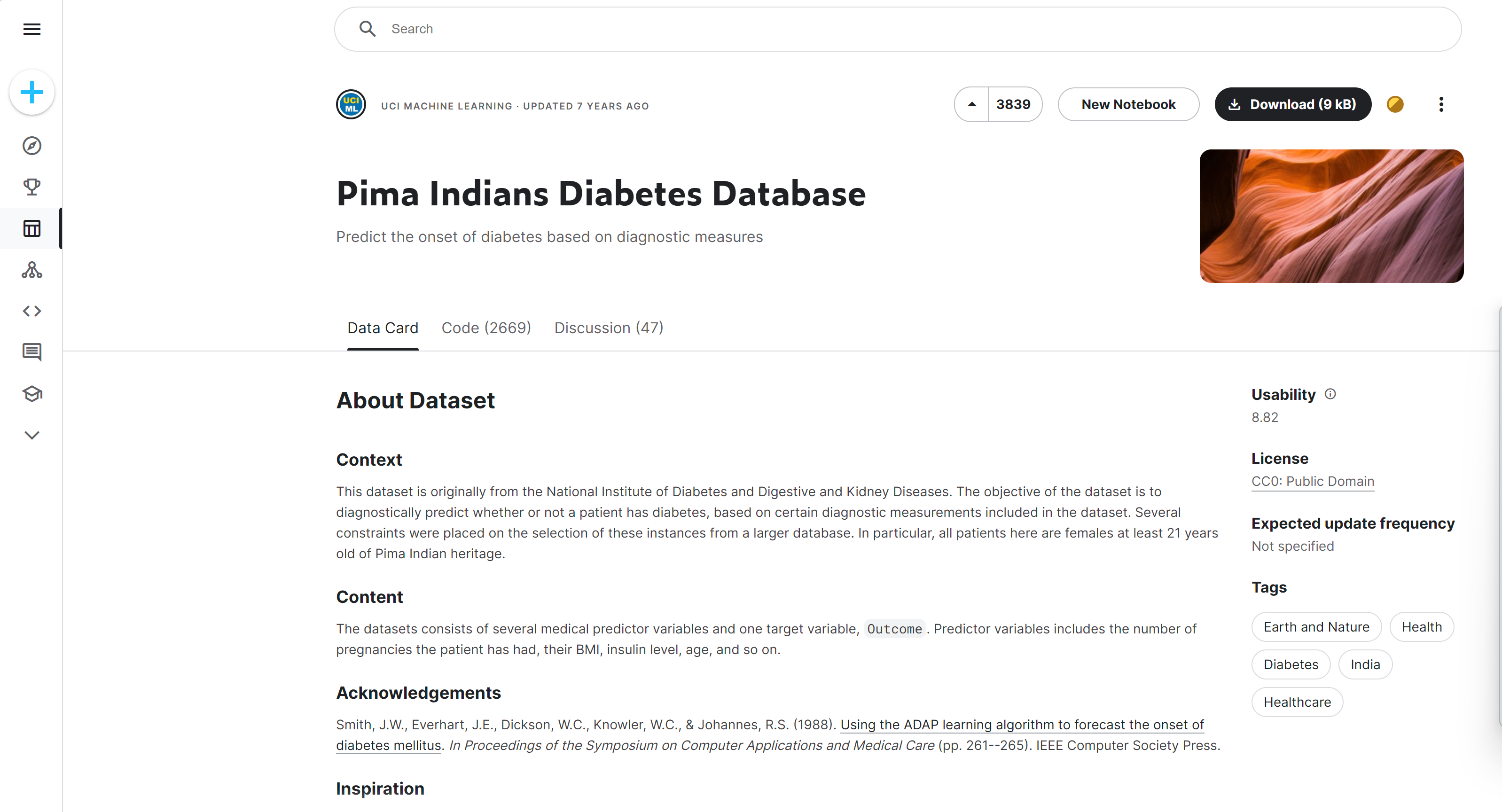
*...*

1. Оптимізація архітектури нейронної мережі. Спробувати змінити архітектуру мережі (додати більше прихованих шарів, змінити функції активації) і подивитися, як це вплине на результати. Проаналізувати результат.
2. Проекспериментувати з параметрами навчання. Змінити швидкість навчання, кількість епох або інші параметри і проаналізувати, як це впливає на процес навчання і якість отриманих прогнозів.
3. Візуалізація результатів. Побудувати графіки, які демонструють процес навчання (наприклад, зменшення помилки втрати від епохи до епохи) або візуалізують вихід мережі на тестовому наборі даних.

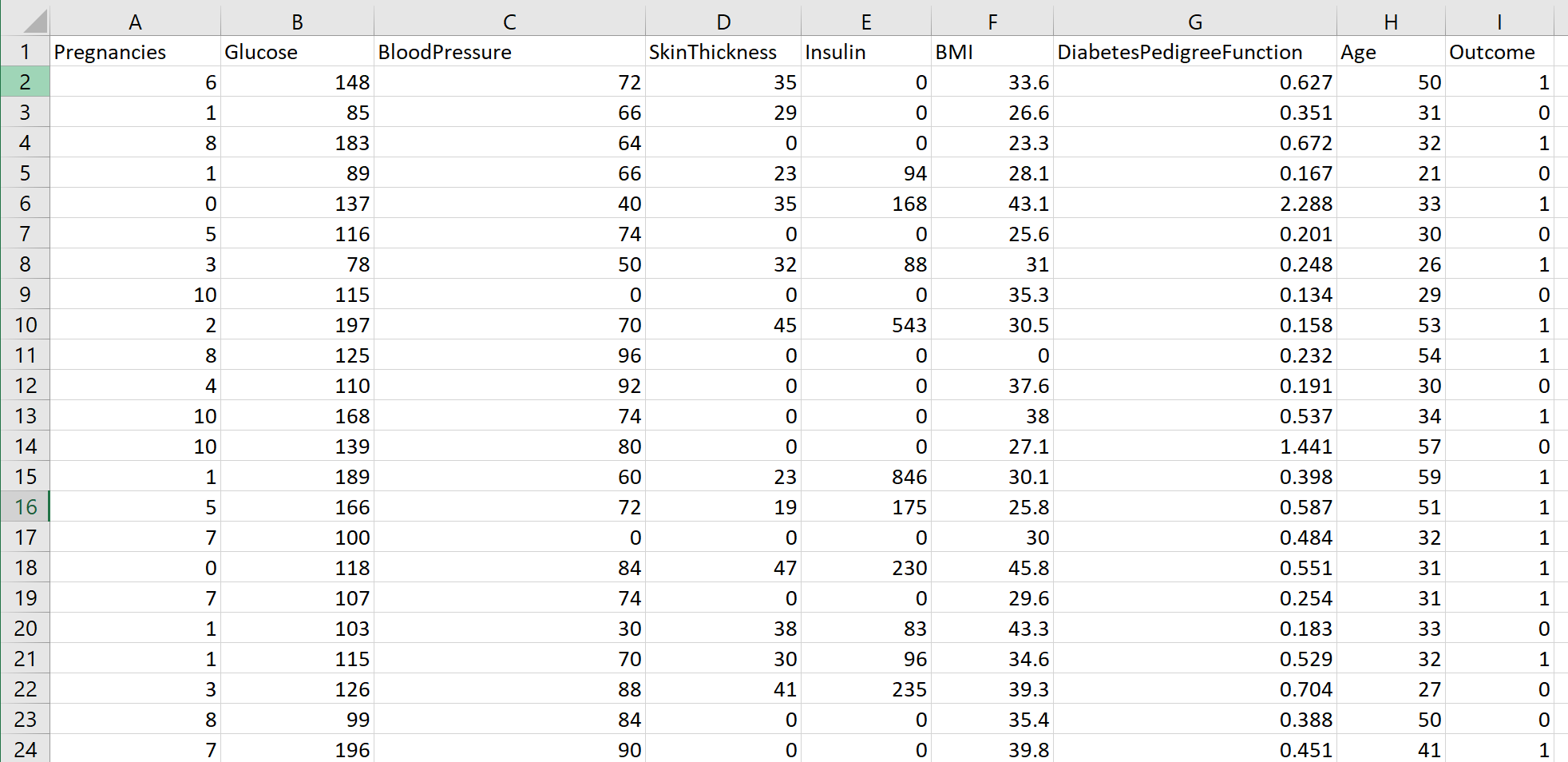
1. Аналіз помилок. Подивіться на приклади, які мережа класифікувала неправильно. Чи можна визначити якісь закономірності в цих помилках? Це може допомогти вам зрозуміти, як покращити вашу модель.
2. Підготувати звіт.

**Хід роботи:**

Завантаження датасету відповідно до варіанту завдання



База даних у Excel



Створення попередньої обробки даних

from sklearn.preprocessing import StandardScaler  
import numpy as np  
from sklearn import datasets  
import openpyxl as op  
import pandas as pd  
# load data  
dataset = pd.read\_excel('diabetes.xlsx')  
  
X= dataset.iloc[:, :8].to\_numpy()  
Y= dataset.iloc[:, 8].to\_numpy()  
norm = StandardScaler()  
X = norm.fit\_transform(X)  
  
Y = Y.reshape(-1,1)  
X=np.array(X)  
Y=np.array(Y)

Застосування розроблену нейронну мережу.

from sklearn.preprocessing import StandardScaler  
import numpy as np  
from sklearn import datasets  
import openpyxl as op  
import pandas as pd  
# load data  
dataset = pd.read\_excel('diabetes.xlsx')  
  
X= dataset.iloc[:, :8].to\_numpy()  
Y= dataset.iloc[:, 8].to\_numpy()  
norm = StandardScaler()  
X = norm.fit\_transform(X)  
  
Y = Y.reshape(-1,1)  
X=np.array(X)  
Y=np.array(Y)  
  
# Визначаємо структуру мережі  
n\_input = 8  
n\_hidden = 768  
n\_output = 1  
# Ваги та зміщення для першого шару (вхідний -> прихований)  
W1 = np.random.randn(n\_input, n\_hidden)  
b1 = np.zeros((1, n\_hidden))  
# Ваги та зміщення для другого шару (прихований -> вихідний)  
W2 = np.random.randn(n\_hidden, n\_output)  
b2 = np.zeros((1, n\_output))  
  
def sigmoid(x):  
 return 1 / (1 + np.exp(-x))  
def sigmoid\_derivative(x):  
 return x \* (1 - x)  
# Прямий прохід  
def forward\_pass(X):  
 Z1 = np.dot(X, W1) + b1  
 A1 = sigmoid(Z1)  
 Z2 = np.dot(A1, W2) + b2  
 A2 = sigmoid(Z2)  
 return Z1, A1, Z2, A2  
# Градієнтний спуск та зворотній прохід  
def backward\_pass(X, Y, Z1, A1, Z2, A2, W1, b1, W2, b2, learning\_rate=0.01):  
 m = X.shape[0]  
 dZ2 = A2 - Y  
 dW2 = 1/m \* np.dot(A1.T, dZ2)  
 db2 = np.sum(dZ2, axis=0, keepdims=True)  
 dA1 = np.dot(dZ2, W2.T)  
 dZ1 = dA1 \* sigmoid\_derivative(A1)  
 dW1 = 1/m \* np.dot(X.T, dZ1)  
 db1 = np.sum(dZ1, axis=0, keepdims=True)  
 W2 -= learning\_rate \* dW2  
 b2 -= learning\_rate \* db2  
 W1 -= learning\_rate \* dW1  
 b1 -= learning\_rate \* db1  
  
 return W1, b1, W2, b2  
# Тренування моделі  
def train\_model(X, Y, W1, b1, W2, b2, num\_iterations=1000):  
 for i in range(num\_iterations):  
 Z1, A1, Z2, A2 = forward\_pass(X)  
 W1, b1, W2, b2 = backward\_pass(X, Y, Z1, A1, Z2, A2, W1, b1, W2, b2)  
  
 return W1, b1, W2, b2  
  
W1, b1, W2, b2 = train\_model(X, Y, W1, b1, W2, b2, num\_iterations=500)  
\_,\_,\_,predictions = forward\_pass(X)  
predictions = (predictions > 0.5).astype(int)  
print(predictions)  
accuracy = np.mean(predictions == Y)  
print("Accuracy:", accuracy)

Використати готову модель нейронної мережі з бібліотеки scikit-learn(MLPClassifier)

from sklearn.neural\_network import MLPClassifier  
from sklearn.metrics import classification\_report  
clf = MLPClassifier(hidden\_layer\_sizes=(n\_hidden,), max\_iter=500, alpha=0.01, solver='sgd', verbose=10, random\_state=42, learning\_rate\_init=.01)  
clf.fit(X, Y.ravel())  
# MLPClassifier з sklearn  
Y\_iris\_pred\_sklearn = clf.predict(X)  
  
accuracy = np.mean(Y\_iris\_pred\_sklearn == Y)  
print("Accuracy:", accuracy)

Результати розрахованих метрики якості





Після порівняння точності, ми зможем визначити, яка модель працює краще на вашому наборі даних. У нашому випадку метрики якості нашої простої нейроної мережі краща порівняння з готовими рішеннями.

from sklearn.preprocessing import StandardScaler  
import numpy as np  
from sklearn import datasets  
import openpyxl as op  
import pandas as pd  
import matplotlib.pyplot as plt  
# load data  
dataset = pd.read\_excel('diabetes.xlsx')  
  
X= dataset.iloc[:, :8].to\_numpy()  
Y= dataset.iloc[:, 8].to\_numpy()  
norm = StandardScaler()  
X = norm.fit\_transform(X)  
  
Y = Y.reshape(-1,1)  
X=np.array(X)  
Y=np.array(Y)  
  
# Визначаємо структуру мережі  
n\_input = 8  
n\_hidden = 1000  
n\_output = 1  
# Ваги та зміщення для першого шару (вхідний -> прихований)  
W1 = np.random.randn(n\_input, n\_hidden)  
b1 = np.zeros((1, n\_hidden))  
# Ваги та зміщення для другого шару (прихований -> вихідний)  
W2 = np.random.randn(n\_hidden, n\_output)  
b2 = np.zeros((1, n\_output))  
  
def sigmoid(x):  
 return 1 / (1 + np.exp(-x))  
def sigmoid\_derivative(x):  
 return x \* (1 - x)  
# Прямий прохід  
def forward\_pass(X):  
 Z1 = np.dot(X, W1) + b1  
 A1 = sigmoid(Z1)  
 Z2 = np.dot(A1, W2) + b2  
 A2 = sigmoid(Z2)  
 return Z1, A1, Z2, A2  
# Градієнтний спуск та зворотній прохід  
def backward\_pass(X, Y, Z1, A1, Z2, A2, W1, b1, W2, b2, learning\_rate=0.01):  
 m = X.shape[0]  
 dZ2 = A2 - Y  
 dW2 = 1/m \* np.dot(A1.T, dZ2)  
 db2 = np.sum(dZ2, axis=0, keepdims=True)  
 dA1 = np.dot(dZ2, W2.T)  
 dZ1 = dA1 \* sigmoid\_derivative(A1)  
 dW1 = 1/m \* np.dot(X.T, dZ1)  
 db1 = np.sum(dZ1, axis=0, keepdims=True)  
 W2 -= learning\_rate \* dW2  
 b2 -= learning\_rate \* db2  
 W1 -= learning\_rate \* dW1  
 b1 -= learning\_rate \* db1  
  
 return W1, b1, W2, b2  
def compute\_loss(Y, predictions):  
 m = Y.shape[0]  
  
 loss = -np.mean(Y \* np.log(predictions) + (1 - Y) \* np.log(1 - predictions))  
 return loss  
  
# Тренування моделі  
def train\_model(X, Y, W1, b1, W2, b2, num\_iterations=1000):  
 loss\_values =[]  
 for i in range(num\_iterations):  
 Z1, A1, Z2, A2 = forward\_pass(X)  
 loss = compute\_loss(Y,A2)  
 loss\_values.append(loss)  
 W1, b1, W2, b2 = backward\_pass(X, Y, Z1, A1, Z2, A2, W1, b1, W2, b2)  
  
 return W1, b1, W2, b2, loss\_values  
  
W1, b1, W2, b2, loss = train\_model(X, Y, W1, b1, W2, b2, num\_iterations=1000)  
\_,\_,\_,predictions = forward\_pass(X)  
predictions = (predictions > 0.5).astype(int)  
  
accuracy = np.mean(predictions == Y)  
print("Нейронну мережа - Accuracy:", accuracy)  
plt.plot(loss)  
plt.xlabel('Iterations')  
plt.ylabel('Loss')  
plt.title('Training Loss over Time')  
plt.show()  
from sklearn.neural\_network import MLPClassifier  
from sklearn.metrics import classification\_report  
clf = MLPClassifier(hidden\_layer\_sizes=(n\_hidden,), max\_iter=1000, alpha=0.01, solver='sgd', verbose=10, random\_state=42, learning\_rate\_init=.01)  
clf.fit(X, Y.ravel())  
  
# MLPClassifier з sklearn  
Y\_iris\_pred\_sklearn = clf.predict(X)  
  
accuracy = np.mean(Y\_iris\_pred\_sklearn == Y)  
print("MLPClassifier - Accuracy:", accuracy)

Збільшимо кількість прихованих шарів з 768 до 1000

Після збільшення прихованих шарів збільшилась точність

Збільшимо кількість прихованих шарів з 768 до 500





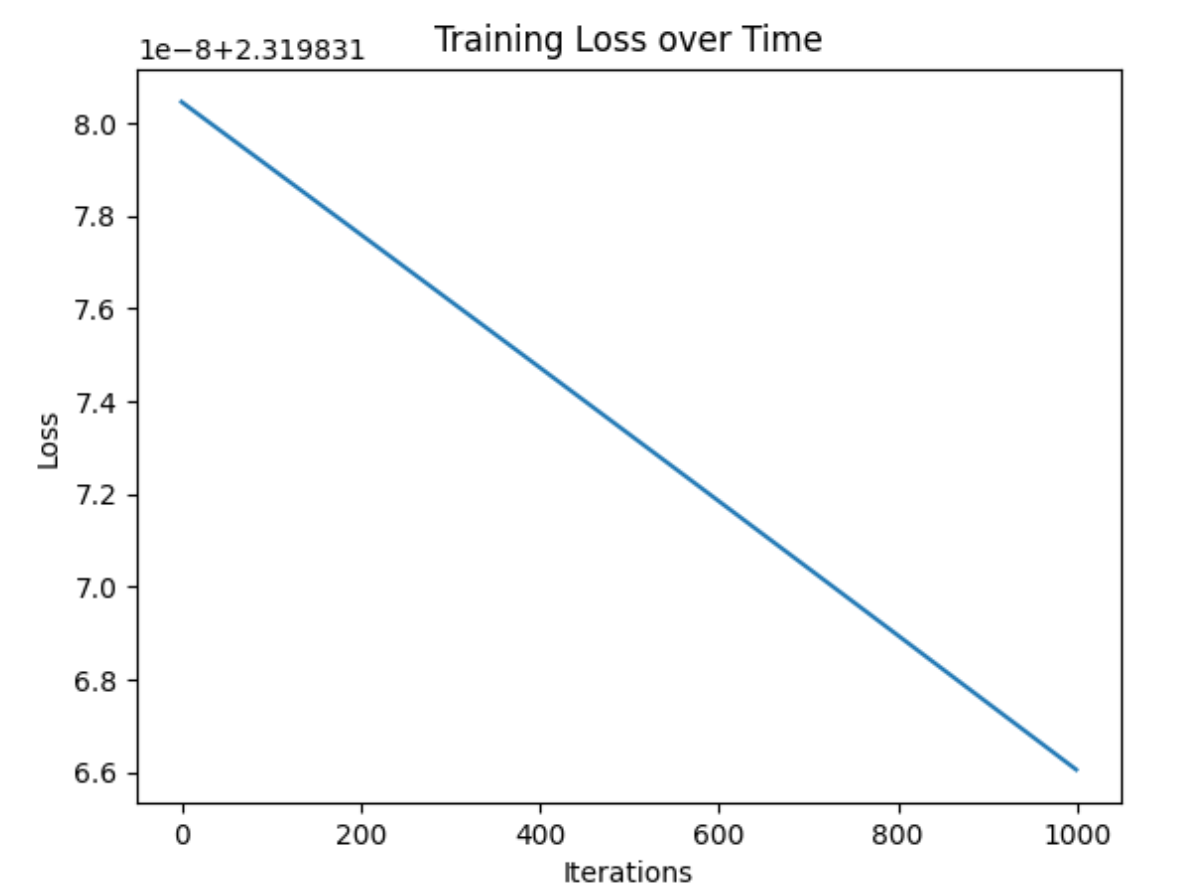
Після зменшення прихованих шарів зменшилась точність

Змінимо функцію активації з sigmoid на softmax

from sklearn.preprocessing import StandardScaler  
import numpy as np  
from sklearn import datasets  
import openpyxl as op  
import pandas as pd  
import matplotlib.pyplot as plt  
# load data  
dataset = pd.read\_excel('diabetes.xlsx')  
  
X= dataset.iloc[:, :8].to\_numpy()  
Y= dataset.iloc[:, 8].to\_numpy()  
norm = StandardScaler()  
X = norm.fit\_transform(X)  
  
Y = Y.reshape(-1,1)  
X=np.array(X)  
Y=np.array(Y)  
  
# Визначаємо структуру мережі  
n\_input = 8  
n\_hidden = 768  
n\_output = 1  
# Ваги та зміщення для першого шару (вхідний -> прихований)  
W1 = np.random.randn(n\_input, n\_hidden)  
b1 = np.zeros((1, n\_hidden))  
# Ваги та зміщення для другого шару (прихований -> вихідний)  
W2 = np.random.randn(n\_hidden, n\_output)  
b2 = np.zeros((1, n\_output))  
  
def sigmoid(x):  
 return 1 / (1 + np.exp(-x))  
def sigmoid\_derivative(x):  
 return x \* (1 - x)  
# Прямий прохід  
def Softmax(x):  
 e\_x = np.exp(x - np.max(x)) # Subtracting max(x) for numerical stability  
 return e\_x / e\_x.sum()  
def Softmax\_derivative(x):  
 s = Softmax(x)  
 return s \* (1 - s)  
def forward\_pass(X):  
 Z1 = np.dot(X, W1) + b1  
 A1 = Softmax(Z1)#sigmoid(Z1)  
 Z2 = np.dot(A1, W2) + b2  
 A2 = Softmax(Z2)#sigmoid(Z2)  
 return Z1, A1, Z2, A2  
# Градієнтний спуск та зворотній прохід  
def backward\_pass(X, Y, Z1, A1, Z2, A2, W1, b1, W2, b2, learning\_rate=0.01):  
 m = X.shape[0]  
 dZ2 = A2 - Y  
 dW2 = 1/m \* np.dot(A1.T, dZ2)  
 db2 = np.sum(dZ2, axis=0, keepdims=True)  
 dA1 = np.dot(dZ2, W2.T)  
 dZ1 = dA1 \* Softmax\_derivative(A1)#sigmoid\_derivative(A1)  
 dW1 = 1/m \* np.dot(X.T, dZ1)  
 db1 = np.sum(dZ1, axis=0, keepdims=True)  
 W2 -= learning\_rate \* dW2  
 b2 -= learning\_rate \* db2  
 W1 -= learning\_rate \* dW1  
 b1 -= learning\_rate \* db1  
  
 return W1, b1, W2, b2  
def compute\_loss(Y, predictions):  
 m = Y.shape[0]  
  
 loss = -np.mean(Y \* np.log(predictions) + (1 - Y) \* np.log(1 - predictions))  
 return loss  
  
# Тренування моделі  
def train\_model(X, Y, W1, b1, W2, b2, num\_iterations=1000):  
 loss\_values =[]  
 for i in range(num\_iterations):  
 Z1, A1, Z2, A2 = forward\_pass(X)  
 loss = compute\_loss(Y,A2)  
 loss\_values.append(loss)  
 W1, b1, W2, b2 = backward\_pass(X, Y, Z1, A1, Z2, A2, W1, b1, W2, b2)  
  
 return W1, b1, W2, b2, loss\_values  
  
W1, b1, W2, b2, loss = train\_model(X, Y, W1, b1, W2, b2, num\_iterations=1000)  
\_,\_,\_,predictions = forward\_pass(X)  
predictions = (predictions > 0.5).astype(int)  
  
accuracy = np.mean(predictions == Y)  
print("Нейронну мережа - Accuracy:", accuracy)  
plt.plot(loss)  
plt.xlabel('Iterations')  
plt.ylabel('Loss')  
plt.title('Training Loss over Time')  
plt.show()  
from sklearn.neural\_network import MLPClassifier  
from sklearn.metrics import classification\_report  
clf = MLPClassifier(hidden\_layer\_sizes=(n\_hidden,), max\_iter=1000, alpha=0.01, solver='sgd', verbose=10, random\_state=42, learning\_rate\_init=.01)  
clf.fit(X, Y.ravel())  
  
# MLPClassifier з sklearn  
Y\_iris\_pred\_sklearn = clf.predict(X)  
  
accuracy = np.mean(Y\_iris\_pred\_sklearn == Y)  
print("MLPClassifier - Accuracy:", accuracy)

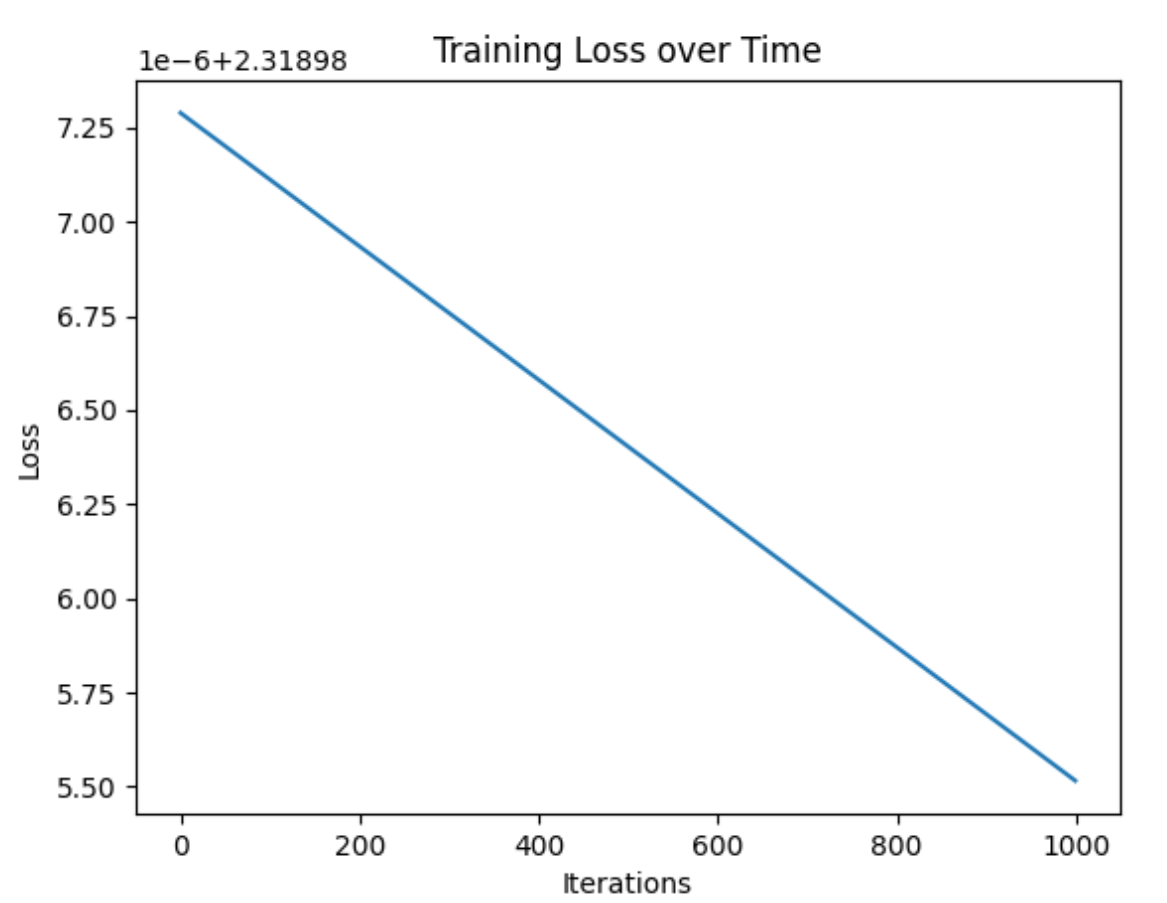
Змінимо функцію активації з sigmoid на softmax тз прихованими 500 шарів





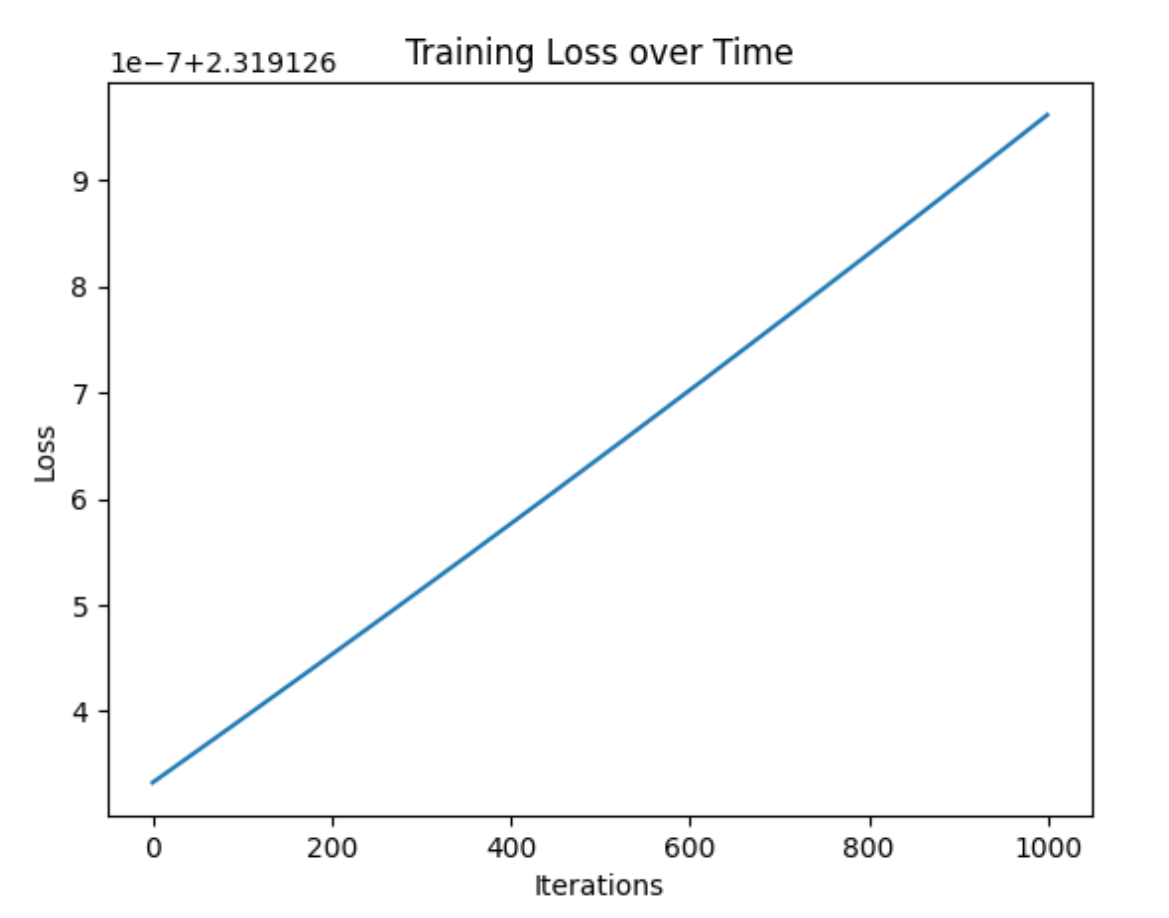
Змінимо функцію активації з sigmoid на softmax тз прихованими 768 шарів





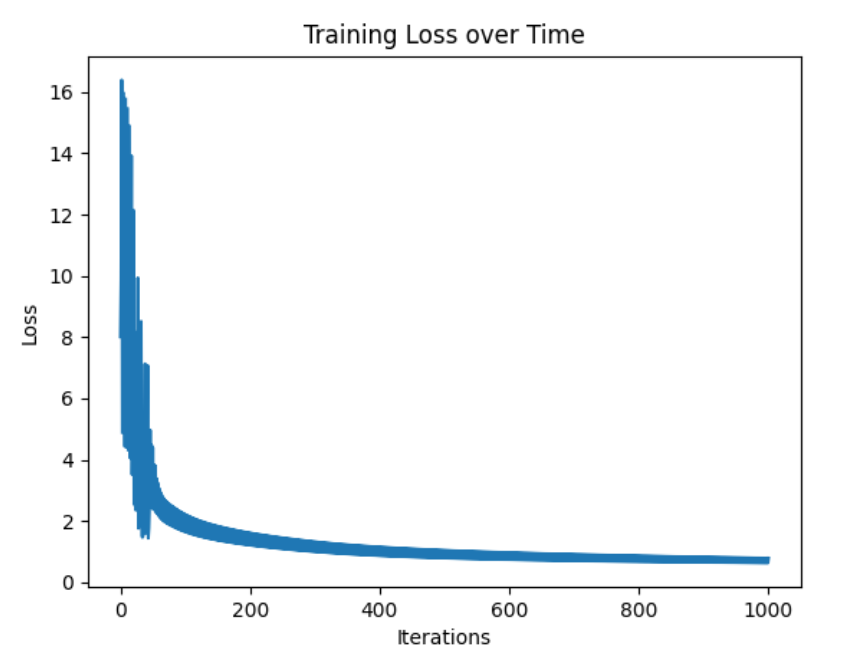
Змінимо функцію активації з sigmoid на softmax тз прихованими 1000 шарів

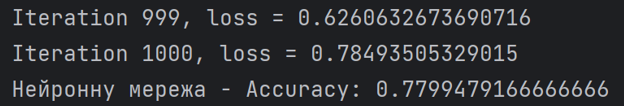


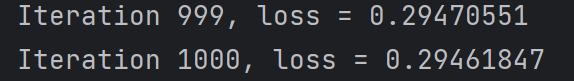


Обидві функції активації є важливими для нейронних мереж та глибокого навчання, і вони використовуються в залежності від характеру задачі. Sigmoid використовується для бінарної класифікації, тоді як Softmax використовується для мультикласової класифікації. Тому для нашої бази данних краще використовувати Sigmoid

При прихованими 768 шарів та функції активації Sigmoid, швидкисть навчання 0.01, кількість епох 1000

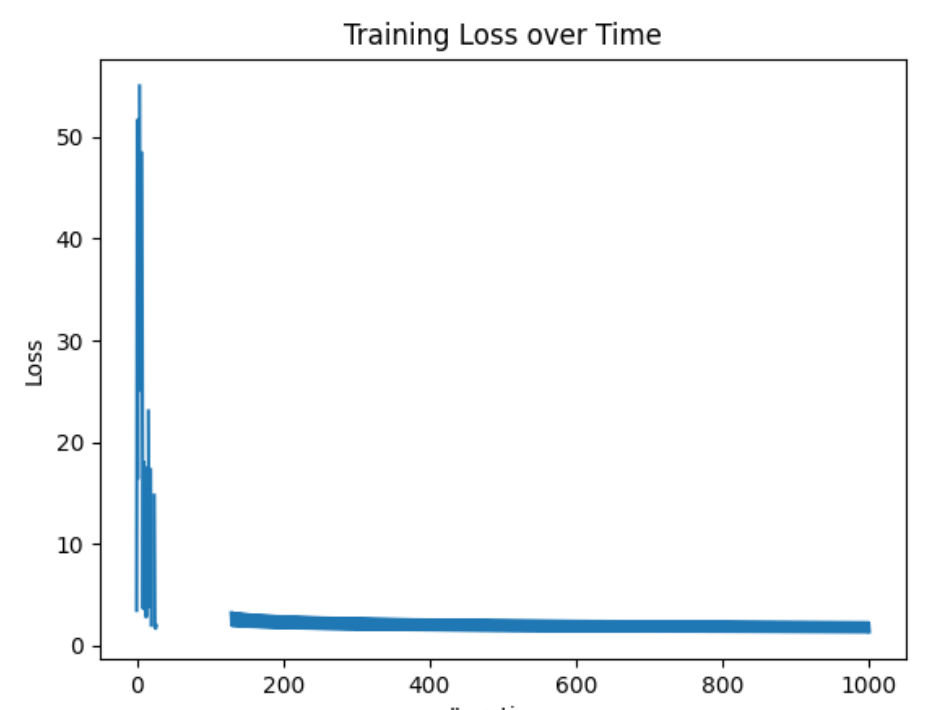


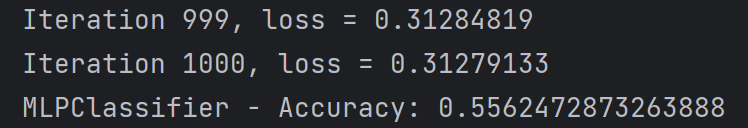
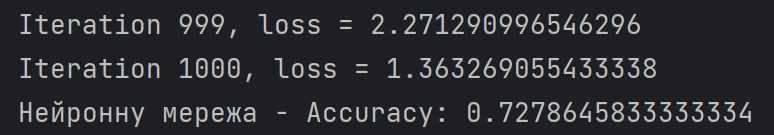




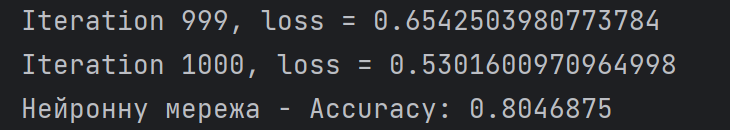


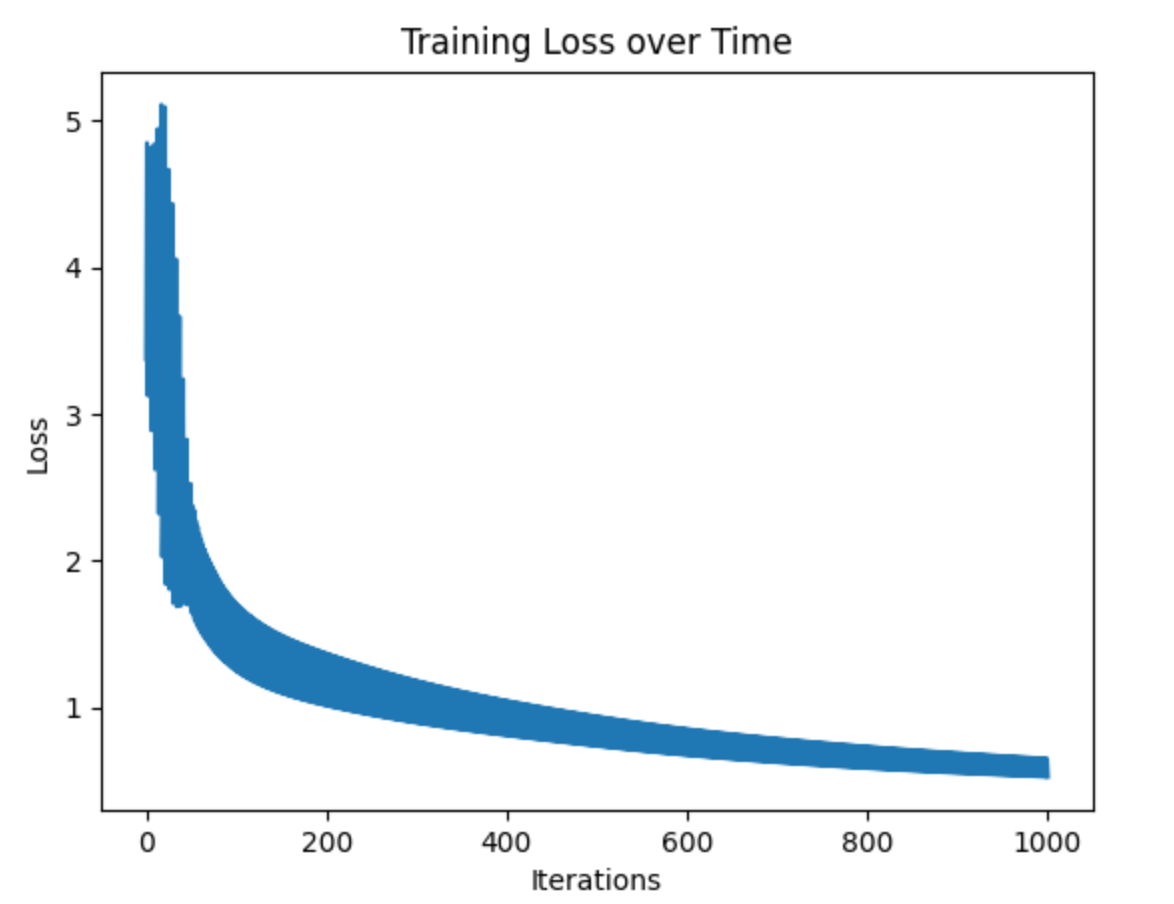
Зміним швидкість навчання, швидкисть навчання 0.05, кількість епох 1000

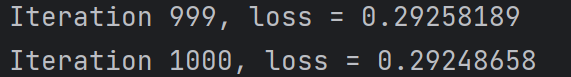




Зміним швидкість навчання, швидкисть навчання 0.005, кількість епох 1000

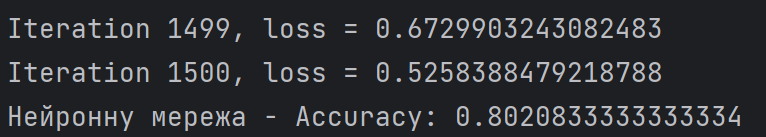


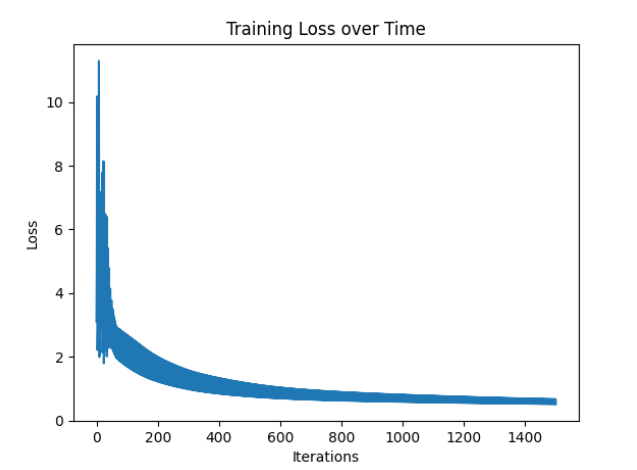


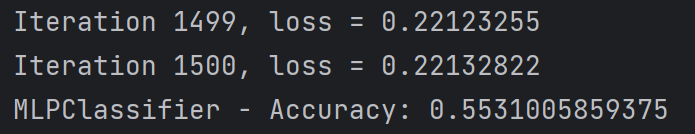




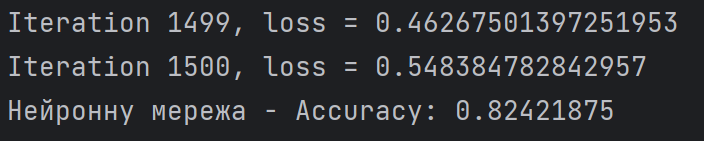
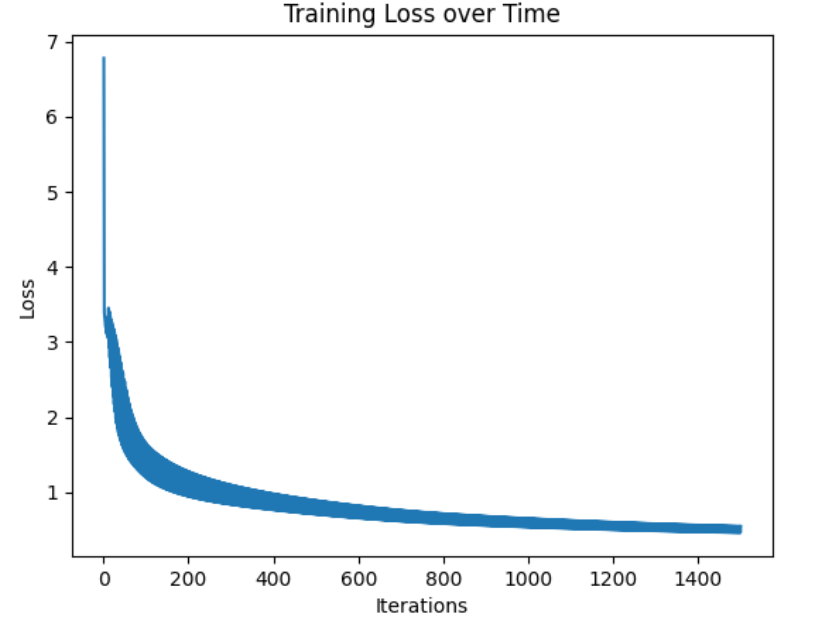
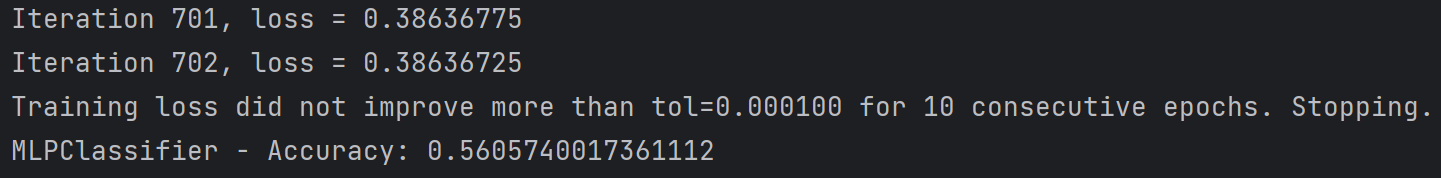
Змінимо кількість епох , швидкисть навчання 0.01, кількість епох 1500



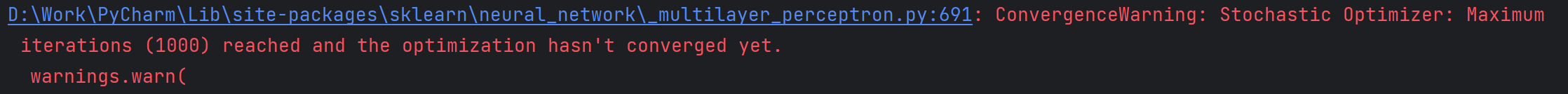




кількість епох , швидкисть навчання 0.005, кількість епох 1500

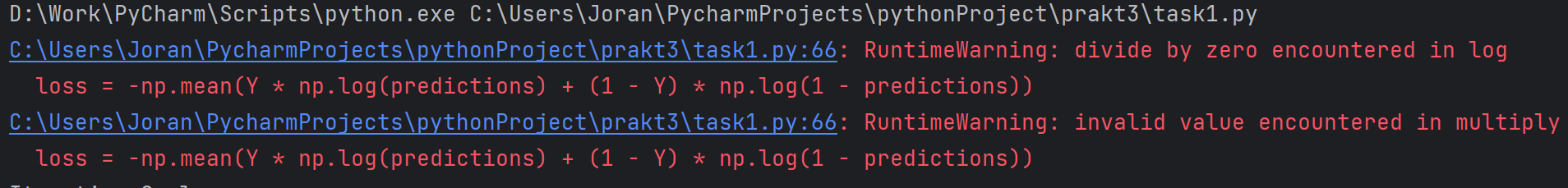
Аналіз помилок



Ця помилка "ConvergenceWarning" повідомляє про те, що процес навчання нейронної мережі не збігся до оптимального рішення за максимальну кількість ітерацій (1000 ітерацій у вашому випадку). Основна причина цієї помилки полягає в тому, що оптимізатор не досягнув точки збіжності або не знайшов оптимальних параметрів мережі. Ось кілька кроків для вирішення цієї проблеми:

Змініть швидкість навчання (learning rate): Збільшіть або зменшіть швидкість навчання. Надто велика швидкість навчання може призвести до перерозходження, а надто мала - до повільного навчання. Спробуйте різні значення і спостерігайте за збіжністю.

Збільшіть максимальну кількість ітерацій: Збільшіть кількість максимальних ітерацій, дозволивши оптимізатору більше часу для навчання. Однак, будьте обережні, оскільки це може збільшити час навчання.



Ці попередження "RuntimeWarning" вказують на те, що в вашій функції compute\_loss виникає ділення на нуль (divide by zero) та невірне значення (invalid value) під час обчислення втрат. Це зазвичай стається, коли значення predictions або 1 - predictions стають дуже близькими до 0 або 1, що призводить до невизначених математичних операцій. Для вирішення помилки можна використати стабільні математичних операцій: Замість обчислення np.log(predictions) і np.log(1 - predictions) безпосередньо, використовуйте стабільні операції, які уникають нульового ділення та невизначених значень. Наприклад, використовуйте np.log(np.maximum(predictions, 1e-15)) та np.log(np.maximum(1 - predictions, 1e-15)), де 1e-15 - невелике значення близьке до нуля.

**Контрольні запитання:**

1. **Що таке нейронна мережа?**

Нейромережа (Штучна нейронна мережа або Нейронка) — це математична модель, яка імітує структуру та функціонування біологічних нейронних мереж з метою вирішення різноманітних задач, таких як класифікація, регресія, прогнозування та генерація.

1. **Які її основні складові?**

Основні складові нейронної мережі включають:

1. Вхідний шар (Input Layer): Це шар, який приймає вхідні дані. Кількість нейронів у вхідному шарі зазвичай відповідає кількості ознак вхідних даних.

2. Приховані шари (Hidden Layers): Це шари між вхідним і вихідним шарами. Нейронна мережа може мати один або кілька прихованих шарів. Кожен прихований шар складається з декількох нейронів, кожен з яких виконує певну лінійну та не лінійну обчислювальну операцію.

3. Вихідний шар (Output Layer): Це останній шар у нейронній мережі, який видає результати передбачення. Кількість нейронів у вихідному шарі відповідає кількості цільових класів для задачі класифікації.

4. Ваги та зміщення (Weights and Biases): Ваги - це параметри, що визначають силу впливу кожного вхідного нейрона на кожен вихідний нейрон. Зміщення - це додаткові параметри, які дозволяють кожному нейрону робити більш гнучкі передбачення.

5. Функції активації (Activation Functions): Функції активації використовуються для введення нелінійності в нейронну мережу, що дозволяє їй моделювати більш складні залежності. Найпоширенішими функціями активації є ReLU (Rectified Linear Unit), сигмоїдна функція, та softmax-функція.

1. **Що таке градієнтний спуск і як він застосовується в нейронних мережах?**

Градієнтний спуск – це оптимізаційний алгоритм, що використовується для знаходження мінімального значення функції (часто використовується в контексті мінімізації функції втрат), шляхом ітеративного переміщення в напрямку найшвидшого спаду – протилежного до градієнта. У контексті навчання нейронних мереж градієнтний спуск використовується для оновлення ваги (W) та зміщення (b) нейронів з метою мінімізації функції втрат (L), що обчислює різницю між реальними та передбаченими вихідними даними. Основні кроки градієнтного спуску в нейронних мережах включають:

1. Ініціалізація параметрів: Ваги та зміщення нейронів “ініціалізуються” (надалі – задаються початкові значення) деякими випадковими числами.

2. Пряме розповсюдження: Вхідні дані передаються через мережу, використовуючи поточні значення ваг та зміщень, щоб обчислити передбачення.

3. Обчислення втрат: Розраховується функція втрат, що представляє різницю між реальними та передбаченими вихідними даними.

4. Зворотне розповсюдження: Градієнти функції втрат розраховуються щодо ваг та зміщень і розповсюджуються назад через мережу.

5. Оновлення параметрів: Ваги та зміщення оновлюються шляхом віднімання продукту градієнтів та швидкості навчання від поточних значень.

Цей процес повторюється (кроки 2-5), поки функція втрат не стане достатньо малою або до тих пір, поки не буде досягнуто максимальної кількості ітерацій.

1. **Які основні відмінності між бінарною та багатокласовою класифікацією в контексті нейронних мереж? Які активаційні функції використовуються в нейронних мережах і чому?**

Основні відмінності між бінарною та багатокласовою класифікацією в контексті нейронних мереж:

Бінарна класифікація: В бінарній класифікації є всього два можливих класи (наприклад, 0 або 1, "так" або "ні"). Вихід нейронної мережі призначений для передбачення ймовірностей належності до одного з цих двох класів. Зазвичай, використовують один вихідний нейрон з активацією Sigmoid, де поріг 0.5 розділяє два класи.

Багатокласова класифікація: В багатокласовій класифікації існує більше ніж два можливих класи (більше ніж два різних значення, наприклад, класифікація зображень об'єктів, де кожне зображення може належати до одного з багатьох класів). Вихід нейронної мережі призначений для передбачення ймовірностей належності до кожного класу. Зазвичай використовують Softmax активацію на останньому шарі, яка розподіляє ймовірності для всіх класів так, що сума всіх ймовірностей дорівнює 1.

Активаційні функції в нейронних мережах:

Бінарна класифікація: У бінарній класифікації зазвичай використовується активація Sigmoid. Функція Sigmoid призначена для обчислення ймовірності належності до класу 1, і її вихід знаходиться в діапазоні [0, 1]. Це дозволяє моделі робити бінарні передбачення (клас 0 або клас 1).

Багатокласова класифікація: У багатокласовій класифікації зазвичай використовується Softmax активація. Функція Softmax обчислює ймовірності належності до всіх можливих класів, і її вихід розподіляє ці ймовірності так, що сума всіх ймовірностей дорівнює 1. Це дозволяє моделі робити передбачення для багатьох класів одночасно.

1. **Що таке функція втрат в контексті нейронних мереж? Наведіть приклади функцій втрат для задач класифікації.**

Функція втрат (або функція втрати) в контексті нейронних мереж це функція, яка визначає, наскільки добре модель нейронної мережі працює під час навчання на певній задачі. Функція втрати дозволяє оцінювати, наскільки відповіді моделі відрізняються від правильних відповідей (заздалегідь відомих відповідей) та визначає, які параметри (ваги) нейронної мережі слід змінювати під час навчання для покращення її результатів.

Приклад

Функція втрат:

def compute\_loss(Y, predictions):

loss = -np.mean(Y \* np.log(predictions) + (1 - Y) \* np.log(1 - predictions)) return loss

1. **Як обрати архітектуру нейронної мережі для конкретної задачі класифікації?**

Обираючи архітектуру нейронної мережі для конкретної задачі класифікації, важливо враховувати кілька факторів та кроків:Розмір вхідних даних,Розмір виходів (кількість класів),архітектурні рішення,глибина мережі,функції активації,регуляризація,функція втрати,оптимізатор,гіперпараметри та налаштування

1. **Які основні кроки для тренування нейронної мережі?**

**основні кроки для тренування нейронної мережі:**

1. Збір та підготовка даних:

Зберіть та підготуйте дані для навчання, включаючи навчальний набір даних та набір для перевірки (тестові дані). Виконайте попередню обробку даних, таку як нормалізація, видалення викидів та розбиття на пакети (batching).

1. Визначення архітектури мережі:

Виберіть архітектуру мережі, включаючи кількість шарів, кількість нейронів в кожному шарі та типи активаційних функцій. Визначте функцію втрати, яку ви будете використовувати для оцінки різниці між прогнозами та правильними відповідями.

1. Ініціалізація параметрів:

Ініціалізуйте ваги та зміщення (параметри) мережі, зазвичай випадковими значеннями.

1. Прямий та зворотній прохід:

Виконайте прямий прохід (forward pass), обчислюючи вихід мережі на основі вхідних даних та поточних параметрів.Розрахуйте значення функції втрати, порівнюючи виходи мережі з правильними відповідями.Виконайте зворотній прохід (backward pass), обчислюючи градієнти функції втрати відносно параметрів мережі.

1. Оновлення параметрів:

Виконайте оновлення параметрів за допомогою градієнтного спуску або іншого методу оптимізації з урахуванням градієнтів, обчислених на попередньому кроці.Повторюйте цей процес протягом багатьох ітерацій (епох) для навчання мережі.

1. Оцінка та валідація:

Виконайте оцінку продуктивності моделі на навчальному наборі даних, а також на окремому тестовому наборі, якщо він є.Враховуйте метрики якості, такі як точність, втрати, F1-оцінка та інші, для оцінки результатів моделі.

1. Налаштування гіперпараметрів:

При необхідності налаштуйте гіперпараметри мережі, такі як швидкість навчання, кількість епох, розмір пакетів тощо.

1. Збереження моделі:

Збережіть навчену модель для подальшого використання та інференсу (прогнозування).

1. **Як використовується Keras для розробки та тренування нейронних мереж?**

Keras - це відкрита бібліотека машинного навчання, яка використовується для розробки та тренування нейронних мереж. Keras можна використовувати для вирішення широкого кола завдань машинного навчання, зокрема:

Класифікація: Keras можна використовувати для класифікації об'єктів, наприклад, для розпізнавання зображень або тексту.

Регресія: Keras можна використовувати для прогнозування значень, наприклад, для прогнозування цін на акції або погоди.

Узагальнення: Keras можна використовувати для навчання моделей, які можуть узагальнювати на нові дані.

1. **Як оцінити ефективність моделі нейронної мережі?**

Основні метрики, які використовуються для оцінки результатів класифікації, включають:

• Точність (Accuracy): це відсоток випадків, для яких модель правильно передбачила клас.

• Матриця помилок (Confusion Matrix): це таблиця, яка показує, скільки випадків було правильно або неправильно класифіковано для кожного класу.

• Precision, Recall, F1-score: ці метрики допомагають оцінити, наскільки добре модель працює для кожного класу, особливо в багатокласових задачах або у випадках, коли набір даних незбалансований.

1. **Які проблеми можуть виникнути під час тренування нейронної мережі і як їх вирішити?**

Перенавчання (Overfitting):

Проблема: Мережа навчається "запам'ятовувати" тренувальні дані, а не вивчає загальний закономірності, і тому не працює добре на нових даних.

Рішення: Використовуйте регуляризацію, яка допомагає обмежити ваги, або збільшіть обсяг тренувальних даних. Також, використовуйте перевірку та тестові дані для контролю над перенавчанням.

Недонавчання (Underfitting)

Проблема: Модель недостатньо складна, щоб вивчити закономірності в даних.

Рішення: Збільшіть складність моделі, додавши більше нейронів або шарів. Також, перевірте, чи ваши дані правильно підготовлені.

Експлозія та гасіння градієнта (Gradient Explosion and Vanishing Gradient):

Проблема: Градієнти можуть стати надзвичайно великими або дуже малими, що гальмує процес навчання.

Рішення: Використовуйте методи ініціалізації ваг, такі як Xavier (Glorot) і He ініціалізації, щоб стабілізувати градієнти. Також, використовуйте функції активації, які не схильні до цих проблем, наприклад, ReLU.

Багатокласова класифікація та дисбаланс класів:

Проблема: В задачах багатокласової класифікації деякі класи можуть бути представлені дуже мало даними, що призводить до дисбалансу класів.

Рішення: Використовуйте взважену функцію втрати, щоб врахувати дисбаланс класів. Також, варто використовувати методи підвищення (oversampling) або зменшення (undersampling) класів, які допоможуть збалансувати набір даних.

Обчислювальні ресурси:

Проблема: Навчання складних мереж може вимагати великих обчислювальних ресурсів, іноді недосяжних для користувачів.

Рішення: Використовуйте оптимізовані бібліотеки та обчислювальні ресурси, такі як графічні процесори (GPU) або спеціалізовані середовища навчання глибоких мереж.

Підбір гіперпараметрів:

Проблема: Вибір правильних гіперпараметрів (швидкість навчання, кількість нейронів, функції активації тощо) може бути складним завданням.

Рішення: Використовуйте крос-валідацію та підбирайте гіперпараметри, які максимізують продуктивність моделі на тестовому наборі.

Пошук та архітектура мережі:

Проблема: Вибір правильної архітектури мережі може бути завданням "перебору".

Рішення: Використовуйте знання про типи задач та архітектурні прийоми, які підходять для цих задач. Також, використовуйте засоби автоматизованого пошуку архітектур.

**Висновок:** Ми ознайомилися як працюють прості нейронні мережі для задачі класифікації; як застосувати градієнтний спуск для оптимізації параметрів нейронної мережі; розробили та натренували просту нейронну мережу для бінарної та багатокласової класифікації, використовуючи Python; зрозуміли принцип роботи активаційних функцій та їхнє використання в нейронних мережах; зрозуміли, як обирати архітектуру нейронної мережі в залежності від конкретної задачі.