

Сравнительный анализ производительности фреймворков TensorFlow, PyTorch и Keras при решении задачи классификации на Python

Гиренко Александр Евгеньевич

Приамурский государственный университет им.Шолом-Алейхема

Студент

Аннотация

В статье представлен сравнительный анализ производительности трёх фреймворков для глубокого обучения — TensorFlow/Keras, PyTorch и ручной реализации нейросети на NumPy — при решении задачи классификации набора данных «Ирисы Фишера». Эксперимент проводился на идентичных условиях: архитектура нейросети (один скрытый слой из 10 нейронов, функция активации ReLU, выходной слой с softmax), оптимизатор Adam, функция потерь — категориальная кросс-энтропия. Сравнение выполнялось по критериям времени обучения, точности классификации и сложности реализации. Результаты показали, что ручная реализация на NumPy оказалась самой быстрой (0.37 секунды), обе реализации на NumPy и PyTorch достигли максимальной точности (100%), а TensorFlow/Keras показал наихудшее время обучения (27.20 секунд) при адекватной точности (93.33%). Полученные результаты позволяют сделать обоснованный выбор фреймворка в зависимости от приоритетов разработчика: скорость (NumPy), точность и гибкость (PyTorch) или простота реализации (TensorFlow/Keras).

Ключевые слова: машинное обучение, нейронные сети, TensorFlow, PyTorch, Keras, сравнительный анализ, классификация, производительность.

Comparative performance analysis of TensorFlow, PyTorch and Keras frameworks for classification task in Python

Girenko Aleksandr Evgen'evich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article presents a comparative performance analysis of three deep learning frameworks — TensorFlow/Keras, PyTorch, and a manual NumPy-based neural network implementation — on the Iris dataset classification task. The experiment was conducted under identical conditions: neural network architecture (one hidden layer with 10 neurons, ReLU activation function, softmax output layer), Adam optimizer, categorical cross-entropy loss function. The comparison was performed using three criteria: training time, test set classification accuracy, and implementation complexity. The results showed that the manual NumPy implementation was the fastest (0.37 seconds), both NumPy and PyTorch achieved

maximum accuracy (100%), and TensorFlow/Keras showed the worst training time (27.20 seconds) with adequate accuracy (93.33%). The obtained results allow an informed choice of framework depending on developer priorities: speed (NumPy), accuracy and flexibility (PyTorch), or ease of implementation (TensorFlow/Keras).

Keywords: machine learning, neural networks, TensorFlow, PyTorch, Keras, comparative analysis, classification, performance.

1. Введение

1.1 Актуальность

В современном мире машинного обучения выбор инструментария для разработки нейросетевых моделей является критически важным этапом любого проекта. Наиболее популярными фреймворками являются TensorFlow (с высокоуровневым API Keras), PyTorch, а также возможность реализации нейросетей «с нуля» на библиотеке NumPy. Актуальность данного исследования обусловлена необходимостью систематизации знаний о производительности различных подходов при решении классической задачи классификации.

1.2 Обзор исследований

В работе Р.Л. Астапова и Р.М. Мухамедеевой рассматривается автоматизация подбора параметров машинного обучения и обучение модели, описываются подходы к оптимизации гиперпараметров с использованием кросс-валидации [1]. Н.В. Суханова и С.А. Шептунов в патенте представляют отказоустойчивую нейронную сеть, способы обнаружения отказов и обеспечения отказоустойчивости, описывая методы маскирования сбоев с помощью аппаратной избыточности [2]. А. Немчинович и Р.А. Киця разрабатывают библиотеку KHNN для вычислений гиперкомплексных нейронных сетей через Keras с использованием TensorFlow и PyTorch, описывая реализацию и возможности программного обеспечения [3]. Д.С. Ситдинов, Б.С. Лещинский и А.А. Шевченко исследуют практики доверия к ML-моделям через контроль SHA-256 и сравнение поведения PyTorch и TensorFlow после десериализации, акцентируя внимание на обеспечении целостности и безопасности моделей [4]. Я.В. Усенко проводит сравнительный анализ мобильных фреймворков для работы с нейросетями — TensorFlow.js, TensorFlow Lite и CoreML, оценивая их применимость в мобильной разработке [5].

1.3 Цель исследования

Целью исследования является проведение сравнительного анализа производительности трёх подходов к реализации нейросетей на Python (NumPy, PyTorch, TensorFlow/Keras) при решении задачи классификации по критериям времени обучения, точности классификации и сложности реализации.

2 Материалы и методы

2.1 Набор данных

Для проведения эксперимента использовался встроенный датасет «Ирисы Фишера» (Iris dataset) из библиотеки scikit-learn. Данные разделены на обучающую (80%, 120 образцов) и тестовую (20%, 30 образцов) выборки с использованием стратифицированного разбиения (параметр `stratify=y`) и фиксированного параметра `random_state=42` для воспроизводимости результатов (Табл. 1).

Таблица 1 — Характеристика набора данных

Параметр	Значение
Количество образцов	150
Количество признаков	4
Количество классов	3
Обучающая выборка	120 (80%)
Тестовая выборка	30 (20%)

2.2 Архитектура нейросети

Для обеспечения корректности сравнения все три реализации использовали идентичную архитектуру нейросети (Табл. 2).

Таблица 2 — Архитектура нейросети

Параметр	Значение
Входной слой	4 нейрона
Скрытый слой	10 нейронов, ReLU
Выходной слой	3 нейрона, softmax
Функция потерь	Категориальная кросс-энтропия
Оптимизатор	Adam

2.3 Метрики сравнения

Для сравнения использовались следующие метрики:

1. время обучения (в секундах);
2. точность классификации на тестовой выборке (в процентах);
3. сложность реализации (количество строк кода).

3 Результаты

3.1 Результаты ручной реализации на NumPy

Ручная реализация нейросети на NumPy показала наилучшее время обучения — 0.37 секунды. Точность на тестовой выборке составила 100%. Листинг реализации представлен на Рисунке 1.

```
def __init__(self, input_size, hidden_size, output_size, lr=0.1):
    self.W1 = np.random.randn(input_size, hidden_size) * 0.5
    self.b1 = np.zeros((1, hidden_size))
    self.W2 = np.random.randn(hidden_size, output_size) * 0.5
    self.b2 = np.zeros((1, output_size))
    self.lr = lr

def sigmoid(self, x):
    return 1 / (1 + np.exp(-np.clip(x, -500, 500)))

def softmax(self, x):
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp_x / np.sum(exp_x, axis=1, keepdims=True)

def forward(self, X):
    self.z1 = np.dot(X, self.W1) + self.b1
    self.a1 = self.sigmoid(self.z1)
    self.z2 = np.dot(self.a1, self.W2) + self.b2
    self.a2 = self.softmax(self.z2)
    return self.a2

def backward(self, X, y):
    m = X.shape[0]
    dz2 = self.a2 - y
    dw2 = np.dot(self.a1.T, dz2) / m
    db2 = np.sum(dz2, axis=0, keepdims=True) / m
    da1 = np.dot(dz2, self.W2.T)
    dz1 = da1 * self.a1 * (1 - self.a1)
    dw1 = np.dot(X.T, dz1) / m
    db1 = np.sum(dz1, axis=0, keepdims=True) / m

    self.W2 -= self.lr * dw2
    self.b2 -= self.lr * db2
    self.W1 -= self.lr * dw1
    self.b1 -= self.lr * db1

def train(self, X, y, epochs):
    losses = []
    for epoch in range(epochs):
        out = self.forward(X)
        loss = -np.mean(np.sum(y * np.log(out + 1e-8), axis=1))
        losses.append(loss)
        self.backward(X, y)
        if epoch % 200 == 0:
            print(f"Эпоха {epoch}, Loss: {loss:.4f}")
    return losses

def predict(self, X):
    return np.argmax(self.forward(X), axis=1)

def accuracy(self, X, y):
    return np.mean(self.predict(X) == y)
```

Рисунок 1 — Листинг реализации нейросети на NumPy

Динамика потерь в процессе обучения представлена на Рисунке 2.

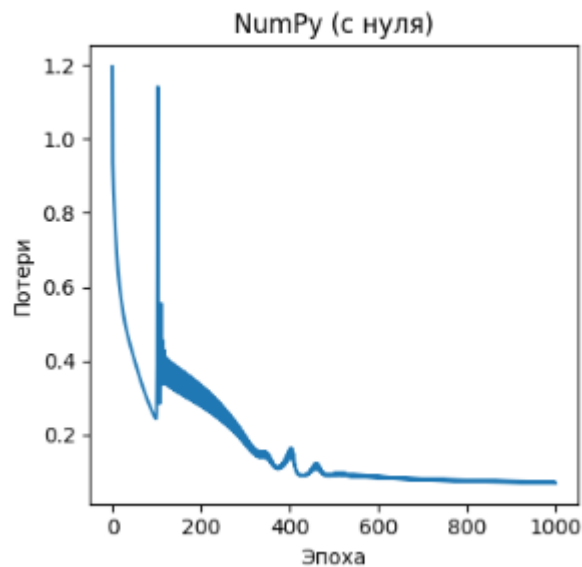


Рисунок 2 — Динамика потерь (NumPy)

3.2 Результаты реализации на PyTorch

PyTorch достиг максимальной точности — 100% на тестовой выборке. Время обучения составило 1.06 секунды. Листинг реализации представлен на Рисунке 3.

```
class PyTorchNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(4, 10)
        self.fc2 = nn.Linear(10, 3)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = PyTorchNet()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

start = time.time()
losses_pytorch = []
for epoch in range(500):
    optimizer.zero_grad()
    outputs = model(X_train_t)
    loss = criterion(outputs, y_train_t)
    loss.backward()
    optimizer.step()
    losses_pytorch.append(loss.item())
    if epoch % 100 == 0:
        print(f" Эпоха {epoch}, Loss: {loss.item():.4f}")
time_pytorch = time.time() - start
```

Рисунок 3 — Листинг реализации нейросети на PyTorch

Динамика потерь представлена на Рисунке 4.

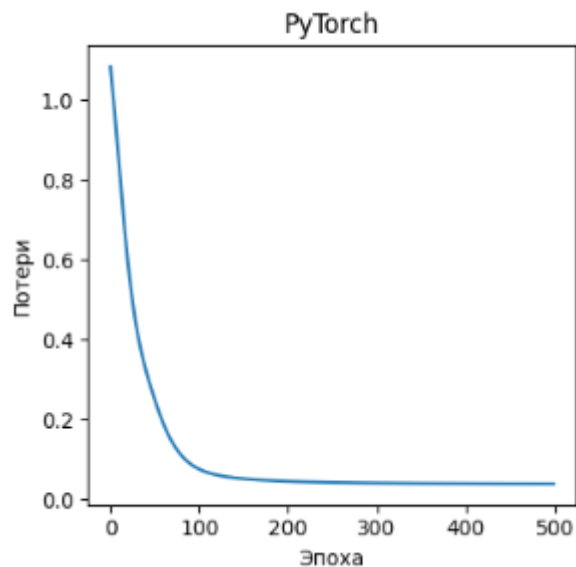


Рисунок 4 — Динамика потерь (PyTorch)

3.3 Результаты реализации на TensorFlow/Keras

TensorFlow/Keras показал наихудшее время обучения — 27.20 секунды. Точность на тестовой выборке составила 93.33%. Листинг реализации представлен на Рисунке 5.

```
tf_model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(4,)),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])

tf_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

start = time.time()
history = tf_model.fit(
    X_train_scaled, y_train_cat,
    epochs=200, batch_size=8, verbose=0,
    validation_split=0.2
)
time_tf = time.time() - start

train_loss, train_acc = tf_model.evaluate(X_train_scaled, y_train_cat, verbose=0)
test_loss, test_acc = tf_model.evaluate(X_test_scaled, y_test_cat, verbose=0)
```

Рисунок 5 — Листинг реализации нейросети на TensorFlow/Keras

Динамика потерь и точности представлена на Рисунке 6.

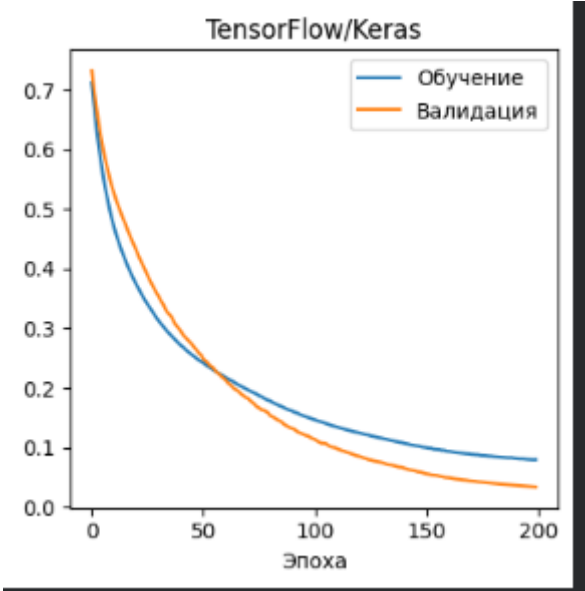


Рисунок 6 — Динамика потерь и точности (TensorFlow/Keras)

3.4 Итоговая таблица сравнения

Сводные результаты представлены в Таблице 3.

Таблица 3 — Сравнительные результаты производительности

Фреймворк	Время обучения (сек)	Точность (тест)
NumPy (с нуля)	0.37	100.00%
PyTorch	1.06	100.00%
TensorFlow/Keras	27.20	93.33%

4 Обсуждение

4.1 Интерпретация результатов

Ручная реализация на NumPy оказалась самой быстрой (0.37 секунды) благодаря отсутствию накладных расходов на абстракции фреймворков. PyTorch показал максимальную точность (100%) при приемлемом времени обучения (1.06 секунды). TensorFlow/Keras, несмотря на простоту разработки, продемонстрировал наихудшее время обучения (27.20 секунд), что объясняется его ориентацией на большие объёмы данных.

4.2 Выбор фреймворка

- 1. NumPy рекомендуется для образовательных целей и работы с очень маленькими датасетами (скорость важнее всего).
- 2. PyTorch рекомендуется для исследовательских проектов, где важна высокая точность, гибкость и приемлемое время обучения.
- 3. TensorFlow/Keras рекомендуется для production-проектов с большими объёмами данных и быстрого прототипирования.

5 Выводы

В ходе проведённого исследования реализован полный цикл сравнительного анализа трёх подходов к реализации нейросетей на Python.

Установлено, что ручная реализации на NumPy является самой быстрой (0.37 секунды). Обе реализации на NumPy и PyTorch показали максимальную точность (100%). TensorFlow/Keras продемонстрировал наихудшее время обучения (27.20 секунд) при адекватной точности (93.33%). Полученные результаты позволяют разработчикам сделать обоснованный выбор инструментария в зависимости от приоритетов: скорость (NumPy), точность и гибкость (PyTorch) или простота разработки (TensorFlow/Keras).

Библиографический список

1. Астапов Р.Л., Мухамадеева Р.М. Автоматизация подбора параметров машинного обучения и обучение модели машинного обучения // Актуальные научные исследования в современном мире. 2021. № 5-2 (73). С. 34–37. URL: <https://elibrary.ru/item.asp?id=46193354>
2. Суханова Н.В., Шептунов С.А. Отказоустойчивая нейронная сеть, способ обнаружения отказов нейронной сети, способ обеспечения отказоустойчивости нейронной сети. Патент RU 2760636 C2. Россия. Заявл. 19.12.2018. Оpubл. 29.11.2021. URL: <https://elibrary.ru/item.asp?id=47430099>
3. Niemczynowicz A., Kycia R.A. KHNN: hypercomplex-valued neural networks computations via Keras using TensorFlow and PyTorch // SoftwareX. 2025. Vol. 30. P. 102163. DOI: 10.1016/j.softx.2025.102163. URL: <https://elibrary.ru/item.asp?id=86854472>
4. Ситдииков Д.С., Лещинский Б.С., Шевченко А.А. Практики доверия к ML-моделям: контроль SHA-256 и сравнение поведения PyTorch и TensorFlow после десериализации // Информационная безопасность регионов России (ИБРР-2025): Материалы XIV Санкт-Петербургской межрегиональной конференции. СПб.: Санкт-Петербургское Общество информатики, вычислительной техники, систем связи и управления, 2025. С. 214–215. URL: <https://elibrary.ru/item.asp?id=88892513>
5. Усенко Я.В. Сравнение мобильных фреймворков для работы с нейросетями: TensorFlow.js, TensorFlow Lite и CoreML // Информационные технологии в экономике: Сборник научных трудов Всероссийской научно-практической Интернет-конференции студентов, аспирантов и молодых ученых с международным участием. М.: Перо, 2025. С. 599–601. URL: <https://elibrary.ru/item.asp?id=86298103>