Софийски университет "Св. Климент Охридски"
Факултет по математика и информатика
Специалност "Извличане на информация и откриване на знания"
Курс "Дълбоко обучение с Тензорфлоу, летен семестър 2024/2025"

NoProp

Трениране на невронни мрежи без back-propagation или forward-propagation

Изготвили:	Даниел Халачев, Цветелина Чакърова, Николета Бейска,	ФН: 4MI3400603; ФН: 8MI3400591; ФН: 2MI3400639;
Ръководители:	проф. Милен Чечев; Александър Захарян; Алексис Датсерис	
Дата:	24 юни 2025 г.	

Съдържание

1 Въведение **2**
1.1 Мотивация . . . 2
1.2 Подобрение на сегашните методи . . . 2
1.3 Цел . . . 2
1.4 Мотивация за избора на NoProp-DT и NoProp-CT . . . 2

2 Кратко въведение в NoProp архитектурата **3**
2.1 Архитектура на NoProp . . . 3
2.1.1 Фаза на извод . . . 3
2.1.2 Фаза на обучение . . . 3
2.2 Структура на дифузионния блок . . . 4
2.3 NoProp-DT . . . 4
2.4 NoProp-CT . . . 5
2.5 Псевдокод на тренирането . . . 6
2.5.1 NoProp-DT . . . 7
2.5.2 NoProp-CT . . . 8
2.6 Препратки към оригиналната статия . . . 9

3 Реализация **9**
3.1 Структура . . . 9
3.2 GitHub . . . 9
3.3 Особености . . . 11
3.4 Хиперпараметри . . . 11

4 Експерименти **13**
4.1 NoProp-DT и MNIST . . . 13
4.2 NoProp-DT и CIFAR-10 . . . 14
4.3 NoProp-DT и CIFAR-100 . . . 15
4.4 NoProp-DT и BLOODMNIST . . . 16
4.5 NoProp-CT и MNIST . . . 17
4.6 NoProp-CT и CIFAR-10 . . . 18
4.7 NoProp-CT и CIFAR-100 . . . 19
4.8 NoProp-CT и BLOODMNIST . . . 20
4.9 Сравнение с оригиналните резултати . . . 21
4.10 Сравнение на използваната памет . . . 21

5 Заключение **22**

1 Въведение

1.1 Мотивация

Тренирането на невронни мрежи чрез методите **back-propagation** и **forward-propagation** е в основата на съвременните подходи за машинно самообучение. Тези методи, макар и доказали своята ефективност, имат **ограничения**. Сред тях са: необходимост от последователно изпълнение на двата етапа; високи изисквания за памет за изчисляване на градиентите; трудности при паралелизация; загуба на контекст. Тези недостатъци мотивират изследователите да търсят алтернативни подходи за обучение, които да са по-ефективни, гъвкави и мащабируеми.

1.2 Подобрение на сегашните методи

В статията *NoProp: Training Neural Networks without Back-propagation or Forward-propagation* [2] е предложен новаторски подход за обучение на невронни мрежи, който избягва използването на посочените традиционни методи. Той е вдъхновен е от дифузионните модели, които се обучават да премахват различни нива на шум. По подобен начин *NoProp* премахва различни нива на шум, но всеки слой се обучава независимо без *forward-propagation* и *back-propagation* през целия модел. Твърди се, че това позволява паралелна обработка, намалява изискванията за памет и ускорява обучението, като същевременно запазва или подобрява точността на класификация в сравнение с други методи без *back-propagation*. *NoProp* има три варианта - *NoProp-DT* (дискретно време), *NoProp-CT* (непрекъснато време) и *NoProp-FM* (flow matching).

1.3 Цел

Статията описва впечатляващи резултати от проведените експерименти, но е публикувана без официален програмен код. В допълнение, измерванията са извършени единствено върху набори от данни с ниска резолюция (`MNIST`, `CIFAR-10`, `CIFAR-100`). Възникват въпроси относно стабилността на модела при по-сложни задачи. Целта на този проект е да се изследва и приложи подходът *NoProp*, за да се провери неговата ефективност и потенциал за подобрения. Проектът имплементира вариантите *NoProp-DT* и *NoProp-CT*, тества ги върху посочените набори от данни, както и върху база данни с изображения с по-висока резолюция - `BLOODMNIST`.

1.4 Мотивация за избора на NoProp-DT и NoProp-CT

Проектът се фокусира върху два от трите възможни разновидности на модела, а именно на *NoProp-DT* и *NoProp-CT*. Този избор се основава на резултатите, докладвани в оригиналната статия, където тези два варианта демонстрират по-висока точност и стабилност в сравнение с *NoProp-FM*.

2 Кратко въведение в NoProp архитектурата

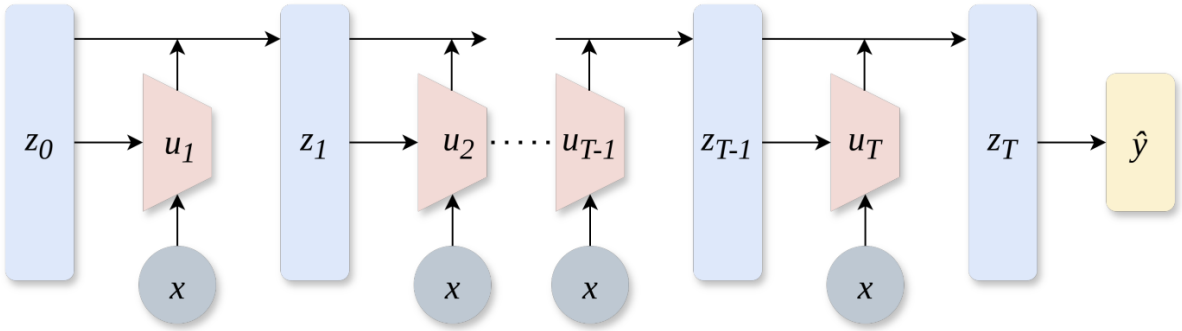
В тази секция се коментират само най-важните моменти от оригиналната *NoProp* статия[2], тъй като основен фокус на текущата документация е да представи разработената имплементация и експериментите, проведени над нея.

2.1 Архитектура на NoProp

2.1.1 Фаза на извод

По време на извод входните данни се обработват в серия от стъпки, като всяка стъпка представлява трансформаця на данните, чрез прекарването им през **дифузионен блок**. Мрежата получава като вход началното изображение x, което е обект на класификация, и Гаусов шум z_0. На всяка стъпка се създава нова латентна променлива z_t. Всеки блок u_t е зависим от променливата от предишната стъпка z_{t-1} и входното изображение за цялата мрежа x.

След t на брой преобразувания получаваме редицата от променливи $z_1, z_2, \ldots, z_t$. Последната от тях се подава на линеен слой и *softmax* активираща функция, които позволяват използването на научените характеристики с цел класификация.



Фигура 1: Архитектура на фазата на извод. $z_1, z_2, \ldots, z_t$ са последователните етапи на трансформация на първоначалния шум z_0. $u_1, u_2, \ldots u_T$ са дифузионните блокове, които се обучават да премахват различни нива на шум. Линейният слой и *softmax* активиращата функция не са изобразени експлицитно.

По време на извод тензорите се подават на слоевете последователно през цялата мрежа, както в традиционна невронна мрежа. Но на фигура 1 се забелязва първата съществена разлика с повечето *backpropagation* алгоритми - входното изображение x се подава на всеки блок u_t, а не еднократно на входа на модела.

2.1.2 Фаза на обучение

Фазата на обучение е ключова за подхода *NoProp*, тъй като всеки дифузионен блок u_t се разглежда като самостоятелна невронна мрежа, която се обучава независимо от останалите. Това позволява по-лесна паралелизация на обучението и намалява изискванията за памет. Линейният слой, използван за класификация, се тренира едновременно с всички

блокове, за да се осигури съгласуваност на научените характеристики и предотвратява срив в ембедингите на класовете.

2.2 Структура на дифузионния блок

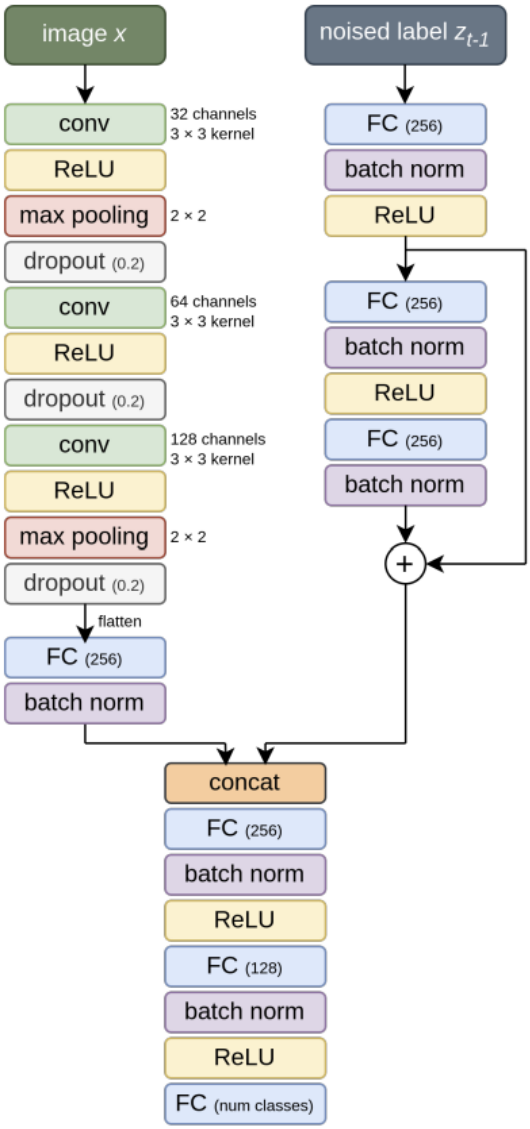
Дифузионният блок в двата варианта *NoProp-CT* и *NoProp-DT* има общи структурни сегменти:

- **кодиране на изображението**
  - **конволюционни слоеве** - извличат характеристики от входното изобраение чрез прилагане на конволюционни филтри върху него. Сегментът наподобява *ResNet* конволюционна мрежа
  - **напълно свързан слой** - свежда извлечените характеристики до ембединги с консистентен размер
- **кодиране на предходния етикет** - силно варира в двата варианта, но включва напълно свързани слоеве и $ReLU$ активации
- **конкатениращ сегмент** - обединява ембедингите на различните кодирания.

В допълнение, блокът на *NoProp-CT* включва и сегмент за **кодиране на времевите последователности**.

2.3 NoProp-DT

Изображението се обработва в краен брой стъпки T, които са параметър на модела.



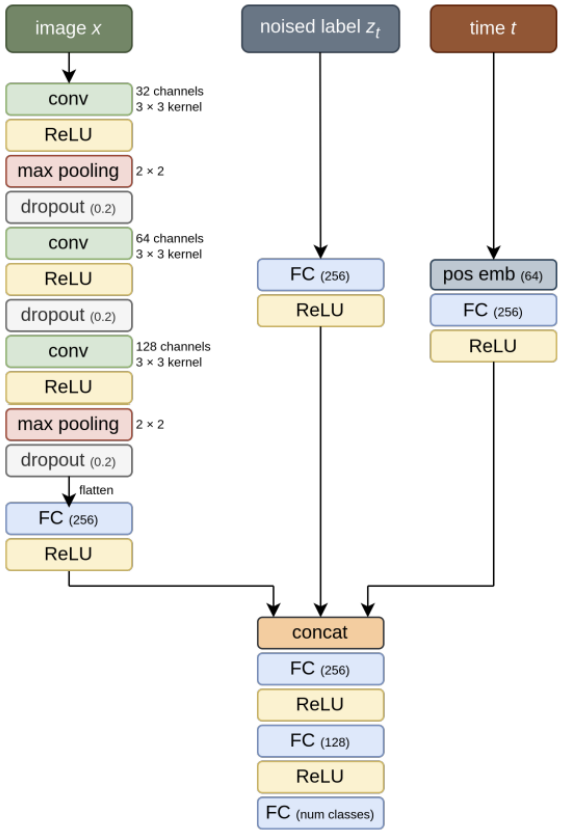
Фигура 2: Модел фазата на трениране на NoProp-DT

Кодирането на изображението x се осъществява чрез конволюционни слоеве *conv* (32, 64, 128 канала; ядра 3x3), слоеве с активационни функции *ReLU*, max pooling 2x2, *dropout* с вероятност 0.2 и *flatten*, последван от напълно свързан *FC* слой и нормализация. Латентната променлива z_{t-1} се обработва чрез FC слоеве и batch нормализация, както и *ReLu* активация, на два етапа, които се свързват чрез *skip connection*. Кодиранията се конкатенират и прекарват през напълно-свързани слоеве, batch нормализация и *ReLU* двукратно. Дифузионният блок завършва с напълно свързан слой с размерност броя възможни класове, за да се извърши класификация.

2.4 NoProp-CT

Архитектурата на NoProp-CT във фазата на трениране е идентична с тази на NoProp-DT със следната разлика: на входа, освен изображението x, се подава и времеви параметър t, който кодира времевата зависимост чрез *positional embeddings*. Съответно всеки блок u_t приема и времевия параметър t. Всяка поетапна трансформация се адаптира според етапа

на дифузия, като по този начин моделът може да "знае"колко е напреднал процесът.



Фигура 3: Модел фазата на трениране на NoProp-CT

Кодирането на изображението x почти аналогично с *NoProp-DT* варианта. Разликата е в последния слой преди конкатенацията - в NoProp-DT има batch нормализация, а тук - *ReLU*. Латентната променлива z_t се обработва чрез FC слой и *ReLU*. В допълнение се приема и променлива за времето t, което липсва в *NoProp-DT*. Тя преминава през слой за позиционно кодиране с 64 единици, FC слой и *ReLU*. Трите кодирания се конкатенират и прекарват през напълно свързa слой и *ReLU* активация двукратно. Блокът отново завършва с напълно свързан слой с размерност броя класове при класификация.

2.5 Псевдокод на тренирането

В статията са представени циклите за обучение на разновидностите на модела чрез псевдокод, за да се илюстрира по-ясно процеса на оптимизация на модела. Обучителният набор от данни стандартно се разделя на части (*batches*) и цикълът се изпълнява за всеки бач. Изчислената загуба за всеки бач оптимизира параметрите на модела.

Вход:

- T - Брой на дискретните стъпки в процеса на дифузия
- $\{(x_i, y_i)_{i=1}^n\}$ - Набор от данни, състоящ се от двойки входни данни и етикети (x_i, y_i)
- B - batch size

- η - Константа, която контролира мащаба на регуляризацията - прави едната част от цялата загубата да има по-голямо значение.
- W_{Embed} - Матрица с кодирания на класовете, която преобразува етикетите y_i във векторни представяния u_{y_i}
- $\{\theta_t\}_{t=1}^{T}$ или θ, θ_{out} - Параметри на модела
- $\{\alpha_t\}_{t=1}^{T}$ или $\bar{\alpha}_t$ - Noise scheduler. Последователност, която определя нивото на шум, добавено към данните в зависимост от времето

2.5.1 NoProp-DT

Algorithm 1 NoProp-DT (Training)

Require: T diffusion steps, dataset $\{(x_i, y_i)\}_{i=1}^{N}$, batch size B, hyperparameter η, embedding matrix W_{Embed}, parameters $\{\theta_t\}_{t=1}^{T}, \theta_{\text{out}}$, noise schedule $\{\alpha_t\}_{t=0}^{T}$
for $t = 1$ to T **do**
  for each mini-batch $\mathcal{B} \subset \{(x_i, y_i)\}_{i=1}^{N}$ of size B **do**
    for each $(x_i, y_i) \in \mathcal{B}$ **do**
      Obtain label embedding $u_{y_i} = \{W_{\text{Embed}}\}_{y_i}$.
      Sample $z_{t,i} \sim \mathcal{N}_d(z_{t,i} | \sqrt{\bar{\alpha}_t} u_{y,i}, 1 - \bar{\alpha}_t)$.
    end for
    Compute the loss function:

$$\begin{aligned}\mathcal{L}_t = &\frac{1}{B} \sum_{i \in \mathcal{B}} [-\log \hat{p}_{\theta_{\text{out}}}(y_i | z_{T,i})] \\ &+ \frac{1}{B} \sum_{i \in \mathcal{B}} D_{\text{KL}}(q(z_0|y_i) \| p(z_0)) \\ &+ \frac{T}{2B} \eta \sum_{i \in \mathcal{B}} (\text{SNR}(t) - \text{SNR}(t-1)) \left\| \hat{u}_{\theta_t}(z_{t-1,i}, x_i) - u_{y_i} \right\|^2 .\end{aligned}$$

    Update θ_t, θ_{out}, and W_{Embed} using gradient-based optimization.
  end for
end for

Фигура 4: Псевдокод за фазата на трениране на NoProp-DT

Основни стъпки: За всяка стъпка времева t обхождаме всеки mini-batch $\mathcal{B}$ от входните данни с размер B. За всяка двойка (x_y, y_i) от $\mathcal{B}$ извличаме векторното представяне u_{y_i} на етикета и генерираме шум $z_{t,i}$ с нормално разпределение. След това изчисляваме функцията на загуба за $\mathcal{B}$. Обновяваме параметрите $theta_t$, $theta_{out}$ и W_{Embed} чрез градиентно-базирана оптимизация.

Функция на загуба: Функцията на загуба е съставена от три събираеми, които имат следните роли:

- $\frac{\mathbf{1}}{\mathbf{B}} \sum_{\mathbf{i} \in \mathcal{B}} [-\log \hat{\mathbf{p}}_{\theta_{\text{out}}}(\mathbf{y_i} | \mathbf{z_{T,i}})]$ - загуба от тип отрицателна логаритмична вероятност (negative log-likelihood loss). Измерва колко добре моделът може да възстанови оригиналните данни от шумния сигнал $z_{T,i}$. Загубата намалява, когато моделът предсказва правилния етикет y_i с по-висока вероятност.

- $\frac{1}{\mathbf{B}} \sum_{\mathbf{i} \in \mathcal{B}} \mathbf{D}_{\mathrm{KL}}(\mathbf{q}(\mathbf{z_0}|\mathbf{y_i}) \| \mathbf{p}(\mathbf{z_0}))$ - загуба от тип KL дивергенция (Kullback-Leibler divergence). Измерва разликата между наученото разпределение и желаното априорно разпределение. Регуларизира латентното пространство, като го принуждава да следва определено разпределение. Загубата намалява, когато латентното пространство $q(z_0|y_i)$ се приближава до априорното разпределение $p(z_0$.

- $\frac{\mathbf{T}}{\mathbf{2B}} \eta \sum_{\mathbf{i} \in \mathcal{B}} (\mathrm{SNR}(\mathbf{t}) - \mathrm{SNR}(\mathbf{t-1})) \left\| \hat{\mathbf{u}_{\theta_t}}(\mathbf{z_{t-1,i}}, \mathbf{x_i}) - \mathbf{u_{y_i}} \right\|^{\mathbf{2}}$ - загуба от тип квадратична разлика (Mean Squared Error) с тегло, базирано на Signal-to-Noise Ratio. Насочва обучението да адаптира параметрите така, че да възстановяват оригиналните представяния на етикетите от шумните данни. Теглото $(\mathrm{SNR}(t) - \mathrm{SNR}(t-1))$ дава по-голямо значение на определени времеви стъпки. η контролира относителната важност на това събираемо. Загубата намалява, когато моделът точно предсказва шума u_{y_i}.

2.5.2 NoProp-CT

Algorithm 2 NoProp-CT (Training)

Require: dataset $\{(x_i, y_i)\}_{i=1}^N$, batch size B, hyperparameter η, embedding matrix W_{Embed}, parameters $\theta, \theta_{\text{out}}$, noise schedule $\bar{\alpha}_t = \sigma(-\gamma_\psi(t))$

for each mini-batch $\mathcal{B} \subset \{(x_i, y_i)\}_{i=1}^N$ with size B **do**

for each $(x_i, y_i) \in \mathcal{B}$ **do**

Obtain label embedding $u_{y_i} = \{W_{\text{Embed}}\}_{y_i}$.

Sample $t_i \sim \mathcal{U}(0, 1)$.

Sample $z_{t_i,i} \sim \mathcal{N}_d(z_{t_i,i} | \sqrt{\bar{\alpha}_{t_i}} u_{y,i}, 1 - \bar{\alpha}_{t_i})$.

end for

Compute the loss function:

$$\begin{aligned}\mathcal{L} = &\frac{1}{B} \sum_{i \in \mathcal{B}} [-\log \hat{p}_{\theta_{\text{out}}}(y_i | z_{1,i})] \\ &+ \frac{1}{B} \sum_{i \in \mathcal{B}} D_{\text{KL}}(q(z_0|y_i) \| p(z_0)) \\ &+ \frac{1}{2B} \eta \sum_{i \in \mathcal{B}} \text{SNR}'(t_i) \left\| \hat{u}_\theta(z_{t_i,i}, x_i, t_i) - u_{y_i} \right\|^2 .\end{aligned}$$

Update θ, θ_{out}, ψ, and W_{Embed} using gradient-based optimization.

end for

Фигура 5: Псевдокод за фазата на трениране на NoProp-CT

Основни стъпки: Обхождаме всеки mini-batch $\mathcal{B}$ от входните данни с размер B. За всяка двойка (x_y, y_i) от $\mathcal{B}$ извличаме векторното представяне u_{y_i} на етикета, избираме случайно време t_i чрез непрекъснато равномерно разпределение $U(0, 1)$ и генерираме шум $z_{t,i}$ с нормално разпределение. След това изчисляваме функцията на загубата за $\mathcal{B}$. Обновяваме параметрите θ, θ_{out} и W_{Embed} чрез градиентно-базирана оптимизация.

Функция на загуба: Функцията на загуба е съставена от три събираеми, които имат следните роли:

- $\frac{\mathbf{1}}{\mathbf{B}} \sum_{\mathbf{i} \in \mathcal{B}} [-\log \hat{\mathbf{p}}_{\theta_{\text{out}}}(\mathbf{y_i} | \mathbf{z_{1,i}})]$ - със същата роля като в *NoProp-DT*.

- $\frac{1}{\mathbf{B}} \sum_{\mathbf{i} \in \mathcal{B}} \mathbf{D}_{\mathrm{KL}}(\mathbf{q}(\mathbf{z_0}|\mathbf{y_i}) \| \mathbf{p}(\mathbf{z_0}))$ - със същата роля като в *NoProp-DT*.
- $\frac{1}{\mathbf{2B}} \eta \sum_{\mathbf{i} \in \mathcal{B}} \mathrm{SNR}'(\mathbf{t_i}) \, \|\hat{\mathbf{u}}_\theta(\mathbf{z_{t_i,i}}, \mathbf{x_i}, \mathbf{t_i}) - \mathbf{u_{y_i}}\|^{\mathbf{2}}$ - загуба от тип квадратична разлика (Mean Squared Error) с тегло, базирано на Signal-to-Noise Ratio. Със същата роля като в *NoProp-DT*. Различава се по липсата на константата . $\mathrm{SNR}'(t_i)$ е производната на Signal-to-Noise Ratio, която дава различни тегла на различните времеви моменти η контролира относителната важност на това събираемо. Загубата намалява, когато моделът точно предсказва шума u_{y_i}.

2.6 Препратки към оригиналната статия

Следните части от оригиналната статия са ключови за по-детайлно запознаване с идеите *NoProp*:

- Техническо описание на *NoProp* (Раздел 2, страници 2–5) - Подробно описание на алгоритъма, включително математическата му основа.
- Експериментални резултати (Раздел 4, страници 6–9)

3 Реализация

3.1 Структура

Алгоритъмът е имплементиран на `Python` с помощта на библиотеката `PyTorch`. Реализацията е базирана на псевдокода, включен в научната работа *NoProp: Training Neural Networks without Back-propagation or Forward-propagation.*[2]

3.2 GitHub

Целият програмен код може да бъде намерен на адрес `https://github.com/DanielHalachev/NoProp`.

```
NoProp
├── config_data # Папка с конфигурационни файлове за всяка двойка набор и алгоритъм
└── src
    ├── components
    │   ├── backbone.py # Екстрактор на характеристики в диф. блок
    │   ├── block.py # Абстрактен клас за диф. блокове
    │   ├── concatenator.py # Абстрактен клас, комбинира кодиранията
    │   ├── ct_block.py # Имплементация на диф. блок за NoProp-CT
    │   ├── ct_concatenator.py # Комбинира изходи за CT версията
    │   ├── ct_noise_scheduler.py # Управлява шума за CT версията
    │   ├── dt_block.py # Имплементация на диф. блок за NoProp-DT
    │   ├── dt_concatenator.py # Комбинира изходи за DT версията
    │   ├── dt_noise_scheduler.py # Управлява шума за DT версията
    │   ├── noise_scheduler.py # Абстрактен клас за управление на шума
    │   ├── resnet_type.py # Дефинира видовете ResNet архитектури
    │   └── time_scheduler.py # Генератор на времеви последователности
    ├── data
    │   ├── cifar100_dataset.py # Зарежда и подготвя данните от CIFAR-100
    │   ├── dataset_manager.py # Менажира всички възможни набори от данни
    │   ├── medmnist_dataset.py # Зарежда и подготвя данните от MedMNIST+
    │   └── mnist_dataset.py # Зарежда и подготвя данните от MNIST
    ├── embedings
    │   ├── ct_label_encoder.py # Енкодер на етикети за NoProp-CT
    │   ├── dt_label_encoder.py # Енкодер на етикети за NoProp-DT
    │   ├── label_encoder.py # Абстрактен базов клас за енкодерите на етикети
    │   ├── sinusoidal_embedding.py  # Изчислява синусоидални ембединги
    │   └── time_encoder.py # Енкодер на времето
    ├── models
    │   ├── base_model.py # Абстрактен базов клас за всички NoProp модели
    │   ├── model_type.py # Дефинира идентификатори за моделите CT, DT и FM
    │   ├── model_wrapper.py # Обвивка около NoProp моделите с тр. метаданни
    │   ├── noprop_ct.py # Имплементация на BaseNoPropModel за CT
    │   └── noprop_dt.py # Имплементация на BaseNoPropModel за DT
    ├── scripts
    │   └── train_test.py # Обучава и тества модела с помощта на train_utils
    └── utils
        ├── device.py # Клас, който определя най-подходящото устройство за обучение
        ├── model_config.py # Дефинира класове за конфигурация на NoProp модел
        ├── test_utils.py # Тества NoProp модел върху тестово множество от данни
        ├── train_config.py # Управлява параметрите за обучение на NoProp
        └── train_utils.py # Полезни функции за обучението
```

3.3 Особености

Инициализиране с прототипи Ембедингите на класовете (матрицата W_{Embed} се инициализират с прототипи, което е един от вариантите, описани в статията.

Алгоритми за извод Тъй като *NoProp* е вдъхновен от дифузионните модели, екипът имплементира алтернативни алгоритми за извод по методите на Ойлер и Хюн. Използването им не доведе до повишаване на резултатите.

Изчисляване на Cross-Entropy Loss В процеса на обучение на модела, екипът установи, че използването на изхода от напълно свързаните слоеве и *softmax*, както е описано в статията, е полезно за *NoProp-CT*, но значително влошава стабилността на трениране при *NoProp-DT*. Екипът постигна поведението на *NoProp-DT*, описано в статията, чрез директно използване на логитите, изведени от дифузионните блокове.

Набор с висока резолюция За да тества възможностите на модела, екипът извърши експеримент с тренировъчния набор `MedMNIST+`, който поддържа резолюция 128x128 пиксела.

Optimizer Екипът експериментира с *Adam* и *AdamW* при по-слабо представящите се комбинации набор-вариант на *NoProp*, но не установи подобрение. Затова изследването се придържаше към оптимизаторите, описани в статията.

Scheduler Екипът експериментира с добавянето на scheduler с 5000 *warmup* стъпки, но това доведе до изненадващо влошаване на стабилността на обучението и много ниски резултати на тренировъчното множество върху всички набори от данни.

3.4 Хиперпараметри

Хиперпараметрите на модела са зададени в конфигурационните файлове спрямо статията. Те могат да бъдат обобщени в следната таблица:

Параметър	**Описание**
`batch_size`	Определя броя на обучителните примери, обработвани заедно в една итерация на обучението, което влияе на ефективността и стабилността му.
`epochs`	Броят на епохите. Епоха представлява един пълен цикъл на обучението през целия набор от данни.
`optimiser`	Оптимизатор на обучението, регулиращ градиентното спускане, адаптирайки скоростта на обучение спрямо временното представяне.
`learning_rate`	Размерът на стъпката при актуализирането на параметрите на модела и влияе върху скоростта и стабилността на сходимостта.
`weight_decay`	Този параметър добавя регуляризация към загубата, предотвратявайки прекомерна сложност в модела чрез намаляване на теглата.
`number_of_timesteps`	Броя на времевите стъпки в NoProp-DT
`eta`	Коефициент за регулиране значимостта на третия член на загубата.
`embedding_dimension`	Размерност на ембедингите z_t.
`label_encoder_hidden_dimension`	Размерността на ембедингите в скритите слоеве на енкодера на етикетите.
`time_encoder_hidden_dimension`	Размерността на ембедингите в скритите слоеве на енкодера на времето.

Таблица 1: Описание на хиперпараметрите

Набор	**Вариант**	**Бач**	**Епохи**	**Опт.**	**lr**	**wd**	**Стъпки**	η
MNIST	NoProp-DT	128	100	AdamW	0.001	0.001	10	0.1
CIFAR-10	NoProp-DT	128	150	AdamW	0.001	0.001	10	0.1
CIFAR-100	NoProp-DT	128	150	AdamW	0.001	0.001	10	0.1
BLOODMNIST	NoProp-DT	128	100	AdamW	0.001	0.001	1000	0.1
MNIST	NoProp-CT	128	100	Adam	0.001	0.001	1000	1
CIFAR-10	NoProp-CT	128	500	Adam	0.001	0.001	1000	1
CIFAR-100	NoProp-CT	128	1000	Adam	0.001	0.001	1000	1
BLOODMNIST	NoProp-CT	128	1000	Adam	0.001	0.001	1000	1

Таблица 2: Стойности на хиперпараметри за различни множества от данни и алгоритми. `lr` обозначава `learning_rate`, `wd` - `weight_decay`, а стъпките се отнасят до фазата на извод.

4 Експерименти

Имплементираният съгласно статията модел премина обучение на следния хардуер:

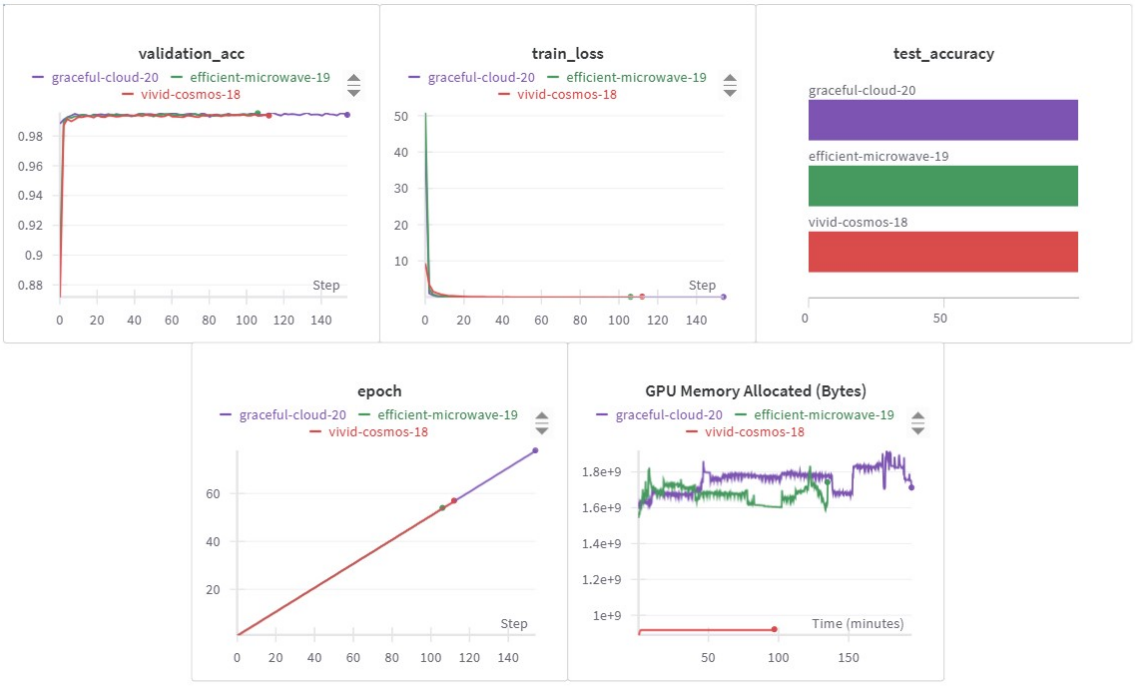
```
CPU: Intel Xeon E5-2690 v4
GPU: NVIDIA GeForce RTX 4070 Ti SUPER

CPU: AMD EPYC 7702 64-Core Processor
GPU: NVIDIA RTX 4090
```

Данните бяха съхранявани систематично с помощта на платформата *Weights & Biases*.

Двата варианта на модела са тренирани над различни множества данни, като постигнатите резултати са представени на графиките по-долу. За данните, представени в оригиналната статия - `MNIST`, `CIFAR-10` и `CIFAR-100`, са използвани стойностите на хиперпараметрите посочени в нея (Таблица 2). Представените стойности на постигнатите резултати са от извадки от три отделни експеримента на обучение, данните за които могат да се видят на графиките. Изборът на извадки от три експеримента балансира статистическа значимост и хардуерните наличности на екипа.

4.1 NoProp-DT и MNIST

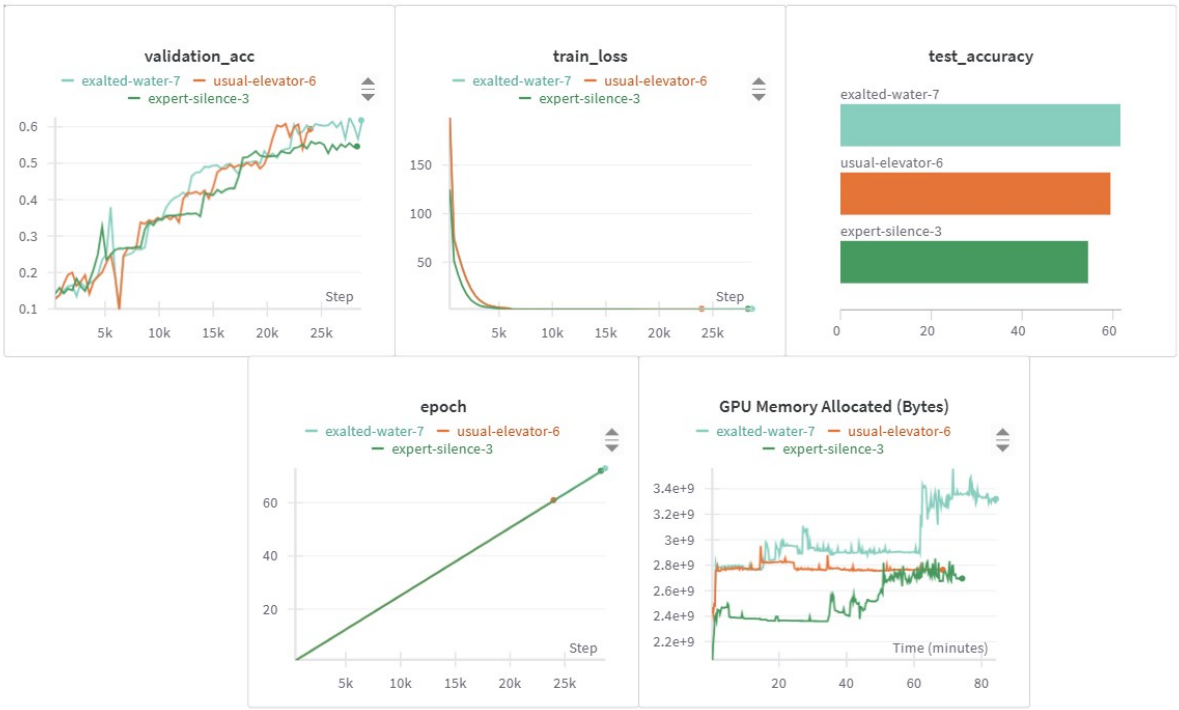


Фигура 6: Графични резултати за NoProp-DT върху MNIST

Име на изпълнение	Загуба	Тестова точност	Брой епохи	GPU памет (GB)
graceful-cloud-20	0.0875	99.42	78	1.92
efficient-microwave-19	0.1151	99.53	54	1.84
vivid-cosmos-18	0.1743	99.37	57	0.86

Таблица 3: Метрики за различни изпълнения на NoProp-DT върху MNIST

4.2 NoProp-DT и CIFAR-10

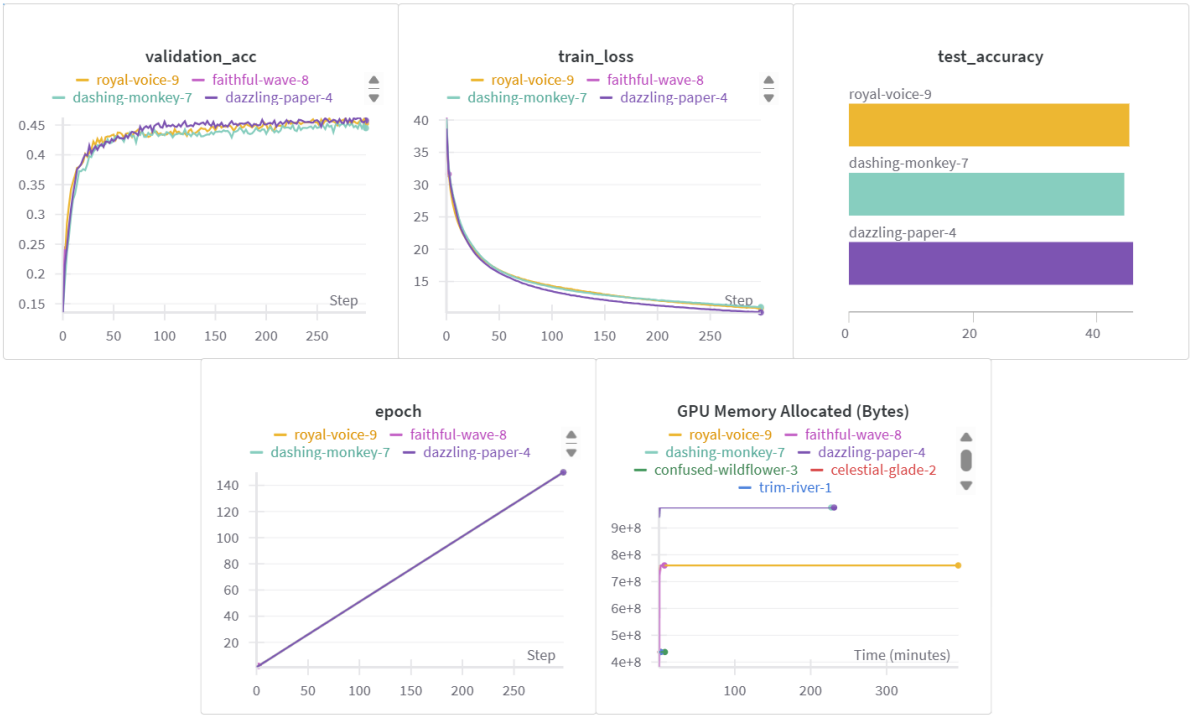


Фигура 7: Графични резултати за NoProp-DT върху CIFAR-10

Име на изпълнение	Загуба	Тестова точност	Брой епохи	GPU памет (GB)
helpful-gorge-6	1.3370	80.54	150	0.69
curious-waterfall-5	1.4325	79.62	150	1.84
fancy-terrain-1	1.2189	80.75	150	0.91

Таблица 4: Метрики за различни изпълнения на NoProp-DT върху CIFAR-10

4.3 NoProp-DT и CIFAR-100

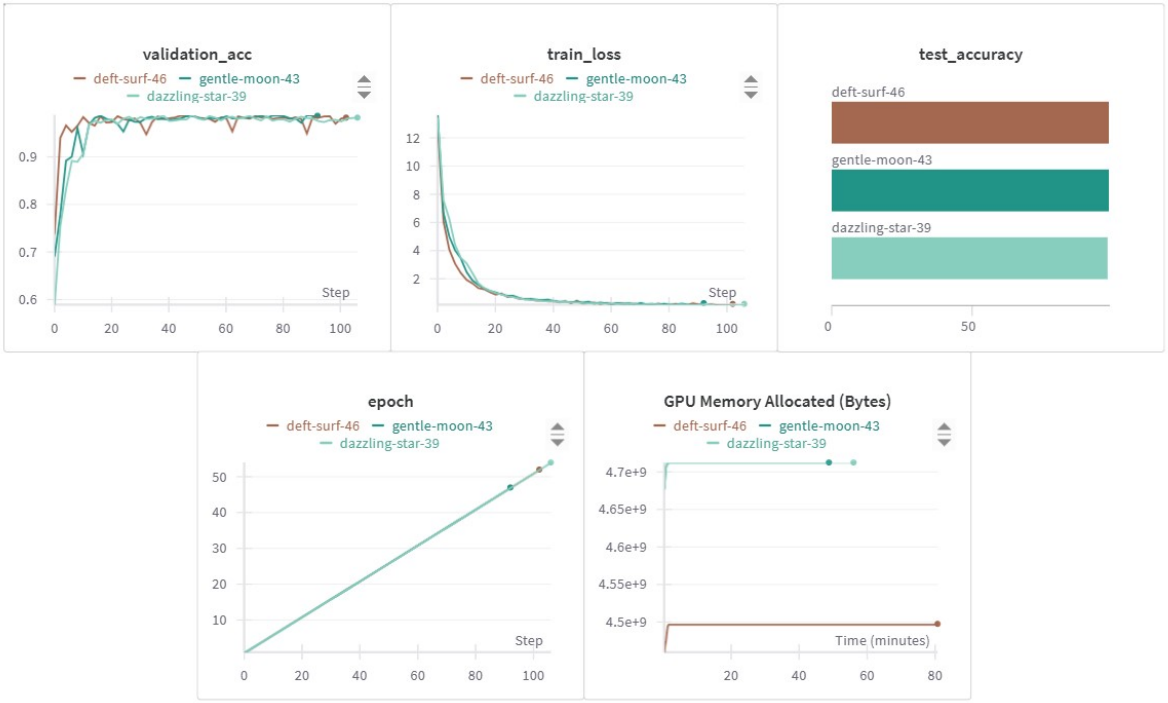


Фигура 8: Графични резултати за NoProp-DT върху CIFAR-100

Име на изпълнение	Загуба	Тестова точност	Брой епохи	GPU памет (GB)
royal-voice-9	10.8631	45.27	150	0.71
dashing-monkey-7	11.0258	44.49	150	0.98
dazzling-paper-4	10.2014	45.75	150	0.98

Таблица 5: Метрики за различни изпълнения на NoProp-DT върху CIFAR-100

4.4 NoProp-DT и BLOODMNIST

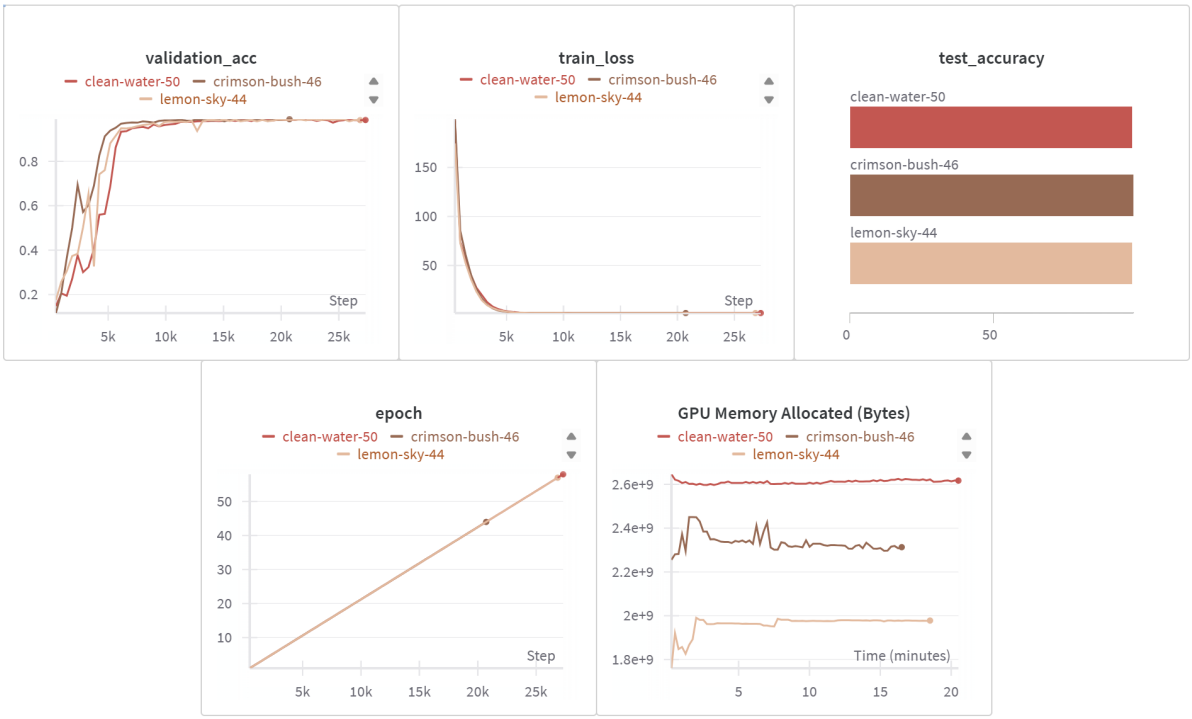


Фигура 9: Графични резултати за NoProp-DT върху BLOODMNIST

Име на изпълнение	Загуба	Тестова точност	Брой епохи	GPU памет (GB)
deft-surf-46	0.2152	98.25	52	4.50
gentle-moon-43	0.2713	98.42	47	4.71
dazzling-star-39	0.2163	97.98	54	4.71

Таблица 6: Метрики за различни изпълнения на NoProp-DT върху BLOODMNIST

4.5 NoProp-CT и MNIST

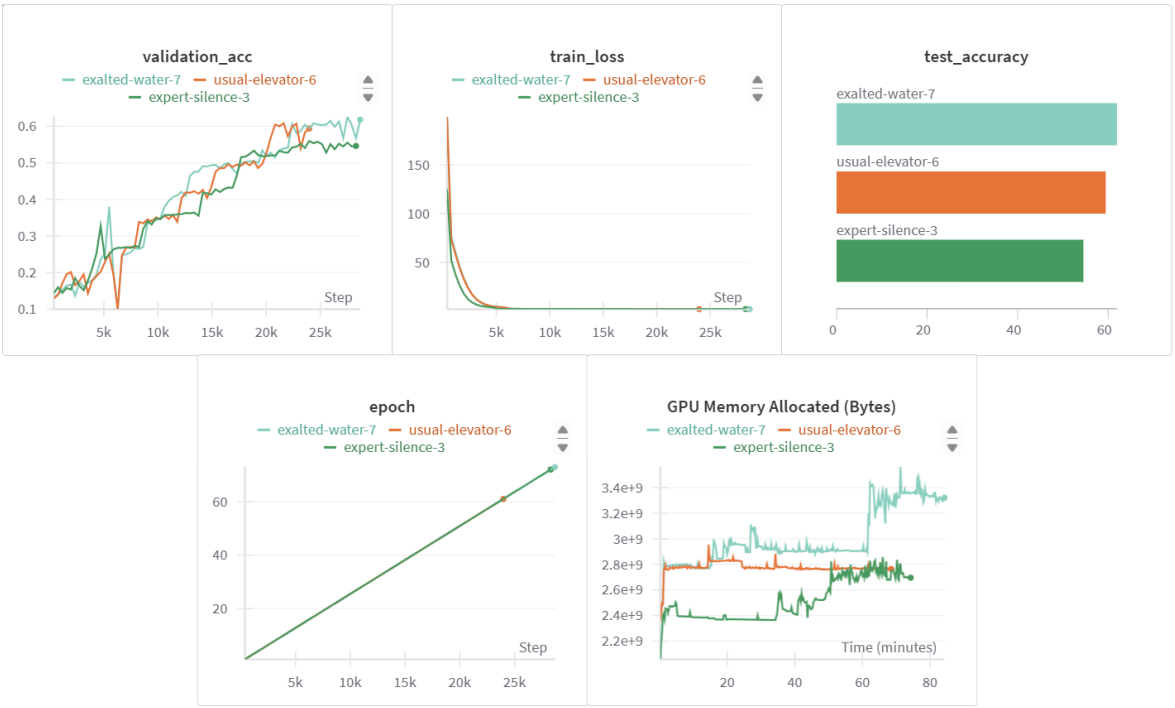


Фигура 10: Графични резултати за NoProp-CT върху MNIST

Име на изпълнение	Загуба	Тестова точност	Брой епохи	GPU памет (GB)
clean-water-50	1.4221	98.63	58	2.65
crimson-bush-46	1.4950	99	44	2.45
lemon-sky-44	1.4934	98.57	57	1.99

Таблица 7: Метрики за различни изпълнения на NoProp-CT върху MNIST

4.6 NoProp-CT и CIFAR-10

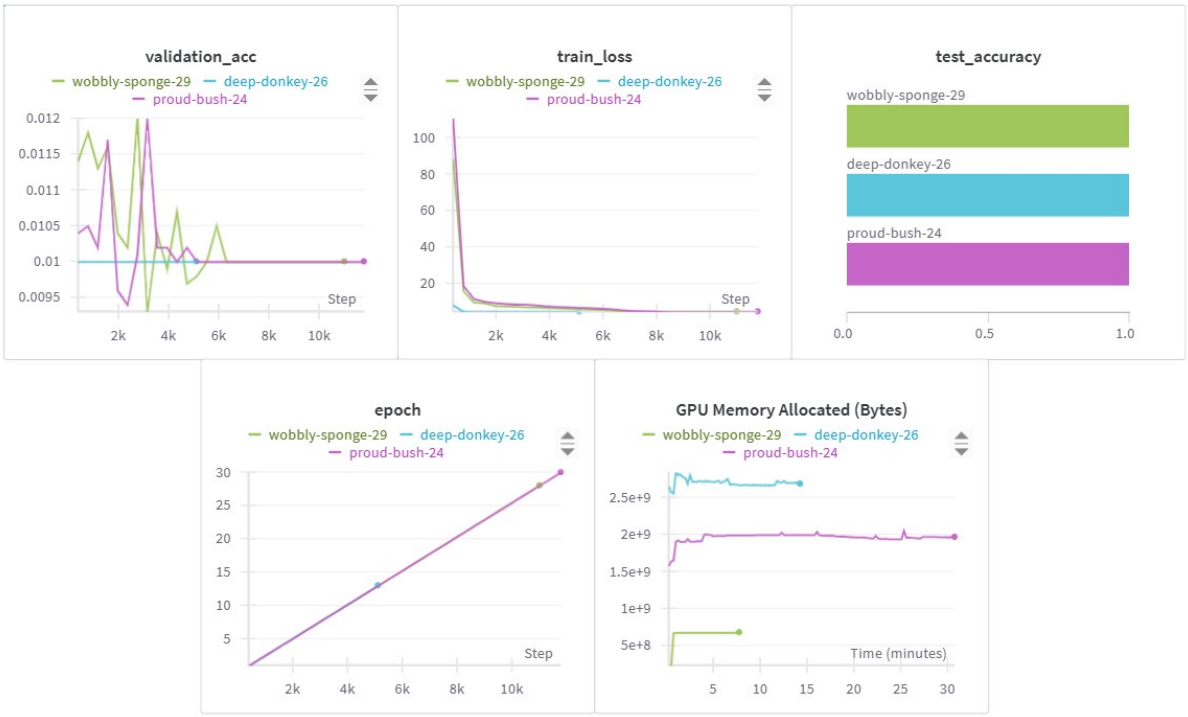


Фигура 11: Графични резултати за NoProp-CT върху CIFAR-10

Име на изпълнение	Загуба	Тестова точност	Брой епохи	GPU памет (GB)
exalted-water-7	1.8715	61.79	73	3.17
usual-elevator-6	1.8880	59.39	61	2.75
expert-silence-3	1.9260	54.62	72	2.50

Таблица 8: Метрики за различни изпълнения на NoProp-CT върху CIFAR-10

4.7 NoProp-CT и CIFAR-100

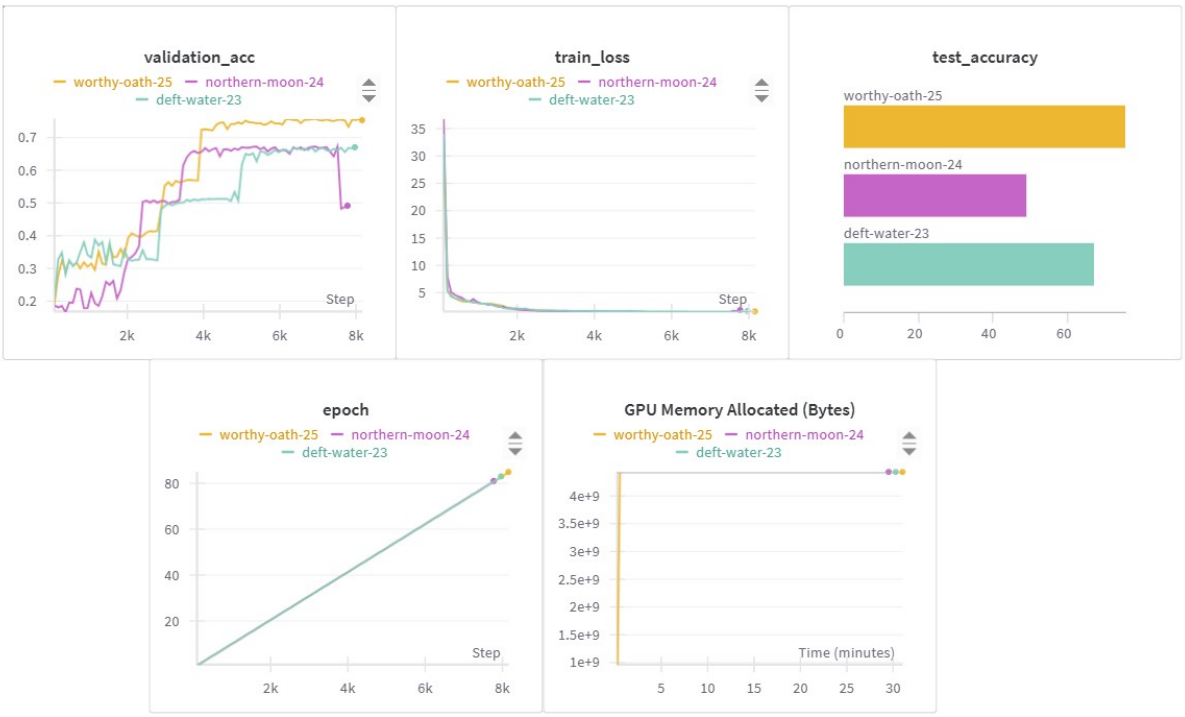


Фигура 12: Графични резултати за NoProp-CT върху CIFAR-100

Име на изпълнение	Загуба	Тестова точност	Брой епохи	GPU памет (GB)
wobbly-sponge-29	4.6167	1	29	0.63
deep-donkey-26	4.6106	1	13	2.64
proud-bust-24	4.6209	1	30	1.91

Таблица 9: Метрики за различни изпълнения на NoProp-CT върху CIFAR-100

4.8 NoProp-CT и BLOODMNIST



Фигура 13: Графични резултати за NoProp-CT върху BLOODMNIST

Име на изпълнение	Загуба	Тестова точност	Брой епохи	GPU памет (GB)
worthy-oath-25	1.5700	75.50	85	4.43
northern-moon-24	1.8103	49.08	81	4.43
deft-water-23	1.6466	67.14	83	4.43

Таблица 10: Метрики за различни изпълнения на NoProp-CT върху BLOODMNIST

4.9 Сравнение с оригиналните резултати

Dataset	Split	NoProp-CT		NoProp-DT	
		Paper	Implementation	Paper	Implementation
MNIST	Train	97.18 ± 1.02	$\mathbf{98.73 \pm 0.23}$	$\mathbf{99.97 \pm 0.0}$	99.44 ± 0.08
	Test	97.17 ± 0.94	$\mathbf{98.73 \pm 0.23}$	$\mathbf{99.54 \pm 0.04}$	99.44 ± 0.08
CIFAR-10	Train	$\mathbf{86.2 \pm 7.34}$	58.6 ± 3.65	$\mathbf{97.23 \pm 0.11}$	80.30 ± 0.6
	Test	$\mathbf{66.54 \pm 3.63}$	58.6 ± 3.65	$\mathbf{80.54 \pm 0.2}$	80.30 ± 0.6
CIFAR-100	Train	$\mathbf{40.88 \pm 10.72}$	0.23†	$\mathbf{90.7 \pm 0.14}$	45.17 ± 0.64
	Test	21.31 ± 4.17	**0.23**†	$\mathbf{46.06 \pm 0.25}$	45.17 ± 0.64
MEDMNIST	Train	-	$\mathbf{69.92 \pm 4.71}$	-	$\mathbf{98.39 \pm 0.24}$
	Test	-	$\mathbf{63.90 \pm 13.50}$	-	$\mathbf{98.22 \pm 0.22}$

Таблица 11: Сравнение на получените резултати от имплементацията с обявените резултати в оригиналната статия.
†Тренирането на модела беше нестабилно

4.10 Сравнение на използваната памет

Вариант	**MNIST**	**CIFAR-10**	**CIFAR-100**
NoProp-DT	1.54 GB	1.15 GB	0.89 GB
NoProp-CT	2.36 GB	2.81 GB	1.73 GB

Таблица 12: Сравнение на употребата на GPU RAM памет на имплементацията за различни методи и набори от данни.

5 Заключение

Проектът за имплементация и тестване на подхода за обучение *NoProp* демонстрира съвпадение с цитираните в статията възможности на модела, което доказва коректността на имплементацията. Резултатите от тестовете върху разработката на екипа показват, че NoProp-DT постига висока точност на класификация върху наборите от данни MNIST (99.44 ± 0.08%), CIFAR-10 (80.30 ± 0.6%), CIFAR-100 (45.17 ± 0.64%) и BLOODMNIST (98.22 ± 0.22%), като в някои случаи надхвърля или се доближава до оригиналните резултати от статията. NoProp-CT, макар и с по-ниска производителност върху по-сложни набори като CIFAR-10 (58.6 ± 3.65%) и CIFAR-100 (1%), показва стабилност върху MNIST (98.73 ± 0.82%).

Основните предимства на *NoProp* включват възможността за паралелна обработка, намалени изисквания за памет и потенциал за ускоряване на обучението. Проект успешно имплементира *NoProp-DT* и *NoProp-CT*, като разширява тестването и върху набора `BLOODMNIST` от `MedMNIST+`. Резултатите демонстрират приложимостта на метода и към изображения с по-висока резолюция.

За съжаление, екипът ни констатира и някои (субективни) недостатъци на подхода, сред които:

- значителна сложност на имплементацията в сравнение с *backpropagation*
- повишена вероятност от бъгове и логически грешки, която наблюдавахме в други неофициални имплементации[1] и нашата собствена
- относителна нестабилност на тренирането, която не се наблюдава при *backpropagation*
- голяма дисперсия и немонотонност в метриките на резултатите от тренирането.

В заключение, екипът констатира, че NoProp е успешна алтернатива на съществуващите алгоритми без използване на *backpropagation*, както и традиционните *backpropagation* алгоритми в сценарии на паралелно обучение и ограничена работна памет на хардуера.

Литература

[1] Potentially incorrect calculation of noisy label? · Issue 4 · yhgon/NoProp — github.com. https://github.com/yhgon/NoProp/issues/4.

[2] Qinyu Li, Yee Whye Teh, and Razvan Pascanu. Noprop: Training neural networks without back-propagation or forward-propagation, 2025.