# NoProp: Training Neural Networks Without Backpropagation or Forward Propagation

Presented by Ivan Brillo

August 1, 2025

# Introduction: The Deep Learning Dilemma

- Traditional Neural Network (NN) training relies heavily on:
  - **Forward Propagation:** Computing outputs from inputs.
  - **Backpropagation:** Calculating gradients to update weights.
- These methods are fundamental but come with significant costs:
  - High memory consumption (storing activations for backprop).
  - Intensive computational requirements.
  - Complexity, especially as models scale to billions of parameters.

# The Rise of Alternative Training Paradigms

- Growing interest in moving beyond traditional backpropagation.
- Motivations include:
  - Seeking more biologically plausible learning mechanisms.
  - Enabling training in constrained environments (IoT/Edge).
  - Allowing parallel training, that is impossible in traditional approaches due to the sequential propagation of gradients.
- NoProp emerges as a radical proposal in this space: **training neural networks without forward or backward propagation**.
  - It treats each layer of the network as a "black box" where only external signals guide learning (input and label embedding).
  - This flips conventional deep learning on its head by eliminating the need to propagate information end-to-end through the network.
  - The network's layers don't discover more abstract features of the input as their depth increases; instead, each layer is trained solely to denoise its fixed, noisy input toward the true label embedding.

# Forward and Reverse Process in NoProp (1)

In order to understand how NoProp works, we start by defining:

1. $(x, y) \sim q_{data}(x, y)$ our input-lable tuple
2. $z_i \in \mathbb{R}^d$ with $i \in \{1, \ldots, T\}$ the **stochastic** activation of layer $i$ in a neural network, in which each layer has as input the tuple $(x, z_{t-1})$
3. $z_0 \sim p(z_0)$ a fixed prior distribution of layer 0 activation

Then we can define, exploiting the chain rule of probabilities and the Markov property:

1. The forward process (generative model)

$$p_\theta(z_{0:T}, y \mid x) = p(z_0)[\prod_{t=1}^{T} p_\theta(z_t \mid z_{t-1}, x)]p_\theta(y \mid z_T)$$

2. The reverse process (variational posterior)

$$q(z_{0:T} \mid y, x) = q(z_T \mid y) \prod_{t=1}^{T} q(z_{t-1} \mid z_t)$$

# Forward and Reverse Process in NoProp (2)

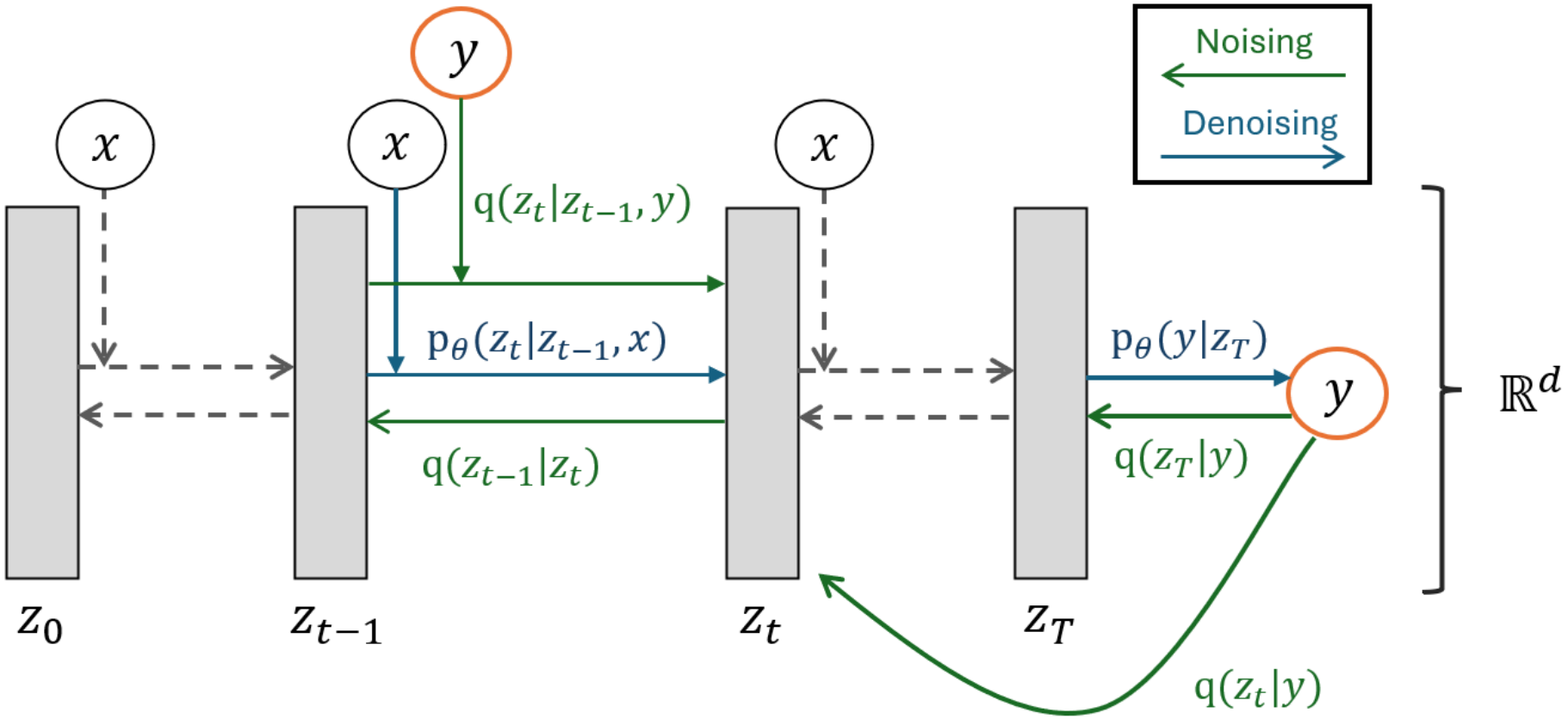


Figure: Representation of forward and reverse process in NoProp

# Variance Preserving Process

**Objective:** Train each $p_\theta(z_t \mid z_{t-1}, x)$ so that, when chained, they "denoise" back to $y$, without ever propagating *between* layers.

The authors fixed the variational posterior to be a tractable Gaussian distribution. Each backward step is defined, as in DDPM, by:

$$q(z_{t-1} \mid z_t) = \mathcal{N}(\sqrt{\alpha_{t-1}}\, z_t,\, (1 - \alpha_{t-1}) I_d)$$

gradually adding noise with a schedule $\alpha_t \in (0, 1)$

And, for the last layer:

$$q(z_T \mid y) = \mathcal{N}(\sqrt{\alpha_T}\, u_y,\, (1 - \alpha_T) I_d)$$

where $u_y \in \mathbb{R}^d$ is the embedding for class $y$.

# Class Embedding

To represent the discrete class $y$ as a continuous vector $u_y$, the authors defined an embedding matrix

$$W_{\text{EMBED}} \in \mathbb{R}^{d \times c},$$

where $c$ is the number of classes and $d$ is the embedding dimensionality. The class embedding is obtained by selecting the $y$-th row of $W_{\text{EMBED}}$:

$$u_y = \{W_{\text{EMBED}}\}_y$$

The embedding matrix $W_{\text{EMBED}}$ can be either:

1. **Fixed**: for instance, $W_{\text{EMBED}} = I_c$ fixing $d = c$, resulting in a one-hot encoding of classes.
2. **Trainable**: $W_{\text{EMBED}}$ is learned jointly during training (with $d$ chosen independently of $c$), allowing embeddings to capture inter-class similarities.

# Properties of this Variance Preserving Process (1)

Defining the variational posterior as we did presents two properties:

1. Closed form marginal $q(z_t \mid y) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\, u_y, (1 - \bar{\alpha}_t) I_d)$ with $\bar{\alpha}_t = \prod_{s=t}^{T} \alpha_s$
   - $\bar{\alpha}_t$ is strictly increasing. As the number of layers tends to infinity, $q$ collapses to a standard Gaussian,

     $$q(z_0 \mid y) \;\to\; \mathcal{N}(0, I_d),$$

     since $\bar{\alpha}_0$ is an infinite product of positive terms smaller than one. This property gives $q$ the name **Variance Preserving** (VP) process, in contrast to Variance Exploding (VE) processes, where the final variance diverges to infinity.
   - Having a closed-form marginal (which can be interpreted as a discrete Gaussian probability path $q_t$) allows parallel training of the different layers, without requiring forward evaluation. At layer $t$, instead of propagating through the forward process, we can directly sample the previous activation $z_{t-1}$ from the closed-form marginal.

# Properties of this Variance Preserving Process (2)

2. The process is invertible:

$$q(z_t \mid z_{t-1}, y) = \mathcal{N}(a_t u_y + b_t z_{t-1},\ c_t I_d)$$

with $a_t, b_t, c_t$ coefficients computable from $\{\alpha_{t-1}, \ldots, \alpha_T\}$.

- With the reparameterization trick and substituting $a_t$, $b_t$ and $c_t$, we can write the target stochastic activation of layer $t > 0$ as:

$$z_t^* = \underbrace{a_t u_y}_{\text{label embedding contribution}} + \underbrace{b_t z_{t-1}^*}_{\text{weighted skip connection}} + \underbrace{\sqrt{c_t}\,\epsilon}_{\text{Gaussian noise, } \epsilon \sim \mathcal{N}(0,I)}$$

$$= \frac{\sqrt{\bar{\alpha}_t}(1-\alpha_{t-1})}{1-\bar{\alpha}_{t-1}} u_y + \frac{\sqrt{\alpha_{t-1}}(1-\bar{\alpha}_t)}{1-\bar{\alpha}_{t-1}} z_{t-1}^*$$

$$+ \sqrt{\frac{(1-\bar{\alpha}_t)(1-\alpha_{t-1})}{1-\bar{\alpha}_{t-1}}}\,\epsilon.$$

# Loss Formulation (1)

We want to maximize the log-likelihood of our dataset w.r.t. our model. Unfortunately the value $\log p_\theta(y \mid x) = \log \int p_\theta(z_{0:T}, y \mid x)\, dz_{0:T}$ is intractable. We can rely on the minimization of the negative ELBO:

$$-\log p_\theta(y \mid x) \leq -\mathbb{E}_{q(z_{0:T}|y)}\Big[\log p_\theta(z_{0:T}, y \mid x) - \log q(z_{0:T} \mid y)\Big] = \mathcal{L}_{\text{NoProp}}$$

By further simplifying the terms, $\mathcal{L}_{\text{NoProp}}(x, y, \theta)$ becomes:

$$\mathcal{L}_{\text{NoProp}}(x, y, \theta) = \underbrace{\mathbb{E}_{q(z_T|y)}[-\log p_\theta^{\text{out}}(y \mid z_T)]}_{\text{final-step CE}} + \underbrace{D_{\text{KL}}(q(z_0 \mid y) \,\|\, p(z_0))}_{\text{prior KL}}$$
$$+ \underbrace{\sum_{t=1}^{T} \mathbb{E}_{q(z_{t-1}|y)}\Big[D_{\text{KL}}(q(z_t \mid z_{t-1}, y) \,\|\, p_\theta(z_t \mid z_{t-1}, x))\Big]}_{\text{denoising error terms}}$$

# Loss Formulation (2)

Thus, in order to minimize the loss we can impose some restrictions on $p_\theta$ in order to match the same distribution family of the posterior and decrease the complexity of the NN. By looking at $\mathcal{L}$ we can fix:

1. $p(z_0) = \mathcal{N}(0, I_d)$
2. Since in $q(z_t|z_{t-1}, y)$ the only unknown during inference is $u_y$ we can define $p_\theta(z_t \mid z_{t-1}, x) = \mathcal{N}(a_t \hat{u}_{\theta_t}(z_{t-1}, x) + b_t z_{t-1}, c_t I_d)$
   - The layer $t$ of the NN needs to learn only an approximation of $u_y$ given $x$ and $z_{t-1}$
   - The NN doesn't need to estimate the variance of the forward process since it is known apriori
   - We can express the activation of the NN at layer $0 < t \leq T$ as:

$$z_t = \underbrace{a_t \hat{u}_{\theta_t}(z_{t-1}, x)}_{\text{residual block contribution}} + \underbrace{b_t z_{t-1}}_{\text{weighted skip connection}} + \underbrace{\sqrt{c_t}\,\epsilon}_{\text{Gaussian noise, } \epsilon \sim \mathcal{N}(0,I)}$$

# How $\hat{u}_{\theta_t}$ is implemented

$\hat{u}_{\theta_t}$ is a function that takes as input the previous layer activation $z_{t-1}$ and the input $x$.

$$\hat{u}_{\theta_t} : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^d$$

This function is implemented by a neural network.

- Each layer $t$ of the NN is defined by the block $\hat{u}_{\theta_t}(z_{t-1}, x)$

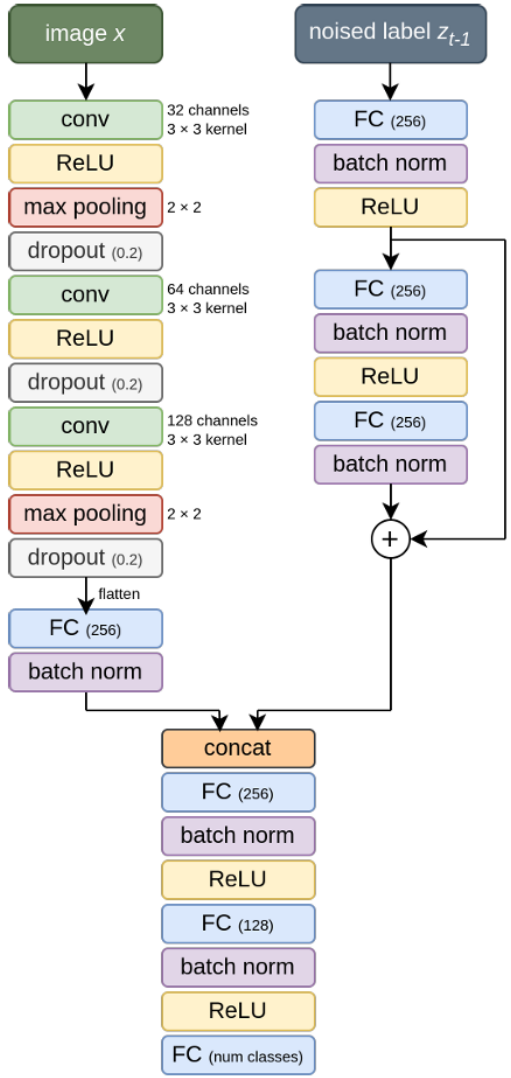


Figure: Neural network block used by the authors

# Loss Formulation (3)

By further simplifying the loss, we note that the prior KL doesn't depend on $\theta$, and thus can be removed. Substituting the new $p_\theta$ formulation, adding a hyperparameter $\eta$ and calling $\mathrm{SNR}(t) = \frac{\bar{\alpha}_t}{1-\bar{\alpha}_t}$

$$\mathcal{L}_{\mathrm{NoProp}} = \underbrace{\mathbb{E}_{q(z_T|y)}[-\log p_\theta^{\mathrm{out}}(y \mid z_T)]}_{\text{final-step CE}} + \underbrace{\frac{\eta}{2}\sum_{t=1}^{T}(\mathrm{SNR}(t) - \mathrm{SNR}(t-1))\,\mathbb{E}_{q(z_{t-1}|y)}[\|\hat{u}_{\theta_t}(z_{t-1}, x) - u_y\|_2^2]}_{\text{L2 denoising terms}}$$

- $\mathrm{SNR}(t)$ measures how much of the true activation $z_t^*$ comes from $z_{t-1}^*$ and how much from the injected noise

# Key Features of the NoProp Loss

1. **Final-step cross-entropy (CE):**

$$\mathbb{E}_{q(z_T|y)}[-\log p_\theta^{\text{out}}(y \mid z_T)]$$

   Measures how well the last latent $z_T$ predicts $y$.

2. **Weighted denoising terms:**
   - $[\text{SNR}(t) - \text{SNR}(t-1)] > 0$ weights each denoising step by its signal-to-noise gain from the previous layer
   - $\hat{u}_{\theta,t}(z_{t-1}, x)$ is the model's denoiser at layer $t$. It tries to predict the true $u_y$ representation of label $y$.
   - Hyperparameter $\eta$ scales the overall denoising penalty.

3. **Independence of layer weights** As we can see, all the NN blocks ($p_\theta^{\text{out}}$ and $\hat{u}_{\theta_t}$) are independent of each other, thus the training can be done independently, without forward or backward propagation
   - $\nabla_{\theta_x} \hat{u}_{\theta_t} = 0 \ \forall x \in \{1, \ldots, T\} \setminus \{t\}$ similar for $p_\theta^{\text{out}}$

# Training the NoProp Network (DT)

**Key Idea:** Each layer learns independently to *denoise a noisy target label*, without forward or backward propagation.

**What is Trained**

- Diffusion dynamics blocks $\hat{u}_{\theta_t}$ (one per timestep $t = 1, \ldots, T$)
- Classifier head $p_{\theta_{\text{out}}}(y|z_T)$
- Class embedding matrix $W_{\text{Embed}}$ [optional]
- SNR($t$) strictly decreasing noise schedule [optional]

**How Training Works** for each $(x, y) \in D$ tuple

1. For each time step $t \in \{0, \ldots, T\}$, sample a noisy activation using the prior $z_0 \sim \mathcal{N}(0, I_d)$ or the posterior $z_t \sim \mathcal{N}(\sqrt{\bar{\alpha}_t} u_y, 1 - \bar{\alpha}_t) I_d)$.
2. Train $\hat{u}_{\theta_t}(z_{t-1}, x)$ to predict the clean embedding $u_y$. Update $\theta_t$.
3. Train $p_{\theta_{\text{out}}}(y|z_T)$ with cross-entropy loss. Update $\theta_{\text{out}}$.

# Training Considerations

- The paper uses SGD to train each NN block sequentially but independently.
  - This is not mandatory since layers can be updated completely in parallel
- Although not used in the paper, we could train each NN block without computing gradients at all. This could be achieved using techniques inspired by:
  - **Evolutionary algorithms:** Perturbing weights, evaluating fitness (loss impact), and selecting better variants.
  - **Reinforcement learning-style updates:** Learning policies for weight adjustments based on reward signals.

# NoProp vs. Backpropagation: Key Advantages

1. **Memory Efficiency**
   - No need to cache intermediate activations or gradients
   - Frees up GPU memory for larger models or higher batch sizes
   - Makes training on resource-constrained devices more feasible
2. **Immunity to Gradient Pathologies**
   - No vanishing or exploding gradients
   - This contributes to more stable and robust training, especially in very deep networks
3. **Massive Parallelism**
   - Unlike backpropagation, which is inherently sequential (gradients flow backward layer by layer), NoProp can update layers concurrently
   - Greatly accelerates distributed and multi-device training
4. **Biological Plausibility**
   - Biological neurons are thought to update based on local signals, without a global backpropagation-like mechanism
   - Aligns more closely with certain theories of brain-like learning, opening new avenues for neuroscience-inspired AI architectures

# NoProp-FM: Flow Matching Formulation

- Let

$$z_0 \sim p_{\text{init}} = \mathcal{N}(0, I), \qquad \mu_{z_1} = u_y = \{W_{\text{EMBED}}\}_y$$

- Define a Gaussian conditional probability path, with constant $\sigma^2$

$$p_t(z_t \mid z_0, \mu_{z_1}) = \mathcal{N}((1-t)z_0 + t\,\mu_{z_1},\ \sigma^2 I) \quad (t \in [0,1]).$$

- The *true* conditional vector field ($u_t^{\text{target}}$) that given $\mu_{z_1} = u_y$, transforms the noise distribution $p_{init}$ into $\mathcal{N}(\mu_{z_1},\ \sigma^2 I)$ is given by the following ODE:

$$z_0 \sim p_{\text{init}}, \quad \frac{\mathrm{d}}{\mathrm{d}t} z_t = u_t^{\text{target}}(z_t \mid z_0, \mu_{z_1}) \quad \Longrightarrow \quad z_t \sim p_t(\cdot \mid z_0, \mu_{z_1})$$

The solution can be simply found using the reparameterization trick and deriving w.r.t. time $z_t = (1-t)z_0 + t\mu_{z_1} + \sigma\epsilon$

$$u_t^{\text{target}}(z_t \mid z_0, \mu_{z_1}) = \frac{d}{dt}[(1-t)z_0 + t\,\mu_{z_1}] = \mu_{z_1} - z_0 = u_y - z_0$$

# NoProp-FM: Flow Matching Loss

Goal: learn a neural vector field $u_\theta(z_t, x, t) \approx u_t^{\text{target}}(z_t \mid z_0, u_y)$.

- Given $(x, y) \sim p_{data}$, sample independently:

$$t \sim U[0,1], \quad z_0 \sim \mathcal{N}(0, I), \quad z_t \sim p_t(z_t \mid z_0, u_y).$$

- Define the local regression loss

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{(x,y),t,z_0,z_t} \, \| \, u_\theta(z_t, x, t) \; - \; (u_y - z_0) \|^2.$$

- Training occurs by sampling time steps independently, without requiring full forward or backward passes through time.
- Note that $u_\theta(z_t, x, t)$ is a single neural network that takes as input also the time $t$. Thus, the total number of parameters can be much lower than NoProp-DT, sharing them among time.

# NoProp-FM: Inference Phase

- After training, the inference problem is solved following the corresponding ODE

$$\frac{\mathrm{d}z_t}{\mathrm{d}t} = u_\theta(z_t, x, t), \quad z_0 \sim \mathcal{N}(0, I).$$

- Discretize the ODE; in this example we will use the Euler method:

$$z_{t+h} = z_t + h\, u_\theta(z_t, x, t), \quad h = 1/N.$$

- Simulate the ODE with $N$ timesteps. At $t = 1$, recover $z_1$ and predict:

$$\hat{y} = \arg\min_y \| z_1 - u_y \|_2^2.$$

- The paper also presents a formulation of the problem as a continuous diffusion model

# Experimental Validation

- The paper demonstrates NoProp's efficacy on various benchmarks: MNIST, CIFAR-10 and CIFAR-100.
- NoProp-DT achieves performance comparable to or better than backpropagation in the discrete-time setting for the same network.
- A detailed analysis of GPU memory usage between DT and backpropagation confirms a significant memory usage reduction (between 40 and 80%)
  - This makes NoProp highly attractive for Edge Computing and Low-Resource Environments

| Method | MNIST | | CIFAR-10 | | CIFAR-100 | |
|---|---|---|---|---|---|---|
| | Train | Test | Train | Test | Train | Test |
| Backprop (one-hot) | 100.0±0.0 | 99.46±0.06 | 99.98±0.01 | 79.92±0.14 | 98.63±1.34 | 45.85±2.07 |
| Backprop (dim=20) | 99.99±0.0 | 99.43±0.03 | 99.96±0.02 | 79.3±0.52 | 94.28±7.43 | 46.57±0.87 |
| NoProp-DT (one-hot) | 99.92±0.01 | 99.47±0.05 | 95.02±0.19 | 79.25±0.28 | 84.97±0.67 | 45.93±0.46 |
| NoProp-DT (dim=20) | 99.93±0.01 | 99.49±0.04 | 94.95±0.09 | 79.12±0.37 | 83.25±0.39 | 45.19±0.22 |
| Backprop | 0.87 GB | | 1.17 GB | | 1.73 GB | |
| NoProp-DT | 0.49 GB | | 0.64 GB | | 1.23 GB | |

# Challenges and Limitations

- **Limited empirical validation:** NoProp has so far only been demonstrated on small benchmarks (MNIST, CIFAR-10/100) and lacks evaluation on large-scale or real-world datasets.
- **Sensitive hyperparameter tuning:** Although it removes the need for global backpropagation parameters hypertuning, it introduces noise schedules, the embedding matrix, step count as well as all the hyperparameters for the training of each layer
- **Uncertain scaling to deeper models:** By design, each layer learns a local denoising task rather than building hierarchical features; its behavior beyond 10 layers remains untested.
- **Continuous variants lag in accuracy:** In experiments, NoProp-CT and NoProp-FM did not reach the same performance as the discrete-time NoProp-DT variant.
- **Scope limited to classification:** So far the method has only been applied to single-label classification; its adaptation to more complex domains (e.g. NLP) is still open.

# References and Code

 Qinyu Li and Yee Whye Teh and Razvan Pascanu, *NoProp: Training Neural Networks without Back-propagation or Forward-propagation*, 2025.

A simple implementation code can be found in the following Jupyter Notebook.