Pretraining with Random Noise for Fast and Robust Learning without Weight Transport
- Supplementary Material -

Jeonghwan Cheon[1]   **Sang Wan Lee**[1,2,3]   **Se-Bum Paik**[1]
[1]Department of Brain and Cognitive Sciences,
[2]Graduate School of Data Science, [3]Kim Jaechul Graduate School of AI
Korea Advanced Institute of Science and Technology, Daejeon, Republic of Korea
{jeonghwan518, sangwan, sbpaik}@kaist.ac.kr

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

A Experimental details and additional results for section 4.1

A.1 Network architecture and training details

Table S1: Parameters and settings used in the experiment.

Name	Setting
Dimensions	[784, 100, 10]
Activation function	ReLU
Number of random noise inputs	5×10^5
Batch size	64
Learning rate	0.0001
Optimizer	Adam

In Section 4.1, we utilized a two-layer feedforward neural network for classification, comprising 100 neurons in the hidden layer and employing the rectified linear unit (ReLU) as the activation function. Detailed hyperparameters related to the network architecture and training are presented in Table S1. The forward weights were initialized using He initialization [1], which is standard for networks utilizing ReLU activations. Specifically, the weights were sampled from a Gaussian distribution with a mean of zero and a standard deviation of:

$$\sigma = \sqrt{\frac{2}{n_{\text{in}}}}$$

, where n_{in} represents the number of input units for each layer. The biases were initialized to zero. For feedback alignment [2], the backward weights were randomly initialized from the same distribution as the forward weights. In contrast to backpropagation, these backward weights remained fixed throughout the entire training process and were not updated. This approach allowed us to circumvent weight transport while still facilitating alignment during random noise pretraining and data training.

a

Example random noise for training
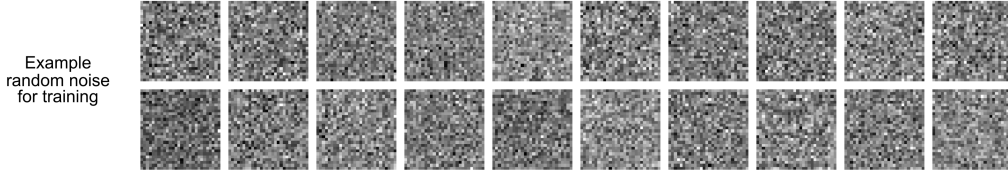
Figure S1: Samples of random noise utilized in the pretraining process. Each pixel value is randomly drawn from a zero-centered Gaussian distribution with a standard deviation of 1. The corresponding label is also randomly assigned from a discrete uniform distribution ranging from 0 to 9.

During the random noise pretraining phase, we sampled random input-output pairs at each iteration. Each sampled pair was used only once throughout the training process. Figure S1 presents an example of a random noise input sampled from a Gaussian distribution. This method of random noise pretraining was applied consistently across all subsequent experiments described in the main text.

It is important to note that random noise pretraining does not depend on specific conditions for the random input. To further investigate the robustness of this method, we tested weight alignment under various input conditions (Figure 1f). Specifically, we varied the standard deviation of the Gaussian distribution from 0 to 2, with a step size of 0.1. Additionally, we conducted similar tests using uniform distributions. The reported results include the mean and standard deviation of the alignment angle, calculated over ten independent trials.

2

B   Experimental details and additional results for section 4.2

B.1   Network architecture and training details

Table S2: Parameters and settings used in the experiment.

Name	Setting
Number of training data	5×10^3
Number of test data	5×10^3
Epochs	100
Batch size	64
Learning rate	0.0001
Optimizer	Adam

a
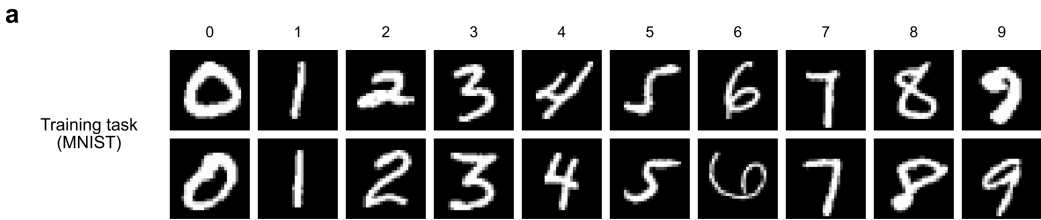
Figure S2: Samples from the MNIST dataset [3] used in the subsequent data training process. The dataset comprises images of handwritten digits across ten classes.

In Section 4.2, we examined the effect of random noise pretraining on subsequent data training. The network architecture and hyperparameters were consistent with those used in Section 4.1. Detailed hyperparameters related to the training process can be found in Table S2. We utilized a subset of the MNIST dataset (Figure S2), consisting of 5000 images for training and an additional 5000 images for testing. The data training was conducted over a total of 100 epochs, ensuring that the total number of inputs encountered during the data training process matched those in the random noise pretraining phase.

Although random noise pretraining does not conform to the conventional definition of an epoch — since it does not involve repeated training on the same inputs — we employed the term "epoch" to facilitate comparison between random noise training and data training along the same axis. Specifically, we divided random noise pretraining into 5000 iterations, analogous to the number of iterations in data training. Thus, one epoch utilized the same number of inputs (along with equivalent training time and computational cost) in both random noise pretraining and data training.

3

B.2   Training results
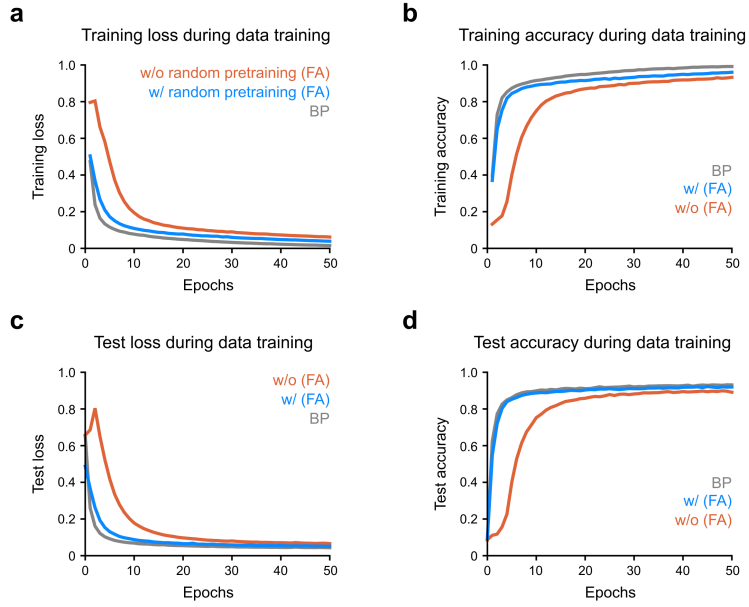
Figure S3: Loss and accuracy outcomes of the network under various training conditions. (a) Training loss. (b) Training accuracy. (c) Test loss. (d) Test accuracy. The blue and orange lines represent the network trained with feedback alignment, with and without random noise training, respectively. The gray lines denote the network trained using backpropagation.

In addition to the test accuracy during training presented in Figure 2b, Figure S3 displays the loss and accuracy for both the training and test sets. The results indicate that the network pretrained with random noise outperforms the network trained solely on data and achieves a learning efficiency comparable to that of backpropagation [4].

4

B.3 Additional results with deeper network

Table S3: Parameters and settings used in the experiment.

Name	Setting
Dimensions	[784, 100, 100, 10]
Activation function	ReLU
Number of random noise inputs	5×10^5
Number of training data	5×10^3
Number of test data	5×10^3
Epochs	100
Batch size	64
Learning rate	0.0001
Optimizer	Adam
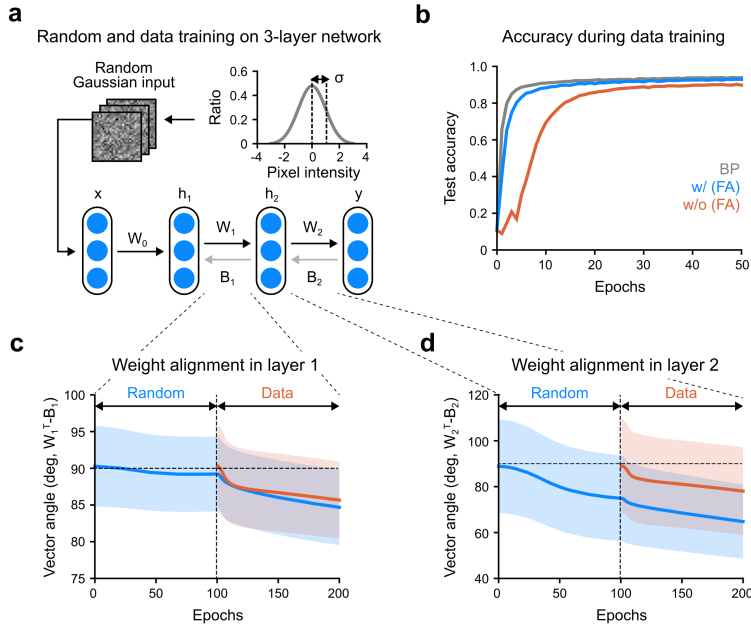
Figure S4: Training with random noise and data in a deeper network. (a) Three-layer network trained sequentially with random noise and data. (b) Test accuracy of the network under various training conditions. (c) Alignment angle of the weights ($\mathbf{W}_1$) and synaptic feedback ($\mathbf{B}_1$) in the second layer of the network. (d) Alignment angle of the weights ($\mathbf{B}_2$) and synaptic feedback ($\mathbf{W}_2$) in the third layer of the network.

We also investigated whether weight alignment improves in deeper networks. Specifically, we employed a three-layer feedforward neural network with a hidden layer size of 100 and examined the effect of random noise pretraining (Figure S4a). Detailed hyperparameters related to the network architecture and training can be found in Table S3. The results indicate that random noise pretraining remains beneficial, achieving faster learning and higher accuracy during subsequent data training (Figure S4b). Additionally, the network pretrained with random noise exhibited enhanced weight alignment, comparable to networks trained solely on data, in both the earlier layer (Figure S4c) and the deeper layer (Figure S4d).

5

B.4 Additional results with CIFAR-10

Table S4: Parameters and settings used in the experiment.

Name	Setting
Dimensions	[3072, 100, 10]
Activation function	ReLU
Number of random noise inputs	5×10^5
Number of training data	5×10^3
Number of test data	5×10^3
Epochs	100
Batch size	64
Learning rate	0.0001
Optimizer	Adam
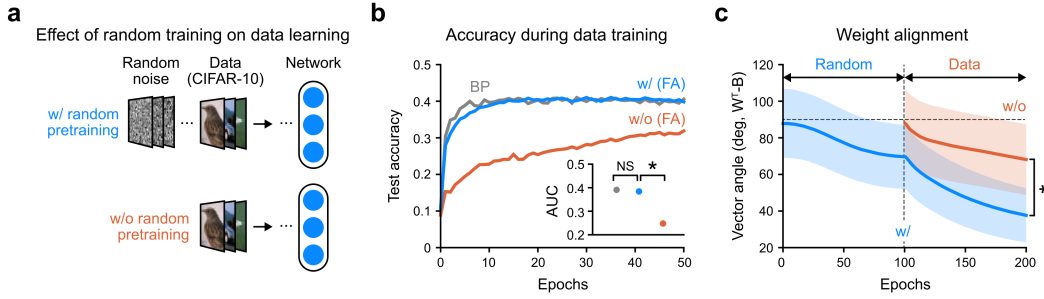
Figure S5: Effect of random noise training on subsequent CIFAR-10 [5] data training. (a) Design of the CIFAR-10 classification task. (b) Test accuracy during the training process, where the inset demonstrates the convergence speed of each training method, calculated by the AUC of the test accuracy. (c) Alignment angle between weights ($\mathbf{W}_1$) and synaptic feedback ($\mathbf{B}_1$) across random training and data training.
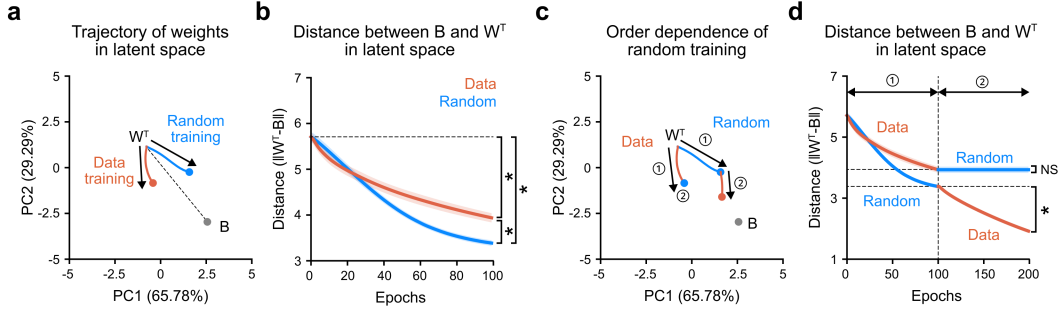
Figure S6: Weight update dynamic of random noise training and data training with CIFAR-10. (a) Trajectory of weights ($\mathbf{W}_1$) toward synaptic feedback ($\mathbf{B}_1$) in latent space obtained by PCA for random and data training. (b) Distance between the weights ($\mathbf{W}_1$) and the synaptic feedback ($\mathbf{B}_1$). (c) Order dependence of the trajectory of the weights ($\mathbf{W}_1$). (d) Distance between the weights ($\mathbf{W}_1$) and the synaptic feedback ($\mathbf{B}_1$) for different orders of random and data trainings.

To extend the results demonstrating the benefits of random noise pretraining on subsequent MNIST training, we conducted a similar experiment using the CIFAR-10 dataset, which contains more naturalistic images across ten different classes of objects and animals (Figure S5, Figure S6). Detailed hyperparameters related to the network architecture and training can be found in Table S4. These results emphasize that the benefits and characteristics of random noise pretraining, as observed in the main results, are not confined to a simple dataset like MNIST but are applicable to other datasets as well.

6

B.5 Validation of model performance across various image datasets and network depths

Table S5: Performance of each model with depth variation (2 – 5 layers) for five different datasets (MNIST, Fashion-MNIST, CIFAR-10, CIFAR-100 and STL-10). Each performance value (%) is presented as the mean $\pm$ standard deviation from three trials.

	Method		2-layer	3-layer	4-layer	5-layer
MNIST	BP		97.82 ± 0.03	97.86 ± 0.04	97.80 ± 0.16	97.71 ± 0.18
	FA	w/o	97.26 ± 0.07	96.95 ± 0.12	96.80 ± 0.20	95.58 ± 0.15
		w/	97.76 ± 0.07	97.56 ± 0.09	97.29 ± 0.24	97.03 ± 0.04
	$\triangle$ACC		▲0.49 ± 0.06	▲0.61 ± 0.11	▲0.48 ± 0.28	▲1.45 ± 0.19
F-MNIST	BP		88.87 ± 0.03	88.76 ± 0.09	88.55 ± 0.00	88.50 ± 0.13
	FA	w/o	87.47 ± 0.25	87.38 ± 0.20	86.25 ± 0.76	83.80 ± 2.22
		w/	88.26 ± 0.07	88.40 ± 0.02	87.87 ± 0.13	87.70 ± 0.21
	$\triangle$ACC		▲0.79 ± 0.31	▲1.02 ± 0.19	▲1.62 ± 0.86	▲3.90 ± 2.35
CIFAR-10	BP		54.01 ± 0.20	53.75 ± 0.22	53.16 ± 0.20	52.32 ± 0.03
	FA	w/o	50.54 ± 0.22	47.34 ± 0.83	45.98 ± 0.33	39.07 ± 2.85
		w/	53.58 ± 0.12	52.38 ± 0.05	51.54 ± 0.28	50.46 ± 0.81
	$\triangle$ACC		▲3.04 ± 0.20	▲5.04 ± 0.82	▲5.55 ± 0.09	▲11.39 ± 2.48
CIFAR-100	BP		24.55 ± 0.10	24.71 ± 0.04	24.54 ± 0.08	24.29 ± 0.13
	FA	w/o	20.17 ± 0.30	17.17 ± 0.54	14.56 ± 0.13	8.22 ± 1.36
		w/	24.45 ± 0.10	22.76 ± 0.42	18.69 ± 0.65	13.33 ± 1.54
	$\triangle$ACC		▲4.28 ± 0.38	▲5.58 ± 0.18	▲4.14 ± 0.52	▲5.11 ± 1.31
STL-10	BP		42.72 ± 0.20	43.00 ± 0.28	42.87 ± 0.08	42.56 ± 0.22
	FA	w/o	36.21 ± 0.91	35.04 ± 0.92	36.74 ± 0.24	31.50 ± 5.10
		w/	41.01 ± 0.16	41.52 ± 0.15	41.38 ± 0.14	39.04 ± 0.59
	$\triangle$ACC		▲4.81 ± 0.86	▲6.49 ± 1.03	▲4.64 ± 0.35	▲7.54 ± 5.37

We benchmarked the final accuracy of networks trained using baseline feedback alignment (FA, w/o random pretraining), feedback alignment with random noise pretraining (FA, w/ random pretraining: our model), and backpropagation (BP) (Table S5). Specifically, we evaluated these models at depths ranging from 2 to 5 layers (Figure S7). Furthermore, we compared these models across several complex and large datasets, including MNIST [3], Fashion-MNIST [6], CIFAR-10 [5], CIFAR-100 [5], and STL-10 [7] (Figure S8). To ensure full convergence during training, we extended the training duration until convergence was confirmed, with validation accuracy showing no further increase (patience: 10 epochs).

We observed that incorporating random noise training at various network depths and across diverse datasets consistently resulted in higher final accuracy, often comparable to that achieved with backpropagation. As the network depth increased, the disparity in final accuracy between models with and without random noise training widened (Figure S7). Importantly, the beneficial impact of random noise training on final accuracy became significantly more pronounced as task complexity increased (Figure S8). For instance, while random noise training improved accuracy by 0.49% in MNIST, it increased by 0.79% in Fashion-MNIST, 3.04% in CIFAR-10, 4.28% in CIFAR-100, and 4.81% in STL-10. This gap widened further as the network depth increased.

7
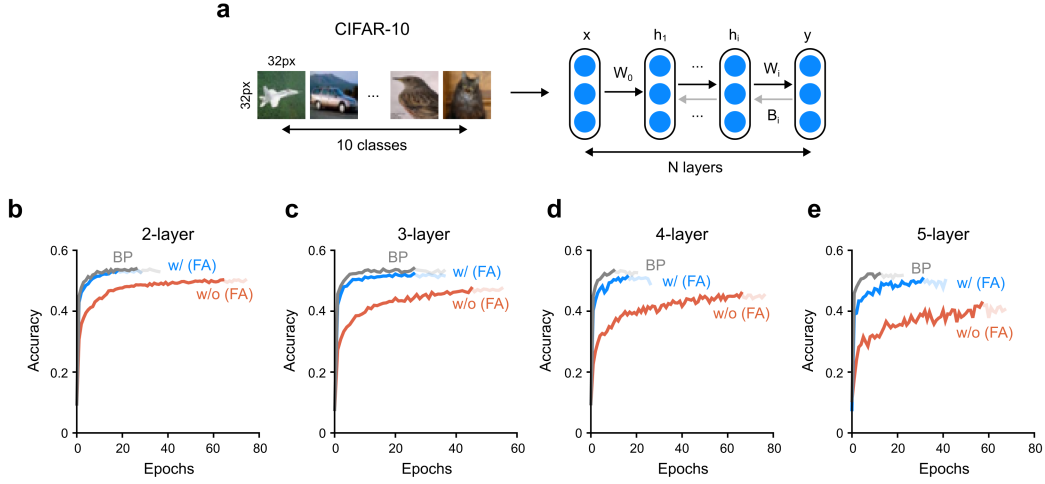
Figure S7: Model performance across different network depths. (a-d) Validation accuracy measured at each epoch during data training. Each colored line represents a network trained using either feedback alignment or backpropagation. Experiments were conducted with various depths of MLP: (a) two-layer, (b) three-layer, (c) four-layer, and (d) five-layer.
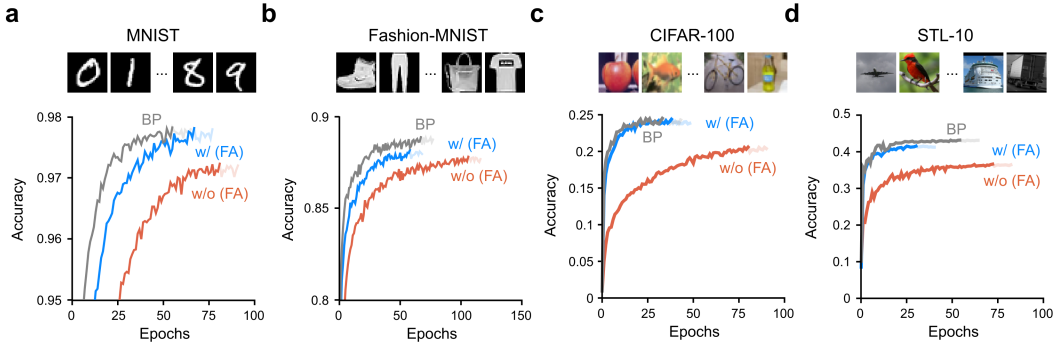
Figure S8: Model performance across different image datasets. (a-d) Validation accuracy is measured at each epoch. (a) MNIST, (b) Fashion-MNIST, (c) CIFAR-100, (d) STL-10.

Indeed, previous studies on biologically plausible backpropagation without weight transport often focus on simple network structures and easy datasets. This limitation results in lower learning capacity and presents challenges in scaling up due to biological constraints that preclude weight transport. Through our additional experiments, we have demonstrated that random noise pretraining consistently outperforms baseline feedback alignment, even in deeper networks. However, we also observed that the performance gap with backpropagation widens as the number of layers increases. This challenge partly arises from the difficulty of achieving precise weight alignment in the early layers using feedback alignment without weight transport. While our approach significantly enhances the learning efficiency of feedback alignment algorithms, the issue of the performance gap in deep networks compared to backpropagation remains unresolved.

8

B.6 Validation of model performance across various training hyperparameters
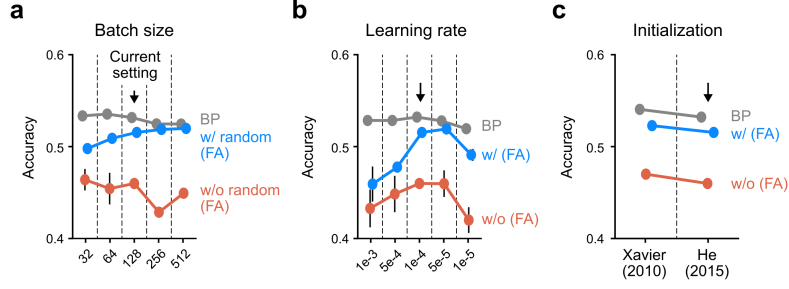
Figure S9: Performance consistency across different hyperparameter settings and initializations. (a-c) Accuracy of networks trained with varying hyperparameters: (a) batch size, (b) learning rate, and (c) initialization.

Our results are based on hyperparameters typically selected for effective learning in both feedback alignment and backpropagation. We conducted and compared baseline feedback alignment (FA, w/o random pretraining), feedback alignment with random noise pretraining (FA, w/ random pretraining), and backpropagation (BP) under carefully controlled conditions. We controlled hyperparameters such as batch size, learning rate, and initial forward weights to isolate differences attributable to the learning algorithms. In each trial, we initialized a single network randomly and duplicated it into three, starting with identical weights. These networks were then trained with the same hyperparameters but using different learning methods.

Additionally, we confirmed that our results were not specific to the choice of hyperparameters — our findings generalized to other conditions. Further experiments using the CIFAR-10 dataset demonstrated that varying hyperparameters (batch size, learning rate) and initialization within a reasonable range consistently reproduced similar results (Figure S9).

9

B.7 Model validation with initialization matching forward and backward weights
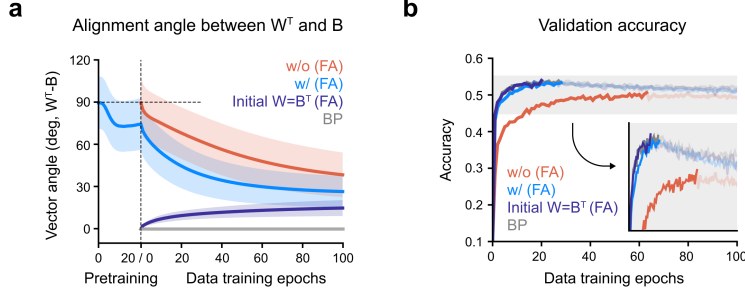
Figure S10: Model validation: Initial alignment of weights ($\mathbf{W}_l$) and synaptic feedback ($\mathbf{B}_l$) ensures learning efficiency comparable to backpropagation. (a) Alignment angle between forward and backward weights using various training methods (purple: $\mathbf{W}_l$ initialized with $\mathbf{B}_l^T$) and its corresponding validation accuracy (b).

Our results indicate that pretraining with random noise effectively pre-aligns the forward weights with the backward weights, enhancing learning efficiency during subsequent data training. To support our claims, we conducted additional experiments using the CIFAR-10 dataset, which further validated our model. Specifically, we initialized the forward weights $\mathbf{W}_l$ to match the transpose of backward synaptic feedback $\mathbf{B}_l$ and proceeded with training (Figure S10. This approach yielded gradients and alignment similar to those observed in backpropagation during training. Consequently, we confirmed that this setup demonstrates learning efficiency comparable to that of backpropagation.

During the learning process, particularly in the initial stages, weight alignment may slightly degrade but largely remains intact. This baseline example illustrates that achieving initially aligned forward and backward weights allows for learning that is comparable to backpropagation, even without the enforced synchronization of backward weights during training. This serves as a specific instance highlighting the validity of random noise training in aligning forward and backward weights in a biologically plausible manner prior to data-driven learning.

10

B.8 Computational benefits of random noise training considering total training epochs
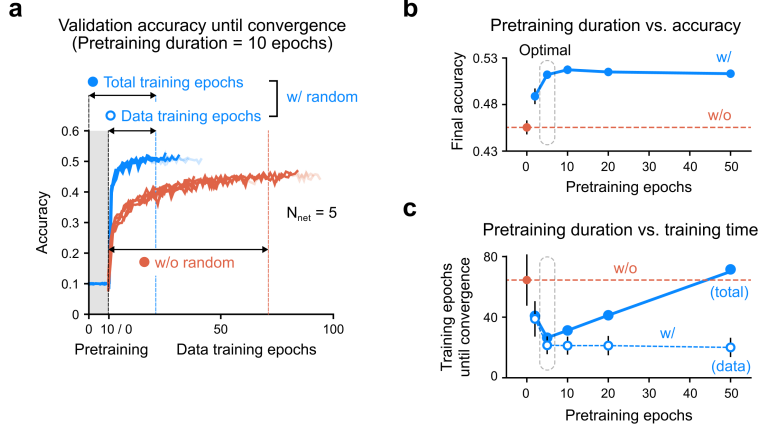
Figure S11: Comparison of training time until convergence including pretraining. (a) Learning curves of a four-layer MLP pretrained for 10 epochs and subsequently trained on CIFAR-10. (b-c) Repeated experiments with different pretraining durations. (b) Accuracy. (c) Training epochs until convergence.

We confirmed that random noise pretraining accelerates convergence and can reduce computational resources, even when considering the total training duration (pretraining + data training) (Figure S11). By varying the duration of noise training, we demonstrated that, despite the additional time required for noise training, the total training time remained significantly shorter than that of training with data alone in most conditions. We conducted subsequent data training on networks pretrained with random noise for 2, 5, 10, 20, and 50 epochs, measuring the epochs required for training to converge (when validation accuracy no longer increased, with patience of 10 epochs). We maintained consistency in the number of samples used per epoch during both random noise training and subsequent data training, ensuring direct comparability of epoch times.

We found that longer periods of random noise training resulted in shorter subsequent data training times to achieve convergence. As reported previously, when comparing data training alone, the learning time was consistently much shorter in networks pretrained with noise. Notably, even when the time for noise training was included, the overall training duration for the noise-training algorithm remained substantially shorter than that for training with data alone in most conditions. Based on this analysis, we estimated an optimal duration for random noise pretraining that ensures the most efficient use of resources (Figure S11b, c, Optimal). We also found that at this optimal duration, the improvement in accuracy remained significant. These additional experimental results demonstrate that, when considering the resources expended in random noise training, it represents a more efficient approach (reduced training time and overall computation) compared to training with data alone.

11

C Experimental details and additional results for section 4.3

C.1 Network architecture and training details

Table S6: Parameters and settings used in the experiment.

Name	Setting
Dimensions	[784, 100, 100, 10]
Activation function	ReLU
Number of random noise inputs	5×10^5
Batch size	64
Learning rate	0.0001
Optimizer	Adam

In Section 4.3, we utilized a three-layer feedforward neural network for classification, comprising 100 neurons in the hidden layer and employing the ReLU as the activation function. Detailed hyperparameters of the network architecture and training can be found in Table S6.

C.2 Random noise training reduces the effective dimensionality of weights
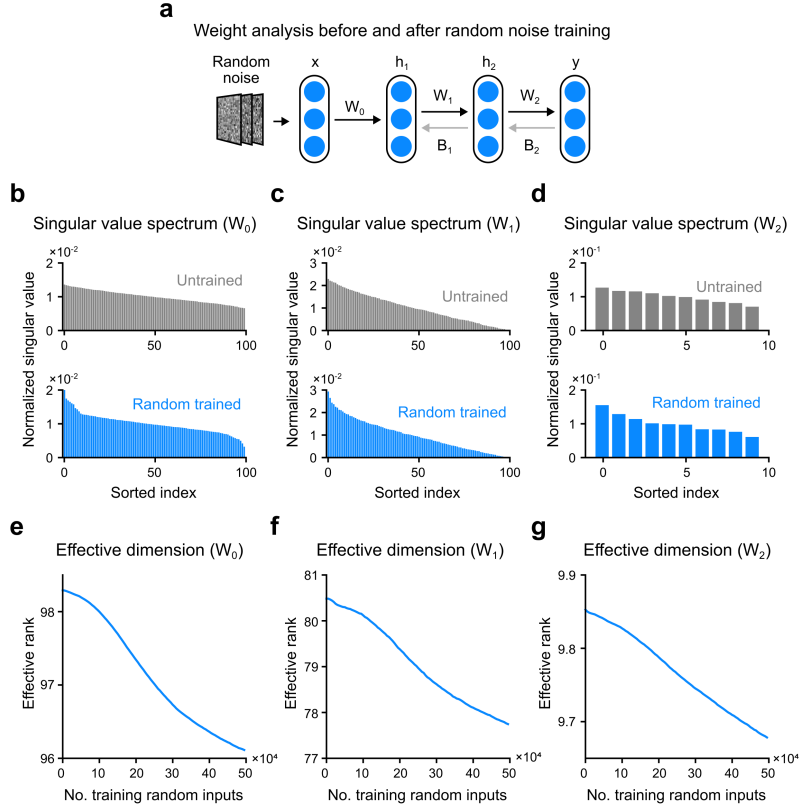
Figure S12: Effective dimensionality of weights in various layers. (a) Architecture of the network used in the experiment, in which the weights of the three layers are analyzed. (b-d) Singular value spectrum of weights in each layer of the untrained and randomly trained networks. (e-g) Effective dimensionality of weights in each layer during random noise training.

We demonstrated that training with random noise effectively reduces the dimensionality of the weights, resulting in the learning of simpler solutions with lower rank. In Figure 4, we only illustrated the decrease in dimensionality in the first layer. Figure S12 shows that the same trend is also observed in the remaining layers.

12

C.3 Enhanced generalization with various training sizes

Table S7: Parameters and settings used in the experiment.

Name	Setting
Dimensions	[784, 100, 100, 10]
Activation function	ReLU
Number of training data	[100, 200, 400, 800, 1600]
Number of test data	1×10^3
Epochs	500
Batch size	64
Learning rate	0.0001
Optimizer	Adam
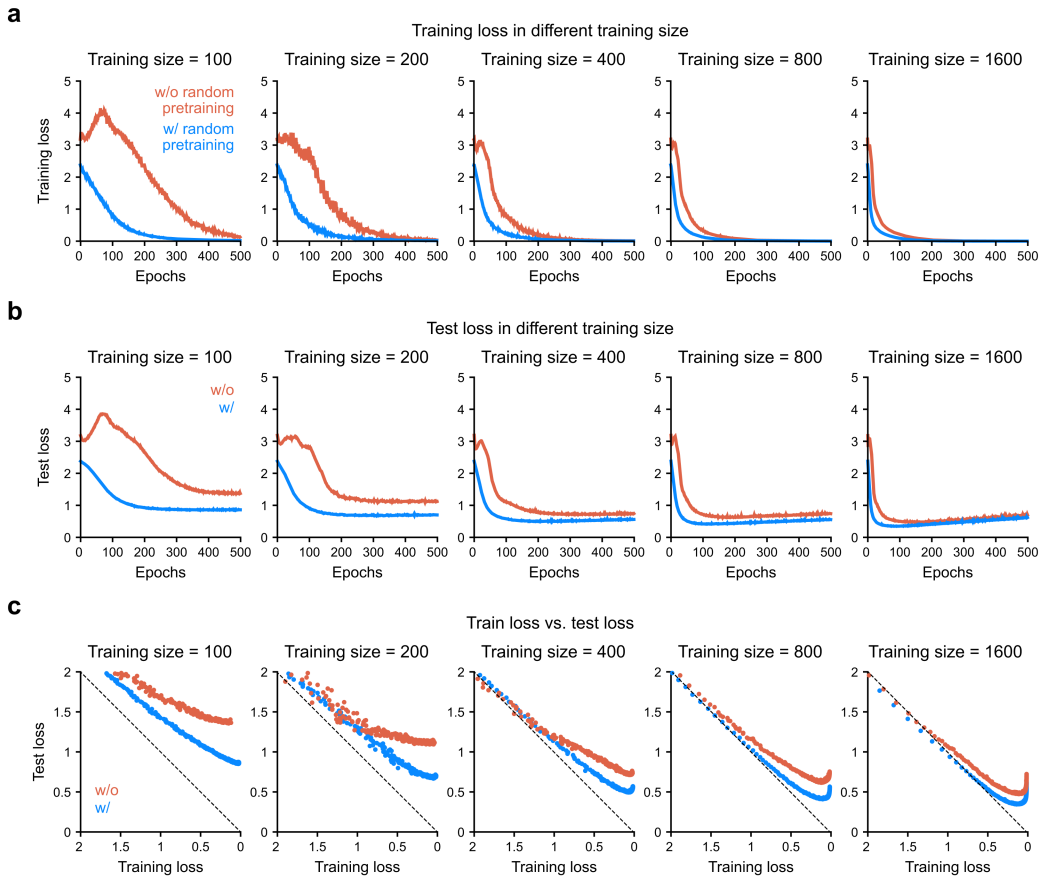
Figure S13: Training process and generalization error for different training set sizes. (a) Training loss. (b) Test loss. (c) Comparison of training loss versus test loss. Each column represents a network trained with a different training set size, with orange lines indicating networks trained solely on data and blue lines representing randomly pretrained networks.

13
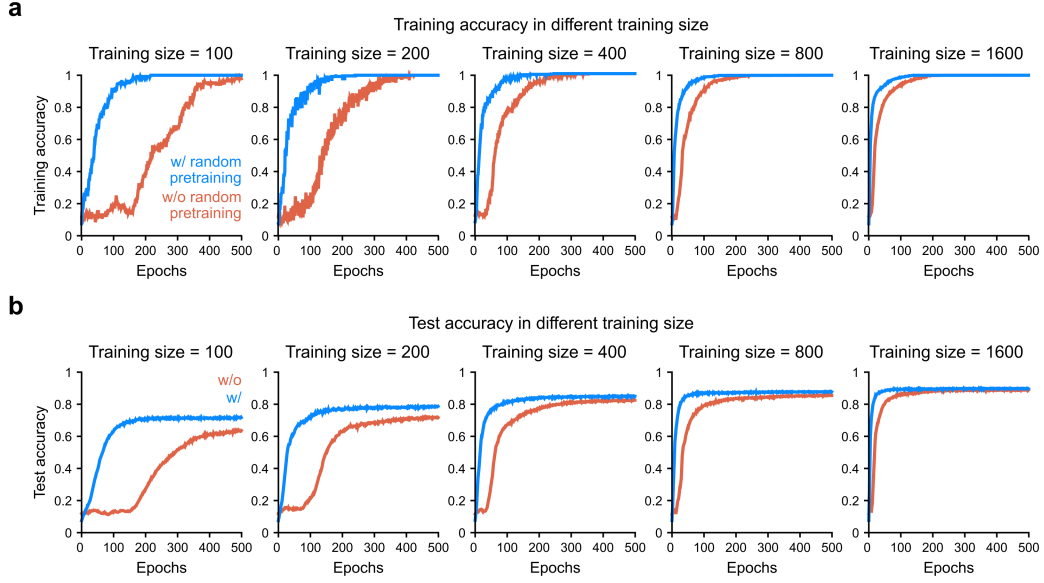
Figure S14: Training and test accuracy for different training set sizes. (a) Training accuracy. (b) Test accuracy. Each column represents a network trained with a different training set size, with orange lines indicating networks trained solely on data and blue lines representing randomly pretrained networks.

We systematically analyzed the effect of random noise pretraining on generalization across various training sizes. Specifically, we trained three-layer neural networks with different training data sizes, including 100, 200, 400, 800, and 1600 samples. Detailed hyperparameters of the network architecture and training can be found in Table S7. Across the different training sizes, we measured the generalization error (Figure S13) and accuracy (Figure S14). We observed that the generalization error was significantly reduced in networks pretrained with random noise, regardless of the training size. The summarized results are displayed in Figure 4f.

14

C.4 Enhanced generalization with various network depths

Table S8: Parameters and settings used in the experiment.

Name	Setting
Input, hidden layer, output dimensions	784, 100, 10
Number of hidden layers	[3, 4, 5, 6, 7]
Activation function	ReLU
Number of training data	1600
Number of test data	1×10^3
Epochs	500
Batch size	64
Learning rate	0.0001
Optimizer	Adam
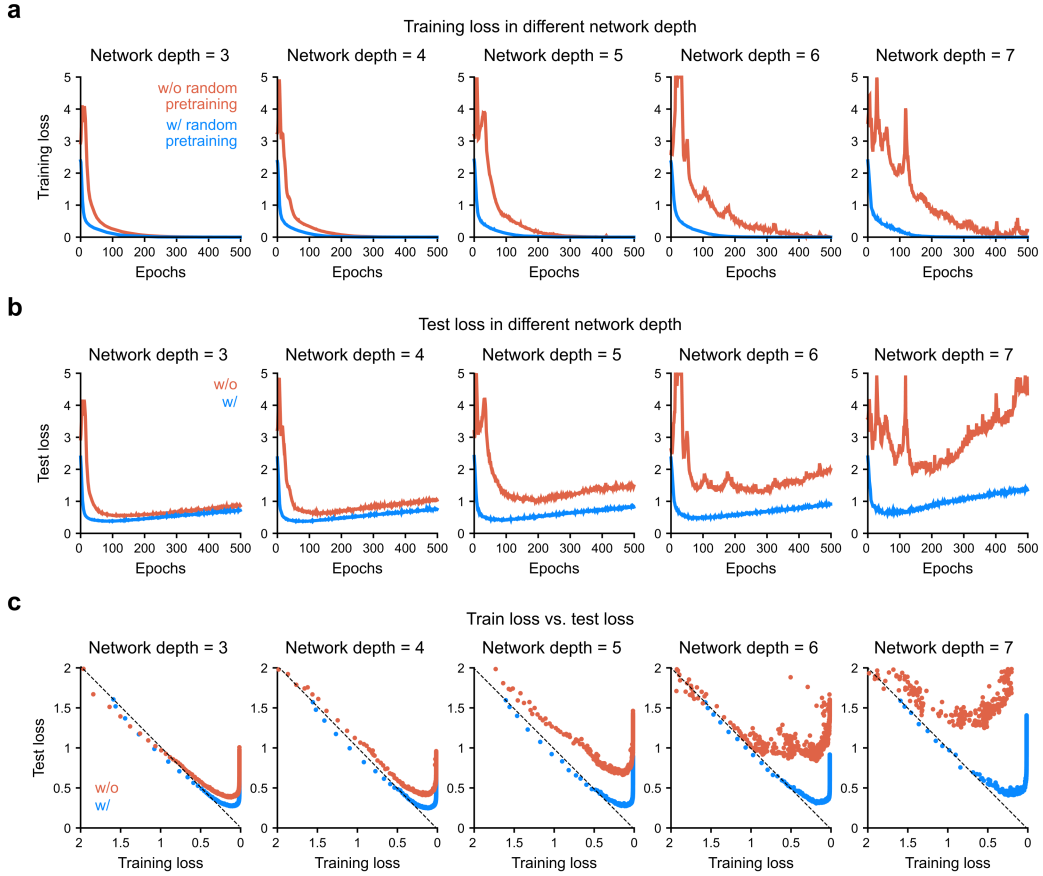
Figure S15: Training process and generalization error for different network depths. (a) Training loss. (b) Test loss. (c) Comparison of training loss versus test loss. Each column represents a network trained with a different training set size, with orange lines indicating networks trained solely on data and blue lines representing randomly pretrained networks.

15

a

Training accuracy in different network depth

b

Test accuracy in different network depth
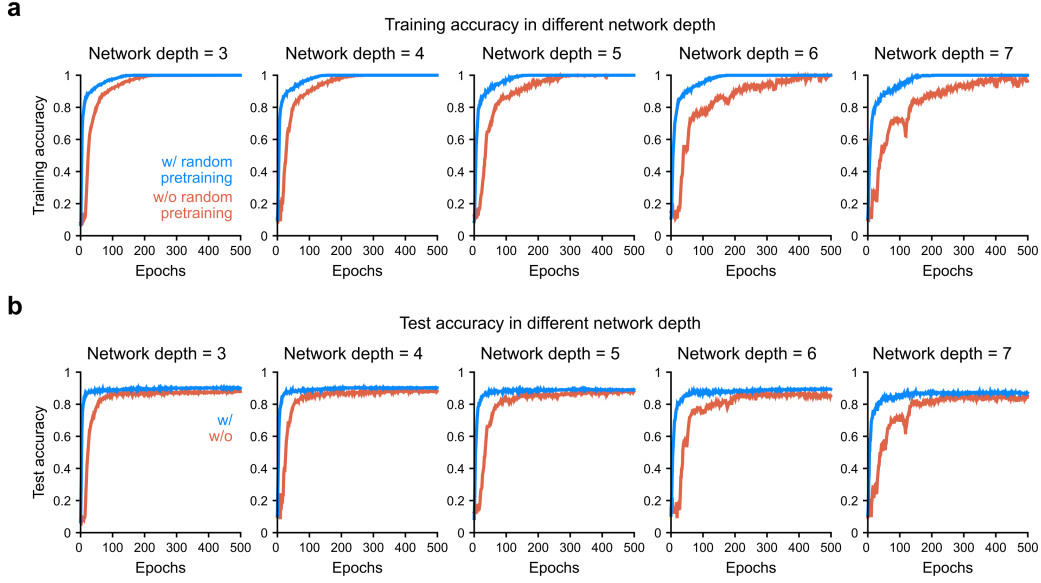
Figure S16: Training and test accuracy with different network depths. (a) Training accuracy. (b) Test accuracy. Each column represents a network trained with a different training set size, with orange lines indicating networks trained solely on data and blue lines representing randomly pretrained networks.

Next, we analyzed the effect of random noise pretraining on generalization across various network depths. In this experiment, we fixed the training data size and varied the number of hidden layers. We tested three-, four-, five-, six-, and seven-layer neural networks with a hidden layer size of 100. Detailed hyperparameters of the network architecture and training can be found in Table S8. Across the different network depths, we measured the generalization error (Figure S15) and accuracy (Figure S16). We observed that the generalization error was significantly reduced in networks pretrained with random noise, regardless of the network depth. Additionally, we assessed the effective rank of the learned features in this experiment and confirmed that random noise pretraining significantly reduces the dimensionality of the solution (Figure 4g, h). This may provide an explanatory scenario for how random noise pretraining enhances generalization across different network depths.

16

C.5   Generalization test for out-of-distribution

a

b

c
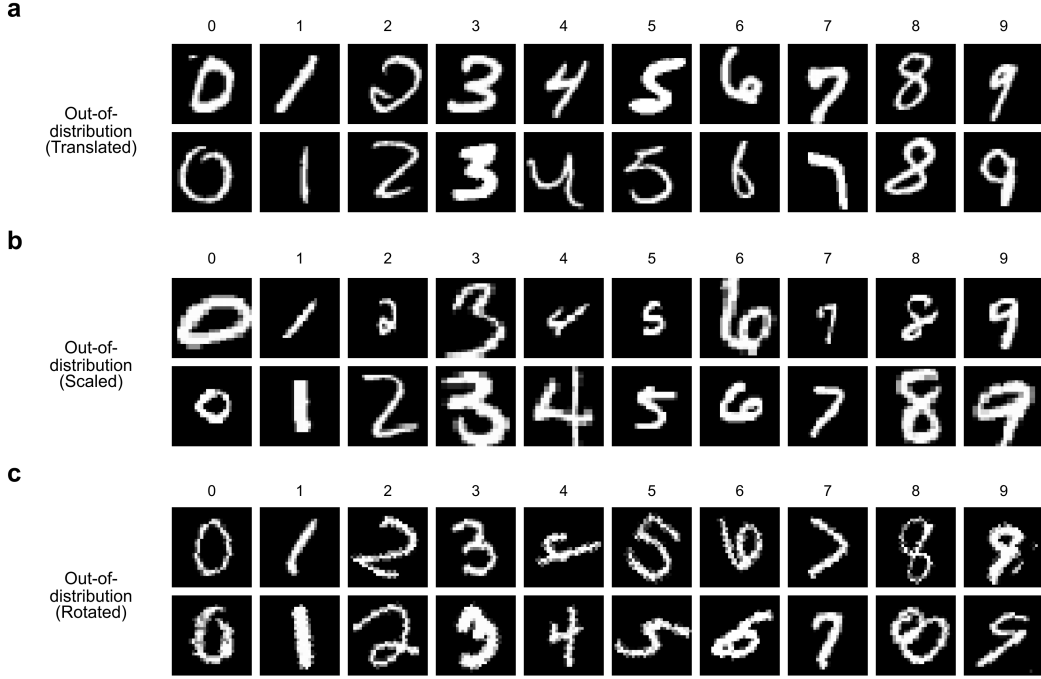
Figure S17: Datasets used in the out-of-distribution generalization test with the transformed dataset. (a) MNIST dataset used in the training in-distribution case. (b-d) Transformed MNIST dataset utilized in the generalization test for the out-of-distribution case. (b) Translated MNIST dataset, where each image is randomly translated in the range of $[-5\%, 5\%]$ of the input image size on the x- and y-axis. (c) Scaled MNIST dataset, where each image is randomly scaled in the range of $[0.8, 1.2]$. (d) Rotated MNIST dataset, where each image is randomly rotated in the range of $[-25, 25]$ degrees.

a
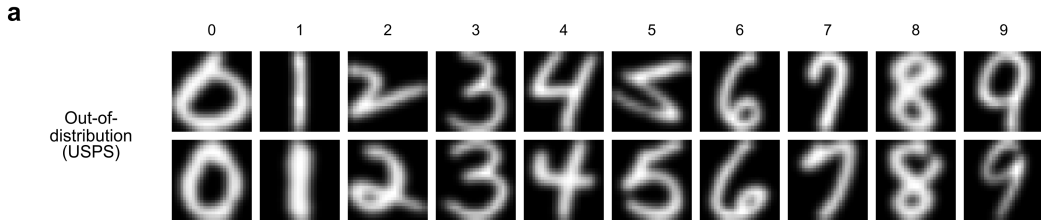
Figure S18: Dataset used in the out-of-distribution generalization test with the benchmark dataset. (a) MNIST dataset [3] used to train the networks in the out-of-distribution case. (b) USPS dataset [8] resized from (16×16) to (28×28), utilized in the generalization test for the out-of-distribution case.

We investigated whether networks trained with random noise can generalize to out-of-distribution data. To assess the model's performance on transformed MNIST, we created a custom dataset by applying translations, scaling, and rotations. Example data used in the experiment can be seen in Figure S17. The results are presented in Figure 5b. Additionally, we evaluated the model's performance on a benchmark out-of-distribution generalization dataset, USPS (Figure S18). The corresponding results are shown in Figure 5c.

17

D   Experimental details and additional results for section 4.4

D.1   Network architecture and training details

Table S9: Parameters and settings used in random noise pretraining.

Name	Setting
Dimensions	[784, 100, 100, 10]
Activation function	ReLU
Number of random noise inputs	5×10^5
Batch size	64
Learning rate	0.0001
Optimizer	Adam

In Section 4.4, we employed a three-layer feedforward neural network for classification, comprising 100 neurons in the hidden layer and utilizing the ReLU as the activation function. Detailed hyperparameters of the network architecture and random noise training can be found in Table S9.

Table S10: Parameters and settings used in meta-loss measurement.

Name	Setting
K	10
Inner steps	10
Inner learning rate	0.001
Inner optimizer	Adam
Task distribution (Dataset)	MNIST, F-MNIST, K-MNIST

During training with random noise, we measured the meta-loss, which evaluates the network's ability to quickly adapt to various tasks. To compute the meta-loss, we utilized the definition from model-agnostic meta-learning (MAML) [9], a widely used meta-learning algorithm. We constructed a task distribution consisting of three datasets: MNIST, Fashion-MNIST, and Kuzushiji-MNIST. We sampled batches of tasks from these datasets and trained the copied networks on the sampled tasks for a few gradient steps. Subsequently, we summed the loss from the adapted networks and used this sum as the meta-loss. Detailed hyperparameters for the meta-loss measurement can be found in Table S10. Unlike conventional meta-learning approaches such as MAML — where the meta-loss is employed to optimize the network's initial parameters for rapid adaptation — we simply measured the meta-loss during random noise training without using it to update the network.

Table S11: Parameters and settings used in task adaptation.

Name	Setting
Number of training data	5×10^3
Number of test data	5×10^3
Epochs	100
Batch size	64
Learning rate	0.0001
Optimizer	Adam
Task distribution (Dataset)	MNIST, F-MNIST, K-MNIST

Next, we compared untrained networks with networks pretrained using random noise in terms of their adaptation to each task. We copied the networks and adapted them to three tasks: MNIST, Fashion-MNIST, and Kuzushiji-MNIST. Detailed hyperparameters used in task adaptation can be found in Table S11.

18

D.2 Measurement of meta-loss

a

b

c
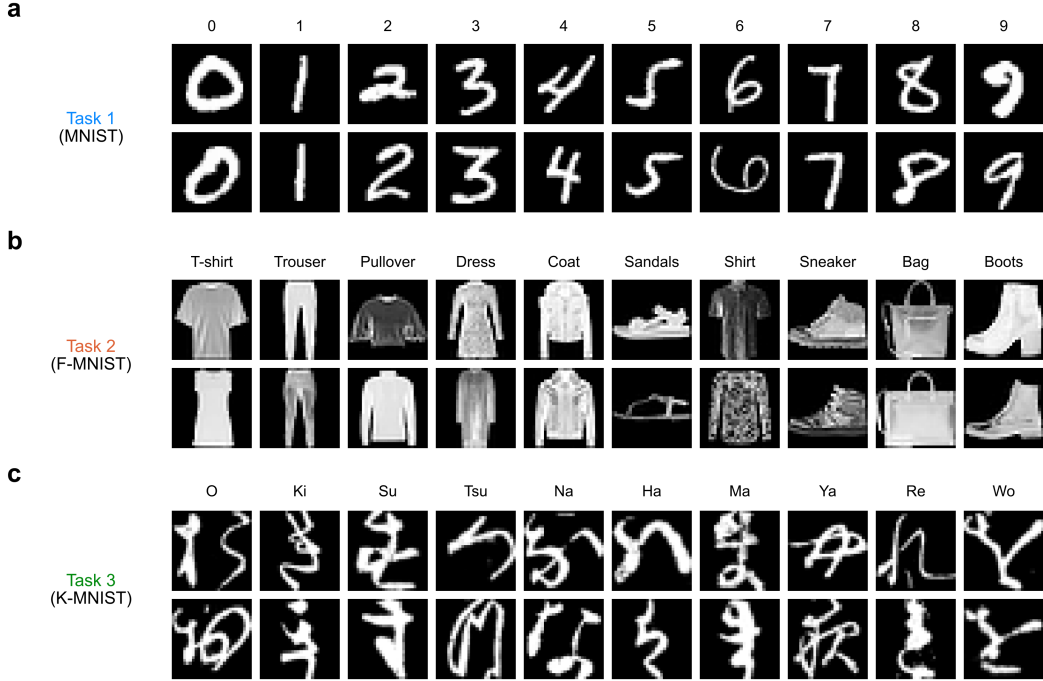
Figure S19: Tasks used to measure the meta-loss. (a) Task 1: MNIST [3] classification task. (b) Task 2: Fashion-MNIST [6] classification task. (c) Task 3: Kuzushiji-MNIST [10] classification task.

a
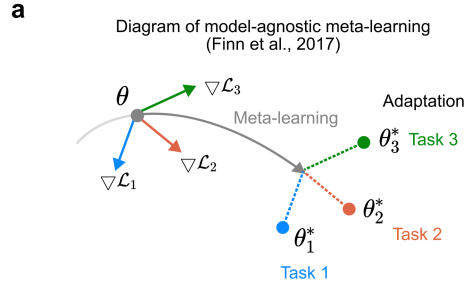
Figure S20: Diagram of meta-learning. Each colored arrow indicates the gradient update direction corresponding to a specific task. The gray arrow represents the overall learning direction of the meta-learning process. Each colored point denotes adaptation to a specific task from the meta-learned network.

We considered three tasks: MNIST classification, Fashion-MNIST classification, and Kuzushiji-MNIST classification (Figure S19) to measure the meta-loss and assess the fast adaptation of networks pretrained with random noise. The diagram of the meta-learning process (Figure S20) illustrates that optimizing the network to minimize the meta-loss enables quicker adaptation to task distributions. Notably, our results (Figure 6a) demonstrate that random noise pretraining efficiently reduces the meta-loss without the need for exposure to task-specific data. Additionally, the trajectory of weights in latent space during adaptation to various tasks (Figure 6b) follows a pattern similar to that depicted in the diagram. This highlights the role of random noise pretraining as a form of meta-learning that facilitates rapid adaptation.

19

E   Our contributions

Our goal is to enhance current strategies in deep learning by drawing insights from brain function. There exists a significant accuracy gap between backpropagation and biologically plausible learning strategies that do not involve weight transport. While backpropagation is effective, it is computationally intensive, requiring dynamic memory access for weight transport (i.e., accessing forward weights in memory to compute backward updates). Pretraining with random noise and aligning forward-backward weights achieves similar outcomes without relying on weight transport. Our interest is not solely in attaining performance comparable to backpropagation, but rather in achieving this without weight transport. Feedback alignment (and random noise pretraining) does not require weight transport and relies exclusively on local information. In the context of energy-efficient neuromorphic chip engineering, feedback alignment is sometimes employed for learning, albeit with some performance trade-offs. Our results demonstrate that such sacrifices are minimal in our model. Overall, our findings provide insights into how to reach levels of learning efficiency similar to backpropagation while circumventing the need for weight transport.

F   Experimental details

Data availability. The datasets used in this study are publicly available: `http://yann.lecun.com/exdb/mnist/` (MNIST [3]), `https://github.com/zalandoresearch/fashion-mnist` (Fashion-MNIST [6]), `https://github.com/rois-codh/kmnist` (Kuzushiji-MNIST [10]), `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#usps` (USPS [8]), `https://www.cs.toronto.edu/~kriz/cifar.html` (CIFAR-10 and CIFAR-100 [5]), `https://cs.stanford.edu/~acoates/stl10/` (STL-10 [7]).

Software used. Python 3.11 (Python software foundation) with PyTorch 2.1 and NumPy 1.26.0 was used to perform the simulation and the analysis. SciPy 1.11.4 was used to perform the statistical test and analysis. The code used in this work is available at `https://github.com/cogilab/Random`.

Computing resources. All simulations were performed on a computer with an Intel Core i7-11700K CPU and an NVIDIA GeForce GTX 1080 GPU. The simulation code was parallelized using PyTorch's built-in parallelization to utilize the GPU resources efficiently.

20

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[2] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1):13276, 2016.

[3] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[4] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 10 1986.

[5] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[6] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[7] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings, 2011.

[8] Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554, 1994.

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[10] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.

21