Alternatives to backpropagation training of (deep) neural models

ANDREA COSSU – ANDREA.COSSU@DI.UNIPI.IT

Once upon a time…

Here I am!

Bachelor Degree in Computer Science @ UniPisa

Master Degree in Computer Science – AI curriculum @ UniPisa

PhD in Data Science @ SNS

Once upon a time…

Here I am!

Bachelor Degree in Computer Science @ UniPisa

Master Degree in Computer Science – AI curriculum @ UniPisa

PhD in Data Science @ SNS

Continual / Lifelong learning

ContinualAI

Avalanche

https://www.continualai.org/

Once upon a time...

Here I am!

Bachelor Degree in Computer Science @ UniPisa

Master Degree in Computer Science – AI curriculum @ UniPisa

PhD in Data Science @ SNS

Post-doc researcher @ CS department, UniPisa

RTD-A @ CS department, UniPisa

Once upon a time…

Here I am!

Bachelor Degree in Computer Science @ UniPisa

Master Degree in Computer Science – AI curriculum @ UniPisa

PhD in Data Science @ SNS

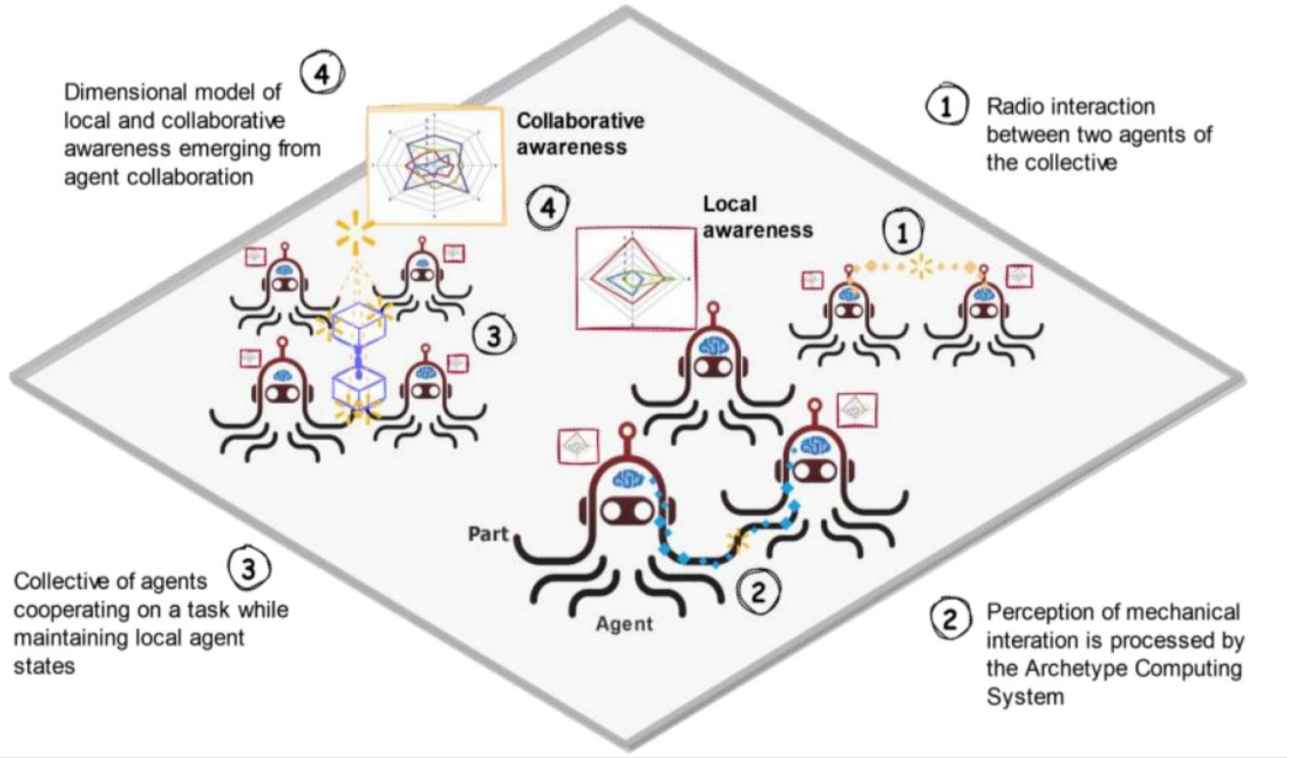
Post-doc researcher @ CS department, UniPisa

RTD-A @ CS department, UniPisa

EMERGE

https://eic-emerge.eu/

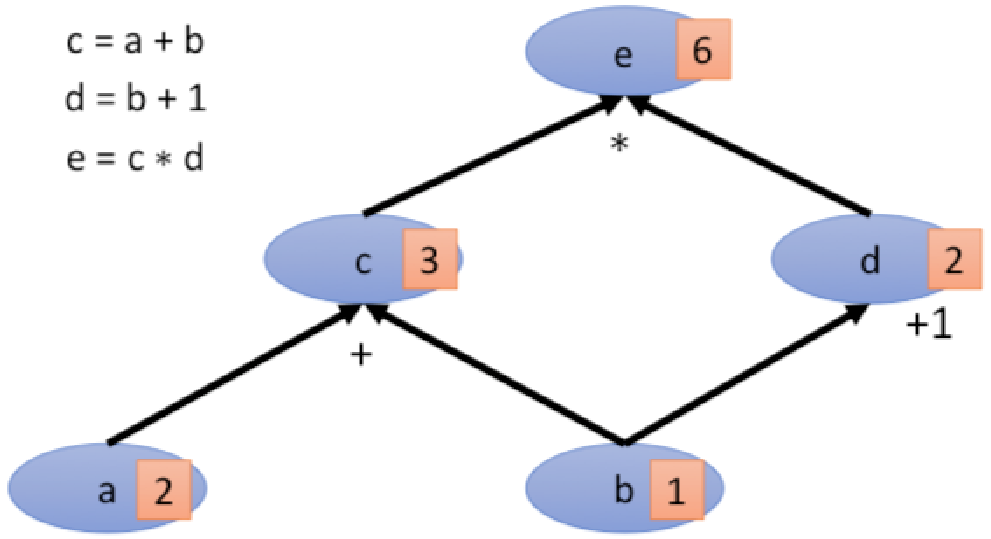
Emergent awareness from minimal collectives



4 Dimensional model of local and collaborative awareness emerging from agent collaboration
Collaborative awareness
4
Local awareness
1 Radio interaction between two agents of the collective
1
3
Part
Agent
2
3 Collective of agents cooperating on a task while maintaining local agent states
2 Perception of mechanical interation is processed by the Archetype Computing System

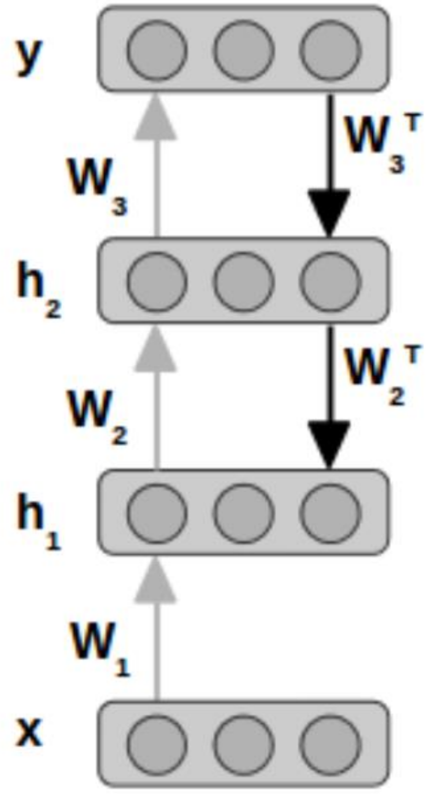
End-to-end backpropagation

- «Differentiable programming»: $y = f(x;\theta) = f_\theta(x)$
  - Computational graph: nodes are variables, edges are operations
- We want to compute $\frac{\partial y}{\partial \theta}$ → chain rule!
  - Reverse-mode automatic differentiation: y fixed, θ varies (**leaf nodes)**
  - Backpropagation → reverse-mode automatic differentiation through chain rule
- $f: R^I \rightarrow R^O$ → O steps required
  - O= 1 for our common case (scalar loss)
  - That's why *forward-mode* is **not** used



Backpropagating errors

- We have $\mathrm{y} = \mathrm{f}_\theta(\mathrm{x})$, our differentiable learning model
- We measure the prediction error through the loss function $L(y, d)$
  - E.g. MSE: $\frac{1}{2} e^T e, \qquad e = y - d$
- Once $\nabla_\theta L(y, d)$ is computed → SGD, Adam…
- $a_l = W_l h_{l-1} + b_l,\ h_l = \sigma(a_l)$ → linear activation in output layer
- Output layer error: easy
  - $\frac{\boldsymbol{\partial L}}{\boldsymbol{\partial a_3}} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial a_3} = \ \boldsymbol{\delta_3} = \ e$
  - $\frac{\partial L}{\partial W_3} = \delta_3 \frac{\partial a_3}{\partial W_3} = e\ h_3^T$
- Hidden layers:
  - $\frac{\partial L}{\partial W_l} = ((\boldsymbol{W_{l+1}^T}\delta_{l+1}) \circ \sigma'(a_l))\ \ h_l^T, \qquad \forall j = 1, 2$



Very effective, but we don’t like it

- BP enabled efficient training of deep architectures
  - de-facto standard for >20 years, still is

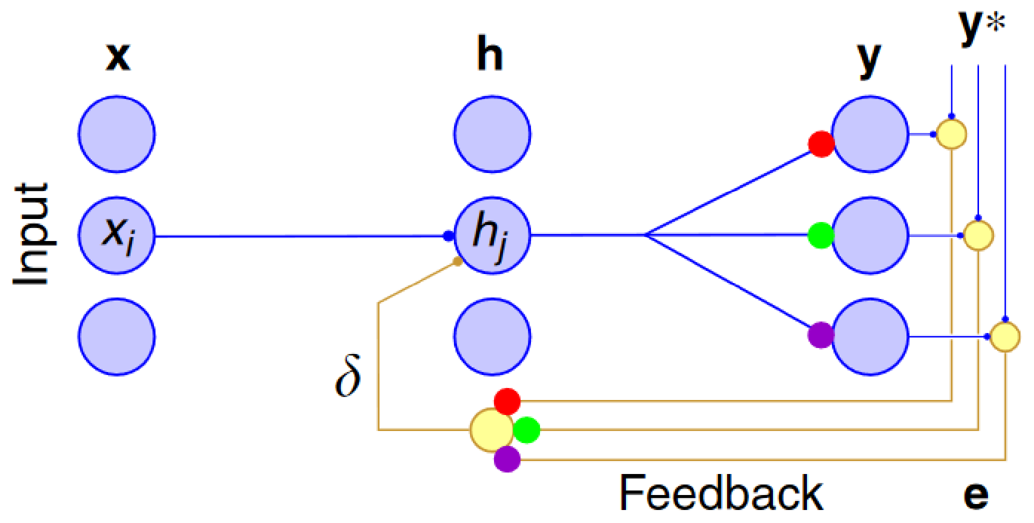
Why don’t we like it?

- Lack of modularity
  - Sparsity is *enforced* with post-hoc pruning or pre-designed block-wise architectures
- «Biologically» unrealistic constraints:
  - Two learning circuits (forward+backward)
  - Symmetric weights (weight transport problem)
  - Non-local information
  - Not really compatible with neuromorphic/physical implementations

Weight transport

- Forward pass: $h_l = \sigma(W_l h_{l-1} + b_l), \ \forall\, l = 2, \dots, L$
- Backward pass: $\delta_l = (W_{l+1}^T \delta_{l+1}) \circ \sigma'(a) \quad \forall\, l = 1, \dots, L-1$
- Two separate computational circuits with **symmetric information** (same units then!)
- Transport of weight information across forward/backward circuits
- Backward circuit does not impact on neural activations in the forward circuit (implausible)

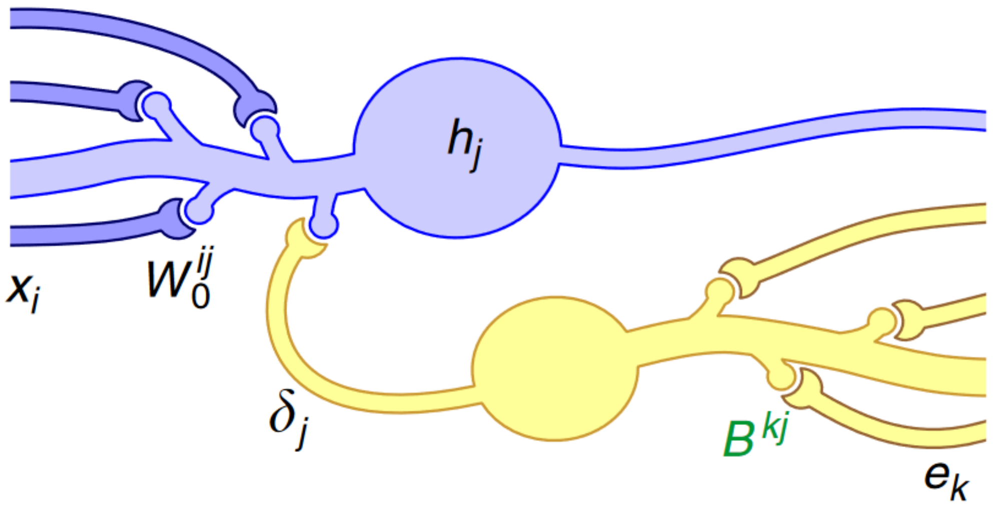
Grossberg, S. Competitive learning: from interactive activation to adaptive resonance. Cogn. Sci. 11, 23–63 (1987).



Removing weight alignment

Feedback Alignment

- Random feedback weights B
- Decouples the forward and the backward passes
- FA pushes the weights in a similar direction wrt backpropagation (how, why, uuh???)
- Still separate circuits (although more easily implementable)



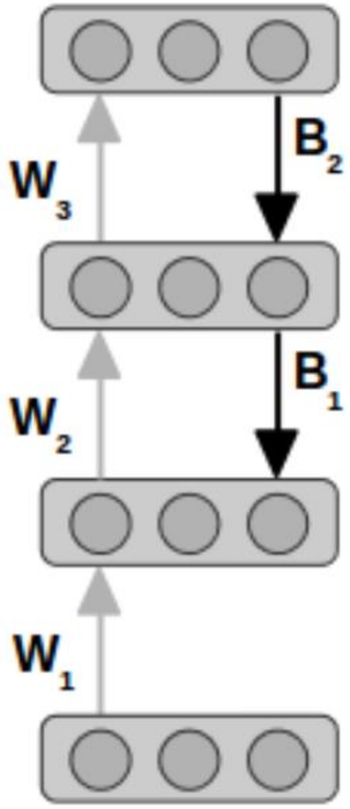
Learning with Feedback Alignment

The error is computed at the output layer and then «backpropagated» via the B_l matrices to lower layers

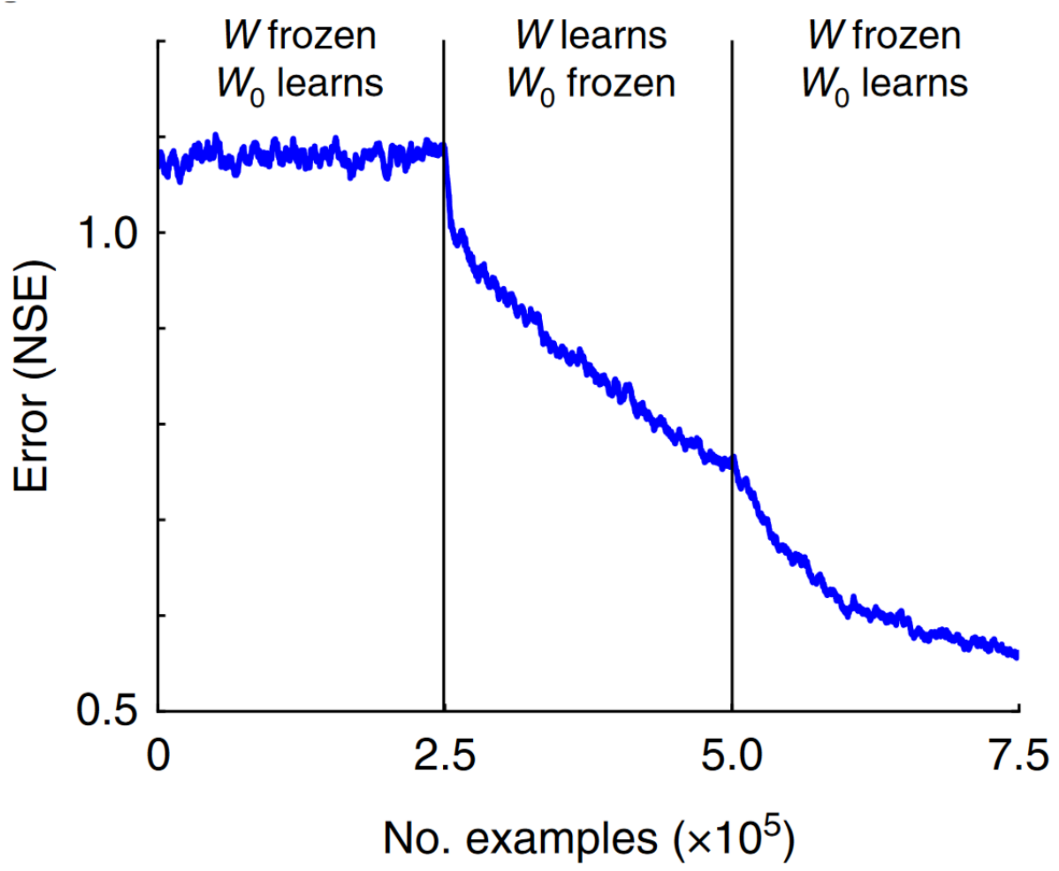
- Output layer → $\delta_3 = e$
- Hidden layer → $\delta_l = \boldsymbol{B_l}\delta_{l+1} \circ \sigma'(a_l)$
- W_l aligns with B_l^T

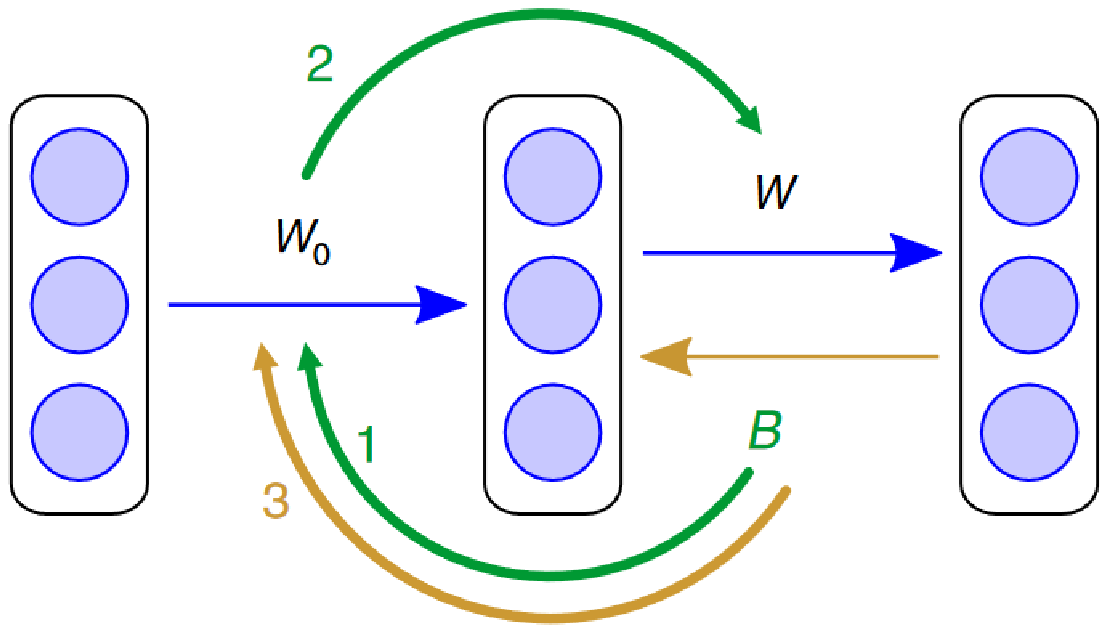
🤯

- Limited theoretical results
  - FA makes training error → 0
  - very restrictive conditions (linear networks, zero-init…)
  - In general: FA update is not the gradient of any function → cannot guarantee to follow any minimization path



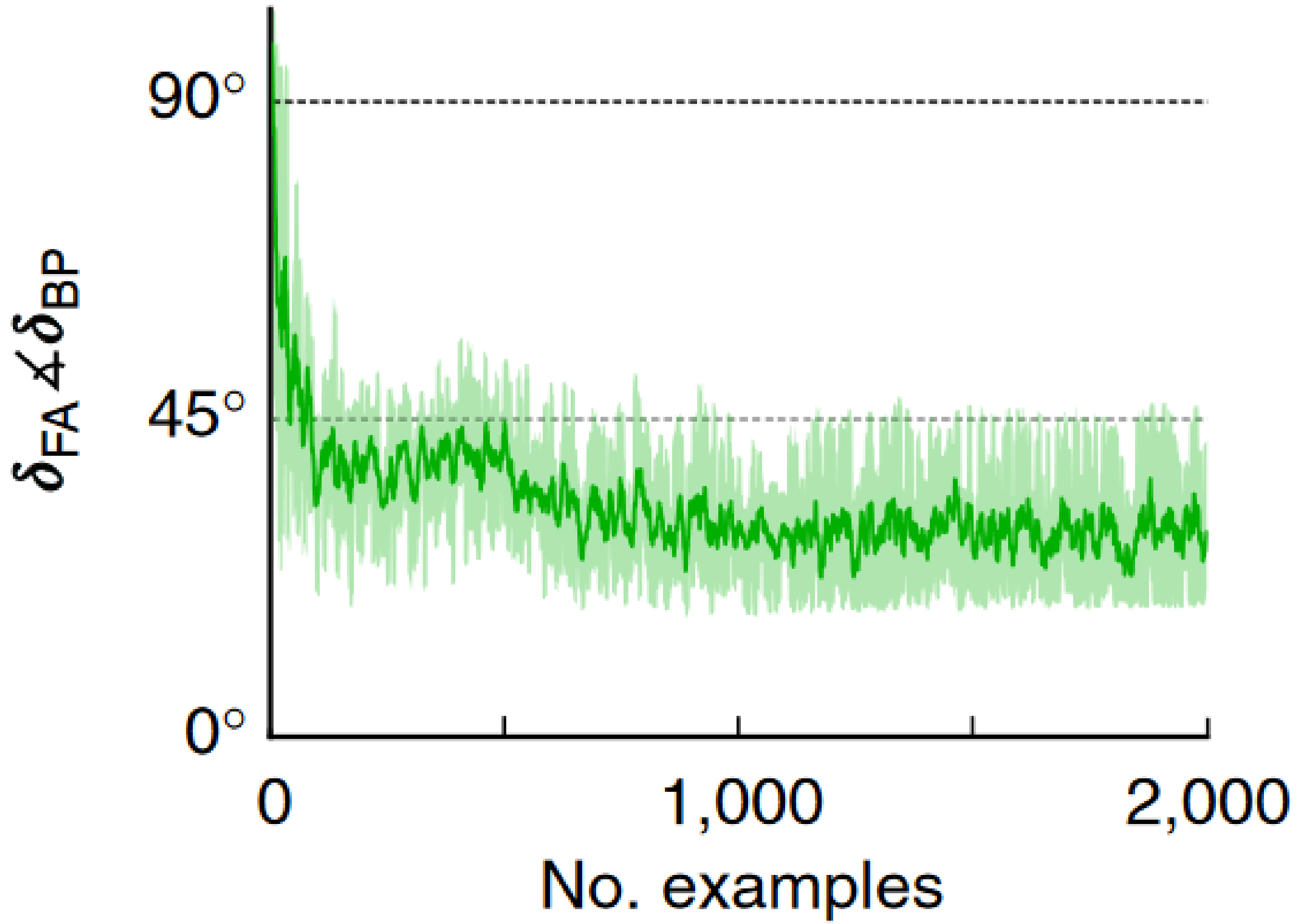
Backward weights influence forward weights





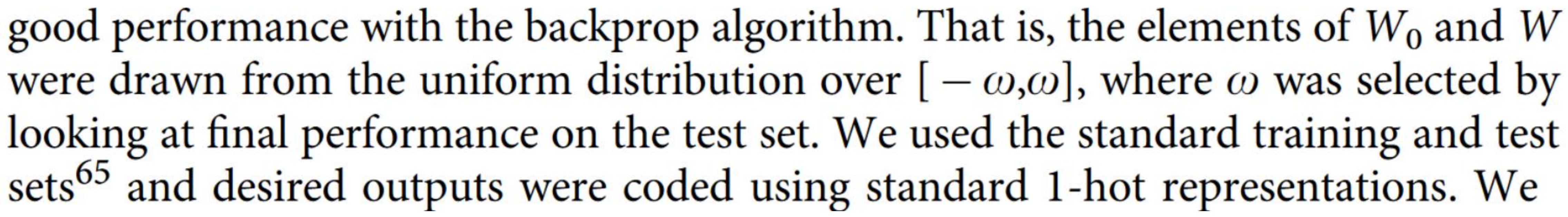
$$\Delta W = eh^T = e(W_0 x)^T = ex^T W_0^T$$

$$\Delta W_0 \propto Bex^T$$



FA update converges to BP update

Please, don’t


good performance with the backprop algorithm. That is, the elements of W_0 and W were drawn from the uniform distribution over $[-\omega,\omega]$, where ω was selected by looking at final performance on the test set. We used the standard training and test sets[65] and desired outputs were coded using standard 1-hot representations. We

(In)Direct Feedback Alignment

Disconnected feedback paths

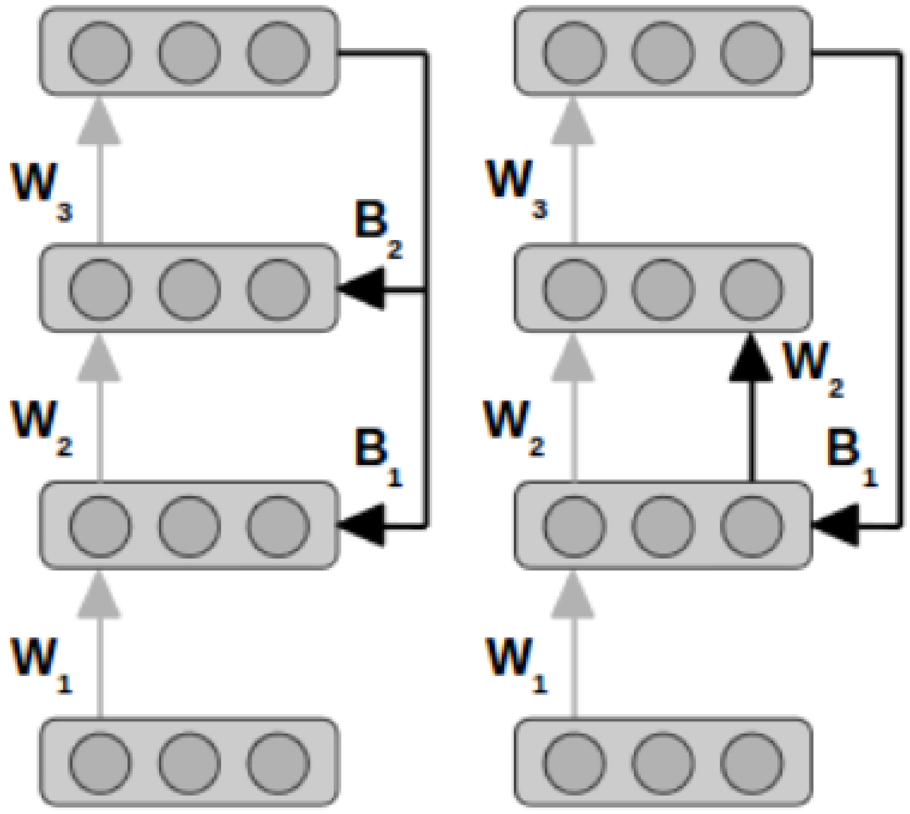
No sequential backpropagation of errors, random propagation through B, instead

Direct Feedback Alignment

- $\delta_l = (B_l e) \circ \sigma'(a_l)$
- B can even be the same for all layers

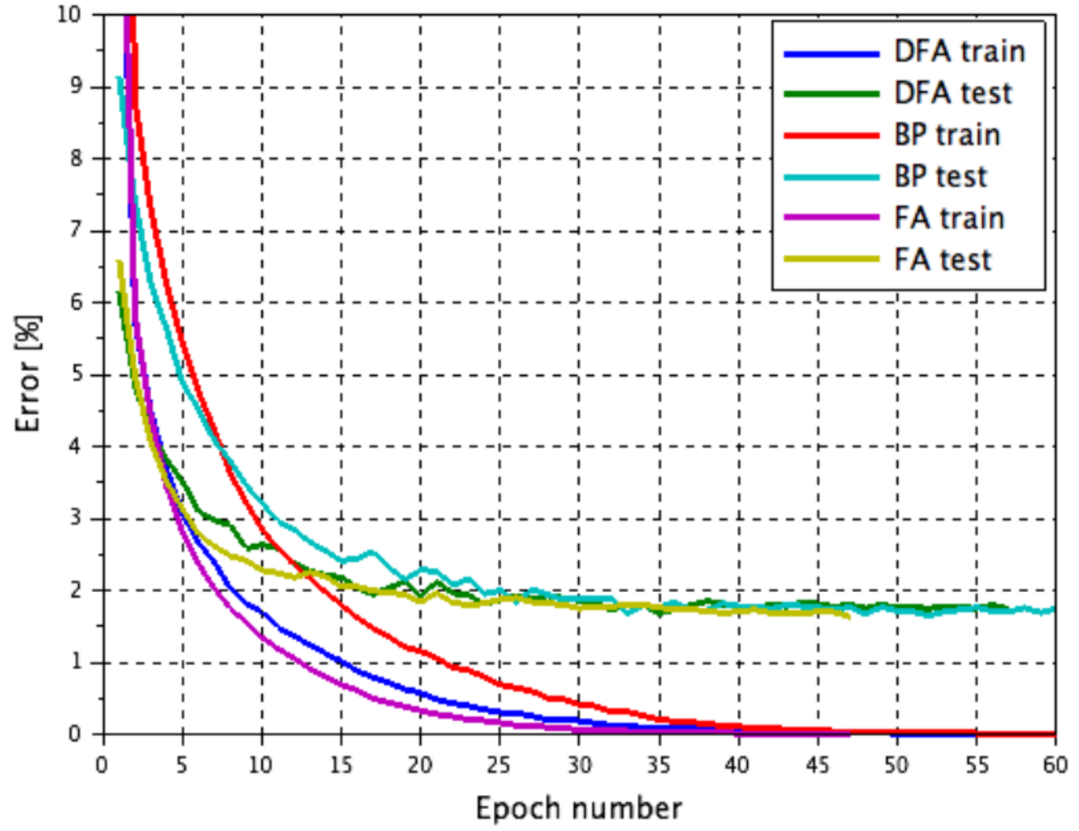
Indirect Feedback Alignment

- $\delta_1 = (B_1 e) \circ \sigma'(a_1)$
- $\delta_l = (W_l \delta_{l-1}) \circ \sigma'(a_l)$
- Feedback goes to the first layer and the proceeds forward



Deep learning with DFA

MNIST



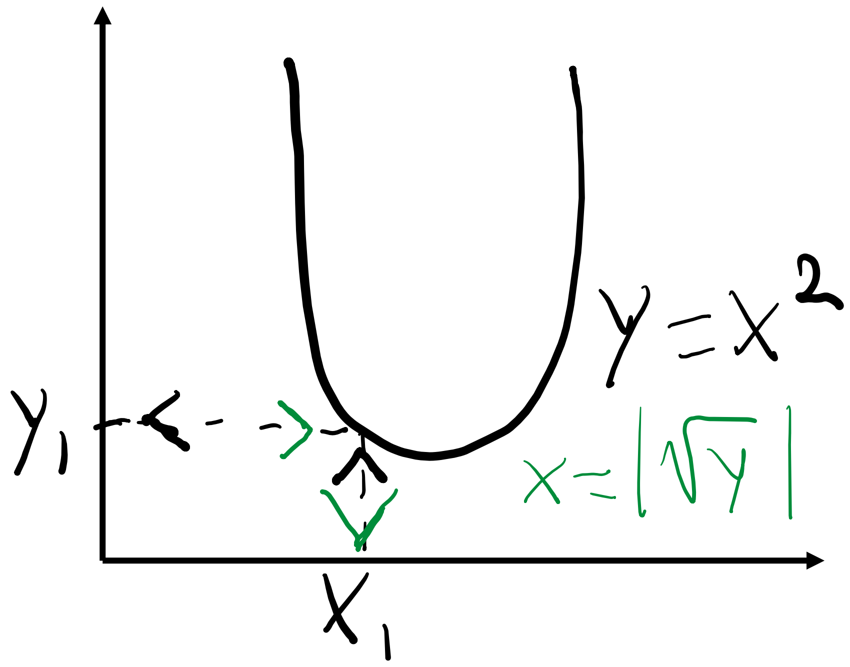
- MLPs on MNIST and CIFAR10/100 → competitive with BP
- DFA learns with very deep networks (100 layers)
  - FA/IFA don't
- BP is clearly better than alternatives with CNNs

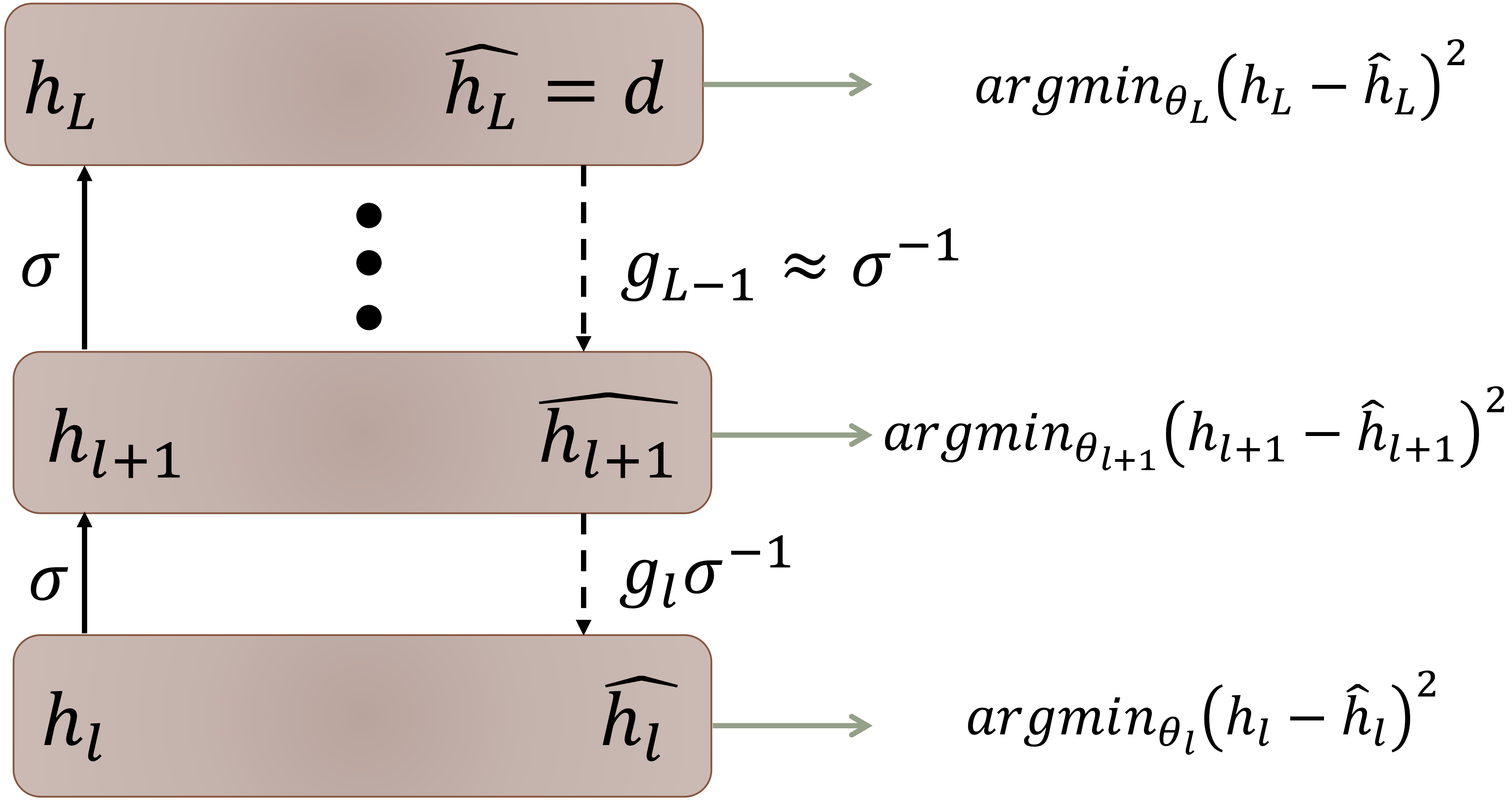
Target Propagation

- No symmetric connections **and a single computational circuit** (both forward and backward)
- Layer-wise target → **local** update
  - *what layer activation would have minimized the loss*
- Last layer (easy) → correct activation = target = $\hat{h}_L$
- Propagation to previous hidden layers: $\hat{h}_l = \sigma^{-1}(W_{l+1}\hat{h}_{l+1} + b_{l+1})$
  - Straightforward when σ^{-1} is known

Invertible neural networks

- Linear layer: $y = Wx + b \rightarrow \mathrm{x} = \mathrm{W}^{-1}(y - b)$
  - What if W not invertible?
- Even more difficult for nonlinear networks
  - Train a layer-wise decoder g_l
  - $g_l = \sigma(V_{l+1}h_{l+1} + c_{l+1}) \approx h_l = \sigma^{-1}(W_{l+1}h_{l+1} + b_{l+1})$
  - many variants…
- The decoder is used to create layer-wise targets
  - $\hat{h}_l = \sigma\left(V_{l+1}\,\hat{h}_{l+1} + c_{l+1}\right) = g_l(\hat{h}_{l+1})$




h_L
$\widehat{h_L} = d$
$argmin_{\theta_L}(h_L - \hat{h}_L)^2$
σ
$g_{L-1} \approx \sigma^{-1}$
h_{l+1}
$\widehat{h_{l+1}}$
$argmin_{\theta_{l+1}}(h_{l+1} - \hat{h}_{l+1})^2$
σ
$g_l \sigma^{-1}$
h_l
$\widehat{h_l}$
$argmin_{\theta_l}(h_l - \hat{h}_l)^2$

Training loss for Target Propagation

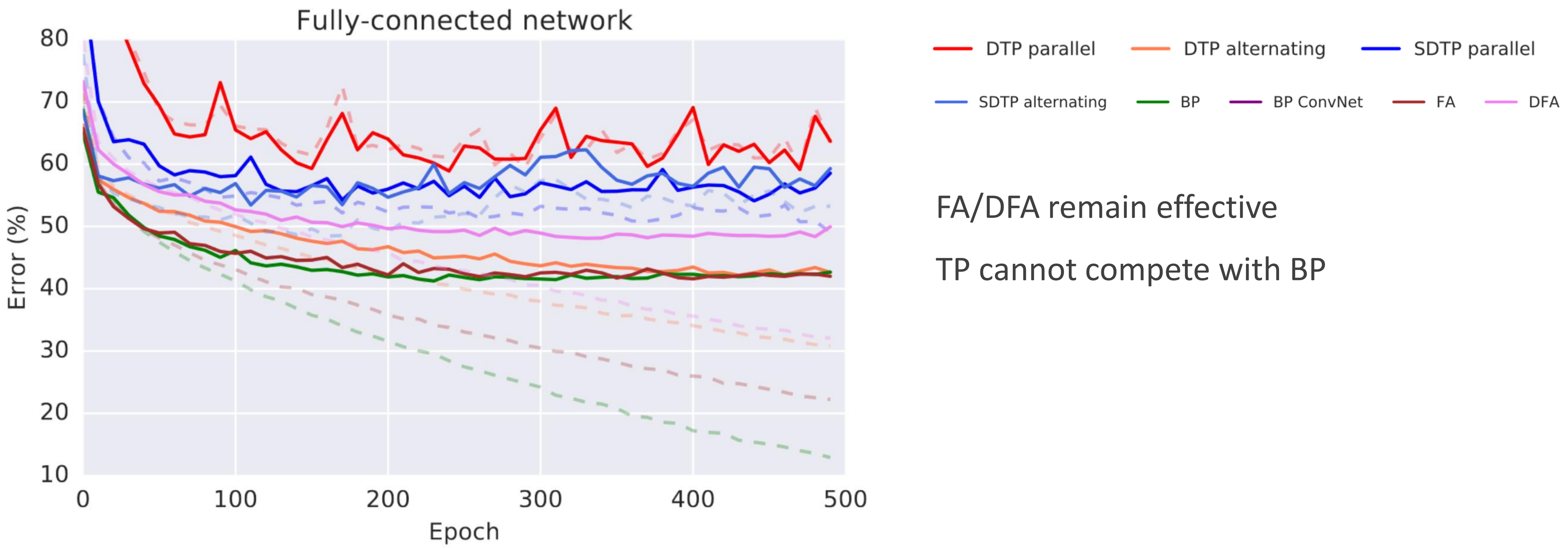
- Reconstruction loss to train the decoder
  - $L^g = ||h_{l-1} - \sigma(V_l \sigma(W_l h_{l-1} + b_l) + c_l)||_2^2$
  - The decoder is trained to reconstruct the forward activations
  - It then generalizes to the target activations (same function – the inverse – different activations)
- Forward loss to train the layers
  - $L^h = ||\sigma(W_{l+1} h_l + b_{l+1}) - \hat{h}_{l+1}||_2^2$
- Less effective in practice (difficult to scale to deep architectures → underperforms on popular benchmarks)

Alternative ways to compute targets

Recall: output layer (easy) → true target $d = \hat{h}_L$
Hidden layers: $\hat{h}_l = \sigma(V_{l+1}\, \hat{h}_{l+1} + c_{l+1})$

- **Difference Target Propagation:** $\hat{h}_l = \sigma\left(V_{l+1}\, \hat{h}_{l+1} + c_{l+1}\right) + (h_l - \sigma(V_{l+1} h_{l+1} + c_{l+1}))$
  - Consider the reconstruction error (in case g computes an imprecise inverse function)
  - Penultimate layer still trained with BP
- **Simplified DTP:** $\hat{h}_{L-1} = \sigma\left(V_L \hat{h}_L + c_L\right) + (h_{L-1} - \sigma(V_L h_L + c_L))$
  - Applies DTP also to the penultimate layer

Not easy to scale – CIFAR10



FA/DFA remain effective

TP cannot compete with BP

ImageNet – even worse

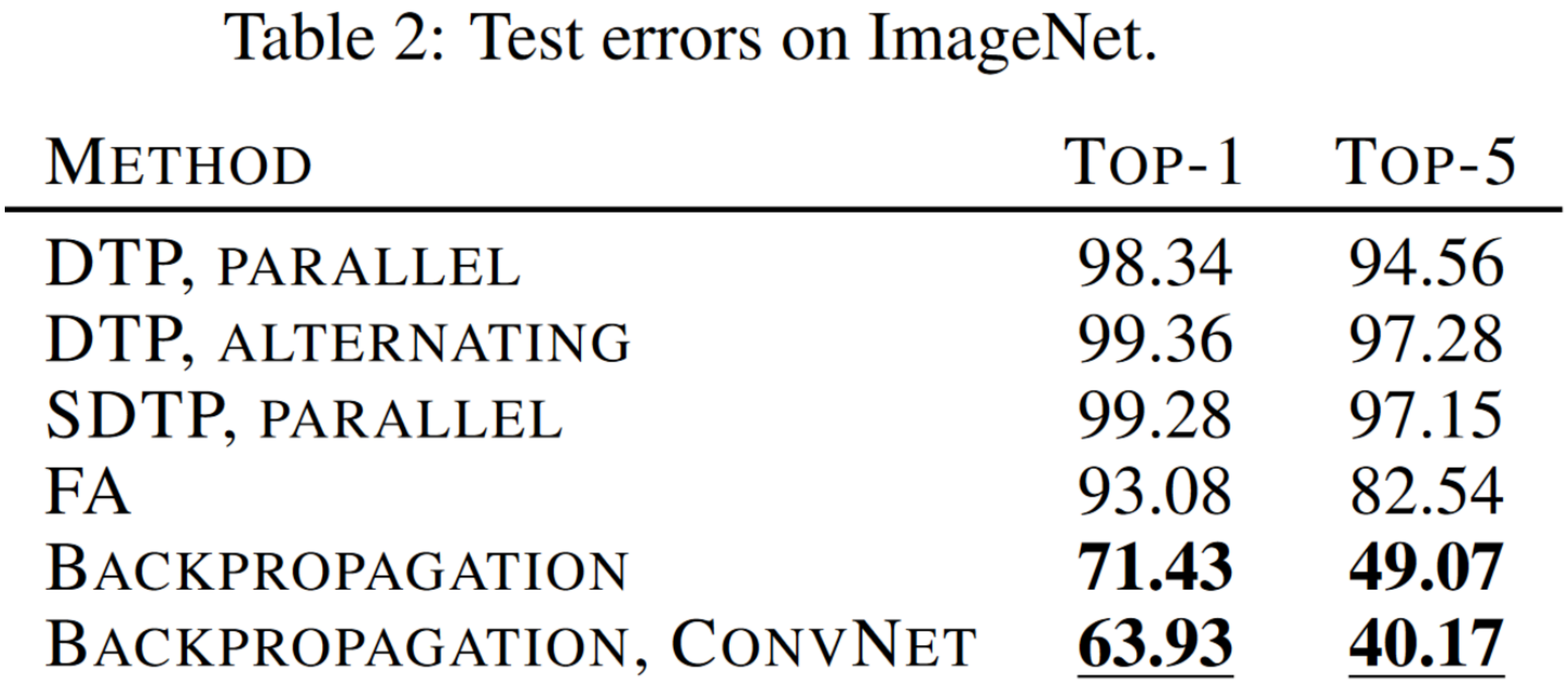


Table 2: Test errors on ImageNet.

METHOD	TOP-1	TOP-5
DTP, PARALLEL	98.34	94.56
DTP, ALTERNATING	99.36	97.28
SDTP, PARALLEL	99.28	97.15
FA	93.08	82.54
BACKPROPAGATION	**71.43**	**49.07**
BACKPROPAGATION, CONVNET	**63.93**	**40.17**

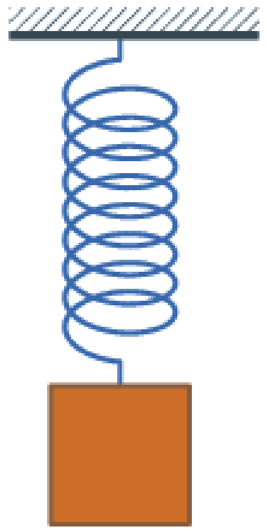
Let’s stop for a moment

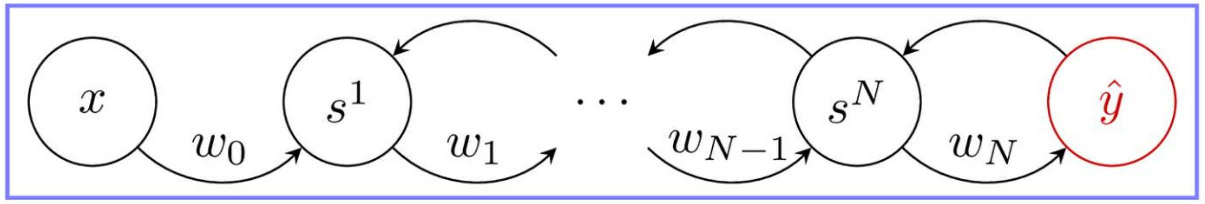
- Removing biologically unrealistic constraints is indeed possible
- Scaling to deep learning benchmarks remains challenging
- … do we need to? Depends on what you want to achieve!
  - Engineering → solving tasks, improving performance etc…
  - Scientific discovery → study a phenomen, testing assumptions, looking for feasibility etc…
  - They sometimes overlap

Equilibrium Propagation

A completely different approach

- Learning algorithm for (some) dynamical systems
  - Suitable for neuromorphic/physical implementations (e.g., spiking networks)
  - You need to know a convenient mathematical model of your system
- Intrinsically recurrent
- The system itself governs both its evolution and its adaptation
- The external input is **not** a time series





High-level summary

The system learns on a *fixed* input x.

- **First, free phase:** the state evolves following a **primitive (potential) function** $\Phi(s;\ \theta, x)$

$$\frac{\mathrm{ds}}{dt} = -\frac{\partial \Phi}{\partial s}$$

- The system evolves until it reaches a fixed point s_*
- **Second, nudging phase:** starting from s_* and under the same input x, the system now evolves according to

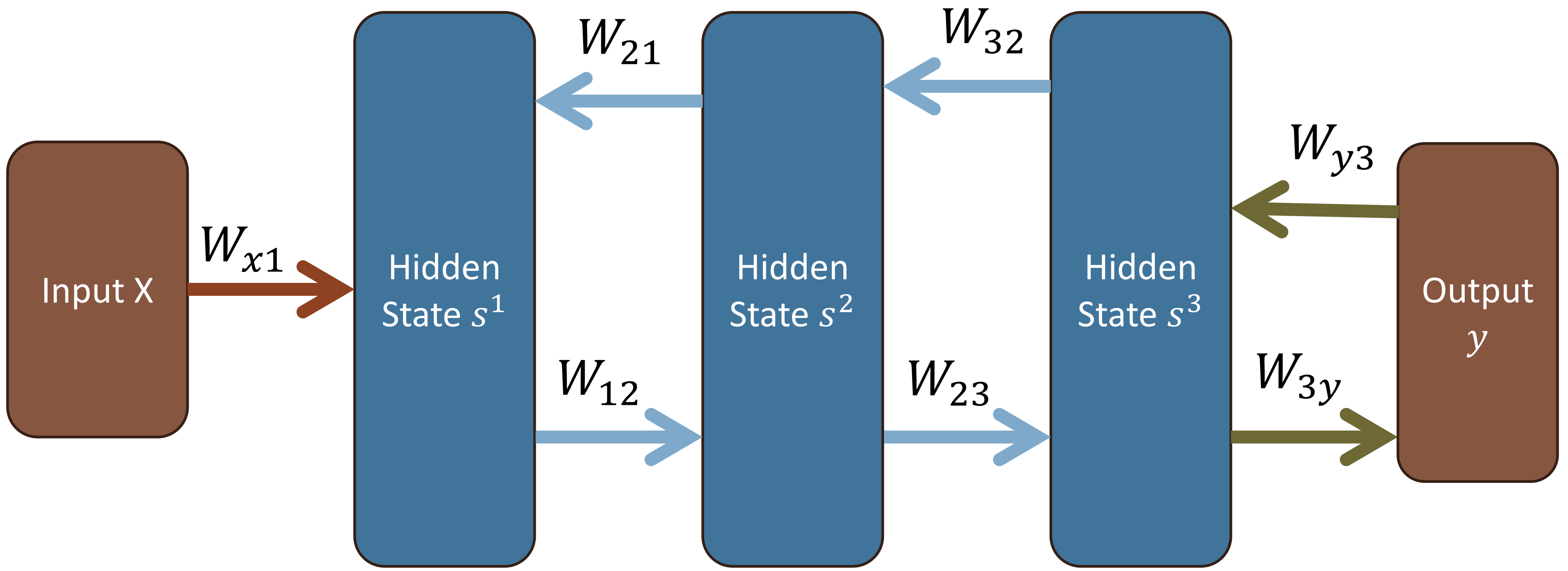
$$\frac{\mathrm{ds}}{dt} = \frac{\partial \Phi}{\partial s} - \beta \frac{dL}{ds}$$

- The evolution proceeds until the system reaches a new fixed point s_*^β
- Update the parameters θ: $\frac{\mathrm{d}\theta}{dt} = -\frac{1}{\beta}\left(\frac{\partial \Phi}{\partial \theta}\left(s_*^\beta\right) - \frac{\partial \Phi}{\partial \theta}(s_*)\right)$

Equilibrium propagation details

- The output of the system y is read-out from the state s (e.g., linear projection)
- The derivative of the loss function can be usually computed efficiently in closed form
  - And implemented physically
  - MSE loss derivative → $d - y$
- In the limit of $\beta \to 0$, the EP update is similar to the corresponding Backpropagation through time update.
- Adaptation to LIF networks for neuromorphic implementation! https://www.sciencedirect.com/science/article/pii/S2589004221001905

An example network



$s_0 = \left[s_0^1, s_0^2, s_0^3, y_0\right]$ = [0, 0, 0, 0]

The state update

$s_0 = [0, 0, 0]$

$\forall t \in [1, T]$:

$\forall l \in [1, 3]$:

$s_t^l = \sigma\left(\frac{\partial \Phi^l}{\partial s^l}(s_{t-1})\right)$; If phase 2 → add $-\beta \frac{dL}{ds}(s_t^\beta, d)$

Primitive function:

$\Phi^l = s^{l-1} W_{l-1,l} s^l$

Assume linear activation function in the primitive and put the nonlinearity manually after computing the derivative for each layer.

EP for CNNs – quite involved

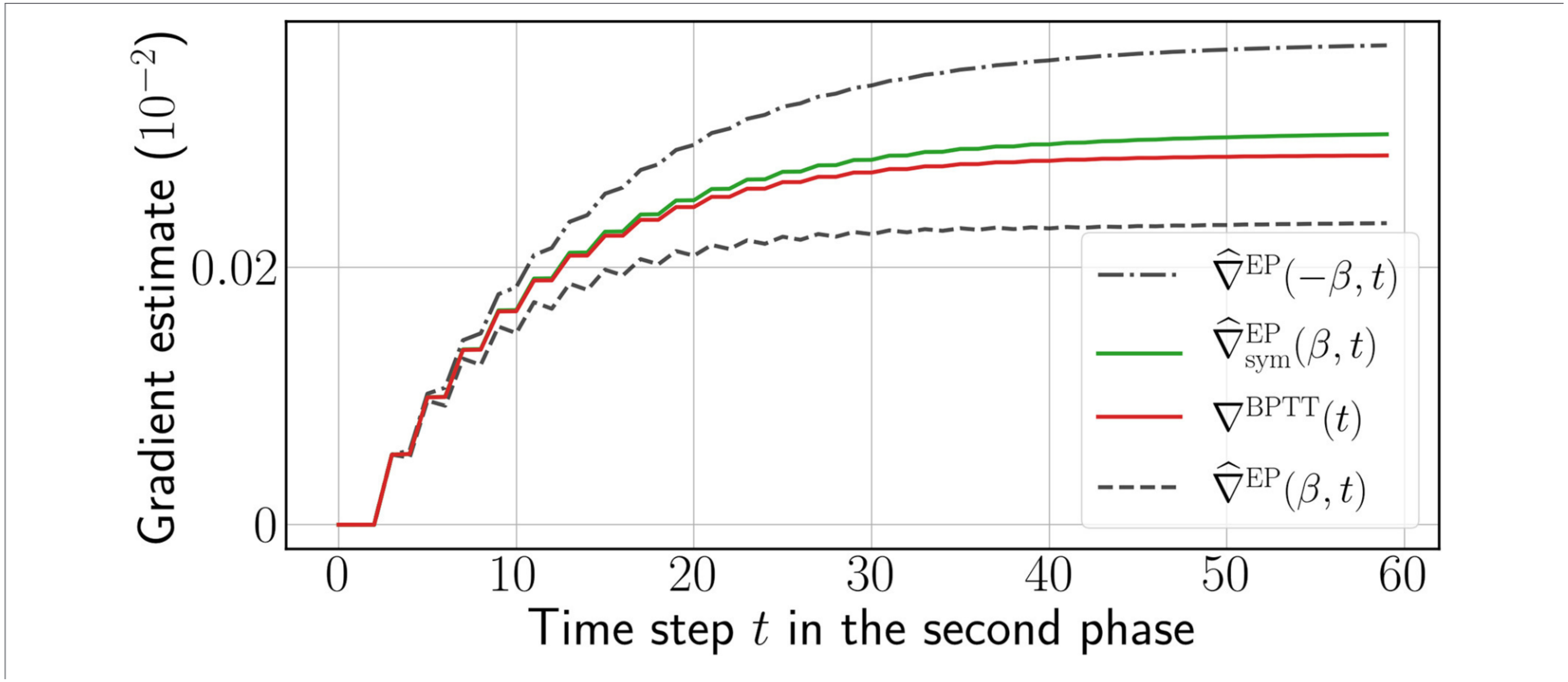
Need to consider primitive of convolution and pooling operations
(and their inverse $\tilde{w}$, flipped kernel, and $\mathcal{P}^{-1}$, unpooling)

$$\Phi(x, \{s^n\}) = \quad \sum_{n \in conv} s^{n+1} \cdot \mathcal{P}(w_{n+1} \star s^n) \\ + \sum_{n \in feed} s^{n+1\top} \cdot w_{n+1} \cdot s^n$$

$$s_{t+1}^n = \sigma\left(\mathcal{P}\left(w_n \star s_t^{n-1}\right) + \tilde{w}_{n+1} \star \mathcal{P}^{-1}\left(s_t^{n+1}\right)\right), \qquad if\ n \in conv$$

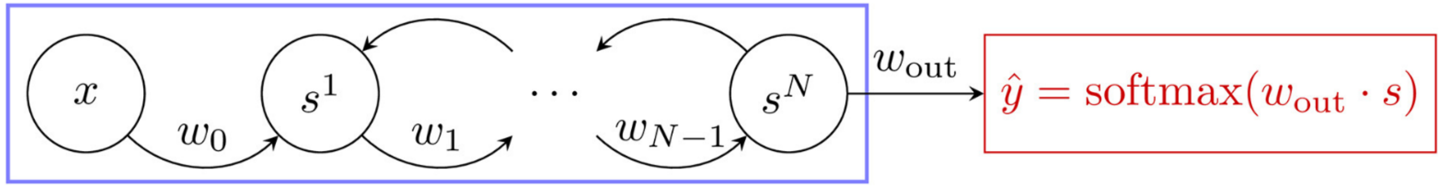
$$s_{t+1}^n = \sigma\left(w_n \cdot s_t^{n-1} + w_{n+1}^\top \cdot s_t^{n+1}\right), \quad if\ n \in feed$$

Convergence to BPTT update ($\beta = 0.1$)

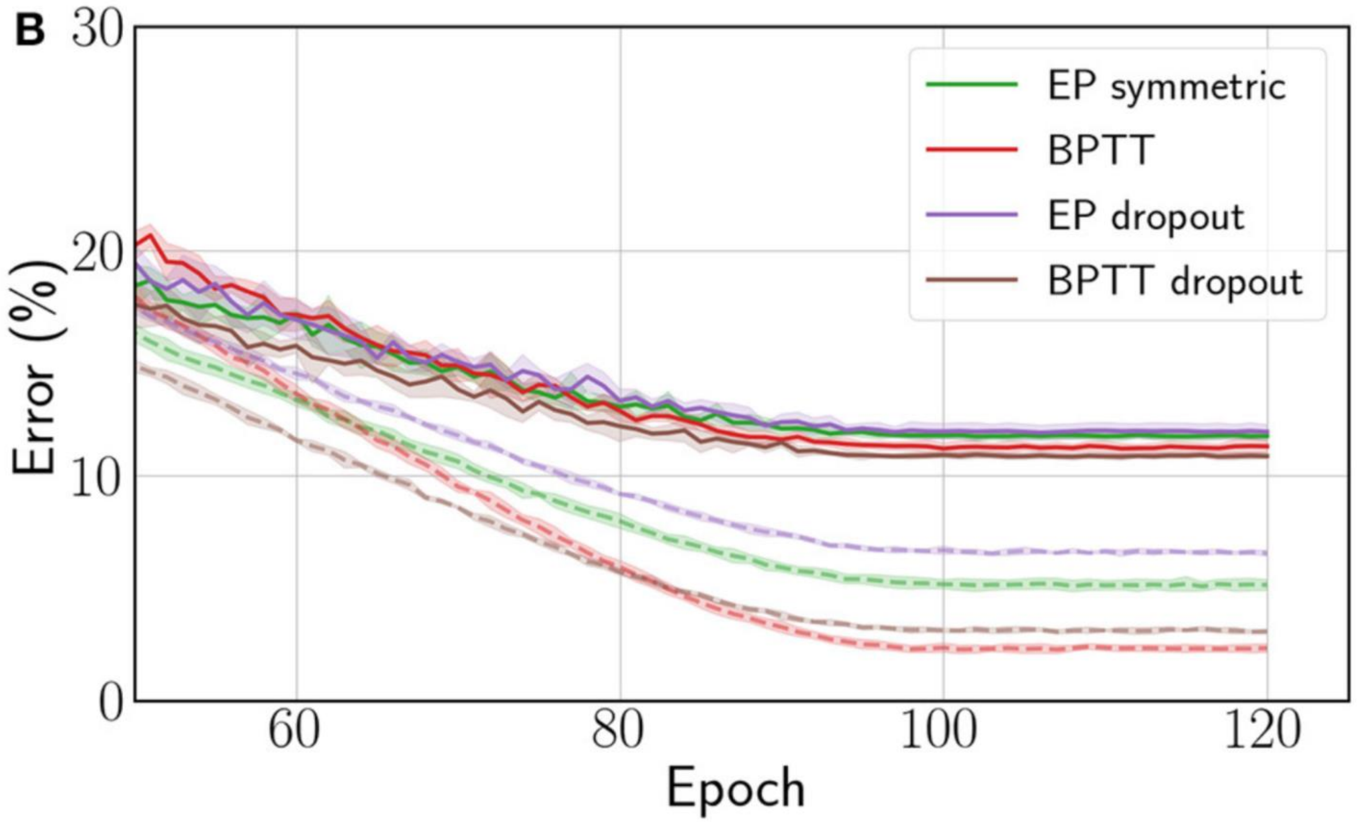


Scaling up Equilibrium Propagation

- Cross-entropy vs. MSE
  - Slightly different second phase (derivative of the loss changes)
  - Separate update rule for w_{out}: $-\left(\hat{y}_*^{\beta} - d\right) s_*^{\beta}$
- Third phase (symmetric gradient estimate)
  - First phase: $\beta = 0$, second phase: $\beta = \beta_1 > 0$ (biased), third phase: $\beta = -\beta_1 < 0$
  - Weight update: $\Delta W = \frac{1}{2\beta}\left(\frac{\partial \Phi}{\partial \theta}\left(s_*^{\beta}\right) - \frac{\partial \Phi}{\partial \theta}\left(s_*^{-\beta}\right)\right)$
- Bidirectional connections
  - Distinct parameters for forward connections and backward connections in the network
  - Easier for physical implementations



CIFAR-10 with cross-entropy



EP In Python

With a little help from automatic differentiation tools in PyTorch…

https://github.com/Laborieux-Axel/Equilibrium-Propagation/blob/93660ed6c5b0ec07978b674a69c169ce32e8cd5f/model_utils.py#L115

Summary

- BP remains the most effective learning algorithm for optimizing on a *given task*
- Feedback alignment removes the need for symmetric connections (weight transport)
- Direct feedback alignment only requires propagation of the top error
  - A single error-projecting area is biologically more sound
- Target propagation achieves local updates
  - But it requires an invertible network or a trained autoencoder
  - Not easy to imagine in biologically neural networks
- Equilibrium propagation learns from the system dynamics itself
  - Implemented physically if $\frac{\partial \Phi}{\partial \theta}$ can be computed easily

Evolution in biological life

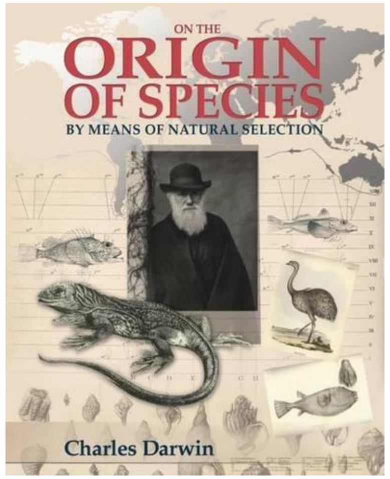
Charles Darwin's «On the Origin of Species» (1859)

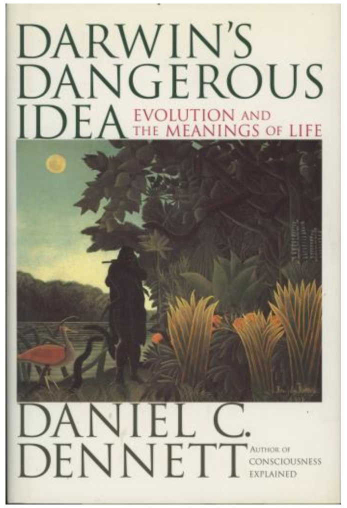
- Seminal book for evolutionary biology

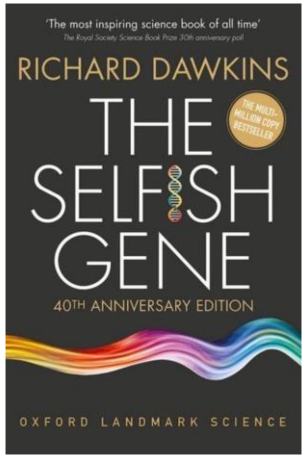
Richard Dawkins' «The Selfish Gene» (1976)

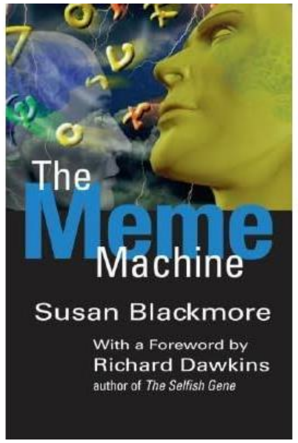
Daniel Dennett's «Darwin's Dangerous Idea» (1995)

Susan Blackmore's «The Meme Machine» (1999)




DARWIN'S
DANGEROUS
IDEA
EVOLUTION AND THE MEANINGS OF LIFE
DANIEL C.
DENNETT
AUTHOR OF CONSCIOUSNESS EXPLAINED


'The most inspiring science book of all time'
RICHARD DAWKINS
THE
SELFISH
GENE
THE MULTI-MILLION COPY BESTSELLER
40TH ANNIVERSARY EDITION
OXFORD LANDMARK SCIENCE



Evolution by natural selection

Necessary and sufficient conditions for *evolution by natural selection*

- Variation → different traits in a population
- Differential reproduction → not all individuals of the population reproduce at the same rate
  - e.g. due to some property of the environment that favors a certain trait
  - Survival of individuals that are *fit enough* (not only the fittest!)
- Heredity → traits can be passed on to the next generation

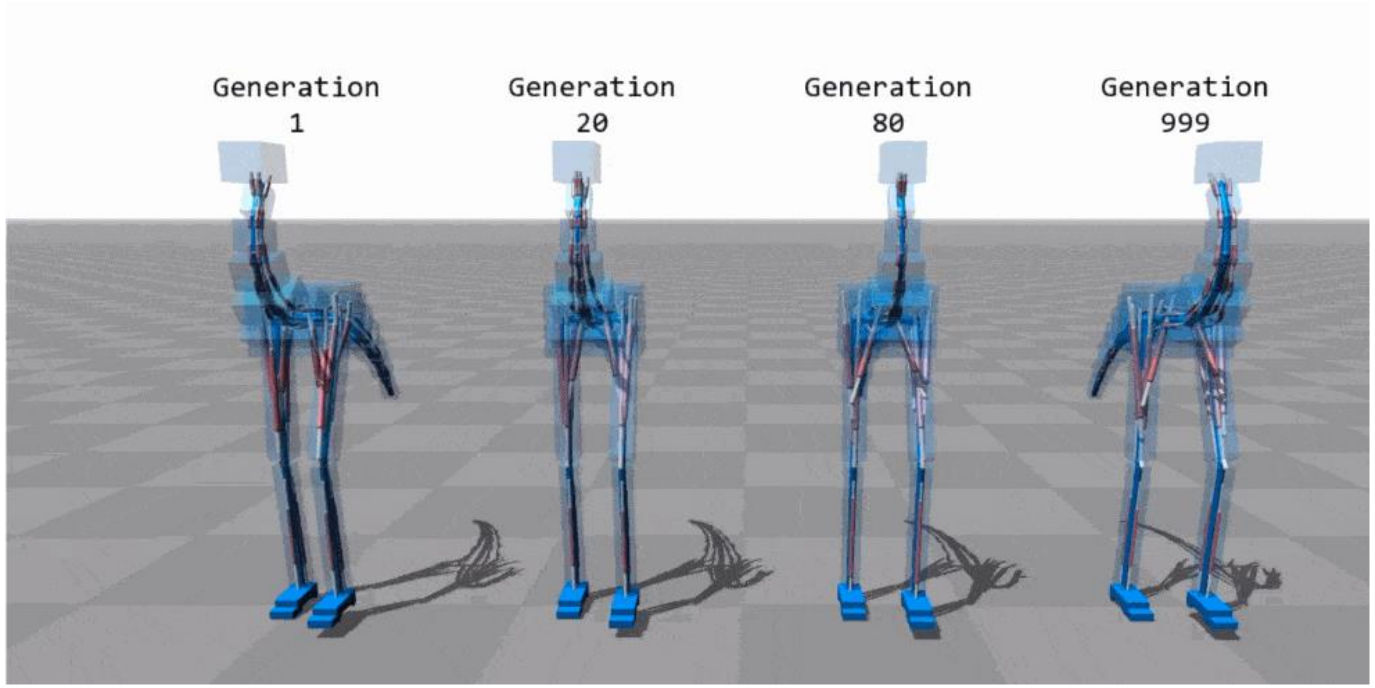
Advantageous traits survive in the population, while others disappear → convergence to a homogenous population?

Digital evolution

Life originated from an iterative process driven by evolution

- Initial conditions, environment
- (biological) Evolution algorithm

If we can simulate both we can “create” artificial life



Evolutionary algorithms

Since the 1950s [7, 8], to solve optimization problems, combinatorial problems, rule-based classifier systems…

We already start from a well-defined process: evolution by natural selection

- which may or may not be a sufficiently accurate approximation of the real process

We can simulate the necessary and sufficient conditions

- Variation
- Differential reproduction
- Heredity

Skeleton of EAs

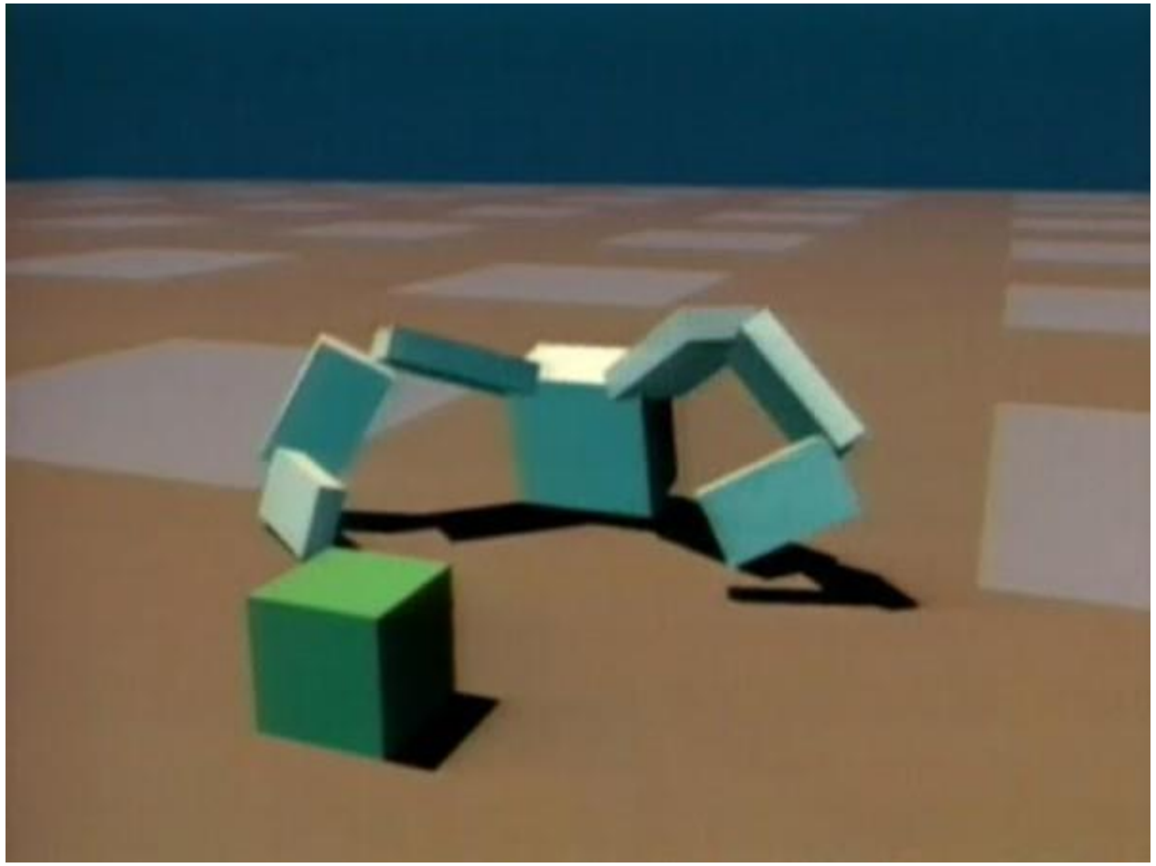
Initialize population P_0 of individuals with N features

Evaluate fitness of population P_0

For each **step** *t=1, 2, …:*

- **Select** parents from P_0 (e.g., based on the fitness)
- **Reproduction** and **recombination** of features to get children (not really present in EP)
- Apply **mutation** to features (e.g., at random, adaptive mutation rate)
- **Evaluate fitness** of new population
- Select subset of K **survived** individuals to get P_t (e.g., random sample weighted by fitness)

Do the parent survive? Do we replace K parents with K children?

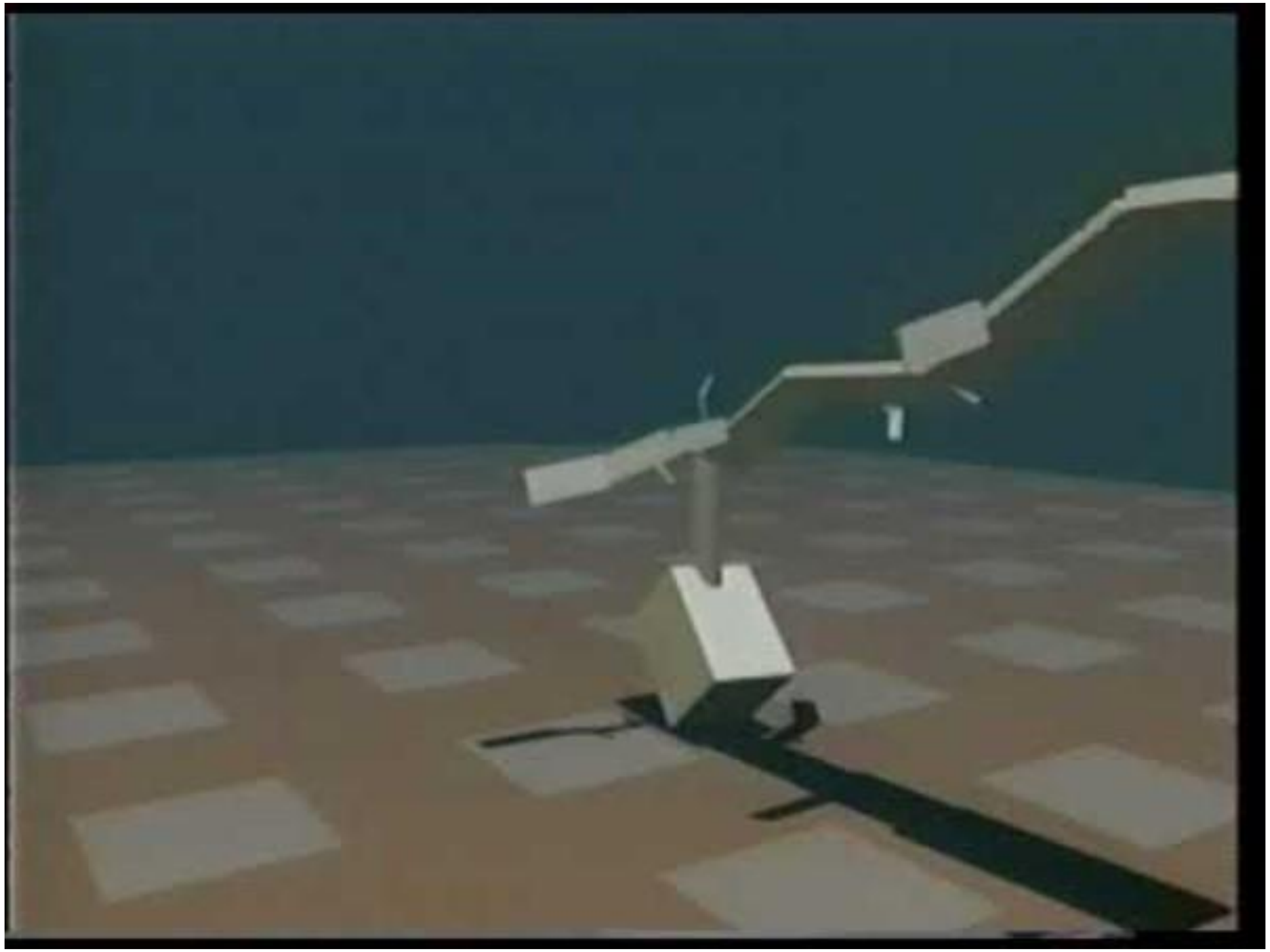


Evolving digital creatures

Creatures that can move in the space and learn to walk / swim / jump [4, 5, 6] (1993-1994)

https://www.karlsims.com/evolved-virtual-creatures.html

Occhio agli imbrogli



Different types of Evolutionary Algorithms

Same structure, different applications/low-level details [9]

- Evolutionary Programming / Strategies
  - Representations are tailored to the problem domain
  - Randomly mutate + evaluate
  - Evolutionary Strategies (randomly mutate + evaluate + *recombine)*
- Genetic Algorithms
  - Mutation is rare, recombination is the main factor (what about diversity?
  - Fitness is relevant also to select parents
- Genetic Programming
  - Genetic algorithms for code generation

NEAT

NeuroEvolution through Augmenting Topologies with genetic algorithms

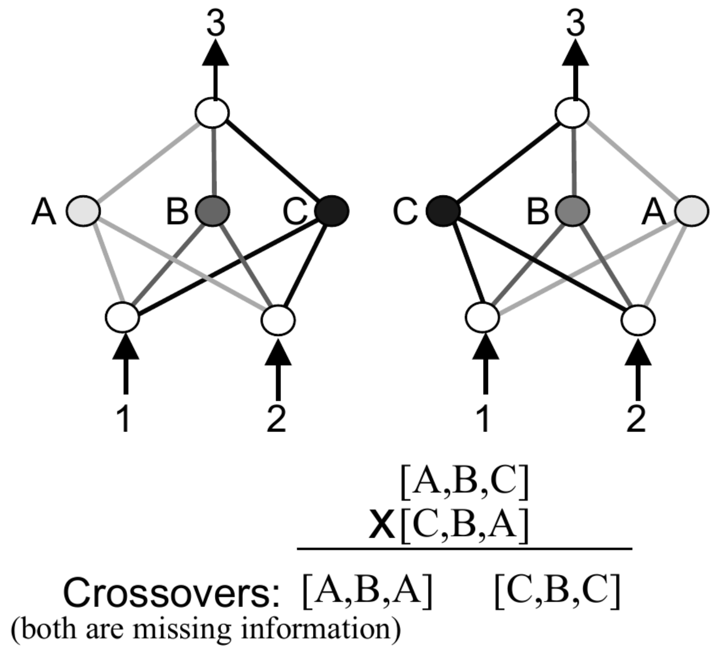
- Evolve topology + weights vs. evolving weights for a fixed topology
- Applied to RL control

Direct encoding: all node and connections in the genome

- Indirect encoding: encode rules on how to build networks
- Linear encoding → allows gene alignment for crossover
  - Also tackles permutations problem

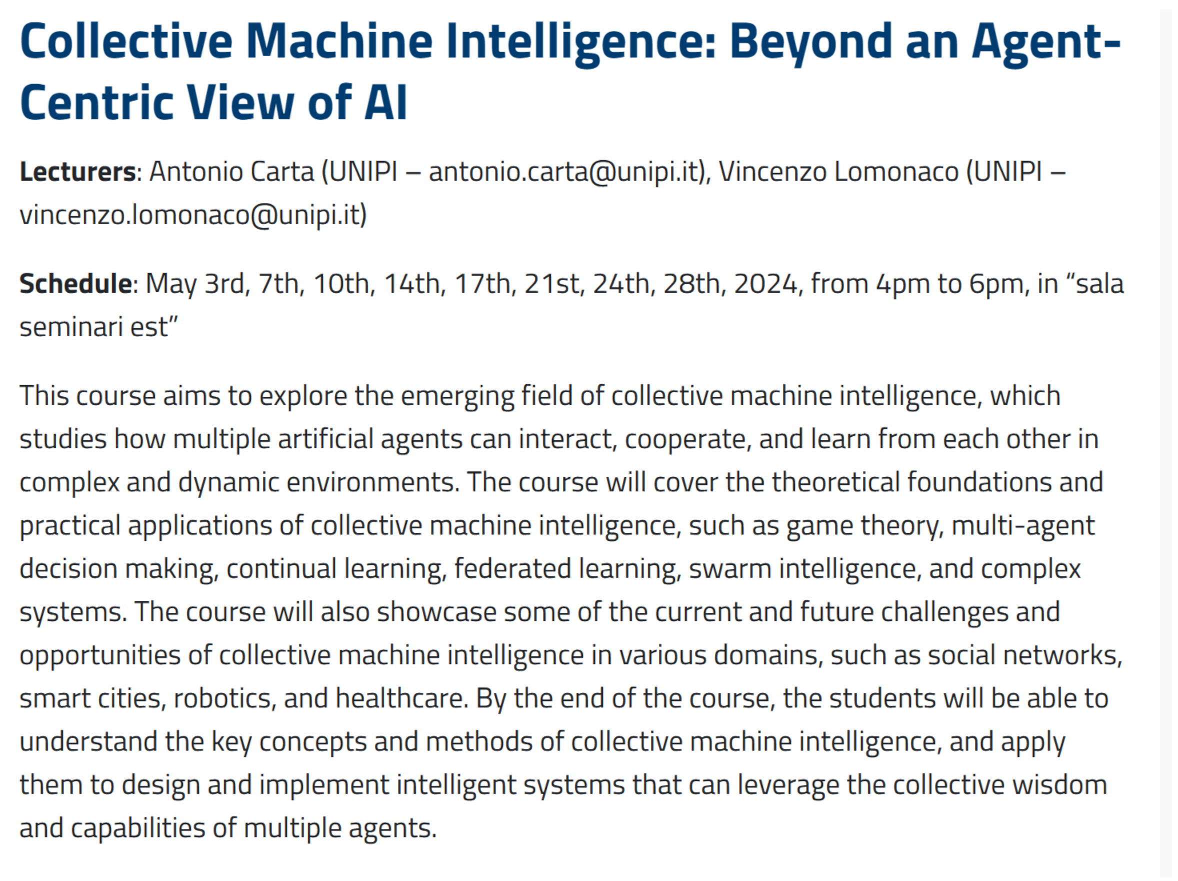
Speciation: divide population into different groups based on genetic similarity

Starting from minimal solutions



Upcoming PhD course on collective intelligence @ Computer Science dept.

https://dottorato.di.unipi.it/teaching-phd-courses-a-y-2023-2024/



Collective Machine Intelligence: Beyond an Agent-Centric View of AI

Lecturers: Antonio Carta (UNIPI – antonio.carta@unipi.it), Vincenzo Lomonaco (UNIPI – vincenzo.lomonaco@unipi.it)

Schedule: May 3rd, 7th, 10th, 14th, 17th, 21st, 24th, 28th, 2024, from 4pm to 6pm, in "sala seminari est"

This course aims to explore the emerging field of collective machine intelligence, which studies how multiple artificial agents can interact, cooperate, and learn from each other in complex and dynamic environments. The course will cover the theoretical foundations and practical applications of collective machine intelligence, such as game theory, multi-agent decision making, continual learning, federated learning, swarm intelligence, and complex systems. The course will also showcase some of the current and future challenges and opportunities of collective machine intelligence in various domains, such as social networks, smart cities, robotics, and healthcare. By the end of the course, the students will be able to understand the key concepts and methods of collective machine intelligence, and apply them to design and implement intelligent systems that can leverage the collective wisdom and capabilities of multiple agents.

References

S. Duan and J. C. Príncipe, “Training Deep Architectures Without End-to-End Backpropagation: A Survey on the Provably Optimal Methods,” *IEEE Computational Intelligence Magazine*, vol. 17, no. 4, pp. 39–51, Nov. 2022, doi: 10.1109/MCI.2022.3199624.

T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, “Random synaptic feedback weights support error backpropagation for deep learning,” *Nature Communications*, vol. 7, no. 1, Art. no. 1, Nov. 2016, doi: 10.1038/ncomms13276.

A. Nøkland, “Direct Feedback Alignment Provides Learning in Deep Neural Networks,” in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., Curran Associates, Inc., 2016, pp. 1037–1045. http://papers.nips.cc/paper/6441-direct-feedback-alignment-provides-learning-in-deep-neural-networks.pdf

S. Bartunov, A. Santoro, B. Richards, L. Marris, G. E. Hinton, and T. Lillicrap, “Assessing the Scalability of Biologically-Motivated Deep Learning Algorithms and Architectures,” in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Curran Associates, Inc., 2018, pp. 9368–9378. http://papers.nips.cc/paper/8148-assessing-the-scalability-of-biologically-motivated-deep-learning-algorithms-and-architectures.pdf **empirical evaluation on target propagation, with references to several papers that explored it first.**

A. Laborieux, M. Ernoult, B. Scellier, Y. Bengio, J. Grollier, and D. Querlioz, “Scaling Equilibrium Propagation to Deep ConvNets by Drastically Reducing Its Gradient Estimator Bias,” *Frontiers in Neuroscience*, vol. 15, 2021. https://www.frontiersin.org/articles/10.3389/fnins.2021.633674

B. Scellier and Y. Bengio, “Equilibrium Propagation: Bridging the Gap between Energy-Based Models and Backpropagation,” *Frontiers in Computational Neuroscience*, vol. 11, 2017. https://www.frontiersin.org/articles/10.3389/fncom.2017.00024

W. M. Spears, K. A. De Jong, T. Bäck, D. B. Fogel, and H. de Garis, “An overview of evolutionary computation,” in *Machine Learning: ECML-93*, P. B. Brazdil, Ed., Berlin, Heidelberg: Springer, 1993, pp. 442–459. doi: 10.1007/3-540-56602-3_163.

K. O. Stanley and R. Miikkulainen, “Evolving Neural Networks through Augmenting Topologies,” *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, Jun. 2002, doi: 10.1162/106365602320169811.