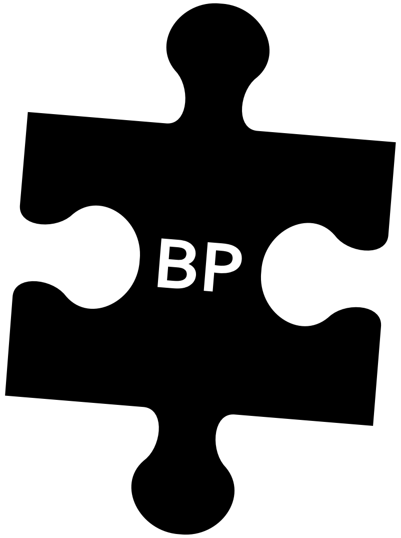
Forward Learning with Top-Down Feedback: Solving the Credit Assignment Problem without a Backward Pass

Dellaferrera & Kreiman, ICML 2022

Srinivasan, Mignacco, Sorbaro, Cooper, Refinetti, Kreiman, Dellaferrera, 2023 (arXiv:2302.05440)

Connecting the puzzle pieces of bio-inspired learning algorithms

1989



Weight symmetry!

Non local!

Activity is frozen!

Update locking!

Biologically unrealistic!

NATURE VOL. 337 12 JANUARY 1989 — COMMENTARY — 129

The recent excitement about neural networks

Francis Crick

Backpropagation and the brain

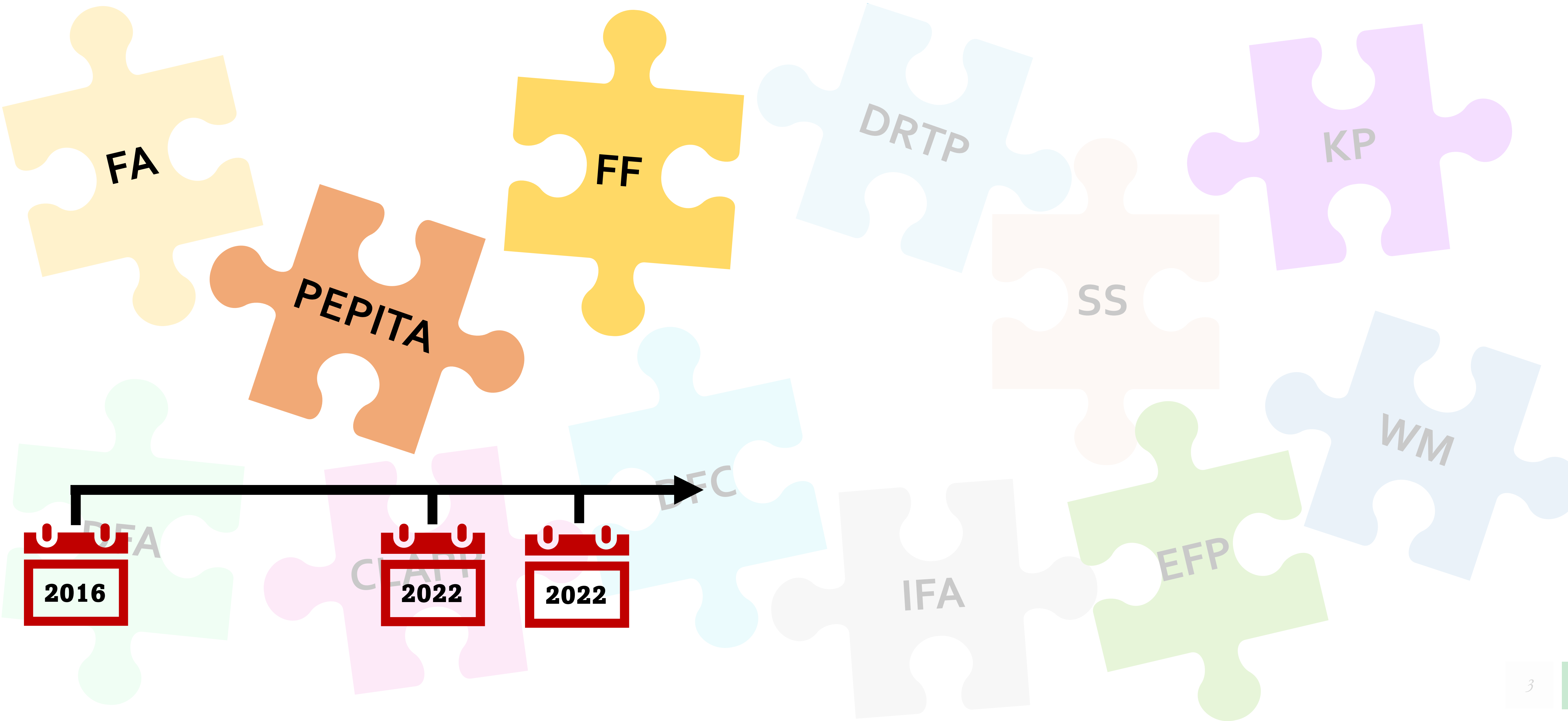
Timothy P. Lillicrap, Adam Santoro, Luke Marris, Colin J. Akerman and Geoffrey Hinton

Review

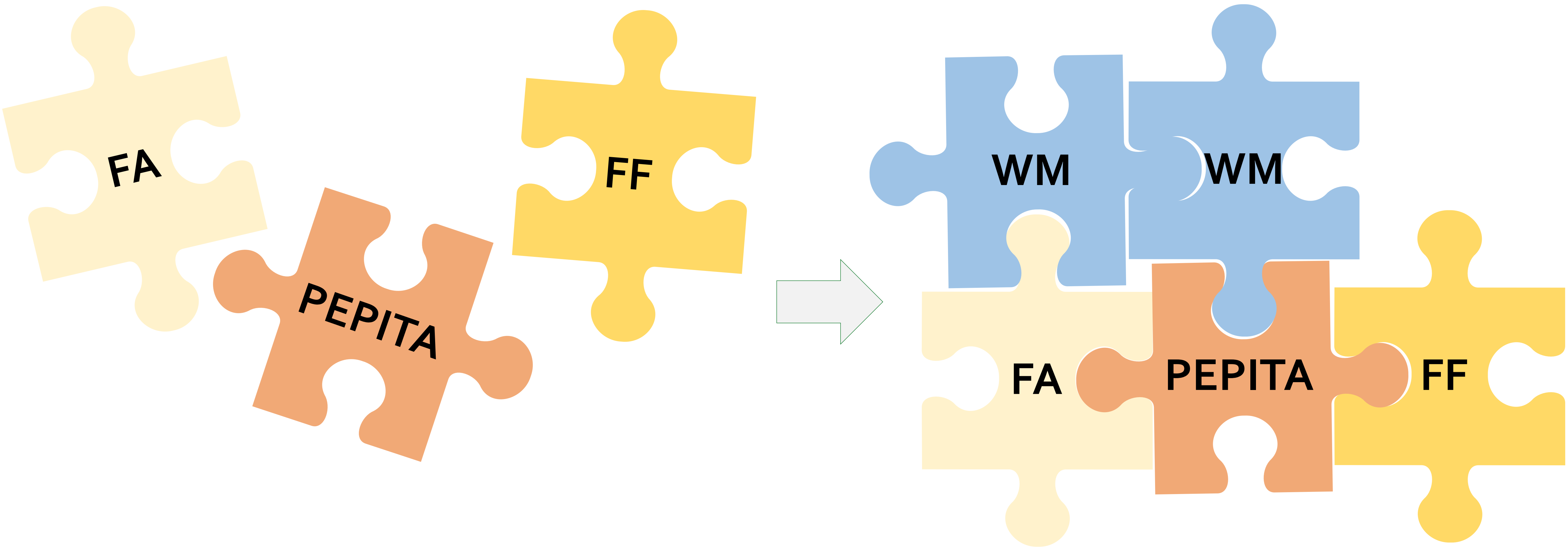
Theories of Error Back-Propagation in the Brain

James C.R. Whittington[1,2] and Rafal Bogacz[1,*]

Connecting the puzzle pieces of bio-inspired learning algorithms



Connecting the puzzle pieces of bio-inspired learning algorithms



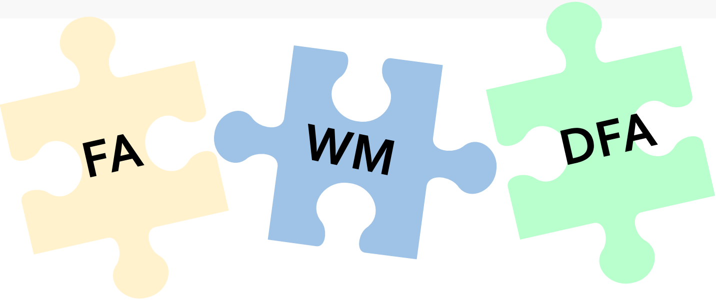
Today: 6 pm ET
12 pm EST

Today: 7 pm ET
1 pm EST

Outline

» **Neuro-inspired AI**

- Why Backpropagation is biologically implausible
- Overview of alternative solutions to credit assignment



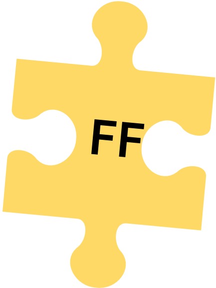
» **PEPITA: error-driven input modulation**

- Replacing the backward pass with a second forward pass
- Results on image classification tasks
- Soft alignment dynamics
- Approximating PEPITA to *Adaptive Feedback Alignment*: analytical characterization
- Improving alignment with weight mirroring

Followed by **coding tutorial**: Implementing PEPITA with Pytorch

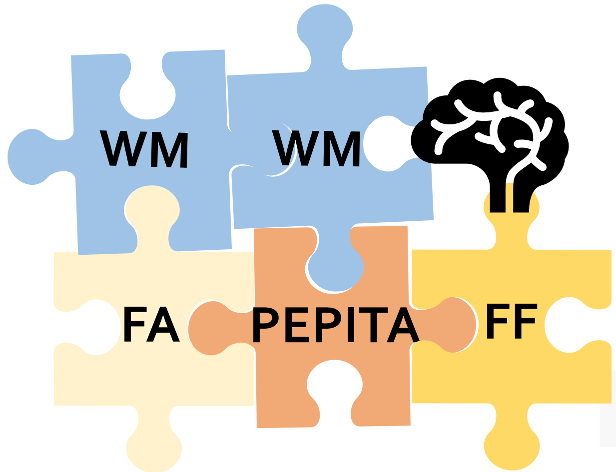
» **Forward-Forward algorithm**

- Idea and results
- Similarities with PEPITA's update rule



» **Forward learning with top-down feedback**

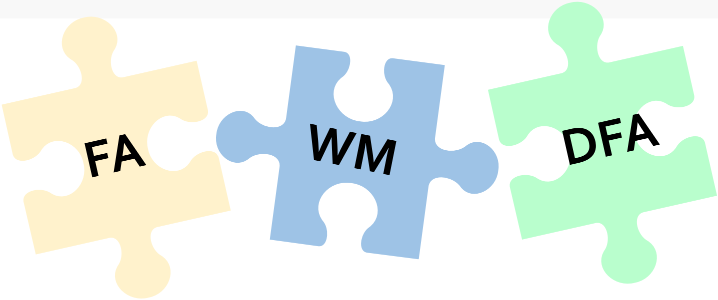
- Biological considerations



Outline

» **Neuro-inspired AI**

- Why Backpropagation is biologically implausible
- Overview of alternative solutions to credit assignment



» **PEPITA: error-driven input modulation**

- Replacing the backward pass with a second forward pass
- Results on image classification tasks
- Soft alignment dynamics
- Approximating PEPITA to *Adaptive Feedback Alignment*: analytical characterization
- Improving alignment with weight mirroring

» **Forward-Forward algorithm**

- Idea and results
- Similarities with PEPITA's update rule

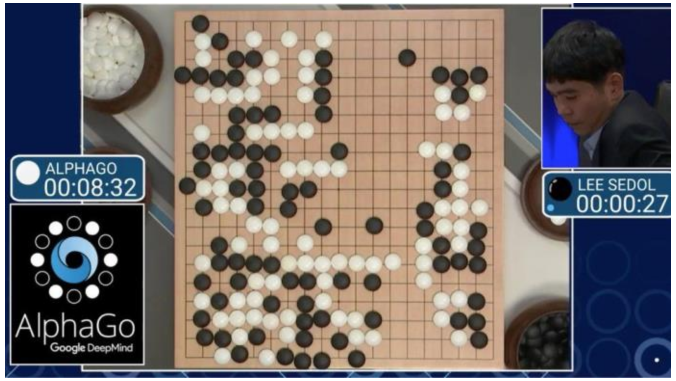
» **Forward learning with top-down feedback**

- Biological considerations

Backpropagation: successful but not biologically plausible

» **Success**
- The most effective training algorithm
- State-of-the-art performance in complex cognitive tasks

» **Algorithm**
- Chain rule of calculus
- Change in synaptic strength $\longleftrightarrow$ change of network's error



https://www.bbc.com/news/technology-35785875

Li et al., 2017

Rumelhart et al., 1989 Crick, 1989 Lillicrap et al., 2020

Glossary

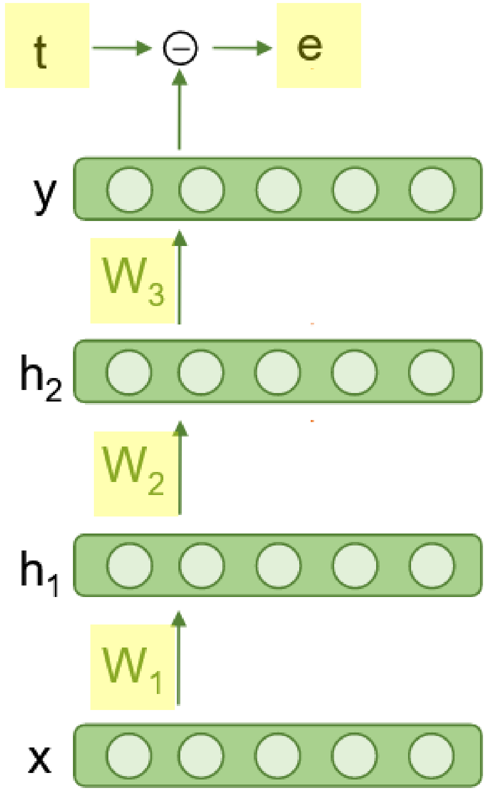
» **Credit assignment**
- Determine the degree to which a parameter (*e.g.*, synaptic weight) has contributed to the network's error

» **Target (t)**
- Desired output of a network

» **Error (e)**
- Deviation of the network's output from the target

» **Weights (W)**
- Parameter corresponding to the strength of the connection between two nodes

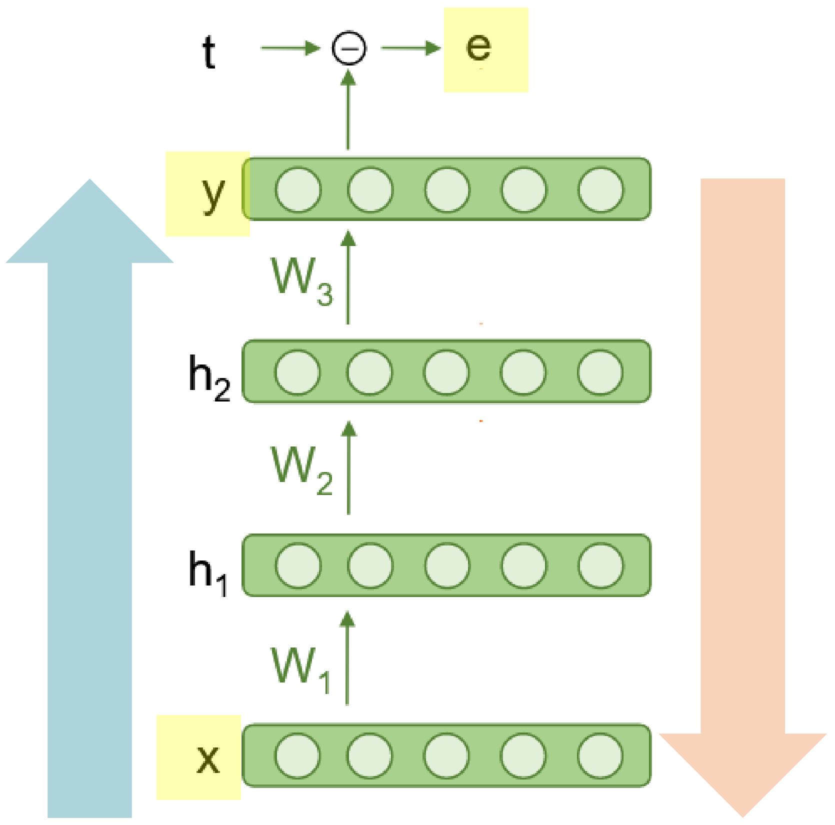
» **ANNs** = Artificial Neural Networks
FCNNs = Fully Connected Neural Networks
CNNs = Convolutional Neural Networks



The backpropagation algorithm

» **Forward pass**
- Network's response to input
- Error function $e = y - t$
- Weight updates proportional to its negative gradient

» **Backward pass**
- Error signal flows backward through the network
- Computed recursively via the chain rule
- Update phase

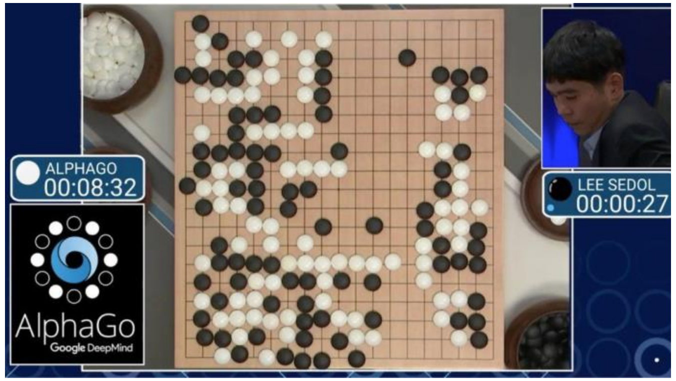


Lillicrap et al., 2020

Backpropagation: successful but not biologically plausible

» Success

- The most effective training algorithm
- State-of-the-art performance in complex cognitive tasks



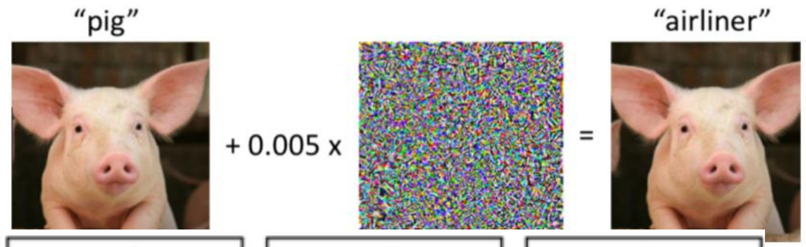
https://www.bbc.com/news/technology-35785875

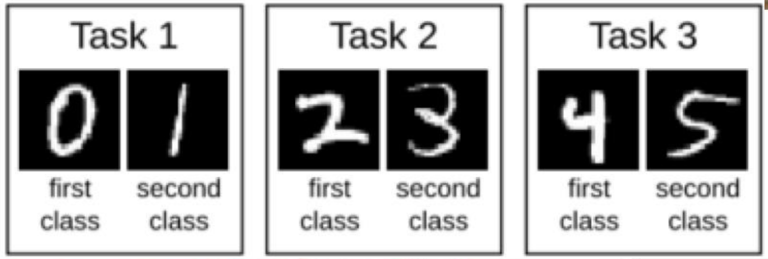
» Algorithm

- Chain rule of calculus
- Change in synaptic strength ⟷ change of network's error

» Criticisms

- Adversarial attack
- Catastrophic forgetting
- Lack of generalization
- Biological implausibility

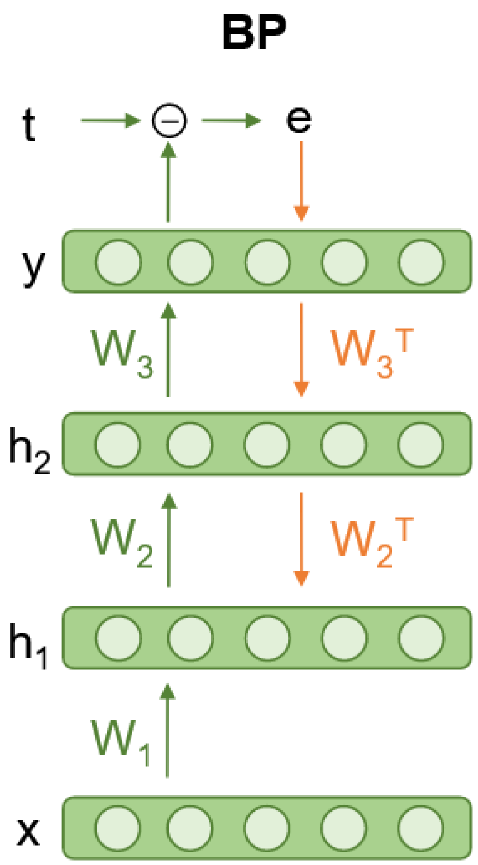




Li et al., 2017

Rumelhart et al., 1989
Crick, 1989
Lillicrap et al., 2020

Backpropagation of the error is not biologically plausible



Rumelhart et al., 1995

» **Weight transport problem**
- Symmetric weights for forward and backward computation

» **Non-local information used for the updates**
- Global error and downstream weights needed for learning

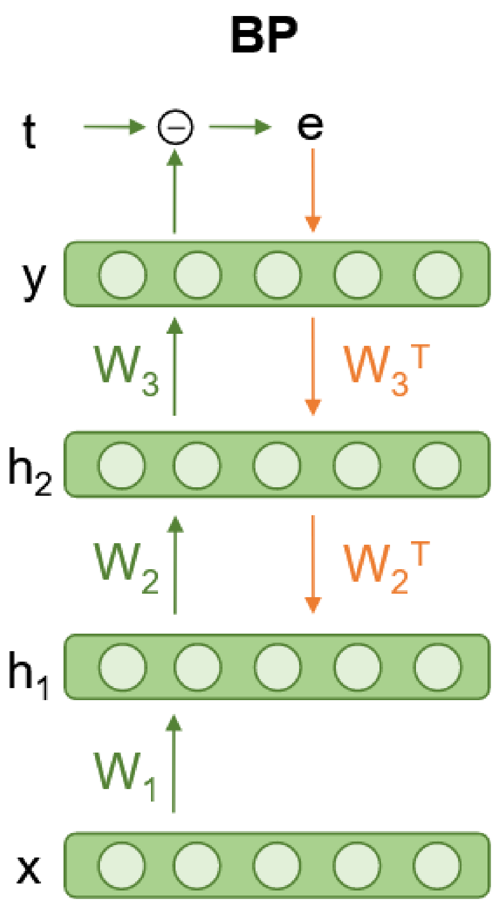
» **Frozen activity during error propagation and parameter updates**
- Separate forward and backward computation

» **Update locking problem**
- Backward computation needs to be complete before the next forward pass

Alternative Training Schemes

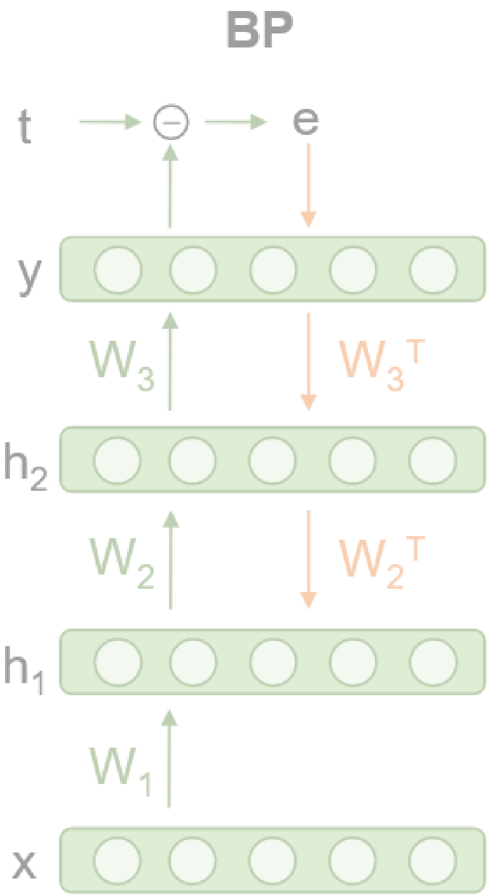
Lillicrap et al., 2020

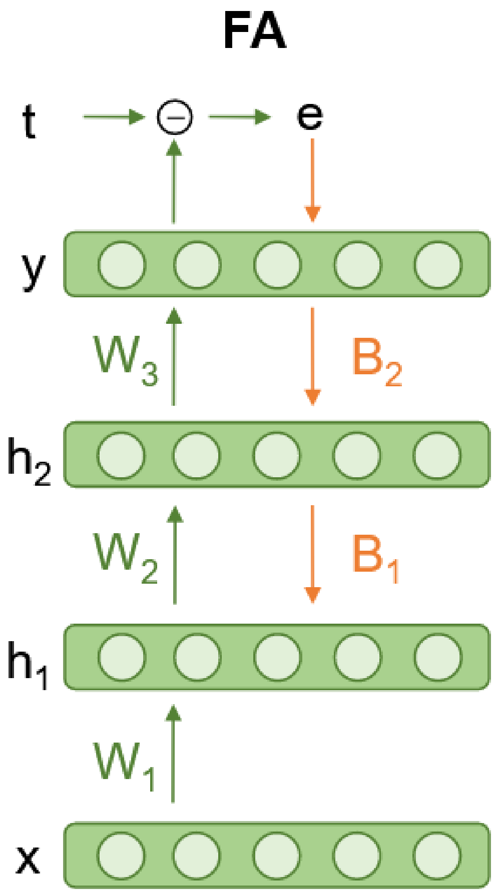
Alternatives to BP: relaxing symmetry requirements

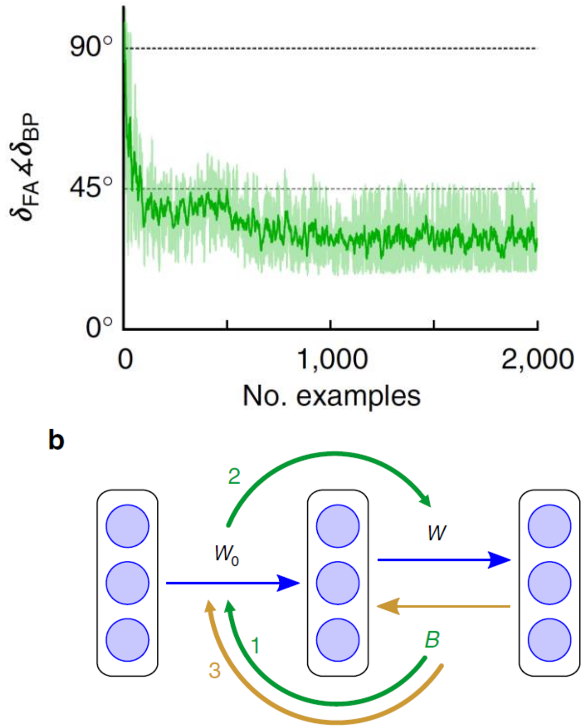


Rumelhart et al., 1995

Alternatives to BP: relaxing symmetry requirements





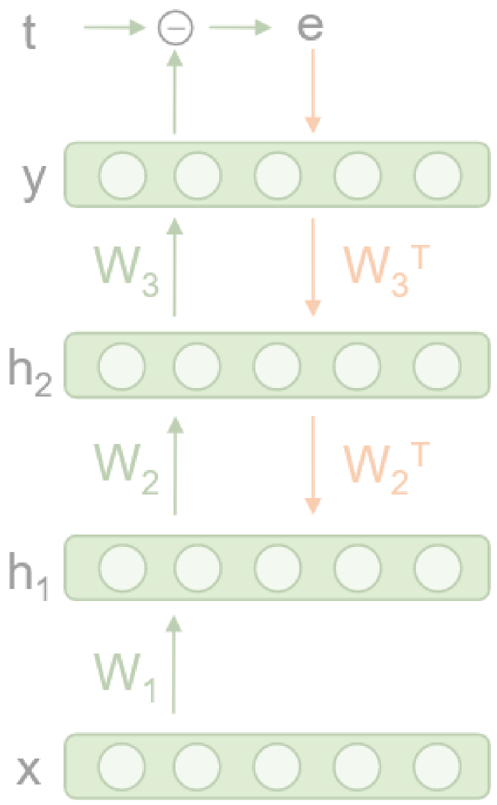


Rumelhart et al., 1995

Lillicrap et al., 2016

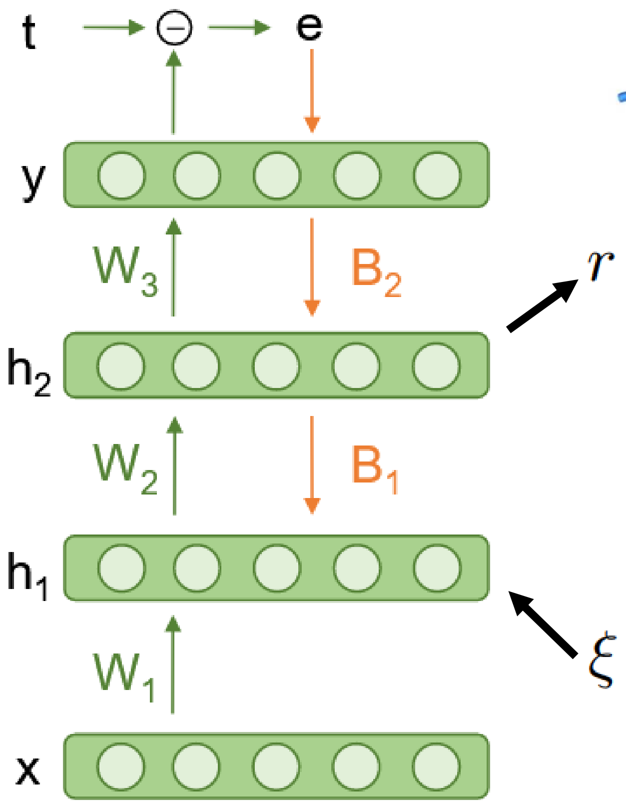
Alternatives to BP: relaxing symmetry requirements

BP

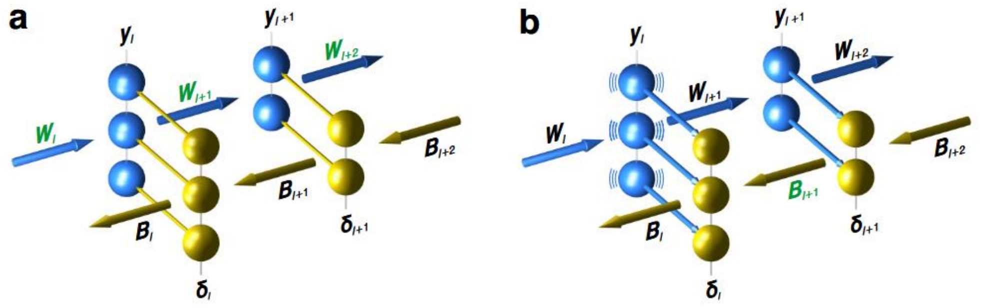


Rumelhart et al., 1995

FA +WM

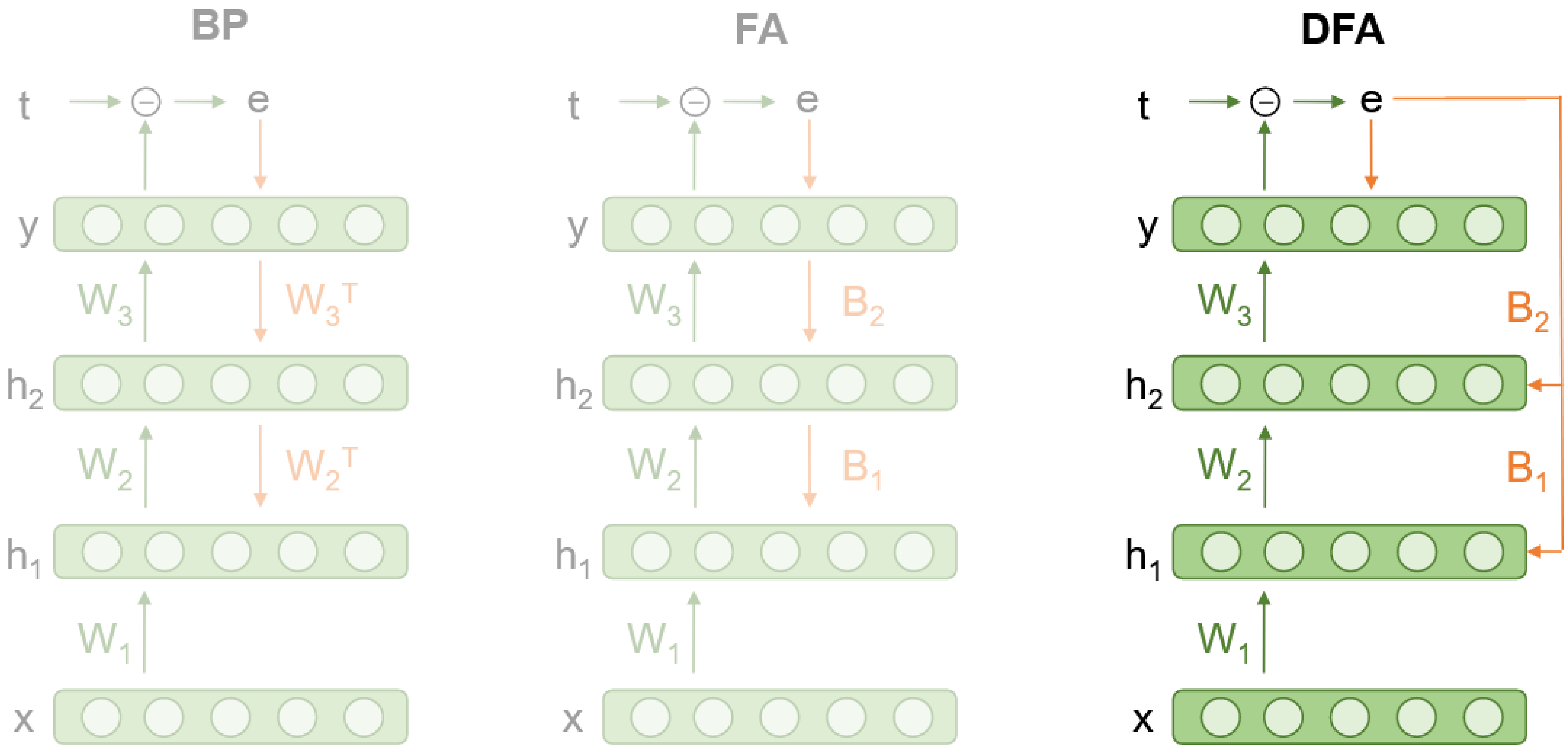


Lillicrap et al., 2016



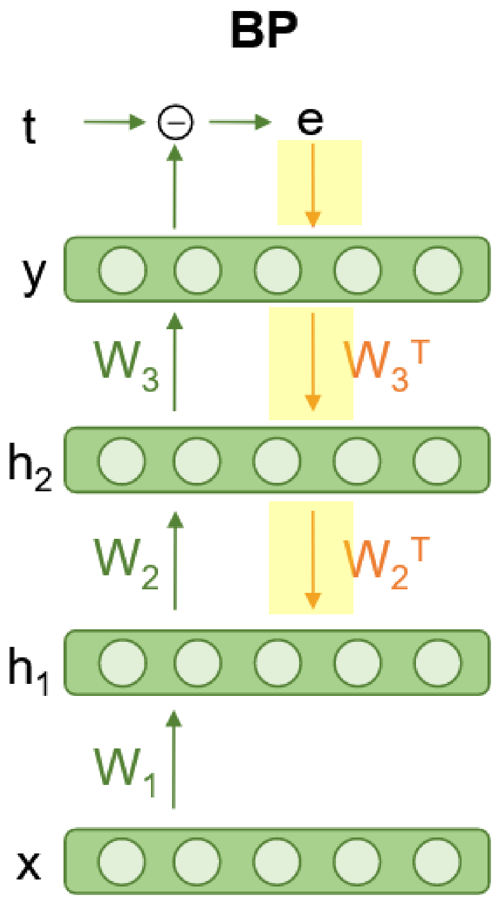
Akrout et al., 2019

Alternatives to BP: relaxing symmetry requirements

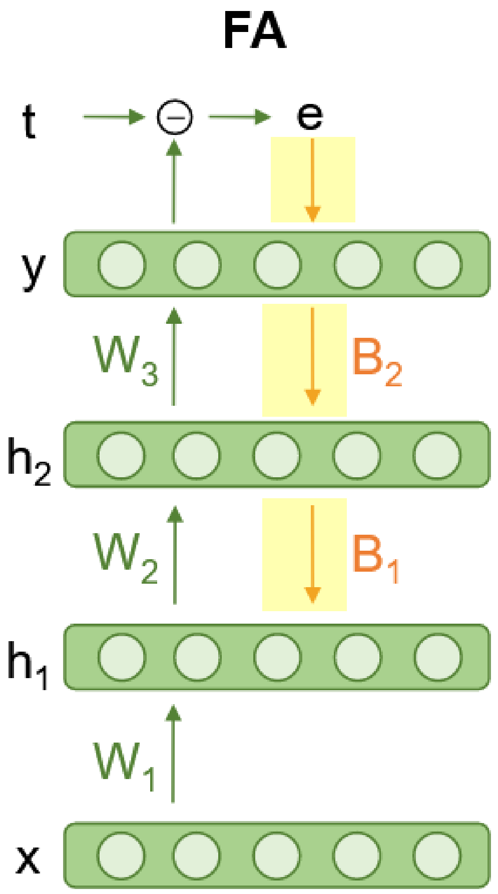


Rumelhart et al., 1995 | Lillicrap et al., 2016 | A. Nokland, 2016

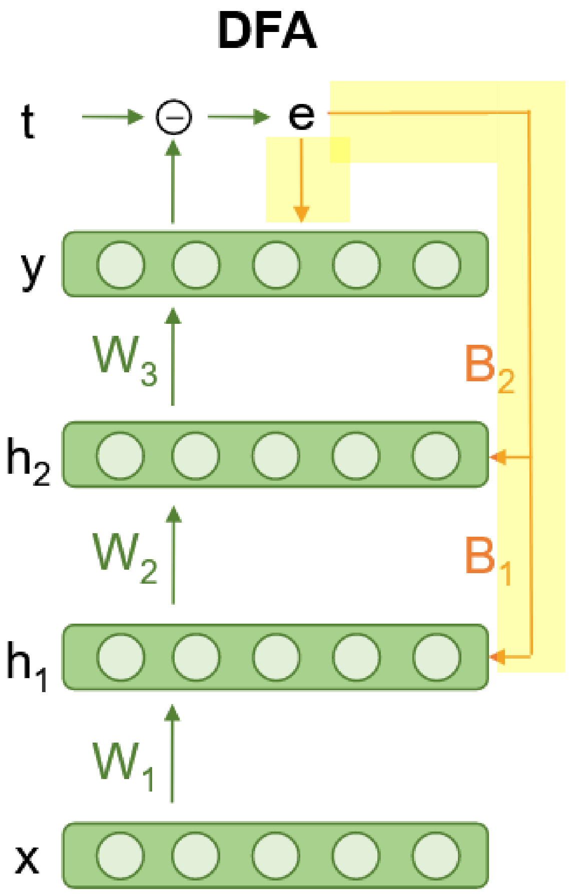
Alternatives to BP: relaxing symmetry requirements



Rumelhart et al., 1995



Lillicrap et al., 2016



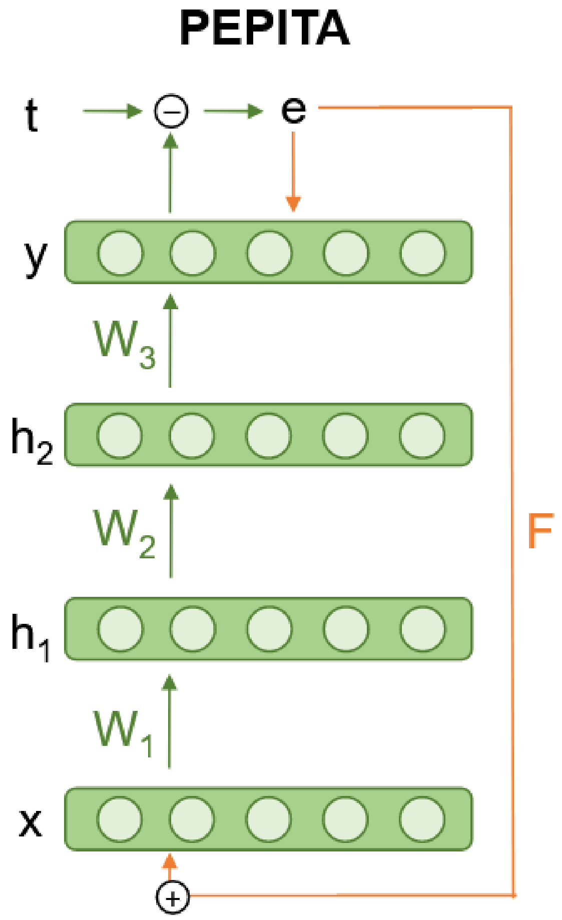
A. Nokland, 2016

The Backward Pass

The backward pass implies:

- Weight updates relying on **non-local** information
- **Freezing activity** for the update phase
- At least **partial update locking**

Remove the backward pass



Dellaferrera & Kreiman, ICML (2022)

Outline

» Neuro-inspired AI
- Why Backpropagation is biologically implausible
- Overview of alternative solutions to credit assignment

» **PEPITA: error-driven input modulation**
- Replacing the backward pass with a second forward pass
- Results on image classification tasks
- Soft alignment dynamics
- Approximating PEPITA to *Adaptive Feedback Alignment*: analytical characterization
- Improving alignment with weight mirroring

» Forward-Forward algorithm
- Idea and results
- Similarities with PEPITA's update rule

» Forward learning with top-down feedback
- Biological considerations

The PEPITA learning rule

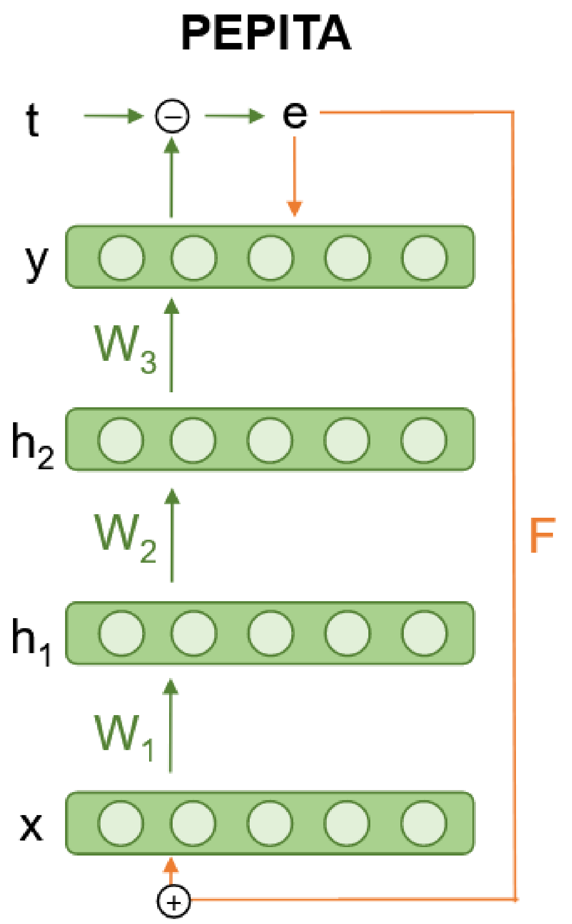
» **PEPITA** = Present the Error to Perturb the Input To modulate Activity

» Substitutes the standard Forward+Backward scheme with **two Forward Passes**

- *<u>Standard</u> Forward pass* → same as for standard algorithms
- *<u>Modulated</u> Forward pass* → input is modulated by the error

» F = **projection matrix** to add the error onto the input

» Update relies **on difference of activations** between *Standard* and *Modulated pass*



The PEPITA learning rule for Fully Connected Neural Networks

» **PEPITA** = Present the Error to Perturb the Input To modulate Activity

Algorithm 1 Implementation of PEPITA

Given: Input (x) and label ($target$)

#standard forward pass

$h_0 = x$

for $\ell = 1, ..., L$

$\quad h_\ell = \sigma_\ell(W_\ell h_{\ell-1})$

$e = h_L - target$ ⟵ » Present the Error ...

#modulated forward pass

$h_0^{err} = x + Fe$ ⟵ » ... to Perturb the Input...

for $\ell = 1, ..., L$

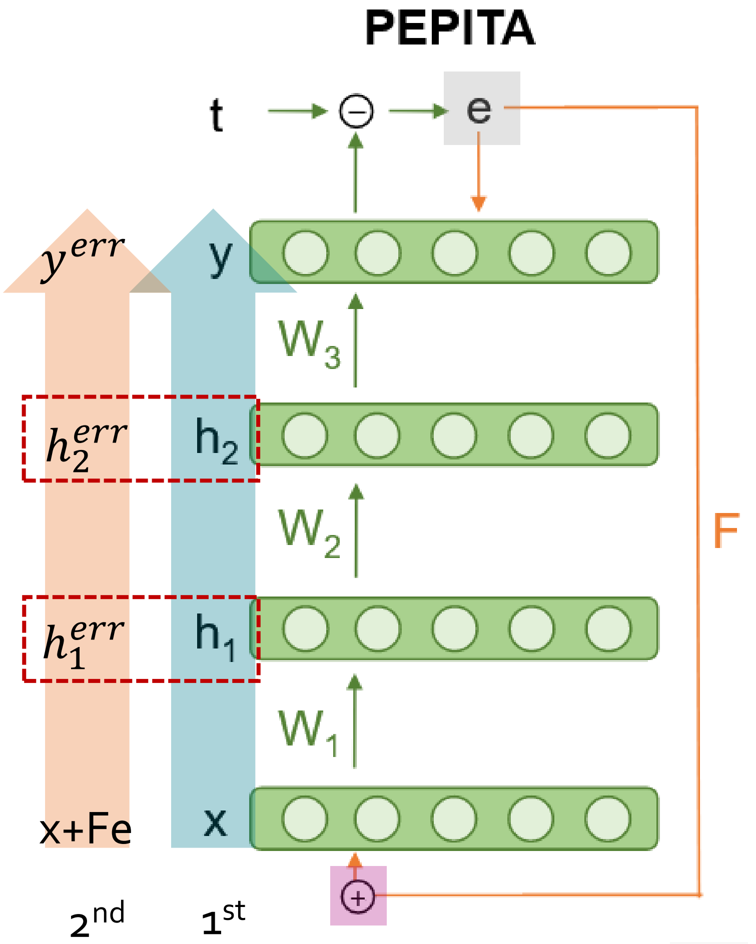
$\quad h_\ell^{err} = \sigma_\ell(W_\ell h_{\ell-1}^{err})$ ⟵ » ... To modulate Activity

$\quad$ **if** $\ell < L$:

$$\Delta W_\ell = \boxed{(h_\ell - h_\ell^{err})} \cdot (h_{\ell-1}^{err})^T$$

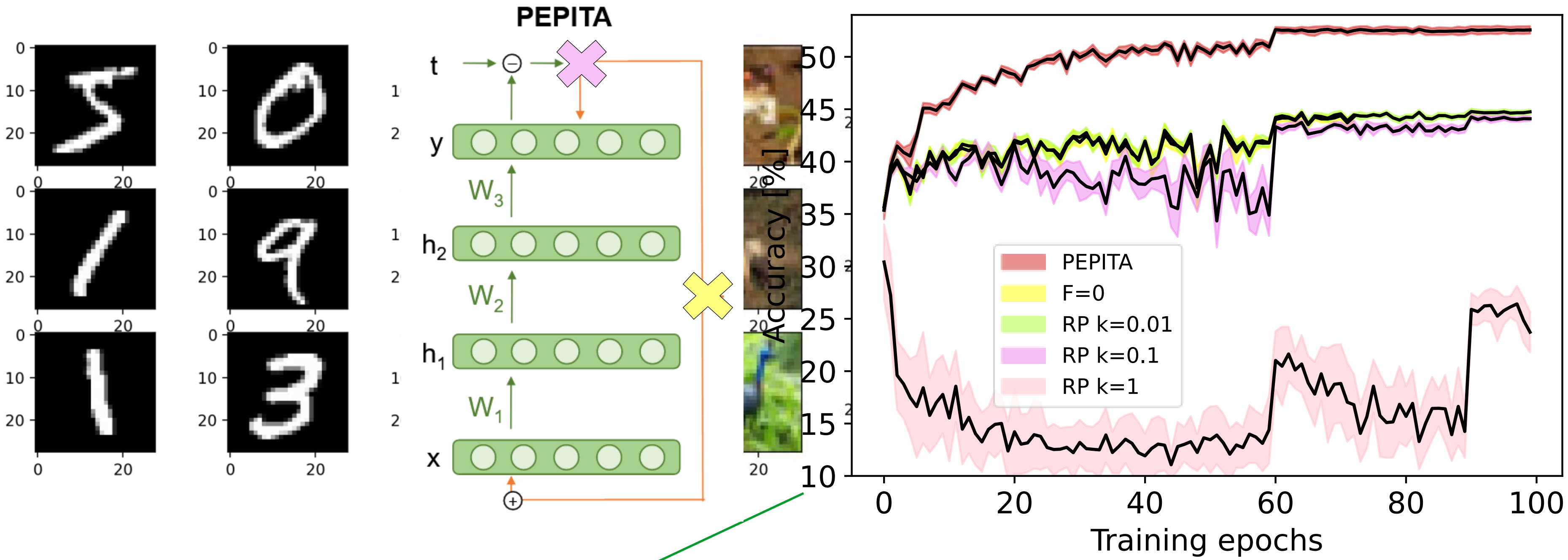
$\quad$ **else:**

$$\Delta W_\ell = e \cdot (h_{\ell-1}^{err})^T$$



Testing PEPITA on image classification tasks - experimental results

- » Results for PEPIT
- » In some tasks, PE
- » PEPITA always o
- » The PEPITA conv
  - Useful 2D featu



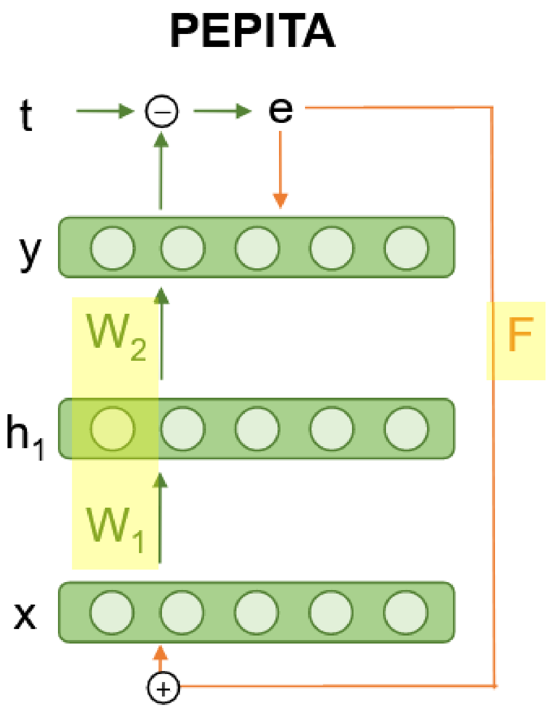
Architecture:
1 hidden layer +
1 output layer

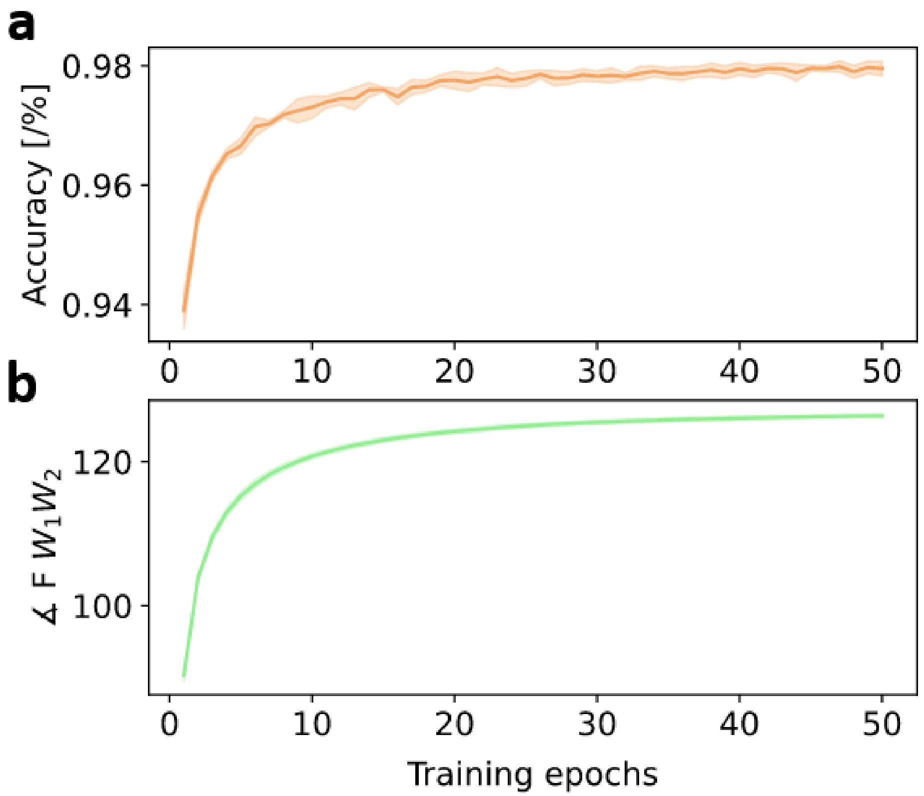
	Fully connected models			Convolutional models		
	MNIST	CIFAR10	CIFAR100	MNIST	CIFAR10	CIFAR100
BP	98.63±0.03	55.27±0.32	27.58±0.09	98.86±0.04	64.99±0.32	34.20±0.20
FA	98.42±0.07	53.82±0.24	24.61±0.28	98.50±0.06	57.51±0.57	27.15±0.53
DRTP	95.10±0.10	45.89±0.16	18.32±0.18	97.32±0.25	50.53±0.81	20.14±0.68
PEPITA	98.01±0.09	52.57±0.36	24.91±0.22	98.29±0.13	56.33±1.35	27.56±0.60

Why it works: soft-antialignment

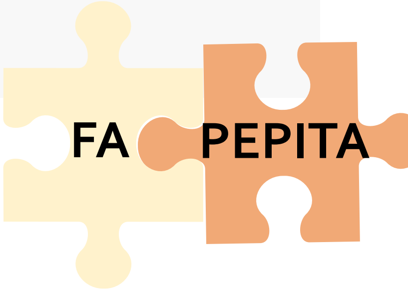
» **Soft-antialignment**

- Angle between
  - projection matrix F and
  - product between the forward weight matrices
- Evolution during learning → soft antialignment
- <u>Analytically proven</u> for one-hidden layer linear network





Approximating PEPITA to an Adaptive Feedback Alignment algorithm



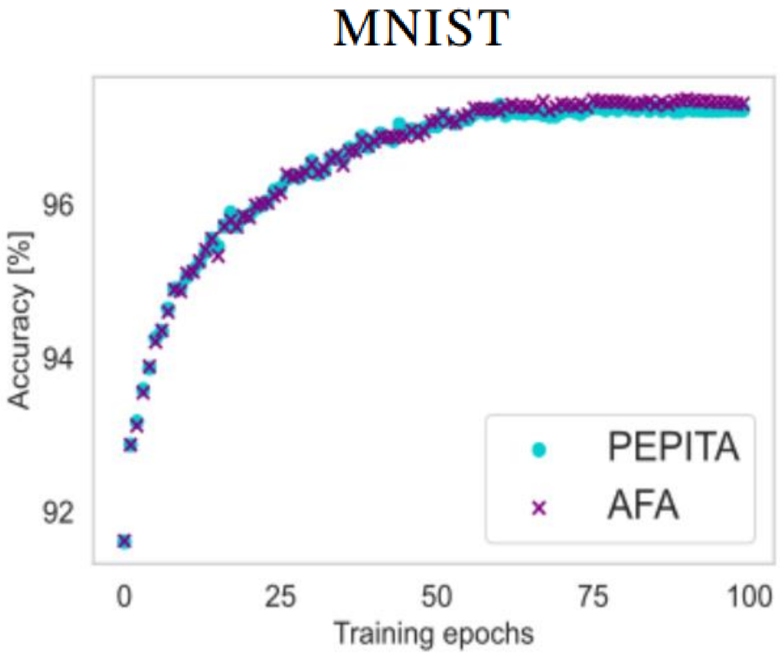
» First order Taylor expansion

$$\Delta W_\ell = (h_\ell - h_\ell^{err}) \cdot (h_{\ell-1}^{err})^T$$

$$f(a+h) \simeq f(a) + h f'(a)$$

$$\begin{aligned}
h_1 - h_1^{err} &= \sigma_1(W_1 x) - \sigma_1(W_1(x - Fe)) = \\
&= \sigma_1(W_1 x) - \sigma_1(W_1 x - W_1 Fe)) = \\
&\simeq \cancel{\sigma_1(W_1 x)} - [\cancel{\sigma_1(W_1(x))} - W_1 Fe \sigma_1'(W_1 x)] = \\
&= W_1 Fe \sigma_1'(W_1 x) = \\
&= \underbrace{W_1 \underbrace{Fe}_{\text{FA}}}_{} h_1'.
\end{aligned}$$

AFA = Adaptive (W) Feedback Alignment

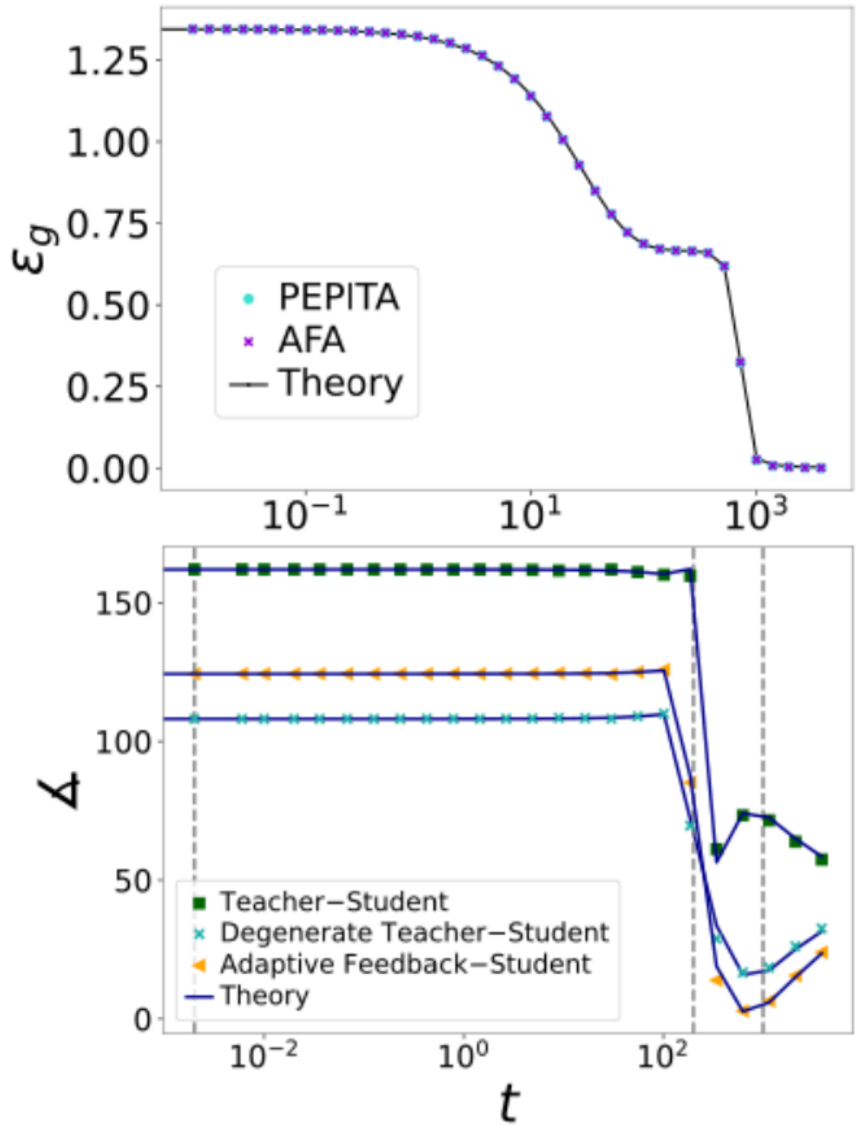


Approximating PEPITA to an Adaptive Feedback Alignment algorithm

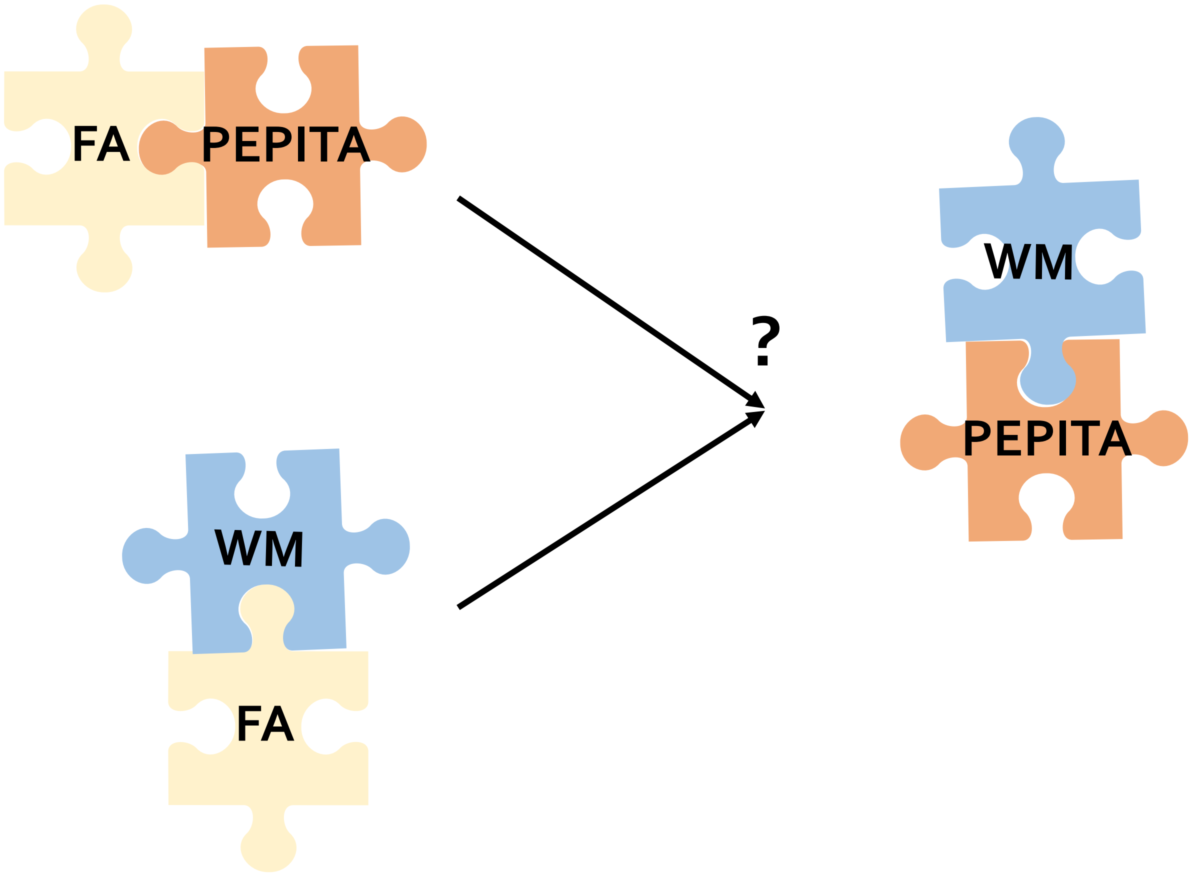
» Analytical characterization of AFA
- Teacher student set-up
  - generative model for the data
  - D–dimensional standard Gaussian input vectors
  - label generated by a two-layer teacher network with fixed random weights
  - Two-layer student network trained with AFA and an online (or one-pass)

- Characterize the dynamics of the mean-squared generalization error
  - infinite-dimensional limit of input dimension and number of samples
  - excellent agreement between infinite-dimensional theory and experiments

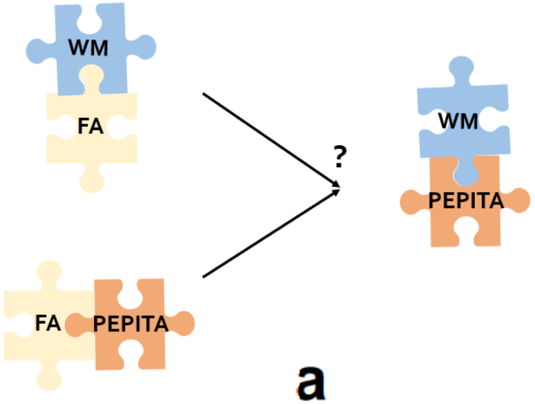
- Dynamics of alignment of the second-layer student weights W2 with:
  - the AF matrix W1F,
  - the second-layer teacher weights,
  - the second-layer teacher weights of the closest degenerate solution.



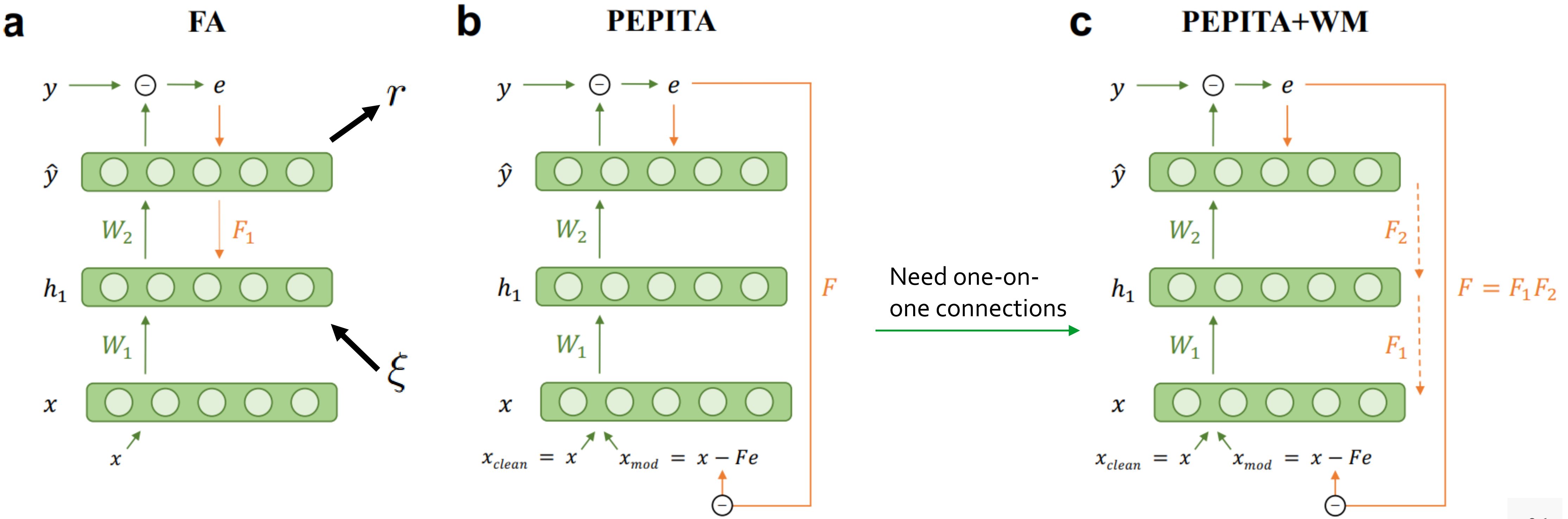
Improving PEPITA's alignment with weight mirroring



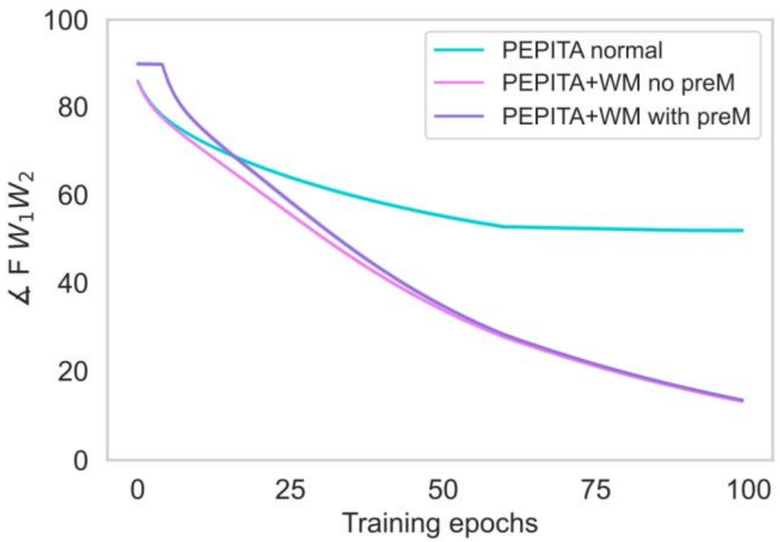
Improving PEPITA's alignment with weight mirroring



$$\Delta F_\ell = \eta_F \xi r^\top$$



Improving PEPITA's alignment with weight mirroring



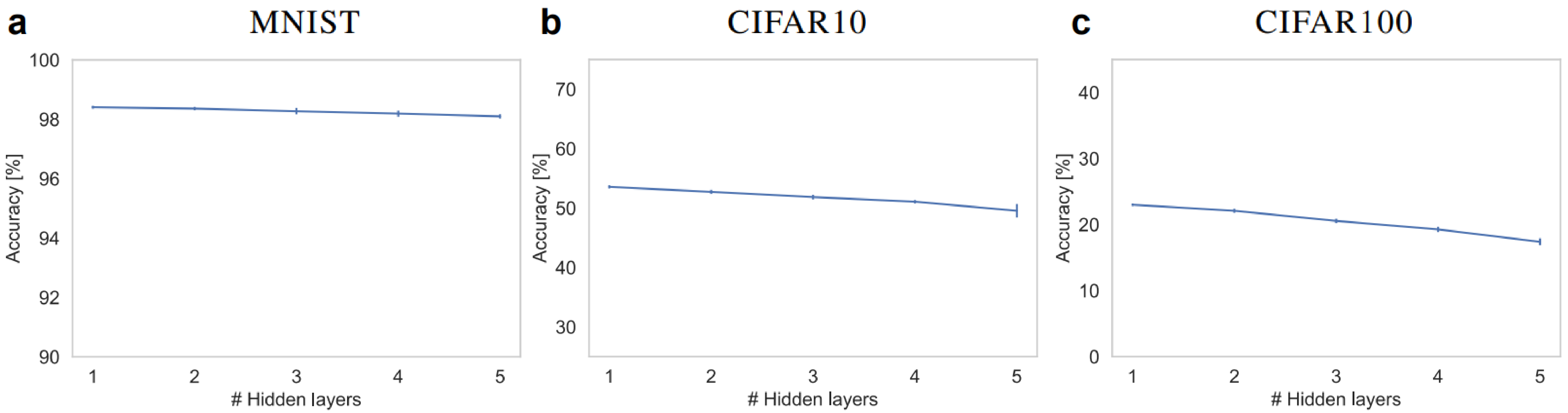
← Note: the sign is flipped x^{err}=x-Fe

Architecture:
1 hidden layer +
1 output layer

	W. Decay	Norm.	Mirror	MNIST	CIFAR10	CIFAR100
PEPITA	✗	✗	✗	98.02±0.08	52.45±0.25	24.69±0.17
	✓	✗	✗	98.12±0.08	53.05±0.23	24.86±0.18
	✗	✓	✗	**98.41**±0.08	**53.51**±0.23	22.87±0.25
	✗	✗	✓	98.05±0.08	52.63±0.30	**27.07**±0.11
	✓	✗	✓	98.10±0.12	53.46±0.26	**27.04**±0.19
	✗	✓	✓	**98.42**±0.05	**53.80**±0.25	24.20±0.36

Training deeper fully connected models

» Adding activation normalization allows to train up to 6 layer networks

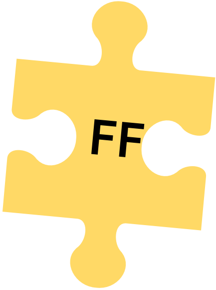


Outline

» **Neuro-inspired AI**
- Why Backpropagation is biologically implausible
- Overview of alternative solutions to credit assignment

» **PEPITA: error-driven input modulation**
- Replacing the backward pass with a second forward pass
- Results on image classification tasks
- Soft alignment dynamics
- Approximating PEPITA to *Adaptive Feedback Alignment*: analytical characterization
- Improving alignment with weight mirroring

» **Forward-Forward algorithm**
- Idea and results
- Similarities with PEPITA's update rule



» **Forward learning with top-down feedback**
- Biological considerations

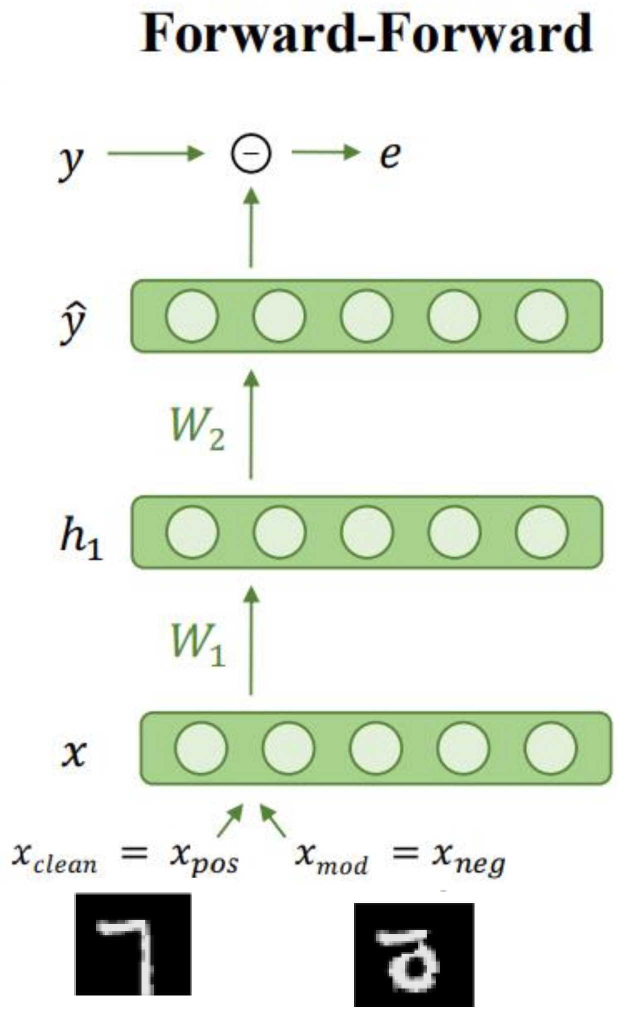
The Forward-Forward algorithm

» Two forward passes per sample:
- the positive pass operate on real data
- the negative pass operates on "negative data"

» In the positive pass:
- weights updated to increase the goodness in hidden layers

» In the negative pass:
- weights updated to decrease the goodness in hidden layers

» One measure of goodness
- sum of the squared neural activities



Beyond PEPITA: towards time locality

PEPITA

Algorithm 1 Implementation of PEPITA

Given: Input (x) and label ($target$)

#standard forward pass

$h_0 = x$

for $\ell = 1, \ldots, L$

$\quad h_\ell = \sigma_\ell(W_\ell h_{\ell-1})$

$e = h_L - target$

#modulated forward pass

$h_0^{err} = x + Fe$

for $\ell = 1, \ldots, L$

$\quad h_\ell^{err} = \sigma_\ell(W_\ell h_{\ell-1}^{err})$

$\quad$ **if** $\ell < L$:

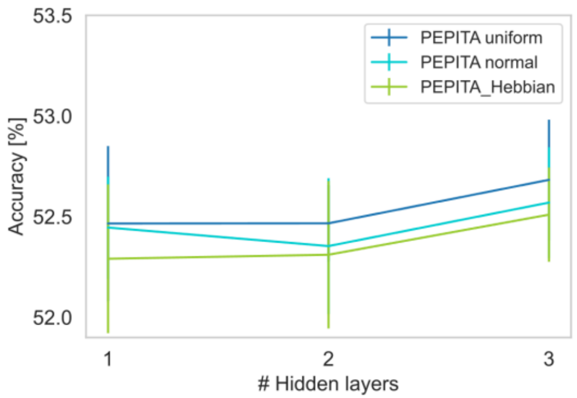
$\quad\quad \Delta W_\ell = (h_\ell - h_\ell^{err}) \cdot (h_{\ell-1}^{err})^T$

$\quad$ **else:**

$\quad\quad \Delta W_\ell = e \cdot (h_{\ell-1}^{err})^T$

$\quad$ #apply update

$\quad W_\ell(t+1) = W_\ell(t) - \eta \Delta W_\ell$



$$\Delta W_\ell = h_\ell \cdot h_{\ell-1}^{errT} - h_\ell^{err} \cdot h_{\ell-1}^{errT}$$
$$\simeq h_\ell \cdot h_{\ell-1}^{T} - h_\ell^{err} \cdot h_{\ell-1}^{errT}$$

PEPITA 2.0

Algorithm 2 Implementation of PEPITA local in space

Given: Input (x) and label ($target$)

#standard forward pass

$h_0 = x$

for $\ell = 1, \ldots, L$

$\quad h_\ell = \sigma_\ell(W_\ell h_{\ell-1})$

$\quad \Delta W_\ell^+ = h_\ell \cdot h_{\ell-1}^T$

$\quad$ #apply update for the positive phase

$\quad W_\ell^+(t+1) = W_\ell(t) - \eta \Delta W_\ell^+$

$e = h_L - target$

#modulated forward pass

$h_0^{err} = x + Fe$

for $\ell = 1, \ldots, L$

$\quad h_\ell^{err} = \sigma_\ell(W_\ell^+ h_{\ell-1}^{err})$

$\quad$ **if** $\ell < L$:

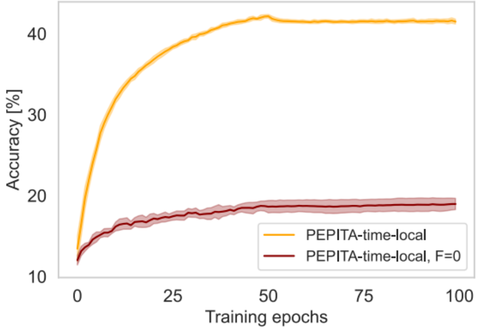
$\quad\quad \Delta W_\ell^- = -h_\ell^{err} \cdot h_{\ell-1}^{errT}$

$\quad$ **else:**

$\quad\quad \Delta W_\ell^- = -target \cdot h_{\ell-1}^{errT}$

$\quad$ #apply update for the negative phase

$\quad W_\ell(t+1) = W_\ell^+(t+1) - \eta \Delta W_\ell^-$



PEPITA: weight update equivalent to the Forward-Forward framework

Forward-Forward framework

- Goodness as the sum of squared neural activities
  - h^2_l for the positive pass and
  - $(h^{err}_l)^2$ for the negative pass.
- Local loss function J_l for layer l = the sum of
  - loss function of the positive pass J^+_l and
  - loss function of the negative pass J^-_l

$$J_\ell = \|h_\ell\|^2 - \|h_\ell^{err}\|^2$$

Equivalence of weight update

$$\begin{aligned}
\frac{1}{2}\frac{\partial J_\ell}{\partial W_\ell} &= \frac{1}{2}\left(\frac{\partial\|h_\ell\|^2}{\partial W_\ell} - \frac{\partial\|h_\ell^{err}\|^2}{\partial W_\ell}\right) \\
&= \frac{1}{2}\left(\frac{\partial\|\sigma(W_\ell h_{\ell-1})\|^2}{\partial W_\ell} - \frac{\partial\|\sigma(W_\ell h_{\ell-1}^{err})\|^2}{\partial W_\ell}\right) \\
&= \sigma(W_\ell h_{\ell-1}) \odot \sigma'(W_\ell h_{\ell-1}) h_{\ell-1}^\top \\
&\quad - \sigma(W_\ell h_{\ell-1}^{err}) \odot \sigma'(W_\ell h_{\ell-1}^{err}) h_{\ell-1}^{err\top} \\
&= (\sigma'(W_\ell h_{\ell-1}) \odot h_\ell h_{\ell-1}^\top \\
&\quad - \sigma'(W_\ell h_{\ell-1}^{err}) \odot h_\ell^{err} h_{\ell-1}^{err}) \\
&= (h'_\ell \odot h_\ell) h_{\ell-1}^\top - (h_\ell^{err\prime} \odot h_\ell^{err}) h_{\ell-1}^{err\top}.
\end{aligned} \tag{9}$$

PEPITA- Hebbian

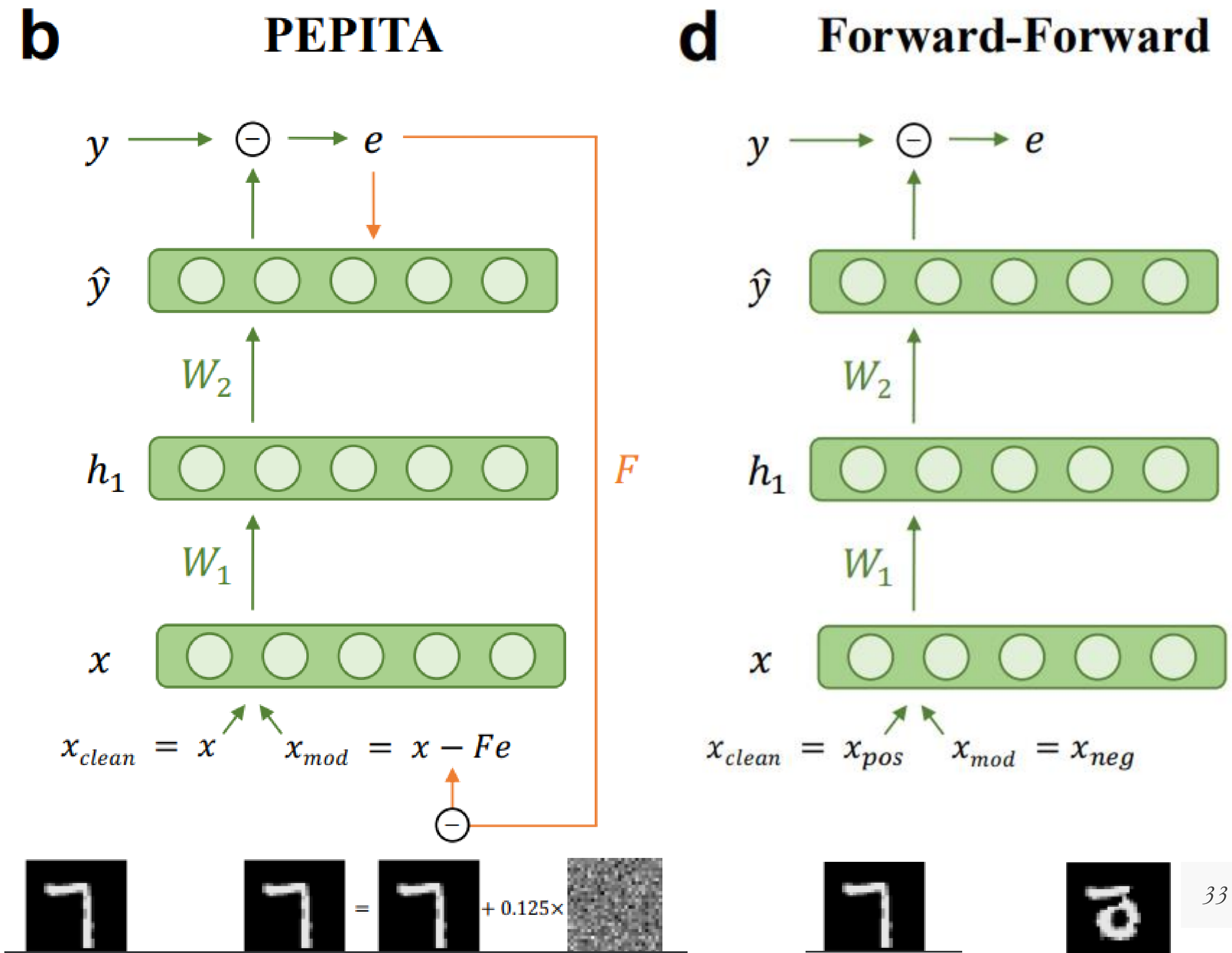
$$\begin{aligned}
\Delta W_\ell &= h_\ell \cdot h_{\ell-1}^{errT} - h_\ell^{err} \cdot h_{\ell-1}^{errT} \\
&\simeq h_\ell \cdot h_{\ell-1}^T - h_\ell^{err} \cdot h_{\ell-1}^{errT}
\end{aligned}$$

If ReLU non-linearity:

$$\frac{1}{2}\frac{\partial J_\ell}{\partial W_\ell} = h_\ell h_{\ell-1}^\top - h_\ell^{err} h_{\ell-1}^{err\top}$$

Differences between PEPITA and FF

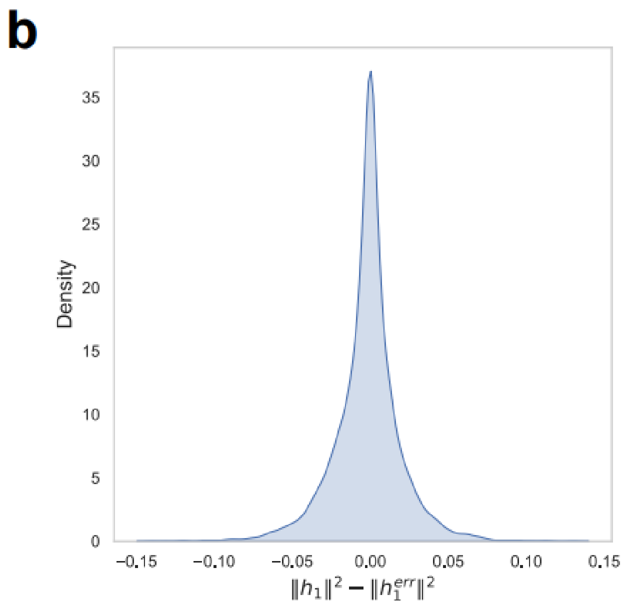
» Method to generate the modulated sample
  - PEPITA → add error → top-down feedback
  - FF → hybrid mask

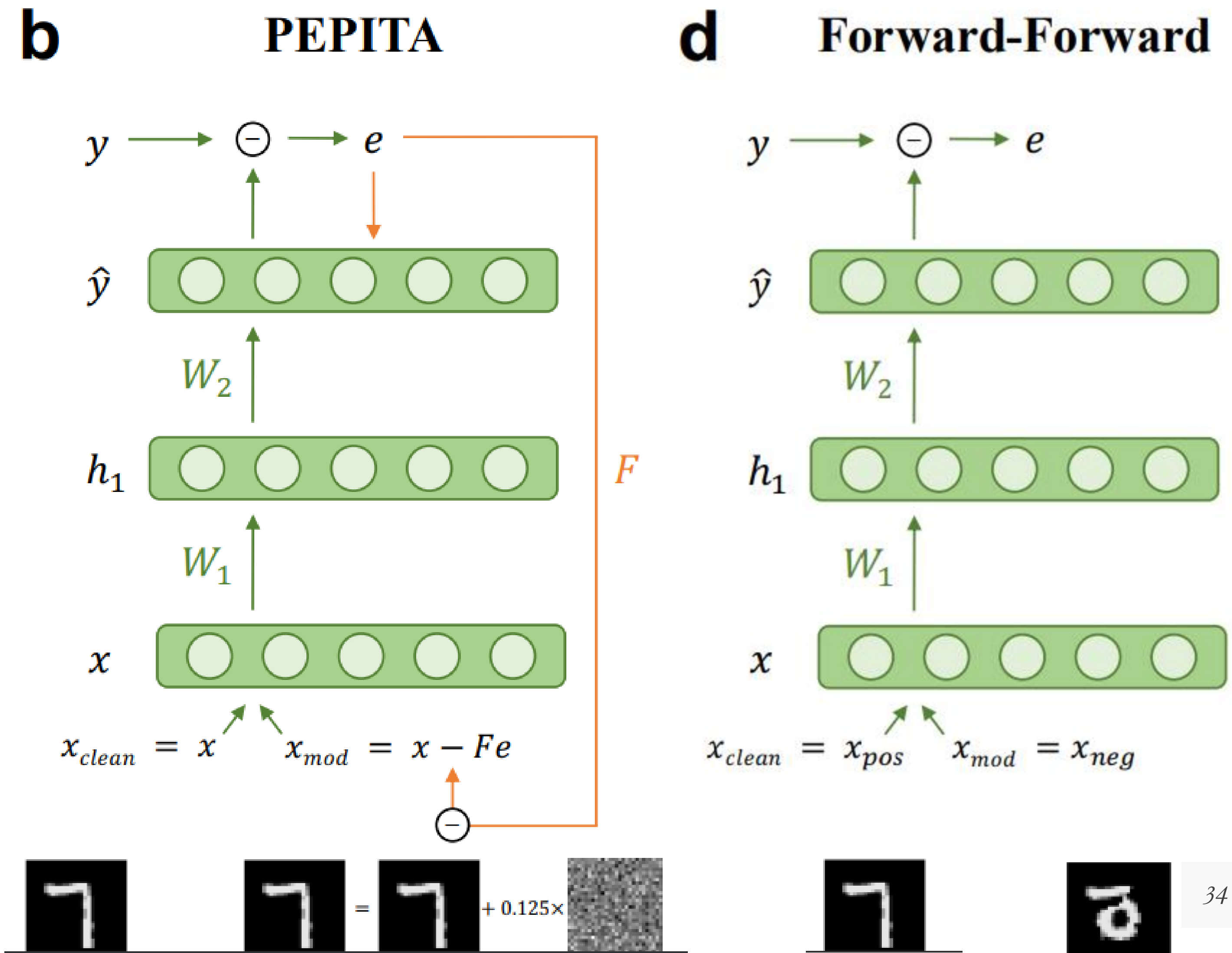


Differences between PEPITA and FF

» Method to generate the modulated sample
- PEPITA → add error → top-down feedback
- FF → hybrid mask

» PEPITA does not maximize (resp. minimize) the activations in the clean (resp. modulated) pass





Outline

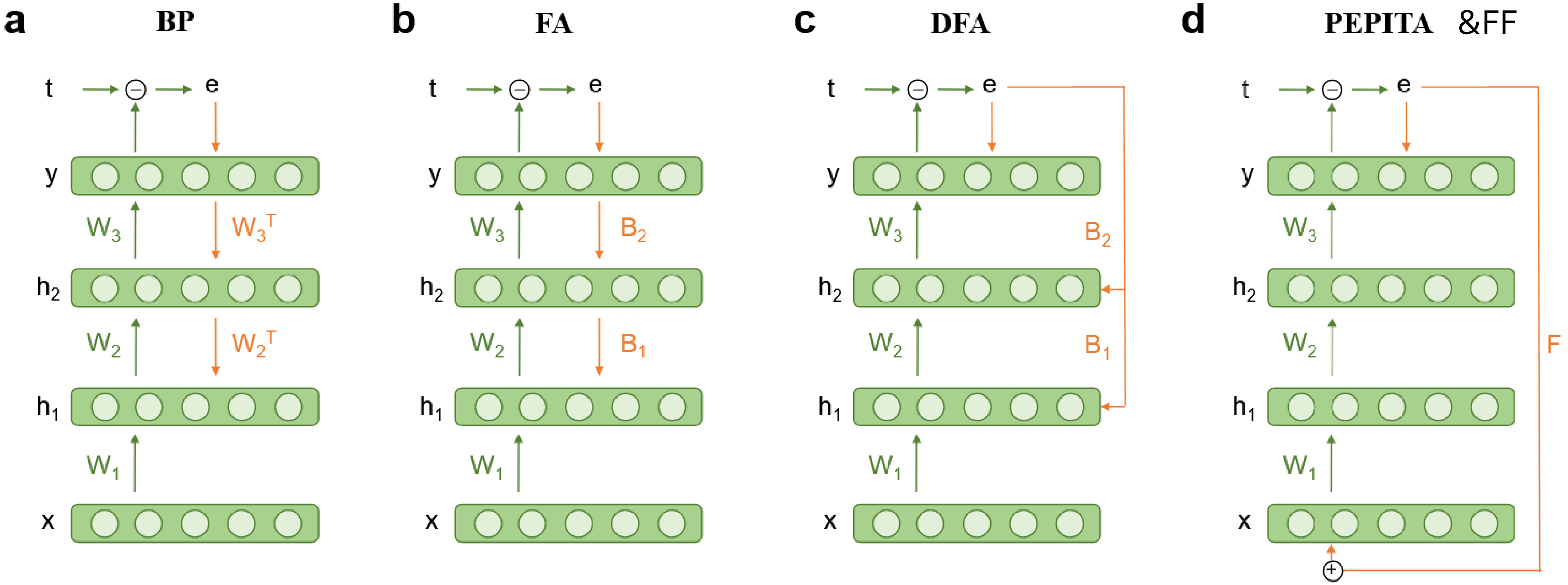
» **Neuro-inspired AI**
- Why Backpropagation is biologically implausible
- Overview of alternative solutions to credit assignment

» **PEPITA: error-driven input modulation**
- Replacing the backward pass with a second forward pass
- Results on image classification tasks
- Soft alignment dynamics
- Approximating PEPITA to *Adaptive Feedback Alignment*: analytical characterization
- Improving alignment with weight mirroring

» **Forward-Forward algorithm**
- Idea and results
- Similarities with PEPITA's update rule

» **Forward learning with top-down feedback**
- Biological considerations

PEPITA solves the biologically implausible aspects of BP

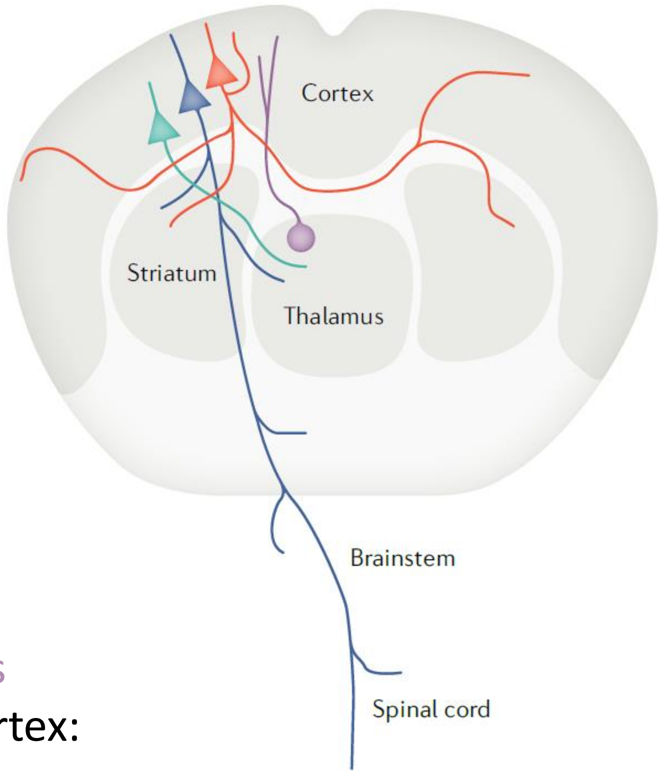


$\Delta W_\ell =$	$-(W_{\ell+1}^T \delta a_{\ell+1}) \odot f'(a_\ell) h_{\ell-1}^T$	$-(B_\ell^T \delta a_{\ell+1}) \odot f'(a_\ell) h_{\ell-1}^T$	$-(B_\ell^T e) \odot f'(a_\ell) h_{\ell-1}^T$	$(h_\ell - h_\ell^{err}) \cdot (h_{\ell-1}^{err\,T})$
WEIGHT-TRANSPORT-FREE	✗	✓	✓	✓
LOCAL RULE	✗	✗	✗	✓
FREEZING OF ACTIVITY	✗	✗	✗	✓
UPDATE-UNLOCKED	✗	✗	PARTIALLY	PARTIALLY

Analysis of PEPITA from a biological standpoint

PEPITA solves the BP's issues of biological plausibility, but it introduces additional elements:

» Projection of the error onto the input through a fixed random matrix
  - Reminiscent of cortico-thalamo-cortical loops



In the thalamus:
- Thalamocortical (TC) neurons

Excitatory neurons in the neocortex:
- Intratelencephalic (IT)
- Pyramidal tract (PT)
- Corticothalamic (CT) neurons

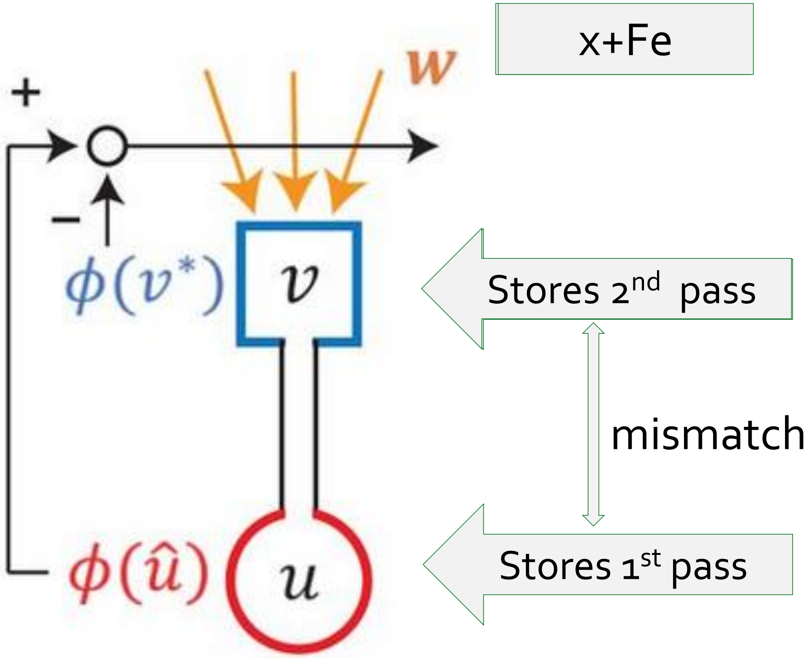
Shepherd & Yamawaki, 2021

Analysis of PEPITA from a biological standpoint

PEPITA solves the BP's issues of biological plausibility, but it introduces additional elements:

» Projection of the error onto the input through a fixed random matrix
- Reminiscent of cortico-thalamo-cortical loops

» Storing of the activation of the *Standard pass* until the *Modulated pass*
- Can be implemented in biological neurons through mismatch between dendritic and somatic activity



Asabuki & Fukai, 2020

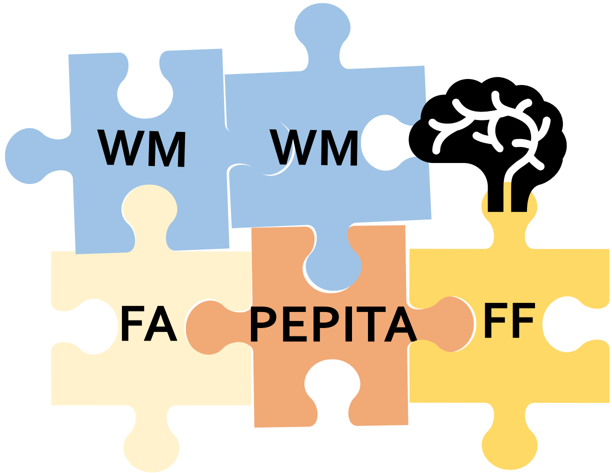
Summary and Outlook

» **PEPITA and FF**

- Are novel training schemes relying only on **forward computations**
- Solve weight transport, freezing of neural activity, non-local weight updates and backward locking
- Achieve performance on-par with FA on simple image classification tasks
- PEPITA's weight update is that of FF with top-down feedback

» **PEPITA**

- Can be approximated to Adaptive Feedback Alignment to characterize its dynamics
- Its performance benefits from better alignment (weight mirroring)



» **Challenges**

- Performance **does not improve with depth**
- Residual connection, intermediate error-driven modulation, training the F matrix

» **Promising avenues for exploration**

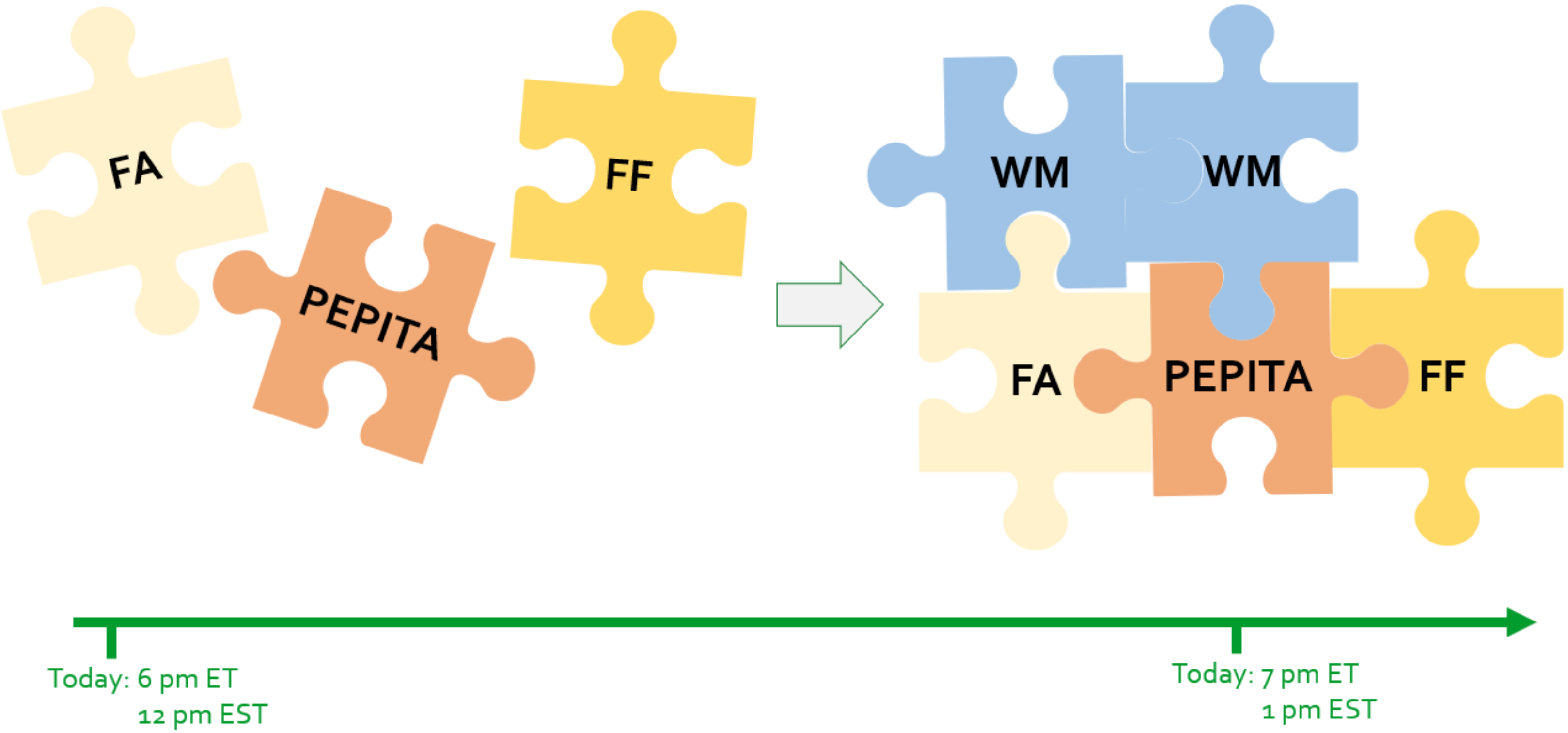
- PEPITA is not gradient-based: could it be more robust to gradient-based adversarial attacks?
- Application to object recognition on videos:
  - Consecutive frames need only one forward pass

Thank you for your attention!

» Questions?

» Ideas?

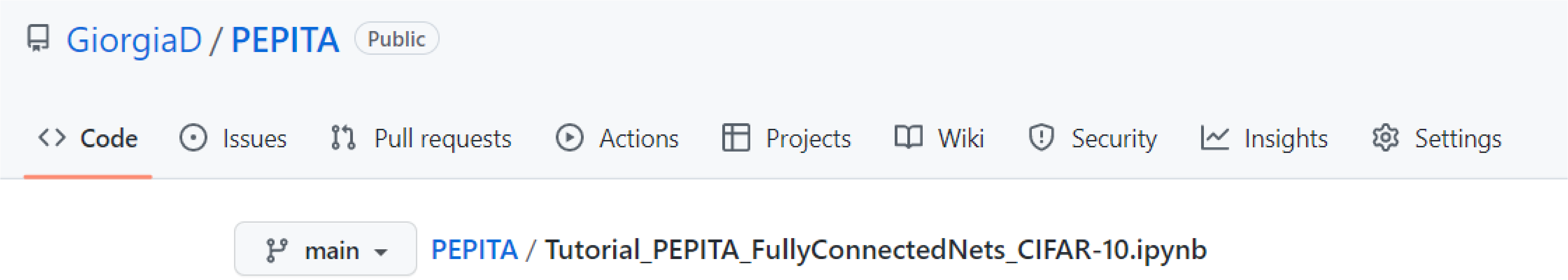
» Suggestions?



giorgia.dellaferrera@gmail.com

Coding tutorial: Implementing PEPITA with Pytorch 1/11

- Today → Code (ICML 2022): https://github.com/GiorgiaD/PEPITA
- Code with Pytorch lightning (arXiv:2302.05440): https://drive.google.com/drive/u/1/folders/1wqHqtZx2NVuxpdjQuYUVVf1A8v-88oFS



Coding tutorial: Implementing PEPITA with Pytorch 1/11

Import libraries

In [1]:
```
# import torch libraries
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable

# import other libraries
import numpy as np
import matplotlib.pyplot as plt
import copy
```

Coding tutorial: Implementing PEPITA with Pytorch 2/11

Define Network architecture

In [2]:
```
# models with Dropout
class NetFC1x1024DOcust(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(32*32*3,1024,bias=False)
        self.fc2 = nn.Linear(1024, 10,bias=False)

        # initialize the layers using the He uniform initialization scheme
        fc1_nin = 32*32*3 # Note: if dataset is MNIST --> fc1_nin = 28*28*1
        fc1_limit = np.sqrt(6.0 / fc1_nin)
        torch.nn.init.uniform_(self.fc1.weight, a=-fc1_limit, b=fc1_limit)
        fc2_nin = 1024
        fc2_limit = np.sqrt(6.0 / fc2_nin)
        torch.nn.init.uniform_(self.fc2.weight, a=-fc2_limit, b=fc2_limit)

    def forward(self, x, do_masks):
        x = F.relu(self.fc1(x))
        # apply dropout --> we use a custom dropout implementation because we nee
        if do_masks is not None:
            x = x * do_masks[0]
        x = F.softmax(self.fc2(x))
        return x
```

Coding tutorial: Implementing PEPITA with Pytorch 3/11

Set hyperparameters and train+test the model

In [3]:
```
# set hyperparameters
## learning rate
eta = 0.01
eta_decay = 0.1
eta_decay_epochs = [60,90]
## number of epochs
num_epochs = 3
## dropout keep rate
keep_rate = 0.9
## loss --> used to monitor performance, but not for parameter updates (PEPITA d
criterion = nn.CrossEntropyLoss()
## optimizer (choose 'SGD' o 'mom')
optim = 'mom' # --> default in the paper
if optim == 'SGD':
    gamma = 0
elif optim == 'mom':
    gamma = 0.9
## batch size
batch_size = 64 # --> default in the paper
```

Coding tutorial: Implementing PEPITA with Pytorch 4/11

In [4]:
```
# initialize the network
net = NetFC1x1024DOcust()
```

In [5]:
```
# define B --> this is the F projection matrix in the paper (here named B becaus
nin = 32*32*3
sd = np.sqrt(6/nin)
B = (torch.rand(nin,10)*2*sd-sd)*0.05  # B is initialized with the He uniform in
```

In [6]:
```
# load the dataset - CIFAR-10
transform = transforms.Compose(
    [transforms.ToTensor()]) # this normalizes to [0,1]
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

Coding tutorial: Implementing PEPITA with Pytorch 5/11

In [7]:
```
# define function to register the activations --> we need this to compare the ac
activation = {}
def get_activation(name):
    def hook(model, input, output):
        activation[name] = output.detach()
    return hook
for name, layer in net.named_modules():
    layer.register_forward_hook(get_activation(name))
```

In [8]:
```
# do one forward pass to get the activation size needed for setting up the dropou
dataiter = iter(trainloader)
images, labels = next(dataiter)
images = torch.flatten(images, 1) # flatten all dimensions except batch
outputs = net(images,do_masks=None)
layers_act = []
for key in activation:
    if 'fc' in key or 'conv' in key:
        layers_act.append(F.relu(activation[key]))
```

Coding tutorial: Implementing PEPITA with Pytorch 6/11

In [9]:
```
# set up for momentum
if optim == 'mom':
    gamma = 0.9
    v_w_all = []
    for l_idx,w in enumerate(net.parameters()):
        if len(w.shape)>1:
            with torch.no_grad():
                v_w_all.append(torch.zeros(w.shape))
```

In [10]:
```
# Train and test the model
test_accs = []
for epoch in range(num_epochs):  # loop over the dataset multiple times

    # learning rate decay
    if epoch in eta_decay_epochs:
        eta = eta*eta_decay
        print('eta decreased to ',eta)

    # loop over batches
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, target = data
        inputs = torch.flatten(inputs, 1) # flatten all dimensions except batch
        target_onehot = F.one_hot(target,num_classes=10)
```

Coding tutorial: Implementing PEPITA with Pytorch 7/11

```
# create dropout mask for the two forward passes --> we need to use the 
do_masks = []
if keep_rate < 1:
    for l in layers_act[:-1]:
        input1 = l
        do_mask = Variable(torch.ones(inputs.shape[0],input1.data.new(in
        do_masks.append(do_mask)
    do_masks.append(1) # for the last layer we don't use dropout --> jus

# forward pass 1 with original input --> keep track of activations
outputs = net(inputs,do_masks)
layers_act = []
cnt_act = 0
for key in activation:
    if 'fc' in key or 'conv' in key:
        layers_act.append(F.relu(activation[key])* do_masks[cnt_act]) # 
        cnt_act += 1

# compute the error
error = outputs - target_onehot

# modify the input with the error
error_input = error @ B.T
mod_inputs = inputs + error_input
```

Coding tutorial: Implementing PEPITA with Pytorch 8/11

```
# forward pass 2 with modified input --> keep track of modulated activat
mod_outputs = net(mod_inputs,do_masks)
mod_layers_act = []
cnt_act = 0
for key in activation:
    if 'fc' in key or 'conv' in key:
        mod_layers_act.append(F.relu(activation[key])* do_masks[cnt_act]
        cnt_act += 1
mod_error = mod_outputs - target_onehot
```

Coding tutorial: Implementing PEPITA with Pytorch 9/11

```
    # compute the delta_w for the batch
    delta_w_all = []
    v_w = []
    for l_idx,w in enumerate(net.parameters()):
        v_w.append(torch.zeros(w.shape))

    for l in range(len(layers_act)):

        # update for the last layer
        if l == len(layers_act)-1:

            if len(layers_act)>1: # if network has more than one layer
                delta_w = -mod_error.T @ mod_layers_act[-2]
            else: # if only one layer network
                delta_w = -mod_error.T @ mod_inputs

        # update for the first layer
        elif l == 0:
            delta_w = -(layers_act[l] - mod_layers_act[l]).T @ mod_inputs

        # update for the hidden layers (not first, not last)
        elif l>0 and l<len(layers_act)-1:
            delta_w = -(layers_act[l] - mod_layers_act[l]).T @ mod_layers_ac

        delta_w_all.append(delta_w)
```

Coding tutorial: Implementing PEPITA with Pytorch 10/11

```
        # apply the weight change
        if optim == 'SGD': # if SGD without momentum
            for l_idx,w in enumerate(net.parameters()):
                with torch.no_grad():
                    w += eta * delta_w_all[l_idx]/batch_size # specify for which

        elif optim == 'mom': # if SGD with momentum
            for l_idx,w in enumerate(net.parameters()):
                with torch.no_grad():
                    v_w_all[l_idx] = gamma * v_w_all[l_idx] + eta * delta_w_all[l
                    w += v_w_all[l_idx]


        # keep track of the loss
        loss = criterion(outputs, target)
        # print statistics
        running_loss += loss.item()
        if i%500 == 499:
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 500))
            running_loss = 0.0
```

Coding tutorial: Implementing PEPITA with Pytorch 11/11

```
    print('Testing...')
    correct = 0
    total = 0
    # since we're not training, we don't need to calculate the gradients for our
    with torch.no_grad():
        for test_data in testloader:
            test_images, test_labels = test_data
            test_images = torch.flatten(test_images, 1) # flatten all dimensions
            # calculate outputs by running images through the network
            test_outputs = net(test_images,do_masks=None)
            # the class with the highest energy is what we choose as prediction
            _, predicted = torch.max(test_outputs.data, 1)
            total += test_labels.size(0)
            correct += (predicted == test_labels).sum().item()

    print('Test accuracy epoch {}: {} %'.format(epoch, 100 * correct / total))
    test_accs.append(100 * correct / total)

print('Finished Training')
```