Feedback Alignment Algorithms

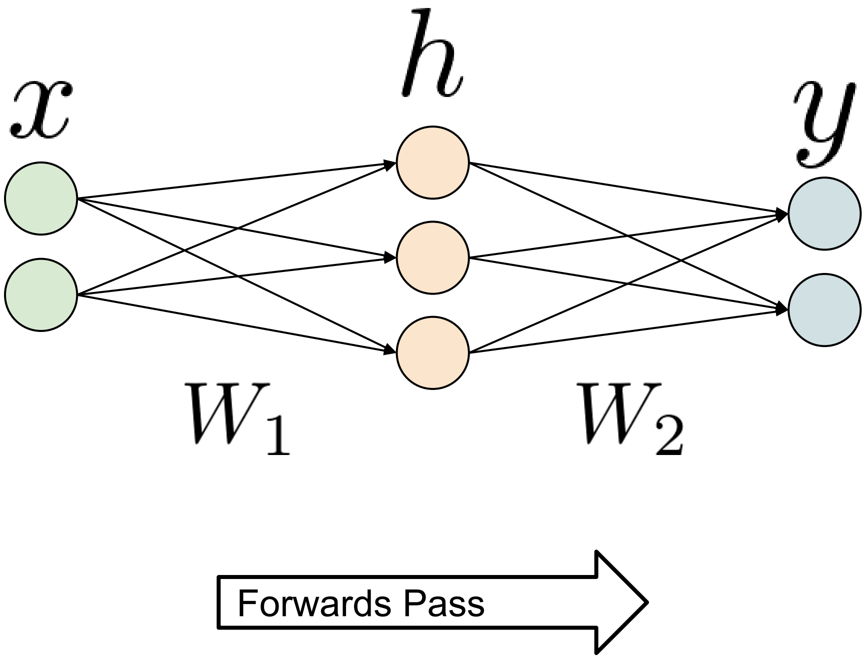
Lisa Zhang, Tingwu Wang, Mengye Ren

Agenda

- Review of Back Propagation
- Random feedback weights support learning in deep neural networks
- Direct Feedback Alignment Provides Learning in Deep Neural Networks
- Critiques
- Code Demos

Review of Back Propagation

Artificial Neural Nets (ANN): review



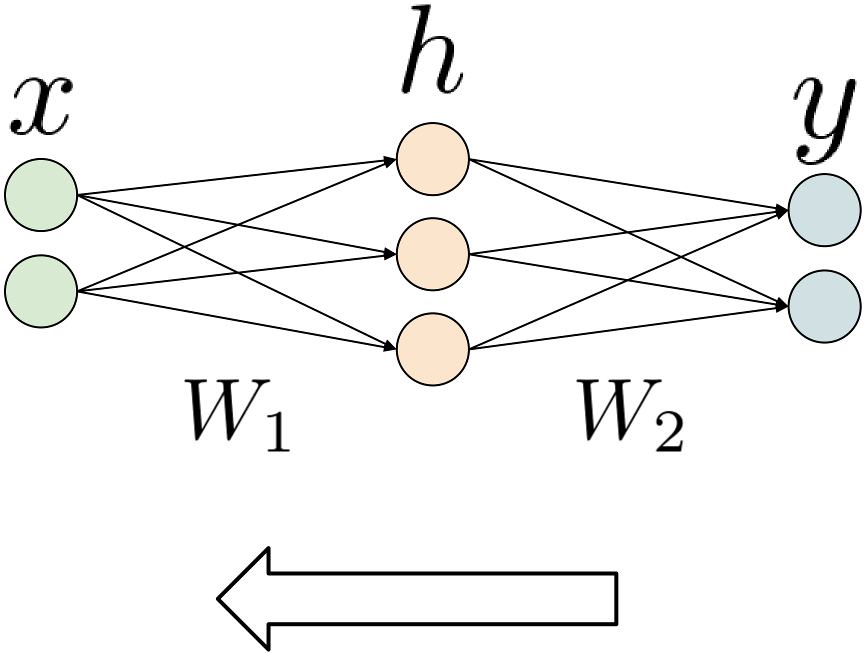
$$z_1 = W_1 x + d_1$$
$$h = \sigma(z_1)$$
$$z_2 = W_2 h + d_2$$
$$y = g(z_2)$$
$$\text{loss} = E(y)$$

Gradient Descent

$$w \leftarrow w - \alpha \frac{\partial E}{\partial w}$$

General method for optimizing a function with respect to some weights.

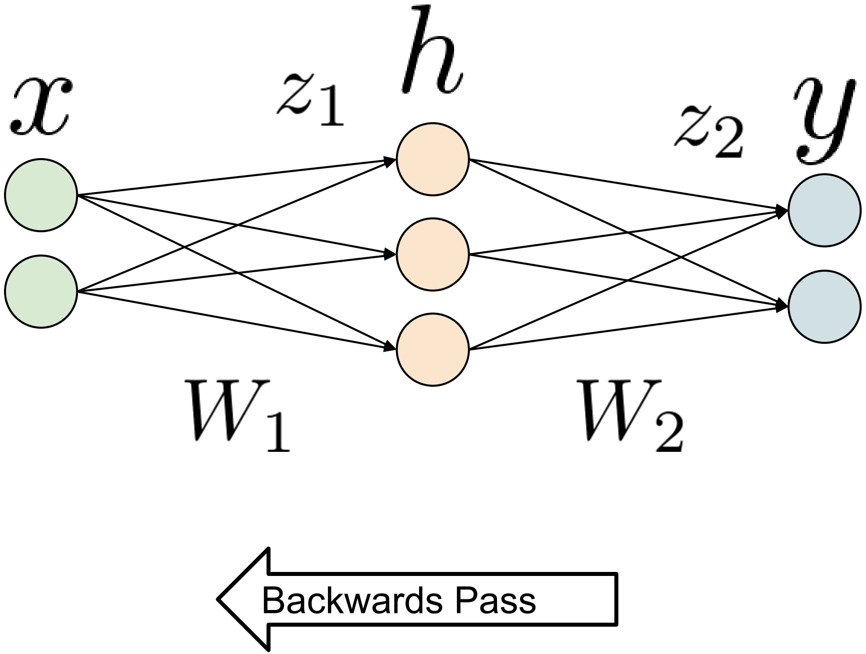
How to efficiently use GD train an ANN?



Way to compute $\frac{\partial E}{\partial w}$ in an efficient way:

… backwards!

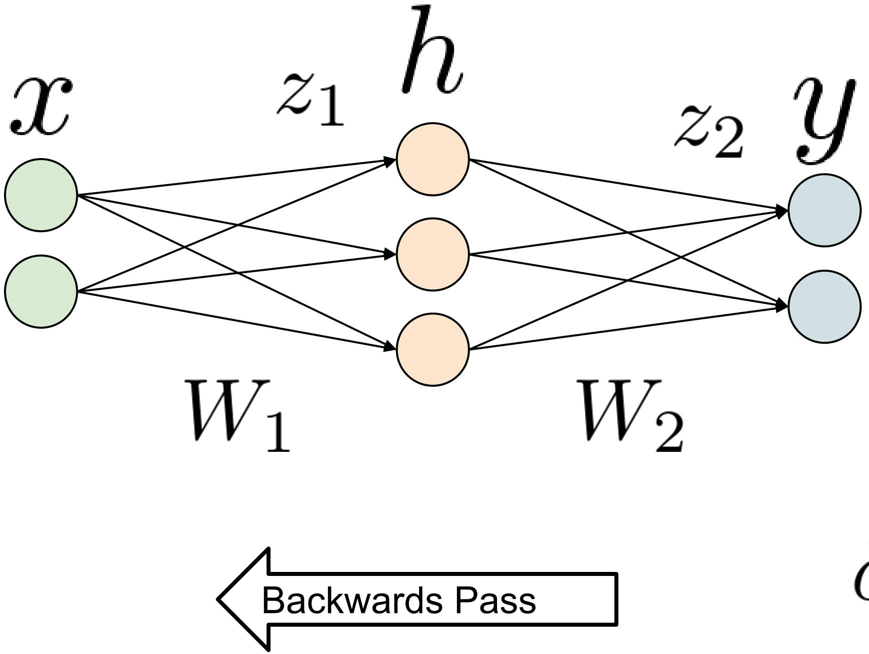
Back-Propagation



$$\frac{\partial E}{\partial z_2} = g'(z_2) \odot \frac{\partial E}{\partial y}$$

$$\frac{\partial E}{\partial h} = W_2^T \frac{\partial E}{\partial z_2}$$

Back-Propagation

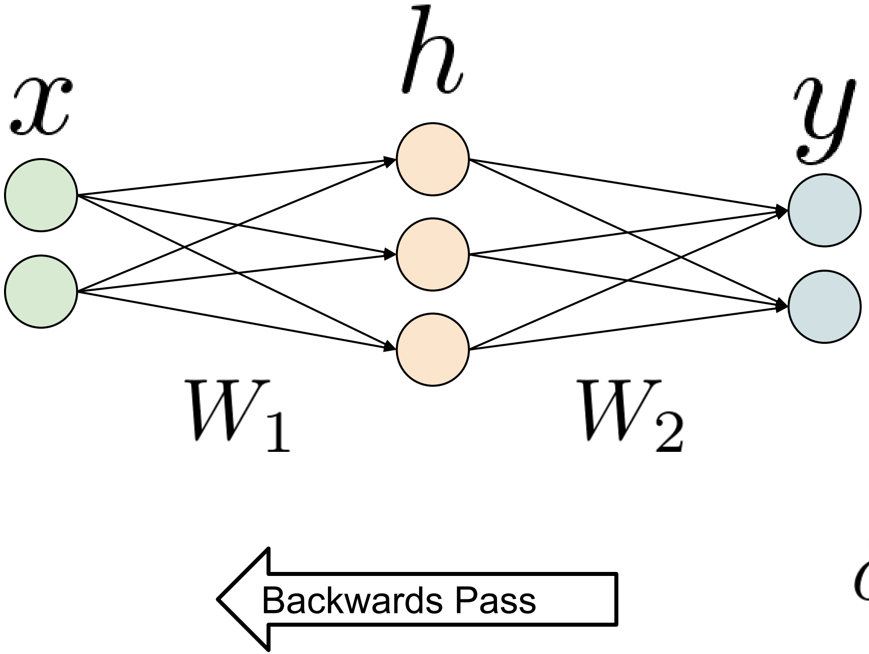


$$\frac{\partial E}{\partial z_2} = g'(z_2) \odot \frac{\partial E}{\partial y}$$

$$\frac{\partial E}{\partial h} = W_2^T \frac{\partial E}{\partial z_2}$$

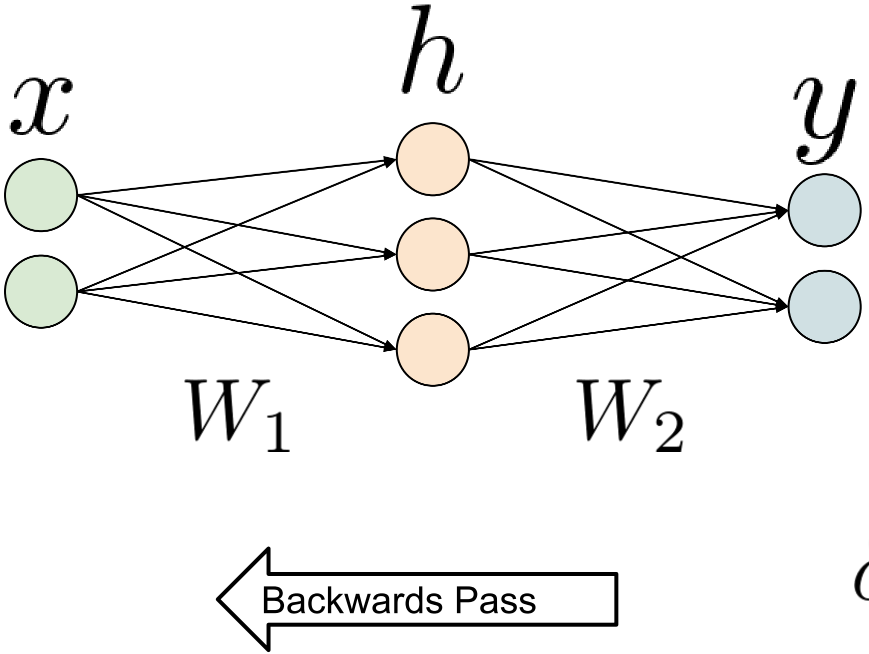
$$\delta h = (W_2^T \delta y) \odot g'(z_2)$$

The issue with back propagation



$$\delta h = (W_2^T \delta y) \odot g'(z_2)$$

The issue with back propagation



Backwards pass uses weights from the forward pass!

$$\delta h = (W_2^T \delta y) \odot g'(z_2)$$

Neuroplausible?

Bioplausible ideas

- Back-propagation is a relatively new player
  - No real evidence that “error” is propagated in the brain
- Contrastive Hebbian Learning
  - clamp output neurons at desired values; spread effects backwards
- Contrastive Divergence (in Restricted Boltzmann Machines)
  - make data more probable while making non-data less probable
- Target Propagation
  - Compute targets rather than gradients, at each layer
  - Propagate targets backwards
  - Target propagation relies on auto-encoders at each layer

Bioplausible ideas

- Back-propagation is a relatively new player
  - No real evidence that “error” is propagated in the brain
- Contrastive Hebbian Learning
  - clamp output neurons at desired values; spread effects backwards
- Contrastive Divergence (in Restricted Boltzmann Machines)
  - make data more probable while making non-data less probable
- Target Propagation
  - Compute targets rather than gradients, at each layer
  - Propagate targets backwards
  - Target propagation relies on auto-encoders at each layer

← Focus in today’s paper.

Random feedback weights support learning in deep neural networks

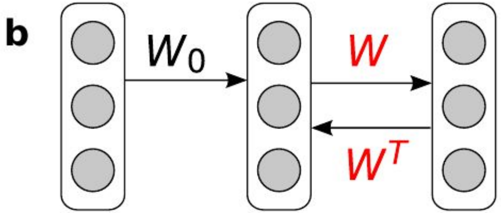
Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, Colin J. Akerman

Early Version: Arxiv 2014

Nature Communication 2016

From Backprop to Bio-plausible Feedback Learning

1. Networks in the brain compute via many layers of interconnected neurons
2. BP assigns blame to a neuron by **exactly** how it contributed to an error
   a. Requires neurons send each other precise information about large numbers of synaptic weights
   b. This implies that feedback is computed using knowledge of all the synaptic weights W

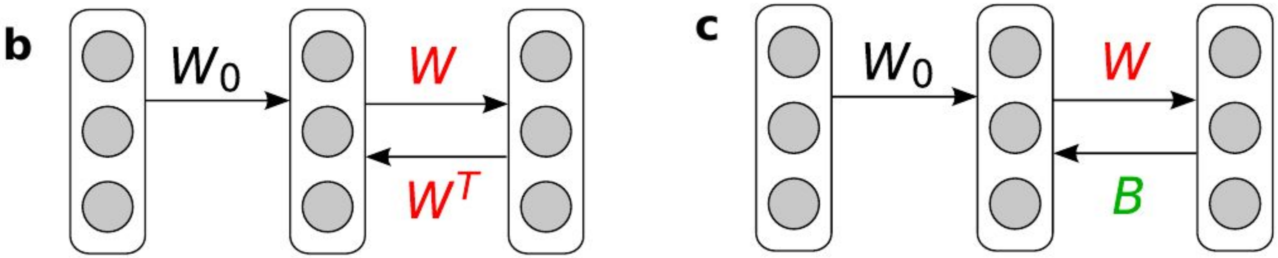


$$\Delta \mathbf{h}_{\mathrm{BP}} = W^T \mathbf{e}$$

3. Other difficulties regarding the biological plausibility (not the focus of this paper)
   a. Gradient?
   b. Spike?
   c. etc.

Random Feedback Weights Support

1. A new deep-learning algorithm that is
   a. Remove the assumption that upstream neuron knows matrix “W”
   b. Might be fast and accurate
   c. But much simpler, avoiding all transport of synaptic weight information.
2. **Feedback Alignment**’s basic idea:
   a. Use some random matrix B to replace transpose of synaptic weights W

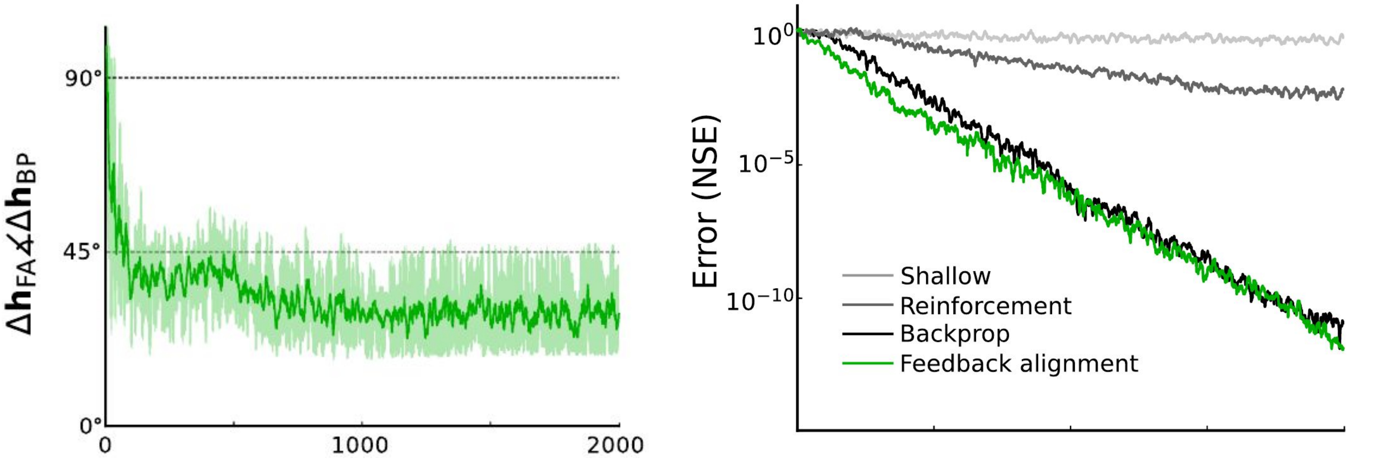


$$\Delta \mathbf{h}_{\mathrm{BP}} = W^T \mathbf{e}$$

$$\Delta \mathbf{h} = B\mathbf{e}$$

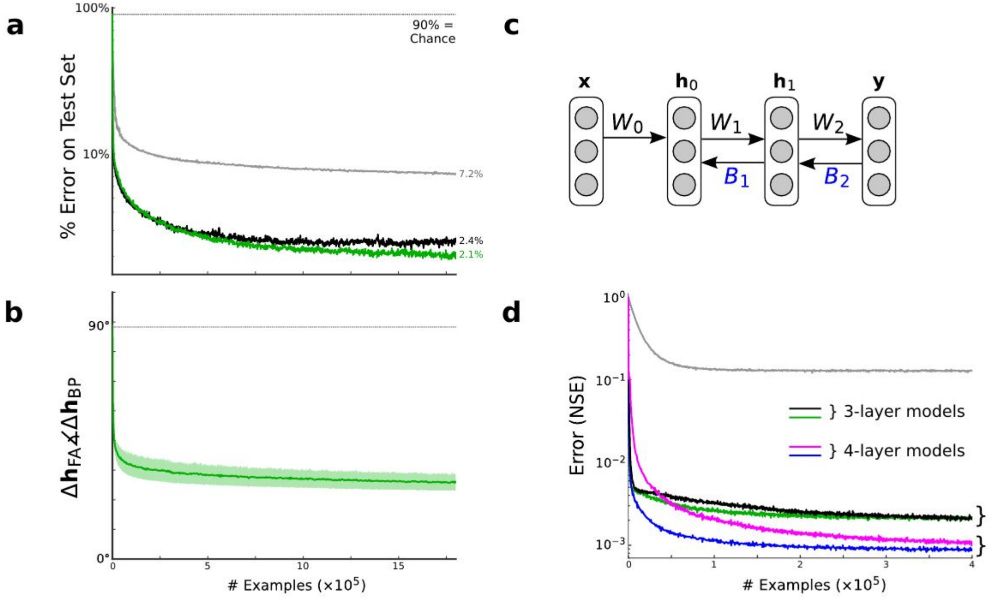
Random Feedback Weights Support

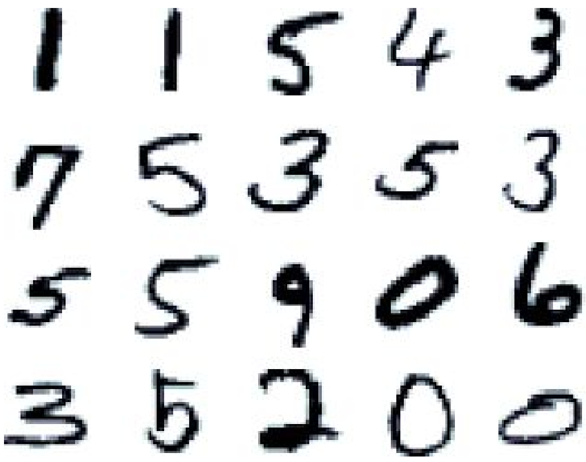
1. Insight behind Random feedback weights support learning
    a. We only need to get the direction roughly right during update $\mathbf{e}^T W B \mathbf{e} > 0$
    b. Even if the network doesn't have this property initially, it can acquire it through learning.
        i. The obvious option is to adjust B to make the equation true
        ii. During training, matrix W might gradually change to make the equation true
            1. can be done very simply, even with a fixed, random B



Results

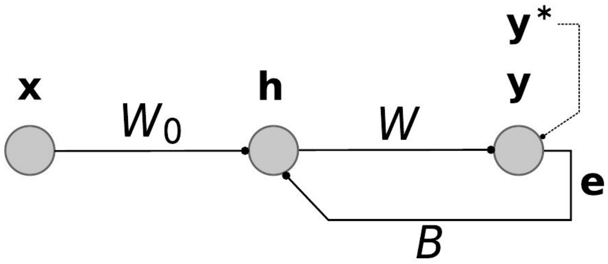
1. Feedback alignment learning also solves nonlinear benchmark classification problem (MNIST)

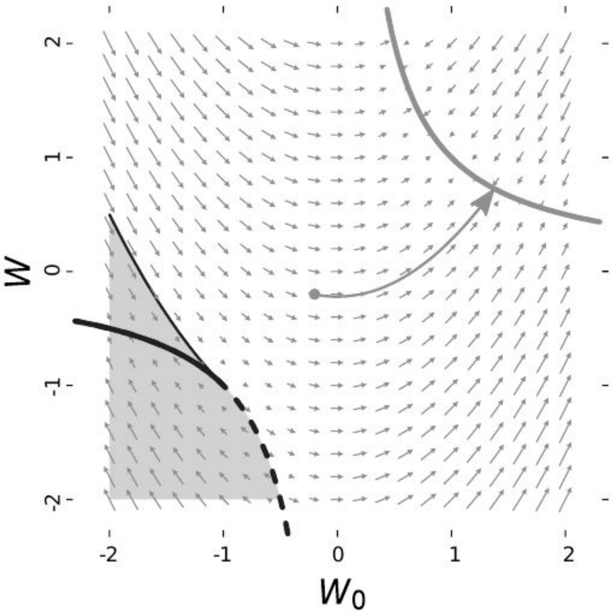




A Taste of Math

1. Why does feedback alignment work: a toy 2d example
   a. 1D Network with two neurons W0 and W1 (1 * 1 matrix)
   b. The feedback weight B is set to 1
   c. Model the mapping: y = x





A Taste of Math

1. The guaranteed convergence
   a. Simple network with one hidden layer (no activation function)

$$\boldsymbol{h} = A\boldsymbol{x} \qquad E := T - WA$$
$$\boldsymbol{y} = W\boldsymbol{h}$$

   b. Normalized input

$$\Delta W = \eta \left[E\boldsymbol{x}\boldsymbol{x}^T A^T \right] \qquad \Delta W = \eta E A^T \quad \dot{W} = EA^T$$
$$\Delta A = \eta \left[BE\boldsymbol{x}\boldsymbol{x}^T \right] \qquad \Delta A = \eta BE. \quad \dot{A} = BE.$$

   c. Could get a matrix relationship by integration (by setting W0, A0 = 0, we have C = 0)

$$BW + W^T B^T = AA^T + C$$

   d. Use Barbalat's lemma

$$V := \mathrm{tr}(BEE^T B^T). \qquad \dot{V} \to 0.$$

A Taste of Math

1. Let’s continue
    a. Both of the addends will be zero

    $$\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{tr}(BEE^TB^T) = -2\mathrm{tr}(BEA^TAE^TB^T) - \mathrm{tr}(A^TBEE^TB^TA) \leq 0$$

    b. Many more properties follow by doing simple linear algebra (note that we assume B has Moore-Penrose pseudo-inverse)

    $$BEA^T = 0. \qquad EA^T = 0.$$

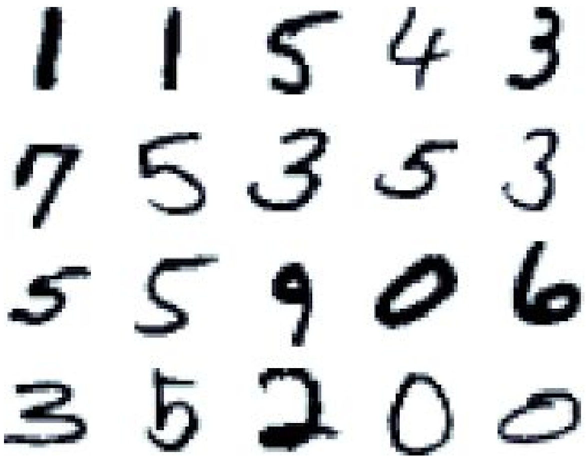
    $$\mathrm{tr}(EE^T) = ||E|| = 0$$

Analytic result suggests more

1. When the weights begin near 0, feedback alignment encourages W to act like a local pseudoinverse of B around the error manifold.
2. This fact is important because if B were exactly the Moore-Penrose pseudoinverse of W, then the network would be performing Gauss-Newton optimization
3. Mathematically very complicated and need strong assumption about the network, see the supplementary materials of the paper.

Code Reproduction

- Available at: https://github.com/xuexue/randombp
- MNIST
- 3-layer (1-hidden-layer) network; 100 hidden units
    - Smaller model than in the paper



Reproducing the results

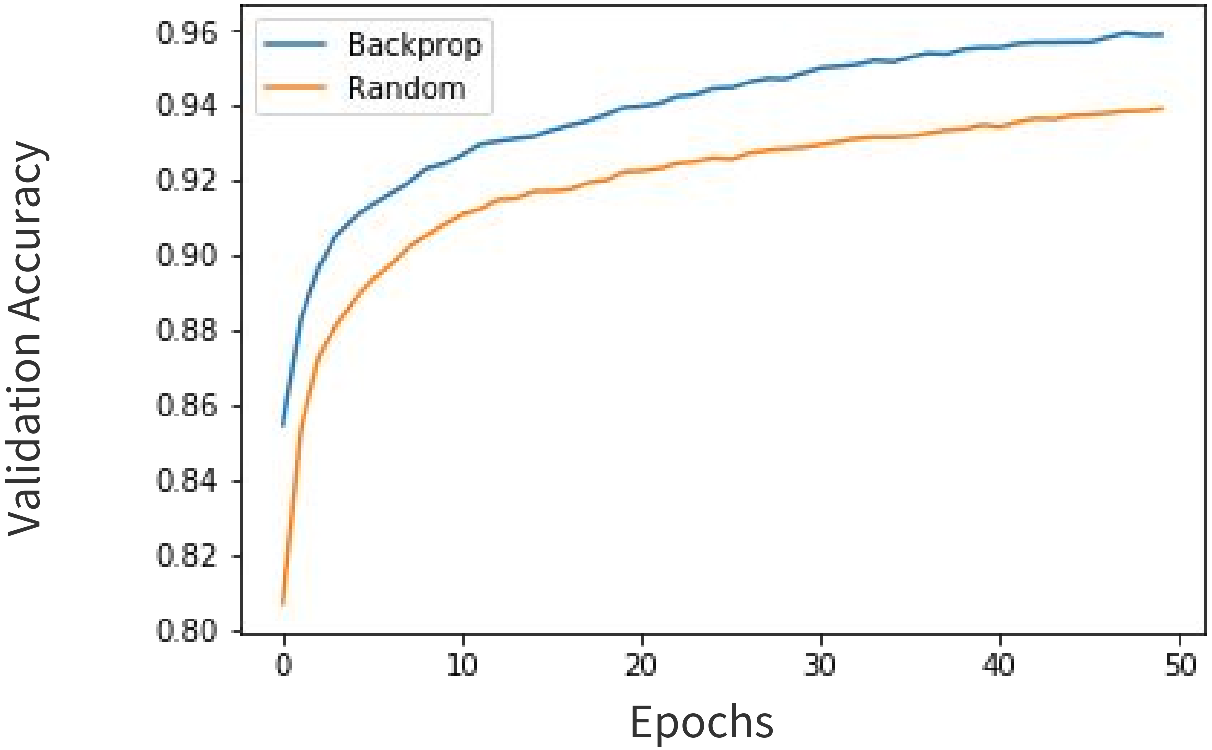
- Sensitive to:
  - Architecture (small-ish)
  - Starting weights (can’t be too small, can’t be too large, zeros DON’T work!)
  - Learning rate (need high-ish learning rate)
  - Weight decay (for direct feedback)
- Lots of configurations refuses to train
- There were times when network began getting better accuracy, then loses (!!) accuracy

Demo Code.

Only difference between standard backprop.

```
class RandomFeedbackNet(BackPropNet):
  def define_train_step(self, num_hidden):
    # define backward weights
    with tf.variable_scope(self.scope):
      b2 = wi("b2", [10, num_hidden])
    # training: derivative w.r.t. activations
    ypred_grad = tf.gradients(self.cross_entropy, self.ypred)[0]
    z2_grad = tf.gradients(self.cross_entropy, self.z2)[0]
    h_grad = tf.matmul(z2_grad, b2)
    z1_grad = tf.multiply(tf.gradients(self.h, self.z1)[0], h_grad)
    # training: derivative w.r.t. weights
    self.w2_grad = tf.reduce_sum(
            tf.multiply(tf.expand_dims(self.h, 2),
                        tf.expand_dims(z2_grad, 1)), [0])
    self.d2_grad = tf.reduce_sum(z2_grad, [0])
    self.w1_grad = tf.reduce_sum(
            tf.multiply(tf.expand_dims(self.x, 2),
                        tf.expand_dims(z1_grad, 1)), [0])
    self.d1_grad = tf.reduce_sum(z1_grad, [0])
    # training: assign weights
    self.train_step= [
        tf.assign(self.w2, self.w2 - self.alpha * self.w2_grad - self.decay),
        tf.assign(self.w1, self.w1 - self.alpha * self.w1_grad - self.decay),
        tf.assign(self.d2, self.d2 - self.alpha * self.d2_grad - self.decay),
        tf.assign(self.d1, self.d1 - self.alpha * self.d1_grad - self.decay),
      ]
```

Results (Validation Accuracy)

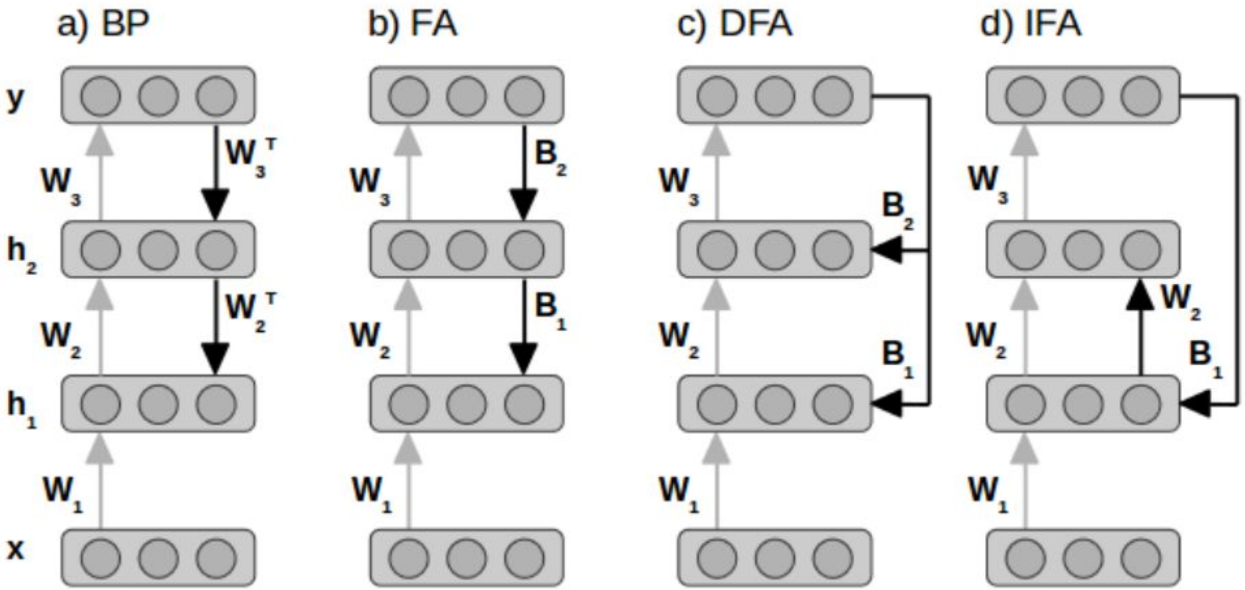


Direct Feedback Alignment Provides Learning in Deep Neural Networks

Arild Nøkland

NIPS 2016

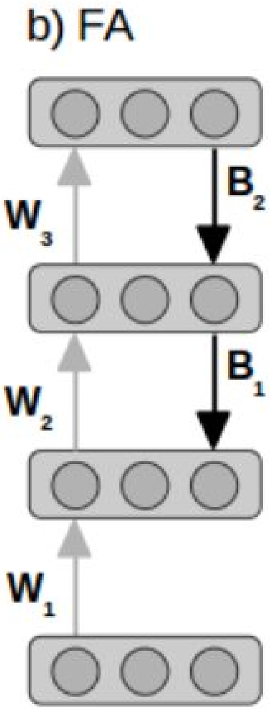
Direct Feedback Alignment



A comparison between Back Propagation (BP), Feedback Alignment (FA), Direct Feedback Alignment (DFA), and Indirect Feedback Alignment (IFA)

Original Feedback Alignment

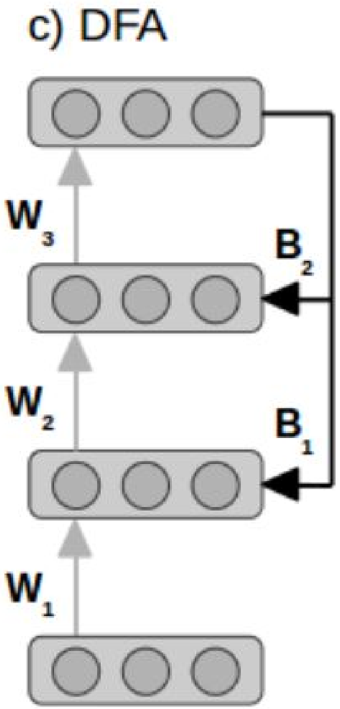
FA $\delta a_2 = (B_2 e) \odot f'(a_2),\ \delta a_1 = (B_1 \delta a_2) \odot f'(a_1)$



Direct Feedback Alignment

FA $\delta a_2 = (B_2 e) \odot f'(a_2),\ \delta a_1 = (B_1 \delta a_2) \odot f'(a_1)$

DFA $\delta a_2 = (B_2 e) \odot f'(a_2),\ \delta a_1 = (B_1 e) \odot f'(a_1)$

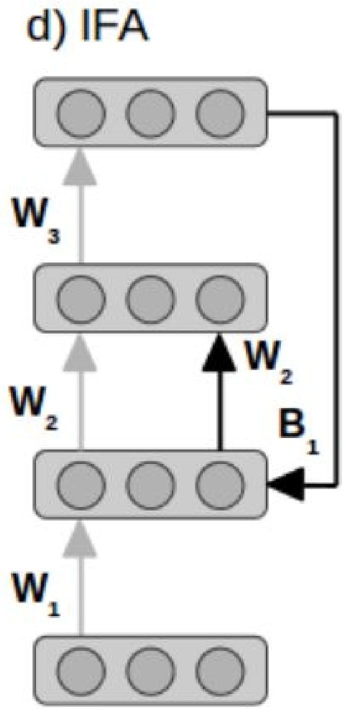


Indirect Feedback Alignment

FA $\delta a_2 = (B_2 e) \odot f'(a_2), \ \delta a_1 = (B_1 \delta a_2) \odot f'(a_1)$

DFA $\delta a_2 = (B_2 e) \odot f'(a_2), \ \delta a_1 = (B_1 e) \odot f'(a_1)$

IFA $\delta a_2 = (W_2 \delta a_1) \odot f'(a_2), \ \delta a_1 = (B_1 e) \odot f'(a_1)$



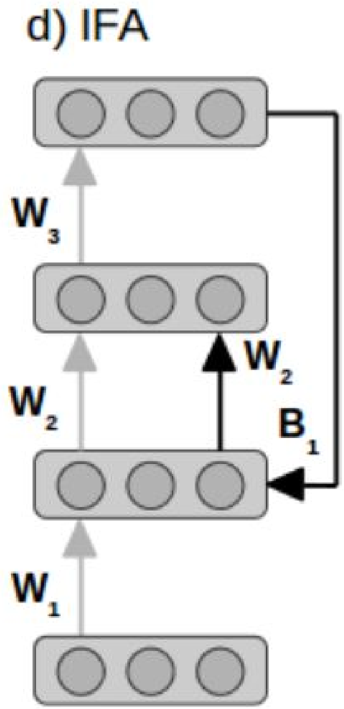
Feedback Alignment

FA $\delta a_2 = (B_2 e) \odot f'(a_2),\ \delta a_1 = (B_1 \delta a_2) \odot f'(a_1)$

DFA $\delta a_2 = (B_2 e) \odot f'(a_2),\ \delta a_1 = (B_1 e) \odot f'(a_1)$

IFA $\delta a_2 = (W_2 \delta a_1) \odot f'(a_2),\ \delta a_1 = (B_1 e) \odot f'(a_1)$

Update $\delta W_1 = -\delta a_1 x^T,\ \delta W_2 = -\delta a_2 h_1^T,\ \delta W_3 = -e h_2^T$



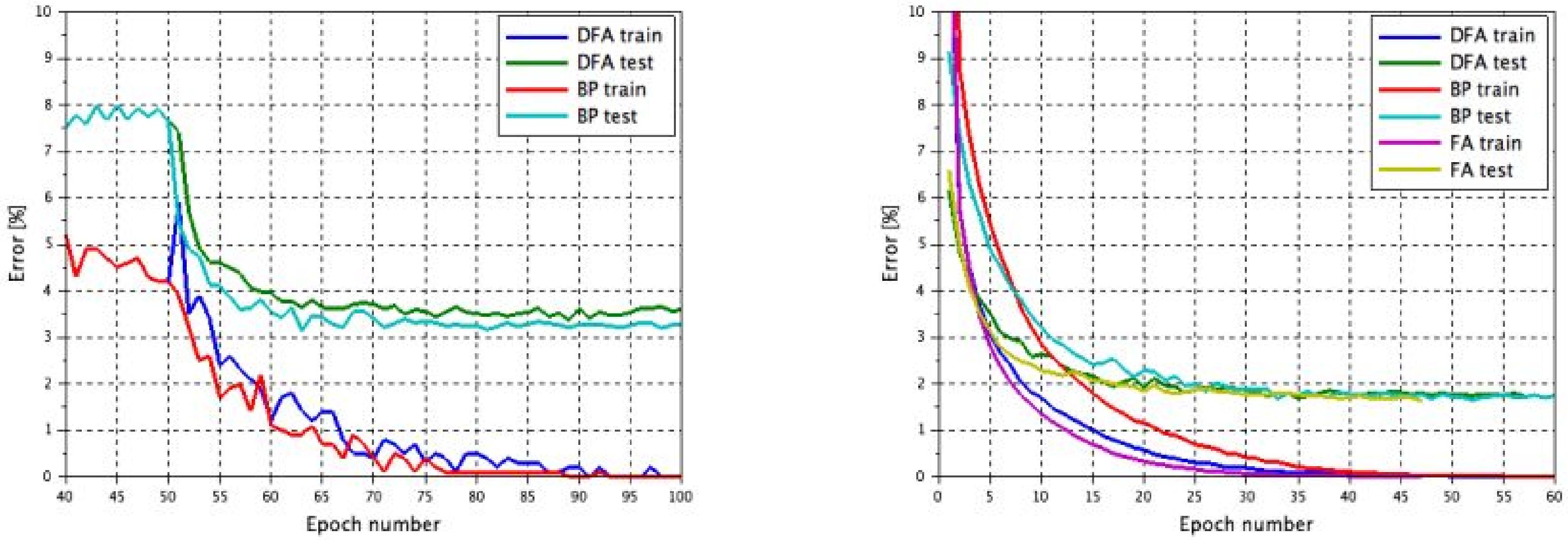
Theoretical Results

- In the FA paper, the authors proved that we can achieve zero training error with FA under the following assumption:
  - Network is **linear with one hidden layer**.
  - Input data have **zero mean and unit variance**.
  - The feedback weight matrix has Moore-Penrose **pseudo-inverse**.
  - The forward weights are **initialized to zero**.
  - The **output layer weights are adapted**.
- However, it is unclear how the training error can approach zero with several non-linear layers.
- This paper gives new theoretical insight with less assumption of the network topology, under the assumption of **constant update direction**.

Theoretical Results

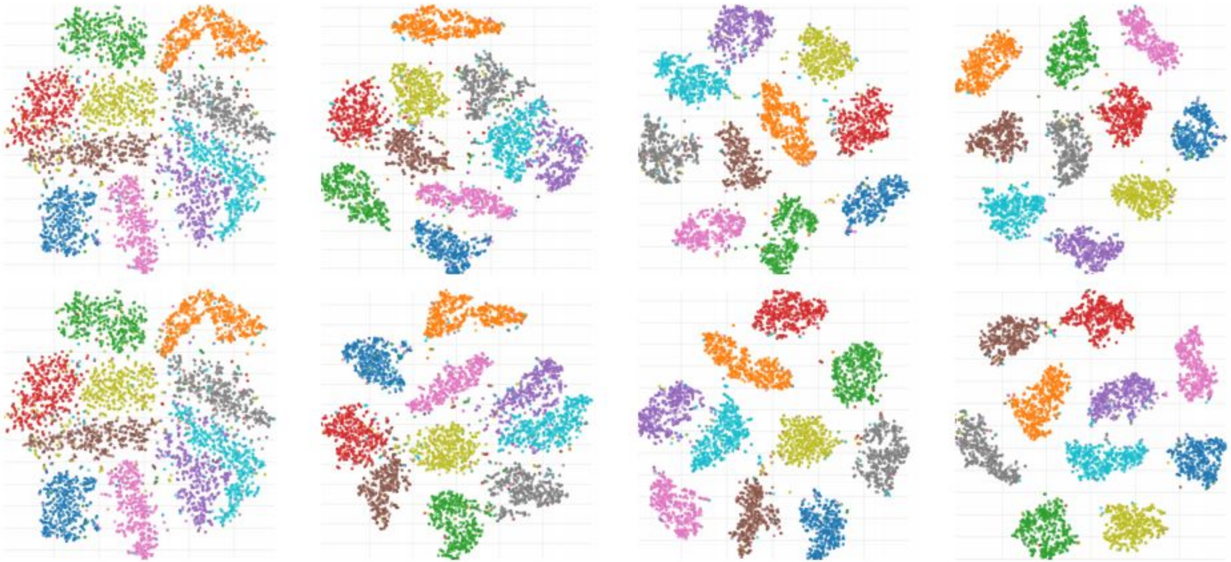
- This paper generalizes previous FA results by considering **two consecutive layers**.
- For any layer k and k+1, δh_k will **end up within 90 degrees of cosine angle with the back-propagated gradient c_k**, and δh_{k+1} with c_{k+1}.
- Although we assume
-
- | δh_k is constant for all data points, it can still **a function of the parameters**. The theorem does not provide convergence guarantee (provided in the original FA paper).

Experiments



Training curve of a two layer network on MNIST, with fixed first hidden layer (left), and full network (right).

Experiments



Upper: Hidden activation of BP network. Lower: Hidden activation of DFA network

Experiments

MODEL	BP	FA	DFA
7x240 Tanh	$2.16 \pm 0.13\%$	$2.20 \pm 0.13\%$ (0.02%)	$2.32 \pm 0.15\%$ (0.03%)
100x240 Tanh			$3.92 \pm 0.09\%$ (0.12%)
1x800 Tanh	$1.59 \pm 0.04\%$	$1.68 \pm 0.05\%$	$1.68 \pm 0.05\%$
2x800 Tanh	$1.60 \pm 0.06\%$	$1.64 \pm 0.03\%$	$1.74 \pm 0.08\%$
3x800 Tanh	$1.75 \pm 0.05\%$	$1.66 \pm 0.09\%$	$1.70 \pm 0.04\%$
4x800 Tanh	$1.92 \pm 0.11\%$	$1.70 \pm 0.04\%$	$1.83 \pm 0.07\%$ (0.02%)
2x800 Logistic	$1.67 \pm 0.03\%$	$1.82 \pm 0.10\%$	$1.75 \pm 0.04\%$
2x800 ReLU	$1.48 \pm 0.06\%$	$1.74 \pm 0.10\%$	$1.70 \pm 0.06\%$
2x800 Tanh + DO	$1.26 \pm 0.03\%$ (0.18%)	$1.53 \pm 0.03\%$ (0.18%)	$1.45 \pm 0.07\%$ (0.24%)
2x800 Tanh + ADV	$1.01 \pm 0.08\%$	$1.14 \pm 0.03\%$	$1.02 \pm 0.05\%$ (0.12%)

MNIST performance of BP, FA, and DFA

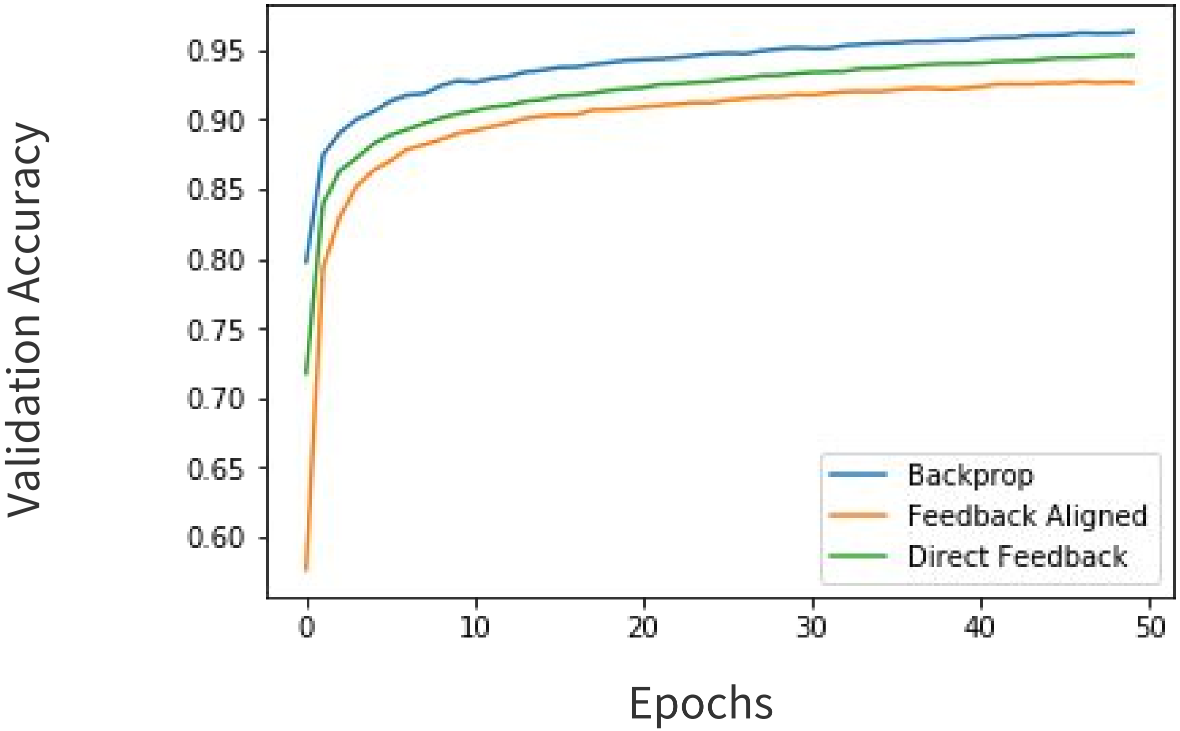
Reproducing the Results

- Smaller model than in the paper
- MNIST
- 4-layer (2-hidden-layer networK)
  - 784 → 200 → 100 → 10
- Weight decay essential
- There were times when network began getting better accuracy, then loses (!!) accuracy

Demo Code: Direct Feedback

```
def define_train_step(self, num_hidden):
  with tf.variable_scope(self.scope):
    n1, n2 = num_hidden
    b2 = wi("b2", [10, n1]) # <--- SUBTLE DIFFERENCE
    b3 = wi("b3", [10, n2])
  # training: derivative w.r.t. activations
  ypred_grad = tf.gradients(self.cross_entropy, self.ypred)[0]
  z3_grad = tf.gradients(self.cross_entropy, self.z3)[0]
  h2_grad = tf.matmul(z3_grad, b3)
  z2_grad = tf.multiply(tf.gradients(self.h2, self.z2)[0], h2_grad)
  h1_grad = tf.matmul(z3_grad, b2) # <--- SUBTLE DIFFERENCE HERE
  z1_grad = tf.multiply(tf.gradients(self.h1, self.z1)[0], h1_grad)
  ...
```

Results (Validation Accuracy)



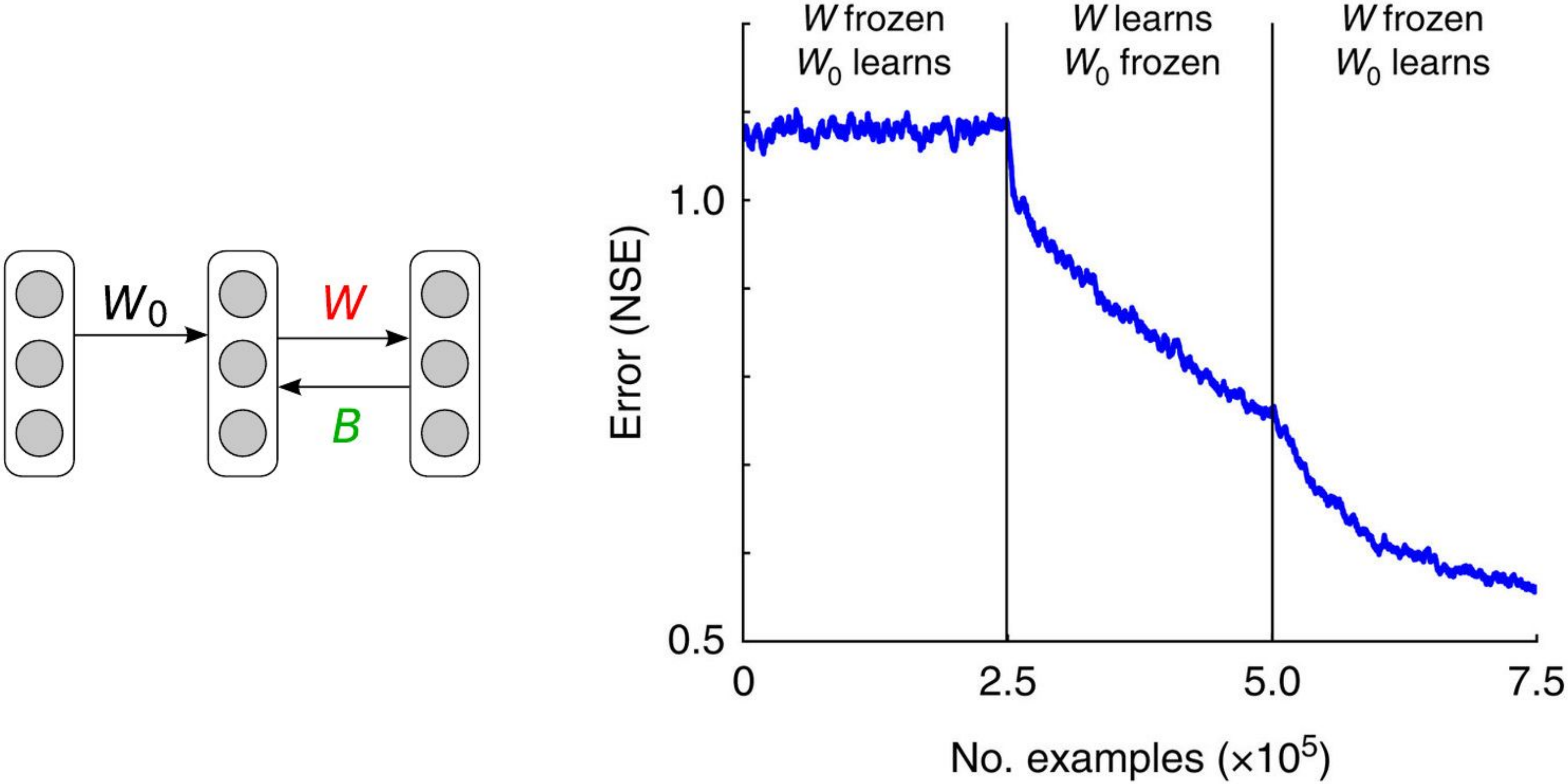
Opinions

Positives

- FA is a thorough exploration using **asymmetric connections**.
- FA guaranteed **convergence under some assumption** of the network.
- Shown in experiments that we can train **very deep 100-layer network** with DFA, whereas BP and FA suffers from gradient vanishing.
- DFA **replaces the reciprocal feedback assumption** with a single feedback layer.
- The relaxed version of DFA, IFA, can be viewed as **skip connections on the feedback path**, which opens up more freedom on the actual form of feedback connections, compared to the original FA.
- One of the first exploration of error-driven learning using **directly connected feedback path**.
- Can be used to send error signals **skipping non-differentiable layers**.

Critiques

- DFA assumes that there is a **global feedback path**, which may be biologically implausible since the single feedback layer need to travel long physical distance.
- Both FA and DFA leverages the principle of feedback alignment to drive the error signal. Due to the alignment stage, **a layer cannot learn before its upper layers are roughly "aligned."** This could also be biologically implausible.
- FA and DFA are presented as **less powerful optimization methods**. A more impactful yet biologically plausible direction could be replace BP with a learning algorithm with **better generalization performance**.
- FA and DFA rely on **synchronous updates**: to update the weights at a layer, we need to fix the activation of the layer below.
- Theoretical results on the negative descending direction is **weak**.



W0
W
B
W frozen
W0 learns
W learns
W0 frozen
W frozen
W0 learns
1.0
0.5
Error (NSE)
0
2.5
5.0
7.5
No. examples (×10^5)

Our reproduction