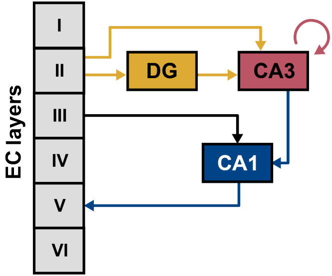
A biologically-plausible learning rule based on pseudoinverse feedback connections

Mia Cameron

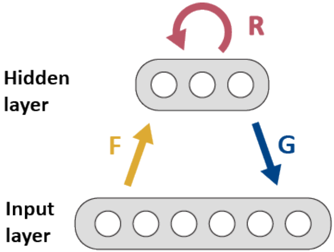
10 - 18 - 2024

Project background: Recurrent Autoencoders

- Task-constrained recurrent autoencoders are successful in recapitulating features of real neural circuits.
- Backpropagation is biologically unrealistic, since it requires access to non-local information.
- **Can these dynamics be learned with a biologically-plausible learning rule?**



(a) Hippocampal connectivity circuit (Chen et. al 2024)



(b) Modelling with a simple, recurrent autoencoder architecture

Local learning of a single, encoder-decoder layer

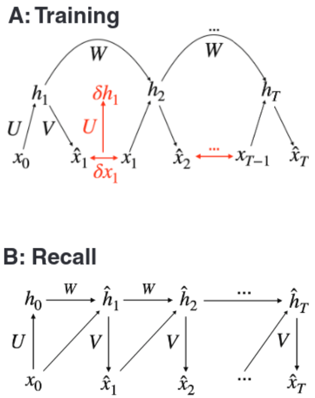
The Recirculation algorithm can be used to train a single layer of encoder (U) and decoder (V) weights.

Learning dynamics

$$h = \sigma(Ux)$$
$$\hat{x} = \lambda x + (1-\lambda)Vh$$
$$\hat{h} = \lambda h + (1-\lambda)\sigma(U\hat{x})$$

$$\Delta V = -(x-\hat{x})h^T$$
$$\Delta U = -(h-\hat{h})x^T$$

(Recurrent weights learned with a simple, asymmetric Hebbian rule)



Local learning of a single, encoder-decoder layer

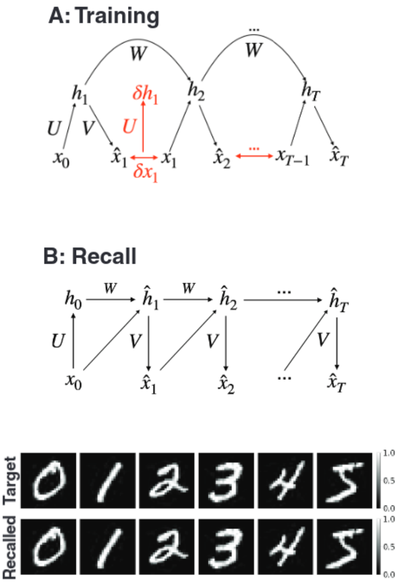
The Recirculation algorithm can be used to train a single layer of encoder (U) and decoder (V) weights.

Learning dynamics

$$h = \sigma(Ux)$$
$$\hat{x} = \lambda x + (1-\lambda)Vh$$
$$\hat{h} = \lambda h + (1-\lambda)\sigma(U\hat{x})$$

$$\Delta V = -(x-\hat{x})h^T$$
$$\Delta U = -(h-\hat{h})x^T$$

(Recurrent weights learned with a simple, asymmetric Hebbian rule)



What is Recirculation really doing?

- To simplify, we first removed the nonlinearity.

What is Recirculation really doing?

- To simplify, we first removed the nonlinearity.
- With random, mean-zero inputs, linearized Recirculation is equivalent to a well-known iterative method for computing the **pseudoinverse.**

What is Recirculation really doing?

- To simplify, we first removed the nonlinearity.
- With random, mean-zero inputs, linearized Recirculation is equivalent to a well-known iterative method for computing the **pseudoinverse.**

Ben-Israel and Cohen (1966) Iterative Pseudoinverse Method

Starting from a matrix X_0 which satisfies $(X_0A)^T = X_0A$, a sequence is generated by:

$$X_{t+1} - X_t = X_t - X_tAX_t$$

where X_t converges to A^+.

What is Recirculation really doing?

- To simplify, we first removed the nonlinearity.
- With random, mean-zero inputs, linearized Recirculation is equivalent to a well-known iterative method for computing the **pseudoinverse.**

Ben-Israel and Cohen (1966) Iterative Pseudoinverse Method

Starting from a matrix X_0 which satisfies $(X_0 A)^T = X_0 A$, a sequence is generated by:

$$X_{t+1} - X_t = X_t - X_t A X_t$$

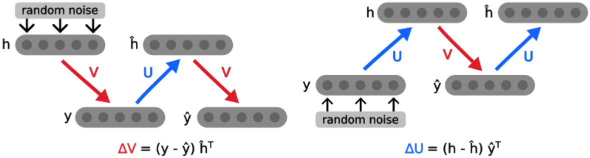
where X_t converges to A^+.

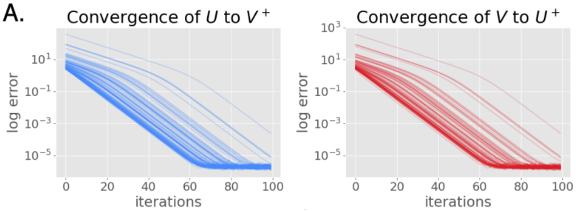
For all y in the row space of A, the pseudoinverse is the unique matrix in which:

$$A^+(Ay) = y$$

What is Recirculation really doing?

- Concurrent training of U and V converges to the pseudoinverse.
- Random initialization with a low condition-number matrix.
- Physically, this could be implemented with random noise inputs at each layer.





Extending to multi-layered networks

- Can we use these locally-learned, pseudoinverse feedback connnections to "backpropagate" error from the final layer of a multi-layered network?

Extending to multi-layered networks

- Can we use these locally-learned, pseudoinverse feedback connnections to "backpropagate" error from the final layer of a multi-layered network?
- Yes! If each layer is full-row rank, the product of the pseudoinverse feedback connections forms a **left-reciprocal.**

Extending to multi-layered networks

- Can we use these locally-learned, pseudoinverse feedback connnections to "backpropagate" error from the final layer of a multi-layered network?
- Yes! If each layer is full-row rank, the product of the pseudoinverse feedback connections forms a **left-reciprocal.**

Hildebrandt-Graves Theorem

If $F : (X_0, B_r(y_0)) \to \mathbb{R}^m$ is a vector function with Jacobian A, with left reciprocal T, Then there exists a solution y^* which for every $x \in X_0$ satisfies

$$TF(x, y^*) = 0$$

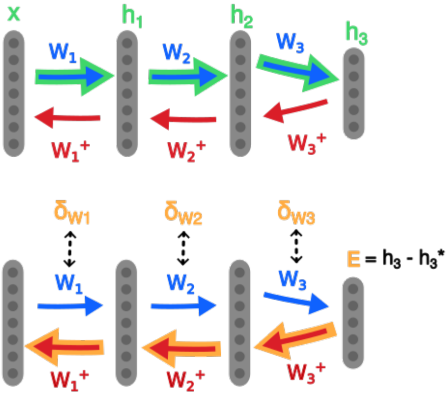
which can be obtained using the iteration:

$$y_{t+1} = y_t - TF(x, y_t)$$

(under certain conditions)

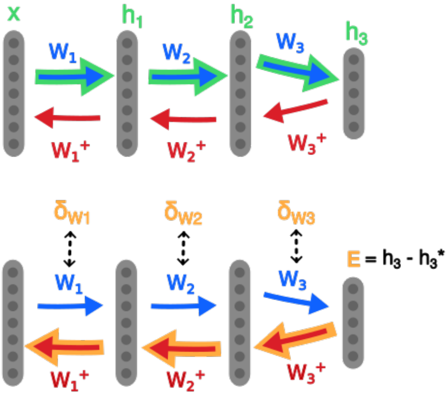
Extending to multi-layered networks

We can use the left-reciprocal in-place of the weight transpose, and use a similar recursion to backpropagation.



Extending to multi-layered networks

We can use the left-reciprocal in-place of the weight transpose, and use a similar recursion to backpropagation.



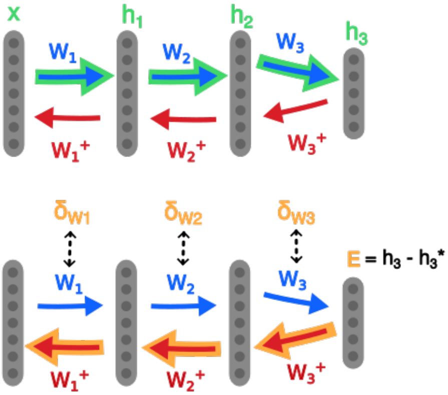
Recursion for the Jacobian:

$$J^E_{W_l} = (a_{l-1} \otimes J^E_{h_l})$$

$$J^E_{h_l} = J^E_{h_{l+1}} \mathcal{D}_\sigma W_{l+1}$$

Extending to multi-layered networks

We can use the left-reciprocal in-place of the weight transpose, and use a similar recursion to backpropagation.



Recursion for the Jacobian:

$$J^E_{W_l} = (a_{l-1} \otimes J^E_{h_l})$$

$$J^E_{h_l} = J^E_{h_{l+1}} \mathcal{D}_\sigma W_{l+1}$$

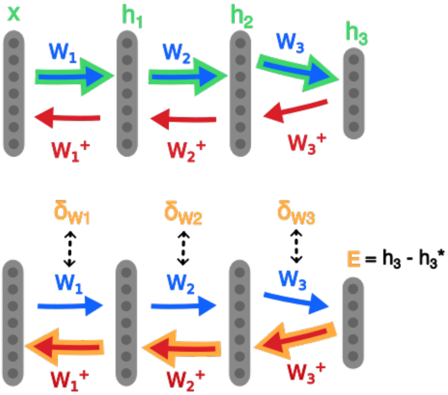
Recursion for the left reciprocal:

$$\mathcal{B}_l = \left(a^+_{l-1} \otimes B_l\right)$$

$$B_l = W^+_{l+1} \mathcal{D}^+_\sigma B_{l+1}$$

Extending to multi-layered networks

We can use the left-reciprocal in-place of the weight transpose, and use a similar recursion to backpropagation.



Recursion for the Jacobian:

$$J^E_{W_l} = (a_{l-1} \otimes J^E_{h_l})$$

$$J^E_{h_l} = J^E_{h_{l+1}} \mathcal{D}_\sigma W_{l+1}$$

Recursion for the left reciprocal:

$$\mathcal{B}_l = \left(a^+_{l-1} \otimes B_l\right)$$

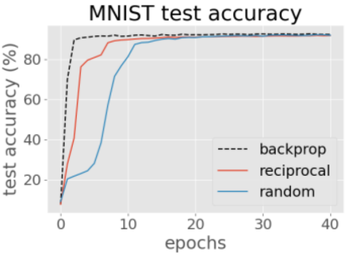
$$B_l = W^+_{l+1} \mathcal{D}^+_\sigma B_{l+1}$$

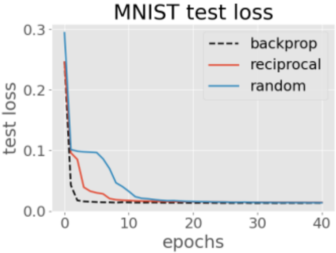
Weight update:

$$\delta^t_{W_l} \propto B_l e(x, W_l) a^T_{l-1}$$

$$W^{t+1}_l = W^t_l - \lambda \delta^t_{W_l}$$

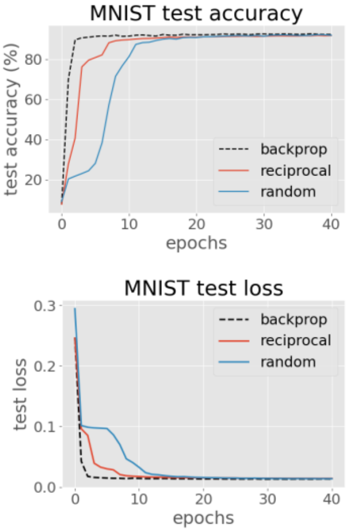
Simulation on machine learning tasks





- Fully connected, feedforward network with 5 hidden layers.
- Same asymptotic error as backpropagation.
- Fewer iterations needed than Random Feedback Alignment (Lillicrap, et al. 2016)

Simulation on machine learning tasks



- Fully connected, feedforward network with 5 hidden layers.
- Same asymptotic error as backpropagation.
- Fewer iterations needed than Random Feedback Alignment (Lillicrap, et al. 2016)
- Local learning of pseudoinverse feedback connections with random noise, and top-down error propagation can be synchronized in two separate phases (wake and sleep).

Related work

- Learning optimal feedback connections with random, mean-zero noise in a separate "sleep" phase. (Akrout et al., 2019)

Related work

- Learning optimal feedback connections with random, mean-zero noise in a separate "sleep" phase. (Akrout et al., 2019)
- Unregularized linear autoencoders naturally learn the pseudoinverse at each layer, when trained using gradient descent on reconstruction loss. (Kunin et al., 2019)

Related work

- Learning optimal feedback connections with random, mean-zero noise in a separate "sleep" phase. (Akrout et al., 2019)
- Unregularized linear autoencoders naturally learn the pseudoinverse at each layer, when trained using gradient descent on reconstruction loss. (Kunin et al., 2019)
- Using the pseudoinverse of the whole network's Jacobian is equivalent to Gauss-Newton optimization (Martens et al., 2015; Botev et al., 2017)

Related work

- Learning optimal feedback connections with random, mean-zero noise in a separate "sleep" phase. (Akrout et al., 2019)
- Unregularized linear autoencoders naturally learn the pseudoinverse at each layer, when trained using gradient descent on reconstruction loss. (Kunin et al., 2019)
- Using the pseudoinverse of the whole network's Jacobian is equivalent to Gauss-Newton optimization (Martens et al., 2015; Botev et al., 2017)
    - Gauss-Newton optimization is a second-order method, which outperforms gradient descent.

Related work

- Learning optimal feedback connections with random, mean-zero noise in a separate "sleep" phase. (Akrout et al., 2019)
- Unregularized linear autoencoders naturally learn the pseudoinverse at each layer, when trained using gradient descent on reconstruction loss. (Kunin et al., 2019)
- Using the pseudoinverse of the whole network's Jacobian is equivalent to Gauss-Newton optimization (Martens et al., 2015; Botev et al., 2017)
  - Gauss-Newton optimization is a second-order method, which outperforms gradient descent.
  - However, we are composing a left-reciprocal with layer-wise pseudoinverses, instead of taking the pseudoinverse of the whole network.