More on Back Propagation:
1) Optimization Algorithms and Weight Initialization
2) Toward Biologically Plausible Implementations

Psychology 209
January 24, 2019

Optimization Algorithms

- Gradient descent variants
    - Batch gradient descent
    - Stochastic gradient descent
    - Mini-batch gradient descent

- Challenges

- Gradient descent optimization algorithms
    - Momentum
    - Nesterov accelerated gradient
    - Adagrad
    - Adadelta
    - RMSprop
    - Adam
    - AdaMax
    - Nadam
    - AMSGrad
- Visualization of algorithms
- Which optimizer to choose?

- Parallelizing and distributing SGD
    - Hogwild!
    - Downpour SGD
    - Delay-tolerant Algorithms for SGD
    - TensorFlow
    - Elastic Averaging SGD

- Additional strategies for optimizing SGD
    - Shuffling and Curriculum Learning
    - Batch normalization
    - Early Stopping
    - Gradient noise

Some of the Algorithms

Vector of weights + biases

Vector of dL/dw

- Batch Gradient Descent $\theta = \theta - \eta \cdot \nabla_\theta J(\theta).$
- Momentum $v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \qquad \theta = \theta - v_t$
- AdaGrad $g_{t,i} = \nabla_\theta J(\theta_{t,i}). \quad \theta_{t+1,i} = \theta_{t,i} - \dfrac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}.$

Sum of Squares of $g_{t',I}$ up to time t

- RMSprop

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

g_t^2 is the vector of squares of the $g_{i,t}$

Corresponds to $RMS[g]_t$

- AdaDelta

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$
$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

RMS of previous change in the parameters

Adam

- Two running averages:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

- Update depends on momentum normalized by variance:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t.$$

The authors propose default values of 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ϵ.

Neural network activation Functions

- All receiving units use a 'net input', here called n_r, given by

$$n_r = \sum_s w_{rs}\, a_s + b_r$$

- This is then used as the basis of the unit's activation using an 'activation function,' usually one of the following:
  - Linear: $a_r = n_r$
  - Logistic: $a_r = 1/(1 + e^{-n_r})$
  - Tanh: $a_r = (e^{n_r} - e^{-n_r})/(e^{n_r} + e^{-n_r})$
  - Relu: $a_r = [n_r]^+$, i.e., 0 if $n_r < 0$ else n_r

The algorithms in action

Weight Initialization

- To small: Learning is too slow
- Too big: you jam units into the non-linear range, loose the gradient signal
- Just right?
- One approach: consider the fan in to each receiving unit:
- For example, 'He Initialization' is as follows:

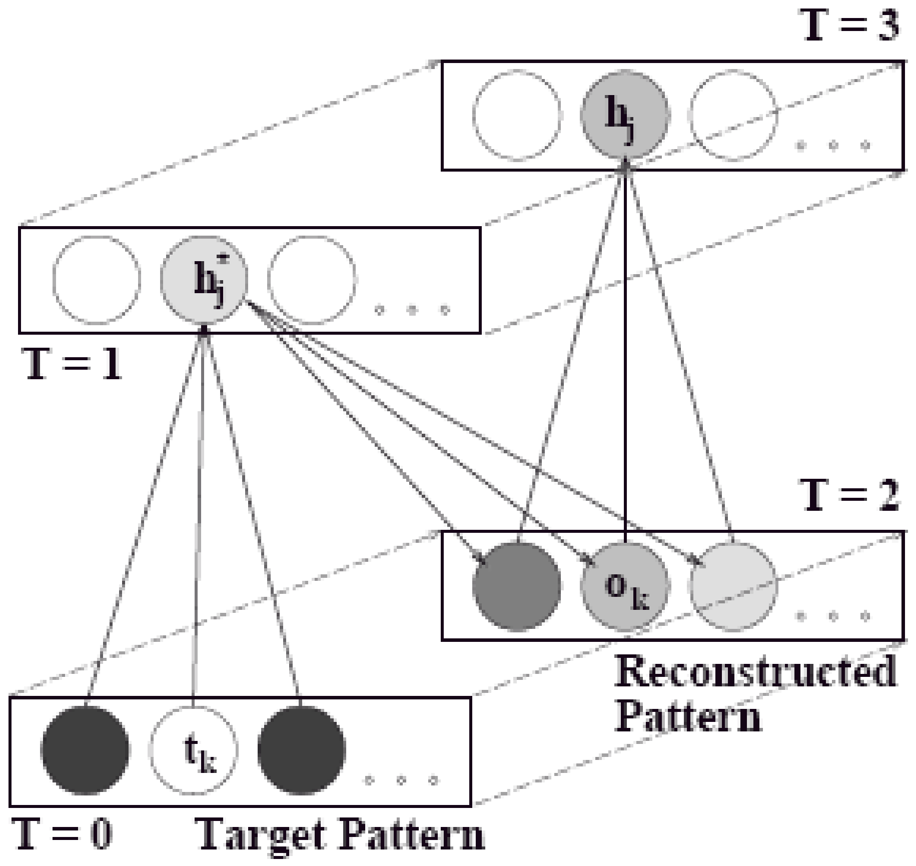
$$W \sim N\left(0, {}^{1}/_{\sqrt{n_{in}}}\right)$$

Alternatives include:

$$W_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right] \quad W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

See appendix slide for details if interested.

Recirculation Algorith



Assuming symmetric connections:

$$\frac{\partial E}{\partial h_j} = -\left(\sum_k t_k w_{jk} - \sum_k o_k w_{jk}\right)$$

$$\begin{aligned}\frac{\partial E}{\partial w_{jk}} &= -(\eta_j^* - \eta_j)\sigma'(\eta_j)t_k \\ &\approx -(h_j^* - h_j)t_k\end{aligned}$$

Hinton, G. & McClelland, J. in *Neural Information Processing Systems*. (ed. Anderson, D.) 358–366, (1988).

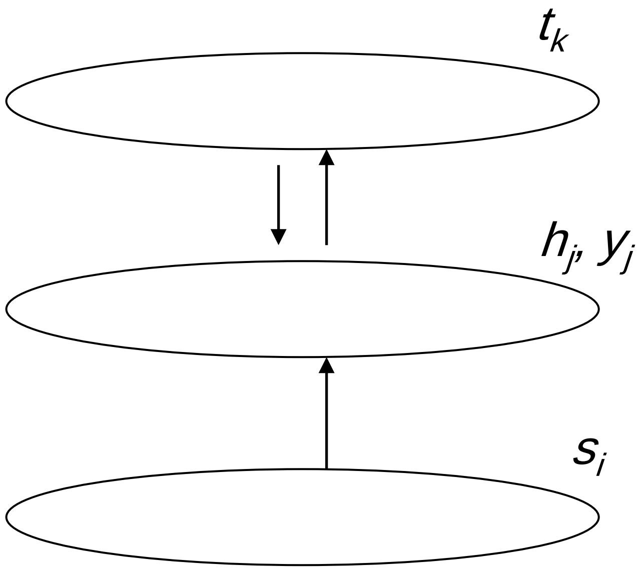
Generalized Recirculation

Present input, feed activation forward, compute output, let it feed back and let the hidden state settle to a fixed point.

Then clamp both input and output units into desired state, and settle again.*

$$
\begin{aligned}
\frac{dy_j^+}{dt} &= -y_j^+ + \sum_k w_{kj} t_k \\
\frac{dy_j^-}{dt} &= -y_j^- + \sum_k w_{kj} \sigma(\eta_k) \\
\frac{\partial E}{\partial h_j} &\approx \sigma'(\eta_j)(y_j^{\infty+} - y_j^{\infty-}) \\
&\approx h_j^+ - h_j^-
\end{aligned}
$$

$$\frac{1}{\epsilon}\Delta w_{ij} = a_i^-(a_j^+ - a_j^-)$$

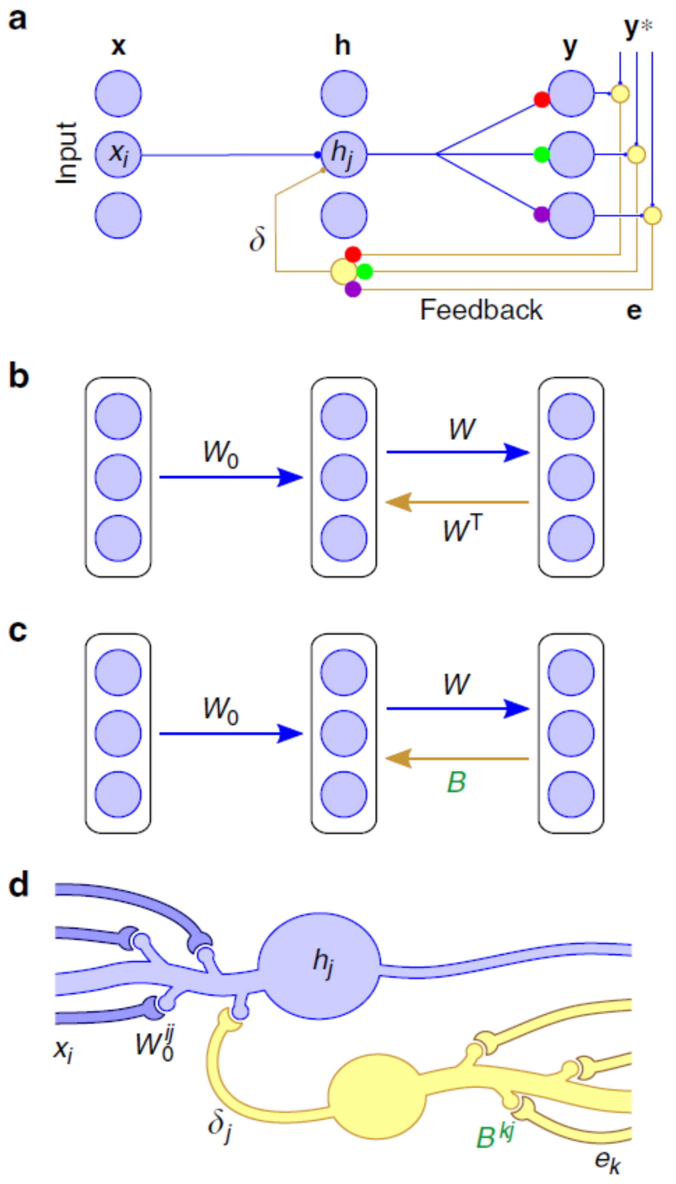


*equations neglect the component to the net input at the hidden layer from the input layer.

O'Reilly, R. C. Biologically plausible error-driven learning using local activation differences: the generalized recirculation algorithm. *Neural Comput.* 8, 895–938 (1996).

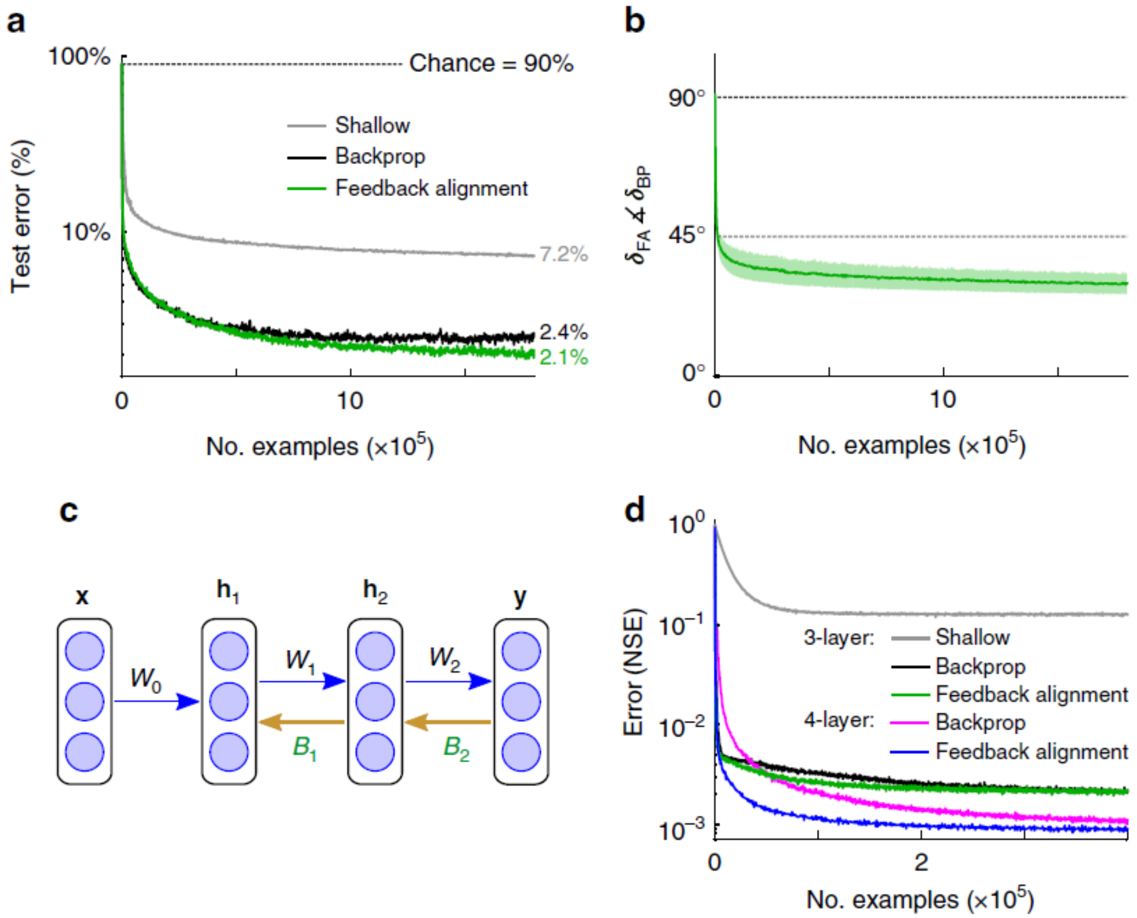
Random feedback weights can deliver useful teaching signals

Figure 1 | Random feedback weights can deliver useful teaching signals. (**a**) The backprop learning algorithm requires that neurons know each others' synaptic weights, for example, the three coloured synapses on the feedback cell at the bottom must have weights equal to those of the corresponding coloured synapses in the forward path. (**b**) Backprop computes teaching, or modulator, vectors by multiplying the error vector **e** by the transpose of the forward weight matrix *W*, that is, $\boldsymbol{\delta}_{\mathrm{BP}} = W^{\mathrm{T}}\mathbf{e}$. (**c**) Our feedback alignment method replaces W^{T} with a matrix of fixed random weights, *B*, so that $\boldsymbol{\delta}_{\mathrm{FA}} = B\mathbf{e}$. Thus, each neuron in the hidden layer receives a random projection of the error vector. (**d**) Potential synaptic circuitry underlying feedback alignment, shown for a single hidden unit (matrix superscripts denote single synapses, see main text for further explanation). This diagram is provide for illustrative purposes. There are many possible configurations that could support learning with feedback alignment, or algorithms like it, and it is this structural flexibility that we believe is important.



Lillicrap TP, Cownden D, Tweed DB, Akerman CJ: Random synaptic feedback weights support error backpropagation for deep learning. Nat Commun 2016, 7:13276 http://dx.doi.org/10.1038/ncomms13276.

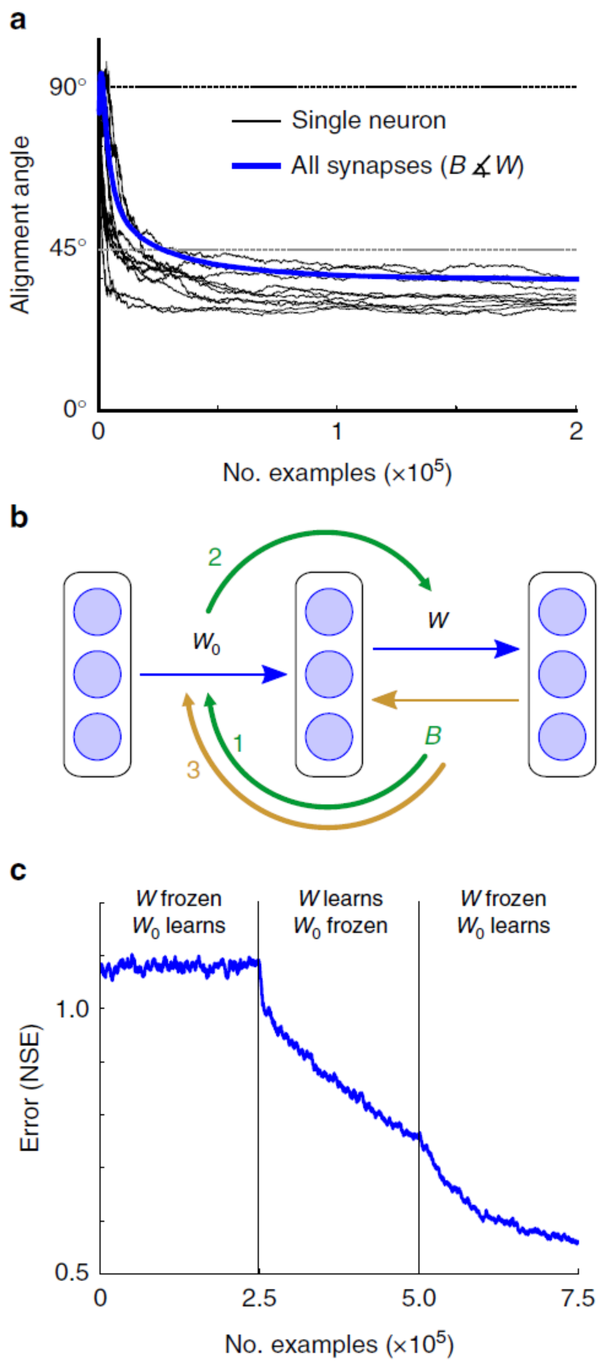
‘Feedback Alignment’ Equals or Beats BackProp on MNIST in 3 and 4 layer nets



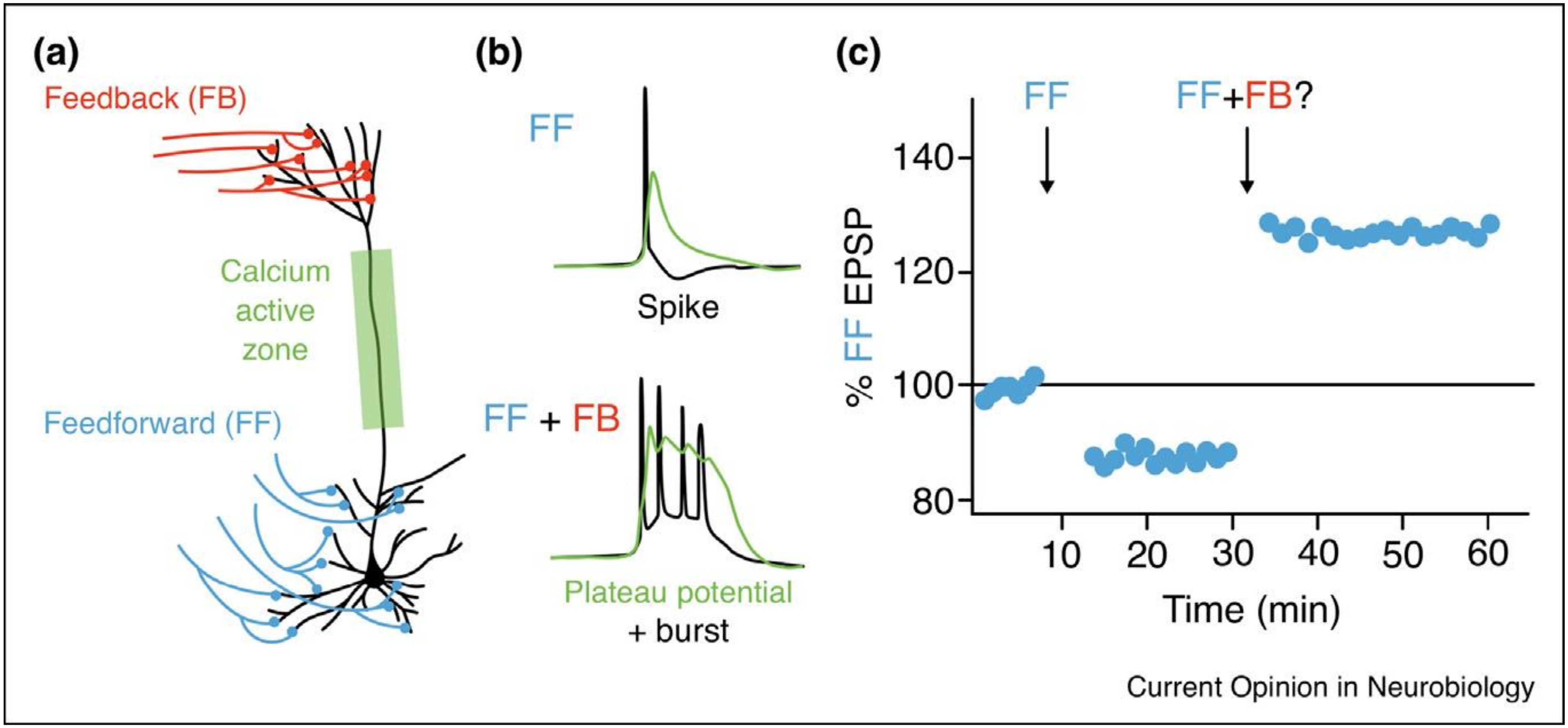
How it works

Insight into the mechanics of feedback alignment. For insight into how feedback alignment operates, we returned to the observation that the modulator signals prescribed by feedback alignment come to resemble those prescribed by backprop

To begin, note that while B and W do not directly communicate, it is still possible for B to influence the development of W in the course of learning (Fig. 5a). From equation (1), we have $\Delta W_0 \propto B\mathbf{e}\mathbf{x}^{\mathrm{T}}$, which means that information about B accumulates in W_0. This information then passes into W by its own learning rule, $\Delta W \propto \mathbf{e}\mathbf{h}^{\mathrm{T}} = \mathbf{e}\mathbf{x}^{\mathrm{T}} W_0^{\mathrm{T}}$. In short, information about B flows into W_0, altering W_0 so that it pushes W into alignment with B^{T} (Fig. 5b).



Can we make it work using Hebbian Learning?

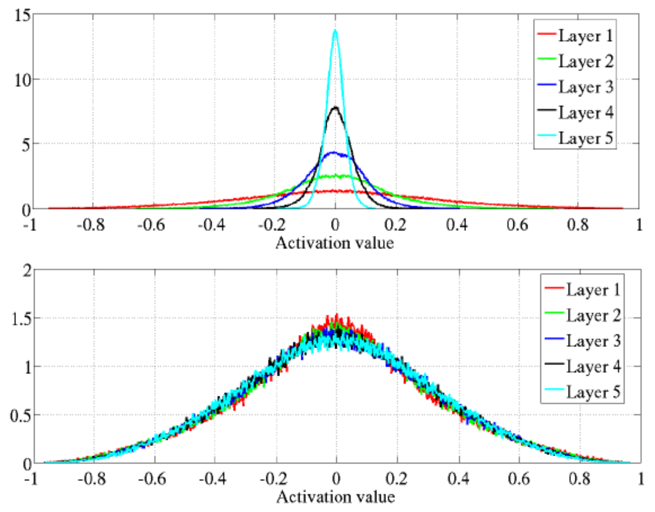


Dendritic solutions to the credit assignment problem

Blake A Richards[1,2,3] and Timothy P Lillicrap[4]

Current Opinion in Neurobiology 2019, **54**:28–36 [Link to article in Science Direct]()

Normalized Initialization



- Start with a more uniform distribution:

$$W_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$$

- This often works ok
- But it leads to degenerate activations at initialization when a *tanh* function is used.
- 'Normalized initialization' solves the problem (figure) and prodces improvement with *tanh* nonlinearity:

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

Now we know even better ways, but we'll consider these later