Supplementary Materials

S1. Code example for differentiable plasticity

The code in Listing 1 shows how to implement differentiable plasticity with PyTorch. The code defines and meta-trains a simple recurrent neural network with plastic connections. The code specific to differentiable plasticity has been highlighted in yellow; without these highlighted passages, the code simply implements a standard RNN. Note that implementing differentiable plasticity requires less than four lines of additional code on top of a simple RNN implementation.

The full program from which this snippet is extracted is available at `https://github.com/uber-common/differentiable-plasticity/blob/master/simple/simplest.py`.

Listing 1: Code snippet for meta-training a plastic RNN. Additional code to implement plasticity is highlighted in yellow.

```
w = Variable(.01 * torch.randn(NBNEUR, NBNEUR), requires_grad=True) # Fixed weights
alpha = Variable(.01 * torch.randn(NBNEUR, NBNEUR), requires_grad=True) # Plasticity coeffs.
optimizer = torch.optim.Adam([w, alpha], lr=3e-4)

for numiter in  range(1000): # Loop over episodes
    y = Variable(torch.zeros(1, NBNEUR)) # Initialize neuron activations
    hebb = Variable(torch.zeros(NBNEUR, NBNEUR)) # Initialize Hebbian traces
    inputs, target = generateInputsAndTarget() # Generate inputs & target for this episode
    optimizer.zero_grad()
    # Run the episode:
    for numstep in  range(NBSTEPS):
        yout = F.tanh( y.mm(w + torch.mul(alpha, hebb)) +
                Variable(inputs[numstep], requires_grad=False) )
        hebb = .99 * hebb + .01 * torch.ger(y[0], yout[0]) # torch.ger = Outer product
        y = yout
    # Episode done, now compute loss, apply backpropagation
    loss = (y[0] - Variable(target, requires_grad=False)). pow(2). sum()
    loss.backward()
    optimizer.step()
```

S2. Details for the image reconstruction task

Stimuli are natural images taken from the CIFAR10 dataset, of size 32×32 pixels. All images are normalized within the $[-1, 1]$ range by subtracting the mean pixel value from each pixel and then dividing the resulting values by the maximum absolute pixel value.

The network is a fully-connected recurrent network of 1,025 neurons (one per image pixel, plus one "bias" neuron with output clamped to 1). Each of the non-bias neurons corresponds to one pixel. During each episode, three images are presented three times each in succession (within each of the three presentations, the three images are shown in random order). Each image is shown for 20 time steps, with 3 time steps of zero input between each image presentation. Then the test stimulus (one of the three images, with either the top or bottom half zeroed out and providing no input) is shown for 3 time steps. The error to be minimized by the network is the sum of squared difference between the pixel values in the (non-degraded) test image and the output of the corresponding neurons at the last time step.

Input is provided to the network by clamping the output of each input-receiving neuron to the value of the corresponding pixel. Zero pixels (i.e. the pixels in the blanked out portion of the test image) provide no input. At each time step, neurons not currently receiving image input update their activation according to Eq. 1, and the plastic component of each connection is updated according to Eq. 2.

At the end of each episode, the loss was computed, and the gradient of this loss over the $\alpha_{i,j}$, $w_{i,j}$ and η parameters was computed using PyTorch's `backward()` function. This gradient was then fed to an Adam optimizer (learning rate 1e-4) to update the parameters.

S3. Discussion of the plasticity structure in trained image-reconstructing networks

The matrix of plasticity coefficients in the trained image-reconstructing network (Figure 4b, bottom) shows significant structure. This pattern contrasts with traditional models of pattern completion in plastic recurrent networks (such as Hopfield networks), which generally use homogenous plasticity across connections. Here we suggest an explanation for this complex structure.

In short, we propose that the network has learned a clever mechanism, not only to reconstruct the missing portion of the image, but also to clear away the remnant, unneeded activity from previous stimuli that might interfere with the reconstruction. This mechanism exploits the fact that partial images are always shown as top or bottom halves, and also the high autocorrelation of natural images.

At test time, if a neuron falls within the blank part of the degraded stimulus, it must determine its correct output from the activity of other neurons. For this purpose, it should take information from neurons from the other half, because they will receive the informative portion of the stimulus. This requirement is reflected in the homogenous domains of small, but significantly positive coefficients in the top-right and bottom-left quadrants of the matrix: neurons receive small positive plastic connections from the other half. Because these connections are nearly homogeneous, this portion of the network operates exactly like a standard Hopfield network (each plastic connection develops a weight that is proportional to the correlation between the two pixels across learned images, providing a total input to each pixel that is roughly the regression of that pixel's value over the values of the source pixels).

The thin diagonal stripes of high, positive plasticity simply reflect the high correlation between neighboring pixels that occurs in natural images: because neighboring neurons are highly correlated, it is useful for each pixel to receive information from its immediate neighbors.

The purpose of the large alternating bands near the mid-section is less obvious. The dark horizontal bands near the midsection mean that every pixel in the image receives connections with a large, *negative* plasticity coefficient from the farthest portion of its own half (i.e. the bottom of the top half if the pixel is in the top half, or the top few of the bottom half if the pixel is in the bottom half). Similarly, the pale bands show that each pixel receives connections with *positive* plasticity coefficients from the closest portion of the other half. What could be the purpose of this arrangement?

Due to Hebbian plasticity, plastic connections store the correlation between the two pixels (positive or negative), and thus at test time a highly plastic incoming connection will instruct the receiving neuron to fire according to their observed historical correlation with the source neuron - "how much should I fire, given your current firing rate and our observed historical correlation during the episode?"

When the plasticity coefficient has negative sign, the opposite occurs: the source neuron not only ignores, but acts contrarily to its expected firing based on current activity of the source neuron – "tell me how much I should fire based on your current firing and our past correlation, and I'll do the opposite!"

Therefore, we propose the following hypothesis: To adequately complete the partial pattern, the network must not only fill the blank portion with adequate data – it must also erase whatever information was already there as a remnant of previous activity. That seems to be the purpose of the negative plasticity coefficient from the extremum of the same half: at test time, if a neuron is within the half receiving the informative portion of the stimulus, then this connection has no effect (because neural output is clamped to the incoming stimulus). However, if they are in the blank half of the test stimulus (the one that must be reconstructed), this negative-plastic connection will instruct the neurons to alter their firing in opposition to what is predicted from the current value of their same-half brethren, thus actively "forgetting" whatever remnant activation is still echoing in this half of the network from previous stimuli. Note that on its own, this "cleaning" mechanism might interfere with correct reconstruction, e.g. when neurons start to settle on the correct reconstruction, this forced decorrelation might impair it. Yet that outcome is prevented by the counter-balancing positive-plasticity connections from the closest portion of the other half. At test time, these opposite-half pixels will always contain correct information (because they receive the informative part of the test pattern), while generally having very similar values to the nearby same-half pixels sending out negative-plasticity connections (because the corresponding pixels are very close, falling immediately on either side of the midsection). As a result, the "cleaning" effect will be strong when distal same-half pixels differ from proximal other-half pixels, but will cancel out if they have similar (correct) activity. This mechanism allows the network to actively discard junk activity while preserving accurate reconstructions generated by the low-level, homogenous plastic connections from the entire opposite half described above.