In this paper, (Meng et al., 2022b) define $m^l_{[t]} = W^l_{\text{out}}(\sigma(W^l_{\text{in}}\gamma(h^{l-1}_{[t]})))$. Given this, define

$$m^l_i = W_{\text{out}} k^l_i + r^l_i$$
$$\text{where } r^l_i = \frac{z_i - h^L_i}{L - l + 1} \quad (8)$$

Note that the denominator of r^l_i allows us to spread out the burden across multiple layer, allowing for a more scalable algorithm. It is hard to compute C^l exactly, however it can be reliably estimated using $C^l = \lambda \mathbb{E}_k[k^l k^{l\top}]$ over randomly sampled inputs, where $\lambda = 1.5 \times 10^4$. Incorporating the update gives us our desired new weights $\hat{W}^l_{\text{out}}$

A.6.3 MEND

Using the fact that the gradient of loss L with respect to the weights W_ℓ of layer ℓ of an MLP has a rank-one decomposition such that $\nabla_{W_\ell} L = \sum_{i=1}^{B} \delta^i_{\ell+1} u^{i\top}_\ell$ for a batch B, (Mitchell et al., 2021) are able to construct an editor network g_ℓ to generate the weight updates. Here, $\delta^i_{\ell+1}$ is the gradient for element i for the preactivations of layer $\ell + 1$ and u^i_ℓ are the inputs of element i into layer ℓ.
To characterize these updates, MEND employs functions that map $\delta^i_{\ell+1}$ and u^i_ℓ to a pseudo-decomposition $\tilde{\delta}^i_{\ell+1}$ and $\tilde{u}^i_\ell$ such that $\tilde{\nabla}_{W_\ell} L = \sum_{i=1}^{B} \tilde{\delta}^i_{\ell+1} \tilde{u}^{i\top}_\ell$. Letting $z_\ell = \text{concat}(\delta_{\ell+1}, u_\ell)$, the form of the network is

$$h_\ell = z_\ell + \sigma(s^1_\ell \odot (U_1 V_1 z_\ell + b) + o^1_\ell) \quad (9)$$
$$g(z_\ell) = h_\ell + \sigma(s^2_\ell \odot U_2 V_2 h_\ell + o^2_\ell) \quad (10)$$

where σ is the ReLu activation function and U_j, V_j are a low rank decomposition of MEND's weight for layer j. Note that, because of the difference in dimensions between weight matrices across layers, MEND learns different parameters for each unique shape of weight matrices to be edited. Additionally, layer-wise offset and scale parameters o_ℓ and s_ℓ are learned for both h_ℓ and g_ℓ. The final update is given by $\tilde{W} = W - \alpha_\ell \tilde{\nabla}_{W_\ell}$ with α_ℓ being another learned parameter per layer.
Given the original weights W and the updated weights $\tilde{W}$, loss is computed aggregating two training losses, editing success and locality. For a desired edit (x_e, y_e), (x'_e, y'_e) is defined as a semantically equivalent wording of the edit. Editing loss is defined as $L_e = -\log_{p_{\theta_{\tilde{W}}}} p(y'_e|x'_e)$. x_{loc} is defined as a locality sample, which is randomly sampled to test the edited model's impact on information unrelated to the edit. The corresponding locality loss is $L_{\text{loc}} = D_{KL}(p_{\theta_W}(\cdot|x_{\text{loc}}) \| p_{\theta_{\tilde{W}}}(\cdot|x_{\text{loc}}))$.
The total loss is computed as $L_{\text{MEND}} = c_e L_e(\theta_{\tilde{W}}) + L_{\text{loc}}(\theta_W, \theta_{\tilde{W}})$ where $c_e = 0.1$. The total loss is optimized using the Adam optimizer. For GPT2-XL, we edit layers 46, 47, and 48. For GPTJ, we edit layers 26, 27, and 28.

A.6.4 FT

The Fine-Tuning procedures used in this paper follow from (Meng et al., 2022b) and (Meng et al., 2022a)'s implementation for both GPT-J and GPT2-XL. MLP weights for a single layer are fine-tuned for both models. We use a constrained fine tuning approach where we add a L_∞ constraint such that $\|\theta_G - \theta_{G'}\|_\infty \leq \epsilon$ at each gradient step. For the constraint, $\epsilon = 5e-4$ for GPT2-XL and $\epsilon = 5e-5$ for GPT-J. It is optimized using Adam with a learning rate of 5e-4 for both GPT2-XL and GPT-J. We fine tune layers 18 and 6 for GPT2-XL and GPT-J respectively.