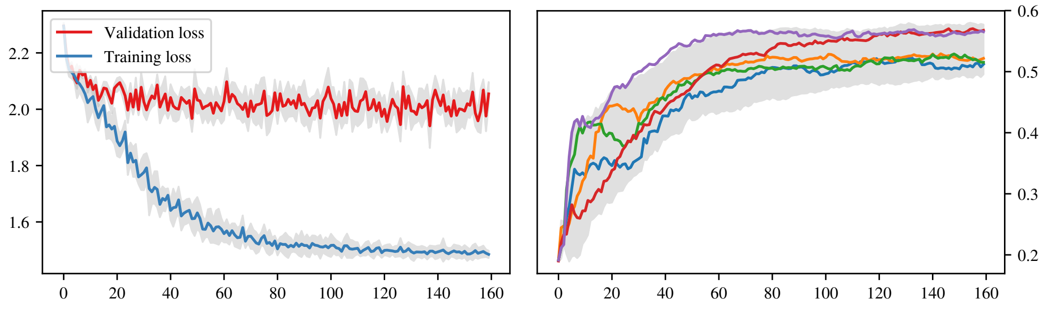
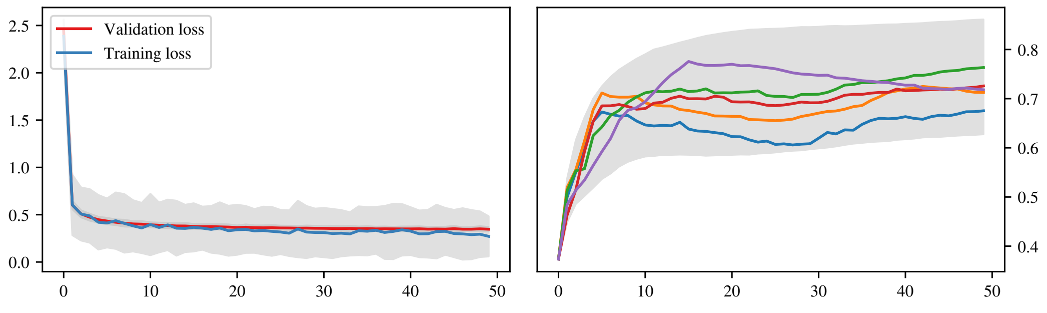
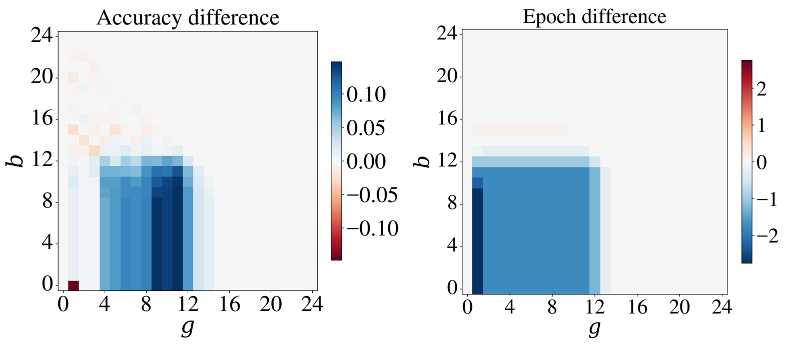


(a) CIFAR-10

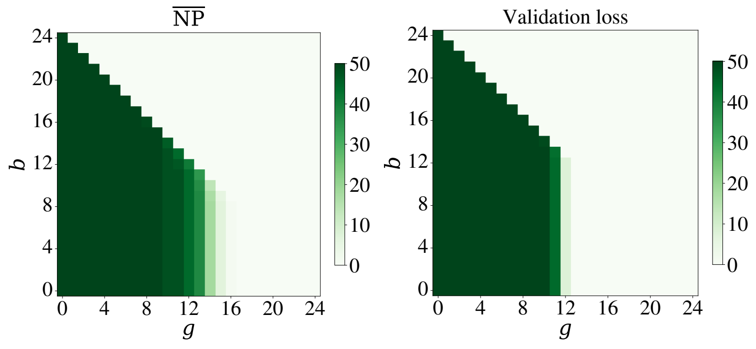


(b) Fashion-MNIST

Figure A.6: A comparison of mean normalized neural persistence curves that we obtain during the training of ‘CIFAR-10’ and ‘Fashion-MNIST’.



(a) Accuracy and epoch differences



(b) Number of triggers

Figure A.7: Additional visualizations for the ‘IMDB’ data set.

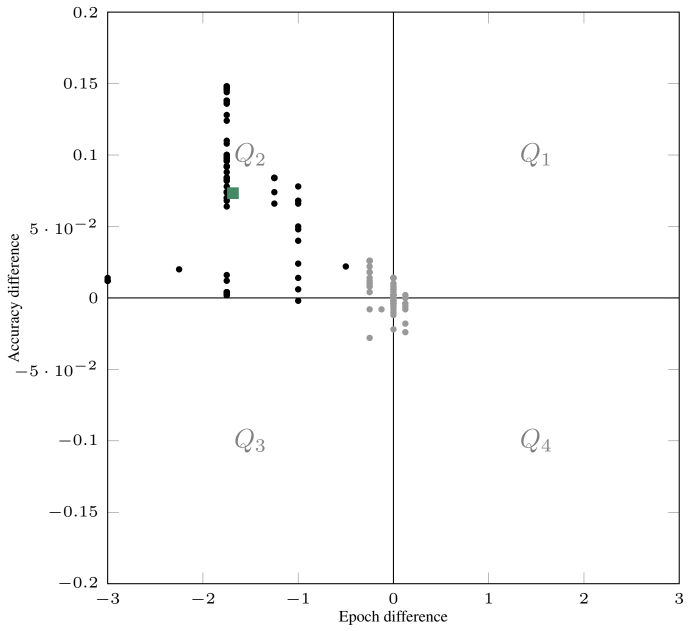


Figure A.8: Scatterplot of epoch and accuracy differences for ‘IMDB’.

### A.4 Neural Persistence for Convolutional Layers

In principle, the proposed filtration process could be applied to any bipartite graph. Hence, we can directly apply our framework to convolutional layers, provided we represent them properly. Specifically, for layer $l$ we represent the convolution of its $i$th input feature map $a_i^{(l-1)} \in \mathbb{R}^{h_{\text{in}} \times w_{\text{in}}}$ with the $j$th filter $H_j \in \mathbb{R}^{p \times q}$ as one bipartite graph $G_{i,j}$ parametrized by a sparse weight matrix $W_{i,j}^{(l)} \in \mathbb{R}^{(h_{\text{out}} \cdot w_{\text{out}}) \times (h_{\text{in}} \cdot w_{\text{in}})}$, which in each row contains the $p \cdot q$ unrolled values of $H_j$ on the diagonal, with $h_{\text{in}} - p$ zeros padded in between after each $p$ values of $\operatorname{vec}(H_j)$. This way, the flattened pre-activation can be described as $\operatorname{vec}(z_{i,j}^{(l)}) = W_{i,j}^{(l)} \cdot \operatorname{vec}(a_i^{(l-1)}) + b_{i,j}^{l} \cdot \mathbb{1}_{(h_{\text{out}} \cdot w_{\text{out}}) \times 1}$.

Since flattening does not change the topology of our bipartite graph, we compute the normalized neural persistence on this sparse weight matrix $W_{i,j}^{(l)}$ as the unrolled analogue of the fully-connected network's weight matrix. Averaging over all filters then gives a per-layer measure, similar to the way we derived mean normalized neural persistence in the main paper.

When studying the unrolled adjacency matrix $W_{i,j}^{(l)}$, it becomes clear that the edge filtration process can be approximated in a closed form. Specifically, for $m$ and $n$ input and output neurons we initialize $\tau = m + n$ connected components. When using zero padding, the additional dummy input neurons have to included in $m$. For all $\tau$ tuples in the persistence diagram the creation event $c = 1$. Notably, each output neuron shares the same set of edge weights.

Due to this, the destruction events—except for a few special cases—simplify to a list of length $\tau$ containing the largest filter values (each value is contained $n$ times) in descending order until the list is filled. This simplification of neural persistence of a convolution with one filter is shown as a closed expression in Equations 7–11, and our implementation is sketched in Algorithm 3. We thus obtain

$$\operatorname{NP}(G_{i,j}) = \|\mathbb{1} - \widetilde{\mathbf{w}}\|_p\,, \tag{7}$$

where we use

$$\|\widetilde{\mathbf{w}}\|_p \leq \left\| \left(0, \mathbf{w}_c^T, \mathbf{w}_{\bar{c},\phi}^T, \operatorname{vec}(A_\phi)^T, \operatorname{vec}(B_\phi)^T\ \right)^T \right\|_p, \tag{8}$$

with

$$\phi = \tau - \dim(\mathbf{w}_c) - 1, \tag{9}$$

$$A_x = \mathbf{w}_{1:\lfloor \frac{x}{n} \rfloor} \otimes \mathbb{1}_{n-1}, \tag{10}$$

$$B_y = \mathbf{w}_{\lfloor \frac{y}{n} \rfloor + 1} \otimes \mathbb{1}_{y \bmod n}, \tag{11}$$

where $\mathbb{1}_0 := 0$. Following this notation, Equation 7 expresses neural persistence of the bipartite graph $G_{i,j}$, with $\widetilde{\mathbf{w}}$ denoting the vector of selected weights (i.e. the destruction events) when calculating the persistence diagram. We use $\mathbf{w}$ to denote the flattened and sorted weight values (in descending order) of the convolutional filter $H_j$, while $\mathbf{w}_c$ represents the vector of all weights that are located in a corner of $H_j$, whereas $\mathbf{w}_{\bar{c},\phi}$ is the vector of all weights which do *not* originate from the corner of the filter while still belonging to the first (and thus *largest*) $\left\lfloor \frac{\phi}{n} \right\rfloor$ weights in $\mathbf{w}$, which we denote by $\mathbf{w}_{1:\lfloor \frac{\phi}{n} \rfloor}$.

For the subsequent experiments (see below), we use a simple CNN that employs $32 + 2048$ filters. Hence, by using the shortcut described above, we do not have to unroll 2080 weight matrices explicitly, thereby gaining *both* in memory efficiency and run time, as compared to the naive approach: on average, a naive exact computation based on unrolling required $8.77\,\text{s}$ per convolutional filter and evaluation step, whereas the approximation only took about $0.000\,38\,\text{s}$ while showing very similar behaviour up to a constant offset.

For our experiments, we used an off-the-shelf 'LeNet-like' CNN model architecture (two convolutional layers each with max pooling and ReLU, 1 fully-connected and softmax) as described in Abadi et al. (2015). We trained the model on 'Fashion-MNIST' and included this setup in the early stopping experiments (100 runs of 20 epochs). In Figure A.9, we observe that stopping based on the neural persistence of a convolutional layer typically *only* incurs a considerable loss of accuracy: given a final test accuracy of $91.73 \pm 0.13$, stopping with this naive extension of our measure reduces accuracy by up to 4%. Furthermore, in contrast to early stopping on a fully-connected architecture, we do not observe any parameter combinations that stop early *and* increase accuracy. In fact, there