TEMPORAL REGULARIZED LEARNING: SELF-SUPERVISED LEARNING LOCAL IN SPACE AND TIME

Davide Wiest
TU Darmstadt
Darmstadt
davide.wiest2@gmail.com

ABSTRACT

Temporal Regularized Learning (TRL) is a compact, highly local and self-supervised prodecure that optimizes each neuron individually. We adapt the self-supervised loss formulation of VICReg, consisting of variance, invariance and covariance to input streams with sequential coherence and for online-compatibility. It removes the need for biphasic updates, negatives or inner-loop convergence, given three scalar leaky-integrators per neuron and an auxiliary lateral network. Knowledge about downstream tasks can be injected through the sequence ordering, allowing for supervised training. We present TRL and its simplified variant, TRL-S. Experiments on MNIST show TRL is competetive with backpropagation, Forward-Forward and Equilibrium Propagation, while TRL-S achieves similar performace despite its simplified setup. We show TRL creates neurons with specialized receptive fields at the first layer. In later layers, some neurons specialize in firing only for some types of input.

1 Introduction

Learning from unlabeled sensory streams is a practical necessity for both biological modeling and for energy efficient machine learning on constrained hardware. Modern self-supervised methods succeed by aligning representations of related samples, called positives, while avoiding collapse by either negative samples or explicit regularizers. Contrastive methods rely on negatives and on large batch sizes to estimate cross-sample statistics. [1, 2]

Regularized alternatives avoid explicit negatives and instead combine an invariance term with variance and covariance penalties. [3, 4, 5, 6] VICReg is a regularized method that enforces invariance without negatives and whose regularizers can be computed from local moments, at least in a batched implementation. [3]

This paper shows how a VICReg style objective can be converted into a per-neuron temporal loss that uses only signals that are local in space and time. The motivation is practical: Locality reduces communication between units. Temporal locality permits streaming implementations with only a small short-term memory per neuron. Both properties are desirable for analog and neuromorphic hardware. [7, 8]

By making the invariance temporal TRL is a type of Slow Feature Analysis (SFA) procedure: SFA aims to extract slowly changing features from a temporal sequence of faster-changing inputs. [9, 10] This can lead to representations that are invariant to augmentations or capture latent variables. There is a case to be made that this applies not just to perceptual recognition but to most latent variables of sequential tasks. In text, the sentiment of a review remains stable across its many words; in finance, stock prices fluctuate faster than the market factors that drive them; in physiology, a patient's underlying state or health condition evolves slowly relative to heartbeat, respiratory cycles, and momentary noise in sensor traces.

The paper makes three contributions. First, it derives the TRL loss starting from VICReg and gives the precise per-neuron loss used in experiments. It analyzes the method from the perspectives of parallelizability, online learning, and discusses the choice of the optimizer for the training setup. Second, the training procedure connects self-supervised representation learning directly to SFA and trace learning, motivating possible improvements to existing methods in those fields. [10, 11] Third, it provides a controlled experimental evaluation on MNIST and synthetic temporal

orderings, together with reference of explorative tests and representation diagnostics. [12, 13] Full reproducibility details, including dataset, seeds and hyperparameter choices are described in the appendix.

2 Method

2.1 Starting point: VICReg

VICReg decomposes a representation level objective into three terms. [3] VICReg uses the following components in its per-batch loss

1. an invariance term that measures distance between paired representations,
2. a hinge loss on the standard deviation that encourages standard deviations to exceed a threshold, and
3. a covariance regularizer that penalizes off-diagonal covariance between features.

From a high level, the VICReg loss can be written as:

$$L_{\mathrm{VIC}} = \lambda_{\mathrm{sim}} L_{\mathrm{sim}}^{\mathrm{rep}} + \lambda_{\mathrm{var}} L_{\mathrm{var}}^{\mathrm{rep}} + \lambda_{\mathrm{cov}} L_{\mathrm{cov}}^{\mathrm{rep}}. \tag{1}$$

(Bardes et al., 2022)

VICReg is a convenient starting point because it does not require negative samples and because its variance and covariance terms can be written in terms of low order statistics that a local unit can maintain. The next subsection shows how these representation-level terms can be converted into local, temporally structured terms.

2.2 From representation VICReg to a per-neuron TRL loss

Contrary to the VICReg setup, we do not apply this loss on expanders, and let the gradients be backpropagated to the backbone, since it obviously conflicts with our aim to derive a local learning procedure. [14] Our first step is to apply the loss directly on the activations of a layer, and not propagating gradients backward. This makes it spatially local in principle. Our next steps are concerned with adapting the loss formulas to fit temporal locality.

The VICReg loss can be written as:

$$L = C_S L_S + C_V L_V + C_C L_C \tag{2}$$

Where the invariance/similarity loss is

$$L_S = (z - z_{\text{pos}})^2 \tag{3}$$

the variance loss is

$$L_V = \text{ReLU}(\tau - \text{std}(z_{:,j})) \tag{4}$$

and the covariance loss is

$$L_C = \sum_{i \neq j} \left(C_{ij}\right)^2 \tag{5}$$

where z is in $\mathbb{R}^{B \times D}$ and C is the Covariance matrix calculated over the batch.

Let's first modify the similarity loss. We introduce a sequential coherence within the batch, and use sequentially adjacent samples as pairwise positives. How we introduce this coherence is specified later. For a neuron j with scalar representation output z_i for an input with index i, this can be written as:

$$L_{S,j} = (z_{i,j} - z_{i-1,j})^2 \tag{6}$$

[10, 9]

Let's now go over the variance term: Firstly, the standard deviation can be exchanged for the variance. For simpler gradient flow and downstream modification, we decompose the hinge loss into the product of a gradient-free *gating* term, and a simpler *gradient* term. For a single neuron with and minimum variance τ and activation vector $z_{:,j}$ for a sequence of inputs, we can write this as:

$$L_{V,:,j} = -\text{ReLU}\big(\tau - \text{detach}(\text{var}\big(z_{:,j}\big))\big)\ \text{var}(z_{:,j}) \tag{7}$$

We can now introduce the sequence-index i and use $\text{var}(z_{i,j}) = (z_{i,j} - \bar{z}_{i,j})^2$:

$$L_{V,i,j} = -\text{ReLU}\big(\tau - \text{detach}(\text{var}\big(z_{:,j}\big))\big)\ (z_{i,j} - \bar{z}_{i,j})^2 \tag{8}$$

Crucially, we do not replace the variance with an instantaneous variance in the gating term to retain the losses functionality.

Finally, we can focus on the covariance term. We introduce a lateral layer lat, which is a linear layer without bias connecting a layer of the model with itself. We optimize the lateral mappings's weights to match the empirical covariance matrix. An elementwide product between the mean-normalized activations and the output of the lateral layer, receiving those activations as input, is a suitable loss function to eliminate covariance. When using the original covariance loss, it essentially calculates a sum of products of activations, normalized by their mean. The gradient with respect to one neuron is the respectively other neurons activations. This holds for the proposed alternative. The difference is that we apply the multiplication after the sum instead of before. Fortunately, we can detach the input to the lateral mapping and avoid breaking locality, since covariance is symmetric and $z_{i,j}$ already includes all neurons in the formula.

$$L_{C,i,j} = z_{i,j} \text{ lat}(\text{detach}(z_i - \bar{z}_i)) \tag{9}$$

[15, 11]

A weighted sum of those three terms make up the TRL loss utilized by the ***TRL*** setup in the experiments. The empirical choice of weights is explored later. If we detach the previous activations $z_{i-1,j}$ in the similarity loss, we have the loss used by ***TRL-S***, standing for TRL simplified. We can do this without sacrificing any significant amount of performance, because we still have $z_{i,j}$ enabling a gradient in the loss formulation.

We now focus on the compatibility with online-learning and hardware-constrained environments. The losses are defined in such a way to consist of either 1) activations, used relatively immediately 2) constants, as far as the error gradient is concerned. We in fact show how this loss simplifies nicely. But first, we focus on memory.

2.3 Memory requirements

The TRL loss requires following information to be stored for each neuron:

1. a running mean estimate m_j,
2. a running variance estimate v_j,
3. the previous activation $z_{i-1,j}$ stored as a short-term memory.

Importantly, this information can be stored at the neuron level and never has to leave a neuron. A natural choice for storing the mean and variance statistics is a leaky integrator. It computes the exponential moving average with a predefined momentum β:

$$\text{stat}_t \leftarrow \beta \text{ stat}_{t-1} + (1 - \beta)\, x_t \tag{10}$$

[7]

The lateral mapping is a way to store the covariance matrix that homogenizes the architecture. At the same time, it enables a local covariance loss formula. Weight updates defined as mean-centered product of two neurons' activations recover their covariance over time. The online update rule can trivially be expressed as:

$$\Delta w_{j,k \neq j} \propto z_{i,j} z_{i,k} \tag{11}$$

2.4 Simplified algebra and the per-weight gradient expression

We will now further simplify the loss formula to show the relatively direct relation between the loss and a weight's gradient. Let C_S, C_V and C_C be the similarity, variance and covariance coefficients, respectively. Here, we move the gradient-free gating term of the variance loss into C_V, such that $C_V \leftarrow -C_V \text{relu}(1.0 - v)$, where v is the running variance statistic. We use the same notation for the loss terms. Furthermore, let z be the current scalar activaton of a neuron and p its previous activaton. Furthermore, let m be the current running mean. We define lat as the lateral mapping and $\phi'(a)$ as derivative of the nonlinearity. This is used in the simplified formulation of the loss:

$$\begin{aligned} L &= C_S L_S + C_V L_V + C_C L_C \\ &= C_S (p - z)^2 + C_V (z - m)^2 + C_C z \text{ lat}(z - m) \\ &= C_S (p^2 + z^2 - 2pz) + C_V (z^2 + m^2 - 2mz) + C_C z \text{ lat}(z - m) \end{aligned}$$

Remove p^2 and m^2 as they do not contribute to the gradient with respect to z.

$$\begin{aligned} L &\approx (C_S + C_V) z^2 - 2(C_S p + C_V m) z + C_C z \text{lat}(z - m) \\ &= z((C_S + C_V) z - 2C_S p - 2C_V m + C_C \text{lat}(z - m)) \end{aligned}$$

Then the gradient for a weight is:

$$\frac{\partial L}{\partial w} = x\ \phi'(a)\ [2(C_S + C_V)z - 2C_S p - 2C_V m + C_C \text{lat}(z)] \tag{12}$$

lat(z) stays the same since we detach z before passing it through the lateral mapping. To put it concisely, the output-related term of the gradient is a simple fourfold weighted sum. As in the general case, the weight updates can be implemented in a hebbian style, as product of input and an output-related term with consideration for the activation function derivative. [11]

2.5 Functionality & Parallelizability

TRL does not attempt to approximate global backpropagation across long paths in the network. At the same time, because the learning objective is local in space and time, the method does not accumulate approximation error with depth in the same way as methods that approximate global gradients. This comes with advantages for parallelizability and throughput too.

The computation of can be modeled neuron is as follows:

```
# new z
m = update_mean(z, m)
inst_v = (v-m)**2
v = update_variance(inst_v, v)
loss = trloss(z, m, v)
send_to_weights(loss)
p = z
```

TRL requires the following training sequence for each layer:

1. a forward pass to produce activations z for the chunk
2. a single forward pass through the lateral mapping to produce lat(z) when the lateral module is used
3. local gradient computation wrt. the activation z followed by weight updates for all neurons in the layer which can proceed in parallel

Since the lateral network exists purely for the covariance loss, we can completely ignore it at inference, if the instantiation of the algorithm allows for it. The only sequential operations at training are that we have to wait for the lateral head and wait until the weights have been updated, before we can execute another layer-level forward pass. During the lateral pass, there should be plenty of time to update the running statistics. Likewise, p can be set during the weight updates.

For streaming implementations one may choose an approximate asynchronous variant where the lateral module consumes previous activations and therefore produces a one-step-shifted decorrelation signal. This shift would enables the layer to apply approximate local updates without waiting for the immediate lateral pass. Such a setup seem promising but requires experimental validation.

2.6 Plausibility for constrained hardware

TRL is designed to be simple. Each neuron requires only two running scalars and one short-term activation memory. These properties make TRL a favorable candidate for implementations on neuromorphic or analog substrates where local state and limited buffering are preferred. [7, 8] Notably, the storage of the running statistics can be implemented as leaky integrators. A simple RC-Circuit could theoretically suffice as instantiation of such leaky integrators.

Here's a full computation graph for a neuron and one weight, using the simplified loss expression derived earier with $C_Z = C_S + C_V$. Green nodes are computational elements, yellow squares are dynamic variables and blue squares represent memory units.

2.7 Interaction between lateral layer and optimizer

The lateral D by D module implements an explicit mapping that approximates off-diagonal covariance. Using a learned lateral layer by default is an architectural choice made for three reasons: it yields a homogeneous layer element that matches common neural simulation practice, it stores covariance related information explicitly as weights which is convenient for analog or neuromorphic devices, and it empirically stabilizes training in our experiments. Because the

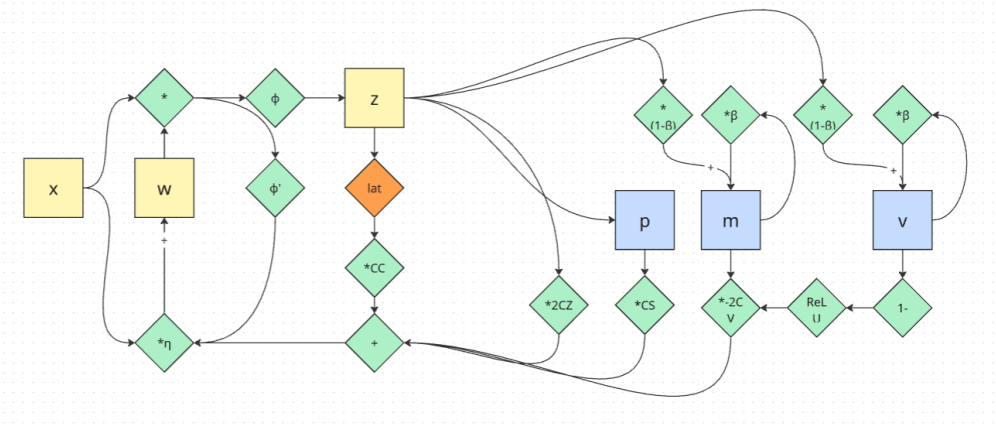

Figure 1: A full computation graph for a neuron and one weight, using the simplified loss expression derived earier with $C_Z = C_S + C_V$. Green nodes are computational elements, yellow squares are dynamic variables and blue squares represent memory units.

covariance objective depends on second moments, training the lateral module and reducing the off-diagonal energy interacts strongly with the effective handling of second moment statistics by the optimizer. Empirically we found that adaptive optimizers that track second moment like Adam produce stable covariance minimization in TRL runs. [16, 17] In contrast, standard stochastic gradient descent with momentum often failed to minimize the lateral covariance term reliably in our setup. A representative SGD run that used the lateral module reached 93 percent test accuracy for one seed but further tuning did not close the gap to Adam based TRL runs. This optimizer interaction should be considered an implementational constraint when choosing whether to use a lateral layer or to rely on direct covariance estimates.

3 Theoretical Discussion

This section a brief review and comparison of closely related literature. The aim is to describe TRL relative to other local learning approaches and biologically plausible learning algorithms such as trace learning.

3.1 Comparative perspective

TRL occupies a point in design space at the intersection of self-supervised and local learning procedures.

Compared to contrastive methods that rely on negative samples, TRL requires only temporally adjacent positives and a regularizer to avoid collapse, by using the VICReg loss as starting point. [1, 2]

Compared to VICReg, TRL moves the regularization to the neuron level and substitutes temporally adjacent positives for augmented pairs. The adaptation preserves the conceptual roles of invariance, variance maintenance, and decorrelation while changing the positive construction from augmented pairs to temporally contiguous samples. [3]

Compared to Forward-Forward, TRL uses a continuous regularized invariance objective rather than layer-level goodness classification, and TRL relies on temporal coherency rather than constructed positive and negative passes. [18]

Compared to energy-based models and equilibrium propagation, TRL does not require iterative network settling and therefore trades iterative dynamics for local computation and a single lateral pass per layer. [19, 20]

An interesting recurring theme of learning algorithms is that most of them need two types of passes through the network. Backpropagation uses a forward and a backward pass, the Forward Forward algorithm uses two forward passes, and Equilibrium Propagation uses a free and a clamped pass. [21, 18, 19] This unfortunately holds for TRL too: The second type of pass is the one through the lateral layer. It is necessitated not by optimization but to reduce correlation, as we have seen in the ablations.

3.2 Relation to slow feature analysis and trace learning

TRL's invariance objective matches the goal of Slow Feature Analysis because it uses temporal adjacency as positives. [9] In fact, the MSE between two contiguous samples is exactly the first derivative of activations in discrete time that is

being squared and minimized in SFA literature. The whitening term of SFA maps to covariance regularization. To make it concrete for the supervised experiments, the faster-changing inputs are the individual digit samples, while the slower feature is the class membership of all those samples within a chunk. TRL is closely related to Bio-SFA, a biologically plausible SFA algorithm. Bio-SFA works with a hard whitening constraint and uses iterative settling, whereas TRL penalizes covariance softly and its training can be implemented as a series of discrete-time passes. [22] We believe soft constraints are better for neural optimization and reference the benefit of a sparser, less restrictive lateral mapping for larger network layouts shown in the experiments. What makes Bio-SFA interesting compared to TRL is that Bio-SFA is derived directly from the SFA objective and therefore inherits analytic characterizations of the slow subspace under specific statistical assumptions.

Trace Learning is a biologically plausible, simple learning procedure that modifies hebbian updates by introducing a temporal aspect. Instead of using a neuron's activations for the weight update, trace learning uses an activation trace. This rule can be used to form view-invariant representation of objects, demonstrating applicability in the perceptual domain. [10, 23]

We can mathematically show the relation between the trace learning rule and the similarity loss term of TRL. Starting from the gradient of the similarity loss for a weight w_{kj} and neuron j with activation $z_{i,j}$ and previous activation $z_{i-1,j}$. Let ϕ be the nonlinearity and $a_{i,j}$ be the activation before the nonlinearity:

$$\begin{aligned}
\Delta w_{kj} &\propto \frac{\partial L}{\partial w_{kj}} \\
&= \frac{\partial L}{\partial \phi(a)} \frac{\partial \phi(a)}{\partial a_{i,j}} \frac{\partial a_{i,j}}{\partial w_{kj}} \\
&= \phi'(a) \frac{\partial a_{i,j}}{\partial w_{kj}} \left(z_{i-1,j} - z_{i,j}\right) \\
&= \phi'(a) x_k \left(z_{i-1,j} - z_{i,j}\right)
\end{aligned}$$

The trace learning rule at a time step i is defined as follows (Földiák):

$$\Delta w_{kj} = \alpha\, \bar{z}_{i,j} \left(x_k - w_{kj}\right) \tag{13}$$

Ignoring the weight decay term and replacing $\bar{z}_{i,j}$ by its definition yields:

$$\Delta w_{kj} \propto x_k \left(\beta\, \bar{z} + (1-\beta)\, z_{i,j}\right) \tag{14}$$

Replacing the running trace with the previous activation can make the algebra superficially similar to the TRL. However, an exact algebraic equivalence requires coefficient constraints that cannot be satisfied without changing the *functional* form of the update.

On the one side, we require the momentum β to in the interval $(1; 2)$. To achieve mathematical equivalence, we require $\beta = 1$, and $1 - \beta = -1 \Leftrightarrow \beta = 2$, for both uses of β, respectively.

Take note that the trace rule furthermore *requires* a weight-dependent decay term to prevent runaway effects. To put it concisely, trace learning and TRL share the same operational goal of promoting temporal invariance but do so in mathematically different ways. [15]

Our ablations on the loss terms show that both regularizations are necessary. This suggests that trace learning can likewise benefit from these terms. Adding those would require more complex per-neuron computation and additional store, but does not violate the locality constraints the make trace learning a biologically plausible learning rule.

3.3 Biological correspondences

It has to be noted that, because the implementations of TRL are digital in nature, the correspondence to biological mechanisms can at most be theoretical and phenomenologic, which is why we will mention them very briefly. The variance regularization serve the same purpose as homeostatic plasticity terms that regulate firing rate. [24] The lateral layer superficially maps to lateral inhibition, since both reduce correlation between neurons in the same level of the processing hierarchy. [15] We tried to implement it as part of the inferential process, ie, to predict and affect activations in a subtractive way, but this was incompatible with the digital setup. The lateral layer learns to predict the activations almost perfectly and eliminates flow of information from flowing through the network.

4 Experiments on MNIST

4.1 Experimental protocol

We performed a controlled, small-scale evaluation of both TRL setups on the MNIST dataset [25] with synthetic sequential coherence. A simple multilayer perceptrons of different layer sizes and widths was used as *encoder*. The downstream evaluation protocol trains a linear classifier (*head*) on encoder activations of the training set and validates on the validation set. We compare the algorithm with standard supervised Backpropagation [21], Forward Forward [18] and Equilibrium Propagation [19] and use the model architecture chosen by them, respectively. Additionally, a smaller layout is tested. Effects of batch normalization [26] and simple affine augmentations are measured for this layout in a separate suite of runs. Besides this, the training uses no regularization. Forward-Forward baselines are reported in their native headless format. As choice for how to introduce the sequential coherence, we create chunks of same-class samples. This is effectively a supervised setup, in line with the other comparison algorithms. As noted by Hinton for his supervised "Forward Forward" setup, the network should pick up only features that are relevant to discrimination of different classes.

Table 1: The Setup

Parameter	**Applies to**	**Value**
Epochs	All	60
Activation Function	All	ReLU
Seeds	TRL, TRL-S, Backprop	42 to 46
Optimizer	TRL, TRL-S, backprop	Adam
Batch size	TRL, TRL-S, Backprop	64
Neuron statistics	TRL, TRL-S	In-Batch detached
Chunk size	TRL, TRL-S	16
Covariance matrix implementation	TRL, TRL-S	Lateral mapping

Ablations include

- dropping each of the three TRL loss terms in turn,
- replacing the lateral head with a direct off-diagonal penalty computed in-batch,
- experimenting with chunk sizes,
- early experiments on the minimum variance τ,
- and testing optimizer interactions such as Adam versus SGD with and without the lateral head.

4.2 TRL and TRL-S setup

As introduced in 2.2, we propose two variations of TRL, TRL and TRL-S (TRL-Simplified). While the goal of TRL was showing the capability of the learning procedure as compared to other alternatives, TRL-S takes the reductionist approach a step further and shows how some implementational trade-offs affect performance. The TRL-setup is trained in batches and layers are trained in sequence/greedily. For this setup, we explicitly use knowledge about synthetic coherent chunks of data by avoiding the similarity loss at the transition between chunks, i.e., where there is no coherence. A head for downstream tasks uses the activations of all layers. In contrast, TRL-S is trained concurrently and does not use such knowledge. Additionally, it detaches the previous activation. The simplified loss formula of 2.4 holds principally. A head receive only the last layer's activations. It should be noted that TRL-S was not tuned to the same extent as TRL and is reported as a conservative baseline for simplified implementations. For the largest layout, layers are wide and the decorrelation penalty therefore is high, so we introduce a sparsity of 50% to the lateral mapping there, by applying a persistent random mask to the covariance matrix. Full run scripts, parameter counts, and references to the chosen hyperparameters appear in the appendix.

Important hyperparameters include the learning rate $\eta = 10^{-4}$, the minimum variance $\tau = 1.0$ and the default loss coefficients:

Additionally, the lateral loss is multiplied by $C_L = 1.2$.

Choice of Loss coefficients and learning rate In the general case, $C_S + C_C < C_V$ prevents collapse, since both similarity and covariance penalize the magnitude of the activation. The main move in terms of parameter tuning is to change the variance loss for some situations:

Table 2: Loss Coefficients

Parameter	Value
C_S	1.0
C_V	5.0
C_C	1.0

- If we detach the previous activation (as we do in the TRL-S setup), experimentation has shown that doubling the variance coefficient is beneficial.
- If we use batch normalization, we can quadruple the learning rate and halve the variance loss coefficient.
- And if we use SGD, we double the variance loss coefficient.

4.3 Results

Table 3: Validation error table

Hidden Layers	Aug & R-BN	Backprop (N=5)	TRL (N=5)	TRL-S (N=5)	FF	EqProp
512, 256	Y	1.11	1.68	2.00		
	N	2.02	3.94	4.39		
500 thrice	N	2.01	3.06	3.56		3.0
4 layers 2000 each	N	1.88	2.16	2.44	1.36	

We can see that TRL and TRL-S achieve satisfactory results. Both are within about 2% error from backpropagation, while the gap closes significantly while scaling up the model size, shrinking to 0.26% for the largest model layout. It achieves comparable performance to Equilibrium Propagation, but lags behind Forward Forward. Secondly, TRL benefits at least as much from augmentation and batch normalization as backpropagation does. Interestingly, we did not match the reported performance of Backpropagation in the Forward Forward paper. This suggests that the hyperparameters of the Adam optimizer are not especially suited for the small-scale MNIST dataset. We should be able to close the gap for backpropagation and, adopting the tuned parameters, reduce the gap between TRL and FF.

4.4 Ablations and further experiments

An untrained encoder with the 512 by 256 layout yields approximately 90% validation accuracy when a linear head is trained, in batches, on last layer's activations. We use this figure as a baseline threshold; any encoder whose linear head accuracy falls substantially below this threshold is considered to be in a failure mode. Ablation of any single TRL loss term reduces peak performance to at most seventy percent in batched offline experiments, showing that all terms are necessary, even for small-scale networks. Activation function changes from ReLU to tanh or sigmoid reduce accuracy by less than four percentage points in the tested configurations. Using the GeLU activation function improved performance [27]. We observe that TRL paired with a learned lateral layer creates an optimization landscape that benefits from optimizers capable of handling second moment structure. In our runs, Adam achieves stable covariance minimization with covariance loss magnitudes around one to two at steady state. SGD with the lateral head tended to produce much larger covariance loss values, in the order of dozens, and poorer final accuracy in the tested configurations: SGD runs achieve 93% accuracy for TRL at seed 42 when halving the variance coefficient. The empirical observations motivate recommending Adam for TRL with lateral layers in the current implementation.

Choosing the minimum variance τ as 1 yielded the best performance. Using a random distribution of τ for different neurons did not improve performance either. Chunk size is critical: A run with the TRL setup without batchnorm with a chunk size of 8 yielded validation accuracy of 96.68% after 20 epochs, which is significantly lower than for a chunk size of 16. A run with chunk_size=32 achieves achieves 93.98% accuracy. We additionally tested a setup with recurring layers. It was not possible to trade more compute for less parameters.

4.5 Representations

We analyze learned representations at multiple layers for the small-scale network after 20 epochs. The following are representative qualitative observations:

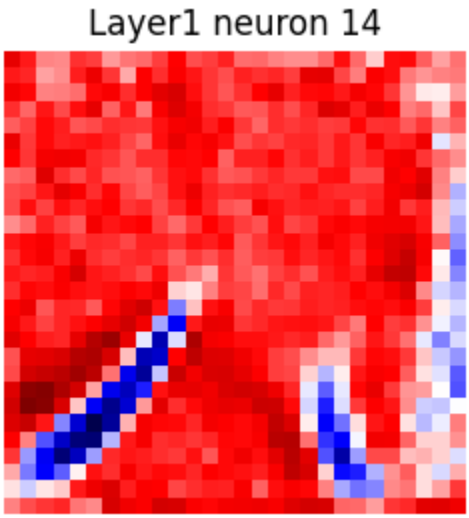

Figure 2: First layer receptive fields are often blob-like and sometimes detect oriented strokes. A subset of first layer units exhibit small, localized feature detectors.

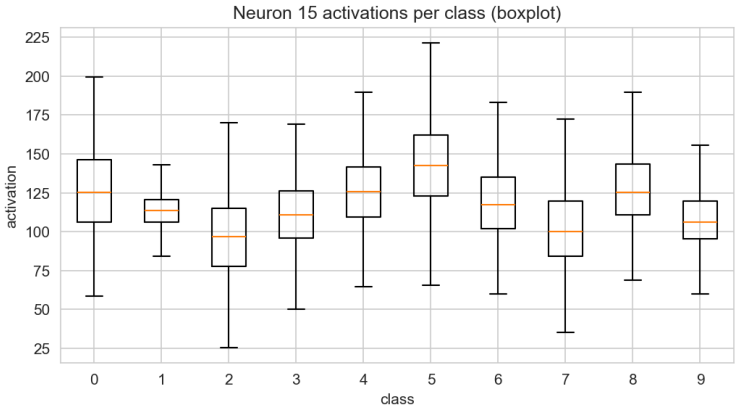

Figure 3: On average, second-layer neurons show relatively poor class selectivity.

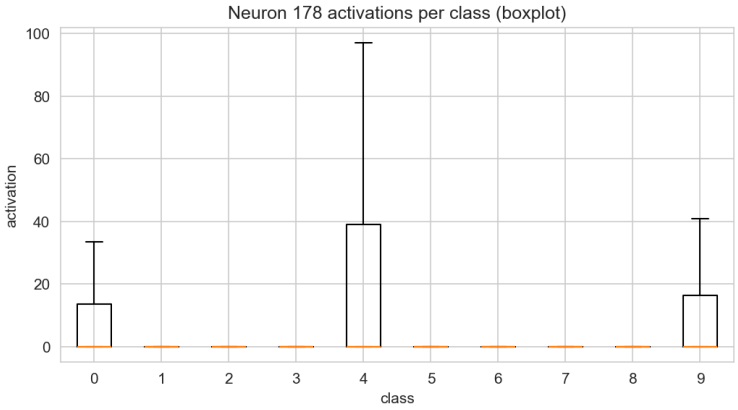

Figure 4: However, a minority of neurons show strong specialization consistent with the covariance pressure in the loss. Those specialized neurons respond selectively to particular digit shapes or digits.

Within the second layer, neurons that respond strongest to certain digits are relatively common, much higher than the prevalence of neurons that respond to particular digits. This is indicative of neurons specializing despite their activation distribution suggesting otherwise.

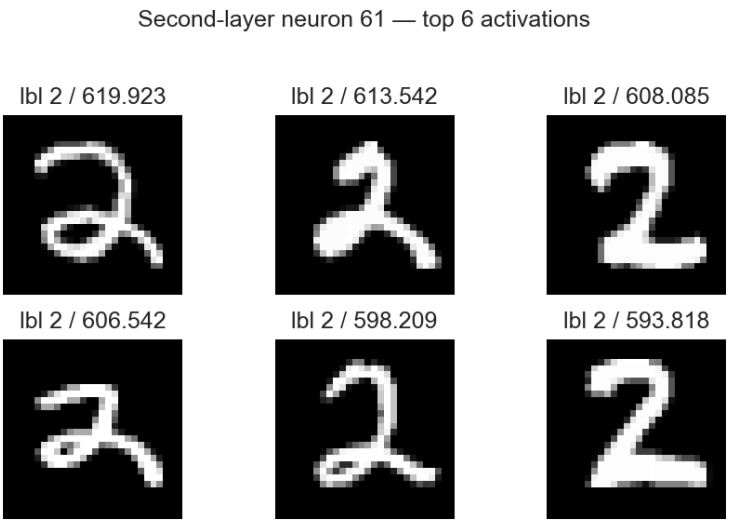

Figure 5: Example of a neuron specializing.

Full collections of receptive field grids and separability plots are in the appendix.

5 Limitation and Future work

5.1 Limitations

- Experiments are limited to MNIST and synthetic temporal orderings.
- Optimizer sensitivity: with the lateral layer in place, TRL favors adaptive optimizers that consider second moment. Further work is required to reduce optimizer sensitivity or to develop lateral updates that are robust under SGD.

5.2 Future work

Interesting research hypothesis/directions building on top of this paper include:

- Implement a true online learning model that uses TRL.
- Measure TRL performance for sequences of real data or alternative orderings.
- Scale evaluation to natural video and to larger image benchmarks with matched compute budgets.
- Adapt TRL to recurrent and spiking architectures and investigate whether temporal traces and spike timing can realize TRL-like regularizers natively.
- Explore alternatives to the lateral layer that reduce optimizer sensitivity while retaining covariance control.
- Examine streaming, asynchronous implementations that use the one-step-shifted lateral input to reduce serialization cost.

6 Conclusion

We presented Temporal Regularized Learning, an approach that adapts variance, invariance, and covariance regularizers to neuron-level temporal losses. TRL produces useful encoders in small-scale controlled tests and creates specialized receptive fields under covariance pressure. The method emphasizes locality and streaming compatibility and therefore maps naturally to constrained hardware targets. TRL builds a conceptual bridge between modern regularized self-supervised learning and classical trace-based ideas in temporal representation learning or Slow Feature Analysis. We make the code and experiment scripts available and provide reproducibility details in the appendix.

References

[1] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations. *arXiv preprint arXiv:2002.05709*, 2020.

[2] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation Learning with Contrastive Predictive Coding. *arXiv preprint arXiv:1807.03748*, 2018.

[3] Adrien Bardes, Jean Ponce, and Yann LeCun. VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning. *arXiv preprint arXiv:2105.04906*, 2022.

[4] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow Twins: Self-Supervised Learning via Redundancy Reduction. *arXiv preprint arXiv:2103.03230*, 2021.

[5] Xinlei Chen and Kaiming He. Exploring Simple Siamese Representation Learning. *arXiv preprint arXiv:2011.10566*, 2021.

[6] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, koray kavukcuoglu, Remi Munos, and Michal Valko. Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning. *arXiv preprint arXiv:2006.07733*, 2020.

[7] Giacomo Indiveri and Shih-Chii Liu. Memory and information processing in neuromorphic systems. *Proceedings of the IEEE*, 103(8):1379–1397, 2015.

[8] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Maghazeh, Shirley Miao, and other. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[9] Laurenz Wiskott and Terrence J Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural computation*, 14(4):715–770, 2002.

[10] P Földiák. Learning invariance from transformation sequences. *Neural computation*, 3(2):194–200, 1991.

[11] Erkki Oja. A simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

[12] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. SVCCA: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in neural information processing systems*, pages 6076–6085, 2017.

[13] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pages 3519–3529. PMLR, 2019.

[14] Timothy P Lillicrap, Daniel Cownden, and Colin J Tweed, Douglas B nd Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1):1–10, 2016.

[15] Peter Földiák. Forming sparse representations by local anti-Hebbian learning. *Biological cybernetics*, 64(2):165–170, 1990.

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[17] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in neural information processing systems*, pages 4148–4158, 2017.

[18] Geoffrey E Hinton. The Forward-Forward Algorithm: Some Preliminary Investigations. *arXiv preprint arXiv:2212.13345*, 2022.

[19] Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

[20] Yann LeCun, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu Jie Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[21] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[22] David Lipshutz, Michael Fauth, Alexander D'Amour, Matt Chalk, and David Sussillo. Bio-SFA: a biologically plausible neural network for Slow Feature Analysis. In *Advances in Neural Information Processing Systems*, volume 33, pages 17758–17769, 2020.

[23] Simon M Stringer, G Perry, Edmund T Rolls, and JH Proske. Invariant object recognition in the visual system with novel views of 3D objects. *Neural Networks*, 15(10):1235–1246, 2002.

[24] Gina G Turrigiano. The self-tuning neuron: synaptic scaling of excitatory synapses. *Cell*, 135(3):422–425, 2008.

[25] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[26] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[27] Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

A Appendix

The appendix collects derivations, exact run details, additional figures, and tables. Below are the markers and explicit todos to fill.

A.1 Reproducibility details

All of the runs are contained in the publicly available repository. Different runs are configured by modifying the unified configuration class with functions at runtime. The setup is configured and the run function called in `train.py`. Depending on the configuration, we use multiple modifying functions. The configuration is set up for a TRL-S run by default. These functions map to the setups in the following way:

Table 4: Configuration Functions for Different Experimental Setups

Function	Description
`long_training`	60 Epoch long training used for the experiments
`standard_setup`	TRL setup
`eqprop_scale_network`	Network layout for comparison to Equilibrium Propagation, i.e. 3 hidden layers with 500 units respectively.
`ff_scale_network`	Network layout for comparison to the Forward Forward procedure, i.e. 4 hidden layers with 2000 units respectively.
`aug_and_rbn_setup`	Augmentations and batchnorm activated.

Backpropagation was tested in separate files, included in the `comparison` folder. We ran the experiments on seeds 42 to 46, inclusive.

A.2 Runs

Hidden Layers	Aug & BN	42	43	44	45	46	Mean	Error
512, 256	Y	98.77	98.94	98.91	98.85	98.98	98.89	1.11
	N	97.70	97.93	98.17	98.14	97.97	97.98	2.02
500 thrice	N	98.13	97.81	98.06	98.07	97.87	97.99	2.01
4 layers 2000 each	N	98.12	98.17	97.99	98.29	98.04	98.12	1.88

Backprop runs by seed (validation accuracy in %)

Hidden Layers	Aug & BN	42	43	44	45	46	Mean	Error
512,256	Y	98.26	98.50	98.23	98.29	98.32	98.32	1.68
	N	95.87	96.10	96.23	96.05	96.02	96.06	3.94
500 thrice	N	97.04	97.04	96.84	96.83	96.94	96.94	3.06
4 layers 2000 each	N	97.74	97.68	97.85	98.01	97.90	97.84	2.16

TRL runs by seed

Hidden Layers	Aug & BN	42	43	44	45	46	Mean	Error
512,256	Y	98.06	98.12	97.86	97.89	98.07	98.00	2.00
	N	95.71	95.67	95.61	95.61	95.44	95.61	4.39
500 thrice	N	96.34	96.45	98.33	96.47	96.57	96.44	3.56
4 layers 2000 each	N	97.46	97.48	97.81	97.40	97.6	97.56	2.44

TRL-S runs by seed

Setup	Hidden Layers	Inference Params	Auxiliary Params
Backprop, FF, EqProp	512, 256	535K	-
	500 thrice	897K	-
	4 x 2000	13.59M	-
TRL	512, 256	540K	328K
	3 x 500	907K	750K
	4 x 2000	13.65M	8M
TRL-S	512, 256	535K	328K
	3 x 500	897K	750K
	4 x 2000	13.59M	8M

Parameter counts by model layout and setup: Auxiliary parameters are the lateral mappings that are optimized to match the covariance matrix. We consider sparsity when reporting them. We enforce sparsity by randomly selecting entries of the target covariance matrix that are consistently zeroed out. The largest model layout has a sparsity of 50%, others have a sparsity of 0%.

A.3 Representations and figures

Representative receptive field examples For illustration purposes, we show a neuron for every digit that fires most for this respective digit.

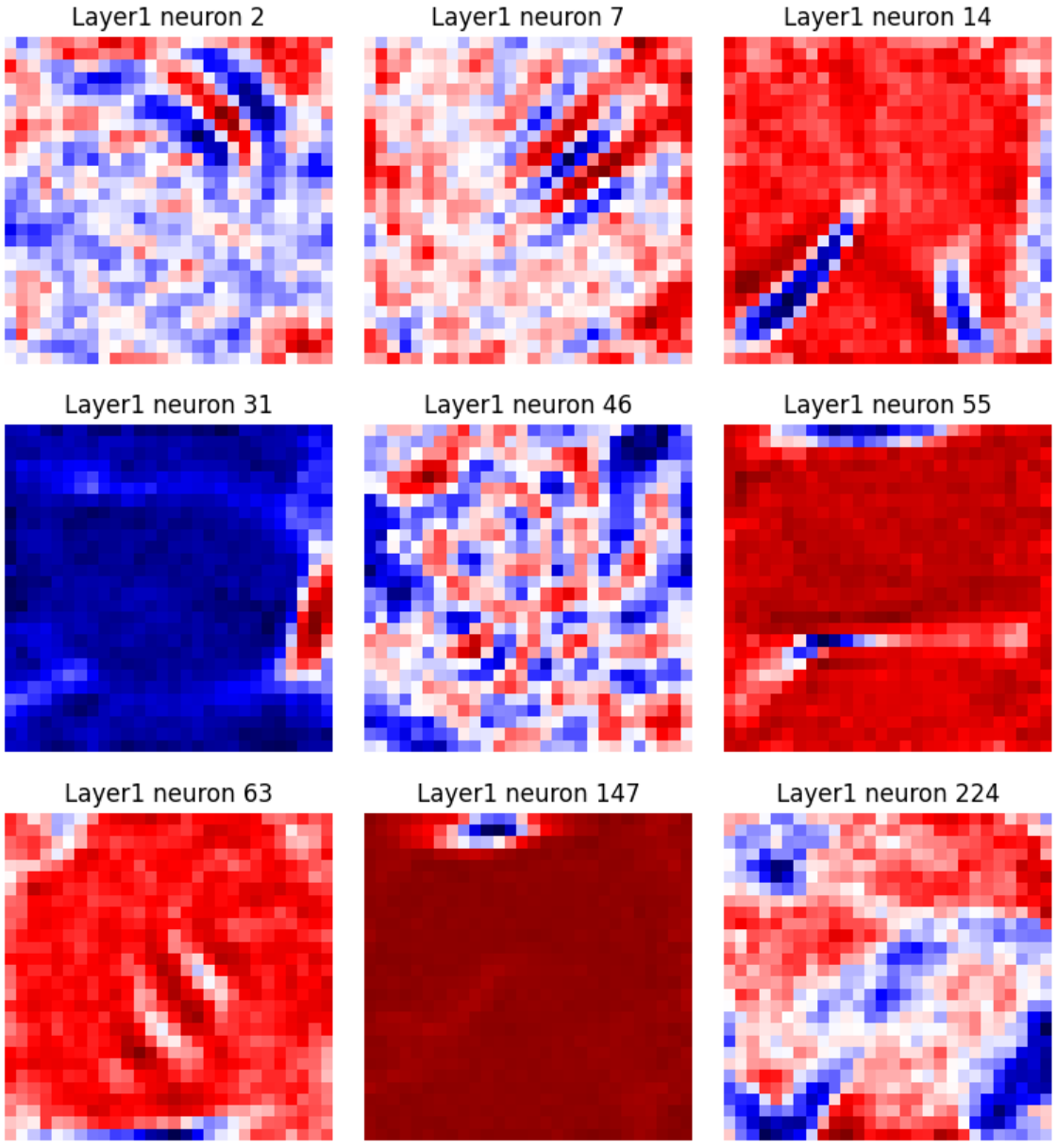

Figure 6: Representative receptive field examples from the first layer.

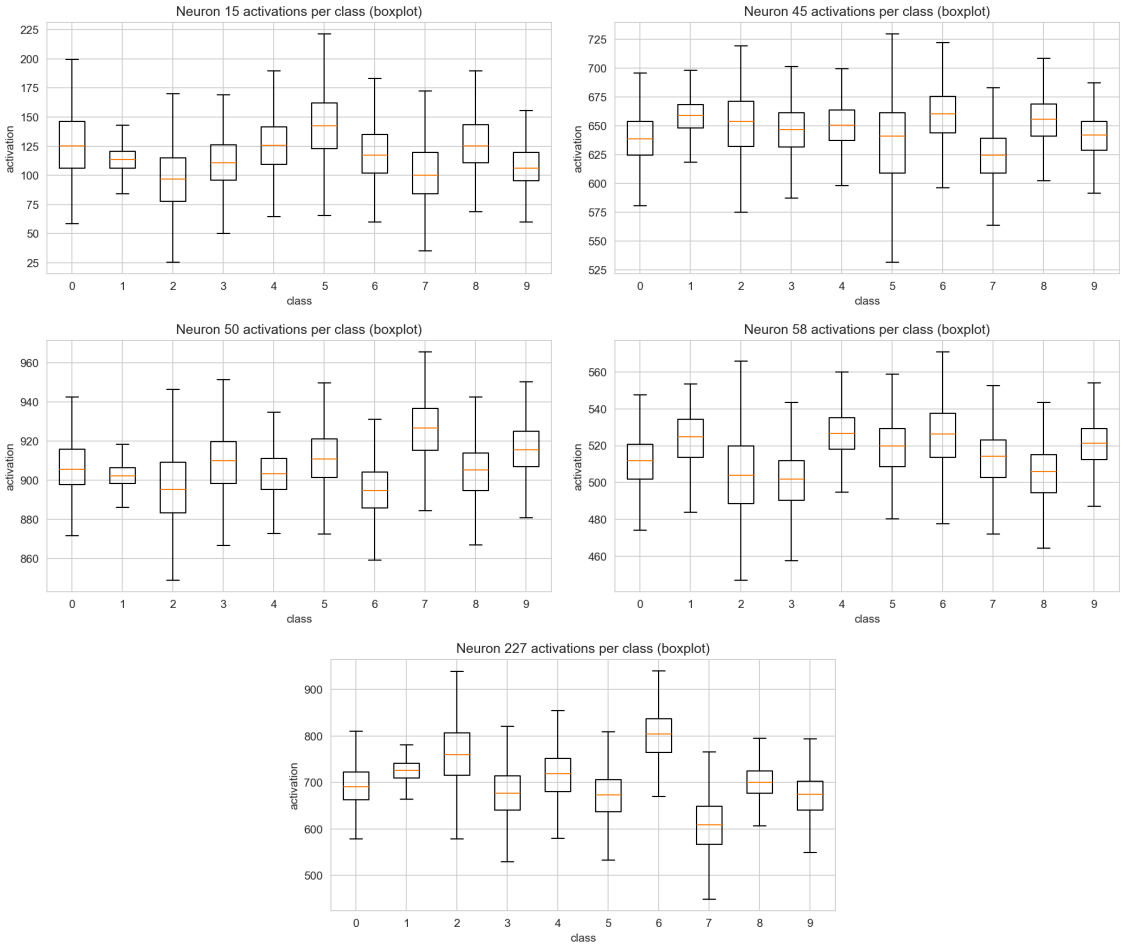

Figure 7: Examples of neurons with poor class separability.

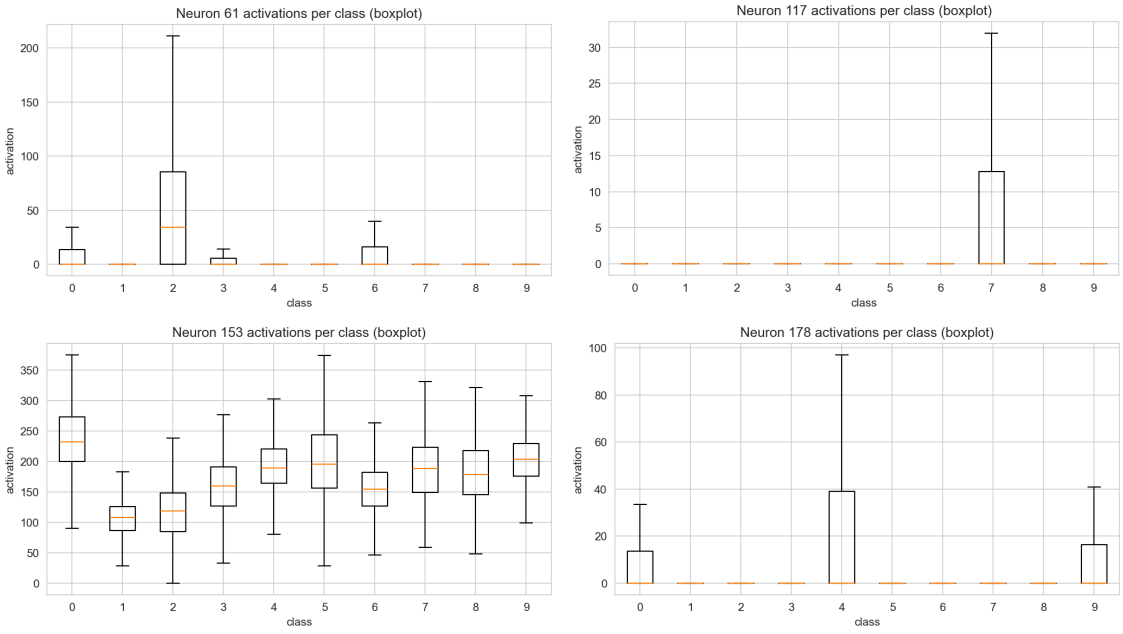

Figure 8: Examples of neurons with good class separability.


Second-layer neuron 178 — top 6 activations
lbl 0 / 491.340
lbl 0 / 396.635
lbl 0 / 374.504
lbl 0 / 367.251
lbl 0 / 366.946
lbl 0 / 361.049
Second-layer neuron 167 — top 6 activations
lbl 2 / 1479.222
lbl 1 / 1467.450
lbl 1 / 1459.965
lbl 7 / 1458.607
lbl 7 / 1452.759
lbl 1 / 1452.129
Second-layer neuron 61 — top 6 activations
lbl 2 / 619.923
lbl 2 / 613.542
lbl 2 / 608.085
lbl 2 / 606.542
lbl 2 / 598.209
lbl 2 / 593.818
Second-layer neuron 63 — top 6 activations
lbl 7 / 807.226
lbl 3 / 745.365
lbl 3 / 744.393
lbl 3 / 742.319
lbl 3 / 739.368
lbl 3 / 736.987
Second-layer neuron 36 — top 6 activations
lbl 9 / 301.584
lbl 4 / 275.292
lbl 4 / 261.218
lbl 4 / 243.881
lbl 4 / 230.173
lbl 4 / 211.176
Second-layer neuron 239 — top 6 activations
lbl 5 / 483.698
lbl 8 / 483.054
lbl 5 / 473.084
lbl 5 / 466.186
lbl 8 / 458.558
lbl 8 / 455.661
Second-layer neuron 73 — top 6 activations
lbl 6 / 460.780
lbl 2 / 456.935
lbl 6 / 455.233
lbl 6 / 415.518
lbl 6 / 370.422
lbl 6 / 368.156
Second-layer neuron 117 — top 6 activations
lbl 7 / 347.979
lbl 7 / 342.797
lbl 7 / 314.293
lbl 7 / 289.129
lbl 7 / 288.601
lbl 7 / 278.349
Second-layer neuron 153 — top 6 activations
lbl 8 / 745.306
lbl 8 / 628.394
lbl 8 / 588.684
lbl 8 / 577.117
lbl 8 / 570.061
lbl 8 / 553.144
Second-layer neuron 249 — top 6 activations
lbl 9 / 456.752
lbl 9 / 378.623
lbl 4 / 288.798
lbl 9 / 244.340
lbl 9 / 234.275
lbl 9 / 231.670