Neuhold Michael

Compact Prediction Learning

Graz University of Technology

Supervisor Dr. Robert Legenstein

Graz, January 2026

Contents

1 Introduction

Neural networks are a central class of models in modern machine learning and form the basis of the methods studied in this thesis. Deep neural networks have achieved remarkable performance across a wide range of tasks. However, the internal representations learned in their hidden layers are typically highly distributed and entangled, which makes them difficult to interpret and hard to reuse across tasks. From both a scientific and practical perspective, it is therefore of interest to investigate mechanisms that encourage neural networks to develop more compact and structured internal representations while maintaining good task performance.
This bachelor thesis studies compact prediction learning, an approach that explicitly regularizes the latent representations of a neural network to encourage the reuse of a small number of internal states. The central idea is to penalize unnecessary variability in the encoder representations. By doing so, the network is encouraged to form representations that are closer to symbolic or concept-like states, rather than highly distributed activation patterns.
The goal of this bachelor thesis is to provide an empirical and conceptual analysis of compact prediction learning across different network architectures and task settings. By systematically comparing models trained with and without the compact objective, this work aims to clarify how compactness constraints influence representation structure, expressivity, and information retention in neural networks.

2 Compactness Loss

The objective of compact prediction learning is to encourage a neural network to reuse a small number of internal representations while still solving the given prediction task accurately. To this end, the network is decomposed into an encoder $g(x)$, which maps inputs into a latent space, and a task-specific output head f, such that the overall model computes $f(g(x))$. In addition to the task loss, a compactness loss is applied directly to the latent representations. Given a set of input patterns $\{x^{(k)}\}$, the compactness loss is defined as

$$E = \sum_{k \neq l} \Big(1 + \log \sum_{i} (|g_i(x^{(k)}) - g_i(x^{(l)})|)\Big),$$

where g_i denotes the i-th component of the latent representation produced by the encoder g.

This loss penalizes pairwise differences between latent representations. If two inputs are mapped to similar encoder states, the inner sum is small and the contribution to the loss is close to zero. Assigning identical or very similar latent codes to multiple inputs is therefore inexpensive. In contrast, mapping inputs to many distinct latent states increases the loss, encouraging the encoder to reuse a limited set of representations whenever possible.

The use of the ℓ_1 distance promotes sparse differences between latent states, while the logarithmic term prevents large distances from dominating the optimization. As a result, the loss remains sensitive to small variations between similar representations, but saturates for large distances, leading to stable gradients during training.

The compactness loss is invariant under permutations of the encoder dimensions. Only the relative distances between latent representations matter, not which specific neuron encodes a given latent feature. Consequently, multiple equivalent solutions exist that differ only by a permutation of latent dimensions.

During training, the compactness loss is combined with a task-specific loss term,

$$E_{\text{total}} = E_{\text{task}} + \lambda \cdot E_{\text{compact}},$$

where the weighting factor λ controls the trade-off between task performance and representation compactness.

3 Simple networks

3.1 Introduction

In this part of the bachelor thesis, the goal is to reproduce experimental results from the work of Dr. Robert Legenstein on *Compact Prediction Learning*. The implementation was carried out in PyTorch. The network consists of a small encoder that transforms each input pattern into a latent vector, and a linear head that predicts the corresponding output. Two losses were combined. A regression loss to minimize the output error and a cluster loss to push the encoder states toward a minimal set of distinct representations.
Each experiment used a different synthetic task to test how well compact representations can form under various functional mappings.

3.2 Architecture

The network used in all experiments is deliberately shallow and simple. An encoder layer $g : \mathbb{R}^n \to \mathbb{R}^m$ maps each input pattern $x^{(k)}$ to a latent state $z^{(k)} = g(x^{(k)}) \in \mathbb{R}^m$. On top of this encoder, a linear output layer $f : \mathbb{R}^m \to \mathbb{R}$ produces the final prediction $\hat{y}^{(k)} = f(z^{(k)})$. The encoder is implemented as a single linear layer without bias, the head is either purely linear or linear followed by a ReLU, depending on the task.
Training minimises a combination of a standard regression loss and the compactness loss on the latent states:

$$E_{\text{total}} = E_{\text{reg}} + \lambda\, E_{\text{compact}}\,.$$

The regression term

$$E_{\text{reg}} = \sum_k \big(\hat{y}^{(k)} - y^{(k)}\big)^2$$

penalises deviations between the network output $\hat{y}^{(k)}$ and the target $y^{(k)}$.
The clustering term encourages the encoder to reuse a small set of latent codes across inputs. For latent states $z^{(k)}$ and $z^{(\ell)}$ the pairwise contribution is based on

their ℓ_1 distance, leading to

$$E_{\text{compact}} = \sum_{k \neq l} \log\Big(1 + \sum_i (|z_i^{(k)} - z_i^{(l)}|)\Big),$$

(see Section 2).

3.3 Tasks and Results

This section introduces the state difference matrix and presents the experimental tasks together with their corresponding results.

3.3.1 State difference matrix

To analyse the structure of the learned latent representations, we compute a state difference matrix, also known as a Representational Dissimilarity Matrix (RDM). This concept follows the framework of representational similarity analysis (Kriegeskorte et al. (2008)).
Given a set of n input samples $\{x^{(1)}, \ldots, x^{(n)}\}$ and their latent representations $\{z^{(1)}, \ldots, z^{(n)}\}$ with $z^{(k)} = g(x^{(k)}) \in \mathbb{R}^m$, the state difference matrix $SM \in \mathbb{R}^{n \times n}$ is defined by

$$SM(i,j) = \|z^{(i)} - z^{(j)}\|_1 = \sum_{p=1}^{m} \left| z_p^{(i)} - z_p^{(j)} \right|,$$

where $z_p^{(i)}$ denotes the p-th component of the latent representation z with the i-th input sample.
By construction, the diagonal elements satisfy $SM(i,i) = 0$, since each state is compared with itself. Furthermore, the matrix is symmetric, $SM(i,j) = SM(j,i)$, which results in a mirrored structure along the main diagonal. Low values indicate similar internal representations, whereas large values correspond to strongly separated latent states.

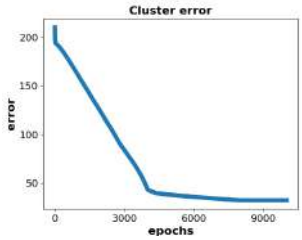
3.3.2 First Component Task

The target function for the first component task is defined as

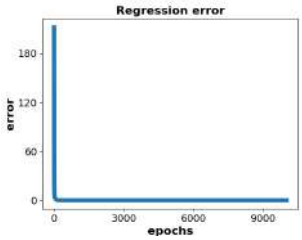
$$t_{\text{first}}(\mathbf{x}) = x^{(0)},$$

where $\mathbf{x} = (x^{(0)}, x^{(1)}, x^{(2)}, x^{(3)})^\top \in \{0,1\}^4 \subset \mathbb{R}^n$ with $n = 4$. This yields a total of $2^4 = 16$ possible input patterns.
The output head is linear and predicts, for each latent representation $z = g(\mathbf{x}) \in \mathbb{R}^m$ with $m = 10$, the scalar output

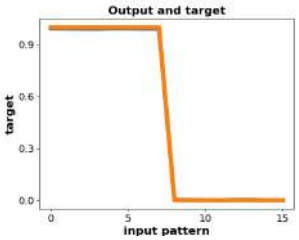
$$\hat{y} = w^\top z.$$



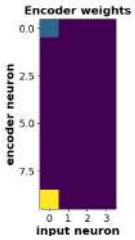
(a) Cluster error over training epochs.



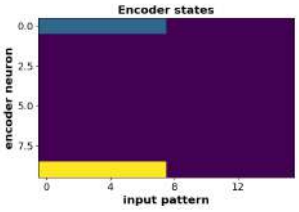
(b) Regression error over training epochs.



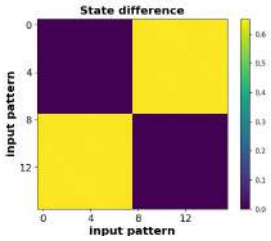
(c) Network output (orange) and target (blue).



(d) Encoder weight matrix from input to encoder neurons.



(e) Encoder state matrix from input pattern to encoder neurons.



(f) State difference matrix over input patterns.

Figure 3.1: First component task.

For the first component task (see Fig 3.1) the regression loss quickly converges to zero, indicating that the network reliably reproduces the first input component. The encoder weights exhibit two active neurons (indices 0 and 8), which encode the relevant input dimension in a sparse manner, while the remaining neurons stay effectively silent. Compared to the NumPy reference, where only neuron 9 was active, the set of active neurons is different but functionally equivalent. The architecture and the loss are invariant under permutations of the encoder neurons, and

the weights are randomly initialised. As a result, gradient descent can converge to solutions where different neuron indices take over the same role. What matters for the loss is the latent state geometry, not which specific neuron index implements a given latent code.

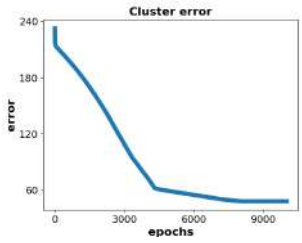
3.3.3 Additive task

The target function for the additive task is defined as

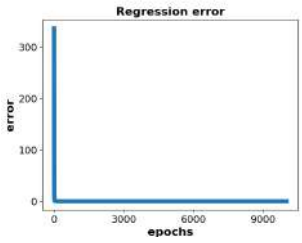
$$t_{\text{add}}(\mathbf{x}) = x^{(0)} + x^{(1)},$$

where $\mathbf{x} = (x^{(0)}, x^{(1)}, x^{(2)}, x^{(3)})^\top \in \{0,1\}^4 \subset \mathbb{R}^n$ with $n = 4$. This yields a total of $2^4 = 16$ possible input patterns.
The output head is linear and predicts, for each latent representation $z = g(\mathbf{x}) \in \mathbb{R}^m$ with $m = 10$, the scalar output

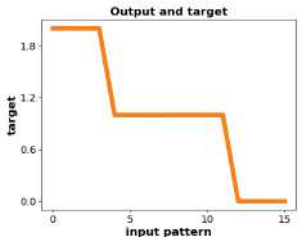
$$\hat{y} = w^\top z.$$



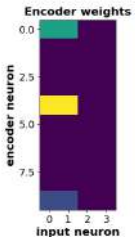
(a) Cluster error over training epochs.



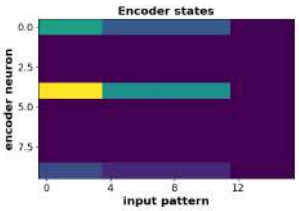
(b) Regression error over training epochs.



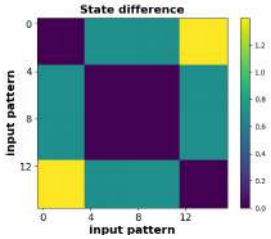
(c) Network output (orange) and target (blue).



(d) Encoder weight matrix from input to encoder neurons.



(e) Encoder state matrix from input pattern to encoder neurons.



(f) State difference matrix over input patterns.

Figure 3.2: Additive task.

For the additive task (see Fig 3.2) the regression error again drops almost to zero, showing that the network learns the mapping of the task without difficulty. The encoder weights and states reveal that only a small subset of neurons becomes active and that inputs with the same sum are mapped to similar latent states. The state-difference matrix exhibits a block structure, reflecting that patterns with identical outputs are close in latent space while patterns with different sums are pushed apart. Relative to the reference implementation, the same number of encoder neurons is active, but their indices differ. This is expected because the encoder units are symmetric: any permutation of neurons that preserves the latent distances and the readout weights produces the same loss. Random initialisation and small numerical differences steer training into slightly different, but equivalent, permutations of the same compact code.

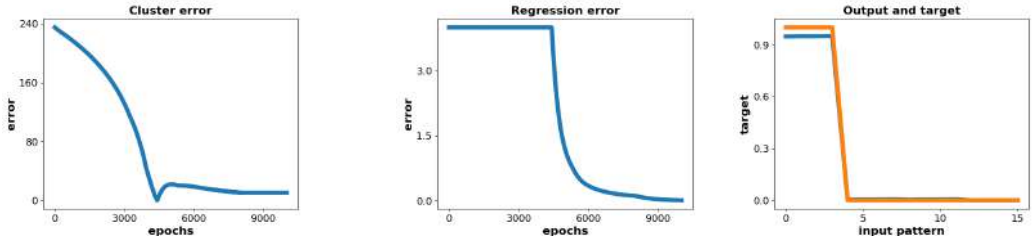
3.3.4 Logical AND task

The target function for the logical AND task is defined as

$$t_{\text{and}}(\mathbf{x}) = x^{(0)} \cdot x^{(1)},$$

where $\mathbf{x} = (x^{(0)}, x^{(1)}, x^{(2)}, x^{(3)})^\top \in \{0,1\}^4 \subset \mathbb{R}^n$ with $n = 4$. This yields a total of $2^4 = 16$ possible input patterns.
A ReLU output head is used and predicts, for each latent representation $z = g(\mathbf{x}) \in \mathbb{R}^m$ with $m = 10$, the scalar output

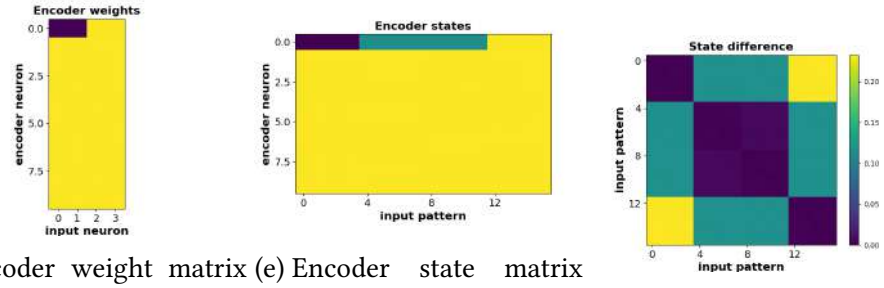
$$\hat{y} = \max\{0,\ w^\top z + b\}.$$



(a) Cluster error over training epochs.

(b) Regression error over training epochs.

(c) Network output (orange) and target (blue).



(d) Encoder weight matrix from input to encoder neurons.

(e) Encoder state matrix from input pattern to encoder neurons.

(f) State difference matrix over input patterns.

Figure 3.3: Logical AND task.

For the logical AND task (see Fig 3.3) the regression loss quickly collapses, as the problem is linearly separable and the head can implement the Boolean rule with a simple threshold on the encoder activations. The encoder states show two clearly separated clusters corresponding to the output classes 0 and 1, and the state-difference matrix contains a clean block pattern: states within each class are almost identical, while states across classes are clearly distinct. As in the other tasks, the particular encoder neuron that carries the informative signal does not match the neuron index in the reference run. Because the loss only depends on the pairwise differences between latent codes and the final output, any permutation of encoder neurons is a valid solution. Gradient descent therefore converges to a solution with the same structure but with the informative neuron at a different row index.

3.3.5 Scaled sum to 2 task

Let

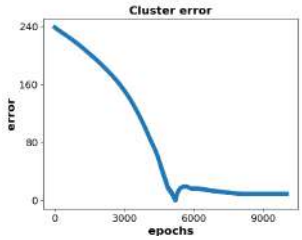
$$s(\mathbf{x}) = \sum_{i=0}^{n-1} x^{(i)},$$

where $\mathbf{x} = (x^{(0)}, x^{(1)}, x^{(2)}, x^{(3)})^\top \in \{0,1\}^4 \subset \mathbb{R}^n$ with $n = 4$. This yields a total of $2^4 = 16$ possible input patterns.
The target function for the scaled sum to 2 task is defined as

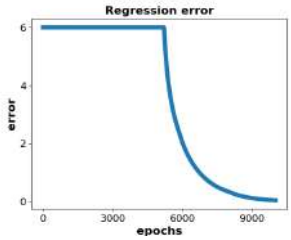
$$t_{\text{sum2}}(\mathbf{x}) = \begin{cases} 1, & \text{if } s(\mathbf{x}) = 2, \\ 0, & \text{otherwise.} \end{cases}$$

A ReLu output head is used and predicts, for each latent representation $z = g(\mathbf{x}) \in \mathbb{R}^m$ with $m = 10$, the scalar output

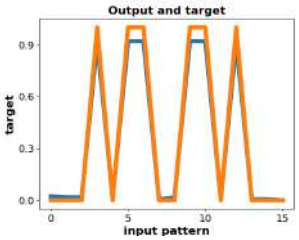
$$\hat{y} = \max\{0,\ w^\top z + b\}.$$



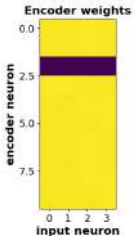
(a) Cluster error over training epochs.



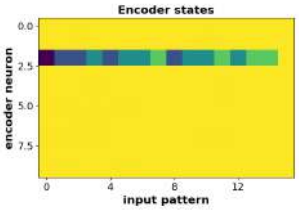
(b) Regression error over training epochs.



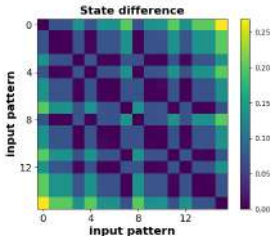
(c) Network output (orange) and target (blue).



(d) Encoder weight matrix from input to encoder neurons.



(e) Encoder state matrix from input pattern to encoder neurons.



(f) State difference matrix over input patterns.

Figure 3.4: Scaled sum to 2 task.

For the scaled sum-to-two task (see Fig. 3.4) only those input patterns whose components sum to two should produce a high output. The regression curve shows that the network gradually discovers this rule: outputs for the relevant patterns are amplified, while all other patterns are driven towards zero. The encoder states cluster the "sum equals two" inputs together, whereas all other inputs occupy a different region of latent space, which is reflected by the mixed but structured pattern in the state-difference matrix. Again, the number of active encoder neurons

matches the reference solution, but the specific indices of these active units are shifted. This difference is a consequence of the permutation symmetry of the encoder layer and the random initialisation; the optimisation only fixes which latent codes exist and how far apart they are, not which neuron index represents which code.

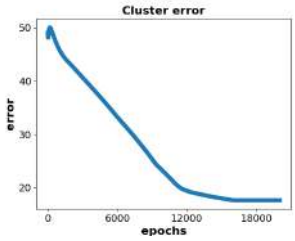
3.3.6 Linear mapping task

In the linear mapping task, each input pattern represents a discrete position along a one-dimensional track. The inputs are encoded as one-hot vectors to one hot encoded vectors (Samuels (2024)), and multiple neighbouring positions are assigned the same target value. As a result, the task defines a piecewise constant mapping from input position to output, where changes in the target occur only at specific locations along the track.
The inputs are one-hot encoded patterns $\mathbf{x} \in \{0,1\}^9 \subset \mathbb{R}^n$ with $n = 9$ and with $\|\mathbf{x}\|_1 = 1$. Each input $x^{(k)}$ is mapped to a scalar target

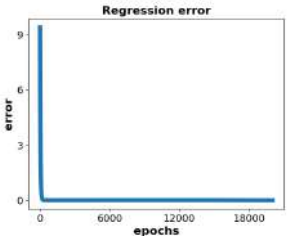
$$t_{\text{lin}}(x^{(k)}) \in \{1,2,3\}, \qquad y = [1,1,1,2,2,2,3,3,3].$$

The output head is linear and predicts, for each latent representation $z = g(\mathbf{x}) \in \mathbb{R}^m$ with $m = 10$, the scalar output

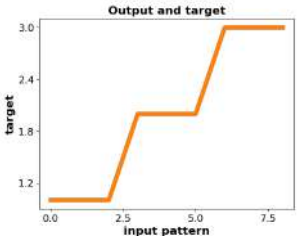
$$\hat{y} = w^\top z.$$



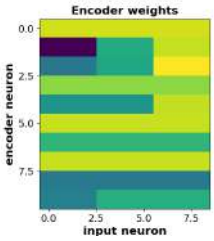
(a) Cluster error over training epochs.



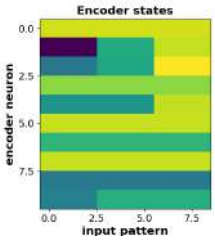
(b) Regression error over training epochs.



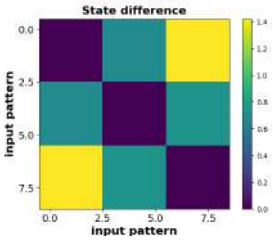
(c) Network output (orange) and target (blue).



(d) Encoder weight matrix from input to encoder neurons.



(e) Encoder state matrix from input pattern to encoder neurons.



(f) State difference matrix over input patterns.

Figure 3.5: Linear mapping task.

For the linear mapping task (see Fig 3.5) the regression loss steadily decreases and eventually the predicted outputs closely follow the desired linear mapping over positions on the track. The encoder weights and states show a monotonic progression across patterns: neighbouring inputs along the track are mapped to neighbouring points in latent space, while more distant positions are mapped further apart. This is also visible in the state-difference matrix, which exhibits a clear banded structure consistent with an underlying one-dimensional manifold. Compared to the NumPy reference, the same number of encoder neurons become strongly tuned to position, but their row indices differ. Because the loss is invariant under renaming encoder neurons, multiple equally good solutions exist that only differ by a permutation of the latent dimensions. The PyTorch implementation converges to one such permutation, which explains why other neuron indices are active while the overall representation and performance remain the same.

4 Convolutional Neural network

4.1 Introduction

In this part of the bachelor thesis, the goal was to investigate compact prediction learning using a convolutional neural network (O'Shea and Nash (2015)) on the pairwise MNIST[1] comparison task.
Each input consists of two MNIST images $x^{(i)}$ and $x^{(j)}$, concatenated along the spatial dimension. The corresponding target is defined as

$$y(x^{(i)}, x^{(j)}) = \begin{cases} 1 & \text{if } C^{(i)} = C^{(j)}, \\ 0 & \text{otherwise}, \end{cases}$$

where $C^{(i)}$ and $C^{(j)}$ denote the digit classes of the two images.
The network is trained to predict whether both digits belong to the same class. The compact objective is applied to the penultimate layer $g(x)$, which serves as the latent representation used by the final output neuron. Two models are trained: a baseline model without the compact objective and a model with compactness regularization.
After training, the learned representations are analyzed in three ways. First, the dimensionality of the penultimate representations is examined to quantify how the compact objective affects the effective dimensionality of the latent space. Second, the network weights are frozen and an auxiliary softmax classifier is trained on $g(x)$ to predict the class of the first digit, and in a separate experiment the class of the second digit. This auxiliary task evaluates how much digit-specific information is retained in the representation. Finally, the geometry of the latent space is analyzed using visualization techniques to compare the structure of representations learned with and without the compact objective.

4.2 Architecture

The convolutional neural network used in this experiment processes two concatenated MNIST digits, resulting in an input of size $1 \times 28 \times 56$. It consists of two

[1] https://www.kaggle.com/datasets/hojjatk/mnist-dataset

convolutional layers followed by ReLU activations and 2×2 max-pooling. The first convolutional layer uses 32 filters, and the second uses 64 filters. After the feature extraction, the output is flattened and passed to a fully connected penultimate layer of dimension 128, which forms the latent representation $g(x)$. Finally, a single linear output neuron predicts whether two digits belong to the same class. The compact loss is applied on the activations of the penultimate layer $g(x)$.
Training minimises a combination of a standard classification loss and the compactness loss on the latent states:

$$E_{\text{total}} = E_{\text{cls}} + \lambda\, E_{\text{compact}}\,.$$

The classification term is given by a binary cross-entropy loss with logits (Goodfellow et al. (2016)),

$$E_{\text{cls}} = -\sum_{k}\Big[y^{(k)}\log\sigma(\hat{y}^{(k)}) + \big(1-y^{(k)}\big)\log\big(1-\sigma(\hat{y}^{(k)})\big)\Big],$$

which penalises incorrect predictions of whether the two input digits belong to the same class. The compactness term encourages the encoder to reuse a small set of latent codes across inputs. For latent states $z^{(k)}$ and $z^{(\ell)}$, the pairwise contribution is based on their ℓ_1 distance, leading to

$$E_{\text{compact}} = \sum_{k\neq\ell}\log\Big(1+\sum_{i}|z_i^{(k)} - z_i^{(\ell)}|\Big),$$

(see Section 2).

4.3 Results

This section presents the experimental results and analyzes the learned latent representations in terms of dimensionality, retained class information, and latent space geometry.

4.3.1 Dimensionality

Although the penultimate layer $g(x)$ has a fixed dimensionality of 128, the effective dimensionality of the learned representation can be significantly lower. To quantify this, a Principal Component Analysis (PCA) (Bishop (2006), scikit-learn developers (2024a)) was applied to the embeddings $g(x)$ extracted from the trained network.
As a quantitative measure, the number of principal components required to explain

90%, 95%, and 99% of the total variance was computed. In addition, the effective dimensionality was estimated using the participation ratio (Gao (2017))

$$d_{\text{eff}} = \frac{(\sum_i \mu_i)^2}{\sum_i \mu_i^2},$$

where μ_i denote the eigenvalues of the covariance matrix of $g(x)$.
For the baseline model without compactness regularization ($\lambda = 0$), the effective dimensionality remains relatively high. For pairs belonging to the same class, approximately 11 principal components are required to explain 95% of the variance, while for pairs of different classes this number increases to around 13. The corresponding effective dimensionality values are $d_{\text{eff}} \approx 3.5$ for same-class pairs and $d_{\text{eff}} \approx 6.9$ for different-class pairs. This indicates that the representation utilizes several independent directions to encode detailed information about the input digits.
In contrast, when applying the compact objective ($\lambda = 0.01$), the dimensionality of the representation collapses drastically. For same-class pairs, more than 95% of the variance is already captured by a single principal component, and even for different-class pairs fewer than six components are sufficient. The effective dimensionality drops to approximately $d_{\text{eff}} \approx 1.1$ for same-class pairs and $d_{\text{eff}} \approx 1.8$ for different-class pairs. This demonstrates that the compact loss enforces a highly compressed latent space, effectively restricting the representation to a very low-dimensional manifold.

4.3.2 Auxiliary Digit Classification

To further analyze which information is preserved in the learned representation, an auxiliary classification experiment was conducted. After training the pairwise network, the weights of the convolutional backbone were frozen, and the penultimate representation $g(x)$ was used as fixed input to a softmax classifier. Two separate classifiers were trained: one to predict the class of the first digit and one for the second digit.
For the baseline model without compactness regularization ($\lambda = 0$), both auxiliary classifiers achieved accuracies of approximately 86-87%. This indicates that the latent representation $g(x)$ still contains substantial class-specific information about the individual digits, even though the network was only trained on the pairwise comparison task.
In contrast, for the model trained with the compact objective ($\lambda = 0.01$), the auxiliary classifiers perform at chance level (approximately 10% accuracy). This

shows that the compact loss effectively removes digit-specific information from the representation, retaining only the information necessary to solve the original *same vs different* task.

4.3.3 t-SNE

t-SNE (t-distributed Stochastic Neighbor Embedding) is a non-linear dimensionality reduction method used to visualize high-dimensional data, and is particularly useful for analyzing internal representations learned by convolutional neural networks. In this context, t-SNE is commonly applied to feature vectors extracted from hidden or penultimate layers to qualitatively inspect the structure of the learned representation space (Maaten and Hinton (2008)).
t-SNE maps pairwise distances between CNN feature vectors in the high-dimensional latent space to probabilities that reflect neighborhood relationships. A low-dimensional embedding is then optimized such that these neighborhood probabilities are preserved as closely as possible. Feature vectors that are similar are expected to form compact clusters, while dissimilar representations are mapped farther apart (Maaten and Hinton (2008)), scikit-learn developers (2024b).

4.3.4 Latent space geometry

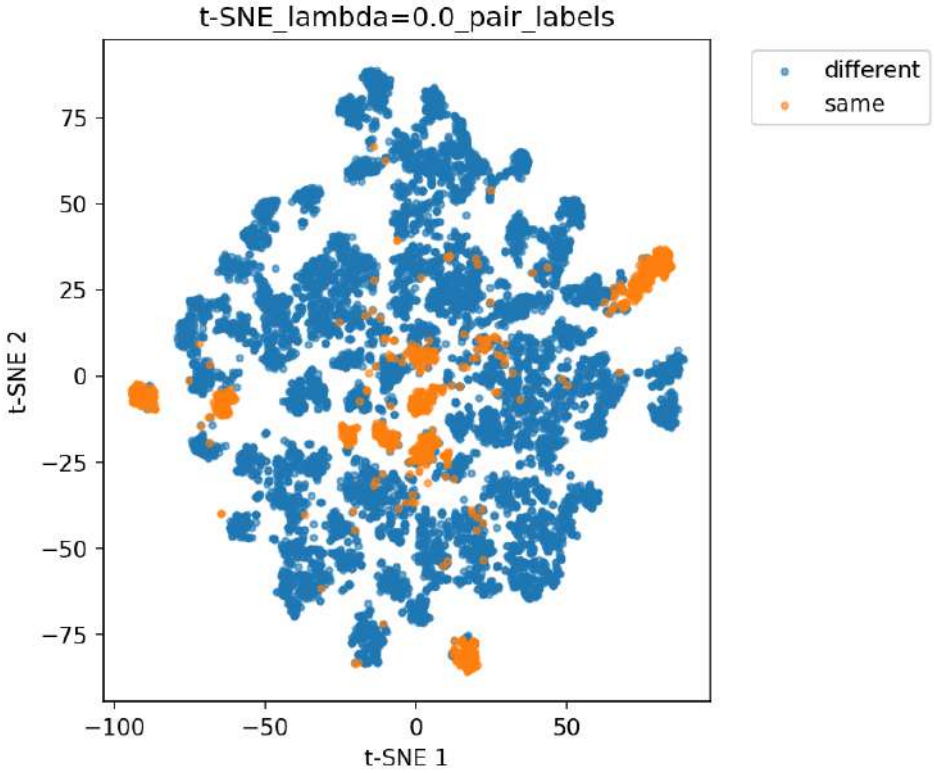


Figure 4.1: Pair labels with $\lambda = 0$.

The model without the compact objective (see Fig. 4.1) ($\lambda = 0$), the training and test accuracy quickly increased from 90% to nearly 98% after eight epochs, indicating that the network easily distinguishes whether two digits belong to the same class. In the corresponding t-SNE (see Section 4.3.3) plot, several distinct orange clusters (class = 1, equal digits) can be observed, separated by broader blue regions (class = 0, different digits). This structure suggests that the representations in the penultimate layer remain more distributed, allowing the network to exploit fine-grained class-specific differences to maximize accuracy.
Result in last epoch:
Epoch 8 - train loss: 0.0535 acc: 98.15% - test loss: 0.0539 acc: 98.19%

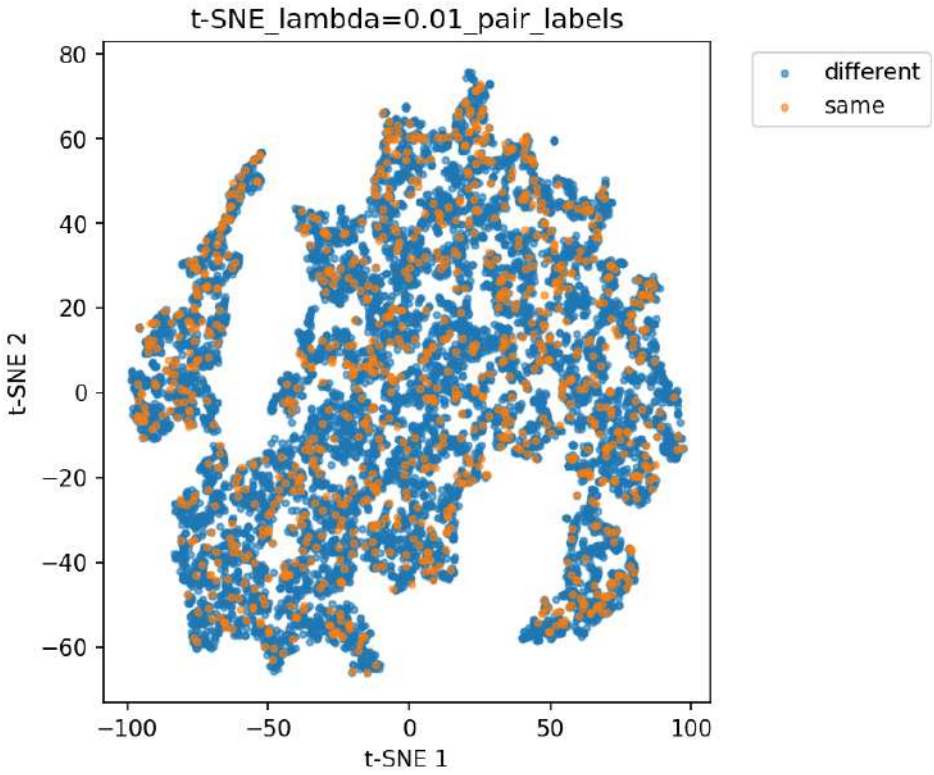


Figure 4.2: Pair labels with $\lambda = 0.01$.

The model with the compact objective (see Fig. 4.2) ($\lambda = 0.01$), the training and test accuracy stabilizes around 90%, slightly lower than the baseline. This drop is expected, as the compact loss acts as a regularization term that penalizes large distances between embeddings, encouraging a more uniform and compact latent space. The t-SNE visualization confirms this effect. The overall structure becomes denser and more homogeneous, and the distinct clusters seen in the baseline merge into a single, compressed manifold. This shows that the compact objective successfully enforces tighter and more structured representations in the penultimate layer, reducing variability within classes while maintaining sufficient separability for accurate predictions.
Result in last epoch:
Epoch 8 - train loss: 0.3247 acc: 90.10% - test loss: 0.3133 acc: 89.79%

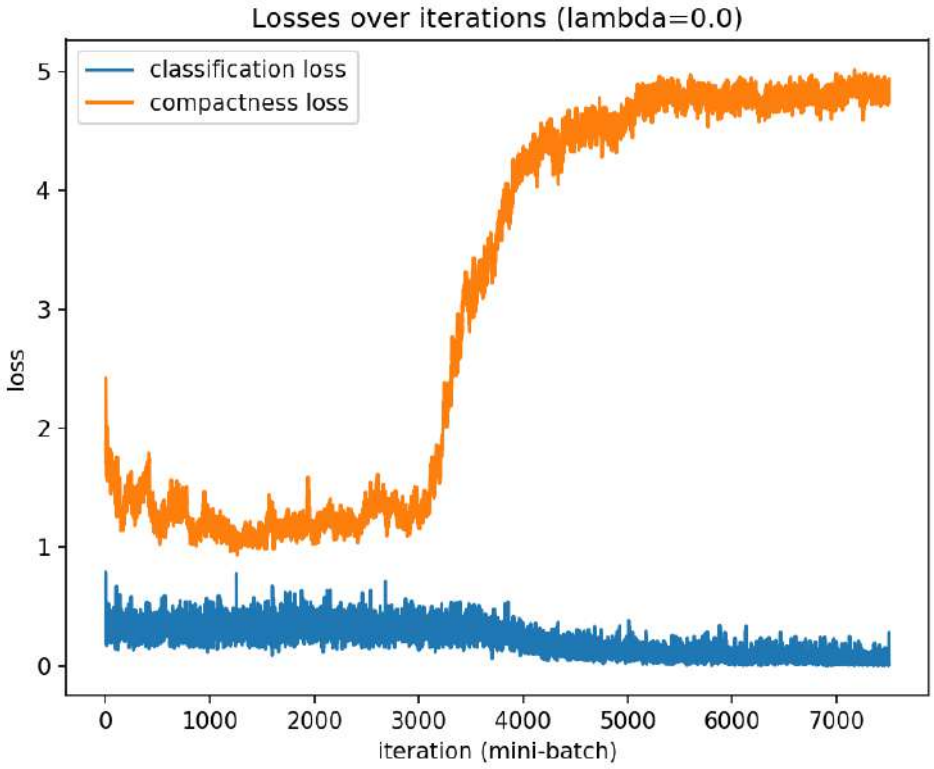


Figure 4.3: Loss with $\lambda = 0$.

For the model without compactness regularization (see Fig. 4.3), the compactness loss increases steadily during training, indicating that the embeddings become increasingly spread out. At the same time, the classification loss decreases consistently, enabling the network to achieve very high test accuracy.

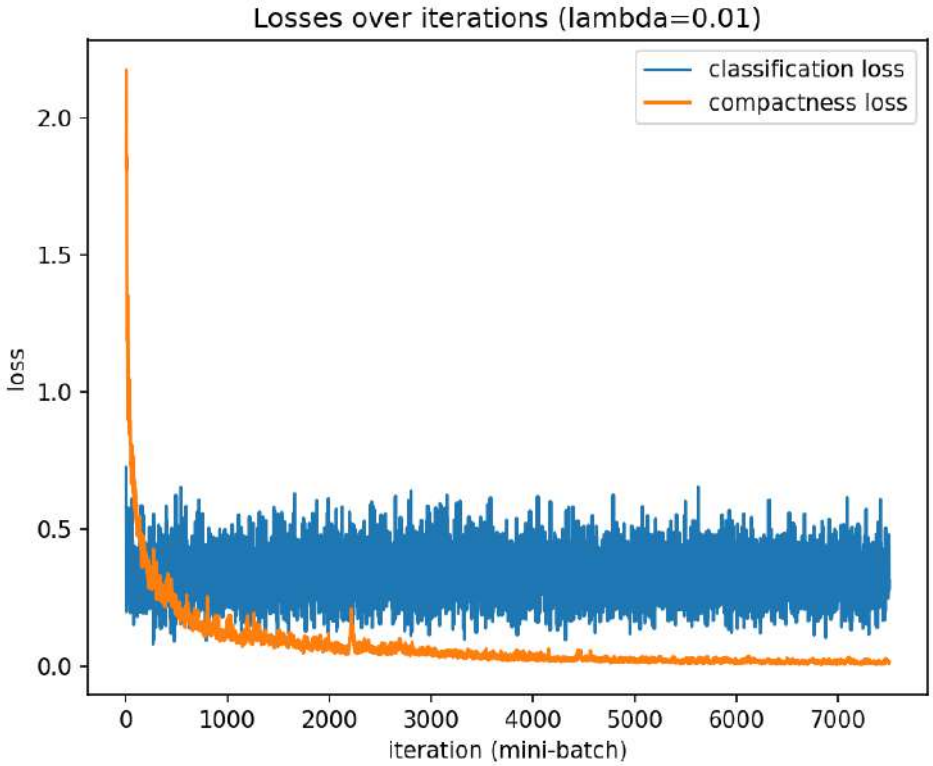
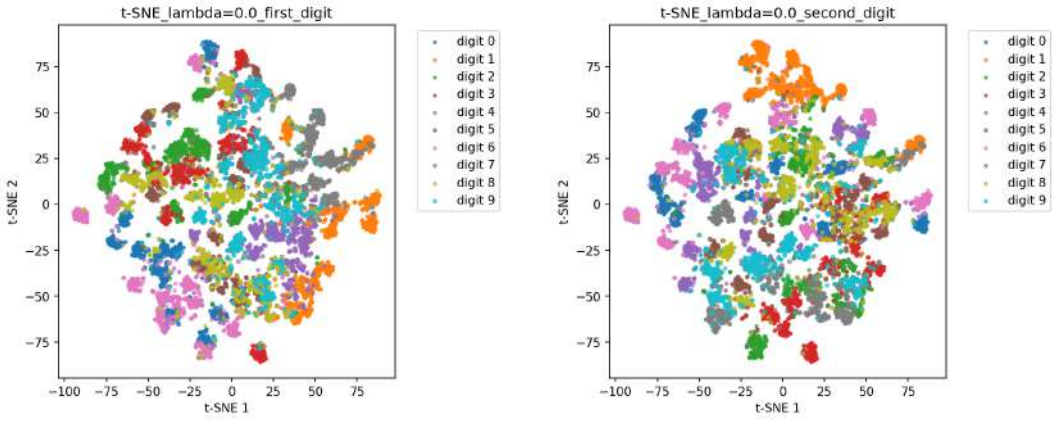


Figure 4.4: Loss with $\lambda = 0.01$.

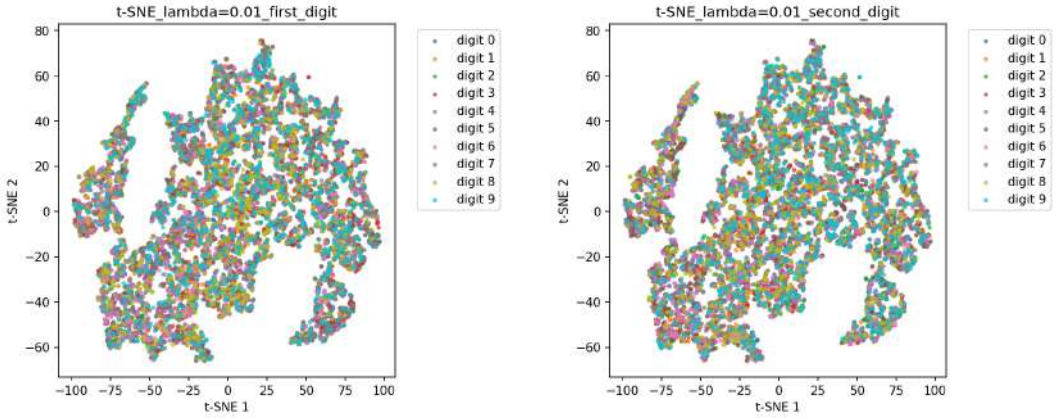
For the model trained with compactness regularization (see Fig. 4.4), the compactness loss rapidly collapses to near zero and remains minimal throughout training, showing that the embeddings are heavily compressed. The classification loss remains higher than in the baseline model, which corresponds to the reduced test accuracy of around 90%. This demonstrates that the compact objective forces the network to prioritize embedding compactness over discriminative performance.



(a) First digit. (b) Second digit.

Figure 4.5: t-SNE Plots with digit cluster $\lambda = 0$.

For the baseline model without compactness regularization (see Fig. 4.5), the t-SNE projections of the penultimate representations reveal multiple clearly separated clusters corresponding to individual digit classes represented by different colours. This indicates that the latent space preserves rich, structured information about both the first and second digit of each input.



(a) First digit. (b) Second digit.

Figure 4.6: t-SNE Plots with digit cluster $\lambda = 0.01$.

For the model trained with compactness regularization (see Fig. 4.6), the t-SNE projections show a single dense and homogeneous manifold without any digit-dependent structure. The colors are uniformly mixed, indicating that the representation $g(x)$ no longer encodes meaningful information about either digit.

5 Conclusion

This thesis investigated compact prediction learning as a mechanism to enforce structured and low dimensional latent representations in neural networks. The central question was whether compactness constraints can encourage reusable, concept-like internal states while preserving task performance.

For simple synthetic tasks, compact prediction learning consistently led to highly structured latent spaces. Across all investigated mappings, the encoder reused a small number of latent states that directly reflected task-relevant structure. The state difference matrices revealed clear block or banded patterns, indicating that inputs with identical or similar targets were mapped to nearly identical latent representations.

In contrast, the convolutional neural network experiments highlight a fundamental trade-off. While the compact objective drastically reduced the effective dimensionality of the penultimate layer, this compression came at a cost. PCA and participation ratio analyses showed a collapse of the latent space onto an almost one-dimensional manifold. Auxiliary classification experiments and t-SNE visualizations confirmed that digit-specific information was largely discarded. The network retained only the information strictly necessary to solve the pairwise same-versus-different task, while all other potentially reusable structure was removed. This behavior led to a measurable drop in classification accuracy compared to the baseline model.

Taken together, the results show that compact prediction learning is effective at enforcing minimal, task-aligned representations. When the task structure itself is simple and discrete, compactness leads to interpretable and well-organized latent codes. However, for richer perceptual tasks, the same mechanism aggressively removes information that could be useful for secondary tasks or generalization. Compact prediction learning therefore does not universally improve representations, but instead biases networks toward extreme task specialization.

In conclusion, compact prediction learning should be understood as a strong inductive bias rather than a general purpose regularizer.

Bibliography

N. Kriegeskorte, M. Mur, and P. A. Bandettini, "Representational similarity analysis-connecting the branches of systems neuroscience," *Frontiers in systems neuroscience*, vol. 2, p. 249, 2008.

A. I. J. I. B. W. Samuels, "One-hot encoding and two-hot encoding: An introduction," *ResearchGate*, 2024, https://www.researchgate.net/publication/377159812_One-Hot_Encoding_and_Two-Hot_Encoding_An_Introduction.

K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 2015. [Online]. Available: https://arxiv.org/abs/1511.08458

I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org, Chapter 6.

C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006, https://www.microsoft.com/en-us/research/wp-content/uploads/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf.

scikit-learn developers, "Pca — principal component analysis," https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html, 2024, accessed: 2026-01-12.

Y. G. S. R. S. Gao, Trautmann, "A theory of multineuronal dimensionality, dynamics and measurement," 2017, https://ganguli-gang.stanford.edu/pdf/17.theory.measurement.pdf.

L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

scikit-learn developers, "Tsne," https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html, 2024, accessed: 2026-01-15.