NEURAL CLUSTERING BY PREDICTING AND COPYING NOISE

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a neural clustering approach, based on the recently proposed Noise As Targets (NAT) framework. The NAT framework allows uncovering important features of objects by fixing target representations and constraining deep features to align with them. We extend this framework and show how it can be used to cluster objects. Our approach combines a NAT loss with an auxiliary loss (such as an autoencoder loss), and iteratively applies a policy for perturbing the set of fixed representations by deleting and copying some of the elements. It does not require deciding on a specific desired number of clusters a piori. We empirically evaluate our framework by clustering both image and text datasets (including MNIST, Fashion-MNIST, the 20 newsgroups dataset and a Twitter hashtag dataset). Our method outperforms multiple baselines, including autoencoder based clustering and other state-of-the-art neural clustering methods.

1 INTRODUCTION

Clustering is one of the fundamental problems of unsupervised learning. It involves the grouping of items into clusters such that items within the same cluster are more similar than items in different clusters. Crucially, the ability to do this often hinges upon learning latent features in the input data which can be used to differentiate items from each other in some feature space. Two key questions thus arise: *How do we decide upon cluster membership?* and *How do we represent data in feature space?*

Spurred initially by studies into the division of animals into taxa Sokol & Sneath (1963), cluster analysis matured as a separate field in the following decades with the advent various models. These included distribution-based models, such as Gaussian mixture models Duda et al. (1973); density-based models, such as DBSCAN Ester et al. (1996); centroid-based models, such as k-means [1] and hierarchical models, including agglomerative Orloci (1967) and divisive models Gower (1967).

Whereas the cluster analysis community mostly concerned itself with the unsupervised learning of *cluster membership*, the deep learning community has a long history of unsupervised representation learning. Deep neural networks of various designs have recently proven to be an extremely powerful tool for a very wide range of tasks LeCun et al. (2015). One key driver of the success of neural networks across many settings is their capability of producing a strong representation, relying on complex features which are automatically identified when the network is trained to predict labels given the training data. Unfortunately, obtaining labels requires significant costly effort. Further, when the network is trained to identify a certain label, it may ignore variability between training instances that does not lead to having a different label.

The success of deep neural networks has lead to an explosion of research on applying them to many supervised learning tasks. Neural networks clearly produce very strong object representations in supervised tasks, but less research has gone into examining how they can be used for unsupervised learning. While some work has been done on neural generative models such as GANs Radford et al. (2015); Chen et al. (2016), applications of neural networks to clustering have received relatively little attention(Yang et al., 2016; Dilokthanakul et al., 2016; Xie et al., 2016).

[1] A good historical review of k-means can be found in Hans-Hermann (2008).

Researchers working on applying neural networks to unsupervised learning domains have recently proposed a method called Noise as Targets (NAT) Bojanowski & Joulin (2017), which allows constructing informative features in unsupervised settings. To evaluate the ability of the NAT framework to identify key features, the authors have relied on a transfer learning approach; they used a dataset of images, which did include a supervision signal in the form of image labels. The NAT framework was then given access only to the images themselves (with no access to the labels), and applied to extract features and build a compound representation of each object. The authors then trained another neural network to map these latent representations of images to their labels. This analysis indicates that the NAT framework can detect important features of objects without any labels, as the extracted features can then be used to predict object labels.

Our work more directly examines the question of whether neural networks can be used for object clustering. Earlier work already uses various forms of autoencoders for clustering purposes Song et al. (2013); Huang et al. (2014); Makhzani et al. (2015). In contrast, we build on the NAT framework, and propose a new neural clustering method. The heart of our approach lies in combining a NAT loss with an autoencoder loss (and potentially a discriminator loss), and iteratively applying a policy for perturbing the set of fixed representations by deleting and copying some of the elements.

We directly evaluate our approach as a clustering method, comparing its performance with existing clustering baselines on multiple datasets, including both image and text data, such as the MNIST, FashionMNIST, the 20 newsgroups datasets, and a Twitter hashtag dataset we have sourced[2].

We show that our clustering approach outperforms various strong baselines, such as various forms of autoencoders or various clustering algorithms applied directly on the raw data.

2 MODEL

We begin by discussing the use of noise as targets (NAT) introduced in Bojanowski & Joulin (2017) which is crucial to the understanding of our method. We then describe the intuition behind our approach, then proceed to describe the mechanism itself.

2.1 NOISE AS TARGETS (NAT)

Bojanowski & Joulin (2017) recently proposed new form of unsupervised learning re-posited as a supervised problem where the task is to learn an encoder function $f_\theta : X \to Z$ from inputs to a latent representation space so as to minimize the L_2 loss between representations $z_i \in Z$, where points in Z are unit normalized, and corresponding targets $y_k \in Y$ where the targets are uniformly sampled from the L_2 unit sphere (thus inhabiting the same space as the z_is). Instead of being tied to corresponding representations as in classic regression, the targets are one-to-one re-assigned to different representations after every training step so as to minimize the total sum of distances between matched pairs (where this optimal assignment is computed using a fast algorithm known as the "Hungarian method" Kuhn (1955)). This pushes the representations z_i of similar inputs $x_i \in X$ to neighborhoods of similar targets y_k.

The motivation behind using NAT was to learn unsupervised features from input data. They show their method performs on par with state-of-the-art unsupervised representation learning methods.

2.2 NOISE AS TARGETS FOR AGGLOMERATIVE CLUSTERING (NATAC)

Viewed from a clustering perspective we can think of the targets y_i as cluster centroids to which the latent representations z_i are (one-to-one) assigned. Note that although the method introduced in Bojanowski & Joulin (2017) brings the representations of similar x_i closer together (by matching and moving them closer to neighborhoods of similar targets) it cannot produce many-to-one matchings or match multiple similar z_i with a single centroid thus forming a cluster. Simply changing the re-assignment policy to allow for many-to-one matchings is difficult because it causes the model to collapse in on a single target. In this paper we use the above to propose a new form of neural clus-

[2]We release the unique IDs of the tweets used in this dataset: `https://github.com/neuralclusteringNAT/paper-resources/tree/master/tweet_clustering`

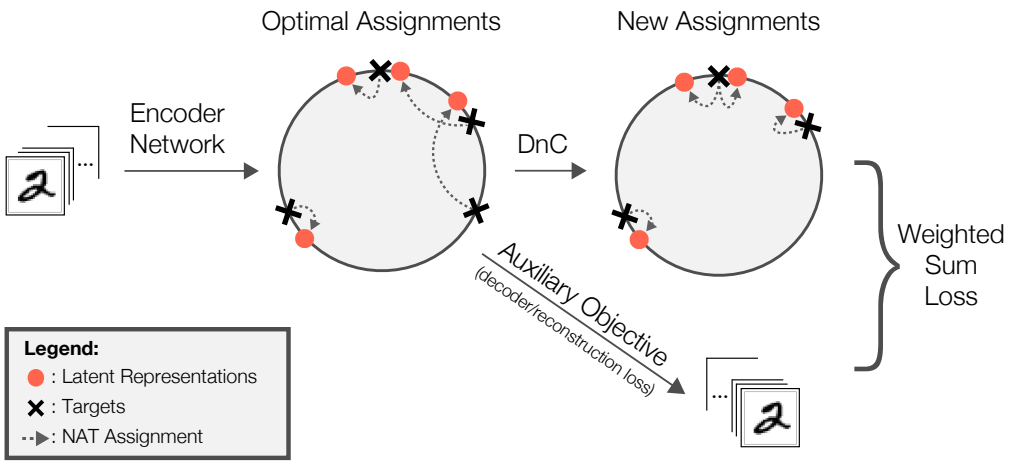

Figure 1: Training diagram for a NATAC model. In this case the model is clustering MNIST digits and uses an autoencoder loss as the auxiliary objective. The delete-and-copy policy removes the target on the bottom-right of the sphere and copies the target at the top. This leads to an agglomeration of the two latent representations at the top of the sphere into the same cluster.

tering where we progressively delete targets over time and re-assign representations to other nearby targets, all while keeping the model stable using an auxiliary objective.

Delete-and-copy To be able to cluster latent representations we need a way of assigning them to cluster centroids. We propose doing this via an additional delete-and-copy step which allows for many-to-one matchings. Similarly to the NAT method, we first assign representations $\mathbf{z}$ to targets $\mathbf{y}$ using the Hungarian method. We then remove the assigned target and reassign a *copy* of the next nearest target for each representations with some probability α as in Algorithm 1 (see also Figure 1). This has the effect of not only reassigning targets so as to minimize the distance between matched pairs, but also to encourage the model to allow similar examples to be assigned to the same target. The new assignments are denoted as y_i^{new}.

Auxiliary objective To prevent the model from collapsing to a single point, we introduce an auxiliary objective in addition to the loss between representations and targets. In our case we set the auxiliary objective $\mathcal{L}_{\text{aux}}$ to be the reconstruction loss $\sum_i \|x_i - f_{\text{dec}}(z_i)\|_2$ where f_{dec} is some decoder network. The final objective $\mathcal{L}$ is then a weighted sum of the NAT loss $\mathcal{L}_{\text{NAT}}$ (which in our case is the L_2 loss) and the auxiliary objective $\mathcal{L}_{\text{aux}}$:

$$\mathcal{L}(\theta|x;y) = \lambda \mathcal{L}_{\text{NAT}}(\theta|x;y) + \mathcal{L}_{\text{aux}}(\theta|x) \tag{1}$$

where:

$$\mathcal{L}_{\text{NAT}}(\theta|x;y) = \sum_i \|y_i^{\text{new}} - z_i\|_2 \tag{2}$$

Importantly, the auxiliary objective not only prevents the model from collapsing, it also informs how the model clusters. For example, when clustering images the reconstruction loss would encourage the model to cluster on similar pixel values.

Model definition During initialization each example x_i in the dataset is paired with a random target y_i, uniformly sampled from a $(d-1)$-sphere. The NATAC model is then trained using mini-batches from the dataset. Each training step can be broken down as follows:

1. **Forward step:** The examples $\mathbf{x}$ from a random batch of example-target pairs $\big((x_1, y_1) \dots (x_i, y_i) \dots (x_N, y_N)\big)$ are embedded using an encoder function $f_\theta : \mathbb{R}^n \to \mathbb{R}^d$ into corresponding latent representations $z_i \in \mathbb{R}^d$ where $\|z_i\| = 1$ for all i.

Algorithm 1: The delete-and-copy policy used in our experiments.

Input : A batch of latent representations $\mathbf{z}$, the optimal NAT assignment $\mathbf{y}^{\text{opt}}$, and a probability of copying α
Output: A new NAT assignment $\mathbf{y}^{\text{new}} = \left(y_1^{\text{new}} \dots y_i^{\text{new}} \dots y_N^{\text{new}}\right)$
for z_i *in* $\boldsymbol{z}$ **do**
  p := sample from $U(0, 1)$
  y^{nearest} := nearest target to z_i
  if $y_i^{opt} \neq y^{nearest}$ *and* $p < \alpha$ **then**
    (y_i^{opt} ***is then deleted, and*** $y^{nearest}$ ***is copied and assigned to*** z_i)
    $y_i^{\text{new}} := y_i^{\text{nearest}}$
  else
    $y_i^{\text{new}} := y_i^{\text{opt}}$
  end
end
return $\left(y_1^{\text{new}} \dots y_i^{\text{new}} \dots y_N^{\text{new}}\right)$

2. **Re-assignment step:** Using the Hungarian method from Kuhn (1955), the representations $\mathbf{z}$ are optimally one-to-one matched with targets $\mathbf{y}$ so as to minimize the total sum of distances between matched pairs in the batch. The newly assigned example-target pairing $\left((x_1, y_1^{\text{opt}}) \dots (x_i, y_i^{\text{opt}}) \dots (x_N, y_N^{\text{opt}})\right)$ has the permutation of labels within the batch to minimize the batch-wise loss.
3. **Delete-and-copy:** With a probability α, delete the optimal target y_i^{opt} and instead assign the nearest target to z_i (which may be the same target) for each example-target pair in the batch producing $\left((x_1, y_1^{\text{new}}) \dots (x_i, y_i^{\text{new}}) \dots (x_N, y_N^{\text{new}})\right)$ as described in Algorithm 1 (see also Figure 1).
4. **Train and update step:** The L_2 loss between targets and latent representations is taken and combined with the auxiliary loss. Gradients w.r.t. θ are then taken and back-propagated along. Notice that although the (re)-assignment step follows a non-differentiable policy, the model is still end-to-end differentiable. Finally, the new example-target assignments are kept after the training step, and persist into the next training step where they are again reassigned.

2.3 Implementation Details

Stopping Criterion During training the number of unique targets is tracked. We stop training when the number of unique targets stops decreasing after an epoch of training.

Multi-Stage Training We found that an initial period of training where the auxiliary objective is prioritized (i.e. the NAT loss is multiplied by a very small coefficient) and the DnC policy is not used, improved overall performance. Additionally, using a gradual transition to a higher NAT loss and a more utilized DnC policy increased the stability of training. Therefore, training a NATAC model has three steps:

1. **Warm-Up Stage**: $\alpha = 0$ and λ are very small.
2. **Transition Stage**: α increases gradually from 0 to 1, λ also increases gradually to a larger value (approximately $100\times$ larger than its initial value).
3. **Clustering Stage**: $\alpha = 1$, λ is large.

Dimensionality Of The Latent Space In all of our experiments, we found that the best performing models tend to have an latent space dimensionality between $d = 6$ and $d = 12$. At dimensionalities much larger than this, the model collapses to very few points during the transition stage of training, possibly due to the high expressiveness of the latent space. On the contrary, using a very low dimensional representation results in an information bottleneck too small to sufficiently learn the auxiliary objective. For example, when trying to cluster a collection of tweets from the twitter dataset, a latent space of two dimensions was too small for a decoder to reliably reconstruct a tweet

from the latent vector. In the absence of an auxiliary objective that can be effectively learned, the centroids collapse to a single point.

3 EXPERIMENTS

In this section we describe the datasets and evaluation metrics used in our experiments followed by the presentation and analysis of the results of our model in comparison to others. The full details regarding the hyperparameters used in our experiments can be found in appendix D.

3.1 DATASETS

We evaluate the performance of our models on four different datasets - two image and two text datasets. For experiments on images we use MNIST (LeCun et al., 1998) and Fashion-MNIST (Xiao et al., 2017). For text we use 20 Newsgroups (Joachims, 1996) and a Twitter dataset which we built ourselves.

3.2 EVALUATION

Our key evaluation metric is the normalized mutual information (NMI) score (Strehl & Ghosh, 2002) which measures the information gained from knowing cluster membership whilst taking the number of clusters into account. In our experiments, we train models on the concatenation of the train and validation sets and evaluate them on the test set. This is done by computing the latent representations of test examples and then assigning them to their nearest respective centroids, then computing the NMI score. Additionally, we provide classification error scores on the MNIST dataset to compare ourselves to other related methods. The motivation behind our experiments is to see how well our model decides upon cluster membership and to see whether it learns useful latent representations.

3.3 MNIST

Introduced by LeCun et al. (1998), MNIST is canonical dataset for numerous machine learning tasks including clustering. MNIST has ten classes, corresponding to the ten digits 0-9, and contains 60,000 train and 10,000 test examples.

We train our model with an auxiliary reconstruction loss and use small convolutional architectures (see Figure 6) for our encoder and decoder networks. As points of comparison we also provide results of using k-means on the latent representations learned by our model (NATAC-K in Table 1) and k-means on representations learned by a simple autoencoder with the same encoder and decoder architecture as our model (AE-K in Table 1).

Table 1 shows that our model's performance is best when $d = 10$ and worse for much lower or higher values of d. This indicates that the dimensionality of the latent space impacts our model's performance (see Section 2.3 for further discussion).

The ability of our model to cluster MNIST examples well is shown by two keys results. First, it beats both NATAC-K and AE-K (Table 1). Second, it achieves state-of-the-art results when compared to other methods (see NMI column in Table 2). Note that the second result reported in (Yang et al., 2016) is achieved using a separate re-clustering step after training.

We note that NATAC-K beats AE-K, indicating that our model learns better representation as compared to a simple autoencoder. Consequently, NATAC achieves a state-of-the-art clustering quality on MNIST, as measured by the classification error result (see Table 2).

Finally, we discuss the number of centroids our model converges on (see Table 1). We show that our model is successfully capable of finding centroids that represent different digits, as shown in the top row of Figure 2. However, the model also learns centroids that contain very few examples for which the decoded images do not represent any handwritten digits, as shown in the second row of Figure 2. Even with these "dead centroids", the model still performs well. Interestingly, the model also differentiates between ones with different slopes, suggesting that it thinks the latent representations of these digits are sufficiently far apart to warrant splitting them into different clusters.

Table 1: NMI scores from varying the dimensionality of the latent space in the NATAC and baseline models. The baselines use K-means with the same number of clusters as the repsective NATAC model converged to. We include NATAC models with a latent dimensionality of $d = 3$, whose latent representations can be viewed without dimensionality reduction. Appendix section A contains links to the visualizations hosted on the tensorflow embedding projector.

	MNIST				**Fashion-MNIST**			
d	**Centroids**	**NATAC**	**NATAC-K**	**AE-K**	**Centroids**	**NATAC**	**NATAC-K**	**AE-K**
3	2251	0.455	0.455	0.452	2330	0.442	0.441	0.439
8	120	0.820	0.681	0.650	101	0.614	0.542	0.538
9	308	0.782	0.640	0.611	77	**0.638**	0.562	0.557
10	152	**0.878**	0.704	0.649	44	0.627	0.570	0.564
11	123	0.865	0.702	0.696	45	0.619	0.566	0.580
12	56	0.767	0.688	0.683	20	0.607	0.572	0.609

3.4 FASHION-MNIST

Introduced in (Xiao et al., 2017), Fashion-MNIST is a convenient swap-in dataset for MNIST. Instead of digits the dataset consists of ten different types of clothes. There are 60,000 train and 10,000 test examples just like in MNIST. Fashion-MNIST is generally considered more difficult than MNIST, with classifiers scoring consistently lower on it than MNIST.[3]

The model and analysis from the previous section carries over for fashion-MNIST with a few additional important points. First, the differences between NATAC-K and AE-K are less pronounced (see Table 1) in fashion-MNIST which indicates that the representations learned by NATAC in comparison to a simple autoencoder are not as important for k-means clustering. Interestingly, our model still outperforms both NATAC-K and AE-K.

Qualitatively, Figure 2 shows that the model has separated garments into slightly different categories than the labels in the dataset. For example, the most dense cluster seems to be a merging of both "pullovers" and "shirts", suggesting that the model finds it difficult to separate the two different garments[4]. Additionally, the "sandal" class has been split into two separate clusters: flip-flops and high-heel shoes with straps. This indicates that our model has found an important distinction between these two type of shoes, that the original Fashion-MNIST labels ignore. Similarly to MNIST, our model also learns "dead clusters", which the model does not decode into any discernible garment. Further visualizations of these experiments can be found appendix section A.

3.5 LARGE DOCUMENT CLUSTERING ON 20 NEWSGROUPS

Introduced in (Joachims, 1996) the 20 Newsgroups dataset is a collection of 18,846 documents pertaining to twenty different news categories. We use the commonly used 60:40 temporal train-test split. Interestingly, because of the temporal split in the data the test set contains document which differ considerably from the train set.

We use an auxiliary reconstruction loss and a two layer fully connected network for both the encoder and decoder, both with hidden layer sizes of 256 and ReLU nonlinearities. We represent each article as a binary TF vector of the 5000 most occurring words in the train set. [6]

[3]A comparison table for MNIST and Fashion-MNIST: `https://github.com/zalandoresearch/fashion-mnist#benchmark`

[4]We ourselves find it difficult to discriminate between the two categories, as the low-resolution images do not easily show whether or not the garment has buttons.

[5]Other clustering techniques in this table used 10 clusters. However, most of the examples in the dataset belong to to the top 10 most dense clusters (shown in figure 2), and many of the smaller clusters have only one example assigned to them.

[6]The documents in this dataset are long, so we opted for a feedforward design rather than an RNN (which is prone to a performance degradation as the processed sequences become longer.)

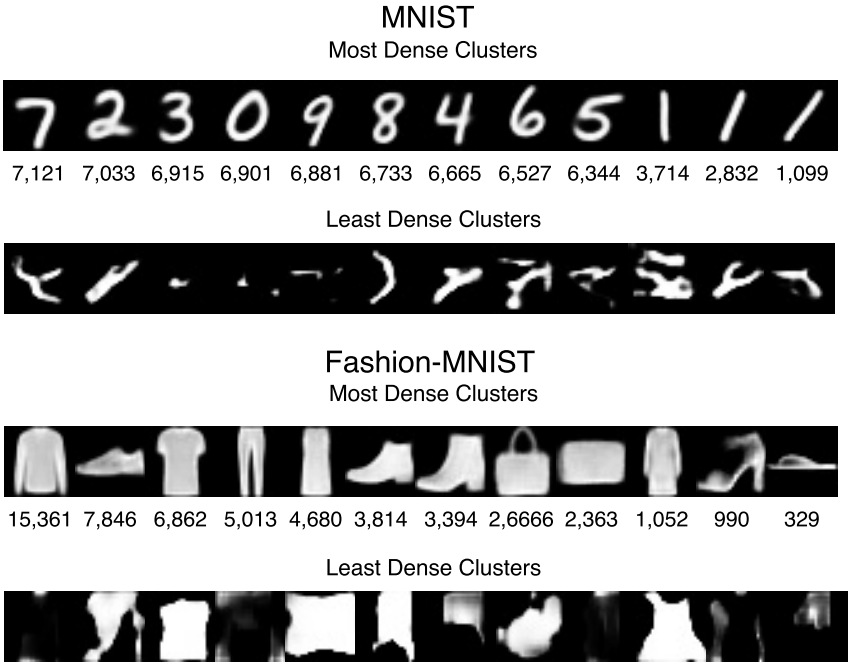

Figure 2: Outputs of the decoder when fed centroids from the MNIST experiment (top) and Fashion-MNIST experiment (bottom). The models used are the those with the highest NMI, $d = 10$ and $d = 9$ for MNIST and Fashion-MNIST respectively. The top rows show decoded images from the most dense centroids (size of the centroid's cluster shown underneath), and the bottom rows show decoded images from the least dense centroids. For both models, the bottom row of images only contains centroids with one example from the dataset assigned to them.

Table 2: Comparison of our best performing architecture on the entire MNIST dataset. NMI and classification error are calculated from the entire data set. We report the evaluation metric used by the authors of each respective model.

Clustering Algorithm	NMI	Classification Error (%)
K-Means (MacQueen et al., 1967) On Raw Pixels	0.500	41.50
DEC (Xie et al., 2016)	-	15.70
Adversarial Autoencoders (30 clusters) (Makhzani et al., 2015)	-	4.10
Deep Gaussian Mixture VAE (Dilokthanakul et al., 2016)	-	3.08
Autencoder based clustering (Song et al., 2013)	0.669	-
Task-specific Clustering With Deep Model (Wang et al., 2016)	0.651	-
Agglomerative Clustering Using Average Linkage (Jain et al., 1999)	0.686	-
Large-Scale Spectral Clustering (Chen & Cai, 2011)	0.706	-
Yang et al.	0.906	-
Yang et al. With Re-clustering	**0.913**	-
NATAC autoencoder (Converged to 61 clusters)[5]	0.912	**2.78**

Two baselines are used as points of comparison for our model. A K-means model and a spherical clustering model, a good description of which can be found in Buchta et al. (2012).

Table 3 shows how the performance of the NATAC model varies with different dimensionalities of the latent space. The best performing NATAC model with 819 centroids performs marginally better than K-Means clustering with 1000 centroids on TF-IDF vectors of the news articles. Additionally, NATAC outperforms spherical K-Means by a substantial margin.

Table 3: Clustering results for the 20News dataset.

	NATAC		**Baselines**		
d	**Centroids**	**Test NMI**	**Method**	**Centroids**	**Test NMI**
2	175	0.088	Spherical K-means	20	0.085
3	303	0.184	Spherical K-means	50	0.103
4	674	0.274	Spherical K-means	100	0.121
5	819	**0.286**	Spherical K-means	1000	0.181
6	899	0.270	TF-IDF + K-Means	20	0.065
7	766	0.244	TF-IDF + K-Means	50	0.089
8	548	0.212	TF-IDF + K-Means	100	0.198
9	424	0.114	TF-IDF + K-Means	1000	**0.284**

3.6 Clustering Tweets

To further explore the performance of our model on text data build a dataset of 38,309 ASCII-only tweets of English speaking users containing exactly one hashtag. The dataset has 647 different hashtags, with each hashtag having at least ten tweets containing the hashtag. We use 5000 of these tweets as the test set. As a preprocessing step, URLs and hashtags are replaced with special characters.

We train a character-based seq2seq autoencoder on the Twitter dataset. Just as before we use an auxiliary reconstruction loss. We set the encoder to be a bidirectional GRU with a hidden size of 64 followed by a fully connected layer which takes the GRU's final output (we also experiment with a mean pool across outputs instead of the final output) and maps it to the latent space. The decoder uses a fully connected layer to map the latent vectors to a 128 dimensional vector which is then used as the initial hidden state for the decoder GRU.

Similarly to section 3.5, we used spherical K-means and TF-IDF + K-means for comparable baselines. Rather than varying the number of clusters, we instead vary the vocabulary size for the term-frequency vectors. As the converged number of centroids for the NATAC models are very similar across different values of d (around 5000), we felt that varying the vocabulary size rather than the number of centroids would give a better range of comparable values.

As shown in table 4, spherical clustering does better on the Twitter dataset than raw TF/IDF clustering (as opposed to then on the 20 newsgroup dataset). The NATAC models with $d = 8, 10$ outperform all of the K-means baselines, while the lower dimensional NATAC models perform on par with spherical K-means. We see a small increase in the test NMI when using a larger encoding for the decoder and mean pooling to the latent representation.

4 Conclusion

We proposed a neural clustering method which works by automatically identifying key features of objects by applying a Noise As Target (NAT) procedure. NAT is a recently proposed approach which extracts features by fixing target representations and constraining deep features to align with them. We extend NAT into a full clustering framework by combining the NAT loss with an autoencoder loss, and iteratively applying a policy for deleting and copying target representations (DnC).

Our empirical evaluation shows that our framework works well, across datasets and across modalities. It improves state-of-the-art clustering performance on MNIST, without needing to receive the

Table 4: Clustering results for the twitter dataset. The K-means models are trained with 5000 centroids.

		NATAC		**Baselines**		
d	e	**Centroids**	**Test NMI**	**Method**	**Vocab.**	**Test NMI**
4	$1 \times d$	4,733	0.753	Spherical K-Means	250	0.743
6	$1 \times d$	3,838	0.742	Spherical K-Means	500	**0.746**
8	$1 \times d$	4,778	0.760	Spherical K-Means	1,000	0.740
10	$1 \times d$	4,000	0.759	Spherical K-Means	2,500	0.723
4	$16 \times d$	2,560	0.719	TF-IDF + K-Means	250	0.706
6	$16 \times d$	4,005	0.749	TF-IDF + K-Means	500	0.712
8	$16 \times d$	6,343	**0.762**	TF-IDF + K-Means	1,000	0.706
10	$16 \times d$	4,795	0.760	TF-IDF + K-Means	2,500	0.691

required number of clusters as input. Further, outperforms powerful baselines on FashionMNIST and on text datasets (20 Newsgroups and a Twitter hashtag dataset). NATAC achieves a moderate improvement over baselines on the text datasets, and its performance is very strong on the image datasets we've examined. However, NATAC does require some hyperparameters to be tuned, namely the dimensionality of the latent space and the function for the loss coefficient λ.

Several questions remain open. First, how can our framework be extended to semi-supervised scenarios? Could we use a set of labels for a small fraction of the data so as to achieve stronger clustering for a large set of unlabeled instances? Second, could one use a training method that does not require hyperparameters, and apply better perturbations for the target set than our simple DnC policy? Finally, could our framework be modified to scale better to a higher latent dimensionality?

REFERENCES

Piotr Bojanowski and Armand Joulin. Unsupervised learning by predicting noise. *arXiv preprint arXiv:1704.05310*, 2017.

Christian Buchta, Martin Kober, Ingo Feinerer, and Kurt Hornik. Spherical k-means clustering. *Journal of Statistical Software*, 50(10):1–22, 2012.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2172–2180, 2016.

Xinlei Chen and Deng Cai. Large scale spectral clustering with landmark-based representation. In *AAAI*, volume 5, pp. 14, 2011.

Nat Dilokthanakul, Pedro AM Mediano, Marta Garnelo, Matthew CH Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders. *arXiv preprint arXiv:1611.02648*, 2016.

Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. Wiley, New York, 1973.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pp. 226–231, 1996.

John C Gower. A comparison of some methods of cluster analysis. *Biometrics*, pp. 623–637, 1967.

Bock Hans-Hermann. Origins and extensions of the k-means algorithm in cluster analysis. *J. EHPS*, 4(2), 2008.

Peihao Huang, Yan Huang, Wei Wang, and Liang Wang. Deep embedding network for clustering. In *ICPR*, pp. 1532–1537. IEEE, 2014.

Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

Thorsten Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. Technical report, Carnegie-mellon univ pittsburgh pa dept of computer science, 1996.

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

Harold W Kuhn. The hungarian method for the assignment problem. *NRL*, 2(1-2):83–97, 1955.

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.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *SMSP*, volume 1, pp. 281–297. Oakland, CA, USA., 1967.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

Lt Orloci. An agglomerative method for classification of plant communities. *J.Ecol*, 1967.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

RS Sokol and Peter HA Sneath. Principles of numerical taxonomy, 1963.

Chunfeng Song, Feng Liu, Yongzhen Huang, Liang Wang, and Tieniu Tan. Auto-encoder based data clustering. In *Iberoamerican Congress on Pattern Recognition*, pp. 117–124. Springer, 2013.

Alexander Strehl and Joydeep Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec):583–617, 2002.

Zhangyang Wang, Shiyu Chang, Jiayu Zhou, Meng Wang, and Thomas S Huang. Learning a task-specific deep architecture for clustering. In *ICDM*, pp. 369–377. SIAM, 2016.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International Conference on Machine Learning*, pp. 478–487, 2016.

Jianwei Yang, Devi Parikh, and Dhruv Batra. Joint unsupervised learning of deep representations and image clusters. In *CVPR*, pp. 5147–5156, 2016.

Appendix

A VISUALIZATIONS OF MNIST/FASHION-MNIST LATENT REPRESENTATIONS

We have published the visualizations of MNIST/Fashion-MNIST representations on the tensorflow embedding projector. We recommend selecting the "Spherize data" option, as this gives a much clearer view of the embeddings. The images used are taken from the test set of MNIST and Fashion-MNIST respectively. We include the NATAC models with the highest NMI from each dataset, along with the models with $d = 3$, which require no dimensionality reduction to visualize.

- MNIST $d = 3$:
  `http://projector.tensorflow.org/?config=https://raw.githubusercontent.com/neuralclusteringNAT/paper-resources/master/mnist_embeddings/digits3/config.json`
- MNIST $d = 10$:
  `http://projector.tensorflow.org/?config=https://raw.githubusercontent.com/neuralclusteringNAT/paper-resources/master/mnist_embeddings/digits10/config.json`
- Fashion-MNIST $d = 3$:
  `http://projector.tensorflow.org/?config=https://raw.githubusercontent.com/neuralclusteringNAT/paper-resources/master/mnist_embeddings/fashion3/config.json`
- Fashion-MNIST $d = 9$:
  `http://projector.tensorflow.org/?config=https://raw.githubusercontent.com/neuralclusteringNAT/paper-resources/master/mnist_embeddings/fashion9/config.json`

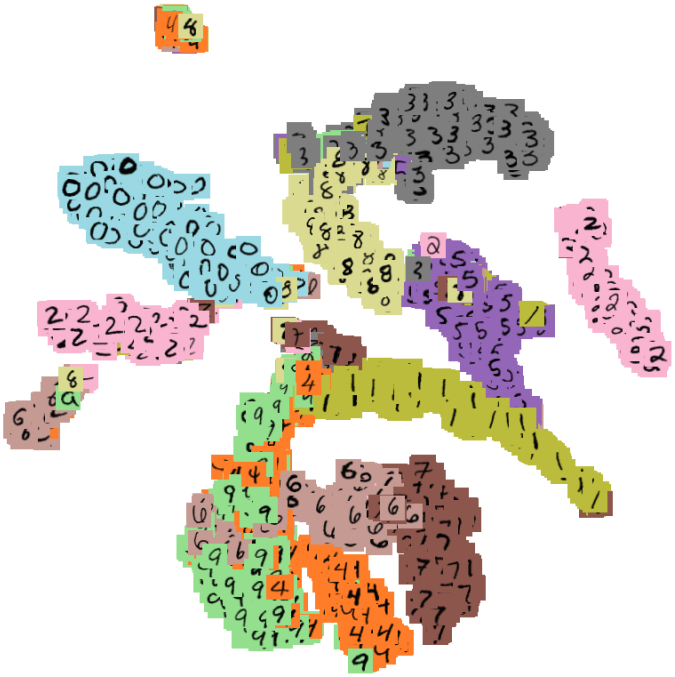

Figure 3: T-SNE visualization of the best performing NATAC model for MNIST $d = 10$

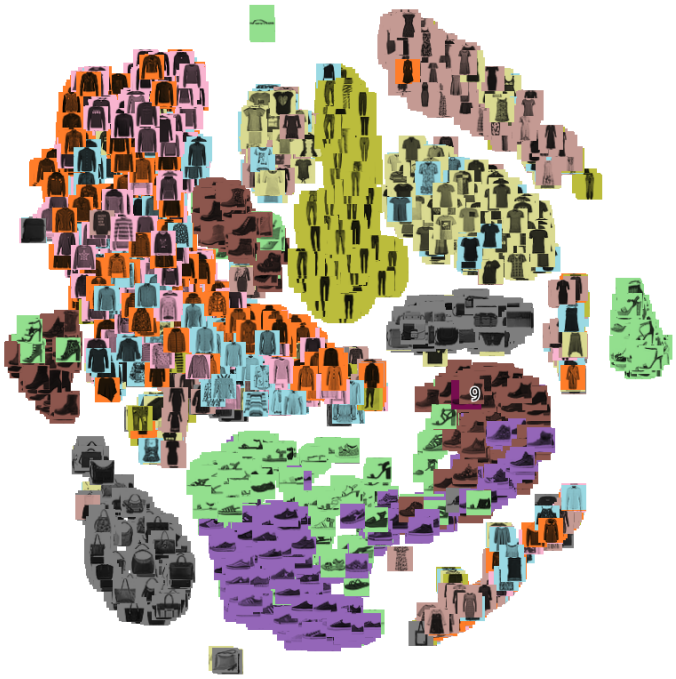

Figure 4: T-SNE visualization of the best performing NATAC model for Fashion-MNIST $d = 9$

B EXAMPLES FROM THE FASHION-MNIST DATASET.

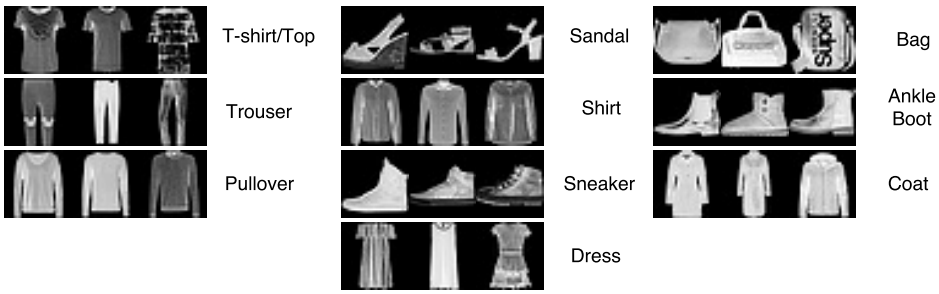

Figure 5:

C ADDITIONAL DISCUSSION ON TRAINING NATAC MODELS

C.1 GEOMETRY OF LATENT SPACE

We experimented with using polar coordinates early on in experimentation. Rather than using euclidean coordinates as the latent representation, z is considered a list of angles $\theta_1, \theta_2 \cdots \theta_n$ where $\theta_1 \cdots \theta_{n-1} \in [0, \pi]$ and $\theta_n \in [0, 2\pi]$. However, we found that the models using polar geometry performed significantly worse than those with euclidean geometry.

Additionally, we also experimented with not l2 normalizing the output of the encoder network. We hypothesized that the model would learn a better representation of the latent space by also 'learning' the geometry of the noise targets. Unfortunately, the unnormalized representation caused the noise targets to quickly collapse to a single point.

D HYPERPARAMETERS FOR EXPERIMENTS

Although each different modality (monochrome images, bag-of-words, sequence of characters) uses a different set of hyperparameters, we followed a similar recipe for determining the values for each one:

- We used a large batch size of 100. This is so each batch has a representative sample of the targets to reassign in each training step.
- The warm-up period was calculated by observing when the auxiliary objective starts to converge.
- The final value for λ in training was set so that the NAT loss was approximately 1% of the total loss.
- The initial value for λ is set as approximately 1% of the final value of λ.
- The transition phase typically lasts 100 epochs of training.
- During the transition phase, the value for α is incrementally increased from 0 to 1.

We now explicitly list the hyperparameters used for each experiment:

D.1 MNIST AND FASHION MNIST

- A batch size of 100.
- A warm-up period of $10 \times d$ epochs, during which $\lambda = 0.001$.
- A transition period lasts for 250 epochs, where λ is incrementally increased to 0.25, and α is incremented from 0 to 1.
- The ADAM optimizer (Kingma & Ba, 2014) with a learning rate ($\alpha = 10^{-5}$)

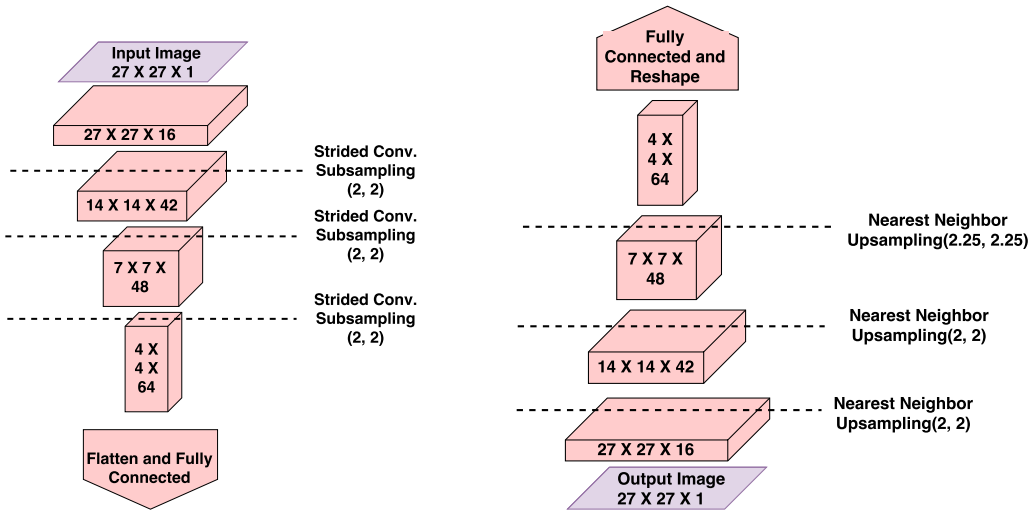

Figure 6: Architecture of the encoder (left) and decoder (right) used for the MNIST experiments. Between each subsampling layer in the encoder, a single convolution layer is applied with a filter shape of 3×3 with border padding to keep the same shape before and after the convolution. Similarly, in the decoder one transpose convolutional layer is applied between each upsampling layer, 3×3 filter shape and shape-preserving padding.

D.2 20 NEWSGROUPS

- A batch size of 100.
- A warm-up period of 200 epochs, during which $\lambda = 10$.
- A transition period lasts for 100 epochs, where λ is incrementally increased to 100, and α is incremented from 0 to 1.

- The ADAM optimizer (Kingma & Ba, 2014) with a learning rate ($\alpha = 10^{-5}$).
- Dropout with a keep probability of 0.99 in the hidden layers of the encoder and decoder.

D.3 TWITTER DATASET

- A batch size of 100.
- A warm-up period of 100 epochs, during which $\lambda = 0.001$.
- A transition period lasts for 100 epochs, where λ is incrementally increased to 0.1, and α is incremented from 0 to 1.
- The ADAM optimizer (Kingma & Ba, 2014) with a learning rate ($\alpha = 10^{-5}$).