adversarial-autoencoder-classif Documentation

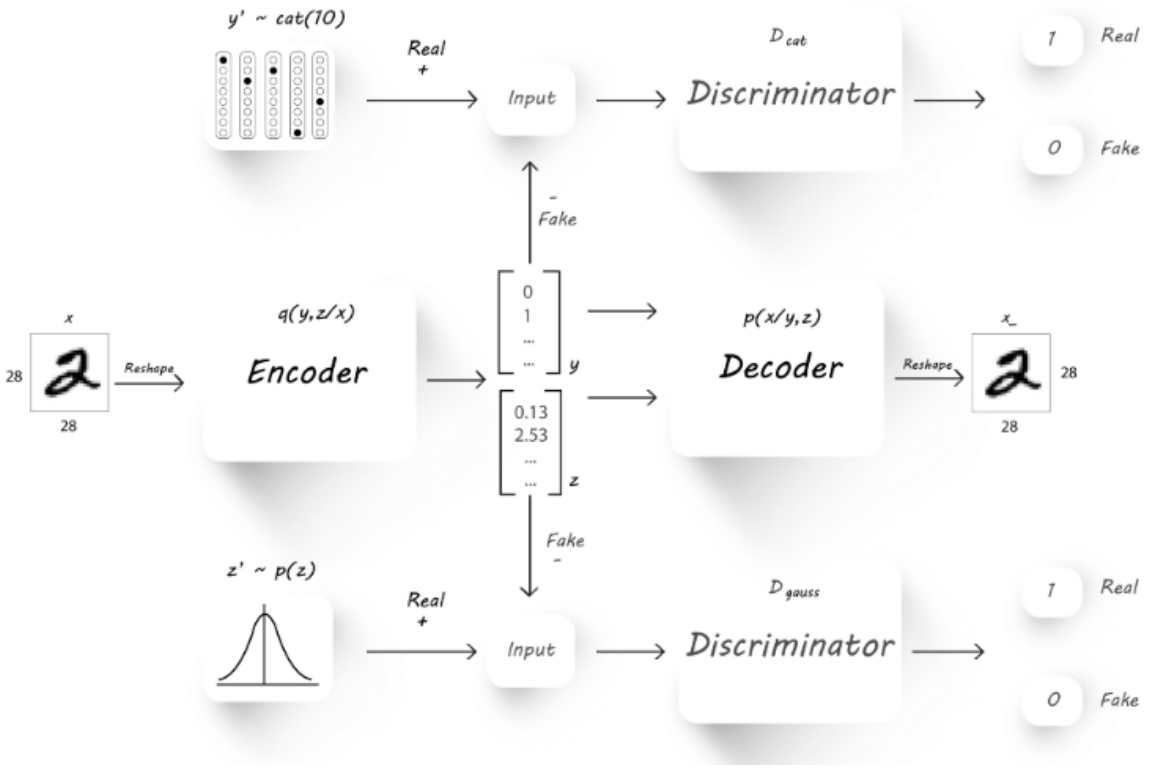
Release latest

Apr 26, 2019

Advesarial AutoEncoders Overview

PyTorch implementation of Non-parametric Unsupervised Classification with Adversarial Autoencoders.

Shahar Azulay



[1] A.Makhzani, J.Shlens, N.Jaitly, I.Goodfellow, B.Frey: Adversarial Autoencoders, 2016, arXiv:1511.05644v2

Usage Examples:

Install the module

```
>>> python setup.py install --user
```

Initialize the Datasets

```
>>> init_datasets --dir-path <path-to-data-dir>
```

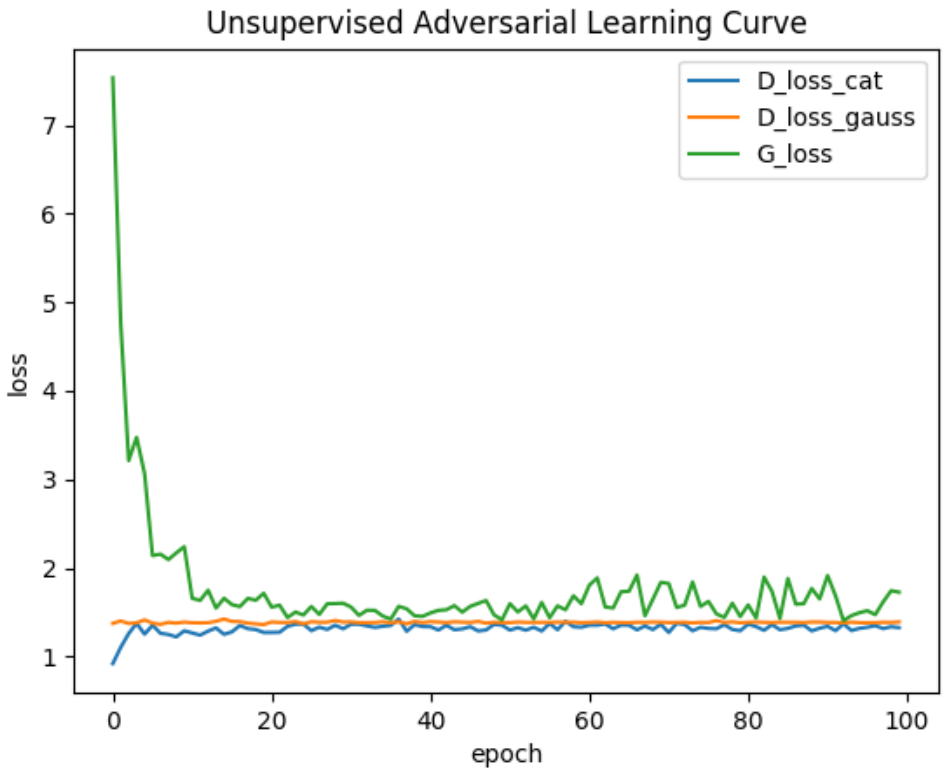
Train a new AAE in an Semi-Supervised setting

```
>>> train_semi_supervised --dir-path <path-to-data-dir> --n-epochs 35 --z-
↪size 2 --n-classes 10 --batch-size 100
loading data started...
dataset size in use: 3000 [labeled trainset]  47000 [un-labeled trainset] ␣
↪10000 [validation]
using configuration:
 {'learning_rates': {'auto_encoder_lr': 0.0008, 'generator_lr': 0.002,
↪'discriminator_lr': 0.0002, 'info_lr': 1e-05,                'mode_lr': 0.
↪0008, 'disentanglement_lr': 0}, 'model': {'hidden_size': 3000, 'encoder_
↪dropout': 0.2}, 'training':                {'use_mutual_info': False, 'use_
↪gumbel_softmax': False, 'use_disentanglement': True, 'use_adam_optimization
↪': True,            'use_adversarial_categorial_weights': True, 'lambda_z_
↪l2_regularization': 0.15}}
```

(continues on next page)

(continued from previous page)

```
current epoch:: [ ==================  ] 99.79%
...
```



Train a new AAE in a Fully Unsupervised setting

```
>>> train_unsupervised --dir-path <path-to-data-dir> --n-epochs 35 --z-size␣
↪2 --n-classes 10 --batch-size 100
loading data started...
dataset size in use: 3000 [labeled trainset]  47000 [un-labeled trainset] ␣
↪10000 [validation]
...
```

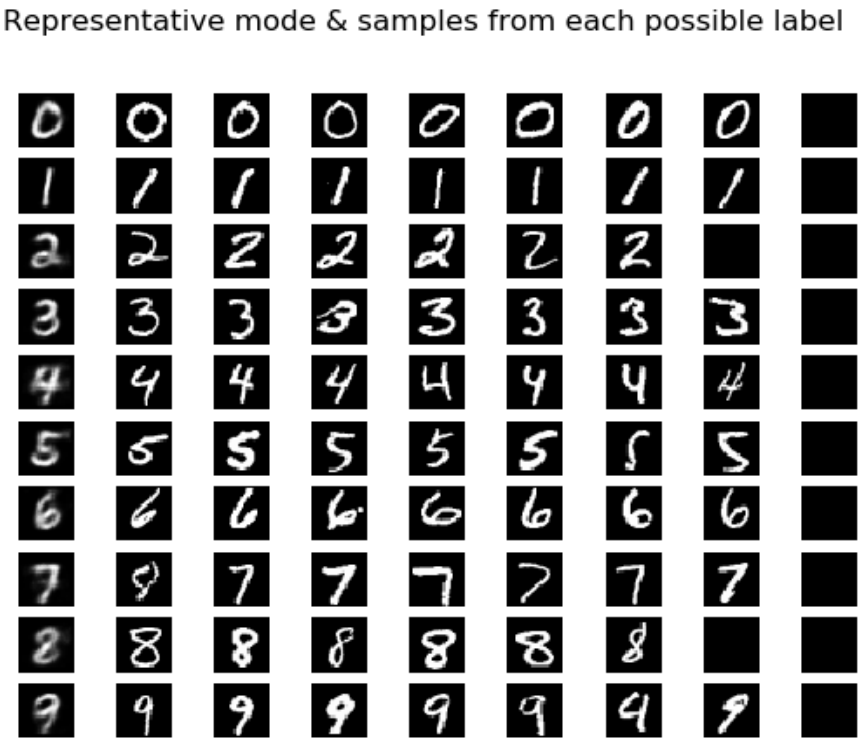
Visualize a trained model using pre-defined visualizations

```
>>> generate_model_visualization --dir-path <path-to-data-dir> --model-dir-
↪path {<path-to-model-dir> --mode unsupervised --n-classes 10 --z-size 5
loading data started...
dataset size in use: 3000 [labeled trainset]  47000 [un-labeled trainset] ␣
↪10000 [validation]
Label 1: 40.2%, Best matching label; 20
Label 2: 41.9%, Best matching label; 14
Label 3: 33.0%, Best matching label; 4
Label 4: 41.1%, Best matching label; 2
Label 5: 53.8%, Best matching label; 11
Label 6: 44.3%, Best matching label; 26
Label 7: 48.6%, Best matching label; 6
Label 8: 47.6%, Best matching label; 0
```

(continues on next page)

(continued from previous page)

```
Label 9: 40.1%, Best matching label; 22
ACCURACY: 0.85%
...
```



Control the model and training hyper-parameters using a YAML configuration file

```
>>> train_unsupervised --dir-path <path-to-data-dir> --config-path <path-
↪to-configuration-file> --n-epochs 35 --z-size 2 --n-classes 10 --batch-
↪size 100
```

CHAPTER 1

The Model

Adversarial AutoEncoders were first described by [1].

The proposes adversarial autoencoder (AAE) uses the concept of generative adversarial networks (GAN) to perform variational inference by matching the posterior of the latent encoding vector of the autoencoder with an pre-selected prior distribution.

Matching the distribution of the latent vector into a known prior ensures that meaningful samples can be generated from any part of prior space. As a result, the decoder of the AAE learns a deep generative model that maps the imposed prior to the input data distribution. In addition to that, the encoder of the AAE learns to encode any given input image into the given prior.

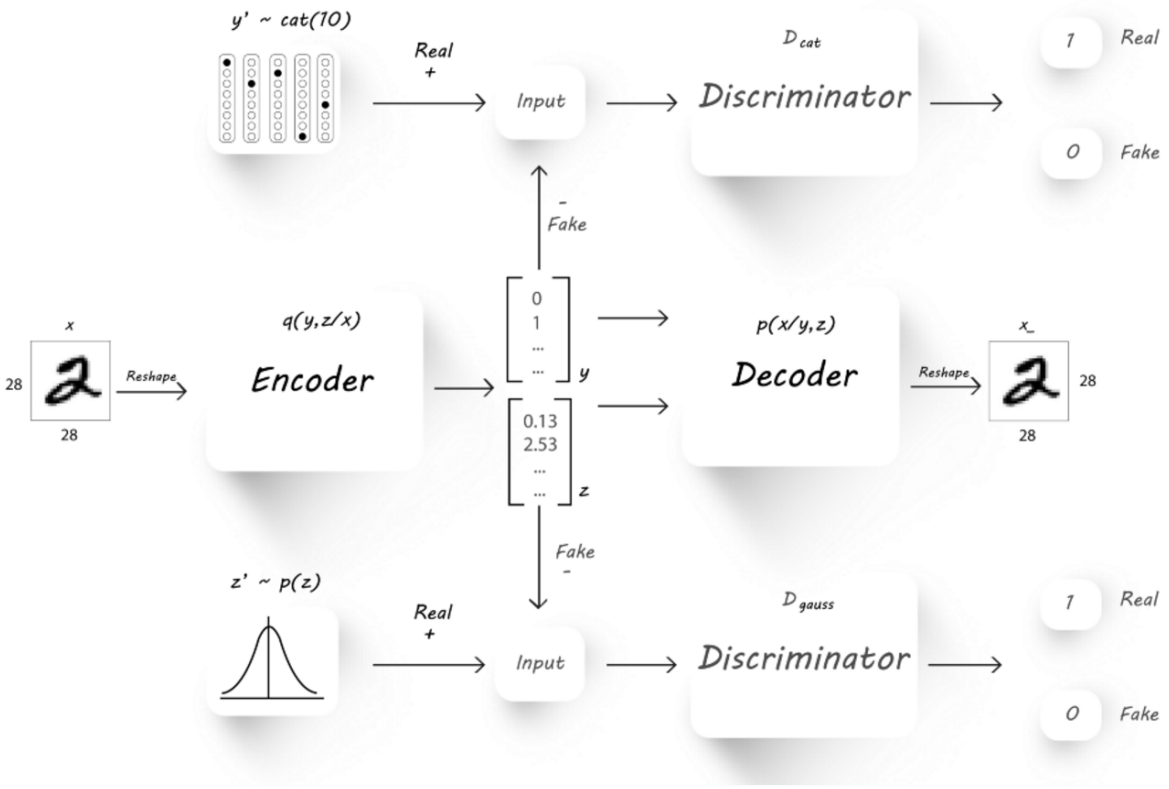
The design of the AAE makes it usable for applications such as semi-supervised classification, disentangling style and content of the input image. By selecting the prior distribution of the latent vector to describe a categorial disrtribution (in part), one can use the latent features to classify the input into a set of pre-known labels.

This can be highly effective for more than just dimensionality reduction (which is the common application when it comes to standard AE), and into supervised classification, semi-supervised classification and unsupervised clustering.

As a result in the rest of this manual the latent vector will be divied into two parts: the **latent y** part - aimed for categorical distribution, and the **latent z** part - aimed for normal zero-centered distribution.

The AAE is built from the following parts:

1. **Encoder Network** usually marked with the latter *Q*, encoding the input into the latent space.
2. **Decoder Network** usually marked with the latter *P*, decoding the latent space into an output of dimensions identical to those of the input.
3. **Categorical Descriminator** usally makred as *D_cat*, used to decide if the latent y is categorically distributed.
4. **Gauss Descriminator** usally makred as *D_gauss*, used to decide if the latent z is normally distributed.
5. **Provide a meaningful id** to the revisions - such as a timestamp, or verison number.


y' ~ cat(10)
Real
+
Input
D cat
Discriminator
1
Real
0
Fake
-
Fake
x
28
28
Reshape
q(y,z/x)
Encoder
0
1
...
...
y
0.13
2.53
...
...
z
p(x/y,z)
Decoder
Reshape
x_
28
28
Fake
-
z' ~ p(z)
Real
+
Input
D gauss
Discriminator
1
Real
0
Fake

CHAPTER 2

The Training Process

The training process combines two major parts:

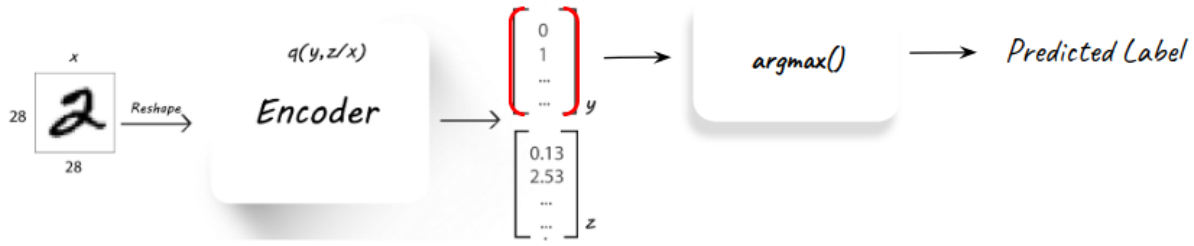
1. **Reconstruction Loss** which measures the loss between the input and image and the reconstructed output image. The reconstruction loss is part the vanilla AE training, forcing the AE to learn meaningful latent encoding as a result of forcing it to create a good reconstruction (past decoding) of the input image.
2. **Advesarial Loss** which is part of the classic advesarial generative training (GAN), and is build from a generator and a descriminator loss.

   In this case, there descriminator loss is the sum of two losses from the categorical and gaussian descriminators, each measuring a different part of the latent vector. The generaor here is also the sum of two losses, measuring the ability of the generator (the encoder Q network) to create quailty "fake" latent features which match the expected posterior the descriminators learn to identify.

CHAPTER 3

Inference

Followed the training process, the AAE is in fact a classifier as any other. The inference is performed using the decoder alone (Q) and observing the latent y part of the latent features, which can provide the predicted label for a new unseen input image.



This is how the model can tested and validated, using the inference process over a pre-known validation set.

[1] A.Makhzani, J.Shlens, N.Jaitly, I.Goodfellow, B.Frey: Adversarial Autoencoders, 2016, arXiv:1511.05644v2

CHAPTER 4

Overview

The dataset in use is the MNIST dataset, which is divided into three parts to be used accross the different training options:

1. **Labeled trainset** a smaller set of train images, with known target labels. used for semi-supervised training.
2. **Unlabeled trainset** a large set of train images, assumed to be stored without known target labels. used for un-supervised training, or as the un-supervised segment in semi-supervised training.
3. **Validation set** used to validate the model's classification accuracy in the case of semi-supervised training.

The sizes chosen for the datasets are as follows: 3000 [labeled trainset] 47000 [un-labeled trainset] 10000 [validation]

CHAPTER 5

Initialize the Datasets

Initializing the datasets requires downloading the MNIST dataset, and its segmentation into the parts described above. Therefore the dataset initialization can be done only once, using a separate entry point.

```
>>> python setup.py install --user
>>> init_datasets --dir-path <path-to-data-dir>
```

CHAPTER 6

Semi Supervised Classification with Advesarial Auto Encoders

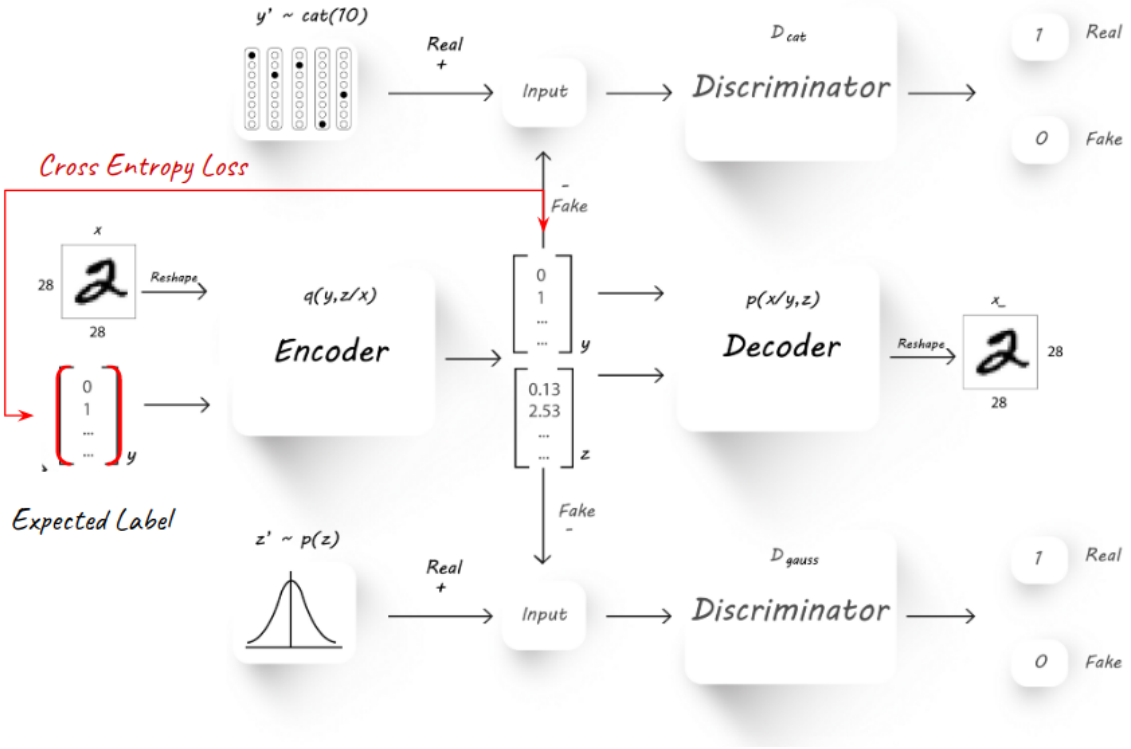
6.1 General Concept

In the concept described in [1], AAE can be submitted to semi-supervised learning, training them to predict the correct label using their latent feature representation, and based on a semi-supervised training set.

As described in the overview of this project, the adversarial autoencoder contains a simple AE at its center. The training method for semi-supervised learning exploits the generative description of the unlabeled data to improve the classification performance that would be obtained by using only the labeled data.

As in many cases, unlabeled data is more abundant and accessible. Using it as part of the adversarial AE learning, will help the encoding improve, alongside it producing better labeling.

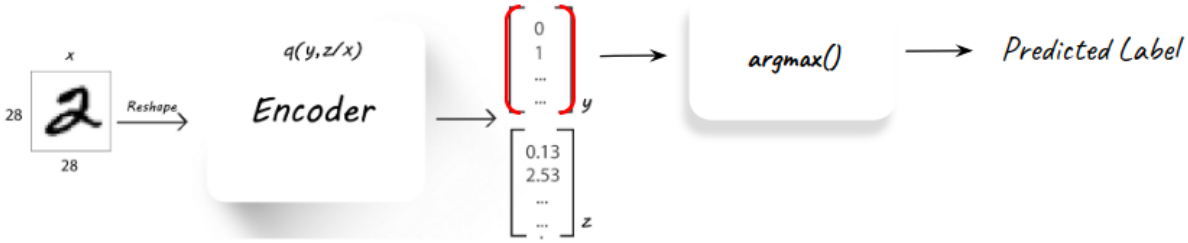
The general schema for semi-supervised learning can be seen here:



6.2 Inference and Perfomance Estimation

The basic schema follows the exact implementation of the AAE, with the only difference of introducing a labeled image from time to time into the training process. The labeled image is treated differently and is measured using a new Cross Entropy loss again the known target label. This loss only effect the Encoder (Q) - causing it to learn how to predict the labled images currectly over the latent y categorical part.

Followed the training process, the semi-supervised AAE is in fact a classifier as any other. The inference is performed using the decoder alone (Q) and observing the latent y part of the latent features, which can provide the predicted label for a new unseen input image.



This is how the model is tested and validated, using the inference process over a pre-known validation set.

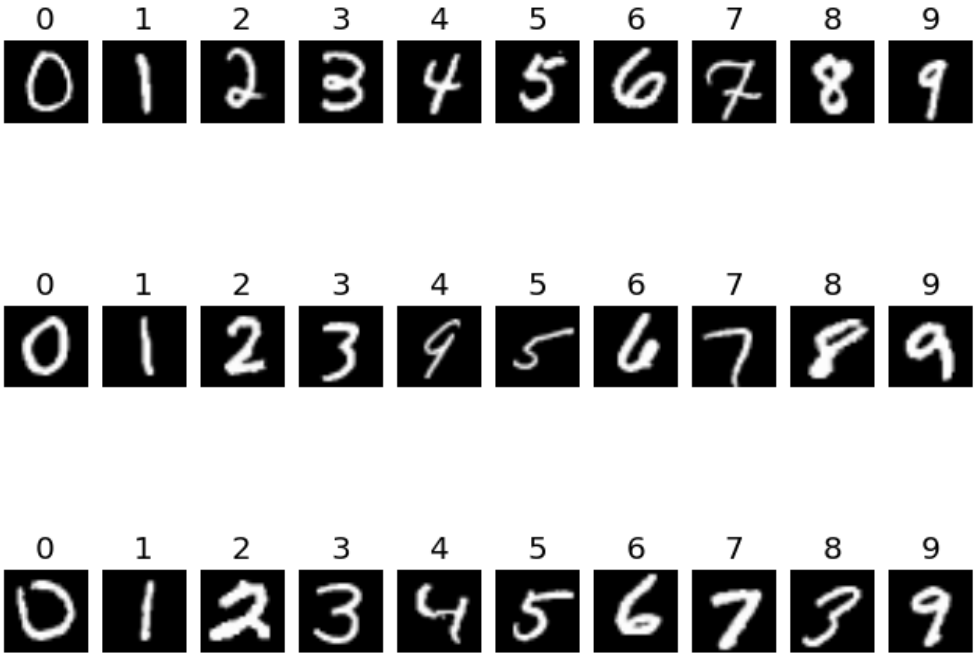
6.3 The Training Process

The training process is divided into three major parts:

1. **Auto Encoding (reconstruction)** where the Encoder-Decoder networks (Q, P) learns to encode and reconstruct the image.
2. **Adversarial** where the encoder learns how to produce latent features y which are categorical and z which are normally distirbuted.
3. **Semi-supervised** where the encoder learns to predict the right label for a pre-known labeled image.

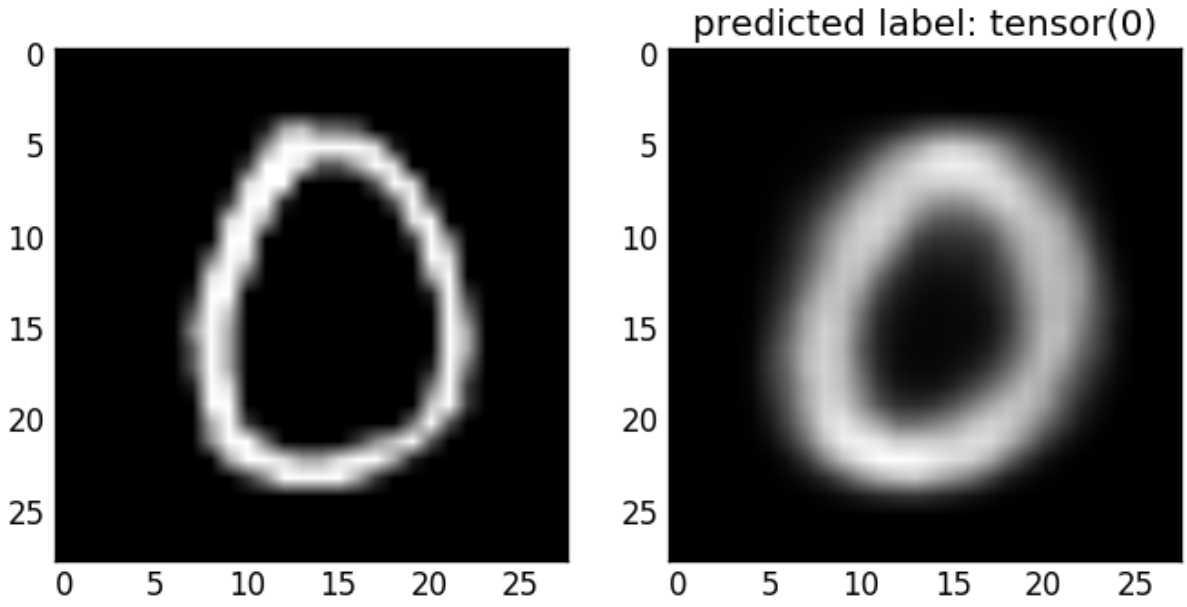
The success of the training process can be measured based on two grounds:

Validation accuracy on a held out labeled validation set. The results of the semi-supervised model reached **97% accuracy**, which shows good performance and that the model learns the labeled part properly.



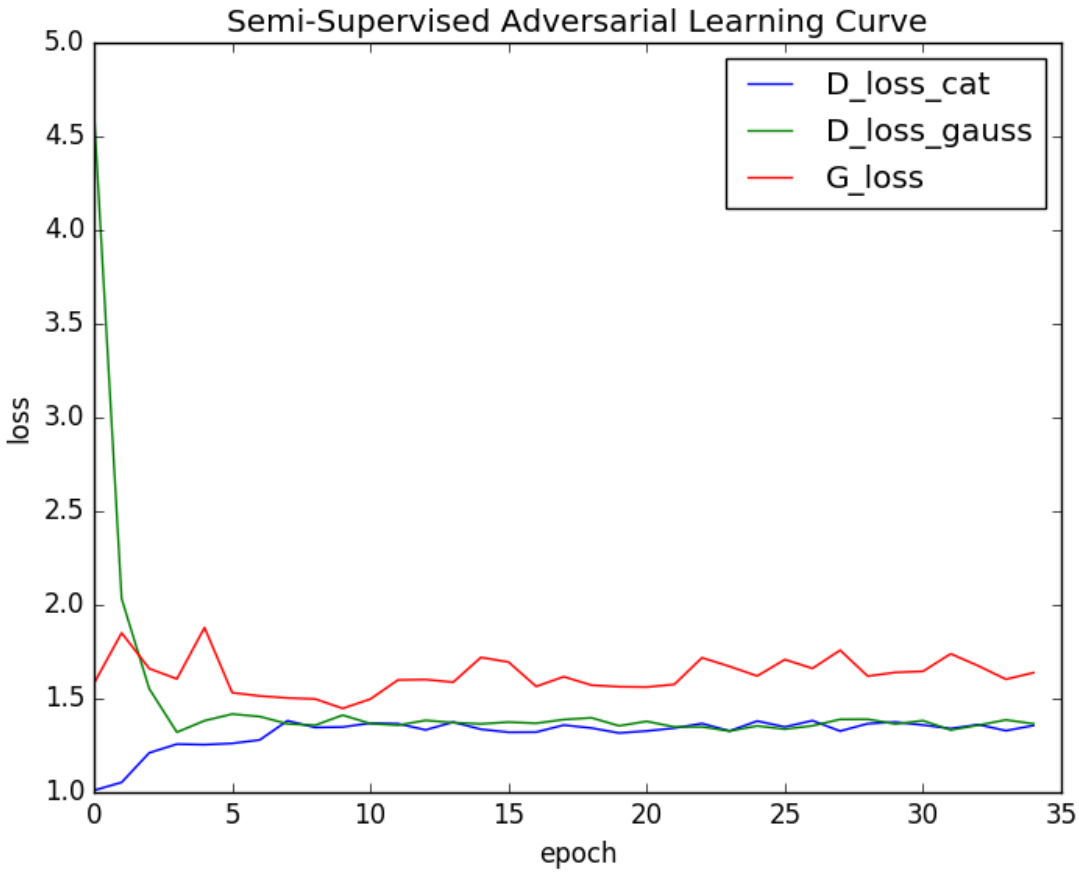
each column representing a predicted label for the original displayed images, showing the high accuracy of the model

Visual reconstruction Here we can see from visual examples that the reconstruction of an image (using the encoding-decoding pipeline) works pretty well. The reconstructed image is slightly blurry, which might be corrected with a slightly different loss function.



an example reconstruction of an original "0" digit image

In order to analyise the success of the adversarial part (which is focused on the latent features) we can examine the learning curve, showing the loss of the generator, and descriminator networks:

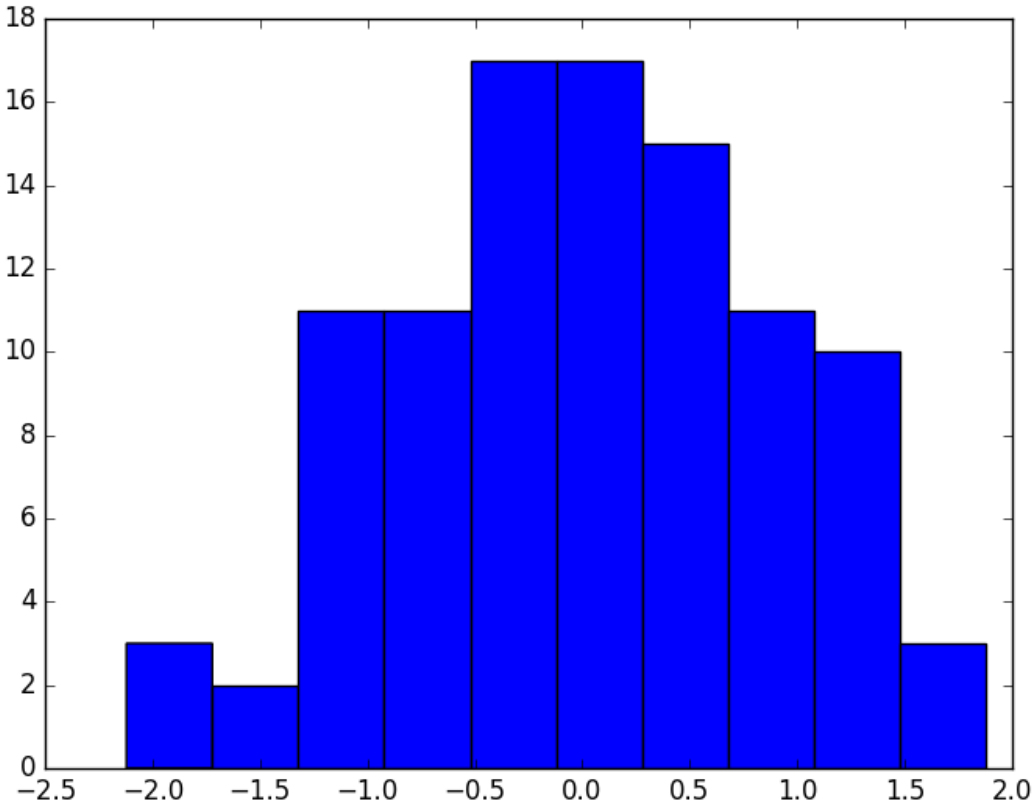


the adversarial learning curve, showing the balance which is created between generator and discriminators

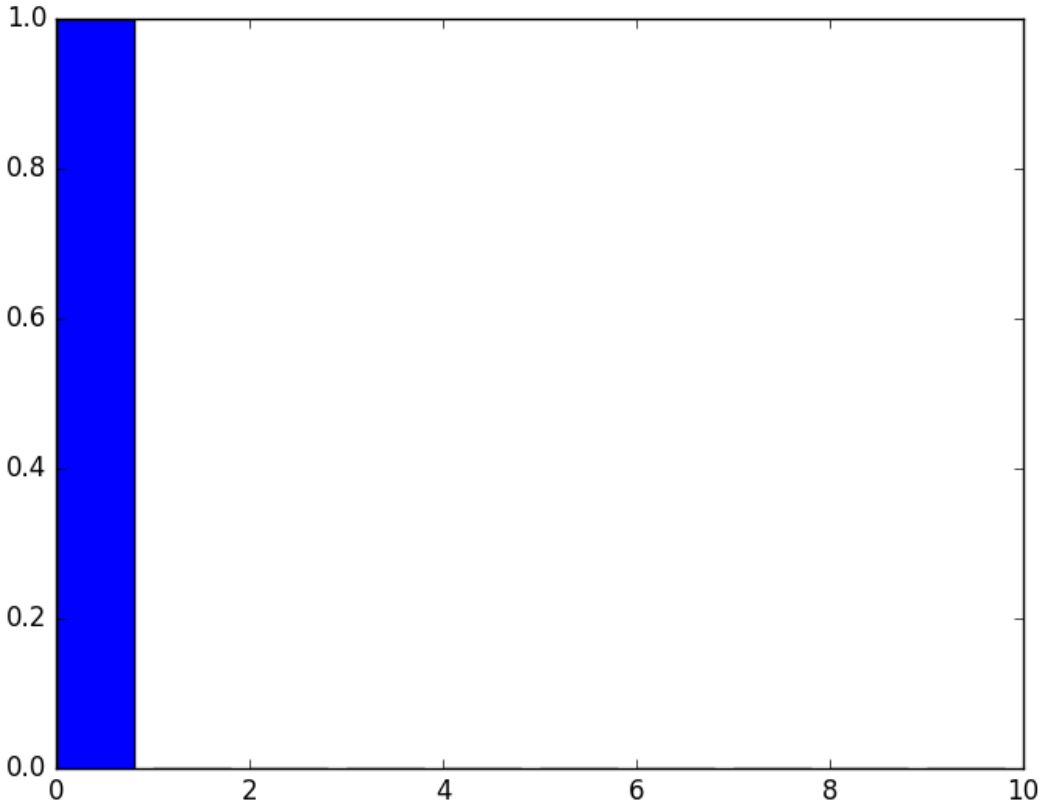
6.4 The Latent Features

The adverserial training pushes the latent features to the desired distribution. The latent y part learns to behave similarly to a categorial distribution, whlie the latent z part learns to distribute as a zero-centered normal.

First, we can see that the latent features were trained properly, using the adversarial balance.



the empricial distribution of the first dimension in the latent z vector, showing that the learned feature is indeed normally distributed around zero.

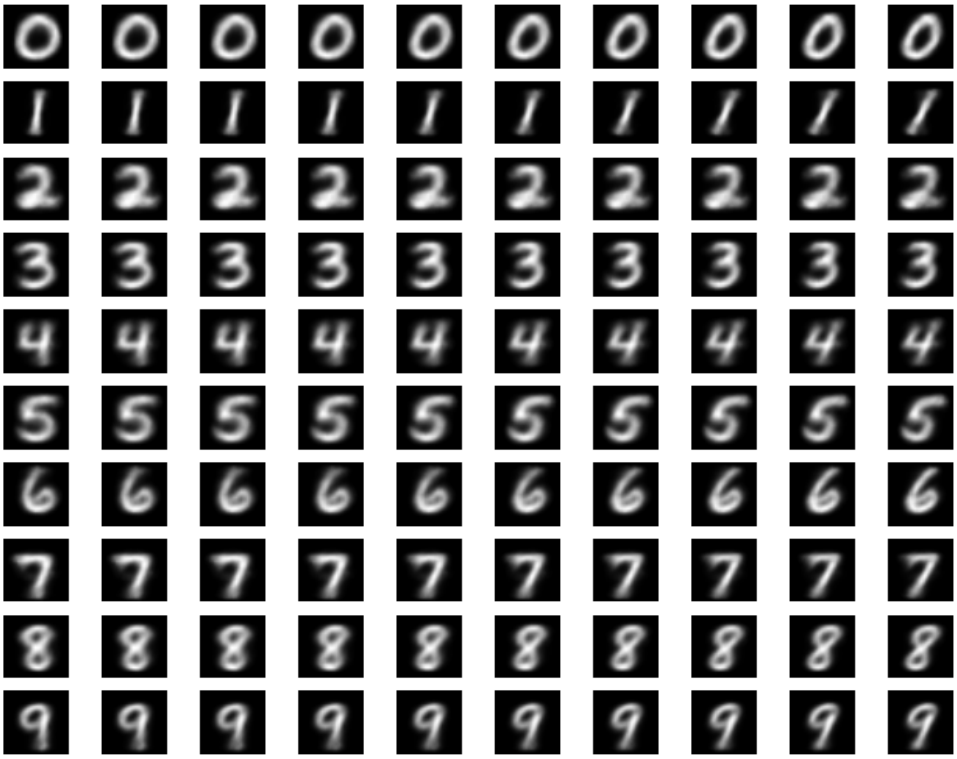


an empricial example of the value of the latent y vector, showing that the learned feature is indeed categorial, showing close to "1" only near the predicted label.

Next we would like to find out if the latent features really perform as expected. The latent y vector is trained to learn the label, or "mode" of the input. We want it to describe the actual digit inside the input, and the semi-supervised procedure helps us reach that target.

The latent z vector is expected to represent "style", and capture the deeper style of writing of a specific input digit. Again, this happens only thanks to the semi-supervision of known labels, pushing the latent y to capture what is neccesary to describe the type of digit.

Here's a simple visualization of the meaning of the latent features:



each row represents a specific latent y value (out of the categorial distribution), and along that row the first dimension of the latent z vector is sampled uniformly from the normal distribution. One can see that indeed, the latent y completely catches the label, while the latent z controls the style and shape of the digit.

[1] A.Makhzani, J.Shlens, N.Jaitly, I.Goodfellow, B.Frey: Adversarial Autoencoders, 2016, arXiv:1511.05644v2

CHAPTER 7

Fully Unsupervised Classification with Advesarial Auto Encoders

7.1 General Concept

In the concept described in [1], AAE can be submitted to semi-supervised learning, training them to predict the correct label using their latent feature representation, and based on a semi-supervised training set.

Training the the network in a fully unsupervised manner requires deeper analysis and handling.

In order to create a better disentanglement in the AAE's latent space the following methods were tested:

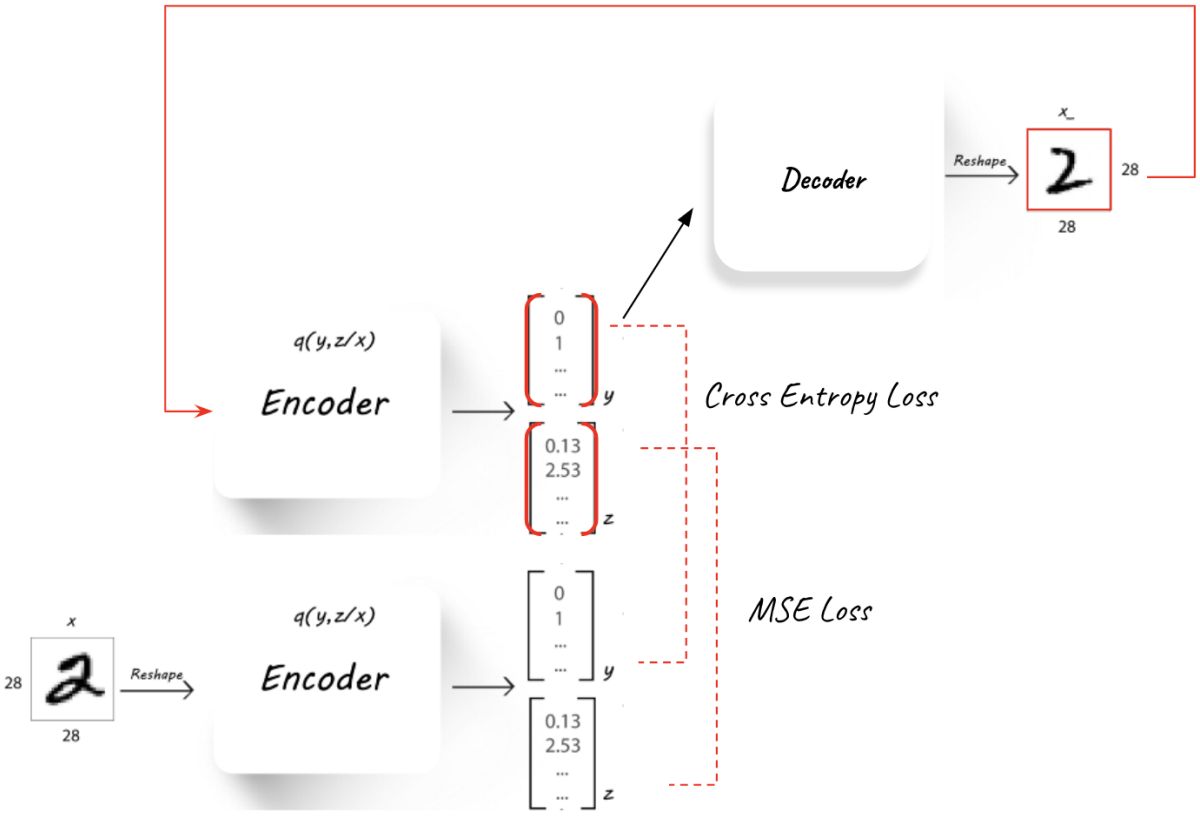
1. **Cyclic mode loss** The use of a cyclic mode loss (Figure 1) - that will measure the mutual information between the latent space after the Encoder and the one after another cycle of Decode-Encoder. Minimizing this loss should push the Encoder and Decoder to use the latent space in a consistent fashion. The cyclic loss is implemented in a similar idea to the one suggested in InfoGAN[2].

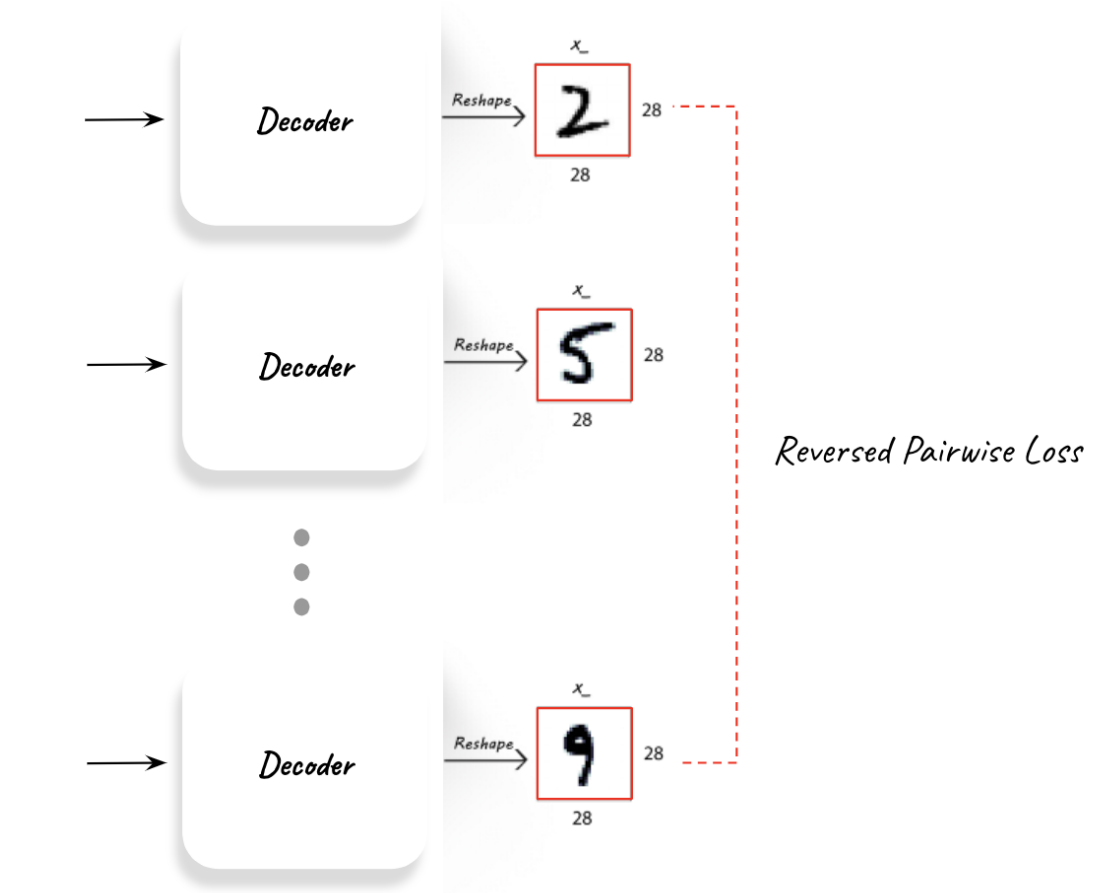
2. **L2 regularization on latent z** Limiting the use of the z part of the latent space using L2 loss regularization over z. Minimizing this loss can assist in pushing more information into the y part when possible, and possibly allow better disentanglement in the latent space.

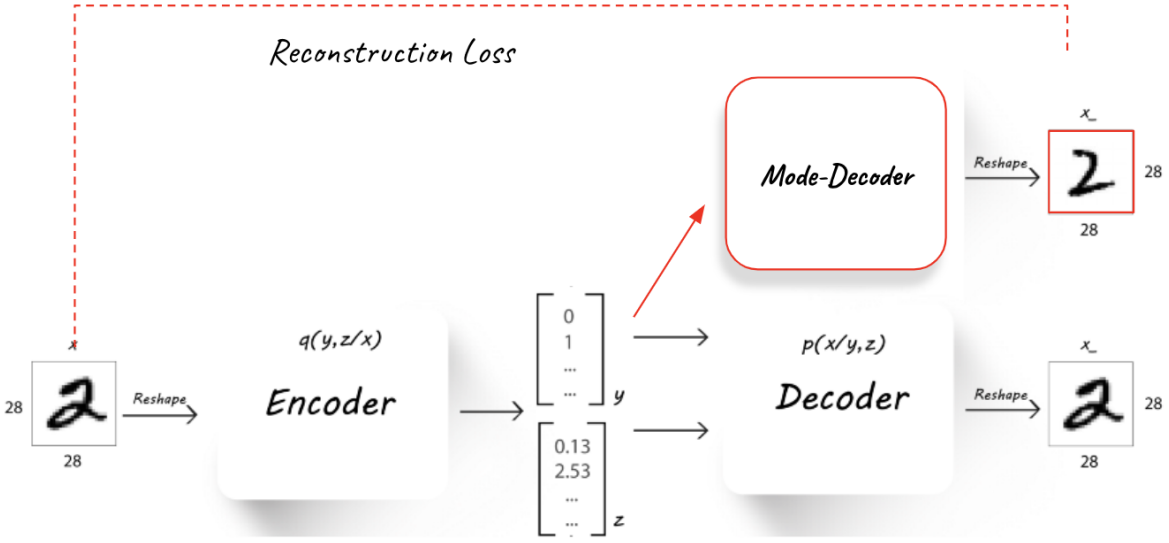
3. **Mode reconstruction loss** Integrating another Decoder (Figure 2) called the Mode-Decoder into the training process, which is trained to learn the "mode" of the image, therefore forcing the Encoder to use the y latent space in a meaningful way.

4. **reversed pairwise mode loss** To try and improve the style-label mixture inside the generated clusters another method was attempted and it was to integrate reversed pairwise mode loss (Figure 15) - that will push the Decoder to create "modes" which are as far apart from one another as possible.

The added componenets to the unsupervised training can be seen here:


Decoder
Reshape
x_
28
28
q(y,z/x)
Encoder
0
1
...
...
y
0.13
2.53
...
...
z
Cross Entropy Loss
MSE Loss
x
28
Reshape
28
q(y,z/x)
Encoder
0
1
...
...
y
0.13
2.53
...
...
z


Decoder
Reshape
28
28
Decoder
Reshape
28
28
Decoder
Reshape
28
28
Reversed Pairwise Loss


Reconstruction Loss
Mode-Decoder
Reshape
28
28
28
Reshape
q(y,z/x)
Encoder
0
1
y
0.13
2.53
z
p(x/y,z)
Decoder
Reshape
28
28

7.2 Unsupervised classification accuracy metric

Before diving into the unsupervised setting, a metric of accuracy performance needs to be agreed upon. In the fully unsupervised scenario the model has no input on the real "cultural" labels stored in the validation set. Meaning it can be a perfect classifier, and label all "0" digits together, all "1" digits together and so on, but label each group under the "wrong" cultural label (for example labeling all "0" digits under the label "8").

Inspired by a metric commonly used for clustering accuracy, the chosen metric used in the following parts of this paper will be referred to as unsupervised classification accuracy.

The metric was measured after the training of an AAE model (and during for debugging purposes only) and it follows the following logic - The trained model was used to predict the labels of the entire validation set. Each possible output label of the AAE was assigned a true MNIST label using the highest appearing MNIST label classified under this output label. Accuracy is determined by counting the percentage [%] of validation samples that are classified in an output label that was assigned to their true MNIST label.

For example, let's say the AAE is built using 10 possible output labels (the latent y is of size 10), and under label "3" (post-training) a 1000 samples (out of the 10K validation set samples) were classified, and 75% of those samples where the MNIST digit "4" and 25% the MNIST digit "6". Then the output label "3" will be assigned to the best matching MNIST label - "4", and all the "6" digits classified under it will be considered a misclassification.

7.3 Latent space disentangelment

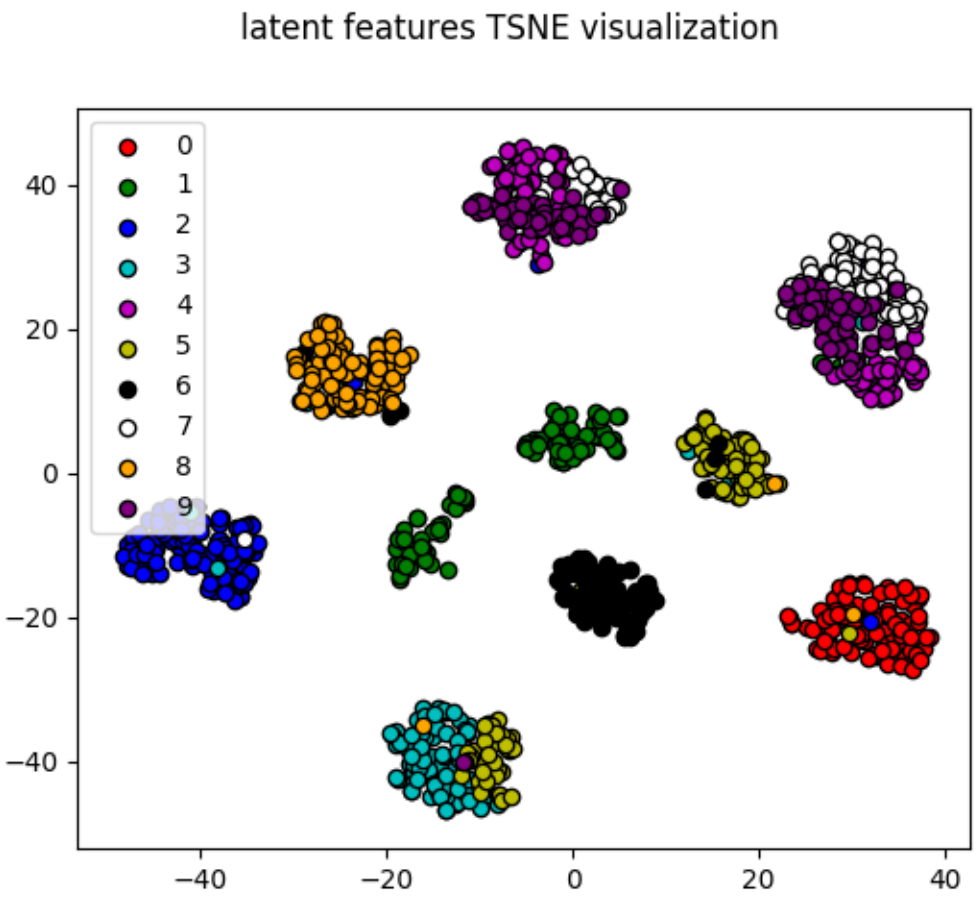
The end goal of the unsupervised training is the hard task of separating style and label. The latent z part create by the Encoder is supposed to take on pure style, allowing the y latent part to represent the pure label.

This task is clearly hard since labels are in many cases more "cultural" than actually represented in the data itself.

To analyze the ability of the AAE to cluster the data into pure separate labels a latent space visualization is a good place to start. It allows for two things:

1. **Clusteting potential assessment** We can assess if the AAE can even cluster efficiently into separate classes using the latent space.
2. **Style-label mixture assessment** Allowing us to see which labels mix together and which don't.

The latent space visualization is an important tool in this unsupervised learning.



[1] A.Makhzani, J.Shlens, N.Jaitly, I.Goodfellow, B.Frey: Adversarial Autoencoders, 2016, arXiv:1511.05644v2

[2] X.Chen, Y.Duan, R.Houthooft, J.Schulman, I.Sutskever, P.Abbeel: InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets, 2016, arXiv:1606.03657v1

CHAPTER 8

Initializing the Datasets

```
>>> python setup.py install --user
>>> init_datasets --dir-path <path-to-data-dir>
```

As a result the three datasets will be created in three different files inside the data directory: *train_labeled.p, train_unlabeled.p, validation.p*

CHAPTER 9

Semi-supervised Training

```
>>> python setup.py install --user
>>> train_semi_supervised --dir-path <path-to-data-dir> --n-epochs 35 --z-size 2 --n-
↪classes 10 --batch-size 100
```

As a result the following files will be created inside the data directory:

1. **Decoder / Encoder networks** the Encoder-Decoder networks (Q, P) will be stored for future use and analysis under *decoder_semi_supervised* and *encoder_semi_supervised*
2. **Learning curves** as *png* images, describing the adversarial, reconstruction and semi-supervised classification learning curves.

CHAPTER 10

Un-supervised Training

```
>>> python setup.py install --user
>>> train_unsupervised --dir-path <path-to-data-dir> --n-epochs 35 --z-size 2 --n-
↪classes 10 --batch-size 100
```

As a result the following files will be created inside the data directory:

1. **Decoder / Encoder networks** the Encoder-Decoder networks (Q, P) will be stored for future use and analysis under *decoder_unsupervised* and *encoder_unsupervised*
2. **Mode Decoder network** (if chosen in the configuration) responsible for the reconstruction of the image based on the categorial latent y, will be stored under *mode_decoder_unsupervised*
3. **Learning curves** as *png* images, describing the adversarial, reconstruction and mutual information learning curves.

CHAPTER 11

Visualization of the Learned Model

```
>>> python setup.py install --user
>>> generate_model_visualization --dir-path <path-to-data-dir> --model-dir-path {
↪<path-to-model-dir> --mode unsupervised --n-classes 10 --z-size 5
```

As a result the following files will be created inside the model directory under a sub-directory called 'visualization':

1. **latent_features_tnse.png** A 2-D visualization of the latent space using TSNE transformation over the validation set.
2. **learned_latent_features.png** The role of the latent features over the reconstructed image.
3. **modes_and_samples_from_each_label.png** Learned style-free modes of the Decoder model alongside real samples labeled by the AAE.
4. **predicted_label_distribution.png** The distribution of the predicted labels.
5. **reconstruction_example.png** An example of a real input and its reconstruction.
6. **y_distribution.png** The distribution of the y latent space, representing a categorical distribution that can be used for labeling at inference. The plot shows the histogram of the maximum probability generated by the y latent vector.
7. **z_distribution.png** The distribution of one of the nodes in the z latent vector.
8. **learned_modes.png** In case of the Mode-Decoder being used as part of the model, this plot shows the modes learned by this decoder.

CHAPTER 12

Hyperparmeter configuration

All hyper-paremeters are configurable via a YAML file. The only exclusions are the number of epochs, and the dimention of the latent z and y - which are all configurable via command line arguments when training the model.

The YAML configuration file contains many training and model configurations (as can be seen below) and in any training command the user can provide a path to the wanted config file. An example config file is found in the source code, and is used by default if another path is not provided.

```yaml
# unsupervised learning configurations
unsupervised:
  learning_rates:
    auto_encoder_lr: 0.0008
    generator_lr: 0.002
    discriminator_lr: 0.0002
    info_lr: 0.00001
    mode_lr: 0.0008
    disentanglement_lr: 0.00005
  model:
    hidden_size: 3000
    encoder_dropout: 0.2
  training:
    use_mutual_info: False
    use_mode_decoder: False
    use_disentanglement: True
    use_adam_optimization: True
    use_adversarial_categorial_weights: True
    lambda_z_l2_regularization: 0.15

# sem-supervised learning configurations
semi_supervised:
  learning_rates:
    auto_encoder_lr: 0.0008
    generator_lr: 0.001
    discriminator_lr: 0.0002
    classifier_lr: 0.001
  model:
    hidden_size: 1000
    encoder_dropout: 0
```