Deeplearning

Chapter 8: Autoencoders

Bernd Bischl

Department of Statistics – LMU Munich

Winter term 2018

Unsupervised learning

UNSUPERVISED LEARNING

- So far, we were dealing with different types of neural networks designed for classification and regression tasks.
- In these **supervised learning** scenarios, we exploit information of class memberships (or numeric values) to train our algorithm. That means in particular, that we have access to labeled data.
- Recall from the very first lecture, that there exists another learning paradigm, **unsupervised learning**, where :
  - training data consists of unlabeled input points $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}$
  - and one aims at finding and describing intrinsic structure in the data.
- There is much more unlabeled data than labeled! But what can we learn from it?

UNSUPERVISED LEARNING - EXAMPLES

1. **Clustering.**

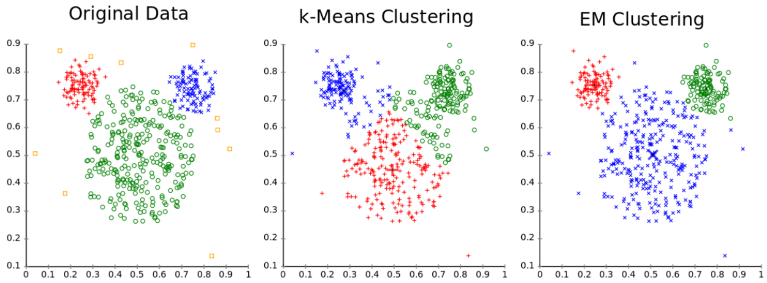


Figure: Different cluster analysis results on a dataset. True labels (the colors in the original data) are shown here but the algorithms only operate on unlabelled data. (Source : Wikipedia)

UNSUPERVISED LEARNING - EXAMPLES

2. **Dimensionality reduction/manifold learning.**
   - E.g. for visualisation in a low dimensional space.

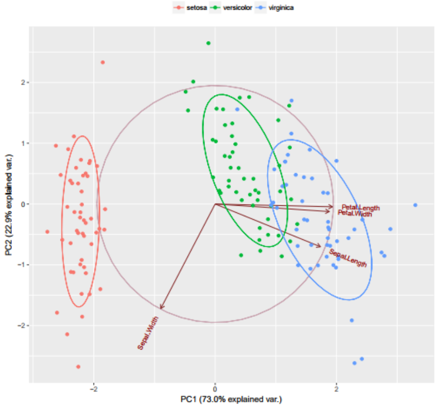


Figure: Principal Component Analysis (PCA)

UNSUPERVISED LEARNING - EXAMPLES

2. **Dimensionality reduction/manifold learning.**
    - E.g. for image compression.

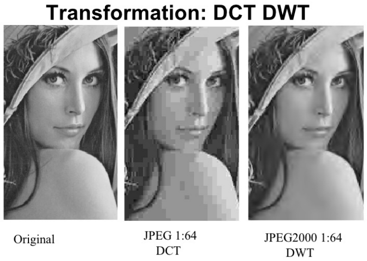


Figure: from `https://de.slideshare.net/hcycon/bildkompression`

UNSUPERVISED LEARNING - EXAMPLES

3. Feature extraction/representation learning.

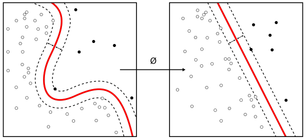


Figure: Source: Wikipedia

- E.g. for **semi-supervised learning**: features learned from an unlabeled dataset are employed to improve performance in a supervised setting.

UNSUPERVISED LEARNING - EXAMPLES

4. Density fitting/learning a generative model.

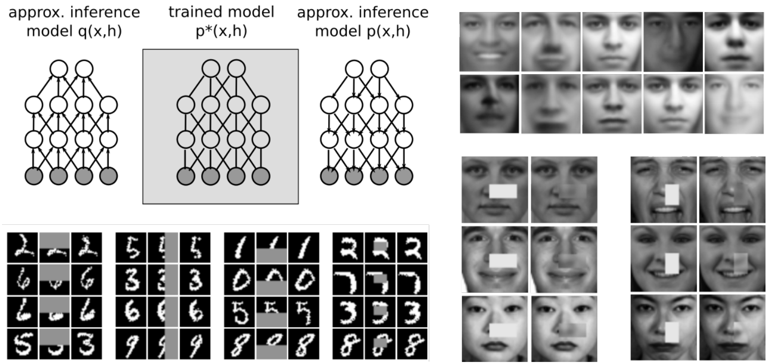

approx. inference model q(x,h)
trained model p*(x,h)
approx. inference model p(x,h)

Figure: A generative model can reconstruct the missing portions of the images. (Bornschein, Shabanian, Fischer & Bengio, ICML, 2016)

MANIFOLD LEARNING

- **Manifold hypothesis**: Data of interest lies on an embedded non-linear manifold within the higher-dimensional space.
- A **manifold**:
  - is a topological space that locally resembles the Euclidean space.
  - in ML, more loosely refers to a connected set of points that can be approximated well by considering only a small number of dimensions.

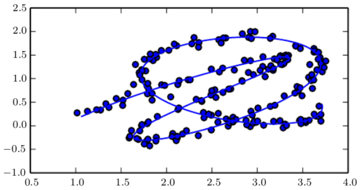


Figure: from Goodfellow et. al

MANIFOLD LEARNING

- An important characterization of a manifold is the set of its tangent planes.
- **Definition**: At a point $\boldsymbol{x}$ on a d-dimensional manifold, the **tangent plane** is given by d basis vectors that span the local directions of variation allowed on the manifold.

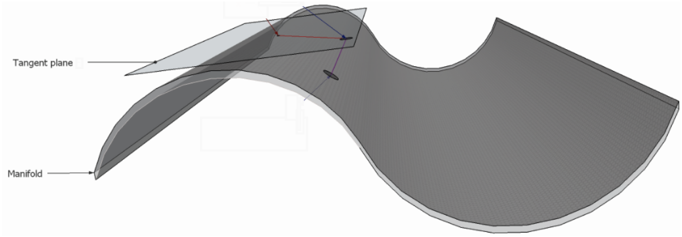


Figure: A pictorial representation of the tangent space of a single point, $\boldsymbol{x}$, on a manifold (Goodfellow et al. (2016)).

MANIFOLD LEARNING

- Manifold hypothesis does not need to hold true.
- In the context of AI tasks (e.g. processing images, sound, or text) it seems to be at least approximately correct, since :
  - probability distributions over images, text strings, and sounds that occur in real life are highly concentrated (randomly sampled pixel values do not look like images, randomly sampling letters is unlikely to result in a meaningful sentence).
  - samples are connected to each other by other samples, with each sample surrounded by other highly similar samples that can be reached by applying transformations (E.g. for images: Dim or brighten the lights, move or rotate objects, change the colors of objects, etc).

REVISION OF PCA

- The purpose of PCA is to project the data $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}$ onto a lower-dimensional subspace (e.g. to save memory).
- For each point $\mathbf{x}^{(i)} \in \mathbb{R}^p$ we need to find a corresponding code vector $\mathbf{c}^{(i)} \in \mathbb{R}^l$ with $l < p$. That step is accomplished by the encoding function which produces the code for an input:

$$f(\mathbf{x}) = \mathbf{c}$$

- Additionally, we need a decoding function to produce the reconstruction of the input given its code:

$$\mathbf{x} \approx g(f(\mathbf{x}))$$

- We can choose matrix multiplication to map the data back into $\mathbb{R}^p$: $g(\mathbf{c}) = \mathbf{Dc}$, with $\mathbf{D} \in \mathbb{R}^{p \times l}$, defining the decoding.

REVISION OF PCA

- To keep the encoding problem easy, PCA constrains the columns of **D** to be orthogonal.
- One way to obtain the optimal code $\mathbf{c}^*$ is to minimize the distance between the input $\mathbf{x}$ and its reconstruction $g(\mathbf{c})$ (that means, linear transformation with minimum reconstruction error):

$$\mathbf{c}^* = \arg\min_{\mathbf{c}} ||\mathbf{x} - g(\mathbf{c})||_2^2$$

- Solving this optimization problem leads to

$$\mathbf{c} = \mathbf{D}^T\mathbf{x}$$

- Thus, to encode a vector, we apply the encoder function

$$f(\mathbf{x}) = \mathbf{D}^T\mathbf{x}$$

REVISION OF PCA

- We can also define the PCA as the reconstruction operation:

$$r(\mathbf{x}) = g(f(\mathbf{x})) = \mathbf{D}\mathbf{D}^T\mathbf{x}$$

- To find the encoding matrix $\mathbf{D}^*$, we minimize the Frobenius norm of the matrix of errors computed over all dimensions and points:

$$\mathbf{D}^* = \arg\min_{\mathbf{D}} \sqrt{\sum_{i,j} \left(x_j^{(i)} - r(x^{(i)})_j\right)^2}, \text{ subject to } \mathbf{D^T}\mathbf{D} = \mathbf{I}_l$$

- for $l = 1$, $\mathbf{D}^*$ collapses to a single vector and we can rewrite the equation as

$$\mathbf{d}^* = \arg\min_{\mathbf{d}} ||\mathbf{X} - \mathbf{X}\mathbf{d}\mathbf{d}^T||_F^2, \text{ subjected to } \mathbf{d}^T\mathbf{d} = 1$$

- The optimal $\mathbf{d}^*$ is given by the eigenvector of $\mathbf{X}^T\mathbf{X}$ corresponding to the largest eigenvalue.

REVISION OF PCA

- In general, for $l = k$ (with $k < p$) , the optimal reconstruction $\mathbf{X}^*$, by the Eckart-Young-Mirsky Theorem, is the truncated **Singular Value Decomposition (SVD)** of $\mathbf{X}$:

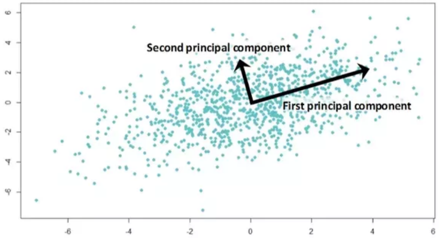
$$\mathbf{X}^* = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top$$

  where, the diagonal matrix $\mathbf{\Sigma}_k$ contains the k largest **singular values** and the columns of the matrices $\mathbf{U}_k$ and $\mathbf{V}_k$ are the corresponding **right singular vectors** and **left singular vectors**, respectively.

- Here, the optimal encoding matrix $\mathbf{D}^*$ consists of the k left singular vectors as columns.

REVISION OF PCA

- The first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible).
- Each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components.



credit:Syed Nazrul

Figure: The vectors shown are the (scaled) eigenvectors. Keeping only the first principal component results in dimensionality reduction.

UNSUPERVISED DEEP LEARNING

Given i.i.d. (unlabeled) data $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n \sim p_{\text{data}}$, in unsupervised deep learning, one usually trains :

- an autoencoder (a special kind of neural network) for **representation learning** (feature extraction, dimensionality reduction, manifold learning, ...), or,

UNSUPERVISED DEEP LEARNING

Given i.i.d. (unlabeled) data $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n \sim p_{\text{data}}$, in unsupervised deep learning, one usually trains :

- an autoencoder (a special kind of neural network) for **representation learning** (feature extraction, dimensionality reduction, manifold learning, ...), or,
- a **generative model**, i.e. a probabilistic model of the data generating distribution p_{data} (data generation, outlier detection, missing feature extraction, reconstruction, denoising or planning in reinforcement learning, ...).

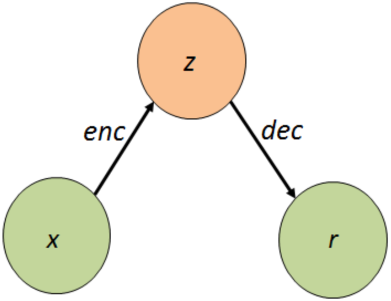
Autoencoder

AUTOENCODER (AE)-TASK AND STRUCTURE

- Autoencoders (AEs) are a special kind of feedforward neural networks.
- Task: reconstruction of the input.
- They consist of two parts:
  - **encoder** function $\mathbf{z} = f(\mathbf{x})$.
  - **decoder** that produces the reconstruction $\mathbf{r} = g(\mathbf{z})$.
- Loss function: $L(\mathbf{x}, g(f(\mathbf{x})))$.
- Goal: Learn good **internal representations z** (also called **code**).

AUTOENCODER (AE)- COMPUTATIONAL GRAPH

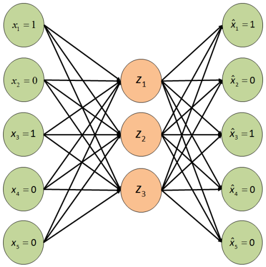
The general structure of an AE as a computational graph:



- An AE has two computational steps:
  - the encoder *enc*, mapping **x** to **z**.
  - the decoder *dec*, mapping **z** to **r**.

UNDERCOMPLETE AUTOENCODERS

- If an AE simply learns the identity $dec(enc(\mathbf{x})) = \mathbf{x}$, it would be of no use. In fact, we want the AE to learn a representation $\mathbf{z}$ that encodes "useful" or "significant" properties of the data.
- One possibility to do so is to restrict the architecture, such that **code dimension** $<$ **input dimension**.
- Such an AE is called **undercomplete**.



UNDERCOMPLETE AUTOENCODERS

- In other words: In an undercomplete AE, the hidden layer has fewer neurons than the input layer.

  $\Rightarrow$ That will force the AE to

  - capture only the most salient features of the training data!
  - learn a “compressed” representation of the input.
- Training an AE is done by minimizing a loss function which penalizes the reconstruction $dec(enc(\mathbf{x}))$ for differing from $\mathbf{x}$. The MSE, $||\mathbf{x} - g(f(\mathbf{x}))||_2^2$, is a typical choice.
- For optimization, the very same optimization techniques as for standard feed-forward nets are applied (SGD, RMSProp, ADAM,...).

UNDERCOMPLETE AUTOENCODERS

- Example: For an undercomplete AE with :
  - linear decoder $g(\boldsymbol{z}) = W\boldsymbol{z}$
  - mean squared error loss $L = ||\boldsymbol{x} - g(f(\boldsymbol{x}))||_2^2$
  - input normalized to have zero mean

  optimal encoder corresponds to Principal Component Analysis (PCA).
- An AE with a non-linear decoder/encoder can be seen as a non-linear generalization of PCA.
- Problem: If an AE is **overcomplete** (code dimension $>$ input dimension) or encoder and decoder are too powerful, the AE can learn to simply copy the input.

OVERCOMPLETE AE – PROBLEM

Example 1: Overcomplete AE (code dimension $\geq$ input dimension).
$\Rightarrow$ even a linear AE can copy the input to the output without learning anything useful.

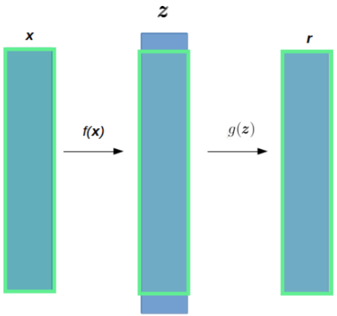


Figure: Overcomplete AE that learned to copy its inputs to the hidden layer and then to the output layer (Credits to A.-L. Popkes and P. Wenker).

VERY POWERFUL AE – PROBLEM

Example 2: Very powerful nonlinear AE that learns a 1D code:

- Encoder: learns to map each training example $\mathbf{x}^{(i)}$ to the code i.
- Decoder: learns to map these integer indices back to the values of specific training examples.

LEARNING MANIFOLDS WITH AES

- AE training procedures involve a compromise between two forces:
  1. Learning a representation $\boldsymbol{z}$ of a training example $\boldsymbol{x}$ such that $\boldsymbol{x}$ can be approximately recovered from $\boldsymbol{z}$ through a decoder.
  2. Satisfying an architectural constraint or regularization penalty.
- Together, they force the hidden units to capture information about the structure of the data generating distribution
- Important principle: AEs can afford to represent only the variations that are needed to reconstruct training examples.
- If the data-generating distribution concentrates near a low-dimensional manifold, this yields representations that implicitly capture a local coordinate system for the manifold.

LEARNING MANIFOLDS WITH AES

- Only the variations tangent to the manifold around $\boldsymbol{x}$ need to correspond to changes in $\boldsymbol{z} = f(\boldsymbol{x})$. Hence the encoder learns a mapping from the input space to a representation space that is only sensitive to changes along the manifold directions, but that is insensitive to changes orthogonal to the manifold.

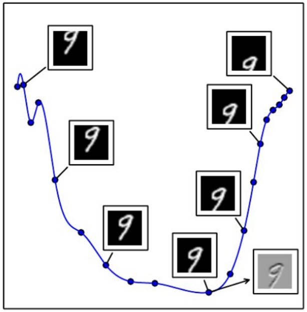


Figure: from Goodfellow et al. (2016)

EXPERIMENT: LEARN TO ENCODE MNIST

- Let us try to compress the MNIST data as good as possible.
- Therefore, we will fit an undercomplete autoencoder to learn the best possible representation,

  $\Rightarrow$ as few as possible dimensions in the internal representation **z**.

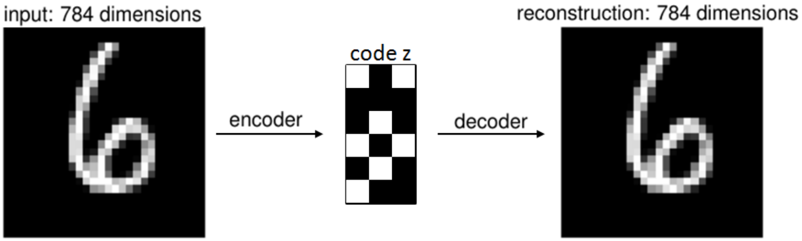


Figure: Flow chart of our our autoencoder: reconstruct the input with fixed dimensions $dim(\mathbf{z}) << dim(\mathbf{x})$.

EXPERIMENT: LEARN TO ENCODE MNIST

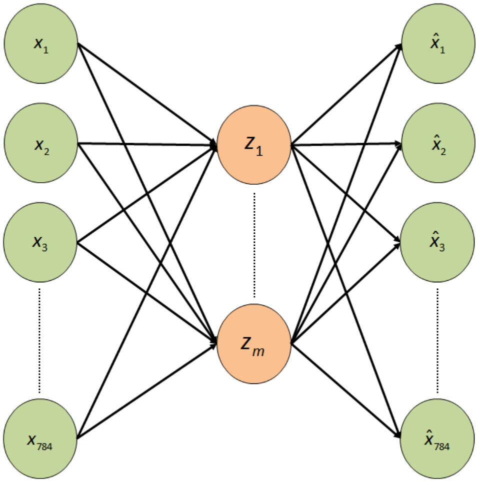


Figure: Architecture of the autoencoder.

EXPERIMENT: LEARN TO ENCODE MNIST

```
z = c(784, 256, 64, 32, 16, 8, 4, 2, 1)

input = mx.symbol.Variable("data") # mnist with 28x28 = 784
encoder = mx.symbol.FullyConnected(input, num_hidden = z[i])
decoder = mx.symbol.FullyConnected(encoder, num_hidden = 784)
activation = mx.symbol.Activation(decoder, "sigmoid")
output = mx.symbol.LinearRegressionOutput(activation)

model = mx.model.FeedForward.create(output,
  X = train.x, y = train.x,
  num.round = 50,
  array.batch.size = 32,
  optimizer = "adam",
  initializer = mx.init.uniform(0.01),
  eval.metric = mx.metric.mse
)
```

EXPERIMENT: LEARN TO ENCODE MNIST

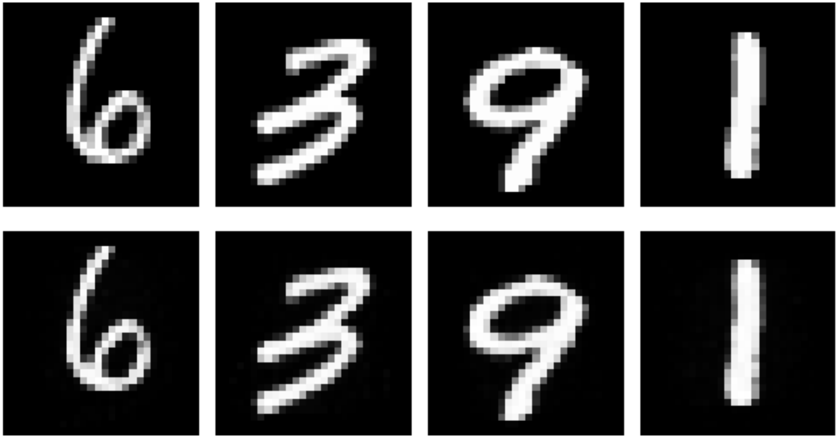


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(\mathbf{z}) = 784 = dim(\mathbf{x})$.

EXPERIMENT: LEARN TO ENCODE MNIST

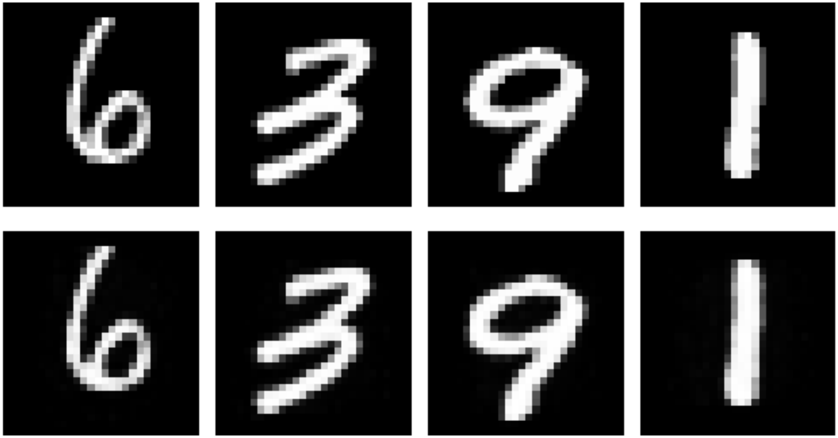


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(\mathbf{z}) = 256$.

EXPERIMENT: LEARN TO ENCODE MNIST

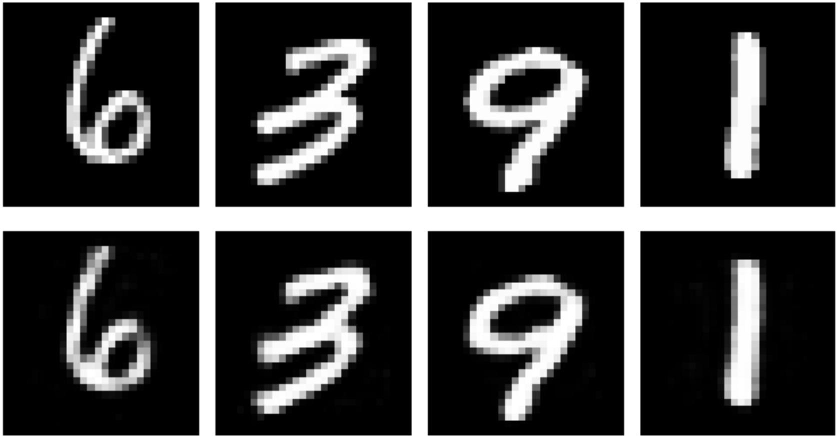


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(\mathbf{z}) = 64$.

EXPERIMENT: LEARN TO ENCODE MNIST

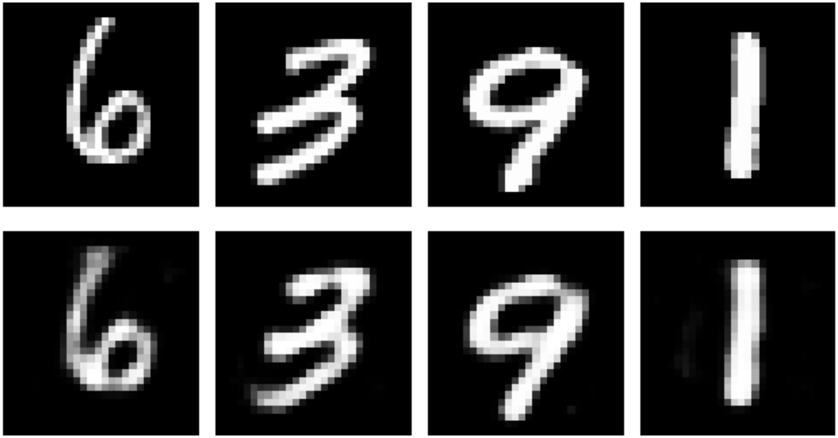


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(\mathbf{z}) = 32$.

EXPERIMENT: LEARN TO ENCODE MNIST

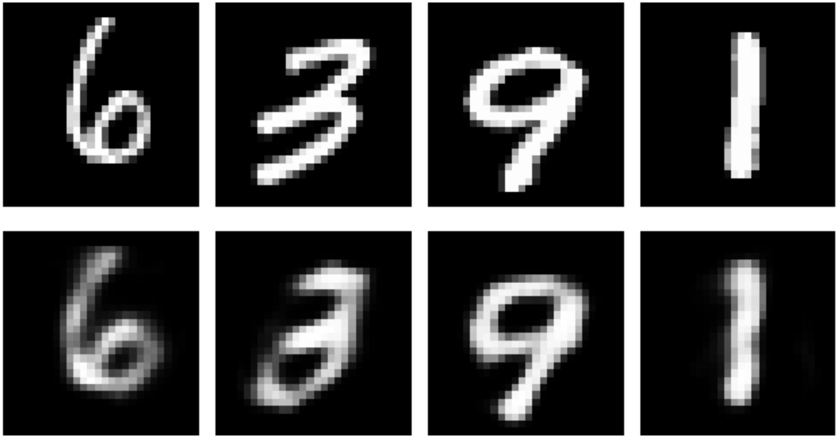


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(\mathbf{z}) = 16$.

EXPERIMENT: LEARN TO ENCODE MNIST

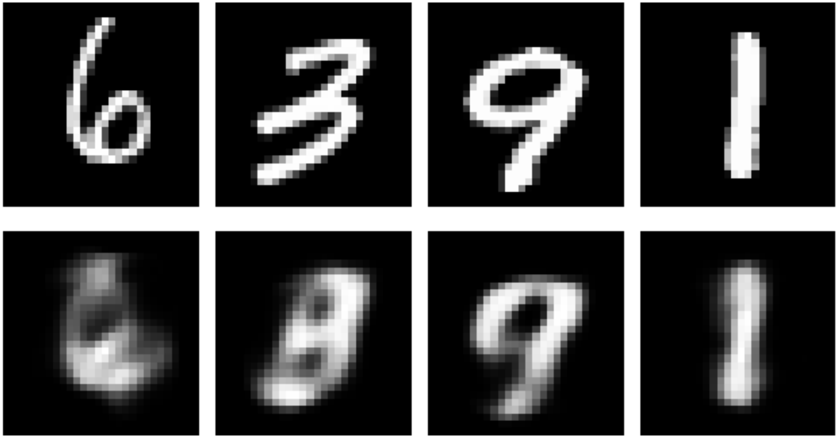


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(\mathbf{z}) = 8$.

EXPERIMENT: LEARN TO ENCODE MNIST

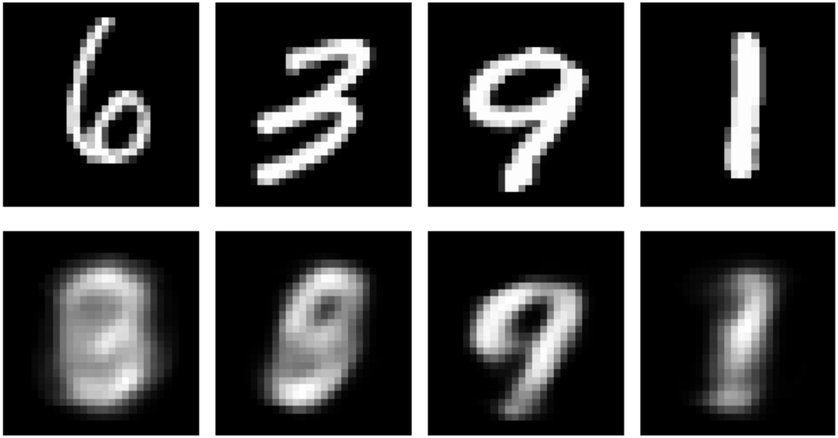


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(\mathbf{z}) = 4$.

EXPERIMENT: LEARN TO ENCODE MNIST

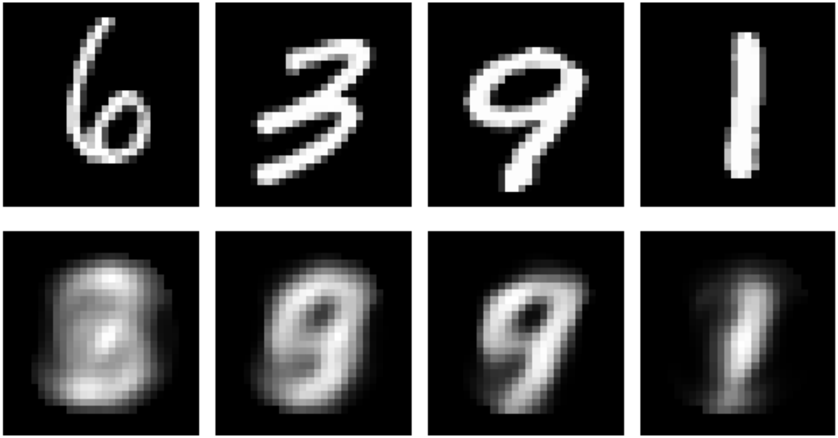


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(\mathbf{z}) = 2$.

EXPERIMENT: LEARN TO ENCODE MNIST

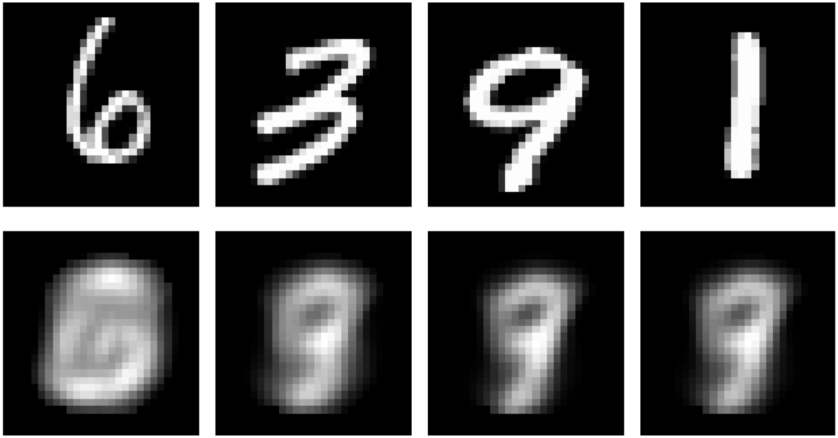


Figure: The top row shows the original digits, the bottom row the reconstructed ones.

- $dim(\mathbf{z}) = 1$.

Regularized Autoencoder

REGULARIZED AUTOENCODER

- Goal: choose code dimension and capacity of encoder/decoder based on the problem.
- **Regularized AEs** modify the original loss function to:
  - prevent the network from trivially copying the inputs.
  - encourage additional properties.
- Examples:
  - **Sparse AE:** sparsity of the representation.
  - **Denoising AE**: robustness to noise.
  - **Contractive AE**: small derivatives of the representation w.r.t. input.

$\Rightarrow$ A regularized AE can be overcomplete and nonlinear but still learn something useful about the data distribution!

DENOISING AUTOENCODER (DAE)

- Idea: representation should be robust to introduction of noise.
- Produce corrupted version $\tilde{\mathbf{x}}$ of input $\mathbf{x}$, e.g. by
  - random assignment of subset of inputs to 0.
  - adding Gaussian noise.
- Modified reconstruction loss: $L(\mathbf{x}, g(f(\tilde{\mathbf{x}})))$
  $\Rightarrow$ denoising AEs must learn to undo this corruption.

DENOISING AUTOENCODER (DAE)

- With the corruption process, we induce stochasticity into the DAE.
- Formally: let $C(\tilde{\mathbf{x}}|\mathbf{x})$ present the conditional distribution of corrupted samples $\tilde{\mathbf{x}}$, given a data sample $\mathbf{x}$.
- Like feedforward NNs can model a distribution over targets $p(\mathbf{y}|\mathbf{x})$, output units and loss function of an AE can be chosen such that one gets a stochastic decoder $p_{decoder}(\mathbf{x}|\mathbf{z})$.
- E.g. linear output units to parametrize the mean of Gaussian distribution for real valued **x** and negative log-likelihood loss (which is equal to MSE).
- The DAE then learns a reconstruction distribution $p_{reconstruct}(\mathbf{x}|\tilde{\mathbf{x}})$ from training pairs $(\mathbf{x}, \tilde{\mathbf{x}})$.
- (Note that the encoder could also be made stochastic, modelling $p_{encoder}(\mathbf{z}|\tilde{\mathbf{x}})$.)

DENOISING AUTOENCODER (DAE)

The general structure of a DAE as a computational graph:

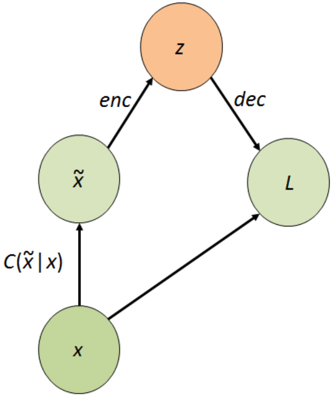


Figure: Denoising autoencoder: “making the learned representation robust to partial corruption of the input pattern.”

DENOISING AUTOENCODER (DAE)

Algorithm 1 Training denoising autoencoders

```
1: Sample a training example $\mathbf{x}$ from the training data.
2: Sample a corrupted version $\tilde{\mathbf{x}}$ from $C(\tilde{\mathbf{x}}|\mathbf{x})$
3: Use $(\mathbf{x}, \tilde{\mathbf{x}})$ as a training example for estimating the AE recon-
   struction $p_{reconstruct}(\mathbf{x}|\tilde{\mathbf{x}}) = p_{decoder}(\mathbf{x}|\mathbf{z})$, where
   - $\mathbf{z}$ is the output of the encoder $enc(\tilde{\mathbf{x}})$ and
   - $p_{decoder}$ defined by a decoder $dec(\mathbf{z})$
```

- All we have to do to transform an AE into a DAE is to add a stochastic corruption process on the input.
- The DAE still tries to preserve the information about the input (encode it), but also to undo the effect of a corruption process!

DENOISING AUTOENCODER (DAE)

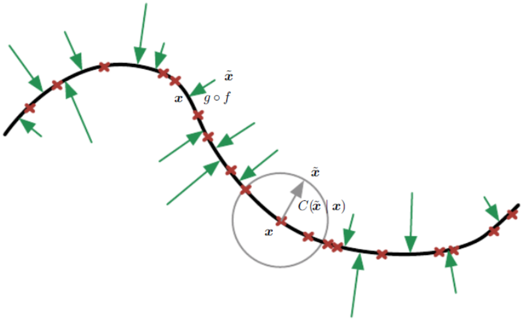


Figure: Denoising autoencoders - “manifold perspective” (Ian Goodfellow et al. (2016))

A DAE is trained to map a corrupted data point $\tilde{\mathbf{x}}$ back to the original data point $\mathbf{x}$.

DENOISING AUTOENCODER (DAE)

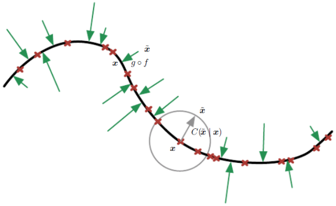


Figure: Denoising autoencoders - “manifold perspective” (Ian Goodfellow et al. (2016))

- The corruption process $C(\tilde{\mathbf{x}}|\mathbf{x})$ is displayed by the gray circle of equiprobable corruptions
- Training a DAE by minimizing $||dec(enc(\tilde{\mathbf{x}})) - \mathbf{x}||^2$ corresponds to minimizing $\mathbb{E}_{\mathbf{x},\tilde{\mathbf{x}} \sim p_{data}(\mathbf{x})C(\tilde{\mathbf{x}}|\mathbf{x})}[\log p_{decoder}(\mathbf{x}|f(\tilde{\mathbf{x}}))]$.

DENOISING AUTOENCODER (DAE)

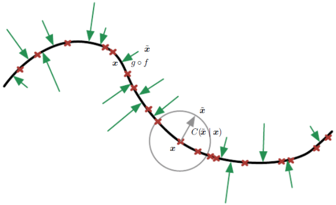


Figure: Denoising autoencoders - “manifold perspective” (Ian Goodfellow et al. (2016))

- The vector $dec(enc(\tilde{\mathbf{x}})) - \tilde{\mathbf{x}}$ points approximately towards the nearest point in the data manifold, since $dec(enc(\tilde{\mathbf{x}}))$ estimates the center of mass of clean points $\mathbf{x}$ which could have given rise to $\tilde{\mathbf{x}}$.
- Thus, the DAE learns a vector field $dec(enc(\mathbf{x})) - \mathbf{x}$ indicated by the green arrows.

DENOISING AUTOENCODER (DAE)

An example of a vector field learned by a DAE.

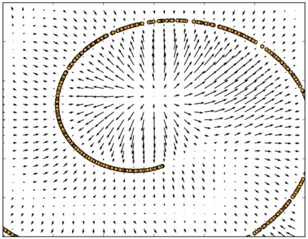


Figure: source : Ian Goodfellow et al. (2016)

EXPERIMENT: ENCODE MNIST WITH A DAE

- We will now corrupt the MNIST data with Uniform noise and then try to denoise it as good as possible.

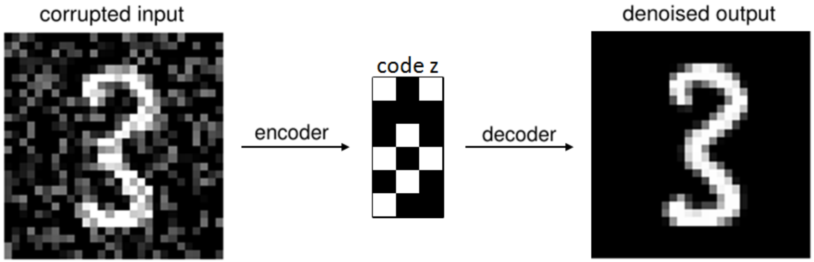


Figure: Flow chart of our our autoencoder: denoise the corrupted input.

EXPERIMENT: ENCODE MNIST WITH A DAE

- To corrupt the input, we randomly add or subtract values from a uniform distribution to each of the image entries.

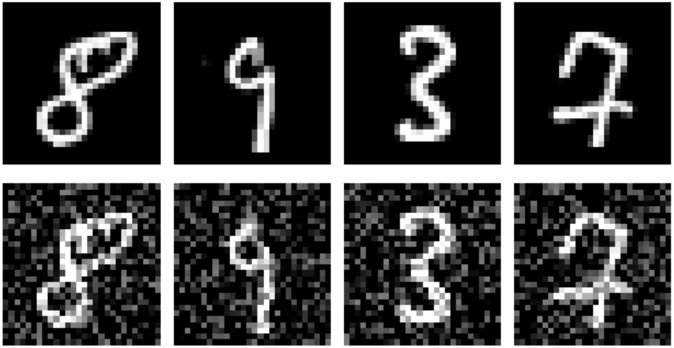
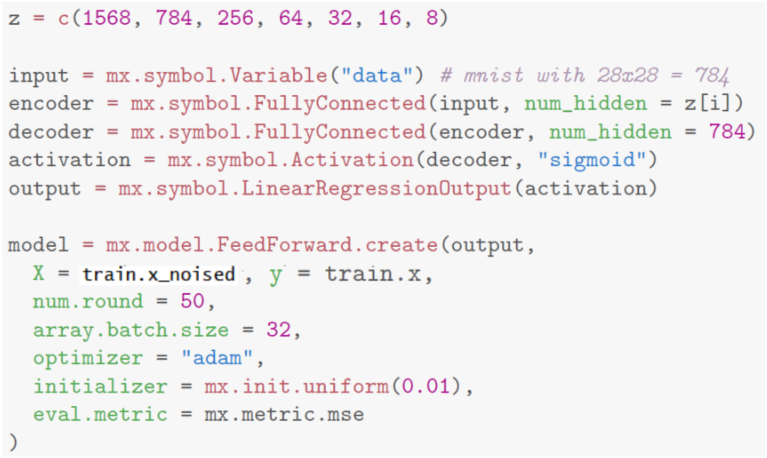


Figure: Top row: original data, bottom row: corrupted mnist data.

EXPERIMENT: ENCODE MNIST WITH A DAE



```r
z = c(1568, 784, 256, 64, 32, 16, 8)

input = mx.symbol.Variable("data") # mnist with 28x28 = 784
encoder = mx.symbol.FullyConnected(input, num_hidden = z[i])
decoder = mx.symbol.FullyConnected(encoder, num_hidden = 784)
activation = mx.symbol.Activation(decoder, "sigmoid")
output = mx.symbol.LinearRegressionOutput(activation)

model = mx.model.FeedForward.create(output,
  X = train.x_noised , y = train.x,
  num.round = 50,
  array.batch.size = 32,
  optimizer = "adam",
  initializer = mx.init.uniform(0.01),
  eval.metric = mx.metric.mse
)
```

EXPERIMENT: ENCODE MNIST WITH A DAE

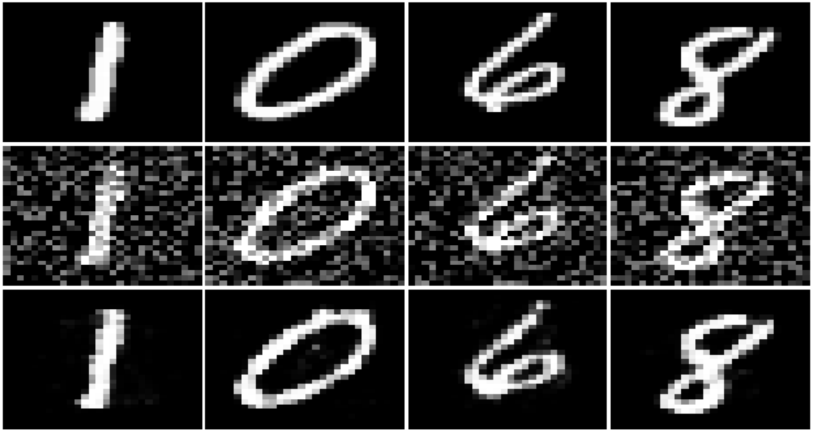


Figure: The top row shows the original digits, the intermediate one the corrupted and the bottom row the denoised/reconstructed digits (prediction).

- $dim(\boldsymbol{z}) = 1568$ (overcomplete).

EXPERIMENT: ENCODE MNIST WITH A DAE

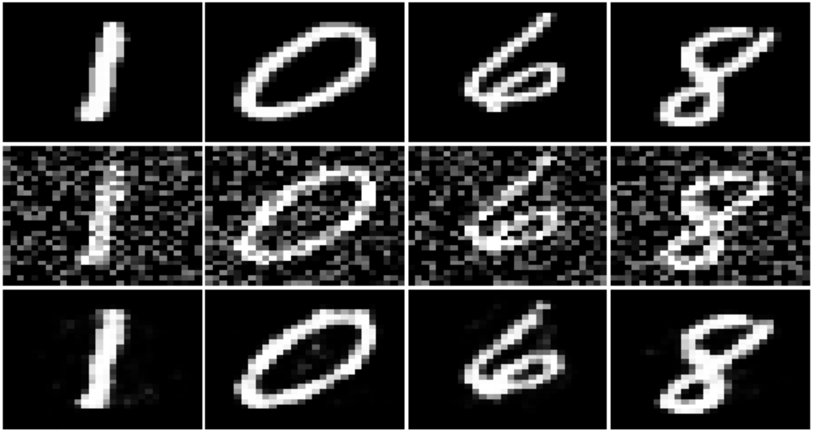


Figure: The top row shows the original digits, the intermediate one the corrupted and the bottom row the denoised/reconstructed digits (prediction).

- $dim(\mathbf{z}) = 784$ $(= dim(\mathbf{x}))$.

EXPERIMENT: ENCODE MNIST WITH A DAE

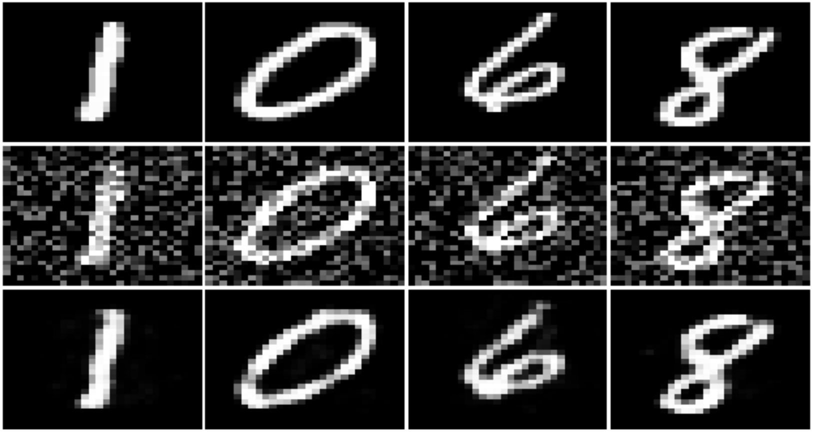


Figure: The top row shows the original digits, the intermediate one the corrupted and the bottom row the denoised/reconstructed digits (prediction).

- $dim(\boldsymbol{z}) = 256$.

EXPERIMENT: ENCODE MNIST WITH A DAE

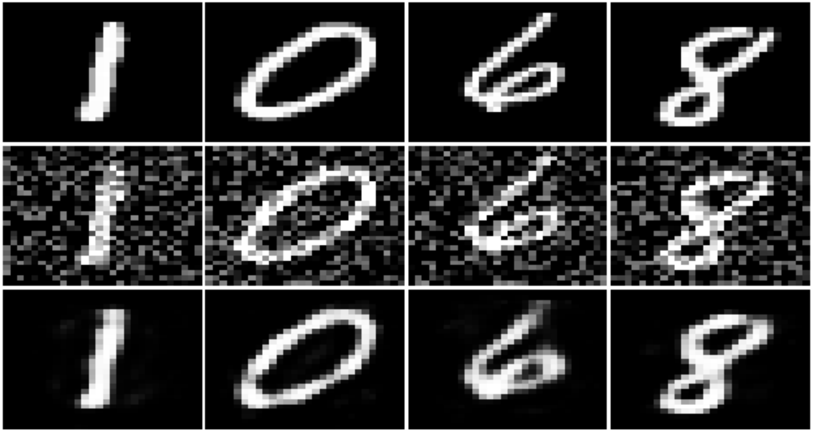


Figure: The top row shows the original digits, the intermediate one the corrupted and the bottom row the denoised/reconstructed digits (prediction).

- $dim(\boldsymbol{z}) = 64$.

EXPERIMENT: ENCODE MNIST WITH A DAE

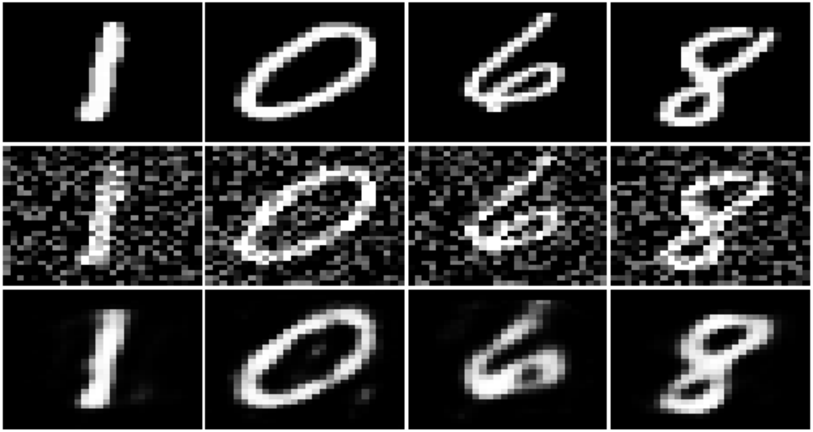


Figure: The top row shows the original digits, the intermediate one the corrupted and the bottom row the denoised/reconstructed digits (prediction).

- $dim(\boldsymbol{z}) = 32$.

EXPERIMENT: ENCODE MNIST WITH A DAE

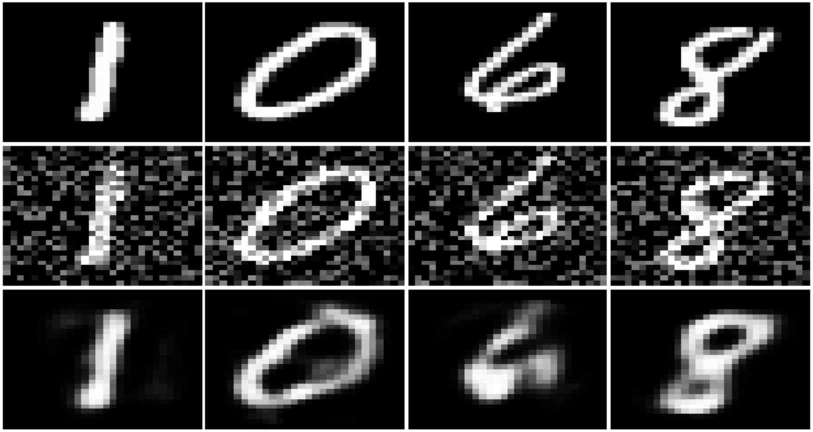


Figure: The top row shows the original digits, the intermediate one the corrupted and the bottom row the denoised/reconstructed digits (prediction).

- $dim(\boldsymbol{z}) = 16$.

EXPERIMENT: ENCODE MNIST WITH A DAE

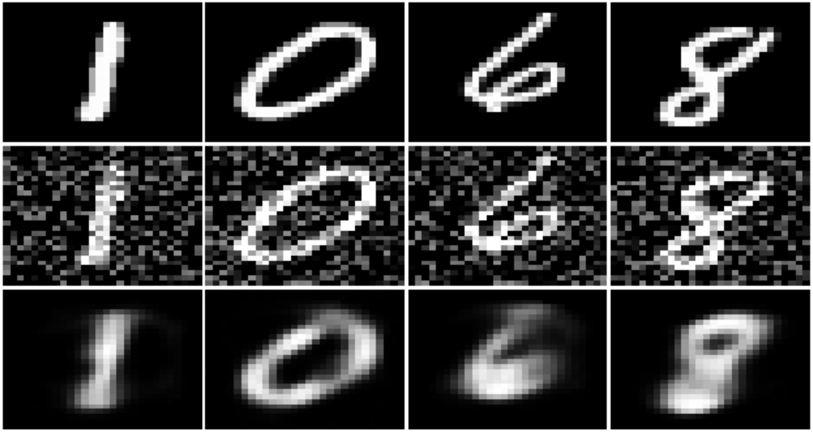


Figure: The top row shows the original digits, the intermediate one the corrupted and the bottom row the denoised/reconstructed digits (prediction).

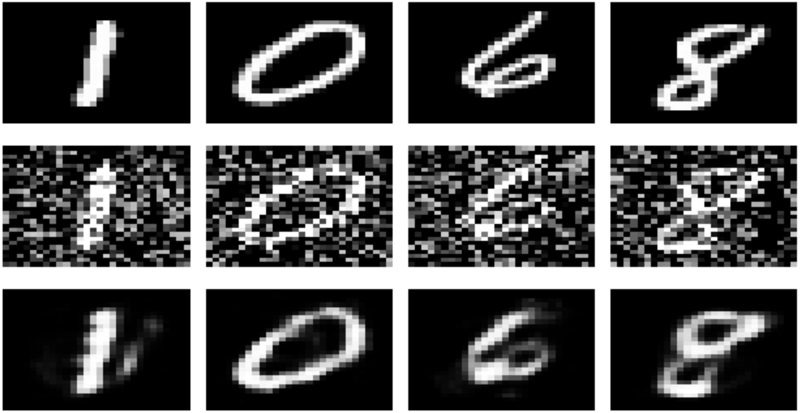
- $dim(\boldsymbol{z}) = 8$.

EXPERIMENT: ENCODE MNIST WITH A DAE

- Let us increase the amount of noise and see how the autoencoder with $dim(z) = 64$ deals with it (for science!).

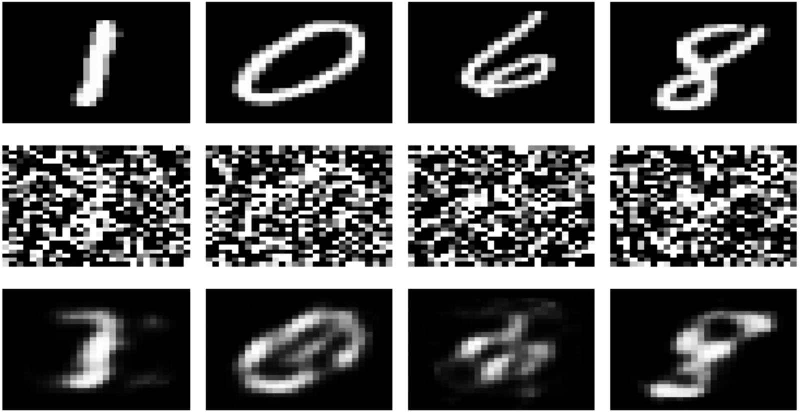
EXPERIMENT: ENCODE MNIST WITH A DAE

- A lot of noise.



EXPERIMENT: ENCODE MNIST WITH A DAE

- **A lot** of noise.



CONTRACTIVE AUTOENCODER

- Idea: extract features that only reflect variations found in the training set.
- Add explicit regularization term to the reconstruction loss:

$$L(\mathbf{x}, g(f(\mathbf{x}))) + \lambda \| \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \|_F^2$$

$\Rightarrow$ Derivatives of the encoder function w.r.t. the input are encouraged to be small.
$\Rightarrow$ Only a small number of input directions will have significant derivatives.
$\Rightarrow$ The encoder function is encouraged to resist infinitesimal perturbations of the input.

DAE VS. CAE

DAE	**CAE**
the *decoder* function is trained to resist infinitesimal perturbations of the input.	the *encoder* function is trained to resist infinitesimal perturbations of the input.

WHICH AUTOENCODER?

- Both the denoising and contractive autoencoders perform well.
- Advantage of denoising autoencoder: simpler to implement
  - requires adding one or two lines of code to regular AE.
  - no need to compute Jacobian of hidden layer.
- Advantage of contractive autoencoder: gradient is deterministic
  - can use second order optimizers (conjugate gradient, LBFGS, etc.).
  - might be more stable than the denoising autoencoder, which uses a sampled gradient.

Specific AEs and applications

CONVOLUTIONAL AUTOENCODER (CONVAE)

- For the image domain, using convolutions is advantageous. Can we also make use of them in AEs?
- In a ConvAE, the encoder consists of convolutional layers. The decoder, on the other hand, consists of transpose convolution layers or simple upsampling operations.

CONVOLUTIONAL AUTOENCODER (CONVAE)

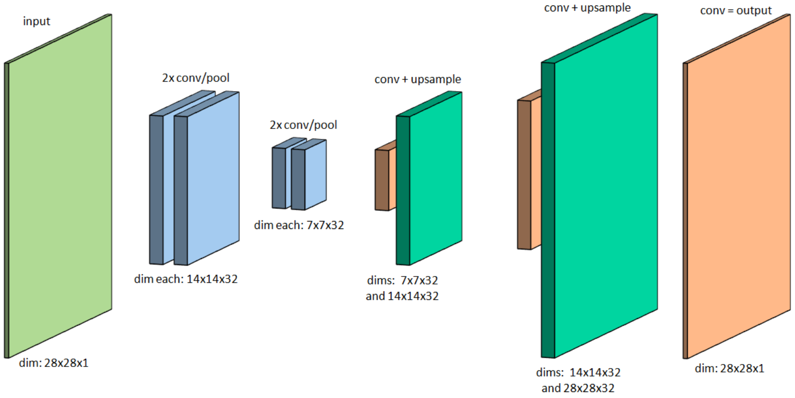


Figure: Potential architecture of a convolutional autoencoder.

We now apply this architecture to denoise MNIST.

CONVOLUTIONAL AUTOENCODER (CONVAE)

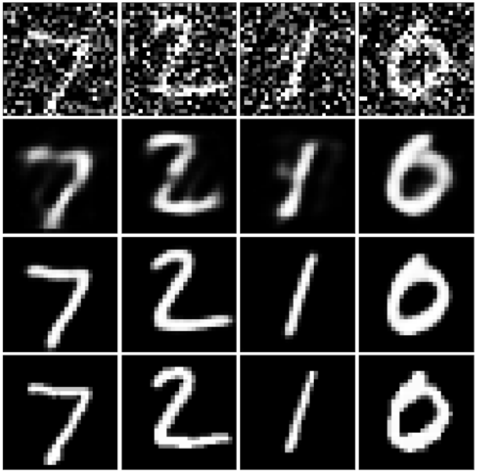


Figure: Top row: noised data, second row: AE with $\mathit{dim}(\boldsymbol{z}) = 32$ (roughly 50k params), third row: ConvAE (roughly 25k params), fourth row: ground truth.

AES FOR UNSUPERVISED PRETRAINING

- Stacked AEs can be used for layer-wise unsupervised pretraining of deep neural networks.
- This corresponds to subsequently training each layer as an AE.
- It aims at yielding better weight initializations for the actual supervised training.
- This usually eliminates the risk of vanishing gradients in feed forward nets.
- It played an important role in the past before general techniques for stabilizing optimization were invented (e.g. ReLUs, batch normalization, dropout, etc.)

REAL-WORLD APPLICATIONS

Today, autoencoders are still used for tasks such as:

- data de-noising,
- compression,
- and dimensionality reduction for the purpose of visualization.

REAL-WORLD APPLICATIONS

Medical image denoising using convolutional denoising autoencoders

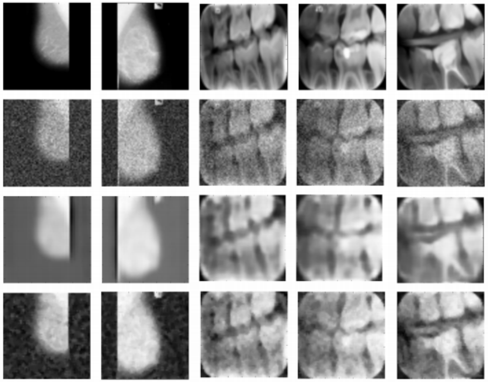


Figure: Top row : real image, second row : noisy version, third row : results of a (convolutional) denoising autoencoder and fourth row : results of a median filter (Lovedeep Gondara (2016))

REAL-WORLD APPLICATIONS

AE-based **image compression**.

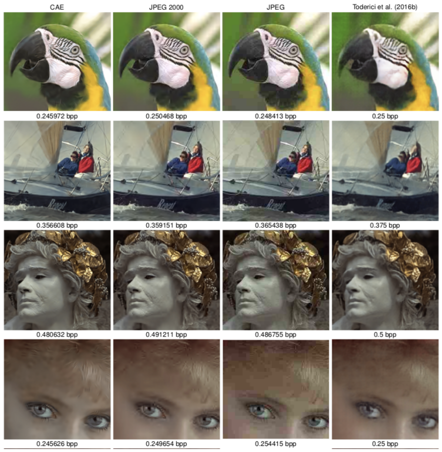


Figure: from Theis et al.

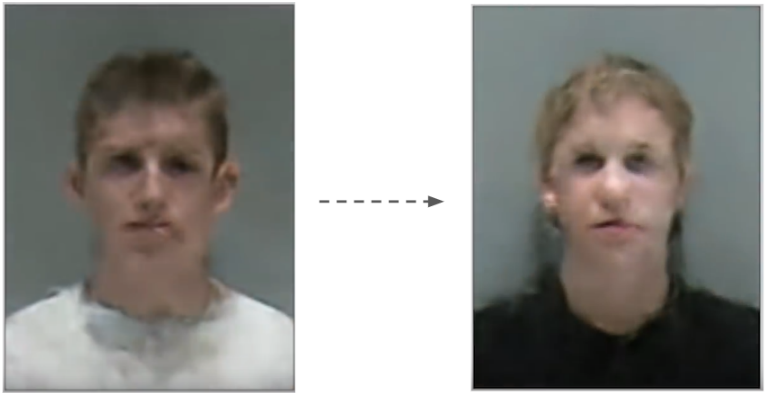
REAL-WORLD APPLICATIONS

Latent space traversal:

- Once an AE has been trained to compress images, it's possible to "generate" new images by feeding arbitrary latent vectors $\mathbf{z}$ to the decoder.
- In fact, certain axes/directions in the latent space can correspond to interpretable factors of variation in the images generated.
- In this example, the decoder was trained to map latent code to high-school yearbook photos and interesting axes were discovered by performing PCA on the latent code of the training data.
- Click here for a video of the experiment.

REAL-WORLD APPLICATIONS

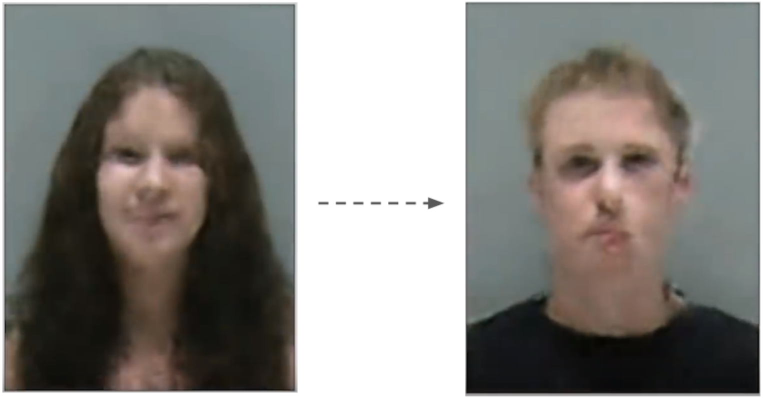
Latent space traversal:



Moving along one particular direction in the latent space caused a change primarily in the color of the shirt in the generated image.

REAL-WORLD APPLICATIONS

Latent space traversal:



A different direction in the latent space seemed to correpond to the gender of the person in the generated image.

REFERENCES

Ian Goodfellow, Yoshua Bengio and Aaron Courville (2016)
Deep Learning
http://www.deeplearningbook.org/

Lovedeep Gondara (2016)
Medical image denoising using convolutional denoising autoencoders
https://arxiv.org/abs/1608.04667

Vijay Badrinarayanan, Alex Kendall and Roberto Cipolla (2016)
SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation
https://arxiv.org/abs/1511.00561

Theis, Lucas; Shi, Wenzhe; Cunningham, Andrew; Huszár, Ferenc (2017)
Lossy Image Compression with Compressive Autoencoders
https://arxiv.org/abs/1703.00395