Deep Learning

Vazgen Mikayelyan

YSU, Krisp

November 26, 2019

Outline

Dilated/Atrous Convolution

Definition 1

Let $F : \mathbb{Z}^2 \to \mathbb{R}$ *be a discrete function. Let* $\Omega_r : [-r, r] \cap \mathbb{Z}^2$ *and let* $k : \Omega_r \to \mathbb{R}$ *be a discrete filter of size* $(2r + 1)^2$*. The discrete convolution operator* $*$ *can be defined as*

$$(F * k)(p) = \sum_{s+t=p} F(s)\, k(t)$$

Dilated/Atrous Convolution

Definition 1

Let $F : \mathbb{Z}^2 \to \mathbb{R}$ *be a discrete function. Let* $\Omega_r : [-r, r] \cap \mathbb{Z}^2$ *and let* $k : \Omega_r \to \mathbb{R}$ *be a discrete filter of size* $(2r+1)^2$*. The discrete convolution operator* $*$ *can be defined as*

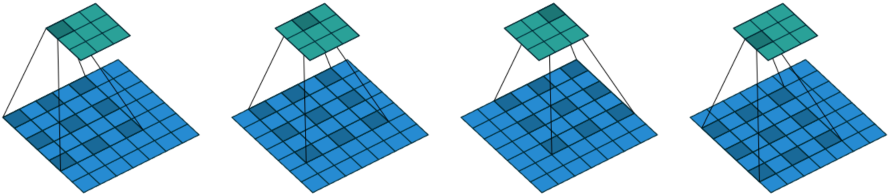
$$(F * k)(p) = \sum_{s+t=p} F(s)\, k(t)$$

Definition 2

Let $F : \mathbb{Z}^2 \to \mathbb{R}$ *be a discrete function. Let* $\Omega_r : [-r, r] \cap \mathbb{Z}^2$ *and let* $k : \Omega_r \to \mathbb{R}$ *be a discrete filter of size* $(2r+1)^2$*. The discrete l-dilated convolution operator* $*_l$ *can be defined as*

$$(F *_l k)(p) = \sum_{s+lt=p} F(s)\, k(t)$$

Dilated/Atrous Convolution



1D Dilated Convolution

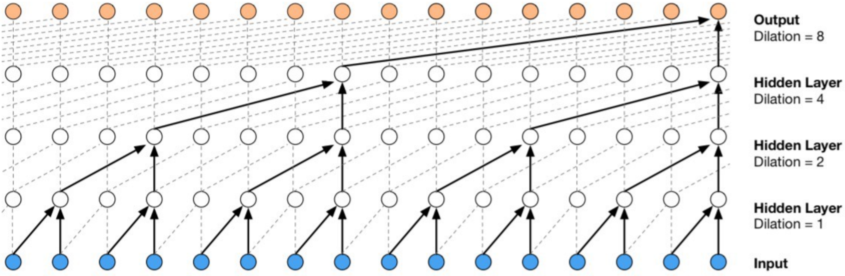
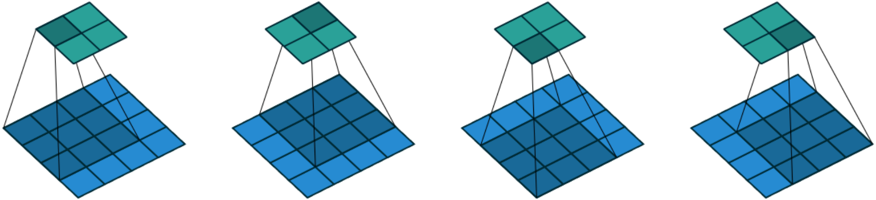
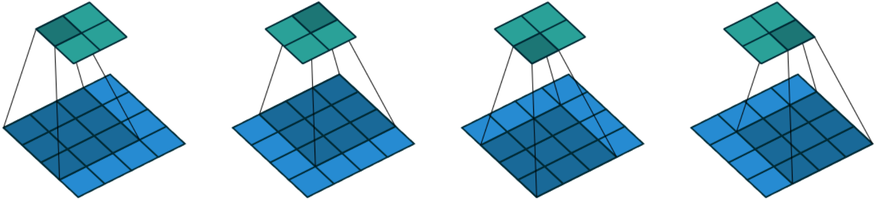


Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

Convolution as a Matrix Operation



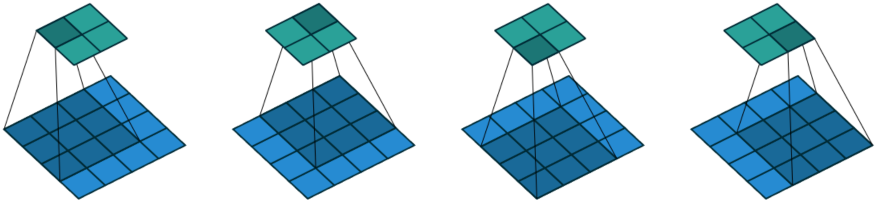
Convolution as a Matrix Operation



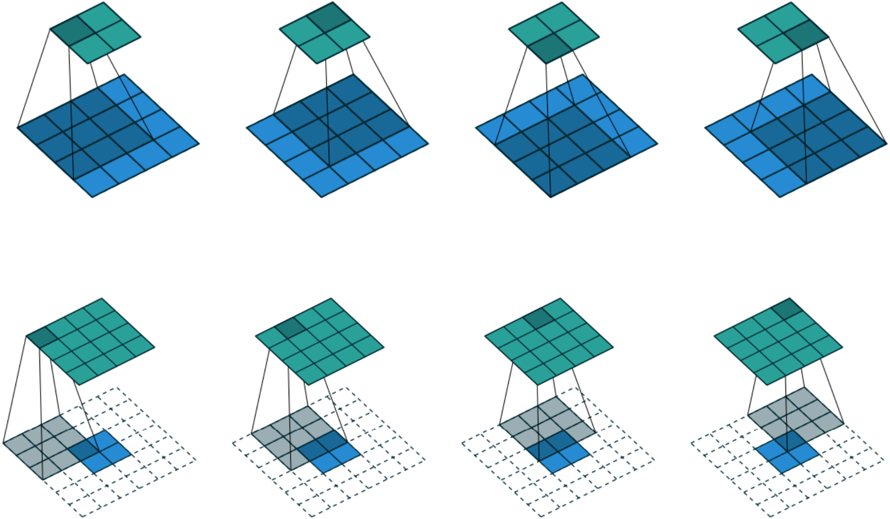
$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

This linear operation takes the input matrix flattened as a 16-dimensional vector and produces a 4-dimensional vector that is later reshaped as the 2×2 output matrix.

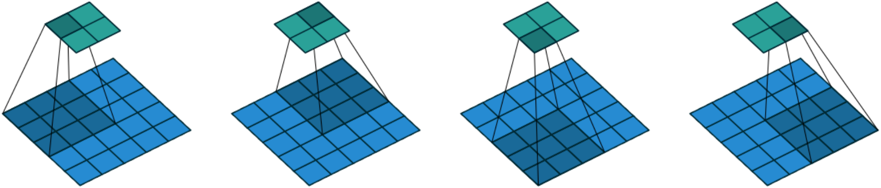
Transposed Convolution (stride=0)



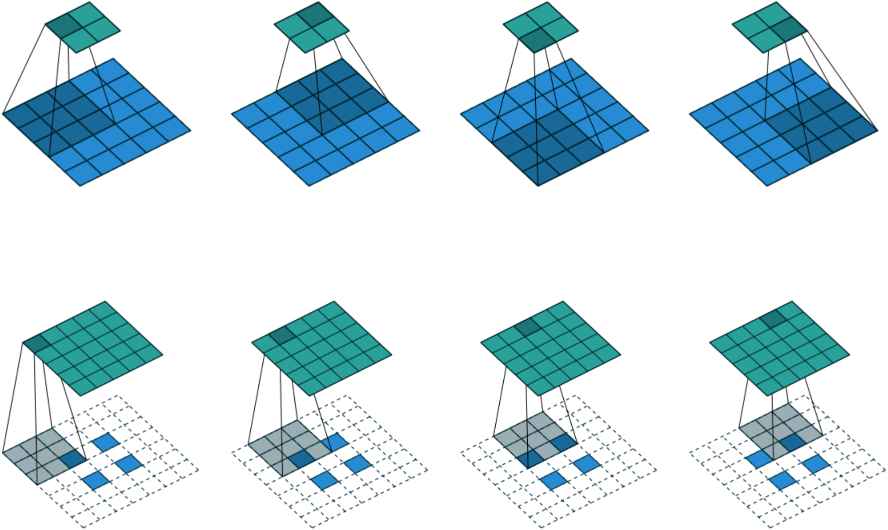
Transposed Convolution (stride=0)



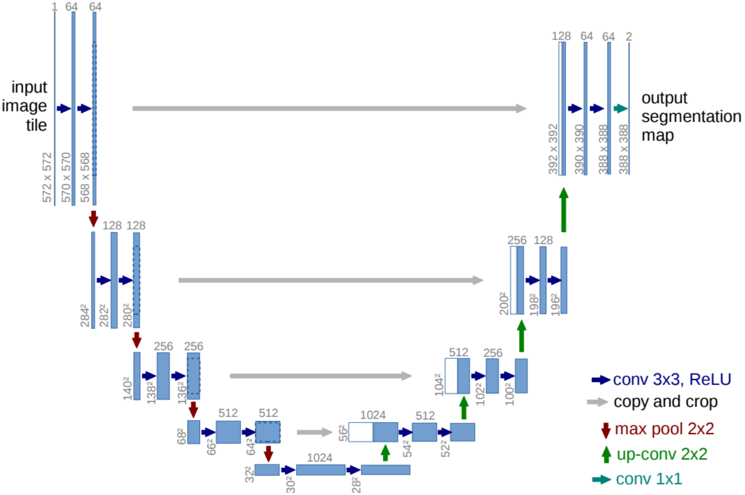
Transposed Convolution (stride=1)



Transposed Convolution (stride=1)



UNet



1
64
64
input
image
tile
572 x 572
570 x 570
568 x 568
128
128
284²
282²
280²
256
256
140²
138²
136²
512
512
68²
66²
64²
1024
32²
30²
28²
1024
56²
54²
512
52²
512
104²
102²
256
100²
256
200²
128
198²
196²
128
64
64
2
392 x 392
390 x 390
388 x 388
388 x 388
output
segmentation
map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

Outline

KL Divergence

The KL divergence (also called relative entropy) is a measure of how one probability distribution is different from a second, reference probability distribution:

$$KL\left(P||Q\right) = -\int_{-\infty}^{+\infty} p\left(x\right)\log\frac{q\left(x\right)}{p\left(x\right)}dx$$

KL Divergence

The KL divergence (also called relative entropy) is a measure of how one probability distribution is different from a second, reference probability distribution:

$$KL(P||Q) = -\int_{-\infty}^{+\infty} p(x) \log \frac{q(x)}{p(x)} dx = \mathbb{E}_p \left[\log \frac{p}{q} \right],$$

where p and q are probability density functions of distributions P and Q.

KL Divergence

The KL divergence (also called relative entropy) is a measure of how one probability distribution is different from a second, reference probability distribution:

$$KL\left(P||Q\right) = -\int_{-\infty}^{+\infty} p\left(x\right)\log\frac{q\left(x\right)}{p\left(x\right)}dx = \mathbb{E}_p\left[\log\frac{p}{q}\right],$$

where p and q are probability density functions of distributions P and Q.
It can be proved that for all probability distributions P, Q, R

- $KL\left(P||Q\right) \geq 0$,

KL Divergence

The KL divergence (also called relative entropy) is a measure of how one probability distribution is different from a second, reference probability distribution:

$$KL\left(P||Q\right) = -\int\limits_{-\infty}^{+\infty} p\left(x\right)\log\frac{q\left(x\right)}{p\left(x\right)}dx = \mathbb{E}_p\left[\log\frac{p}{q}\right],$$

where p and q are probability density functions of distributions P and Q.
It can be proved that for all probability distributions P, Q, R

- $KL\left(P||Q\right) \geq 0$,
- $KL\left(P||Q\right) = 0$ if and only if $P = Q$,

KL Divergence

The KL divergence (also called relative entropy) is a measure of how one probability distribution is different from a second, reference probability distribution:

$$KL\left(P||Q\right) = -\int_{-\infty}^{+\infty} p\left(x\right)\log\frac{q\left(x\right)}{p\left(x\right)}dx = \mathbb{E}_{p}\left[\log\frac{p}{q}\right],$$

where p and q are probability density functions of distributions P and Q. It can be proved that for all probability distributions P, Q, R

- $KL\left(P||Q\right) \geq 0$,
- $KL\left(P||Q\right) = 0$ if and only if $P = Q$,
- $KL\left(P||Q\right) \leq K\left(P||R\right) + K\left(R||Q\right)$,

KL Divergence

The KL divergence (also called relative entropy) is a measure of how one probability distribution is different from a second, reference probability distribution:

$$KL\left(P||Q\right) = -\int_{-\infty}^{+\infty} p\left(x\right) \log \frac{q\left(x\right)}{p\left(x\right)} dx = \mathbb{E}_p \left[\log \frac{p}{q}\right],$$

where p and q are probability density functions of distributions P and Q. It can be proved that for all probability distributions P, Q, R

- $KL\left(P||Q\right) \geq 0$,
- $KL\left(P||Q\right) = 0$ if and only if $P = Q$,
- $KL\left(P||Q\right) \leq K\left(P||R\right) + K\left(R||Q\right)$,

but there is no symmetry, i.e. $K\left(P||Q\right) \neq K\left(Q||P\right)$.

Jensen-Shannon Divergence

JS Divergence is the following

$$JS\left(P||Q\right) = \frac{1}{2}K\left(P||M\right) + \frac{1}{2}K\left(M||Q\right),$$

where $M = \dfrac{P+Q}{2}$.

KL Divergence for Gaussians

KL Divergence for Gaussians

Recall that probability density function (if it exists) of multivariate normal distribution with mean μ and with (non-singular, symmetric, positive definite) covariance matrix Σ is the following function:

$$f(x) = \frac{\exp\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right\}}{\sqrt{(2\pi)^k |\Sigma|}}, x \in \mathbb{R}^k.$$

KL Divergence for Gaussians

Recall that probability density function (if it exists) of multivariate normal distribution with mean μ and with (non-singular, symmetric, positive definite) covariance matrix Σ is the following function:

$$f(x) = \frac{\exp\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right\}}{\sqrt{(2\pi)^k |\Sigma|}}, x \in \mathbb{R}^k.$$

Suppose that we have two multivariate normal distributions: $\mathcal{N}_1(\mu_1, \Sigma_1), \mathcal{N}_2(\mu_2, \Sigma_2)$. Then

$$KL(\mathcal{N}_1, \mathcal{N}_2) = \frac{1}{2}\left(\operatorname{tr}\left(\Sigma_2^{-1}\Sigma_1\right) + (\mu_2 - \mu_1)^{\mathsf{T}}\Sigma_2^{-1}(\mu_2 - \mu_1) - k + \ln\frac{|\Sigma_2|}{|\Sigma_1|}\right).$$

KL Divergence for Gaussians

Recall that probability density function (if it exists) of multivariate normal distribution with mean μ and with (non-singular, symmetric, positive definite) covariance matrix Σ is the following function:

$$f(x) = \frac{\exp\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right\}}{\sqrt{(2\pi)^k |\Sigma|}}, x \in \mathbb{R}^k.$$

Suppose that we have two multivariate normal distributions: $\mathcal{N}_1(\mu_1, \Sigma_1), \mathcal{N}_2(\mu_2, \Sigma_2)$. Then

$$KL(\mathcal{N}_1, \mathcal{N}_2) = \frac{1}{2}\left(\operatorname{tr}\left(\Sigma_2^{-1}\Sigma_1\right) + (\mu_2-\mu_1)^\mathsf{T}\Sigma_2^{-1}(\mu_2-\mu_1) - k + \ln\frac{|\Sigma_2|}{|\Sigma_1|}\right).$$

In one dimensional case we will have

$$KL(\mathcal{N}_1, \mathcal{N}_2) = \log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1-\mu_2)^2}{2\sigma_2^2} - \frac{1}{2}.$$

Outline

What is Unsupervised Learning?

What is Unsupervised Learning?

Definition 3

Unsupervised learning is a machine learning technique that finds and analyzes hidden patterns in unlabeled data.

What is Unsupervised Learning?

Definition 3

Unsupervised learning is a machine learning technique that finds and analyzes hidden patterns in unlabeled data.

Examples?

Autoencoders

Autoencoders are neural networks that aims to copy their inputs to their outputs. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation.

Autoencoders

Autoencoders are neural networks that aims to copy their inputs to their outputs. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation. This kind of network is composed of two parts

- **Encoder**:

Autoencoders

Autoencoders are neural networks that aims to copy their inputs to their outputs. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation. This kind of network is composed of two parts

- **Encoder**:
  This is the part of the network that compresses the input into a latent-space representation.

Autoencoders

Autoencoders are neural networks that aims to copy their inputs to their outputs. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation. This kind of network is composed of two parts

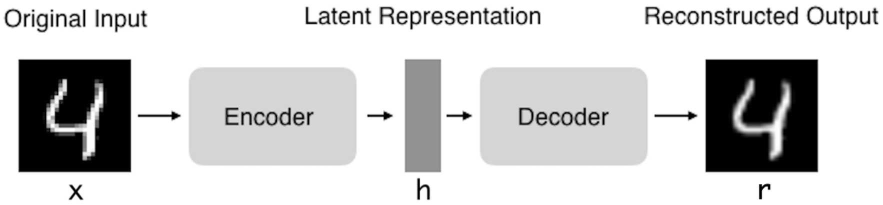
- **Encoder**:
  This is the part of the network that compresses the input into a latent-space representation.
- **Decoder**:

Autoencoders

Autoencoders are neural networks that aims to copy their inputs to their outputs. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation. This kind of network is composed of two parts

- **Encoder**:
  This is the part of the network that compresses the input into a latent-space representation.
- **Decoder**:
  This part aims to reconstruct the input from the latent space representation.

Autoencoders



What are autoencoders used for?

- Anomaly/Outlayer detection.

What are autoencoders used for?

- Anomaly/Outlayer detection.
- Dimensionality reduction.

What are autoencoders used for?

- Anomaly/Outlayer detection.
- Dimensionality reduction.
- Data denoising.

What are autoencoders used for?

- Anomaly/Outlayer detection.
- Dimensionality reduction.
- Data denoising.
- In a lot of different tasks.

Types of Autoencoders

- Vanilla Autoencoders

Types of Autoencoders

- Vanilla Autoencoders
- Multilayer/Deep Autoencoders

Types of Autoencoders

- Vanilla Autoencoders
- Multilayer/Deep Autoencoders
- Convolutional Autoencoders

Types of Autoencoders

- Vanilla Autoencoders
- Multilayer/Deep Autoencoders
- Convolutional Autoencoders
- Contractive Autoencoders

Types of Autoencoders

- Vanilla Autoencoders
- Multilayer/Deep Autoencoders
- Convolutional Autoencoders
- Contractive Autoencoders
- Regularized Autoencoders

Types of Autoencoders

- Vanilla Autoencoders
- Multilayer/Deep Autoencoders
- Convolutional Autoencoders
- Contractive Autoencoders
- Regularized Autoencoders
    - Sparse Autoencoders

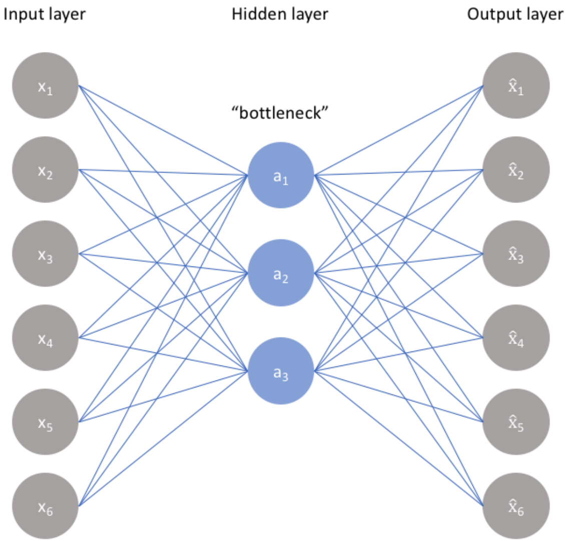
Types of Autoencoders

- Vanilla Autoencoders
- Multilayer/Deep Autoencoders
- Convolutional Autoencoders
- Contractive Autoencoders
- Regularized Autoencoders
  - Sparse Autoencoders
  - Denoising Autoencoders

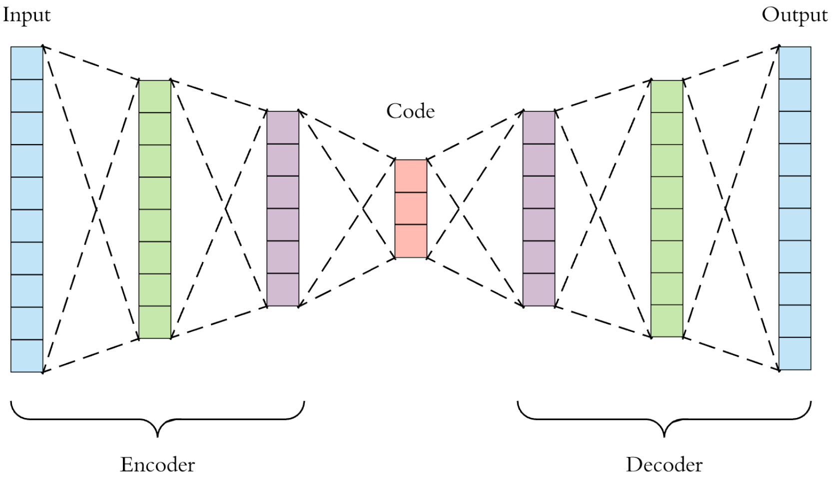
Types of Autoencoders

- Vanilla Autoencoders
- Multilayer/Deep Autoencoders
- Convolutional Autoencoders
- Contractive Autoencoders
- Regularized Autoencoders
    - Sparse Autoencoders
    - Denoising Autoencoders
- Variational Autoencoders

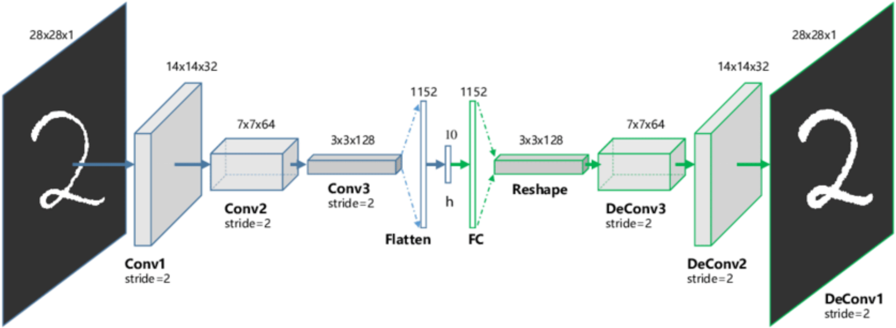
Vanilla Autoencoders



Multilayer/Deep Autoencoders



Convolutional Autoencoders



Contractive Autoencoders

- A contractive autoencoder makes this encoding less sensitive to small variations in its training dataset.

Contractive Autoencoders

- A contractive autoencoder makes this encoding less sensitive to small variations in its training dataset.
- This is accomplished by adding a regularizer, or penalty term, to whatever cost or objective function the algorithm is trying to minimize.

Contractive Autoencoders

- A contractive autoencoder makes this encoding less sensitive to small variations in its training dataset.
- This is accomplished by adding a regularizer, or penalty term, to whatever cost or objective function the algorithm is trying to minimize.
- The end result is to reduce the learned representation's sensitivity towards the training input.

Contractive Autoencoders

Let f is our encoder, g is the decoder and D is our training dataset. In the previous cases we minimize this kind of loss function:

$$\sum_{x \in D} L\left(x, g\left(f\left(x\right)\right)\right).$$

Contractive Autoencoders

Let f is our encoder, g is the decoder and D is our training dataset. In the previous cases we minimize this kind of loss function:

$$\sum_{x \in D} L\left(x, g\left(f\left(x\right)\right)\right).$$

In the case of contractive autoencoders we will minimize this one

$$\sum_{x \in D} \left(L\left(x, g\left(f\left(x\right)\right)\right) + \lambda \left\| J_f\left(x\right) \right\|_F^2 \right),$$

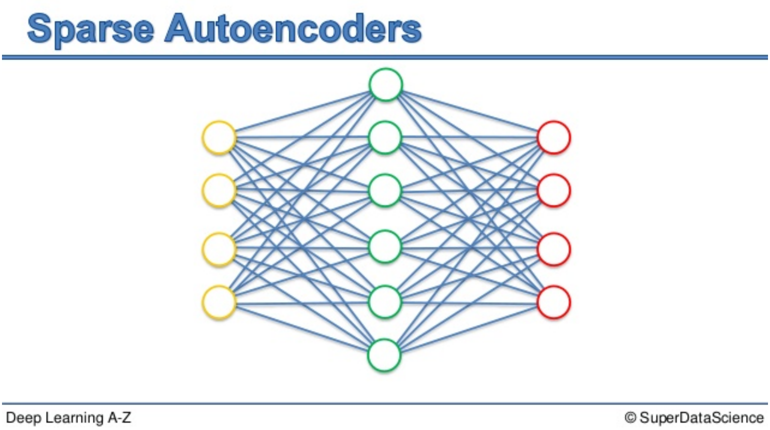
where the added summand is the square of Frobenius norm of the following Jacobian matrix:

$$\left[J_f\left(x\right)\right]_{i,j} = \frac{\partial f_j\left(x\right)}{\partial x_i}$$

i.e.

$$\left\| J_f\left(x\right) \right\|_F^2 = \sum_{i,j} \left(\frac{\partial f_j\left(x\right)}{\partial x_i} \right)^2.$$

Sparse Autoencoders



Sparse Autoencoders

- Sparse autoencoders have hidden nodes greater than input nodes. They can still discover important features from the data.

Sparse Autoencoders

- Sparse autoencoders have hidden nodes greater than input nodes. They can still discover important features from the data.
- Sparsity penalty is introduced on the hidden layer. This is to prevent output layer copy input data. This prevents overfitting.

Sparse Autoencoders

Let f is our encoder, g is the decoder and D is our training dataset, which has n samples. Denote

$$\rho_j = \frac{1}{n} \sum_{x \in D} f_j\left(x\right).$$

Sparse Autoencoders

Let f is our encoder, g is the decoder and D is our training dataset, which has n samples. Denote

$$\rho_j = \frac{1}{n} \sum_{x \in D} f_j\left(x\right).$$

We would like to (approximately) enforce the constraint $\rho_j = \rho$, where ρ is a sparsity parameter, typically a small value close to zero (say $\rho = 0.05$).

Sparse Autoencoders

Let f is our encoder, g is the decoder and D is our training dataset, which has n samples. Denote

$$\rho_j = \frac{1}{n} \sum_{x \in D} f_j(x).$$

We would like to (approximately) enforce the constraint $\rho_j = \rho$, where ρ is a sparsity parameter, typically a small value close to zero (say $\rho = 0.05$). To achieve this we will minimize the following loss function

$$\sum_{x \in D} L(x, g(f(x))) + \lambda \sum_j KL(\rho || \rho_j),$$

Sparse Autoencoders

Let f is our encoder, g is the decoder and D is our training dataset, which has n samples. Denote

$$\rho_j = \frac{1}{n} \sum_{x \in D} f_j(x).$$

We would like to (approximately) enforce the constraint $\rho_j = \rho$, where ρ is a sparsity parameter, typically a small value close to zero (say $\rho = 0.05$). To achieve this we will minimize the following loss function

$$\sum_{x \in D} L(x, g(f(x))) + \lambda \sum_j KL(\rho || \rho_j),$$

where

$$KL(\rho || \rho_j) = -\rho \log \frac{\rho_j}{\rho} - (1-\rho) \log \frac{1-\rho_j}{1-\rho}.$$