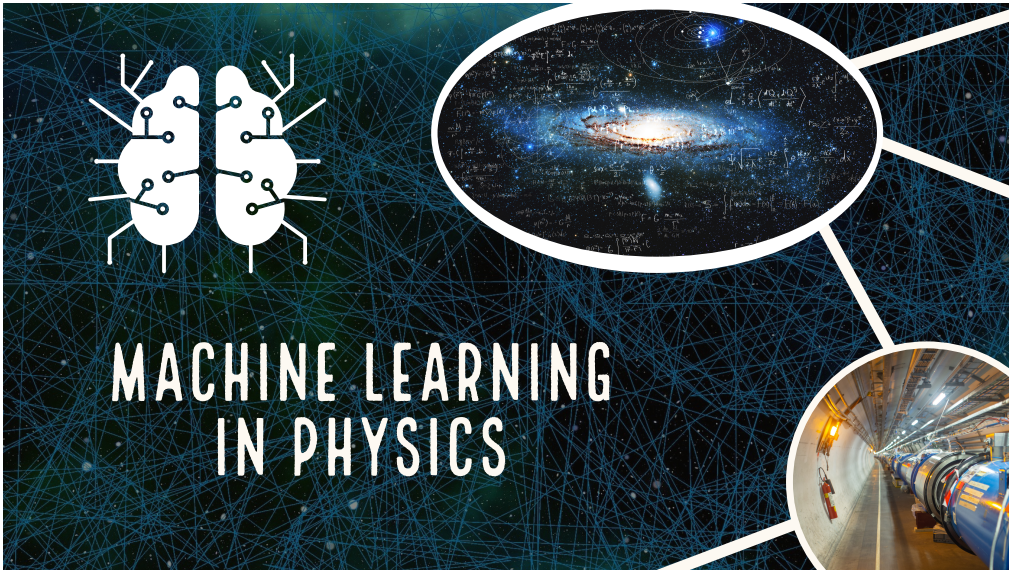
Machine Learning in Physics: Neural Networks and Deep Learning

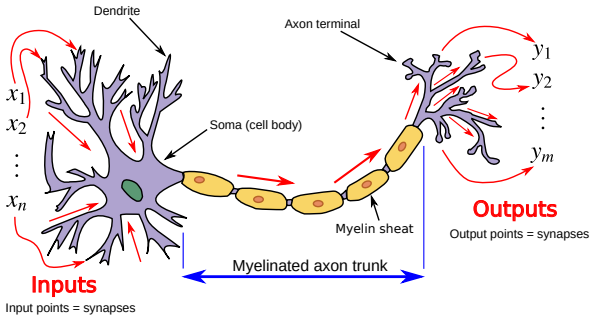
Selected chapters on astrophysics

Pavel Baláž & Martin Žonda

Department of Condensed Matter Physics, Charles University, Prague


MACHINE LEARNING
IN PHYSICS

Biological neuron



source: Wikipedia

- generate output only if the input signal is strong enough
- otherwise do nothing

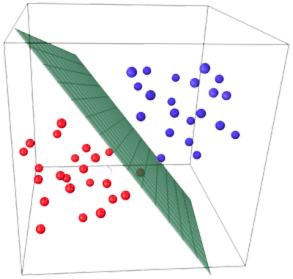
➔ binary system: 0 or 1 output

From linear regression to artificial neuron

Linear regression

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \dots w_n x_n = f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^\intercal \boldsymbol{x} \qquad \text{(model)}$$

- input: $\boldsymbol{x}^\intercal = (x_0, x_1, x_2, \dots, x_n)$, where $x_0 = 1$
- weigths: $\boldsymbol{w}^\intercal = (w_0, w_1, w_2, \dots, w_n)$, where w_0 is *bias*

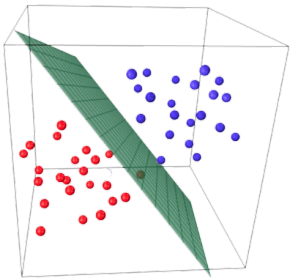


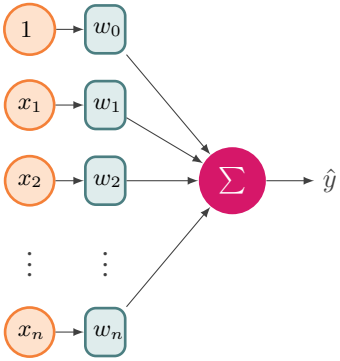
From linear regression to artificial neuron

Linear regression

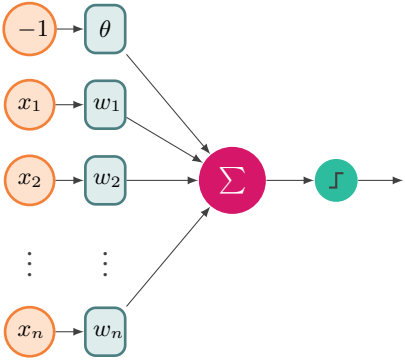
$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \dots w_n x_n = f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^\intercal \boldsymbol{x} \qquad \text{(model)}$$

- input: $\boldsymbol{x}^\intercal = (x_0, x_1, x_2, \dots, x_n)$, where $x_0 = 1$
- weigths: $\boldsymbol{w}^\intercal = (w_0, w_1, w_2, \dots, w_n)$, where w_0 is *bias*

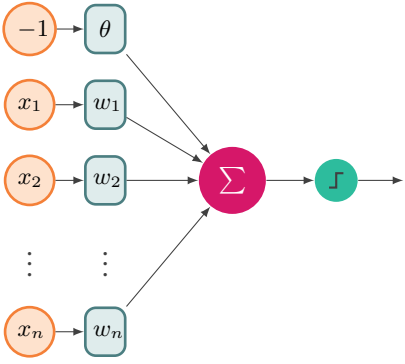




Artificial neuron – Threshold Logic Unit

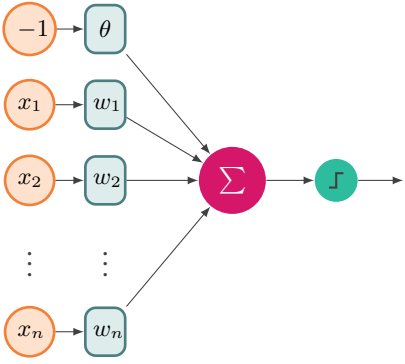


Artificial neuron – Threshold Logic Unit



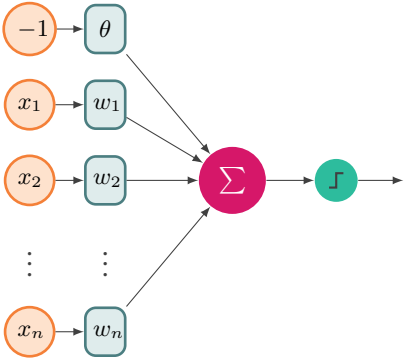
- input $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$

Artificial neuron – Threshold Logic Unit



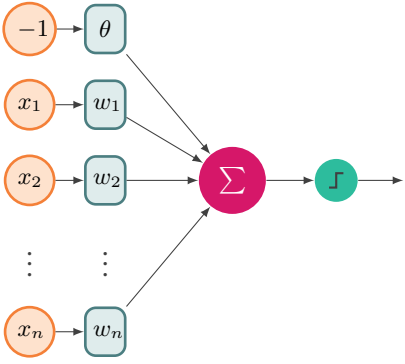
- input $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$
- weights $\boldsymbol{w} = (w_1, w_2, \ldots, w_n)$

Artificial neuron – Threshold Logic Unit



- input $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$
- weights $\boldsymbol{w} = (w_1, w_2, \ldots, w_n)$
- threshold θ

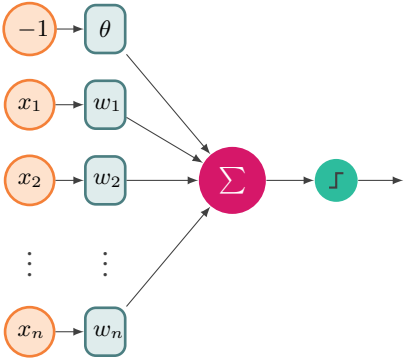
Artificial neuron – Threshold Logic Unit



- input $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$
- weights $\boldsymbol{w} = (w_1, w_2, \ldots, w_n)$
- threshold θ
- calculate internal potential

$$\xi = \sum_{i=1}^{n} w_i x_i - \theta$$

Artificial neuron – Threshold Logic Unit



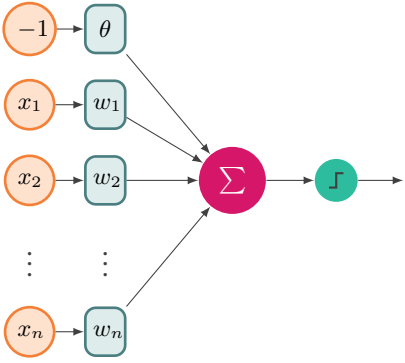
- input $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$
- weights $\boldsymbol{w} = (w_1, w_2, \ldots, w_n)$
- threshold θ
- calculate internal potential

$$\xi = \sum_{i=1}^{n} w_i x_i - \theta$$

- apply activation function

$$f(\xi) \to \Theta(\xi) = \begin{cases} 1 & \text{if } \xi > 0 \\ 0 & \text{otherwise} \end{cases}$$

Artificial neuron – Threshold Logic Unit



- input $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$
- weights $\boldsymbol{w} = (w_1, w_2, \ldots, w_n)$
- threshold θ
- calculate internal potential

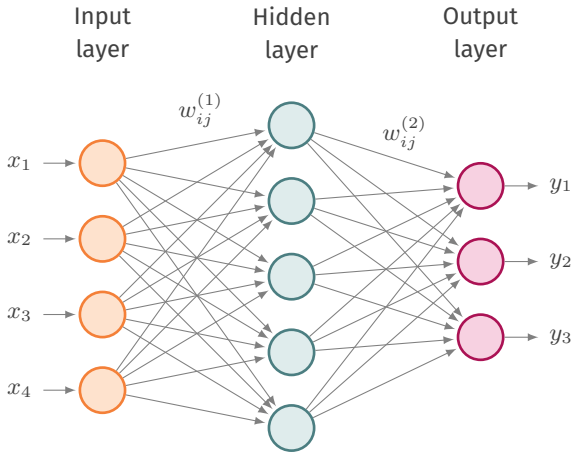
$$\xi = \sum_{i=1}^{n} w_i x_i - \theta$$

- apply activation function

$$f(\xi) \to \Theta(\xi) = \begin{cases} 1 & \text{if } \xi > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Perceptron is a binary classifier

Artificial neural network



How to train a neural network?

Training neural networks

- for training a neural network we need training dataset
- set of inputs $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ with required outputs (targets) $\boldsymbol{y}$

Training neural networks

- for training a neural network we need training dataset
- set of inputs $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ with required outputs (targets) $\boldsymbol{y}$

Forward pass ⏭

- send an input sample $\boldsymbol{x}$ to your neural network
- you obtain a prediction prediction $\hat{\boldsymbol{y}}$
- calculate prediction error with respect to your target $\boldsymbol{y}$

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = (\boldsymbol{y} - \hat{\boldsymbol{y}})^2 \quad \text{(loss function)}$$

Training neural networks

- for training a neural network we need training dataset
- set of inputs $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ with required outputs (targets) $\boldsymbol{y}$

Forward pass ⏭

- send an input sample $\boldsymbol{x}$ to your neural network
- you obtain a prediction prediction $\hat{\boldsymbol{y}}$
- calculate prediction error with respect to your target $\boldsymbol{y}$

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = (\boldsymbol{y} - \hat{\boldsymbol{y}})^2 \qquad \text{(loss function)}$$

Reverse pass ⏮

- reduce the error $\mathcal{L}(\boldsymbol{w})$
- change the connection weights

$$w_{ij}^{(L)} = w_{ij}^{(L)} - \eta \frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial w_{ij}^{(L)}}$$

  where η is the learning rate
- gradient descent
- backpropagation algorithm

Important parameters

- activation function
- loss function
- optimizer
- size of the neural network

Activation function

- logistic function

$$\sigma(\xi) = \frac{1}{1+\exp(-\xi)} \quad \in (0,1) \qquad \text{(logistic)}$$

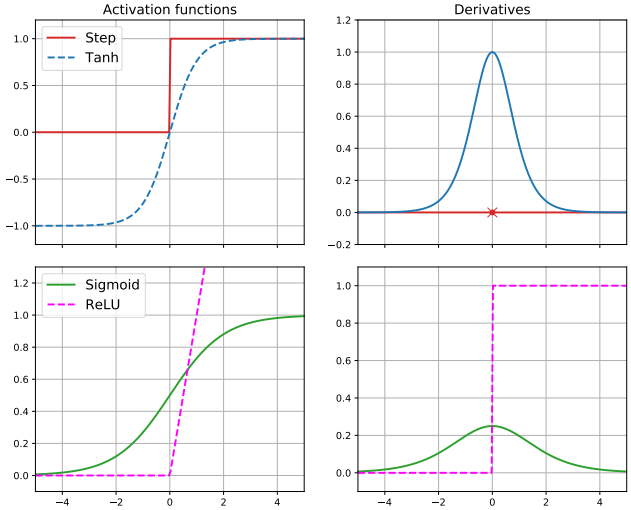
- hyperbolic tangent

$$\tanh(\xi) = 2\,\sigma(2\xi) - 1 \quad \in (-1,1) \qquad \text{(tanh)}$$

- ReLU (rectified linear unit)

$$\mathrm{ReLU}(\xi) = \max(0,\xi) \quad \in (0,\infty) \qquad \text{(ReLU)}$$

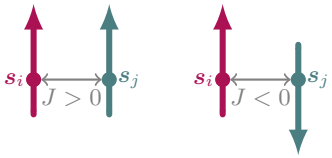
Activation function



Example: phase classification

Exchange interaction

$$\mathcal{H}_{\text{exch}} = -J \sum_{\langle i,j \rangle} \boldsymbol{s}_i \cdot \boldsymbol{s}_j$$



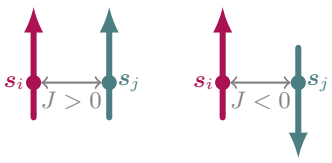
- supports collinear configurations

↪ $J > 0$ ferromagnetic (parallel)

↪ $J < 0$ antiferromagnetic (antiparallel)

Exchange interaction

$$\mathcal{H}_{\text{exch}} = -J \sum_{\langle i,j \rangle} \boldsymbol{s}_i \cdot \boldsymbol{s}_j$$



- supports collinear configurations

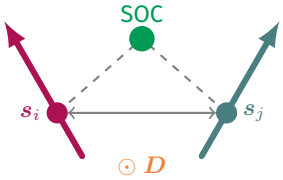
↪ $J > 0$ ferromagnetic (parallel)

↪ $J < 0$ antiferromagnetic (antiparallel)

Dzyaloshinskii-Moriya interaction

$$\mathcal{H}_{\text{DMI}} = -\boldsymbol{D} \cdot \sum_{\langle i,j \rangle} (\boldsymbol{s}_i \times \boldsymbol{s}_j)$$

- $\boldsymbol{D}$ is the Dzyaloshinskii-Moriya vector
- importance of spin-orbital coupling (SOC)



- supports spin canting

Phases in noncentrosymmetric magnets

Magnetic Helix

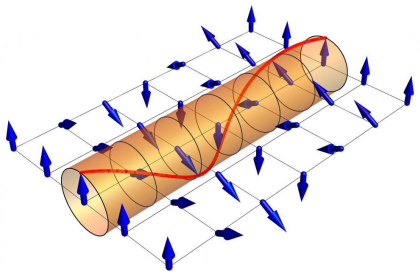


figure: london-nano.com

Phases in noncentrosymmetric magnets

Magnetic Helix

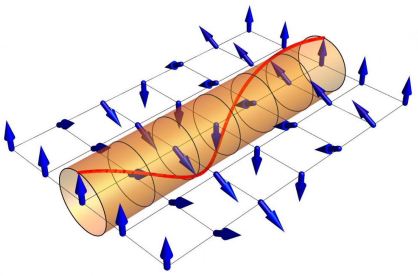


figure: london-nano.com

Skyrmion lattice

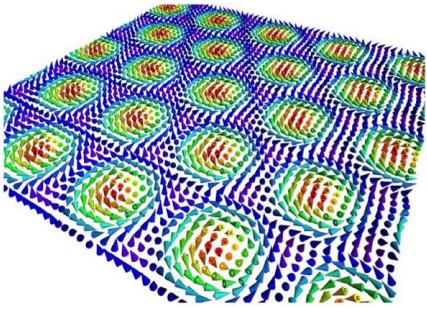


figure: phys.org

Example: phase classification

I. A. Iakovlev, O. M. Sotnikov, and V. V. Mazurenko
Supervised learning approach for recognizing magnetic skyrmion phases
Phys. Rev. B 98, 174411 (2018)

Example: phase classification

I. A. Iakovlev, O. M. Sotnikov, and V. V. Mazurenko
Supervised learning approach for recognizing magnetic skyrmion phases
Phys. Rev. B 98, 174411 (2018)

- model of 2D magnetic lattice of 48×48 magnetic moments

$$\mathcal{H} = -\sum_{\langle i,j \rangle} \boldsymbol{s}_i \cdot \boldsymbol{s}_j - \boldsymbol{D} \cdot \sum_{\langle i,j \rangle} \boldsymbol{s}_i \times \boldsymbol{s}_j - B \sum_i s_i^{(z)}$$

Example: phase classification

I. A. Iakovlev, O. M. Sotnikov, and V. V. Mazurenko
Supervised learning approach for recognizing magnetic skyrmion phases
Phys. Rev. B 98, 174411 (2018)

- model of 2D magnetic lattice of 48×48 magnetic moments

$$\mathcal{H} = -\sum_{\langle i,j \rangle} \boldsymbol{s}_i \cdot \boldsymbol{s}_j - \boldsymbol{D} \cdot \sum_{\langle i,j \rangle} \boldsymbol{s}_i \times \boldsymbol{s}_j - B \sum_i s_i^{(z)}$$

- they simulated magnetic configurations for various values of D and B

Example: phase classification

I. A. Iakovlev, O. M. Sotnikov, and V. V. Mazurenko
Supervised learning approach for recognizing magnetic skyrmion phases
Phys. Rev. B 98, 174411 (2018)

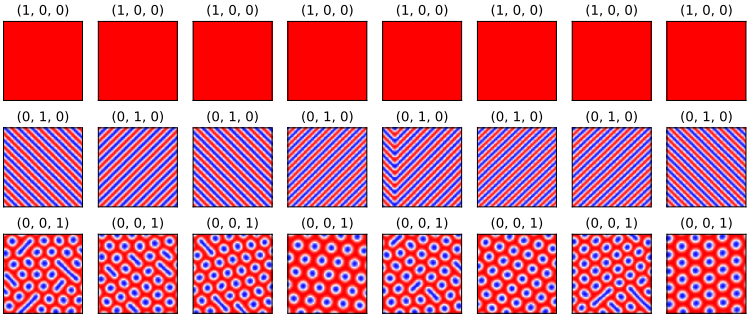
- model of 2D magnetic lattice of 48×48 magnetic moments

$$\mathcal{H} = -\sum_{\langle i,j \rangle} \boldsymbol{s}_i \cdot \boldsymbol{s}_j - \boldsymbol{D} \cdot \sum_{\langle i,j \rangle} \boldsymbol{s}_i \times \boldsymbol{s}_j - B \sum_i s_i^{(z)}$$

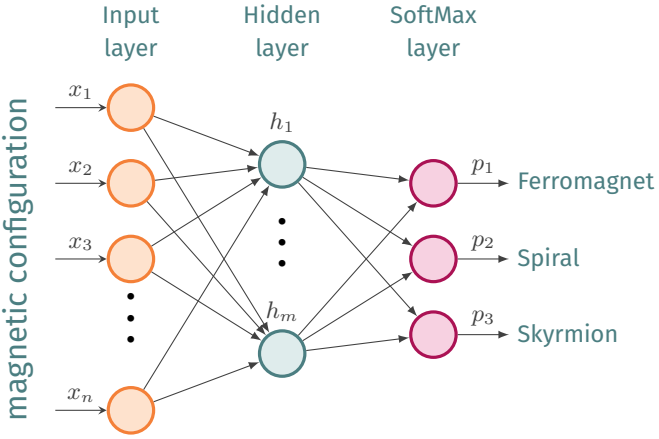
- they simulated magnetic configurations for various values of D and B
- they used z-component of magnetization as an input of the neural network

Example: phase classification

- typical training samples



Example: phase classification



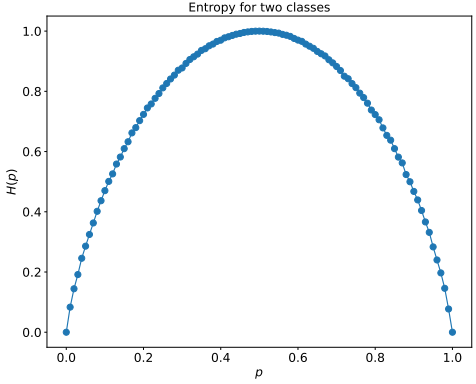
$$p_1 + p_2 + p_3 = 1$$

(probability density)

How to compare two probability densities?

Entropy

$$H(\boldsymbol{p}) = -\sum_{j=1}^{P} p_j \log(p_j)$$



Loss function: cross–entropy

- output: $\boldsymbol{q} = (q_1, q_2, \ldots, q_P)$
- target: $\boldsymbol{p} = (p_1, p_2, \ldots, p_P)$

➜ to estimate the error, we need to compare two probability mass functions

- Loss function: cross–entropy

Cross–entropy

$$H(\boldsymbol{p}, \boldsymbol{q}) = -\sum_{j=1}^{P} p_j \log(q_j)$$

- if $\boldsymbol{p} = \boldsymbol{q}$ we obtain entropy

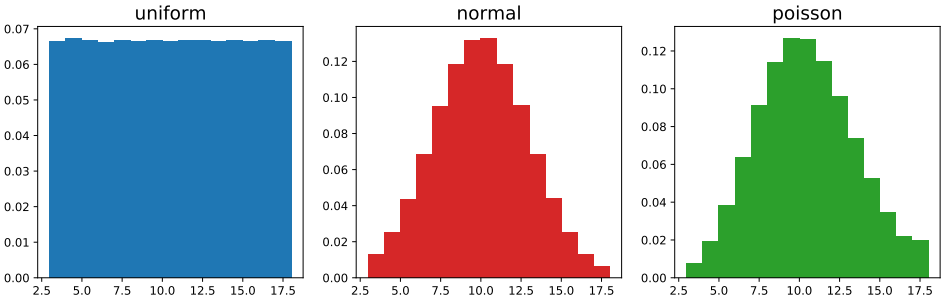
$$H(\boldsymbol{p}, \boldsymbol{p}) = H(\boldsymbol{p})$$
$$H(\boldsymbol{p}, \boldsymbol{q}) \neq H(\boldsymbol{q}, \boldsymbol{p})$$
$$H(\boldsymbol{p}, \boldsymbol{q}) = H(\boldsymbol{p}) + D_{\mathrm{KL}}(\boldsymbol{p} \parallel \boldsymbol{q})$$

Kullback-Leibler (KL) Divergence

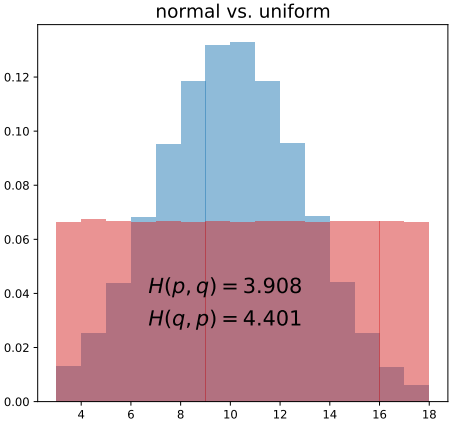
$$D_{\mathrm{KL}}(\boldsymbol{p} \parallel \boldsymbol{q}) = -\sum_{j=1}^{P} p_j \log\left(\frac{q_j}{p_j}\right)$$

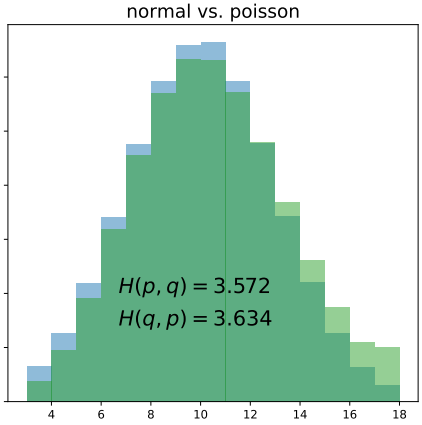
$$D(\boldsymbol{p} \parallel \boldsymbol{p}) = 0$$
$$D(\boldsymbol{p} \parallel \boldsymbol{q}) \neq D(\boldsymbol{q} \parallel \boldsymbol{p})$$

Cross-entropy

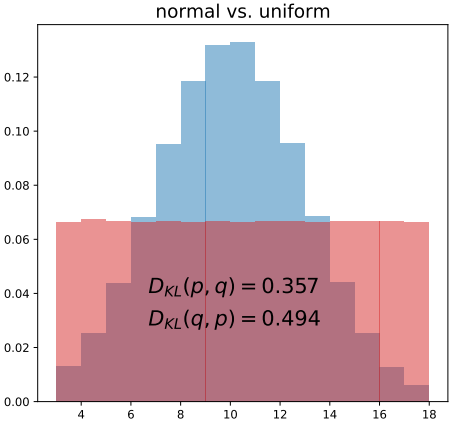


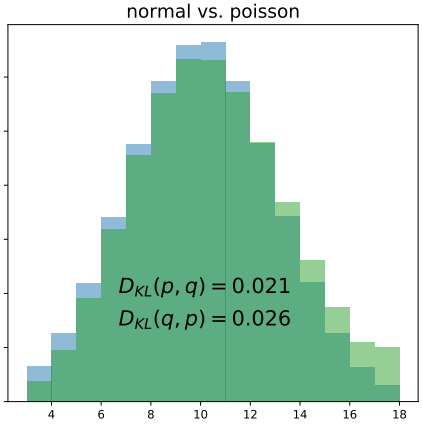
Cross-entropy





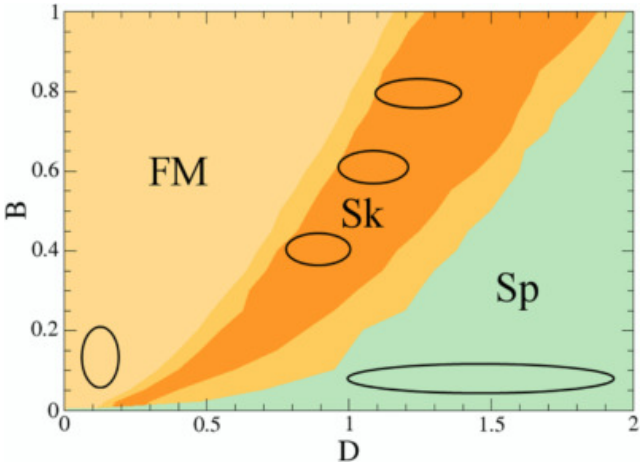
KL–divergence





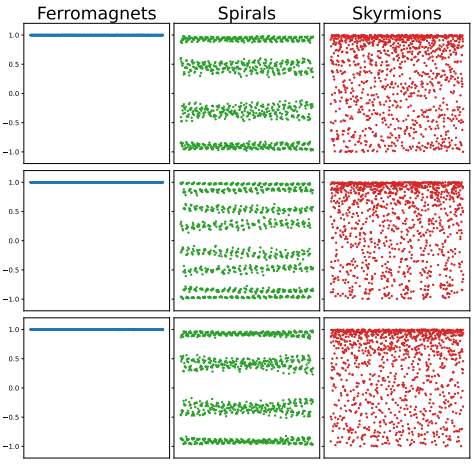
Example: phase classification

I. A. Iakovlev, O. M. Sotnikov, and V. V. Mazurenko, Phys. Rev. B 98, 174411 (2018)



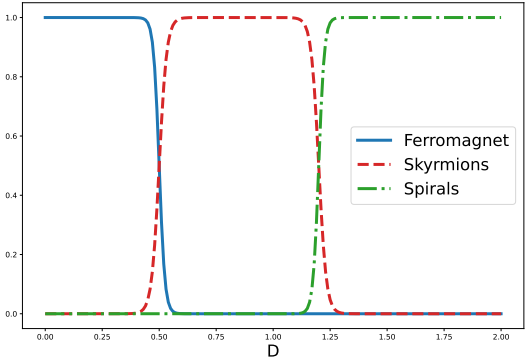
Example: phase classification

- typical input data as 1D vectors
  $48 \times 48 = 2304$ items
- neural network analysis the correlations between the input values



Ordering parameter

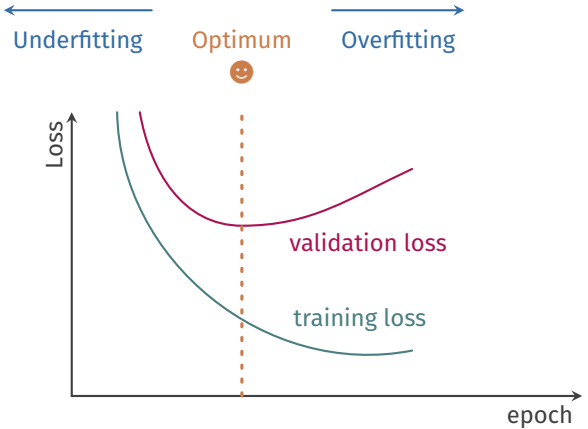
- neural networks are very good in interpolation
- we can plot the probabilities as a function of D for a constant value of magnetic field, B



- neural network can be used as a ordering parameter

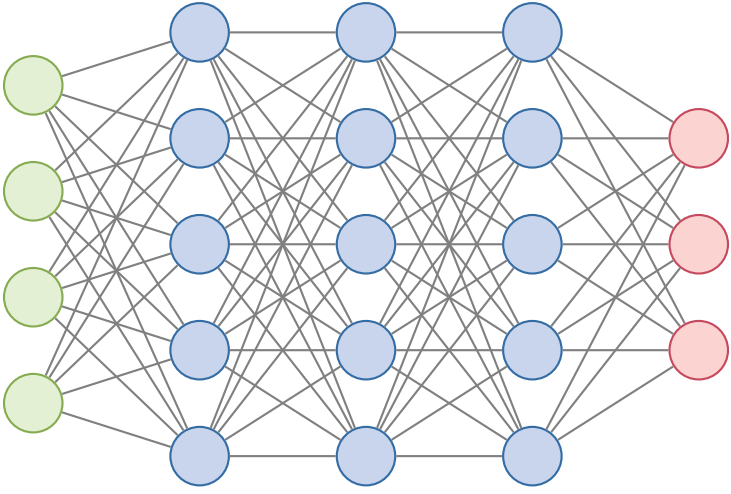
Training of the neural network

- neural network is train in several epochs
- usually mini-batch stochastic gradient descent is used
- for each epoch estimate training loss and validation loss

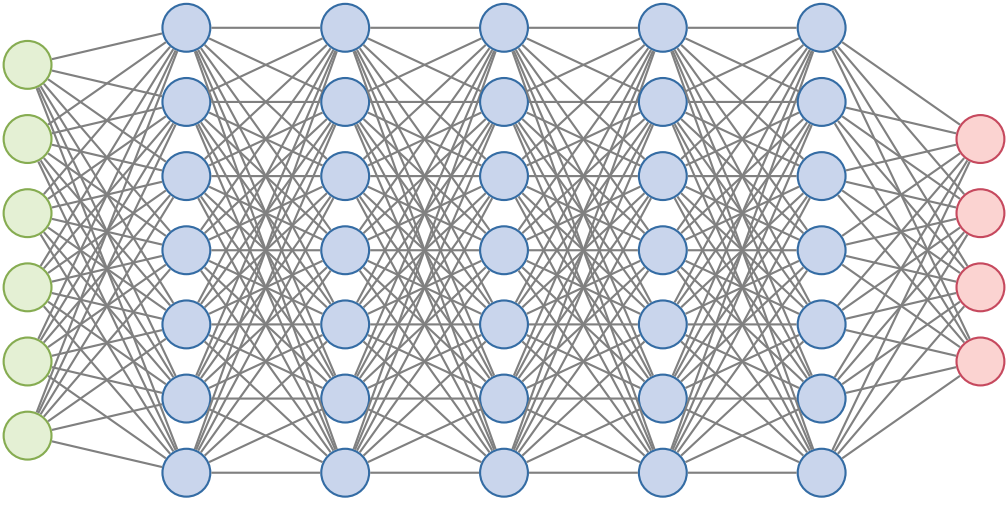


Deep Learning

Deep learning



Very deep learning



Problems with depth

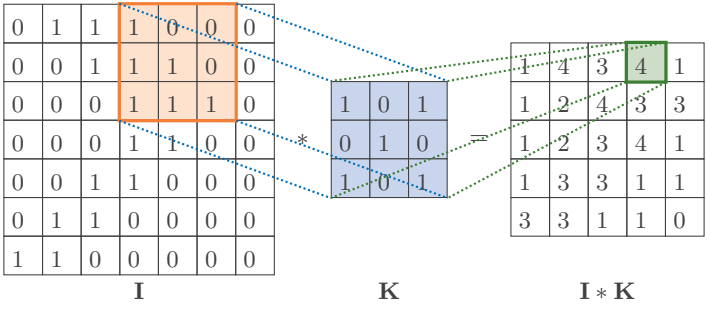
- too many parameters: serious risk of overfitting
- big training datasets required
- too long training time
- vanishing / exploding gradient

How to solve the problems?

- 👍 new types of approaches: convolution and pooling layers
- 👍 architectures
- 👍 regularization
- 👍 initialization

Convolutional Neural Networks

Convolutional layer

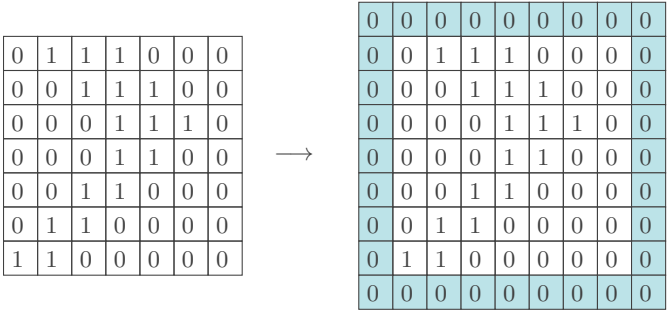


$$z_j = \sum_i w_i x_i$$

(convolution)

Anatomy of convolutions

- filter: *receptive field*, rectangle of width f_w, and height f_h
- a neuron located in row i and column j of a given layer is connected to neurons located in rows $i \to i + f_w - 1$ and columns $j \to j + f_h - 1$ of the previous layer
- next layer has smaller size than the previous one
- padding: in order for a layer have the same height and width as the previous one, one can add *zeros* around the input (*zero-padding*)



Anatomy of convolutions

- stride: the distance between two consecutive receptive fields (s_w, s_h)

0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	1	1	0	0	0	0
1	1	0	0	0	0	0

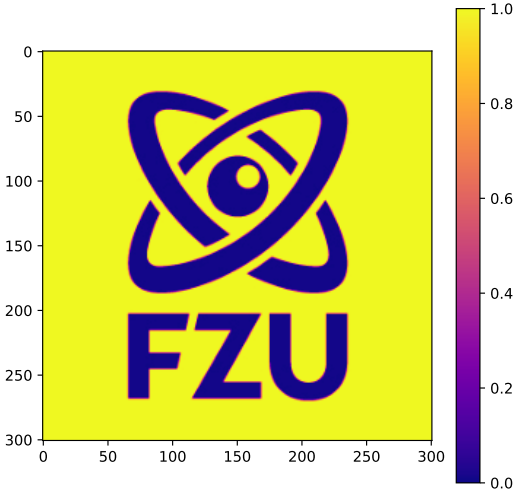
$s_w = 2$

0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	1	1	0	0	0	0
1	1	0	0	0	0	0

$s_w = 4$

- as a result the next layer will be smaller

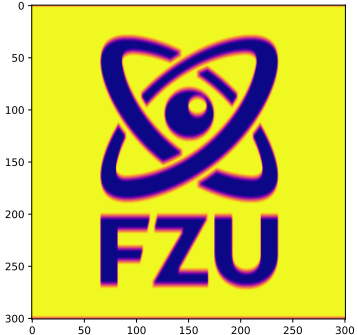
Example of convolution filters



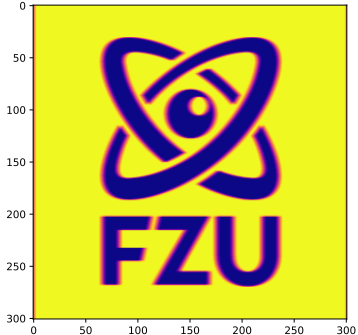
Vertical and Horizontal Filters

$$F_{\text{vert}} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix} \qquad F_{\text{hor}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

vertical filter



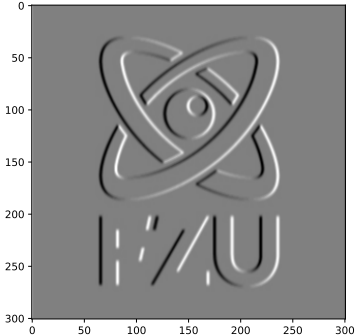
horizontal filter



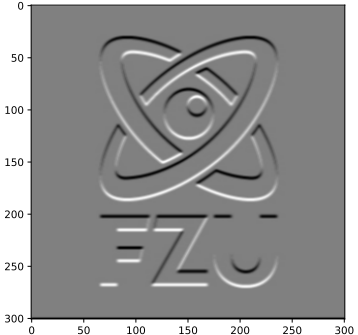
Sobel Filters: enhancement of the edges

$$S_{\text{vert}} = \begin{pmatrix} -0.25 & -0.2 & 0 & 0.2 & 0.25 \\ -0.4 & -0.5 & 0 & 0.4 & 0.5 \\ -0.5 & -1 & 0 & 1 & 0.5 \\ -0.4 & -0.5 & 0 & 0.4 & 0.5 \\ -0.25 & -0.2 & 0 & 0.2 & 0.25 \end{pmatrix} \qquad S_{\text{hor}} = \begin{pmatrix} -0.25 & -0.4 & -0.5 & -0.4 & -0.25 \\ -0.2 & -0.5 & -1 & -0.5 & -0.2 \\ 0 & 0 & 0 & 0 & 0 \\ 0.2 & 0.5 & 1 & 0.5 & 0.2 \\ 0.25 & 0.4 & 0.5 & 0.4 & 0.25 \end{pmatrix}$$

vertical Sobel filter



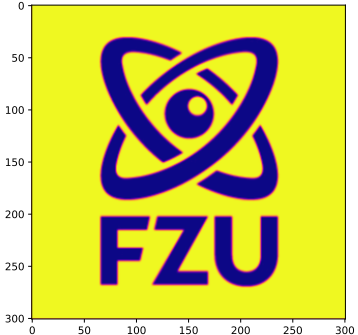
horizontal Sobel filter



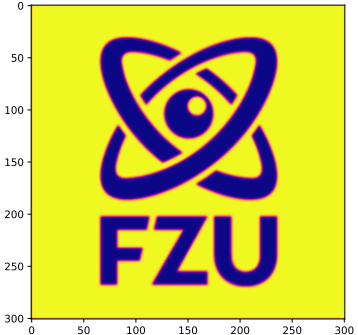
Gaussian Blur Filters

$$G_3 = \frac{1}{16}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 2 & 4 & 2 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad G_5 = \frac{1}{256}\begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix}$$

3x3 Gaussian blur filter



5x5 Gaussian blur filter



Stacking multiple feature maps

- in CNN each layer consists of more feature maps of equal size
- different feature maps may have different parameters
- a neuron in row i, and column j of the feature map k in a given convolutional layer l is connected to the outputs of neurons in the previous layer $l-1$, located in rows from $i\,s_h$ to $i\,s_h + f_h - 1$ and columns from $j\,s_w$ to $j\,s_w + f_w - 1$, across all the feature maps (in layer $l-1$)

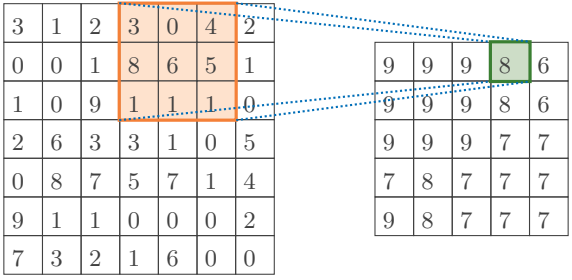
- the outputs of given neuron in a convolutional layer

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k}$$

  where $i' = i\,s_h + u$, $j' = j\,s_w + v$, and b_k is the bias term for feature map k

- $w_{u,v,k',k}$ connection weight between any neuron of in feature map k (layer l) and and its input in row u, column v, and feature map k' (layer $l-1$)
- $f_{n'}$ number of feature maps in layer $l-1$

Pooling



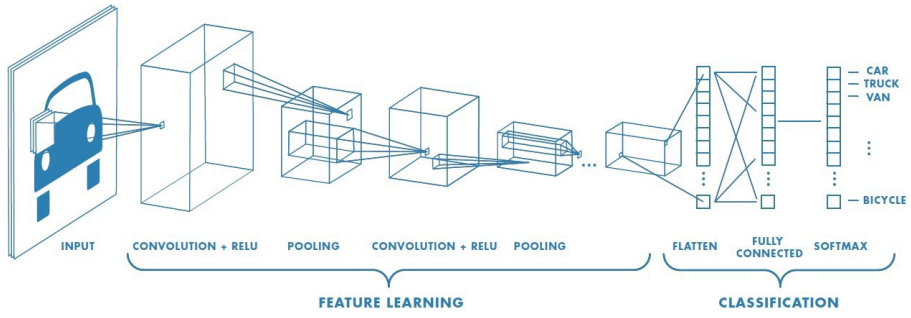
- max pooling

$$z_j = \max_i x_i$$

- average pooling

$$z_j = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Structure of convolutional neural networks



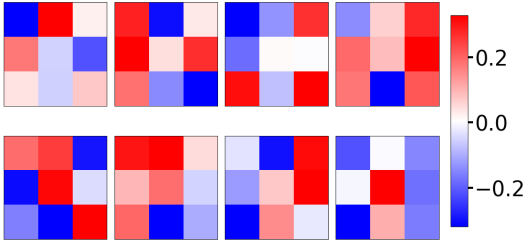
source: towardsdatascience.com

Features extraction: example with skyrmions

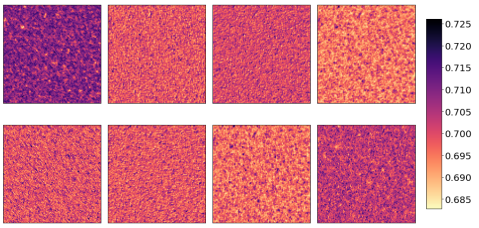
Ondřej Dušek: Analysis of magnetic skyrmions using machine learning methods (bachelor thesis, 2020)

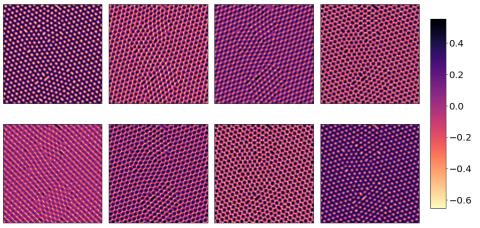
- Convolutional neural network with 8 filters of size $(3, 3)$
- MaxPooling of size $(2, 2)$

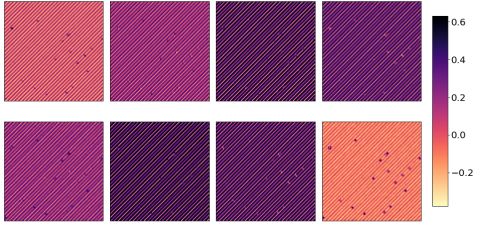
- trained 3×3 kernels



Features extraction: example with skyrmions

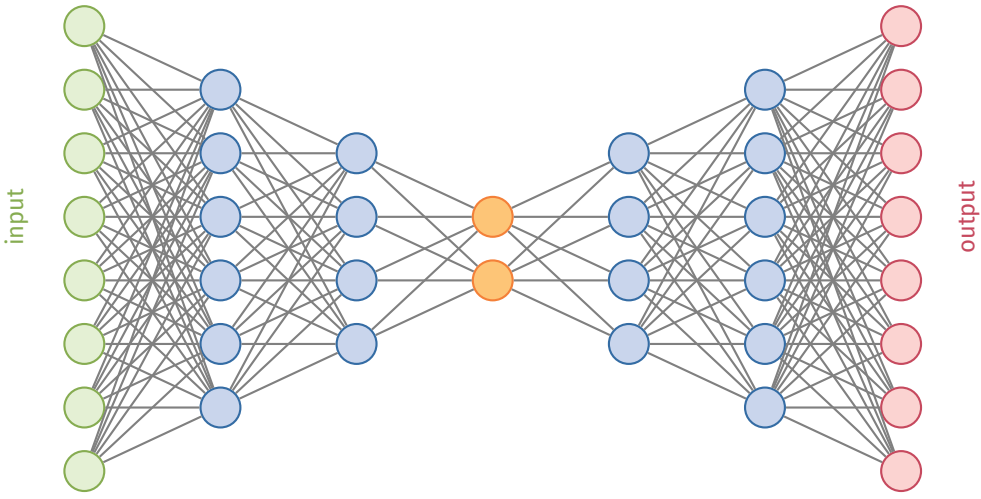




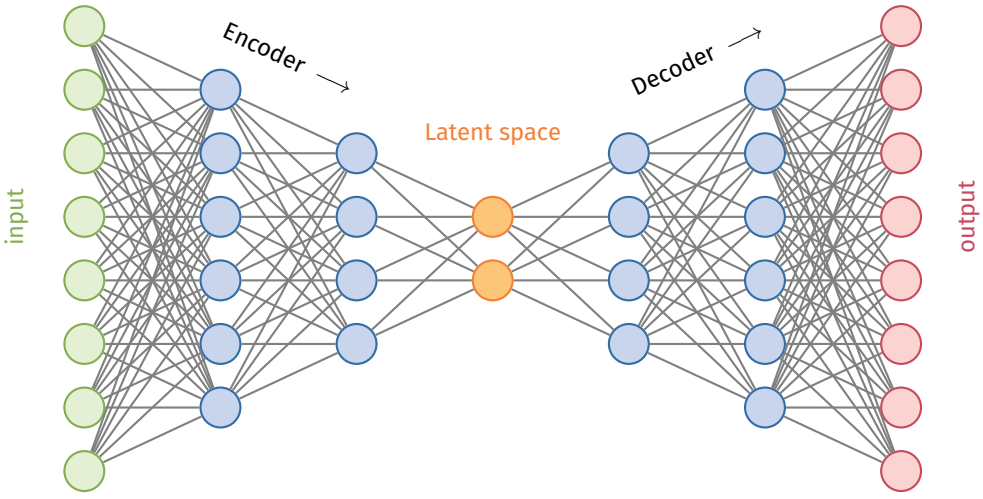


Autoencoder

Autoencoder



Autoencoder



Autoencoder

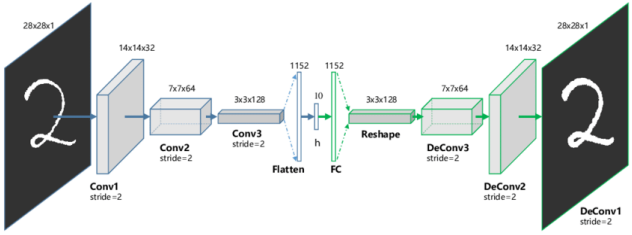
- coding: neural network capable of learning effective representation of the inputs
- unsupervised: autoencoder does not require any supervision (data are unlabeled)
- the loss functions compares just the distance between input and output
- autoencoders work as feature detectors
- can be used for pretraining of deeper neural networks

Autoencoder

- coding: neural network capable of learning effective representation of the inputs
- unsupervised: autoencoder does not require any supervision (data are unlabeled)
- the loss functions compares just the distance between input and output
- autoencoders work as feature detectors
- can be used for pretraining of deeper neural networks

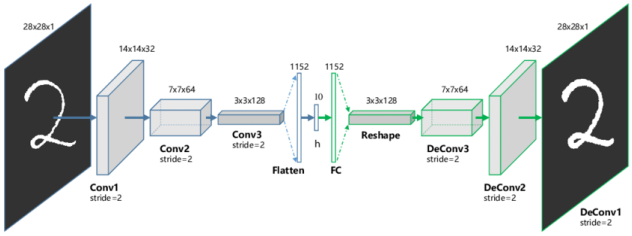
- dimensionality reduction
- noise reduction
- anomaly detection
- generation of new (unseen) data

Convolutional autoencoder



source: Guo, X., Liu, X., Zhu, E., Yin, J.: Deep Clustering with Convolutional Autoencoders (2017)

Convolutional autoencoder



source: Guo, X., Liu, X., Zhu, E., Yin, J.: Deep Clustering with Convolutional Autoencoders (2017)

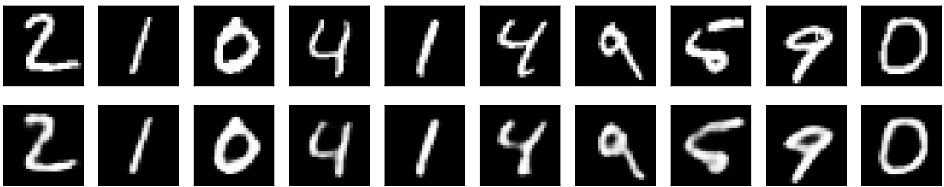
Upsampling

- `tf.keras.layers.UpSampling2D()`
  has to be used with convolutional layers
- `tf.keras.layers.Conv2DTranspose()`
  a.k.a. *deconvolution*

Dimensionality reduction

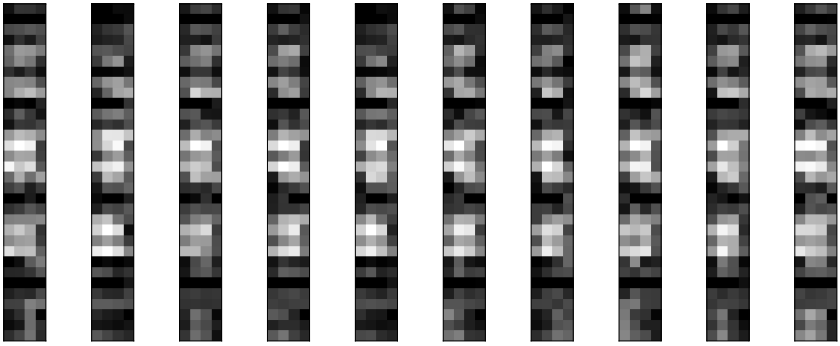
- latent space of autoencoder is smaller than its input $\Rightarrow$ dimensionality reduction

! if the autoencoder uses only linear activations and the cost function is the mean squared error it performs Principal Component Analysis (PCA)



outputs of convolutional autoencoder with 8 layers of size 4×4 in the latent space (128-dimensions)

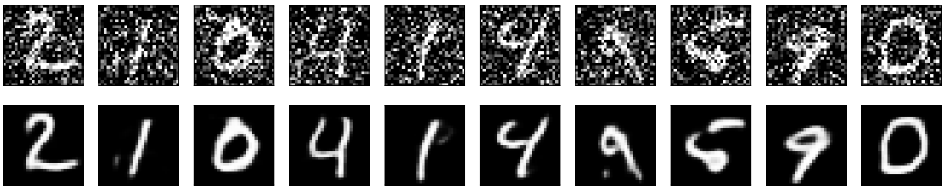
Latent space



8 layers of size 4×4

Denoising autoencoder

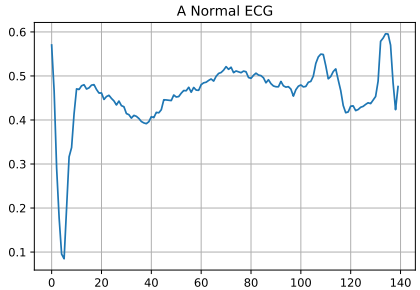
- training: add noise to the input
- inference: if input data contain noise, autoencoder will remove the noise in the output

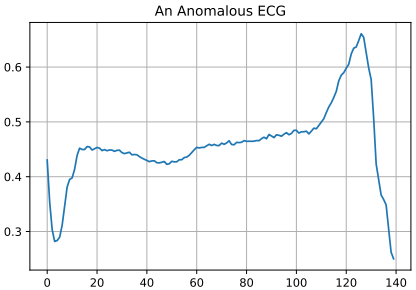


adding noise during the training might improve the optimization

Anomaly detection

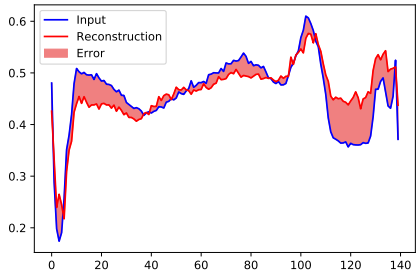
- autoencoder is train to encode certain kind of data
- during the inference the loss function is small, when the input data is of the same kind
- when the input departs from the training samples, the loss function increases

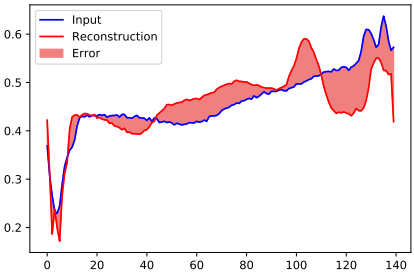




Anomaly detection

- mean squared error (MSE)





- set certain threshold Θ (hyperparameter)

if $\mathrm{MSE} \leq \Theta$ normal (negative)

if $\mathrm{MSE} > \Theta$ anomal (positive)

(classification)