# Anomalies Detection

## Advanced Machine Learning and Artificial Intelligence

Yuri Balasanov

University of Chicago, MScA

For any inquiries contact the author, Yuri Balasanov, at yuri.balasanov@iLykei.com .

# Outline of the Session

- Autoencoders and their applications
- General architecture of autoencoders
- Stacked autoencoders
- Sequential training of stacked autoencoders
- Unsupervized pretraining with stacked autoencoders
- Denoising autoencoders
- Sparse autoencoders
    - Kullback-Leibler Divergence
    - Application of Kullback-Leibler Divergence in sparse autoencodrers
- Other types of autoencoders

**Main text:** Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Technologies to Build Intelligent Systems, Aurelien Geron, 2017

# Autoencoders and Their Applications

- Autoencoders are artificial neural networks capable of unsupervised learning of representation of inputs (**codings**)
- Codings typically have lower dimensionality than the input data. This makes them a popular method for reduction of dimensionality
- Autoencoders are also commonly used as feature detectors and feature creators
- Autoencoders can help pretraining lower levels of deep neural networks, especially when data have limited labels
- Autoencoders can be used as **generative models**, i.e. they can generate new data similar to the training data
- Finally, autoencoders have become a popular method for anomalies detection

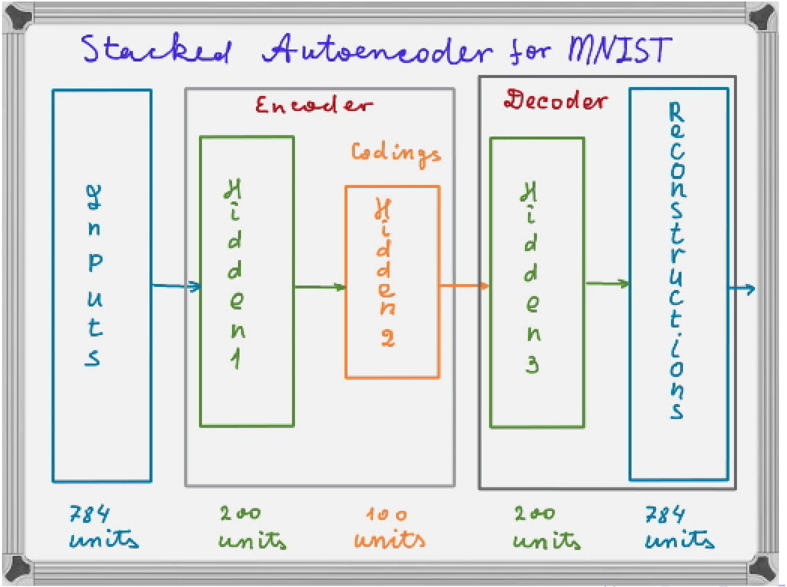
# General Architecture of Autoencoders

- Autoencoder contains 2 parts:
  - Encoder (**recognition network**), converts inputs to an internal representation
  - Decoder (**generative network**), converts internal representation to the outputs
- Since autoencoder trains making outputs equal to inputs, numbers of inputs and outputs must be equal
- Outputs of autoencoder are also called **reconstructions** and the loss function is **reconstruction loss**
- Autoencoder is typically **undercomplete**, i.e. internal representation has lower dimensionality than inputs. This prevents trivial solutions when inputs are simply copied to outputs
- Autoencoder can be undercomplete in different ways:
  - Layer of internal representations is smaller than number of inputs (**bottleneck**)
  - Internal representation layer has many units, but has smaller number of connections (**sparse autoencoder**)

# Stacked Autoencoders

- Autoencoder that has both encoder and decoder containing multiple layers of neurons is called **stacked autoencoder** or **deep autoencoder**
- Typically stacked autoencoders have equal layer sizes for layers symmetrically located relative to the bottleneck
- In addition weights of decoding layer often equal to transposed weights of encoding layer symmetrical to it. This is called **tying weights**. Tying weights reduces number of weights in the network and speeds up training
- Motivation behind additional layers is the same as for deep neural networks: higher complexity the subject for learning with fewer parameters
- However, adding too many layers may result in trivial solutions or loss of efficiency due to lack of undercompleteness

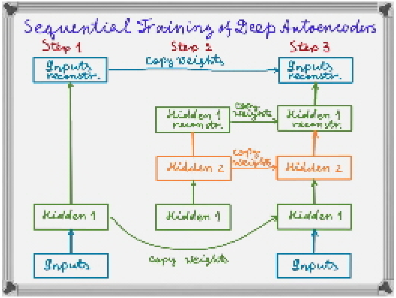
# Stacked Autoencoder Example



# Sequential Training of Stacked Autoencoders

One additional convenience of training stacked autoencoders is that they can be trained sequentially:

1. Train first layer of deep autoencoder as a shallow one, reconstructing the inputs
2. Train second shallow autoencoder, reconstructing outputs of the first one
3. Stack layers of the deep autoencoder in the right order

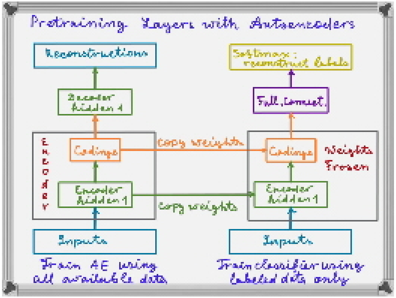


# Unsupervised Pretraining with Autoencoders

If there is not enough labeled data use pretrained lower layers by autoencoder:

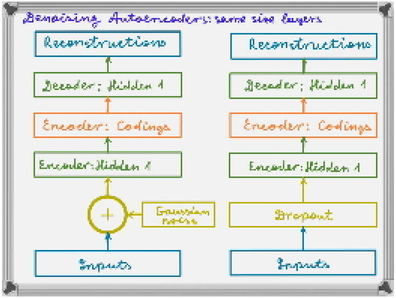
1. Use all train data to train an unsupervised autoencoder
2. Use the encoder layers as low level layers
3. Train higher layers using labeled data

See example for a classification model



# Denoising Autoencoders

- Autoencoders can work even if they are not undercomplete
- In order to enforce learning features, but not allowing trivial solution if decoder size is not smaller than size of inputs:
  1. Add noise to inputs, or
  2. Pass inputs through a dropout layer



# Sparse Autoencoders

- Sparsity is another way of putting constraints pushing autoencoders to learn features
- Sparse autoencoder is motivated to reduce the number of active neurons in the coding layer by a type of regularizator added to the cost function
- Regularizator may reduce the number of active neurons in encoder to, say, 5% on average. As a result each input is reproduced as a combination of a limited number of activations
- Let $a_i^{(l)}(x)$ be activation (output) of neuron $i$ in layer $l$ for input $x$. Sparsity at each training iteration can be measured as average activation in the encoding layer over a batch, assuming the batch size is not too small: $q_i = \frac{1}{m}\sum_{j=1}^{m} a_i^{(l)}(x_j)$, where $x_1, \ldots, x_m$ are inputs of batch of size $m$
- Given the mean activation per neuron regularizator punishes neurons that are too active by adding sparsity loss to the cost function

# Kullback-Leibler Divergence I

- A common sparsity loss function is **Kullback-Leibler divergence** (**KL-divergence**). It has much stronger gradient than mean squared error

## Definition

Let $P$ and $Q$ be two binomial random variables with probabilities of success $p$ and $q$ correspondingly. Then KL-divergence between them is

$$D_{KL}(P \| Q) = p \ln \frac{p}{q} + (1-p) \ln \frac{1-p}{1-q}$$

- This divergence measure has property that when $p = q$ then $D_{KL}(P \| Q) = 0$ and $D_{KL}(P \| Q)$ monotonically increases very quickly as $p$ and $q$ become more different from each other

Kullback-Leibler Divergence:
Binomial Distributions.

Definition: $\boxed{D_{KL}(P\|Q)=E_p\left[\ln\frac{P}{Q}\right]=\int p(x)\ln\frac{P(x)}{q(x)}dx}$

$$p(x)=\binom{n}{x}p^x(1-p)^{n-x} \qquad q(x)=\binom{n}{x}q^x(1-q)^{n-x}$$

$$\boxed{\begin{aligned}&\ln\frac{p^i(1-p)^{n-i}}{q^i(1-q)^{n-i}}\\ &=i\ln\frac{p}{q}\\ &+(n-i)\ln\frac{1-p}{1-q}\end{aligned}}$$

$$\boxed{D_{KL}(P\|Q)}=\sum_{i=0}^{n}\binom{n}{i}p^i(1-p)^{n-i}\ln\frac{p^i(1-p)^{n-i}}{q^i(1-q)^{n-i}}$$

$$=\sum_{i=0}^{n}\binom{n}{i}p^i(1-p)^{n-i}\left[i\ln\frac{p}{q}+(n-i)\ln\frac{1-p}{1-q}\right]$$

$$=\ln\frac{p}{q}\overbrace{\sum_{i=0}^{n}i\binom{n}{i}p^i(1-p)^{n-i}}^{=np}+\ln\frac{1-p}{1-q}\overbrace{\sum_{i=0}^{n}(n-i)\binom{n}{i}p^i(1-p)^{n-i}}^{=n(1-p)}$$

$$=\boxed{\ln\frac{p}{q}\,np+\ln\frac{1-p}{1-q}\,n(1-p)} \quad \text{Or for } n=1 \quad \boxed{\ln\frac{p}{q}p+\ln\frac{1-p}{1-q}(1-p)}$$

# Application of KL-Divergence in Sparse Autoencoders

- To train sparse autoencoder it is necessary to minimize loss equal to sum of KL-divergences between the target activation $p$ (usually a small number) and actual mean activations $q_i$ of neurons $i$ in the encoding layer

$$D_{KL}(P\|Q) = \sum_{i=1}^{k} \left( p \ln \frac{p}{q_i} + (1 - p_i) \ln \frac{1 - p_i}{1 - q_i} \right)$$

- For sparse autoencoders it is important to have activations $a_i^{(l)}(x)$ for all neurons strictly between 0 and 1. If activation equals either 0 or 1 exactly $D_{KL}(P\|Q)$ may return NaN
- A simple solution ensuring that activations are not equal 0 or 1 is to use logistic sigmoid as activation function for the encoding layer
- A good reference on sparse autoencoders is the following lecture notes article:

Sparse autoencoder, Andrew Ng, CS294A Lecture notes, https://web.stanford.edu/class/cs294a/sparseAutoencoder.pdf

# Other Types of Autoencoders

- Development of new variations of autoencoders have been going on adding to their success
- Some of the variations are below:
  - **Contractive autoencoder**: Constrains derivatives of the codings with respect to inputs
  - **Stacked convolutional autoencoder**: Extracts visual features by reconstructing images decoded by convolutional layers
  - **Generative stochastic network**: Denoising autoencoder with generative capability
  - **Winner-Take-All (WTA) autoencoder**: Only $k\%$ activations of each neuron over training batch are kept
  - **Generative Adversarial Network (GAN)**: Two networks: "generator" and "discriminator" - play against each other, one trying to present either objects from the real sample or generate fake data; the other trying to detect fakes. Makes a very high quality generative model