INTELLIGENT SENSOR PROCESSING USING MACHINE LEARNING (2)

Dr TIAN Jing

tianjing@nus.edu.sg

Module objective

Module: Intelligent sensor processing using machine learning

Knowledge and understanding

- Understand the fundamentals of intelligent sensor processing using machine learning and its applications

Key skills

- Design, build, implement intelligent sensor processing using machine learning for real-world applications

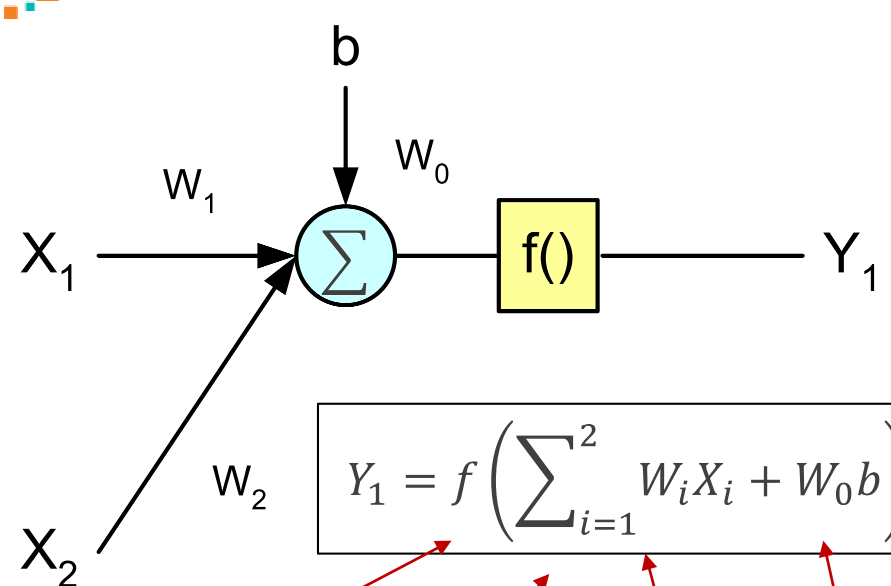
Major reference

- [Introduction] MIT 6.S191: *Introduction to Deep Learning*, http://introtodeeplearning.com/
- [Intermediate] *Machine Learning for Signal Processing*, UIUC, https://courses.engr.illinois.edu/cs598ps/fa2018/index.html
- [Intermediate] *Neural Networks for Signal Processing*, UFL, http://www.cnel.ufl.edu/courses/EEL6814/EEL6814.php
- [Comprehensive] M. Hoogendoorn, B. Funk, *Machine Learning for the Quantified Self: On the Art of Learning from Sensory Data*, Springer, 2018, https://ml4qs.org

Topics

- Signal representation using machine learning
- Applications of signal representation learning using machine learning
- Workshop

Recall: Neural network



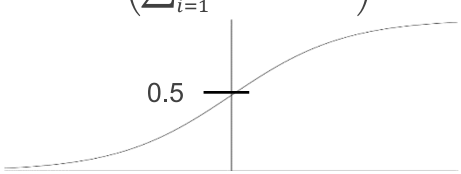
$$Y_1 = f\left(\sum_{i=1}^{2} W_i X_i + W_0 b\right)$$

Activate Sum up Weigh Bias

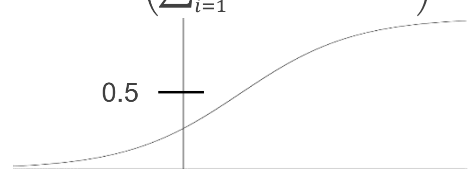
Notation	
X_1, X_2	Input
Y_1	Output
b	Bias ($b = 1$)
W_i	Weighting factor (for each arrow) for the $i-$th input data
$\sum()$	Summation function
$f()$	Activation function (e.g., sigmoid function)
A training data record: X_1, X_2, Y_1	

Note: A bias b is similar to intercept in regression model, such as $y = \alpha + \beta x$ with a slop β and intercept α. It can be considered as adjusting input before sending it to the subsequent activation function.
Reference: https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks

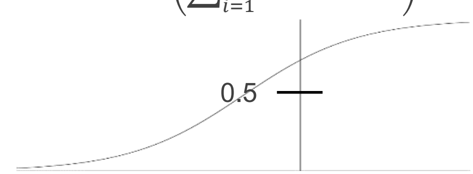
$$\text{sigmoid}\left(\sum_{i=1}^{2} W_i X_i + 0 \times b\right)$$



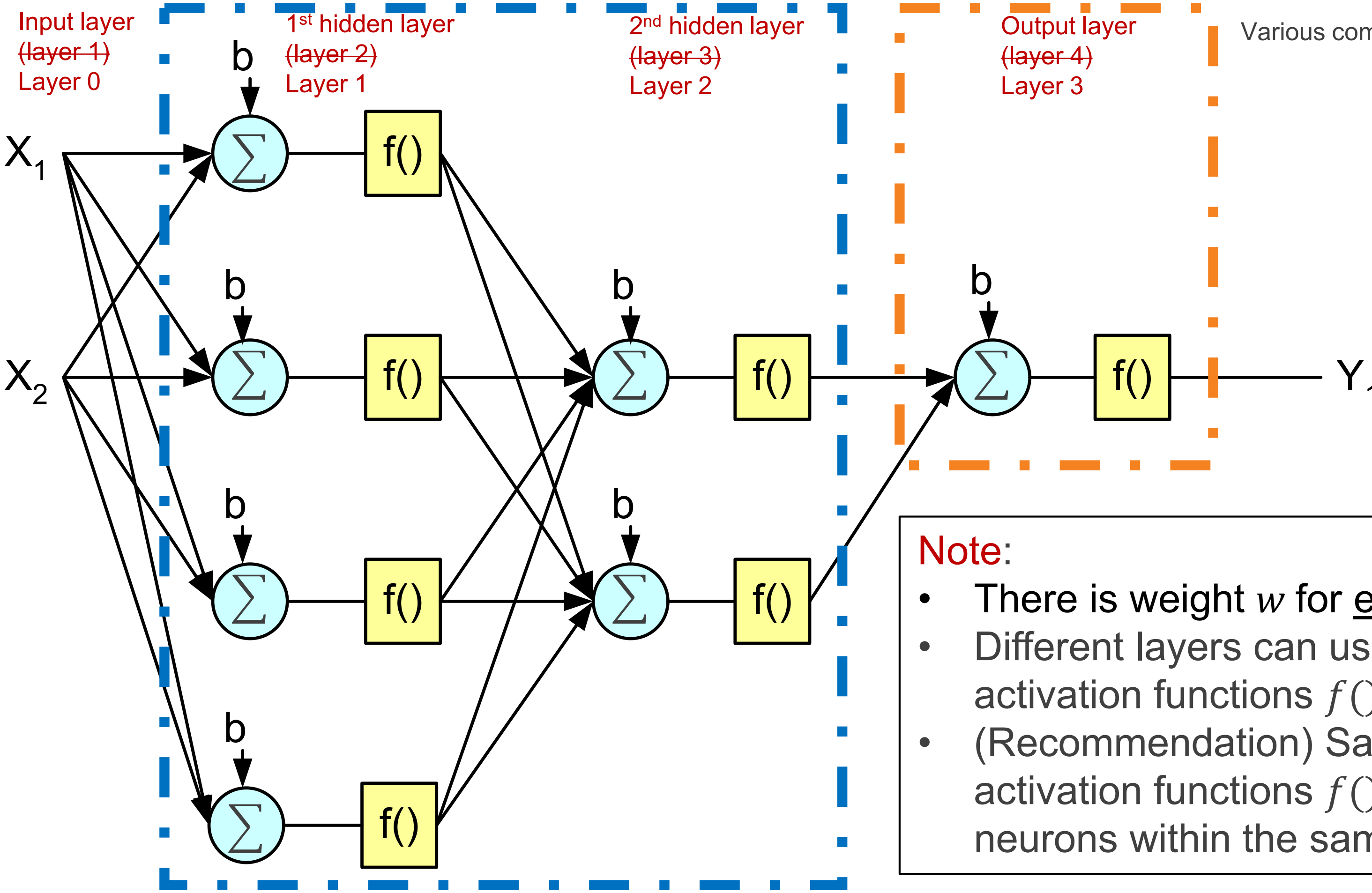
$$\text{sigmoid}\left(\sum_{i=1}^{2} W_i X_i + (-1) \times b\right)$$



$$\text{sigmoid}\left(\sum_{i=1}^{2} W_i X_i + 1 \times b\right)$$



Recall: Neural network



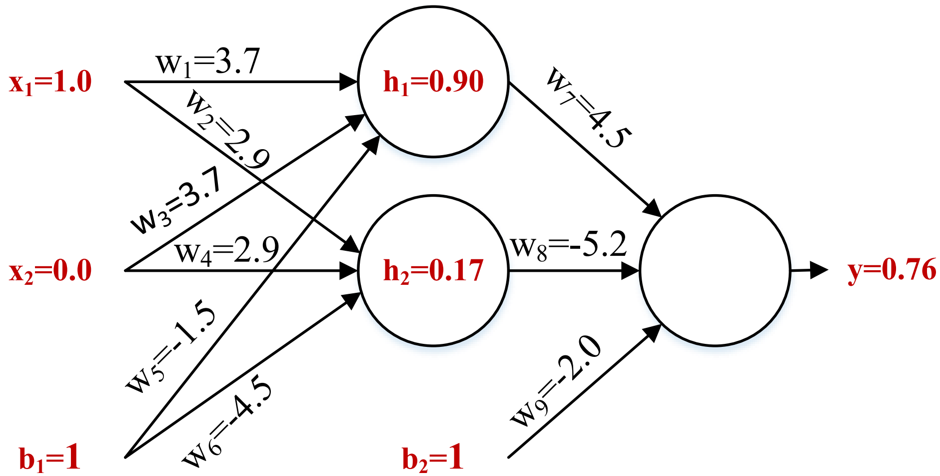
Various common counting/naming schemes

Note:

- There is weight w for <u>each arrow</u>.
- Different layers can use different types of activation functions $f()$.
- (Recommendation) Same type of activation functions $f()$ is used for all neurons within the same layer.

No. of Input	No. of Hidden		No. of Output
2	Layer	2	1
	No. of nodes at each hidden layer	4, 2	

Recall: Neural network



- Input data: $x_1 = 1.0, x_2 = 0.0$
- Output result: $y = 0.76$ (that will be interpreted as $y = 1$ in binary classification)
- Sigmoid function $f(x) = \frac{1}{1+e^{-x}}$ is used in nodes h_1, h_2 in hidden layer

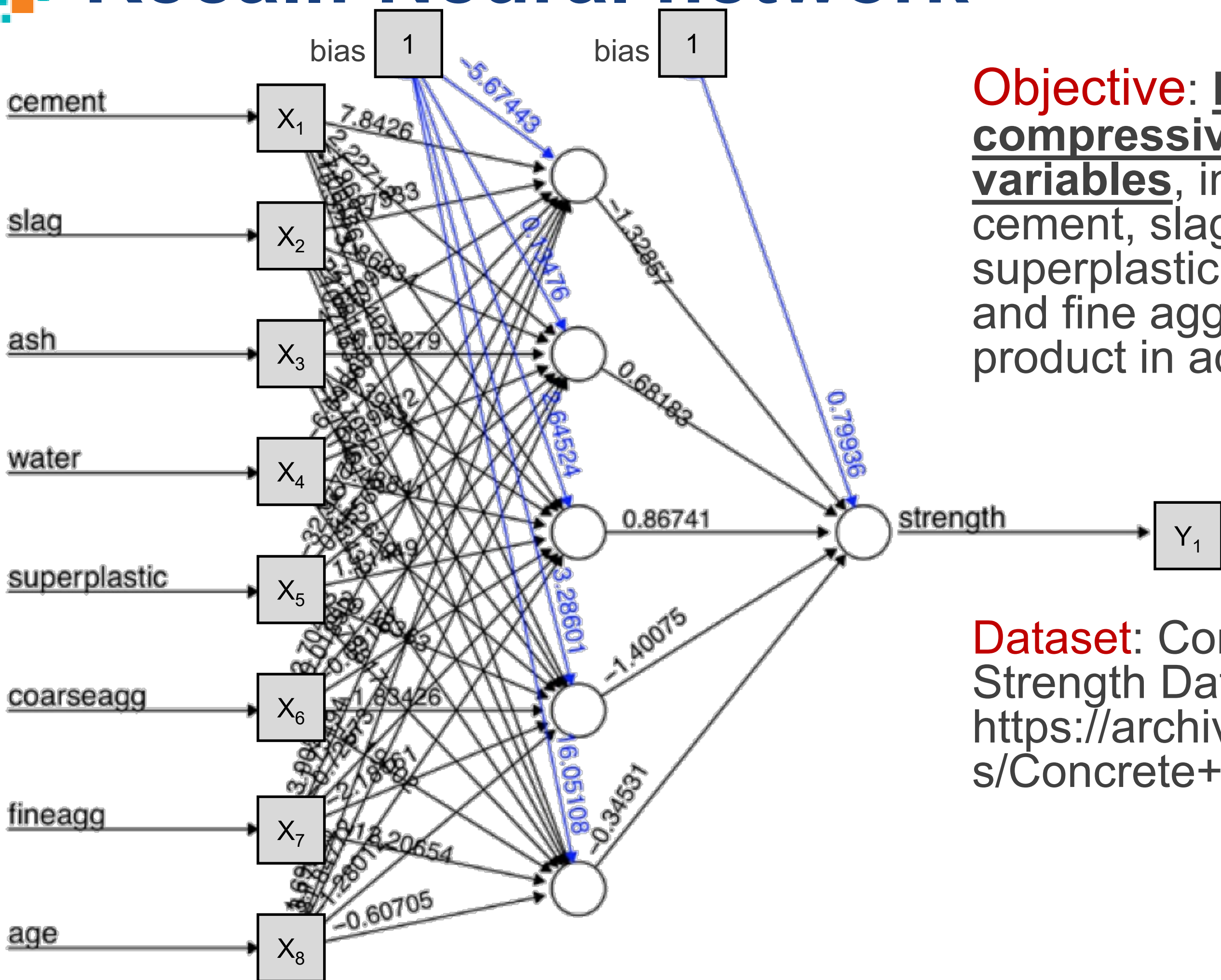
$$h_1 = \text{sigmoid}\big(1.0 \times 3.7 + 0.0 \times 3.7 + 1 \times (-1.5)\big) = \text{sigmoid}(2.2) = \frac{1}{1+e^{-2.2}} = 0.90$$

$$h_2 = \text{sigmoid}\big(1.0 \times 2.9 + 0.0 \times 2.9 + 1 \times (-4.5)\big) = \text{sigmoid}(-1.6) = \frac{1}{1+e^{1.6}} = 0.17$$

$$y = \text{sigmoid}\big(0.90 \times 4.5 + 0.17 \times (-5.2) + 1 \times (-2.0)\big) = \text{sigmoid}(1.17) = \frac{1}{1+e^{-1.17}} = 0.76$$

<table>
<tr><th>No. of Input</th><th colspan="2">No. of Hidden</th><th>No. of Output</th></tr>
<tr><td rowspan="2">2</td><td>Layer</td><td>1</td><td rowspan="2">1</td></tr>
<tr><td>No. of nodes at each hidden layer</td><td>2</td></tr>
</table>

Recall: Neural network



Objective: **Predict concrete compressive strength from input variables**, including the amount of cement, slag, ash, water, superplasticizer, coarse aggregate, and fine aggregate used in the product in addition to the aging time.

Dataset: Concrete Compressive Strength Data Set https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength

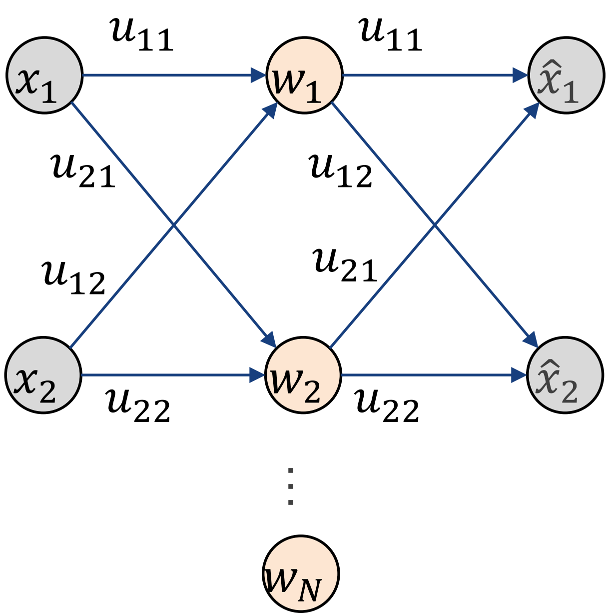
No. of Input	No. of Hidden		No. of Output
8	Layer	1	1
	No. of nodes at each hidden layer	5	

Signal representation learning

Recall: Given a two-dimensional signal $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$,

and N basis vectors $\mathbf{u}_1 = \begin{pmatrix} u_{11} \\ u_{12} \end{pmatrix}, \mathbf{u}_2 = \begin{pmatrix} u_{21} \\ u_{22} \end{pmatrix}, \cdots, \mathbf{u}_N = \begin{pmatrix} u_{N1} \\ u_{N2} \end{pmatrix}$.

- Decompose signal as $w_i = \langle \mathbf{x}, \mathbf{u}_i \rangle$, for example, $w_1 = \langle \mathbf{x}, \mathbf{u}_1 \rangle = x_1 u_{11} + x_2 u_{12}$, $w_2 = \langle \mathbf{x}, \mathbf{u}_2 \rangle = x_1 u_{21} + x_2 u_{22}$
- Reconstruct signal as $\hat{\mathbf{x}} = \sum_{i=1}^{N} w_i \times \mathbf{u}_i$, for example, $\hat{x}_1 = w_1 u_{11} + w_2 u_{21} + \cdots + w_N u_{N1}$, $\hat{x}_2 = w_1 u_{12} + w_2 u_{22} + \cdots + w_N u_{N2}$

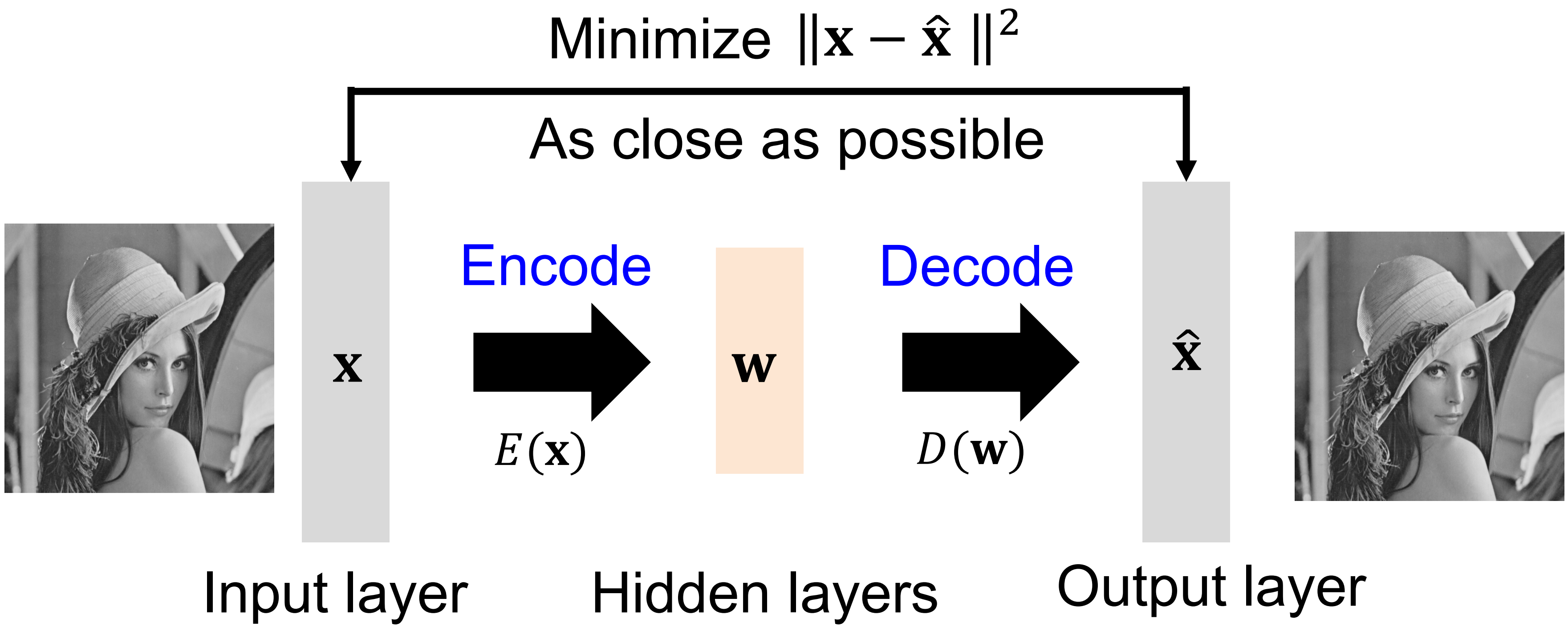


Signal representation as neural network	
Input	Signal
Output	Reconstructed signal
Model weights	Basis vectors
Hidden layer output	Signal representation coefficients

Idea: Train a model that is optimized (in terms of the optimal basis vectors) to output the signal as close as the input signal itself, that is called 'auto-encoder'.

Signal representation learning

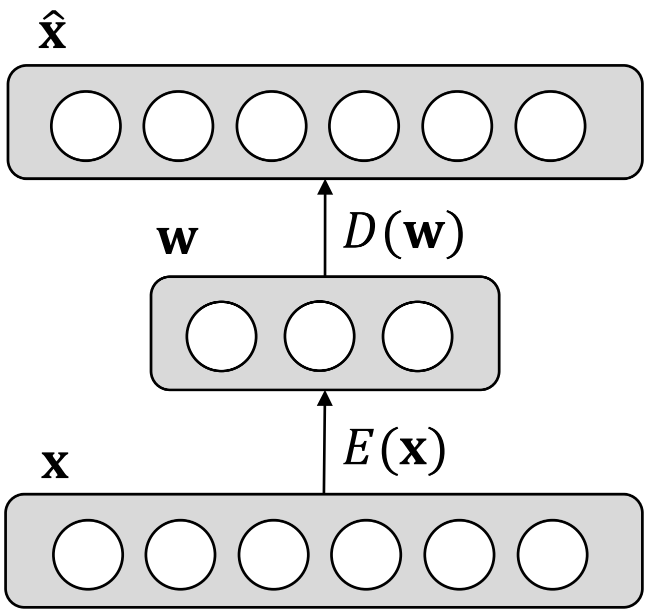
- Learn signal representation as neural network.

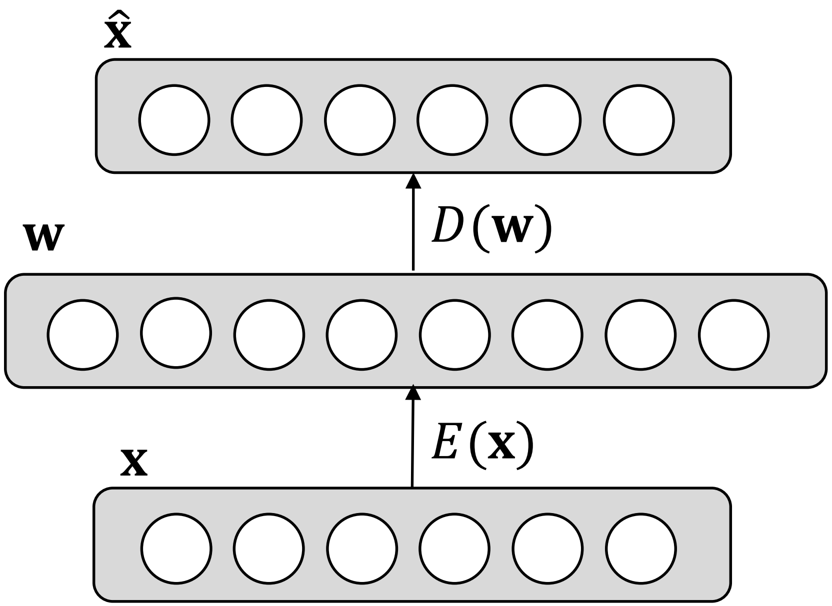


Signal representation: Under-complete and over-complete

Under-complete
- Hidden layer has less dimensions than the input layer
- Compresses the input
- Hidden nodes will be good features for the training distribution.

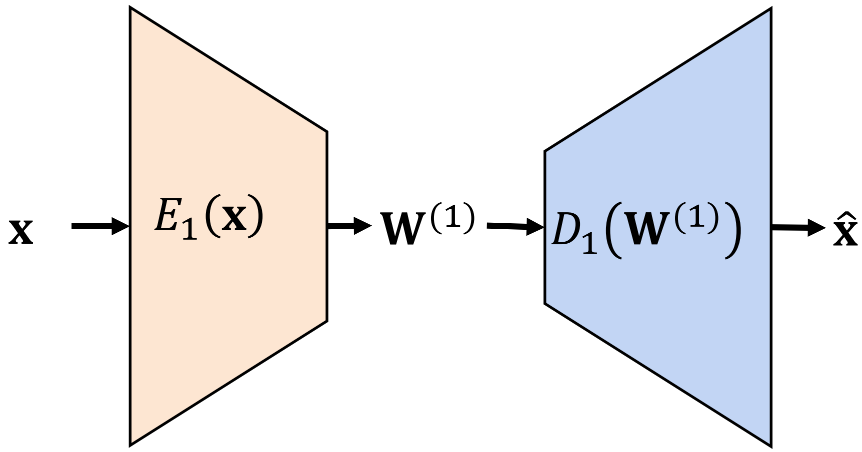
Over-complete
- Hidden layer has more dimensions than the input layer
- No compression in hidden layer.
- A higher dimension code helps model a more complex distribution.





Stacked signal representation learning

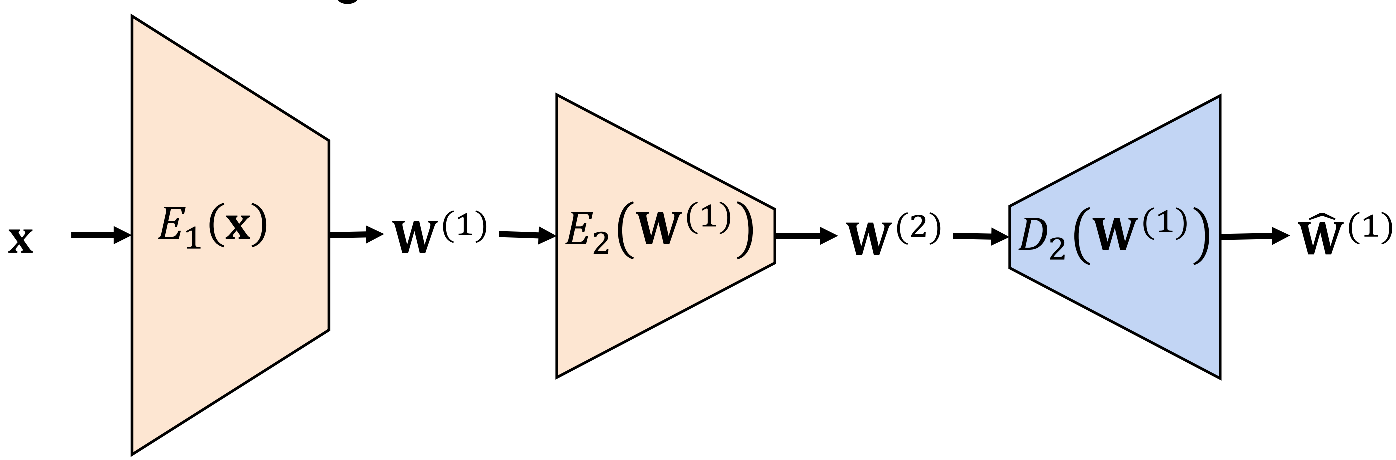
First stage model



Stage-wise training of stacked autoencoders:

1. Train the first-stage autoencoder.
2. After training, remove the decoder layer, construct a new autoencoder by taking the *latent representation* of the previous autoencoder as input.
3. Train the new autoencoder. Note the weights and bias of the encoder from the previously trained autoencoders are fixed when training the newly constructed autoencoder.
4. Repeat steps 2 and 3 until enough layers are trained.

Second stage model



Where can we use feature representation learning?

Given a training dataset, first train the feature representation learning model.

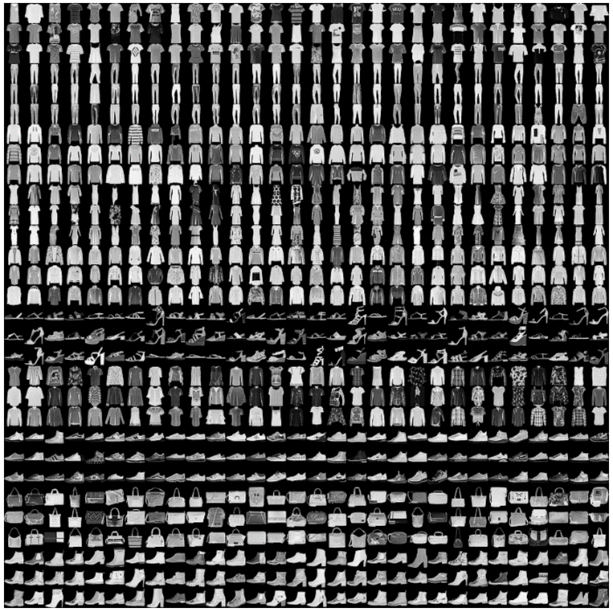
- Feature embedding: The embedded features are used for other machine learning tasks (e.g., classification).
- Anomaly detection: The embedded features are assumed to be ‘common’ feature of such training dataset, which serves as the reference for anomaly detection.

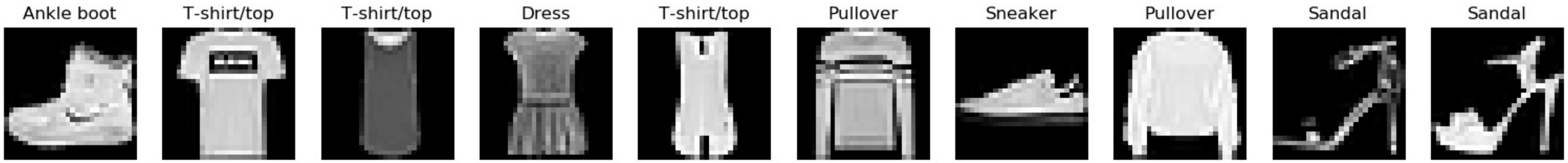
Topics

- Introduction to signal representation
- Data driven signal representation
- Signal representation using machine learning
- Applications of signal representation learning using machine learning
- Workshop

Case 1: Signal representation with dimension reduction

Fashion-MNIST dataset, https://github.com/zalandoresearch/fashion-mnist
Fashion-MNIST is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.





Case 1: Signal representation with dimension reduction

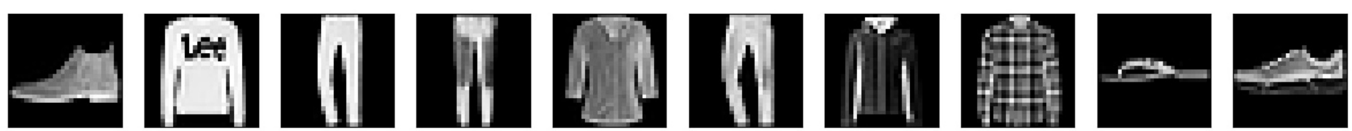
Model

```
# input placeholder
input_image = Input(shape=(ENCODING_DIM_INPUT, ))

# encoding layer
hidden_layer = Dense(ENCODING_DIM_OUTPUT, activation='relu')(input_image)

# decoding layer
decode_output = Dense(ENCODING_DIM_INPUT, activation='relu')(hidden_layer)

# build autoencoder, encoder, decoder
autoencoder = Model(inputs=input_image, outputs=decode_output)
encoder = Model(inputs=input_image, outputs=hidden_layer)
```

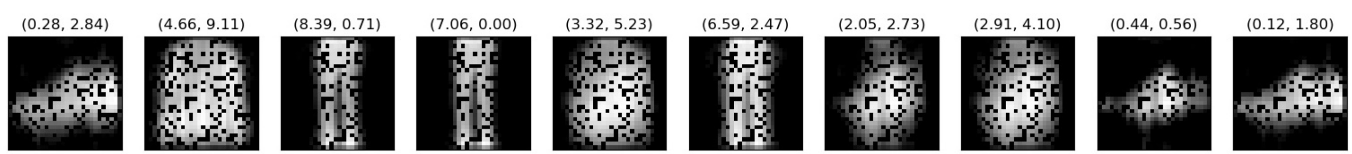
Model architecture

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 1570
_________________________________________________________________
dense_2 (Dense)              (None, 784)               2352
=================================================================
Total params: 3,922
Trainable params: 3,922
Non-trainable params: 0
_________________________________________________________________
```

Original image

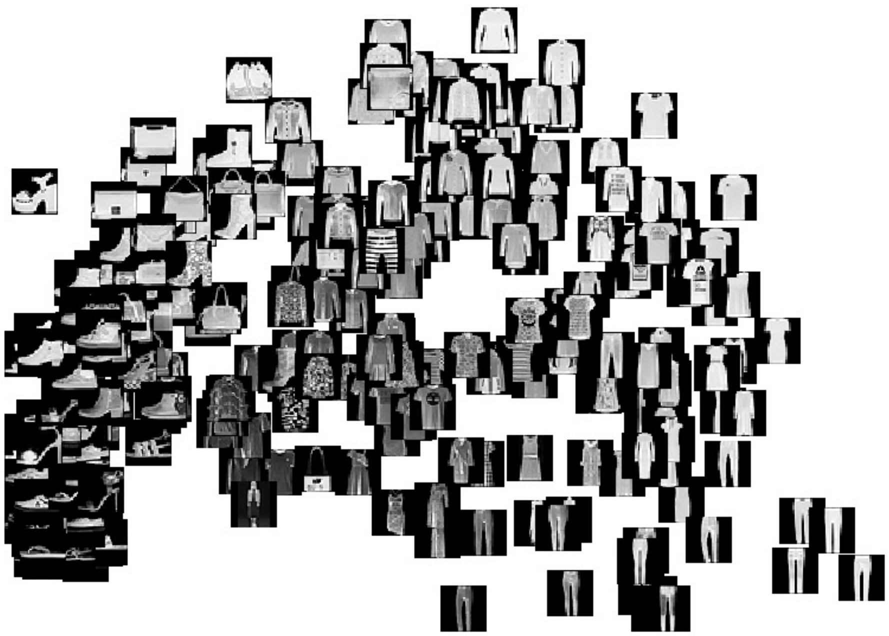


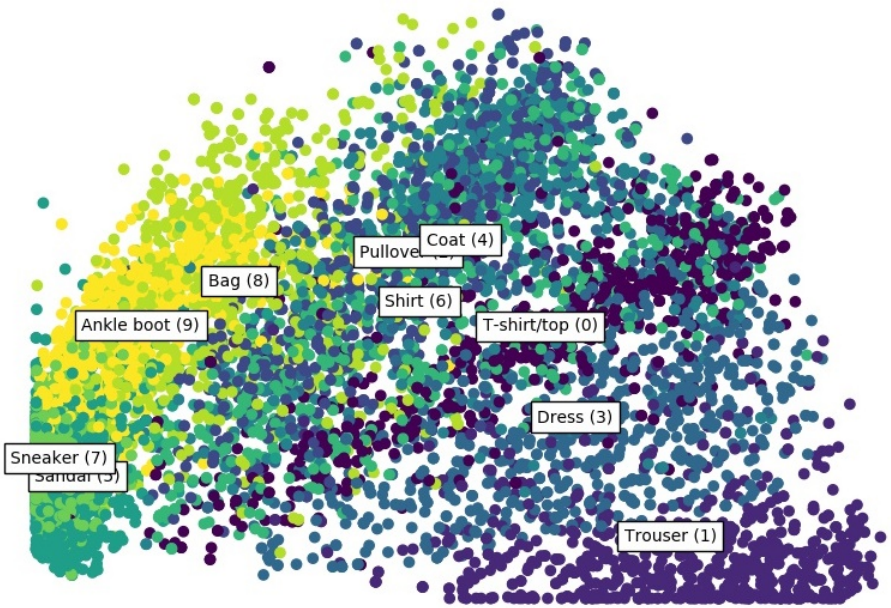
Reconstructed image and code



Case 1: Signal representation with dimension reduction

Visualization using two codes





Case 2: Learned signal representation for classification

Leaned signal representation can be further used as features for other classifiers.

Learned signal representation (10 dimensions)

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 784)	0
dense_1 (Dense)	(None, 10)	7850
dense_3 (Dense)	(None, 128)	1408
dense_4 (Dense)	(None, 10)	1290

Learned signal representation (100 dimensions)

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 784)	0
dense_1 (Dense)	(None, 100)	78500
dense_3 (Dense)	(None, 128)	12928
dense_4 (Dense)	(None, 10)	1290

Classification performance, precision, recall, F1-score

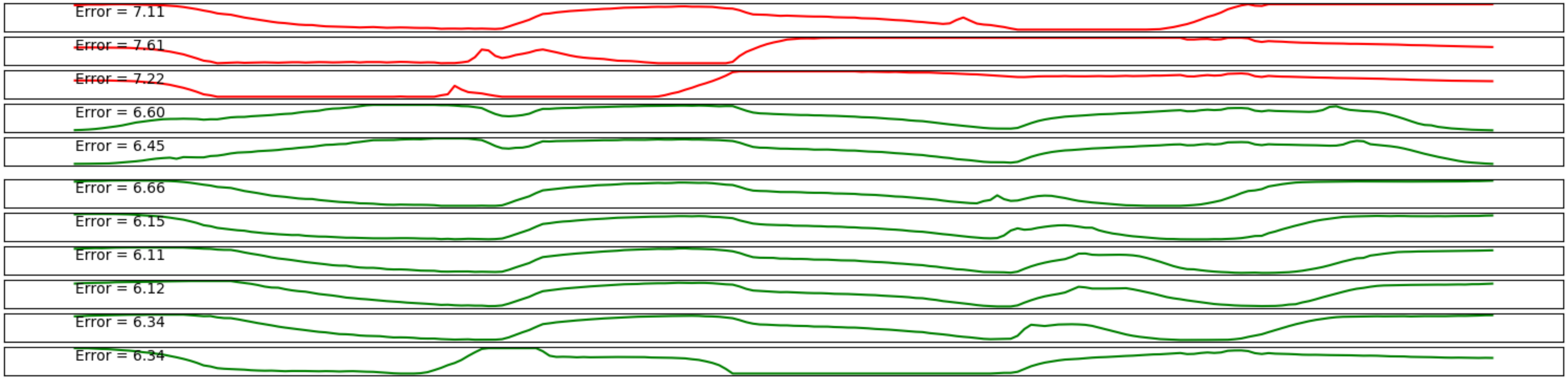
Class 0	0.58	0.90	0.70
Class 1	0.99	0.95	0.97
Class 2	0.85	0.68	0.76
Class 3	0.87	0.87	0.87
Class 4	0.75	0.81	0.78
Class 5	0.95	0.95	0.95
Class 6	0.79	0.49	0.60
Class 7	0.94	0.93	0.94
Class 8	0.97	0.96	0.96
Class 9	0.95	0.95	0.95

Class 0	0.65	0.91	0.76
Class 1	0.99	0.96	0.98
Class 2	0.90	0.61	0.72
Class 3	0.90	0.88	0.89
Class 4	0.73	0.88	0.80
Class 5	0.97	0.96	0.97
Class 6	0.77	0.61	0.68
Class 7	0.95	0.95	0.95
Class 8	0.98	0.96	0.97
Class 9	0.96	0.96	0.96

Reference: https://www.datacamp.com/community/tutorials/autoencoder-classifier-python

Case 3: Anomaly detection

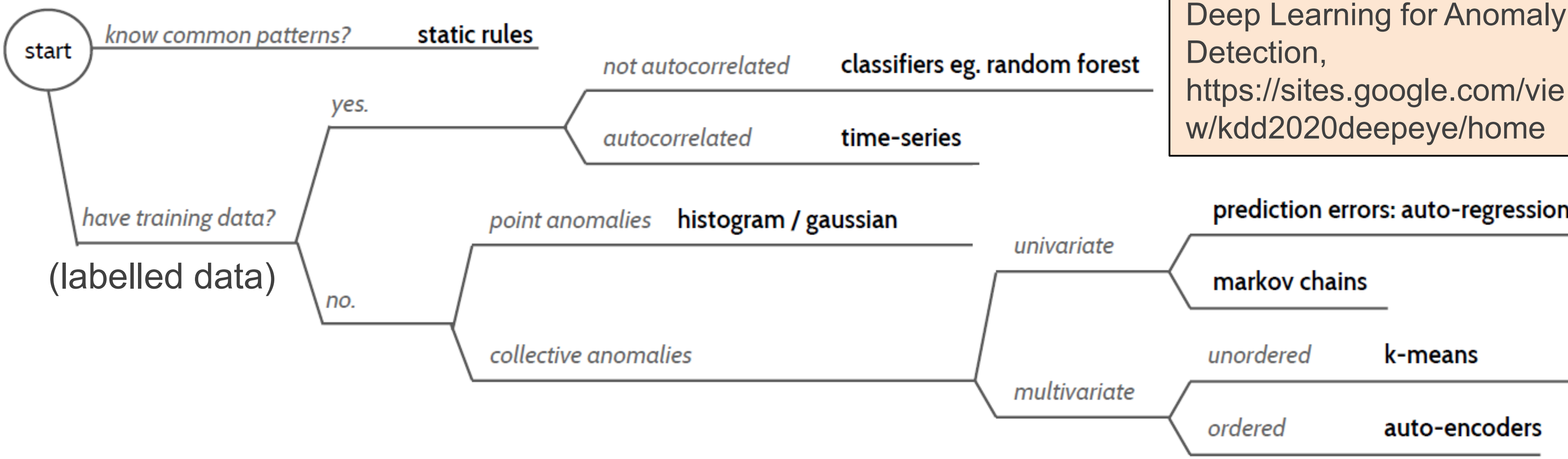
- Supervised
  - Requires labeled anomaly data
- Unsupervised
  - Train an auto-encoder on the training data.
  - Evaluate it on the validation data and the reconstructed error plot.
  - Choose a threshold, which determines whether a value is an outlier (anomalies) or not. This threshold can by dynamic and depends on the previous errors (moving average, time component).

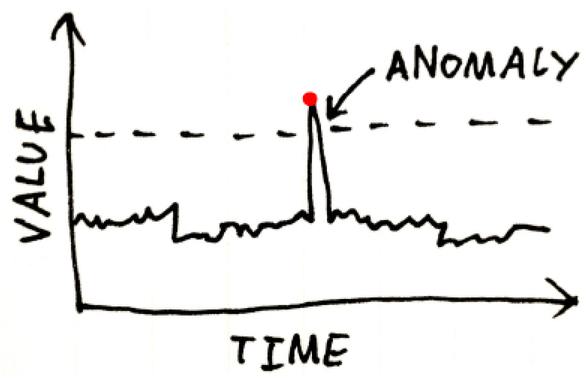


Reference: https://github.com/chen0040/keras-anomaly-detection

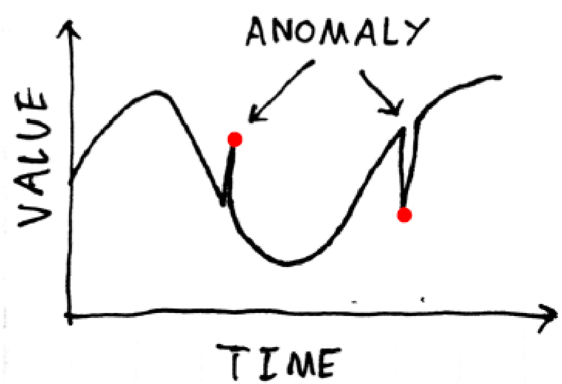
Case 4: Anomaly detection

KDD 2020 Tutorial,
Deep Learning for Anomaly Detection,
https://sites.google.com/view/kdd2020deepeye/home

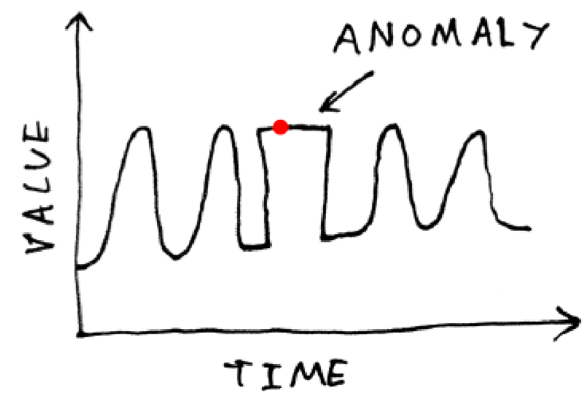




Point anomaly



Contextual anomaly



Collective anomaly

Reference: https://www.aisingapore.org/forums/forum-ai-meetups/the-science-of-anomaly-detection-meetup-slides/

Use of data labels in anomaly detection

- Supervised anomaly detection
  - Labels available for both normal data and anomalies
  - Similar to classification with high class imbalance
- Unsupervised anomaly detection
  - No labels assumed
  - Based on the assumption that anomalies are very rare compared to normal data

Output of anomaly detection

- Label
  - Each test instance is given a *normal* or *anomaly* label
  - Typical output of classification-based approaches
- Score
  - Each test instance is assigned an anomaly score
    - allows outputs to be ranked
    - requires an additional threshold parameter

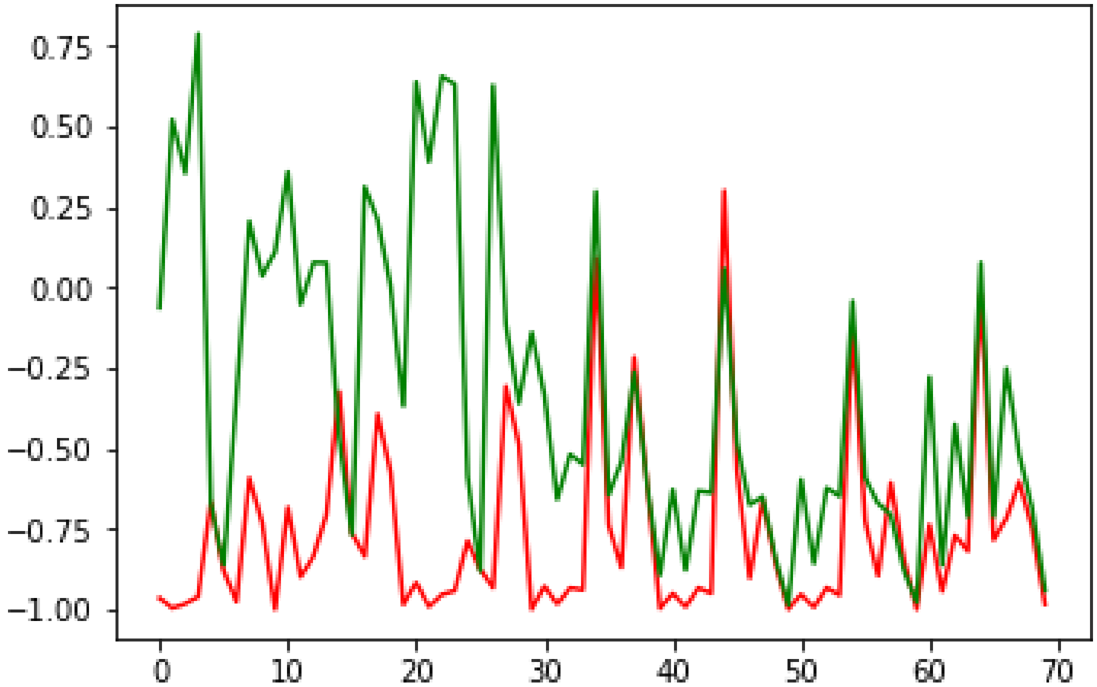
Workshop

- Perform machine health monitoring using autoencoder as feature extraction
- Keras reference: https://keras.io/getting_started/intro_to_keras_for_engineers/
- PyTorch reference: https://pytorch.org/tutorials/

Workshop

- Perform anomaly detection using autoencoder as reconstruction model

Once you finish the workshop, rename your .ipynb file to your name, and submit your .ipynb file into LumiNUS.



Reference: https://github.com/ClockworkBunny/MHMS_DEEPLEARNING

Summary

- Signal representation learning
- Applications of machine learning for sensor signal

(Following topics are not covered in this course)

- Other machine learning methods, such as Convolutional neural network, Recurrent neural network, Long short-term memory network
- Visualization and interpretable machine learning
- Best practice of machine learning, e.g., augmentation, loss, optimizer, etc

Thank you!

Dr TIAN Jing
Email: tianjing@nus.edu.sg

