# 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

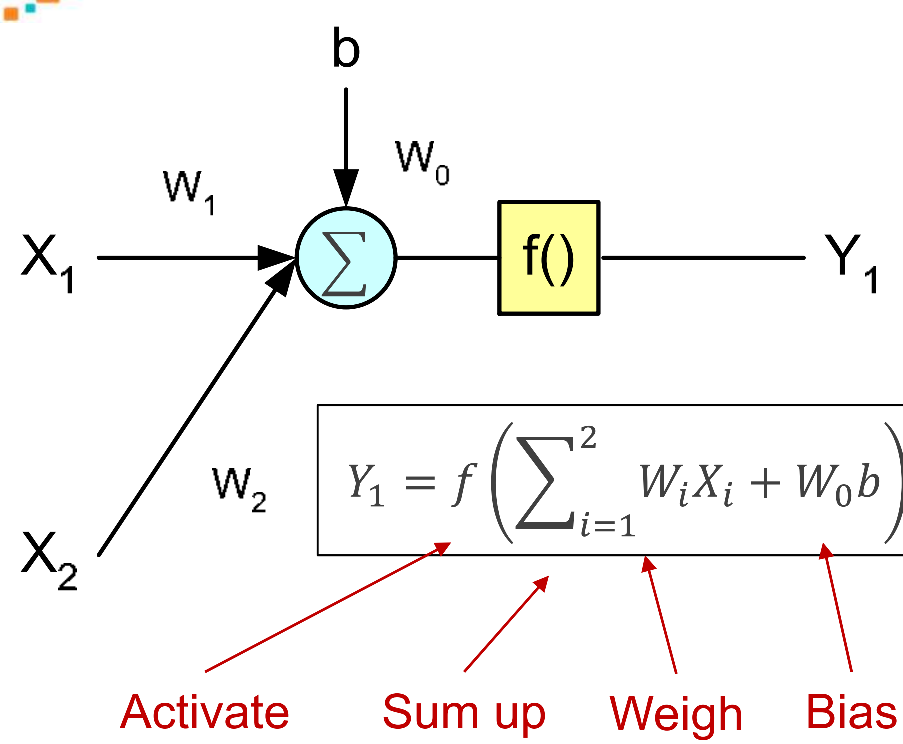
# Major reference

- [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

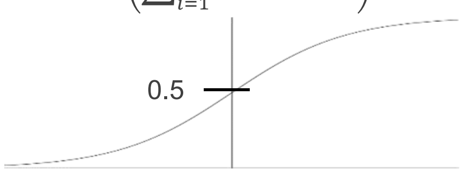



$$Y_1 = f\left(\sum_{i=1}^{2} W_i X_i + W_0 b\right)$$

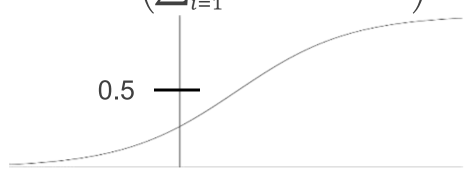
| Notation | |
|---|---|
| $X_1, X_2$ | Input |
| $Y_1$ | Output |
| $b$ | Bias ($b = 1$) |
| $W_i$ | Weighting factor (for each arrow) for the $i-$th input data |
| $\sum()$ | Summation function |
| $f()$ | Activation function (e.g., sigmoid function) |
| A training data record: $X_1, X_2, Y_1$ | |

Note: A bias $b$ is similar to intercept in regression model, such as $y = \alpha + \beta x$ with a slop $\beta$ and intercept $\alpha$. 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

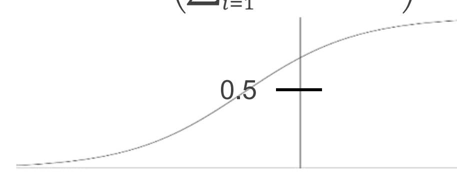
$$\text{sigmoid}\left(\sum_{i=1}^{2} W_i X_i + 0 \times b\right)$$



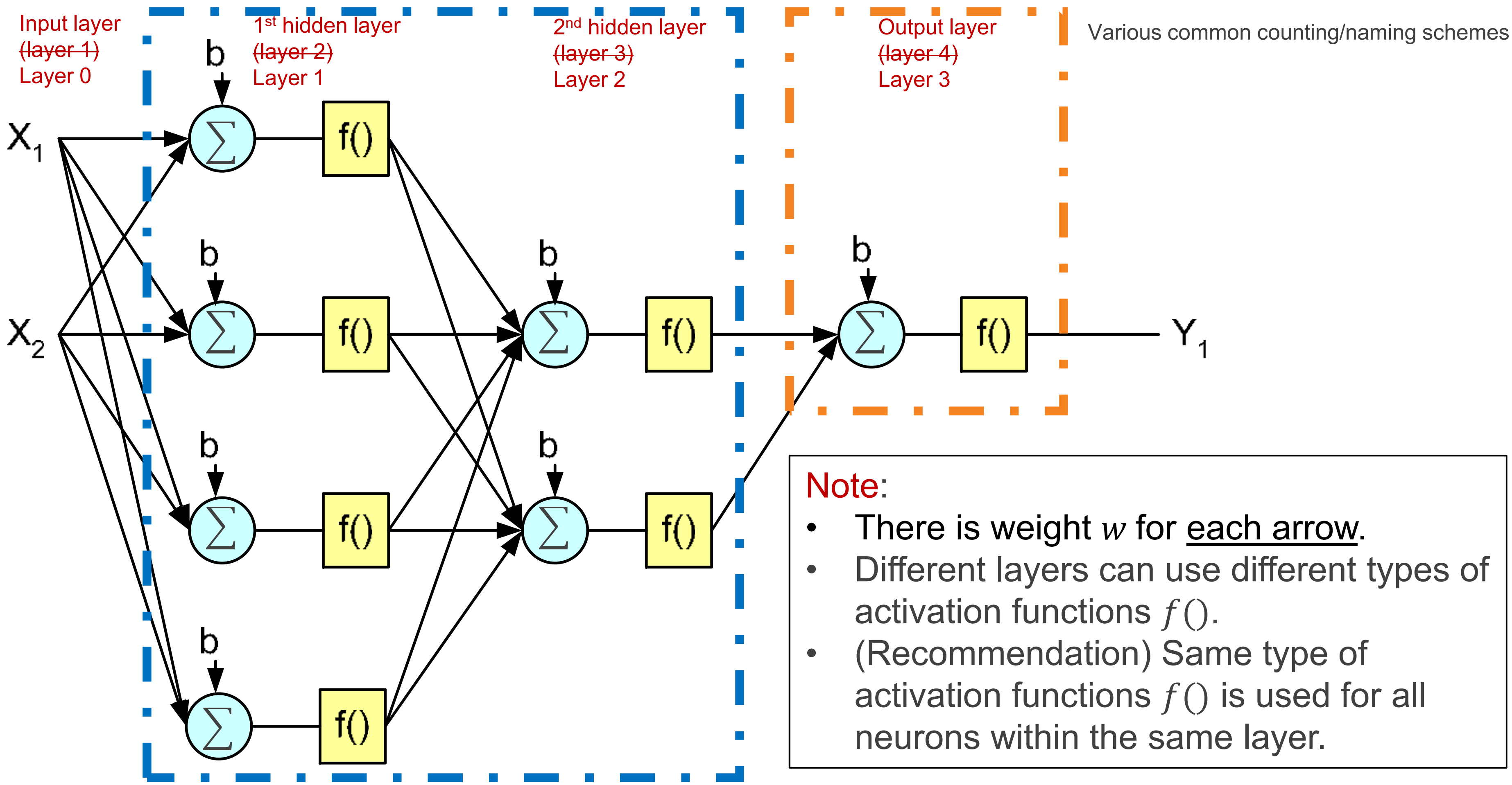

$$\text{sigmoid}\left(\sum_{i=1}^{2} W_i X_i + (-1) \times b\right)$$




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



# Recall: Neural network

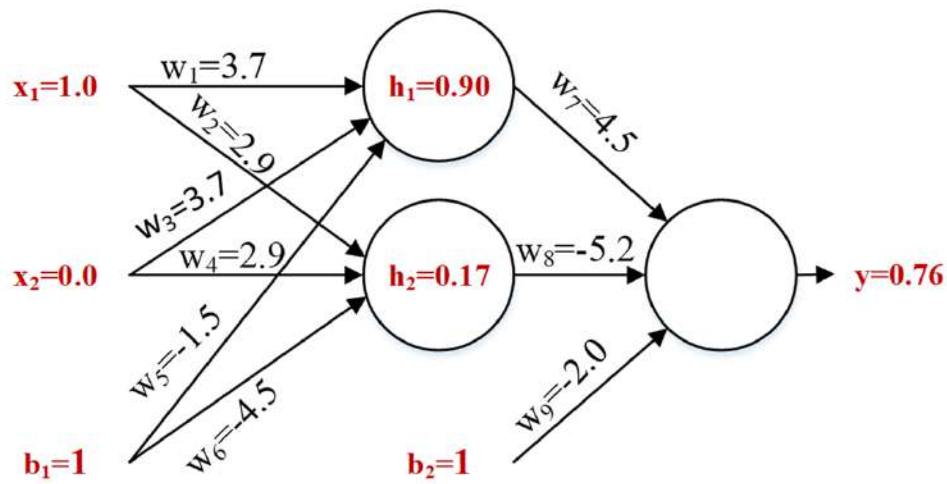



Note:

- 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

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

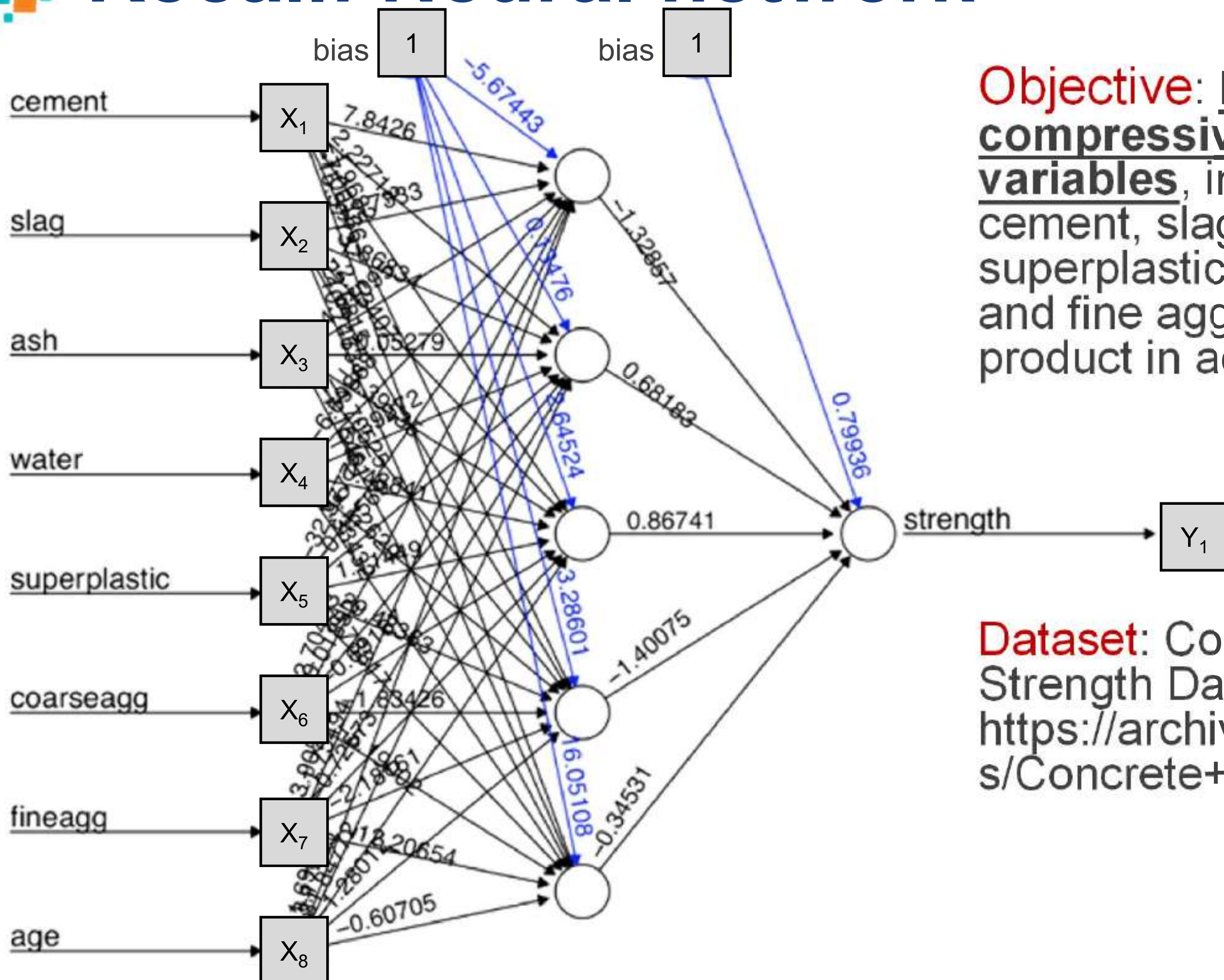
$$h_2 = \text{sigmoid}\big(1.0 \times 2.9 + 0.0 \times 2.9 + 1 \times (-4.5)\big) = \text{sigmoid}(-1.6) = \frac{1}{1+e^{1.6}} = 0.17$$

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

<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

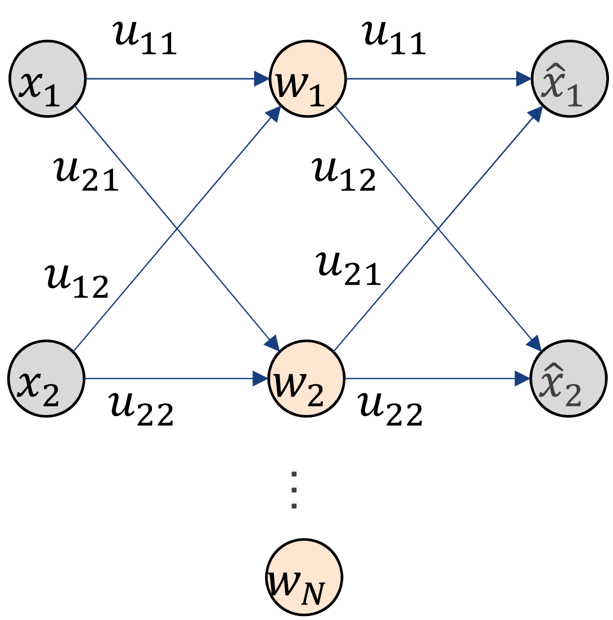
| No. of Input | No. of Hidden | | No. of Output |
|---|---|---|---|
| 8 | Layer | 1 | 1 |
| | No. of nodes at each hidden layer | 5 | |

# 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 $\hat{\mathbf{x}} = \sum_{i=1}^{N} w_i \times \mathbf{u}_i$, for example, $\hat{x}_1 = w_1 u_{11} + w_2 u_{21} + \cdots + w_N u_{N1}$, $\hat{x}_2 = w_1 u_{12} + w_2 u_{22} + \cdots + w_N u_{N2}$



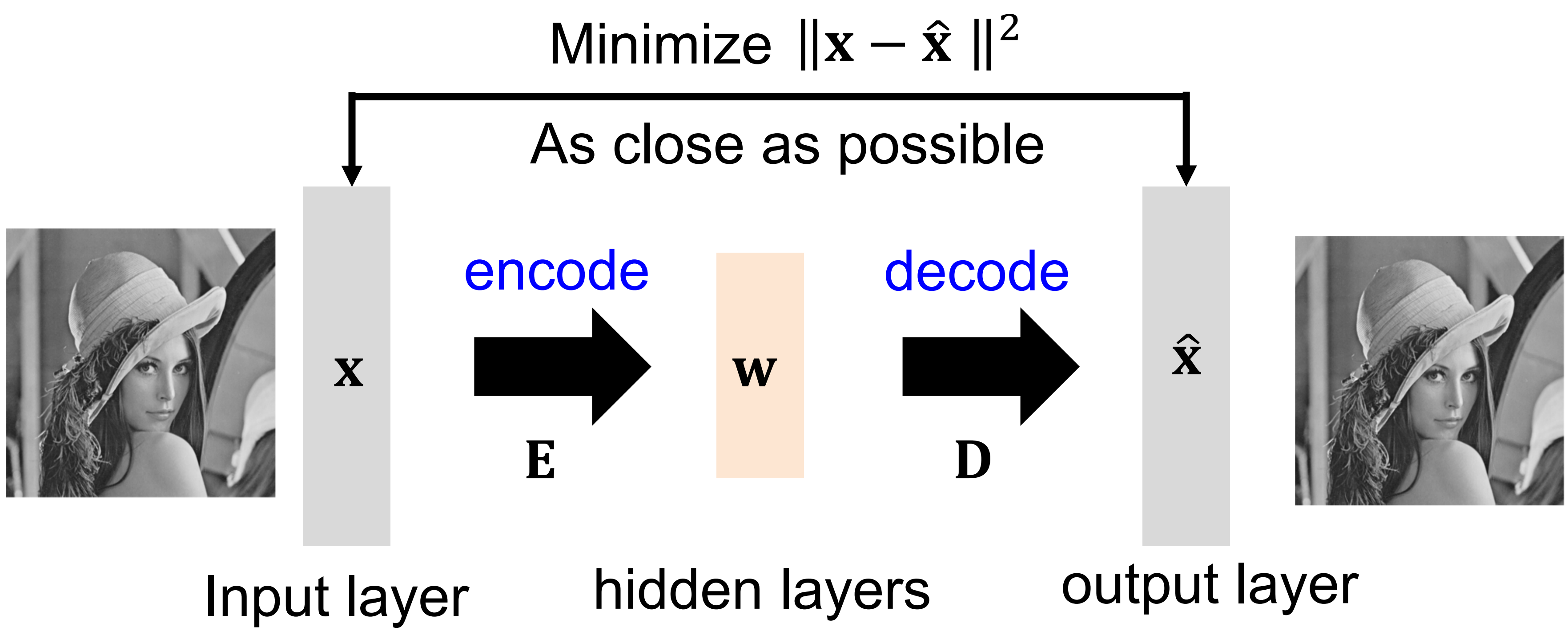

| Signal representation as neural network | |
|---|---|
| Input | Signal |
| Output | Reconstructed signal |
| Model weights | Basis vectors |
| Hidden layer output | Signal representation coefficients |

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

# Signal representation learning

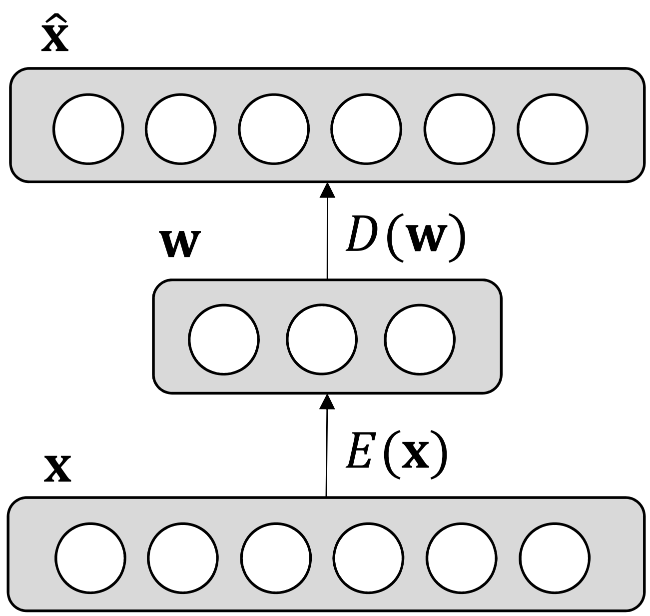
- Can we learn signal representation as neural network?

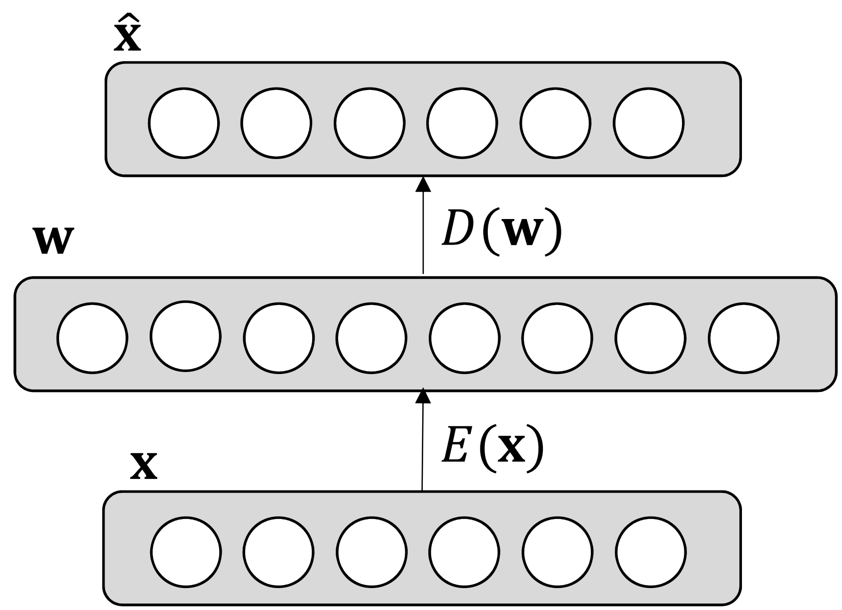


# Signal representation: Under-complete and over-complete

**Under-complete**
- Hidden layer is smaller than the input layer
- Compresses the input
- Hidden nodes will be good features for the training distribution.

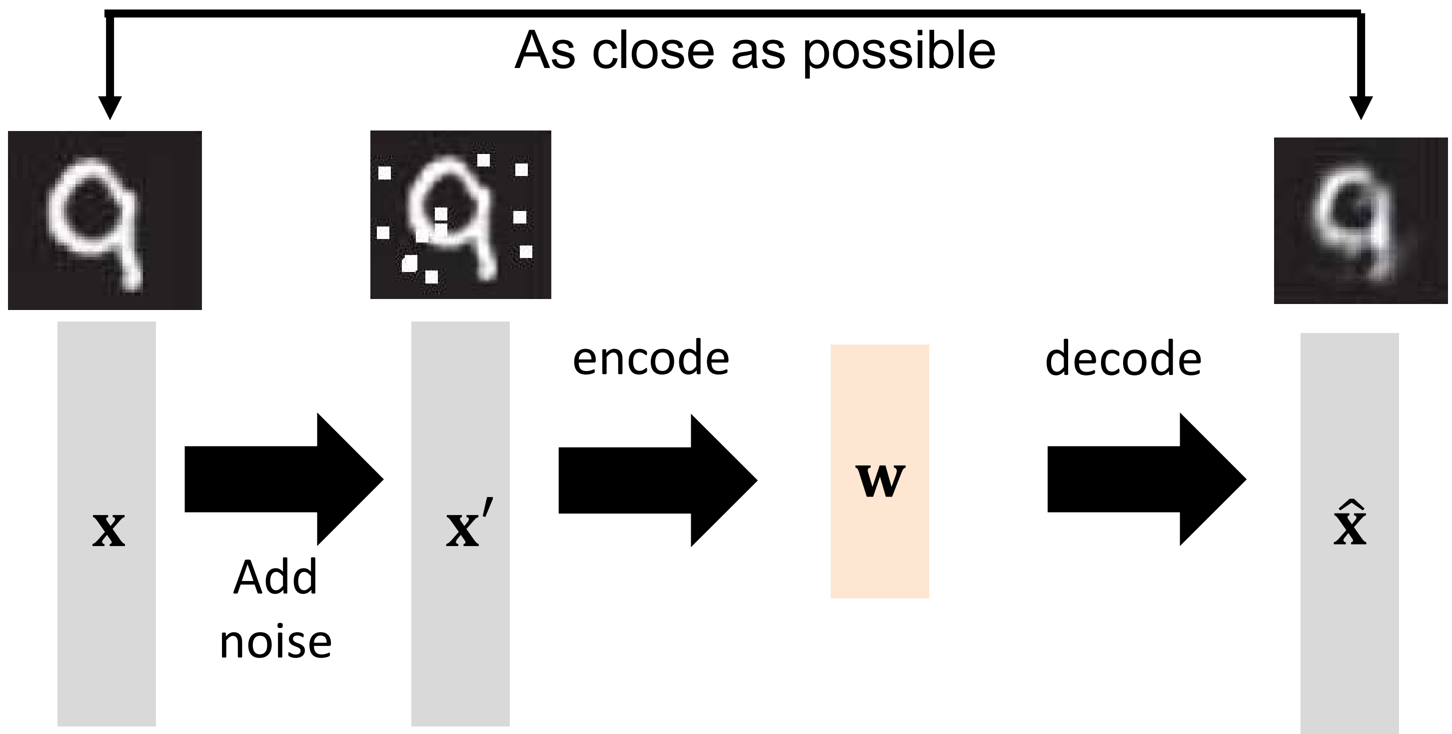
**Over-complete**
- Hidden layer is greater than the input layer
- No compression in hidden layer.
- A higher dimension code helps model a more complex distribution.



# Signal representation learning

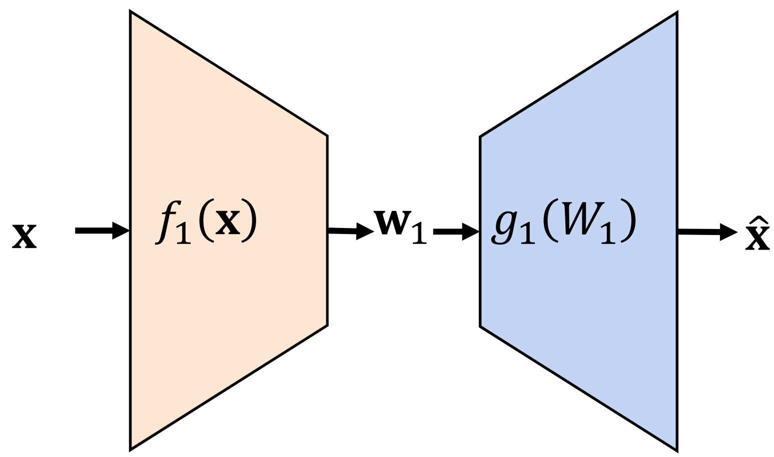
- De-noising auto-encoder: Corrupts input data by injecting noise (e.g., Gaussian noise).



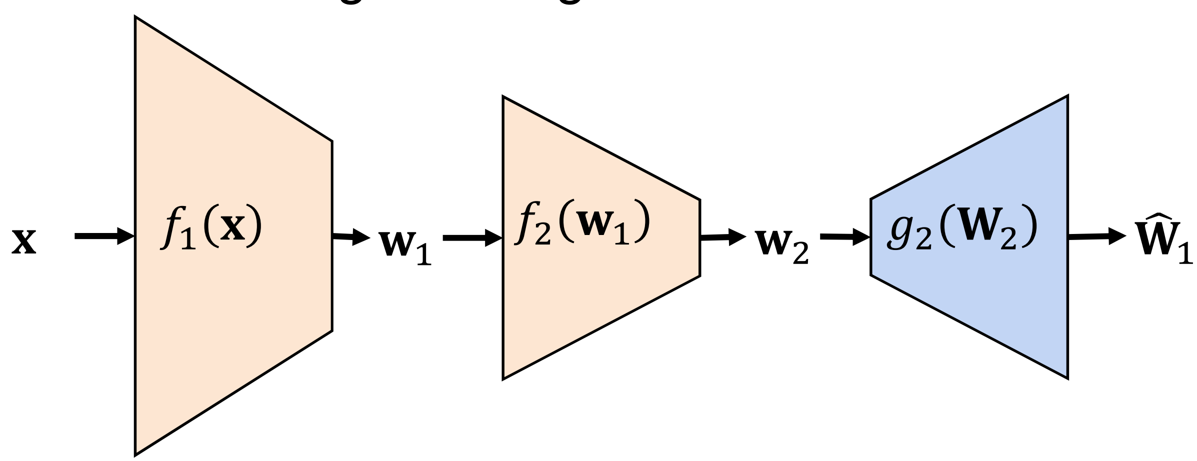

Reference: P. Vincent, H. Larochelle, Y. Bengio, P. Manzagol, “Extracting and composing robust features with denoising autoencoders,” *Int. Conf. on Machine Learning*, Helsinki, Finland, Jul. 2008, pp. 1096-1103.

# Stacked signal representation learning

First stage training




Second stage training

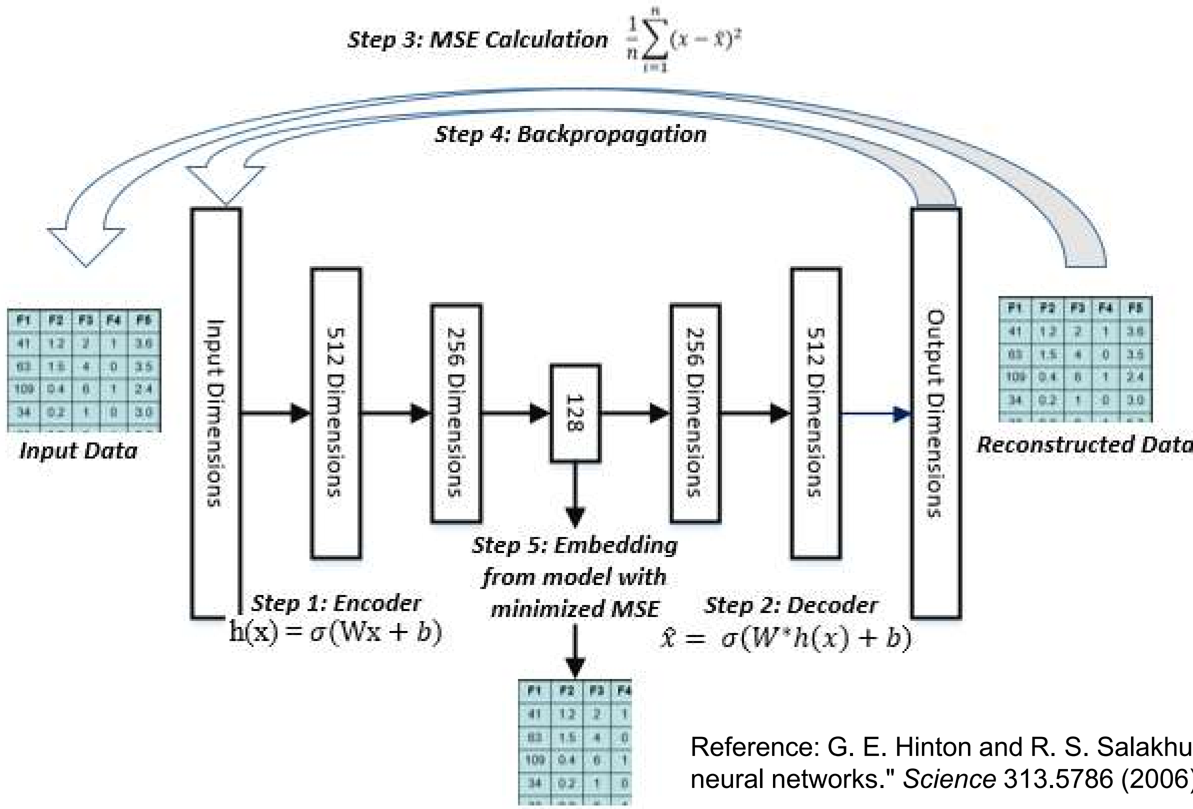



Stage-wise training of stacked autoencoders:

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.

# Signal representation learning can be deeper

- **Step 1:** Encoder “encodes” input data into a embedding using non-linear activation functions.
- **Step 2:** Decoder reconstructs output by using non-linear layers to “decode” embedding.
- **Step 3:** *Mean Squared Error* (MSE) is calculated between the reconstructed output and original input. Error is back-propagated to adjust autoencoder weights.
- **Step 4:** Steps 1-3 are repeated until MSE is minimized.




Reference: G. E. Hinton and R. S. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *Science* 313.5786 (2006): 504-507.

# 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).
- Feature comparison: The embedded features are used to compare similarity between data.
- Anomaly detection: The embedded features are assumed to be ‘common’ feature of such training dataset, which serves as the reference for anomaly detection.

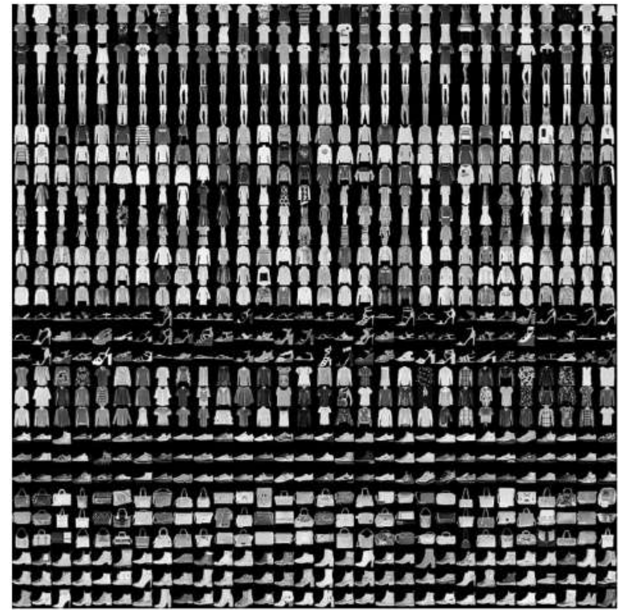
# Topics

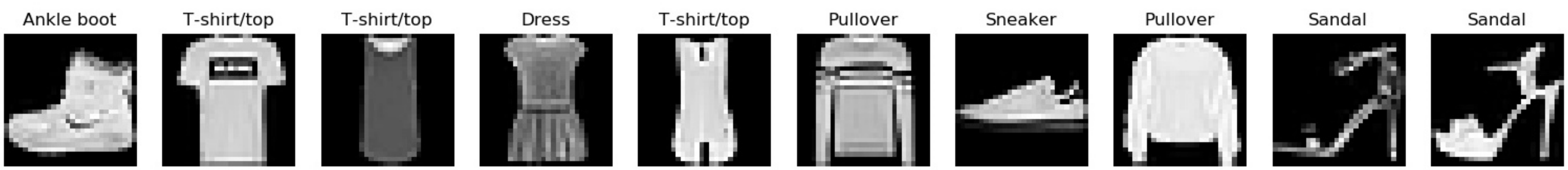
- Introduction to signal representation
- Data driven signal representation
- Signal representation using machine learning
- Applications of signal representation learning using machine learning
- Workshop

# 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

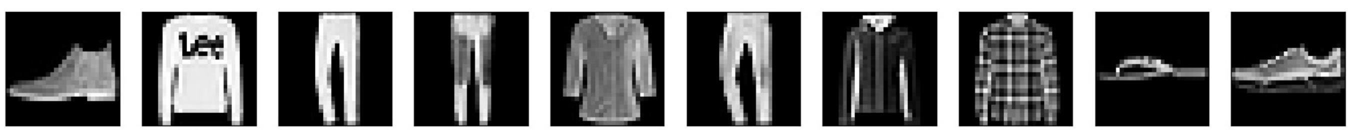
Model

```
# 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

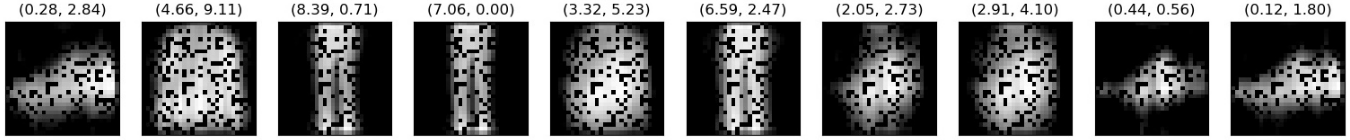
```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)      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

Original image



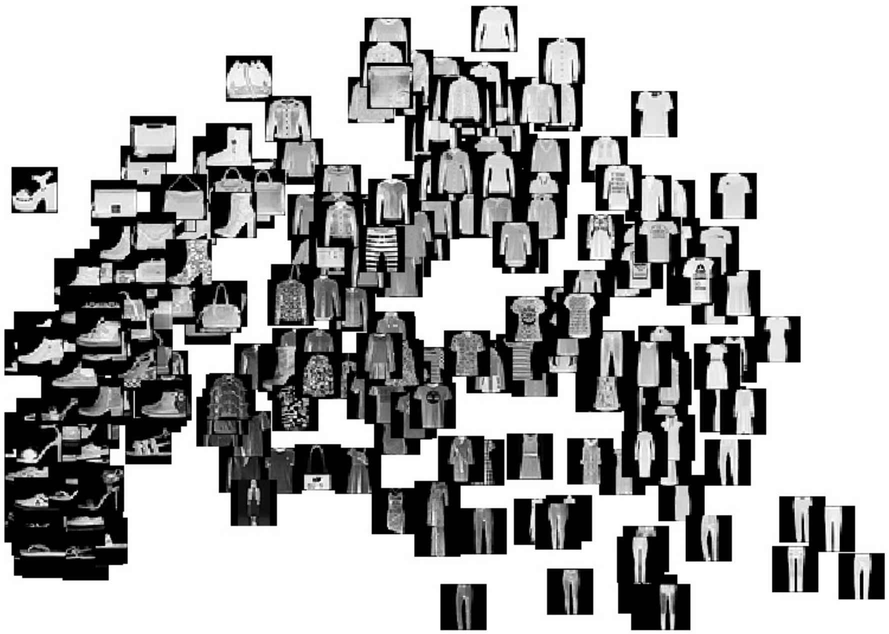
Reconstructed image and code

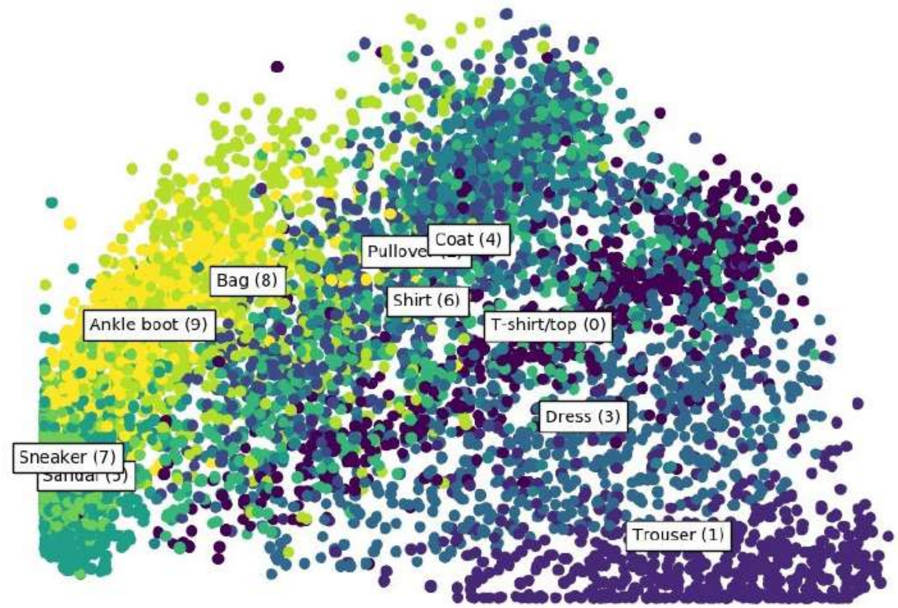
(0.28, 2.84) (4.66, 9.11) (8.39, 0.71) (7.06, 0.00) (3.32, 5.23) (6.59, 2.47) (2.05, 2.73) (2.91, 4.10) (0.44, 0.56) (0.12, 1.80)



# 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 |

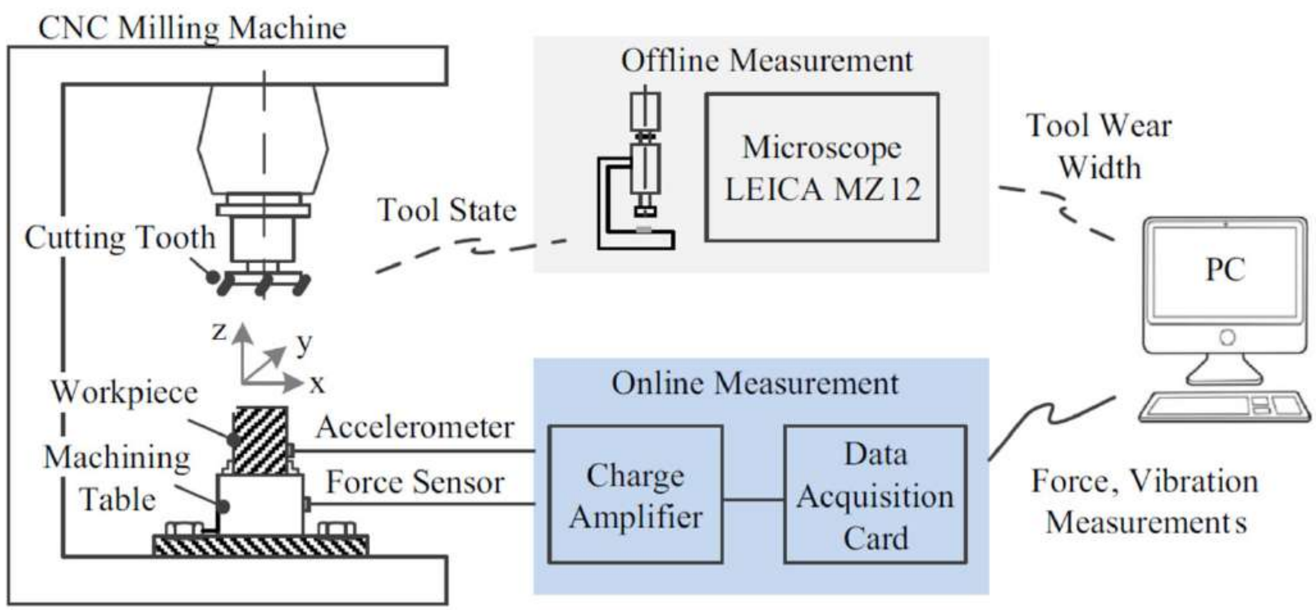
Reference: https://www.datacamp.com/community/tutorials/autoencoder-classifier-python

# Case 3: Machine health monitoring

Dataset description: dataset were sampled from a high speed CNC machine during dry milling operations and its schematic diagram of experimental platform, where seven sensors including force and vibration ones in three directions and AE-RMS have been placed. The ground-truth value were obtained by using a LEICA MZ12 microscope to measure each individual flute after finishing each surface, i.e., each cut number.

Objective: Predict the actual flank wear from the sensory data. c4 is used as testing data while the other records c1 and c6 are used as training data. The input data is the hand-crafted feature vector with dimension of 70.

Raw data: https://www.phmsociety.org/competition/phm/10




Source: R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. Gao, "Deep learning and its applications to machine health monitoring," *Mechanical Systems and Signal Processing*, Vol. 115, Jan. 2019, pp. 213-237.

# Case 3: Machine health monitoring

Model architecture

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense_5 (Dense) | (None, 140) | 9940 |
| dense_6 (Dense) | (None, 280) | 39480 |
| dense_7 (Dense) | (None, 900) | 252900 |
| dense_8 (Dense) | (None, 1) | 901 |

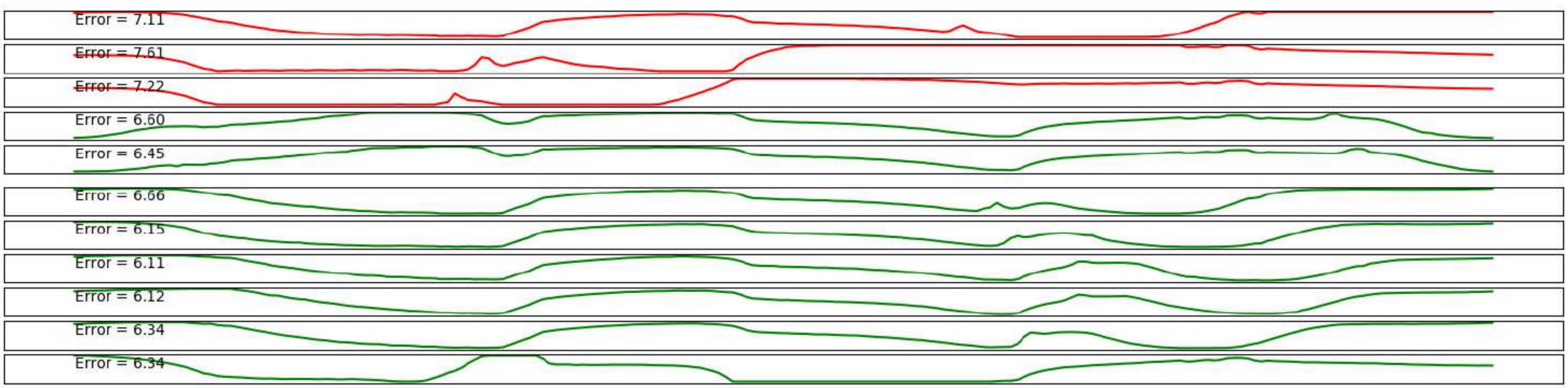
Model

```
ae = Sequential()
ae.add(Dense(hidDim[0],activation='tanh', input_shape=(data_dim, )))
ae.add(Dense(hidDim[1], activation='tanh'))
ae.add(Dense(hidDim[0], activation='tanh'))
ae.add(Dense(data_dim, activation='linear'))
ae.compile(optimizer='rmsprop', loss='mse')
ae.fit(data_train, data_train, epochs=epoch_pretrain, batch_size=24, shuffle=True, verbose=0)
model = Sequential()
model.add(Dense(hidDim[0], input_dim=data_dim, activation='tanh'))
model.add(Dense(hidDim[1], activation='tanh'))
model.add(Dense(FINAL_DIM, activation='tanh'))
model.add(Dense(1))
model.layers[0].set_weights(ae.layers[0].get_weights())
model.layers[1].set_weights(ae.layers[1].get_weights())
model.compile(loss="mean_squared_error", optimizer="rmsprop")
```

Reference: R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. Gao, "Deep learning and its applications to machine health monitoring," *Mechanical Systems and Signal Processing*, Vol. 115, Jan. 2019, pp. 213-237. Sample data and code at available at https://github.com/ClockworkBunny/MHMS_DEEPLEARNING

# Case 4: Anomaly detection

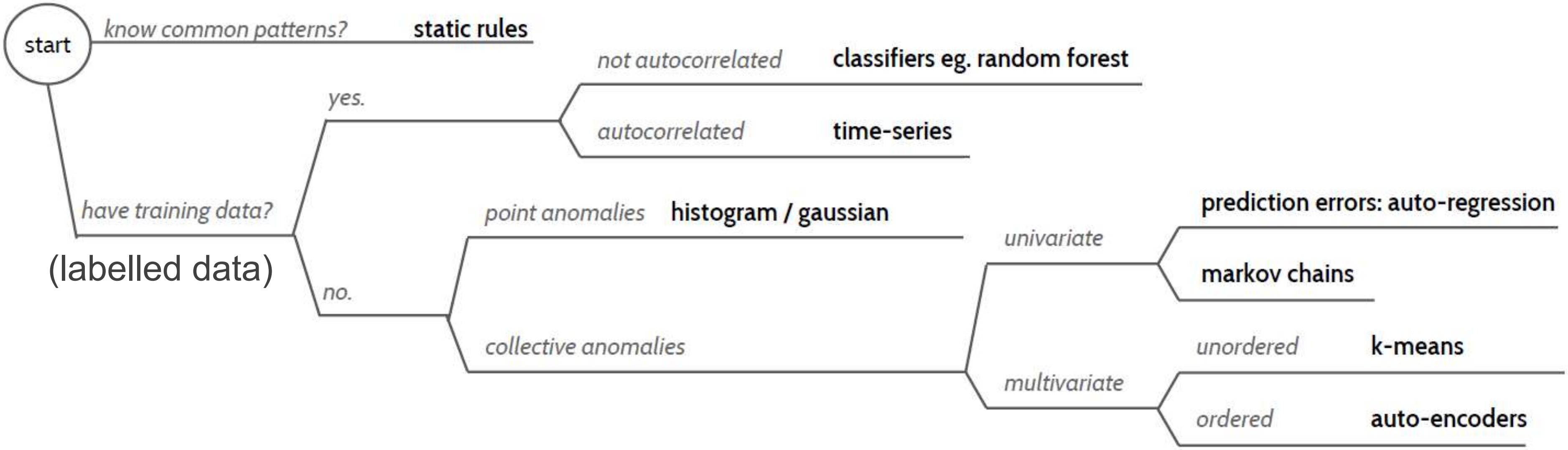
- Supervised
  - Requires labeled anomaly data
- Unsupervised
  - Train an auto-encoder on the training data.
  - Evaluate it on the validation data and the reconstructed error plot.
  - Choose a threshold, which determines whether a value is an outlier (anomalies) or not. This threshold can by dynamic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

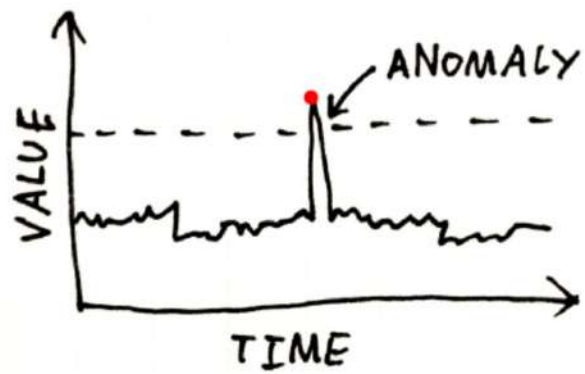



Reference: https://github.com/chen0040/keras-anomaly-detection

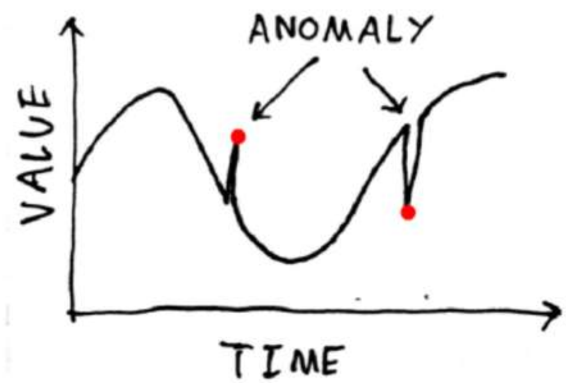
  

# Case 4: Anomaly detection

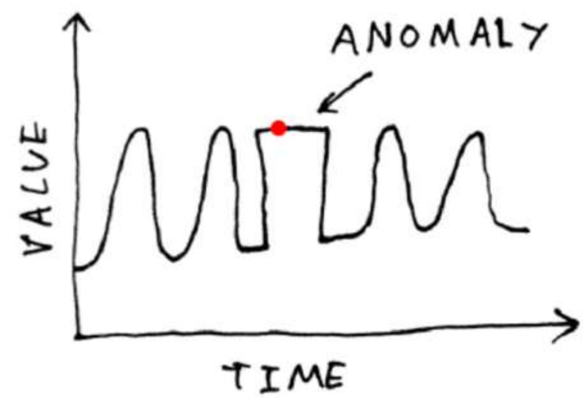



Point anomaly




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

# Auto encoder for anomaly detection

- Unsupervised neural networks or auto encoders are used to replicate the input dataset by restricting the number of hidden layers in a neural network. A reconstruction error is generated upon prediction. Higher the reconstruction error, higher the possibility of that data point being an anomaly.
- Use autoencoder as feature extraction methods, the learned features are used to train another classifier/regression engine.

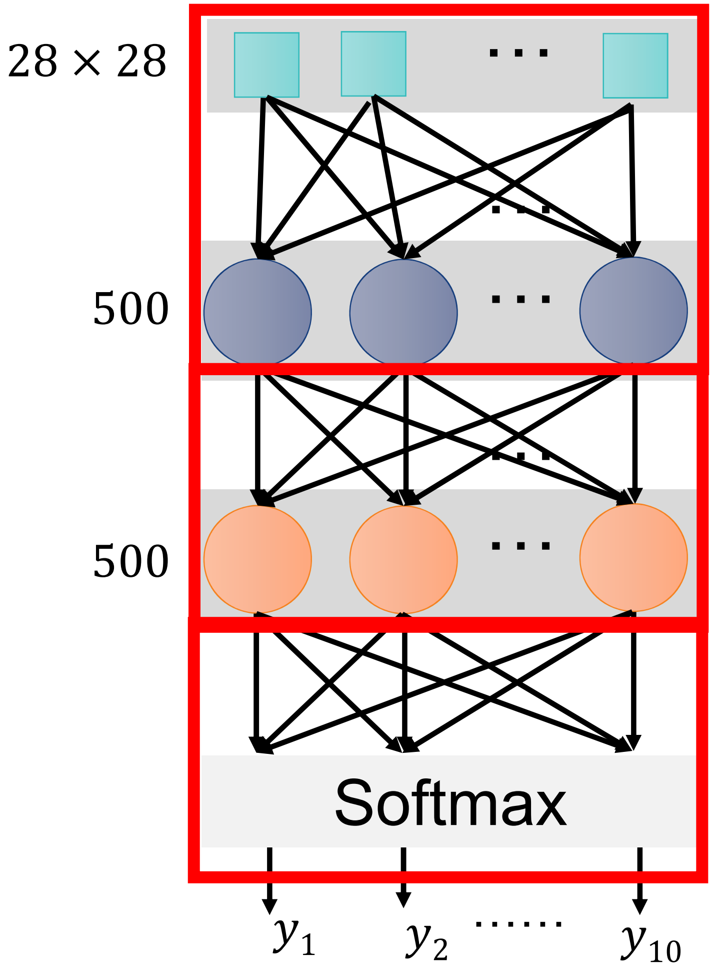
# Workshop

- Introduction to autoencoder model and Keras
- Perform anomaly detection using autoencoder
- Perform machine health monitoring using autoencoder as feature extraction
- Keras reference: https://keras.io/#getting-started-30-seconds-to-keras

# Workshop

Keras Sequential API

Step1: Define a set of function → Step 2: Goodness of function → Step 3: Pick best function




```
model = Sequential()
```

```
model.add( Dense( input_dim=28*28,
                  output_dim=500 ))
model.add( Activation('sigmoid') )
```

```
model.add( Dense( output_dim=500 ) )
model.add( Activation('sigmoid') )
```

```
model.add( Dense(output_dim=10 ) )
model.add( Activation('softmax') )
```

Reference: Hung-Yi Lee, http://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2017/Lecture/Keras.pptx

# Workshop

Keras Functional API

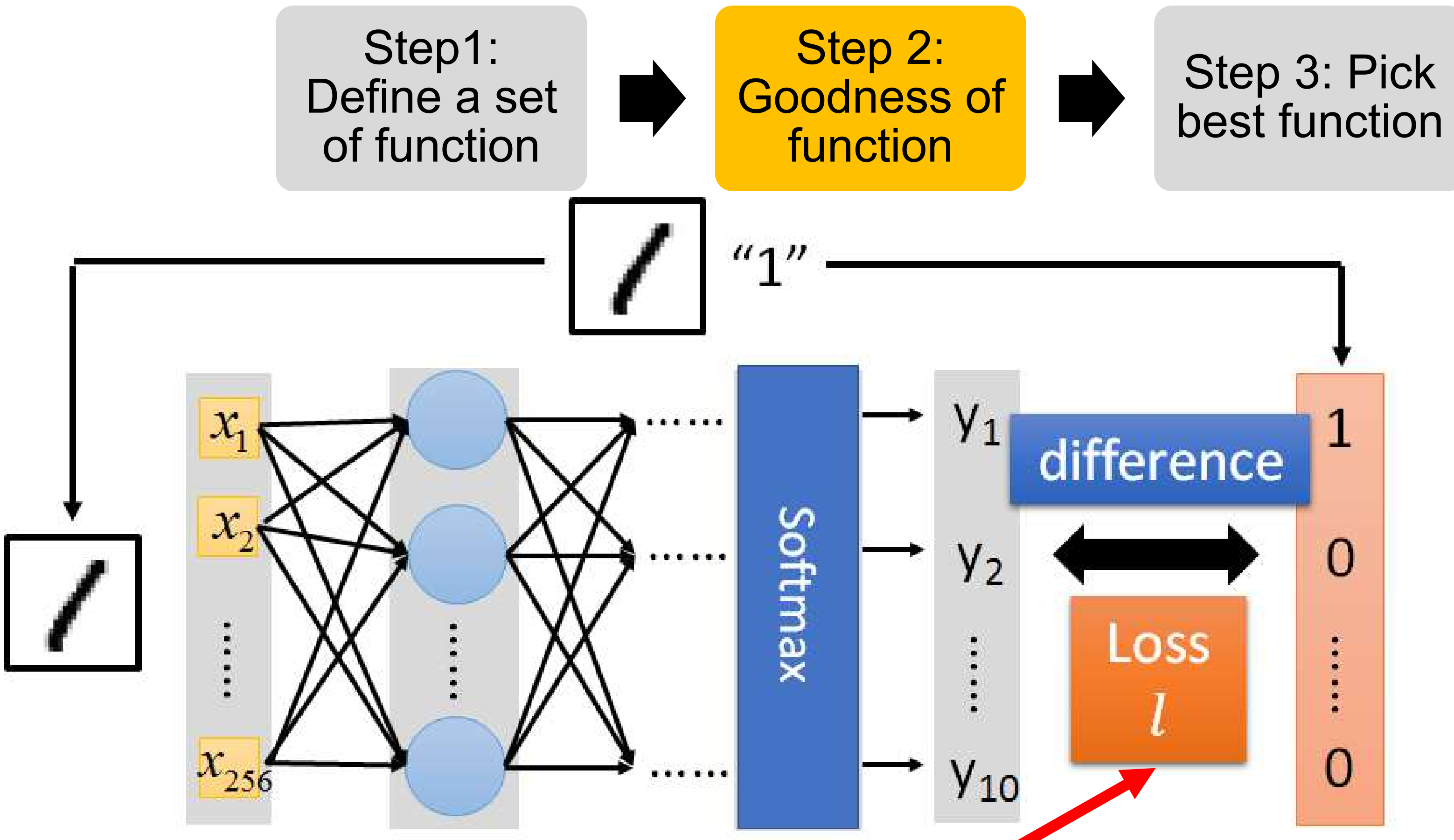
Step1: Define a set of function → Step 2: Goodness of function → Step 3: Pick best function

```
input_image = Input(shape=(28*28, ))
hidden_layer1 = Dense(500, activation='sigmoid')(input_image)
hidden_layer2 = Dense(500, activation='sigmoid')(hidden_layer1)
output_label = Dense(10, activation='softmax')(hidden_layer2)
model = Model(inputs=input_image, outputs=output_label)
```

- The **sequential** API allows you to create models layer-by-layer for most problems. It is limited in that it does not allow you to create models that share layers or have multiple inputs or outputs.
- Alternatively, the **functional** API allows you to create models that have a lot more flexibility as you can easily define models where layers connect to more than just the previous and next layers.

Reference: https://jovianlin.io/keras-models-sequential-vs-functional/

# Workshop




```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Reference: Hung-Yi Lee, http://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2017/Lecture/Keras.pptx

# Workshop

Step1: Define a set of function → Step 2: Goodness of function → **Step 3: Pick best function**

Step 3: Find the optimal network parameters

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

Training data (Images) — `x_train`

Labels (digits) — `y_train`

Training configuration — `batch_size=100, nb_epoch=20`

How to use the neural network (testing)

case 1:

```
score = model.evaluate(x_test,y_test)
print('Total loss on Testing Set:', score[0])
print('Accuracy of Testing Set:', score[1])
```

case 2:

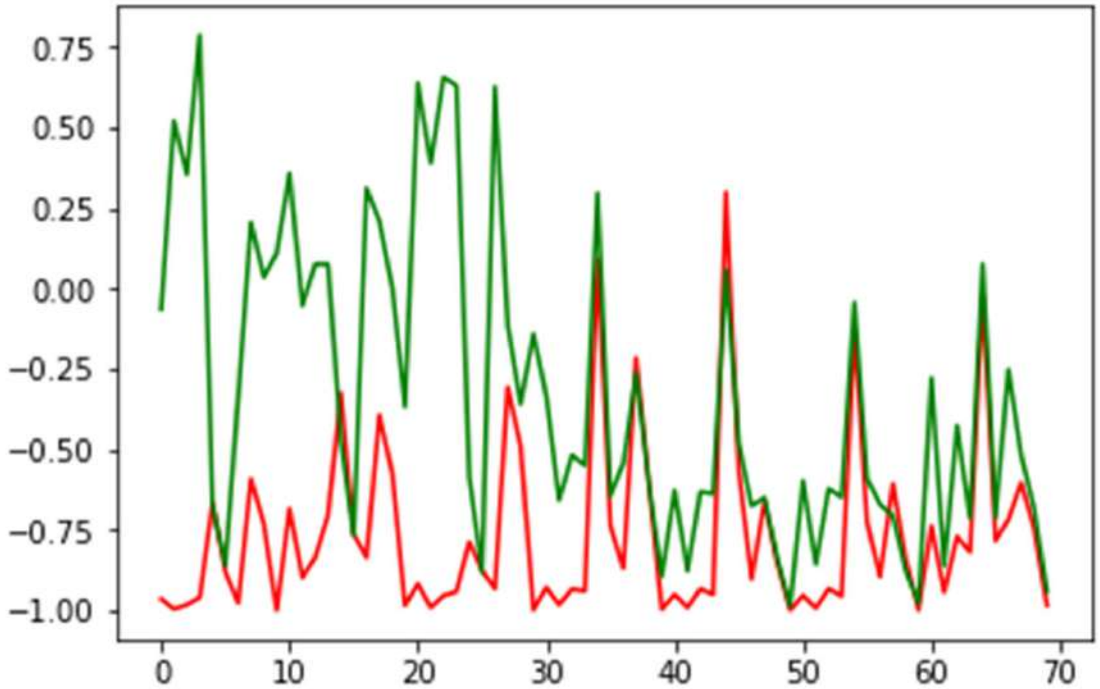
```
result = model.predict(x_test)
```

Reference: Hung-Yi Lee, http://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2017/Lecture/Keras.pptx

# 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

# Thank you!

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