INTELLIGENT SENSOR PROCESSING USING MACHINE LEARNING

Dr TIAN Jing

tianjing@nus.edu.sg

Agenda

- Intelligent sensor processing using machine learning
- *<Morning Break>*
- Intelligent sensor processing using machine learning (Cont'd)
- *<Lunch Break>*
- Intelligent sensor processing using machine learning (Cont'd)
- *<Afternoon Break>*
- Workshop on intelligent sensor processing using machine learning

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

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

Machine learning for signal processing

- Machine learning can be applied on component of signal processing pipeline.

- Various types of sensors, audio, vision, IoT, etc
- Sensing
- Denoising
- Source coding, channel coding, compression
- Deterministic features
- Data-driven features
- Feature representation learning
- Classification
- Regression
- Predication

- Representation: How to represent signals for effective processing
- Modeling: How to model the systematic and statistical characteristics of the signal
- Classification: How do we assign a class to the data
- Prediction: How do we predict new or unseen values or attributes of the data

Sensor signal data

A measurement is one value for an attribute recorded at a specific time point.

Time point	The time point at which the measurement took place (considered in hours for this example)
Heart rate	Beats per minute, integer value
Activity level	Can be either low, medium or high
Speed	Speed in kilometers per hour, real value
Facebook post	A string representing the Facebook message posted
Activity type	The type of activity: inactive, walking, running, cycling, gym

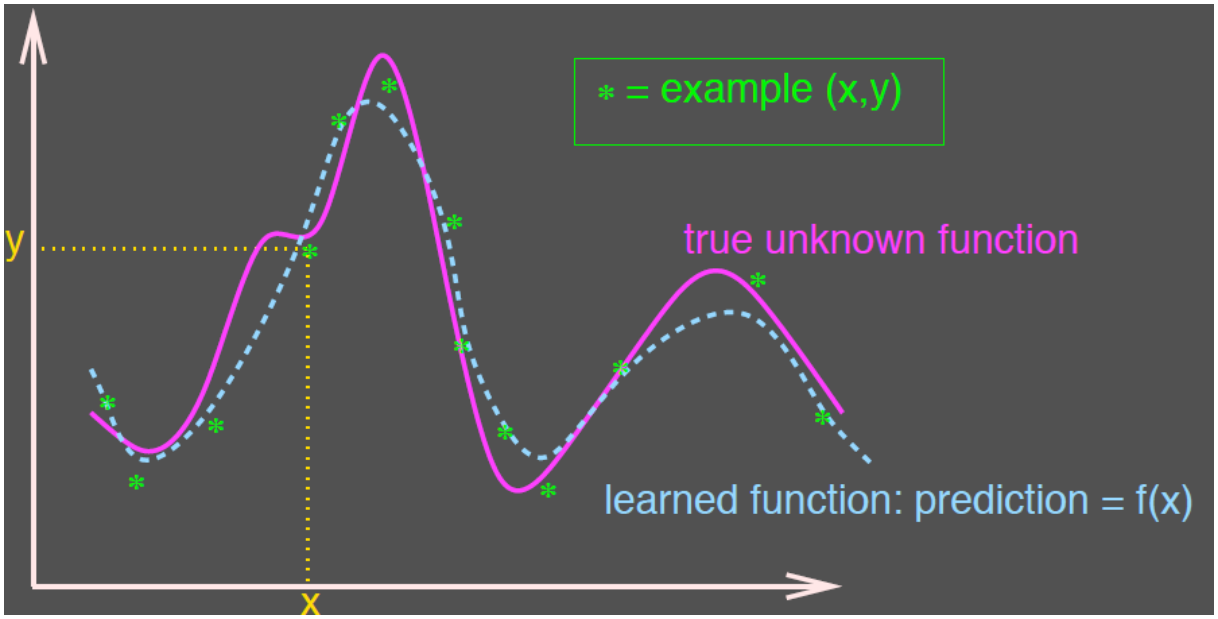
A time series is a series of measurements in temporal order.

Time point	Heart rate	Activity level	Speed	Facebook post	Activity type
14:30	55	low	0	getting ready to hit the gym	inactive
14:45	55	low	0	having trouble getting off the couch	inactive
15:00	70	medium	5	walking to the gym, it's gonna be a great workout, I feel it	walking
15:10	130	high	0	-	gym
15:50	120	high	12	the gym didn't do it for me, running home	running

Signal representation requirements

1. Smoothness

- By “smoothness” we mean that close inputs are mapped to close outputs (representations).
- Smoothness-based learners and linear models can be useful on top of learned representations.

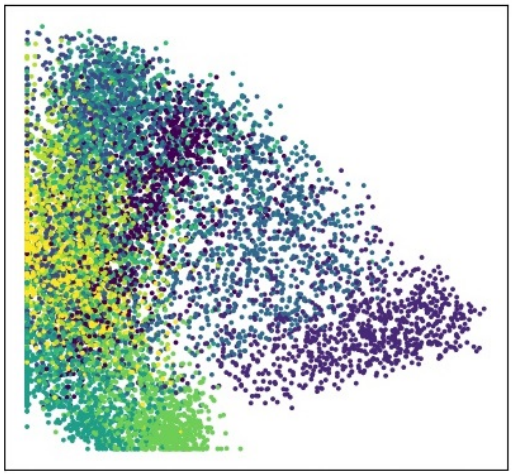


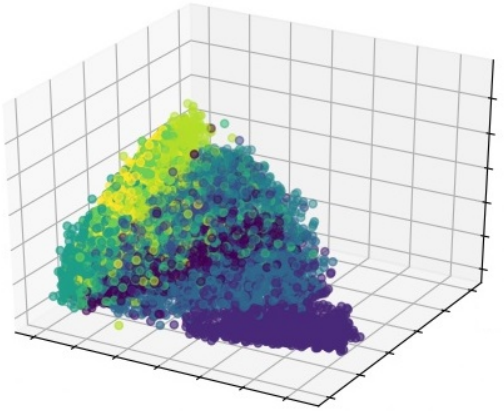
Source: ECE 8527, Introduction to Machine Learning and Pattern Recognition, https://www.isip.piconepress.com/courses/temple/ece_8527/

Signal representation requirements

2. The curse of dimensionality

- We need a representation with a lower dimension than the input dimension, to avoid complicated calculations or having too many configurations.
- That is, the quality of the learned representation increases as its dimension is smaller, but note that we mustn't loose too much important data.



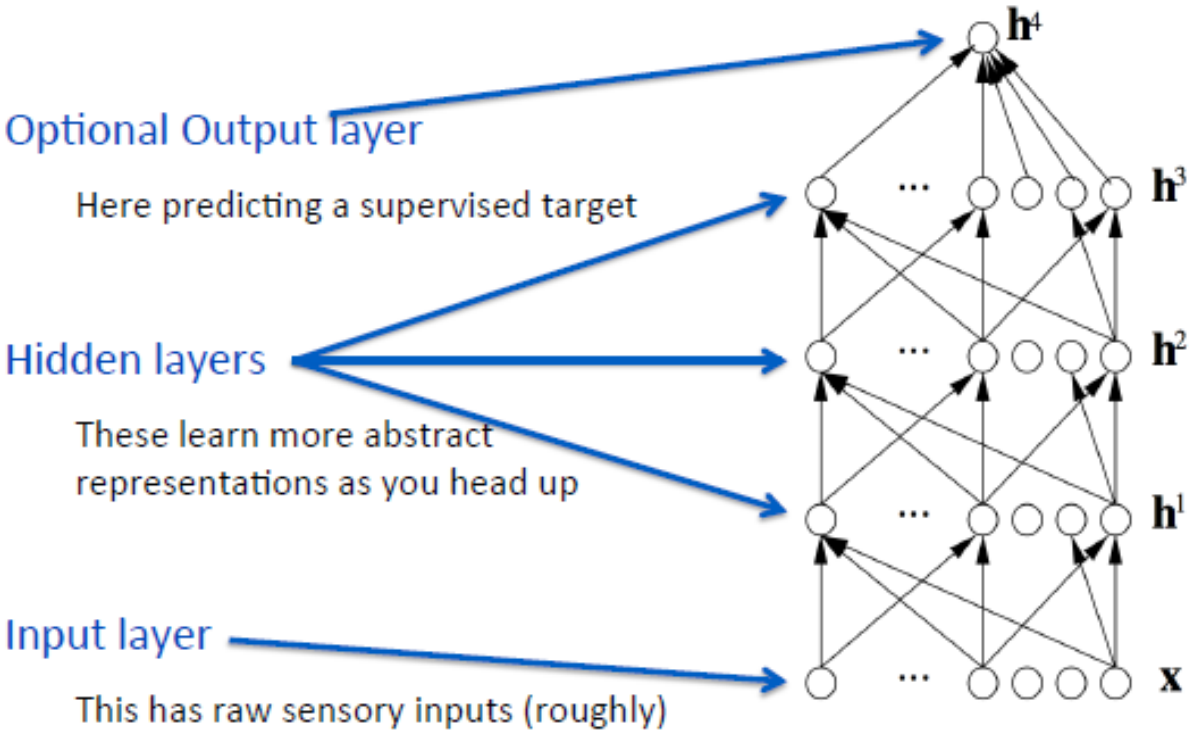


Source: ECE 8527, Introduction to Machine Learning and Pattern Recognition, https://www.isip.piconepress.com/courses/temple/ece_8527/

Signal representation requirements

3. Depth and abstraction

- A good representation expresses high level and abstract features. In order to achieve such representations we can use deep architectures that allow reuse of low level features to potentially get more abstract features at higher layers.



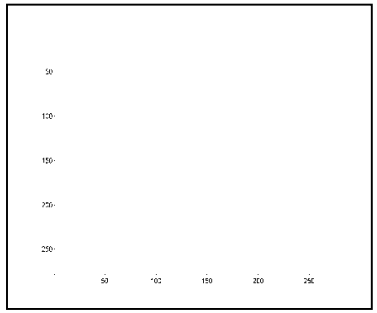
Source: ECE 8527, Introduction to Machine Learning and Pattern Recognition, https://www.isip.piconepress.com/courses/temple/ece_8527/

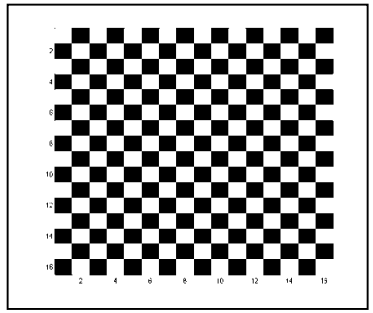
Topics

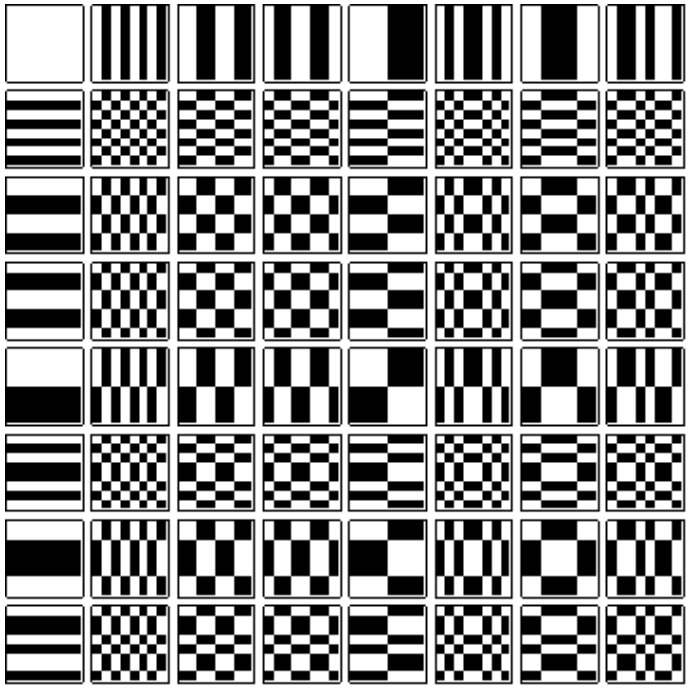
- Introduction to signal representation
- Data driven signal representation
- Signal representation using machine learning
- Applications of signal representation learning using machine learning
- Workshop

Recall: Signal representation

- All the basis we have considered so far are data agnostic.
  - Checkerboards, Complex exponentials, Wavelets..
  - We use the same bases regardless of the data we analyze
- How about data specific bases that consider the underlying data? Is there something better than checkerboards?







Requirement: Energy compaction property

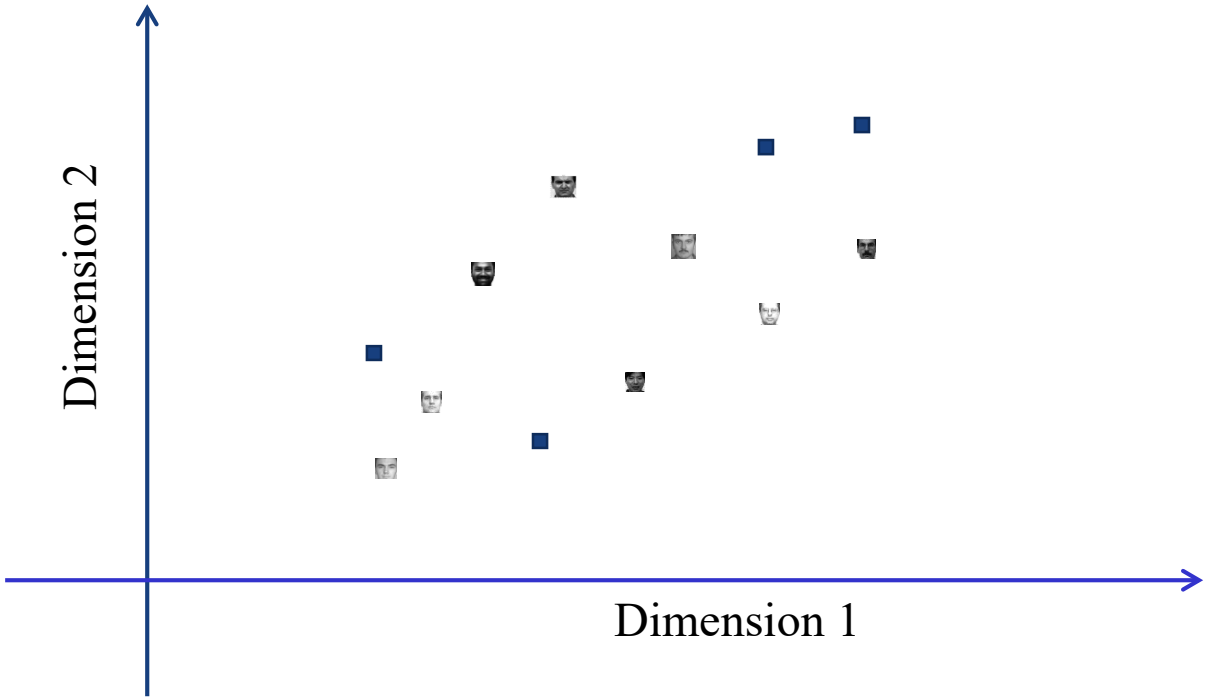
How to define better basis for signal representation

- Given the signal representation as $\mathbf{x} = \sum_i w_i \mathbf{u}_i$
- The ideal is $\hat{\mathbf{x}}_i = w_1\mathbf{u}_1 + w_2\mathbf{u}_2 + w_3\mathbf{u}_3 + \cdots + w_i\mathbf{u}_i$ based on i basis components. Its error is defined as $\text{Error}_i = \|\mathbf{x} - \hat{\mathbf{x}}_i\|^2$
- If the description is terminated at any point, we should still get most of the information about the data, that means $\text{Error}_i < \text{Error}_{i-1}$

Example

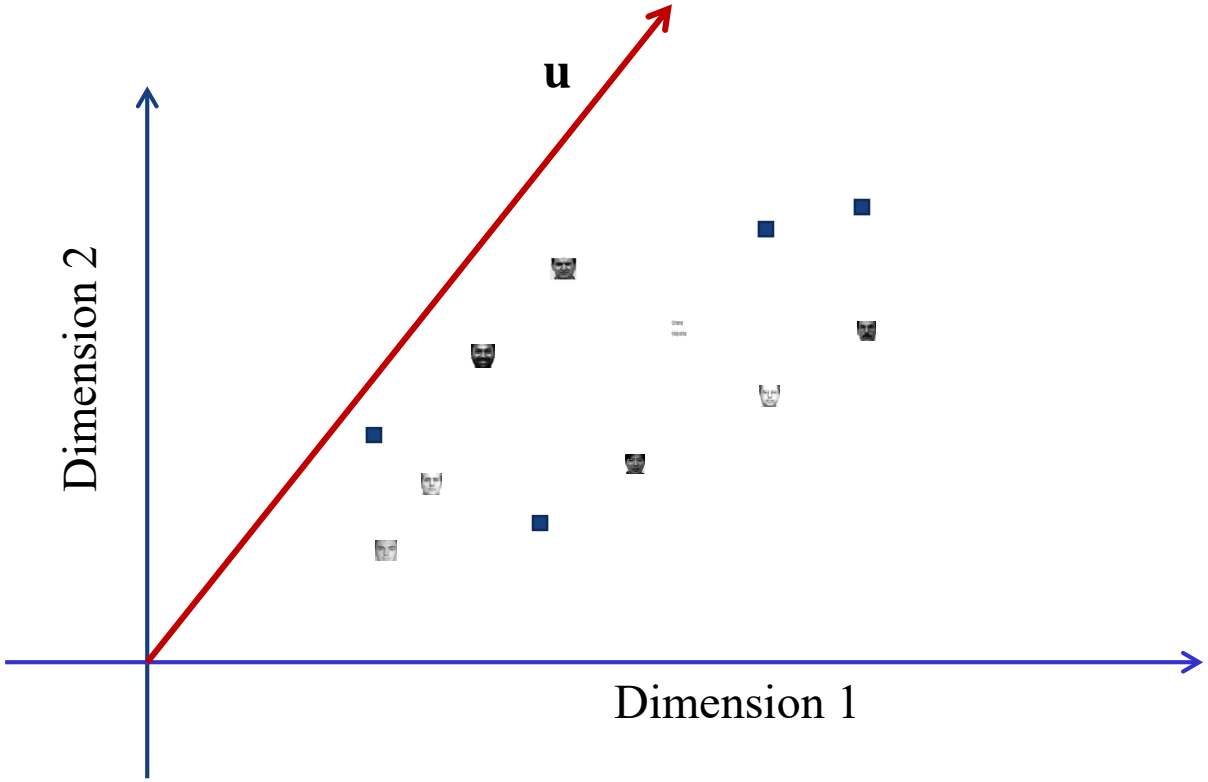
- Assumption: There are a set of K "typical" images that captures most of all image
- Approximate **every** image $\mathbf{x}$ as $\hat{\mathbf{x}} = w_{f,1}\boldsymbol{u}_1 + w_{f,2}\boldsymbol{u}_2 + \cdots + w_{f,K}\boldsymbol{u}_K$
  - $\mathbf{u}_2$ is used to "correct" errors resulting from using only $\mathbf{u}_1$.
  - $\left\|\mathbf{x} - \left(w_{f,1}\boldsymbol{u}_1 + w_{f,2}\boldsymbol{u}_2\right)\right\|^2 < \left\|\mathbf{x} - w_{f,1}\boldsymbol{u}_1\right\|^2$
  - $\mathbf{u}_3$ corrects errors remaining after correction with $\mathbf{u}_2$
  - $\left\|\mathbf{x} - \left(w_{f,1}\boldsymbol{u}_1 + w_{f,2}\boldsymbol{u}_2 + +w_{f,3}\boldsymbol{u}_3\right)\right\|^2 < \left\|\mathbf{x} - \left(w_{f,1}\boldsymbol{u}_1 + w_{f,2}\boldsymbol{u}_2\right)\right\|^2$
  - And so on
  - Totally, $\mathbf{u} = [\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \cdots]$
- Estimate $\mathbf{u}$ to minimize the squared error.

Find the first basis image



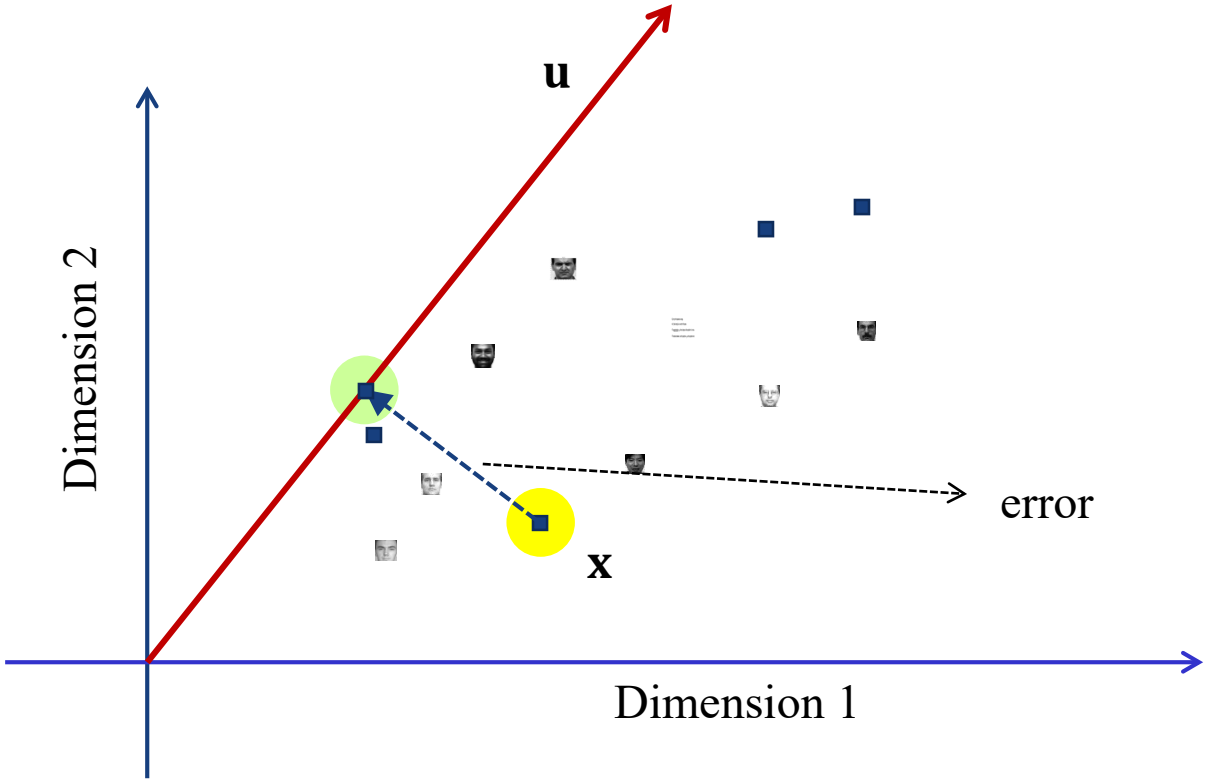
- Each “point” represents an image.

Find the first basis image



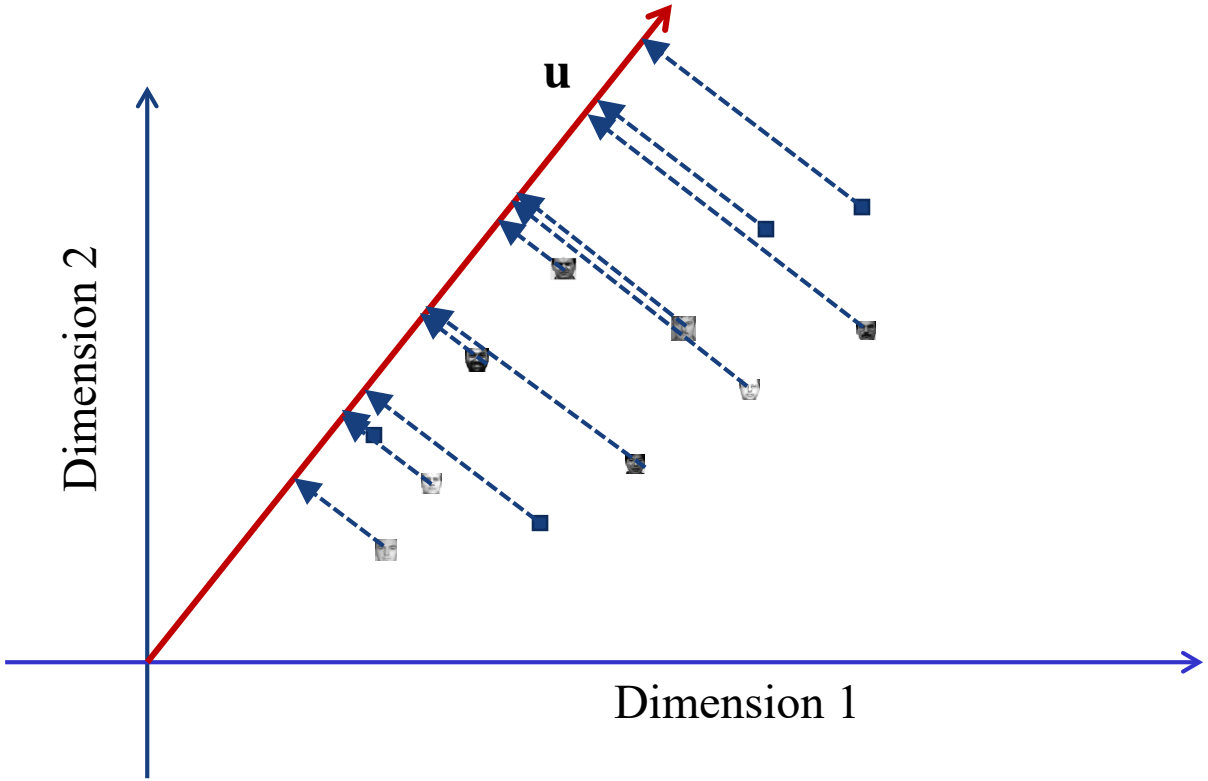
- Each “point” represents an image.
- Any “basis image” **u** is a vector in this space.

Find the first basis image



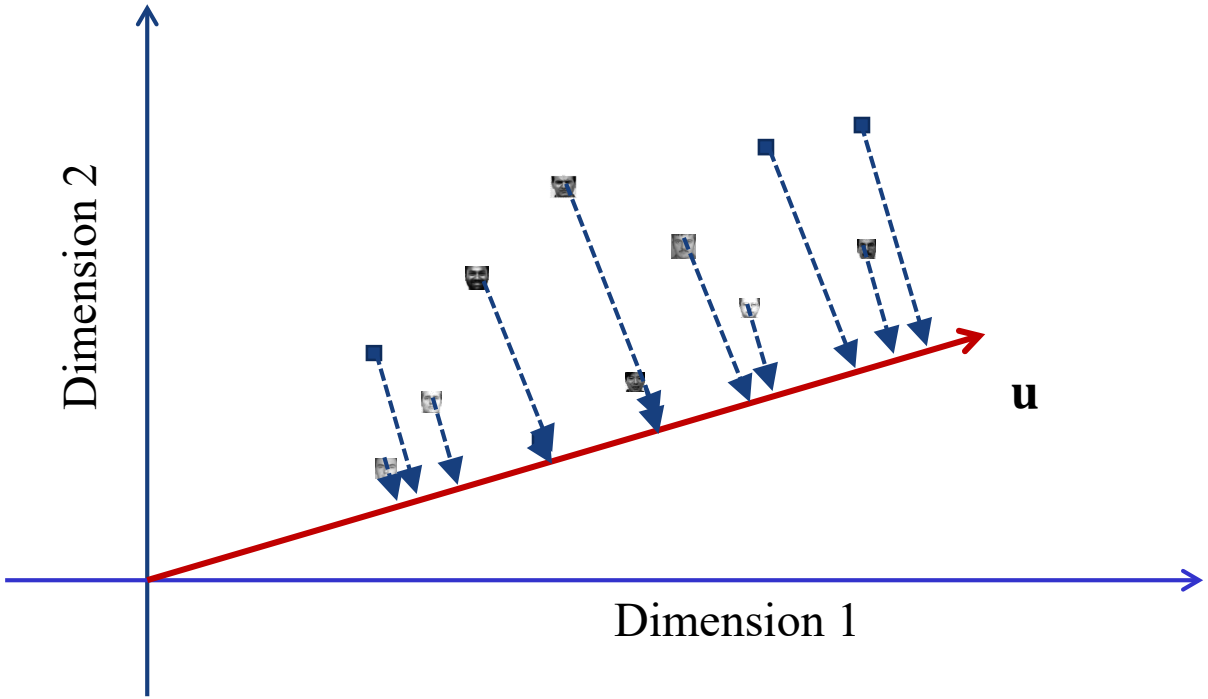
- Each “point” represents an image.
- Any “basis image” $\mathbf{u}$ is a vector in this space.
- The approximation $w_{f,1}\mathbf{u}$ for any image $\mathbf{x}$ is the *projection* of $\mathbf{x}$ onto $\mathbf{u}$.
- The distance between $\mathbf{x}$ and its projection $w_{f,1}\mathbf{u}$ is the *projection error.*

Find the first basis image



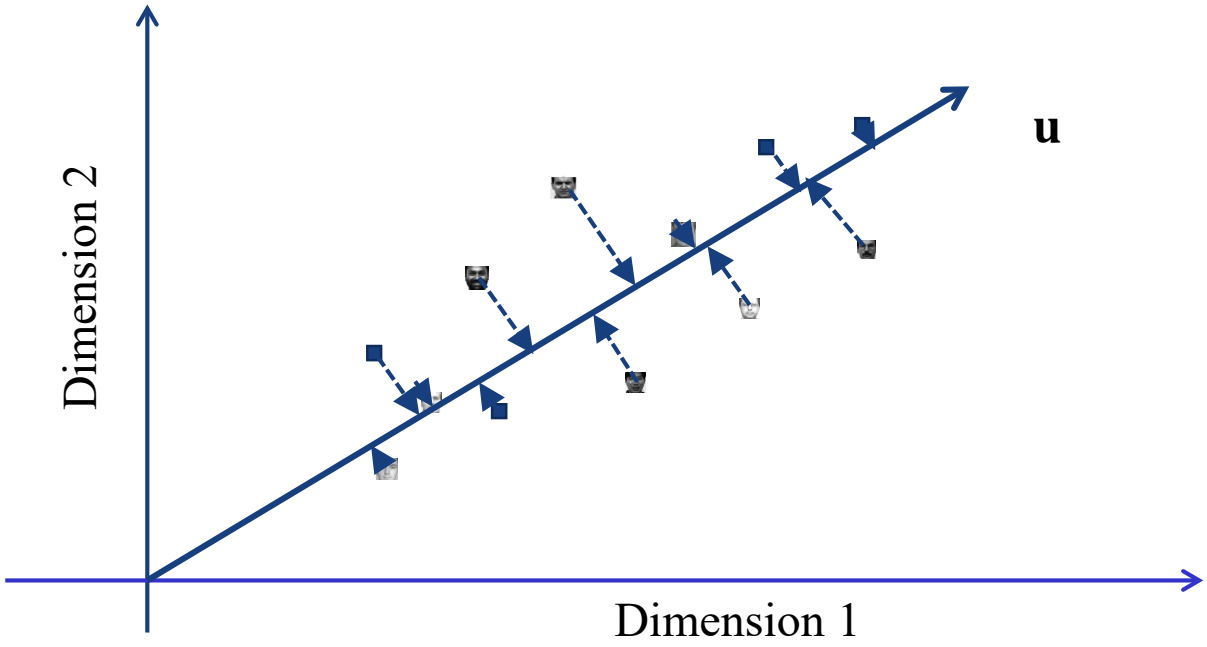
- *Every* image in our data will suffer error when approximated by its projection on **u**
- The total squared length of all error lines is the *total squared projection error.*
- The problem of finding the first basis image: Find the **u** for which the total projection error is minimum!
- This “minimum squared error” **u** is our “best” first typical basis image.

Find the first basis image



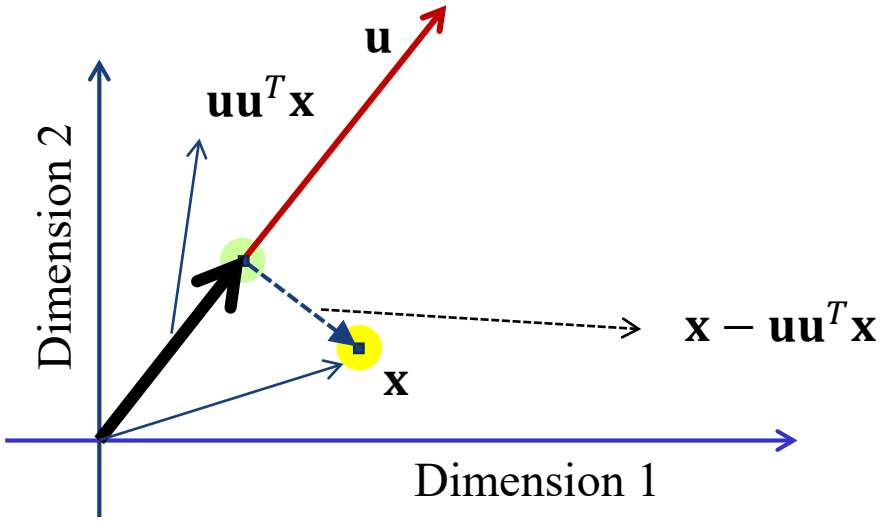
- *Every* image in our data will suffer error when approximated by its projection on **u**
- The total squared length of all error lines is the *total squared projection error.*
- The problem of finding the first basis image: Find the **u** for which the total projection error is minimum!
- This “minimum squared error” **u** is our “best” first typical basis image.

Find the first basis image



- *Every* image in our data will suffer error when approximated by its projection on **u**
- The total squared length of all error lines is the *total squared projection error.*
- The problem of finding the first basis image: Find the **u** for which the total projection error is minimum!
- This “minimum squared error” **u** is our “best” first typical basis image.

Find the first basis image



- Projection of a vector $\mathbf{x}$ on to a vector $\mathbf{u}$, and further assume $\mathbf{u}$ is of unit length ($\|\mathbf{u}\| = \mathbf{1}$), $\hat{\mathbf{x}} = \mathbf{u}\frac{\mathbf{u}^T\mathbf{x}}{\|\mathbf{u}\|} = \mathbf{uu}^T\mathbf{x}$

$$Error = \|\mathbf{x} - \hat{\mathbf{x}}\|^2 = \|\mathbf{x} - \mathbf{uu}^T\mathbf{x}\|^2$$
$$= (\mathbf{x} - \mathbf{uu}^T\mathbf{x})^T(\mathbf{x} - \mathbf{uu}^T\mathbf{x})$$
$$= \mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{uu}^T\mathbf{x} - \mathbf{x}^T\mathbf{uu}^T\mathbf{x} + \mathbf{x}^T\mathbf{uu}^T\mathbf{uu}^T\mathbf{x}$$
$$= \mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{uu}^T\mathbf{x}$$

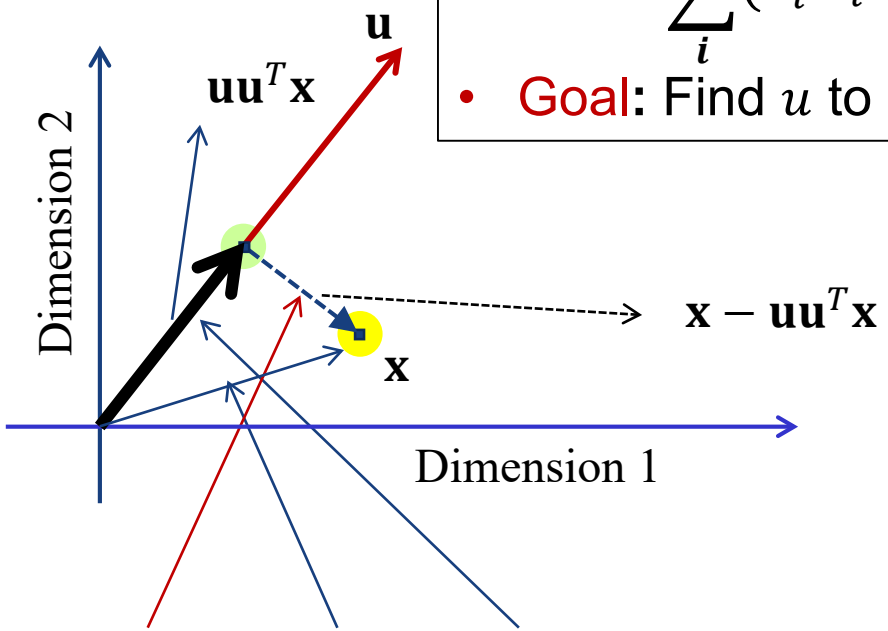
$\mathbf{u}^T\mathbf{u}$ = 1

Find the first basis image

- Error for one signal vector $\mathbf{x}$ $Error = \mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{u}\mathbf{u}^T\mathbf{x}$
- Error for all signal vectors $\mathbf{x}_i$

$$Error = \sum_i \left(\mathbf{x}_i^T\mathbf{x}_i - \mathbf{x}_i^T\mathbf{u}\mathbf{u}^T\mathbf{x}_i\right) = \sum_i \mathbf{x}_i^T\mathbf{x}_i - \sum_i \mathbf{x}_i^T\mathbf{u}\mathbf{u}^T\mathbf{x}_i$$

- Goal: Find u to minimize this error.



$$Error = \mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{u}\mathbf{u}^T\mathbf{x}$$

Find the first basis image

Find $\mathbf{u}$ to minimize the error $\sum_i \mathbf{x}_i^T \mathbf{x}_i - \sum_i \mathbf{x}_i^T \mathbf{u}\mathbf{u}^T \mathbf{x}_i$ subject to $\mathbf{u}^T\mathbf{u} = 1$

Using Lagrange multiplier α and set derivative to be zero

$$C = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \sum_i \mathbf{x}_i^T \mathbf{u}\mathbf{u}^T \mathbf{x}_i + \alpha(\mathbf{u}^T\mathbf{u} - 1)$$

$$-2\sum_i \mathbf{x}_i\mathbf{x}_i^T\mathbf{u} + 2\alpha\mathbf{u} = 0 \Rightarrow \left(\sum_i \mathbf{x}_i\mathbf{x}_i^T\right)\mathbf{u} = \alpha\mathbf{u}$$

- $\mathbf{u}$: eigenvector of matrix $\sum_i \mathbf{x}_i\mathbf{x}_i^T$
- α is eigenvalue

Minimize $C = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \sum_i \mathbf{x}_i^T \mathbf{u}\mathbf{u}^T \mathbf{x}_i$	Recall $\mathbf{x}_i^T\mathbf{u} = \mathbf{u}^T\mathbf{x}_i$
$= \sum_i \mathbf{x}_i^T \mathbf{x}_i - \sum_i \mathbf{u}^T \mathbf{x}_i\mathbf{x}_i^T\mathbf{u}$	
$= \sum_i \mathbf{x}_i^T \mathbf{x}_i - \mathbf{u}^T\left(\sum_i \mathbf{x}_i\mathbf{x}_i^T\right)\mathbf{u} = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \mathbf{u}^T\alpha\mathbf{u}$	Recall $\left(\sum_i \mathbf{x}_i\mathbf{x}_i^T\right)\mathbf{u} = \alpha\mathbf{u}$
$= \sum_i \mathbf{x}_i^T \mathbf{x}_i - \alpha\mathbf{u}^T\mathbf{u}$	Recall $\mathbf{u}^T\mathbf{u} = 1$
$= \sum_i \mathbf{x}_i^T \mathbf{x}_i - \alpha$	Choose the largest α to minimize this error

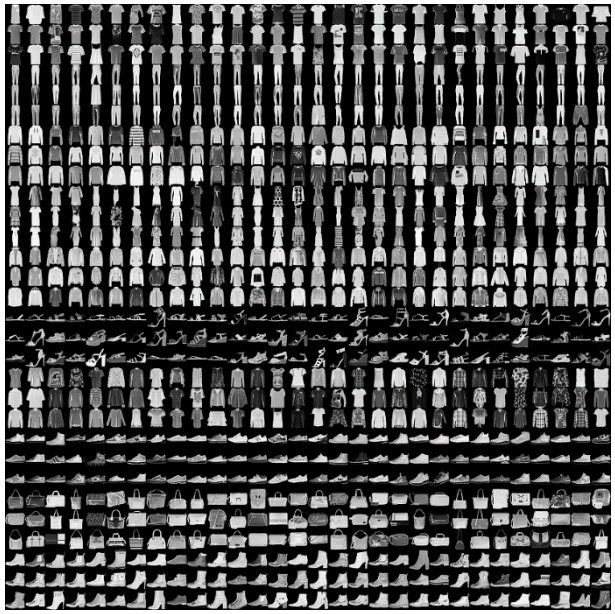
To find more basis

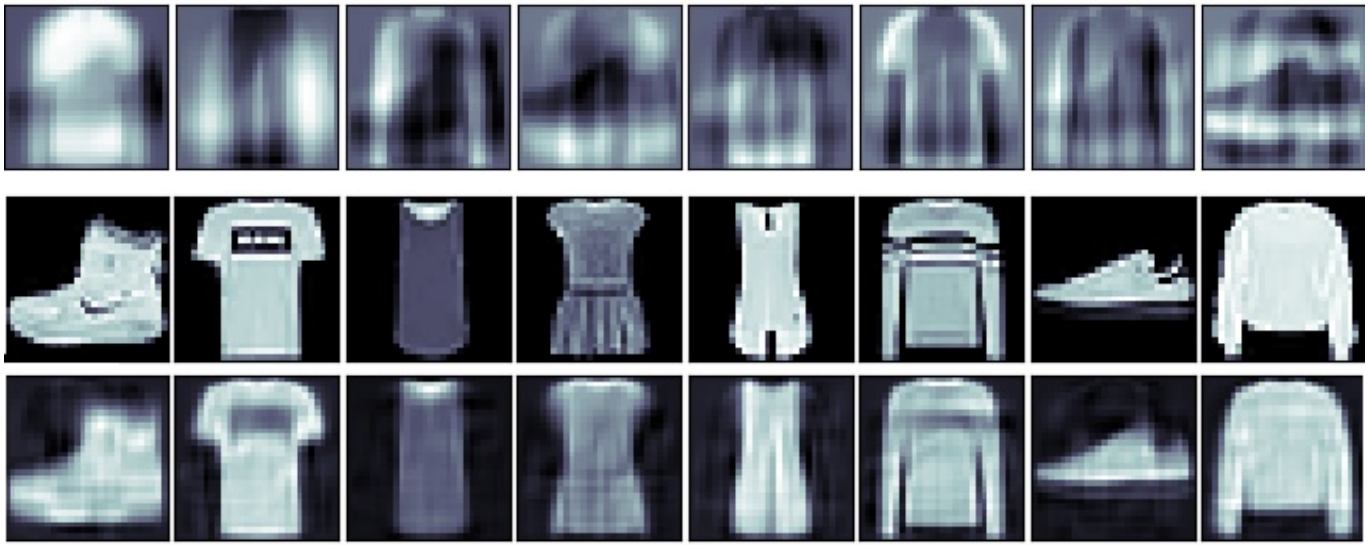
So far we already have the first basis, to get more basis, we need to

- Get the “error” signal by subtracting the first-level approximation from the original signal.
- Repeat the estimation on the “error” signal.
- Get the second-level “error” faces by subtracting the scaled second typical basis from the first-level error.
- Repeat the estimation on the second-level “error” signal.
- We can continue the process, refining the error each time.

Example

Fashion-MNIST dataset,
https://github.com/zalandoresearch/fashion-mnist





Examples of basis images

Original images

Reconstructed images (with 64 basis)

$$\hat{\mathbf{x}} = \sum_{i=1}^{64} w_i \mathbf{u}_i$$

Sparse signal representation

Suppose we represent the signal $\mathbf{x}$ using K basis as $\mathbf{x} = \sum_{i=1}^{K} w_i \mathbf{u}_i$

Rewrite it into matrix form as

Input data →

$$\mathbf{x} = \underbrace{[\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_K]}_{K} \left.\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \end{bmatrix}\right\} K = \mathbf{UW}$$

Basis vectors

Unknowns

- When #(Basis vectors) = dim(Input data), we have a unique solution
- When #(Basis vectors) < dim(Input data), we may have no solution
- When #(Basis vectors) > dim(Input data), we have infinitely many solutions

Sparse signal representation

Sparse representations are representations that account for most or all information of a signal with a linear combination of a small number of atoms.
Objective: We want to use as few basis vectors as possible to represent signal.

$$\min \|\mathbf{W}\|_0 \quad s.t. \mathbf{x} = \mathbf{UW}$$

← Counts the number of non-zero elements in $\mathbf{W}$

How to solve it?

Matching pursuit (greedy algorithm)

- Finds a basis vector in the dictionary that best matches the input signal
- Remove the weighted value of this basis vector from the signal
- Again, find a basis vector in the dictionary that best matches the remaining signal.
- Continue till a defined stop condition is satisfied.

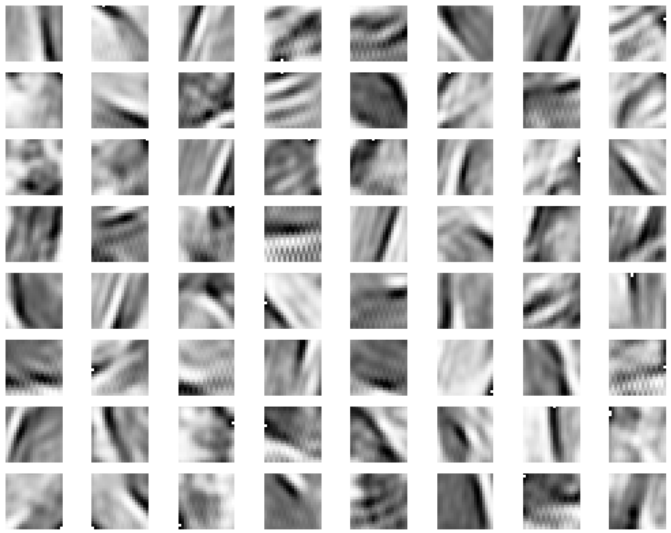
Basis pursuit

- Change the optimization problem as $\min(\|\mathbf{x} - \mathbf{UW}\|^2 + \lambda\|\mathbf{W}\|_1)$, where λ is a penalty term on the non-zero elements and promotes sparsity.

Sparse signal representation

- Model image patches with $m \times m$ pixels.
- Suppose the basis dictionary has 64 vectors.
- Every patch can be represented as a linear combination of a few vectors.
- Sparsity and redundancy.

Input image



Dictionary of basis vectors

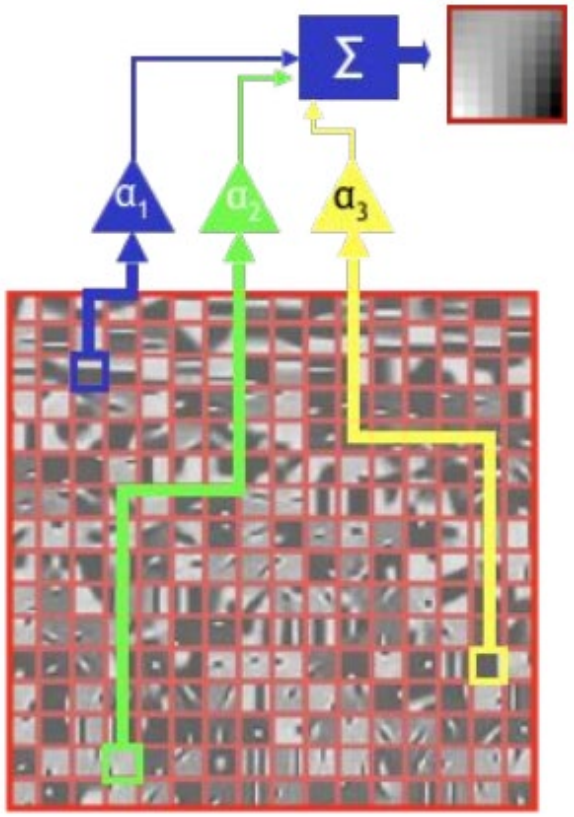


Image patch representation

Sparse signal representation

Comparison between sparse representation and principal component analysis (PCA)

Quoted from Andrew Ng's tutorial, http://ufldl.stanford.edu/tutorial/unsupervised/SparseCoding/

Sparse coding is a class of unsupervised methods for learning sets of over-complete bases to represent data efficiently. The aim of sparse coding is to find a set of basis vectors ϕ_i such that we can represent an input vector $\mathbf{x}$ as a linear combination of these basis vectors:

$$\mathbf{x} = \sum_{i=1}^{k} a_i \phi_i$$

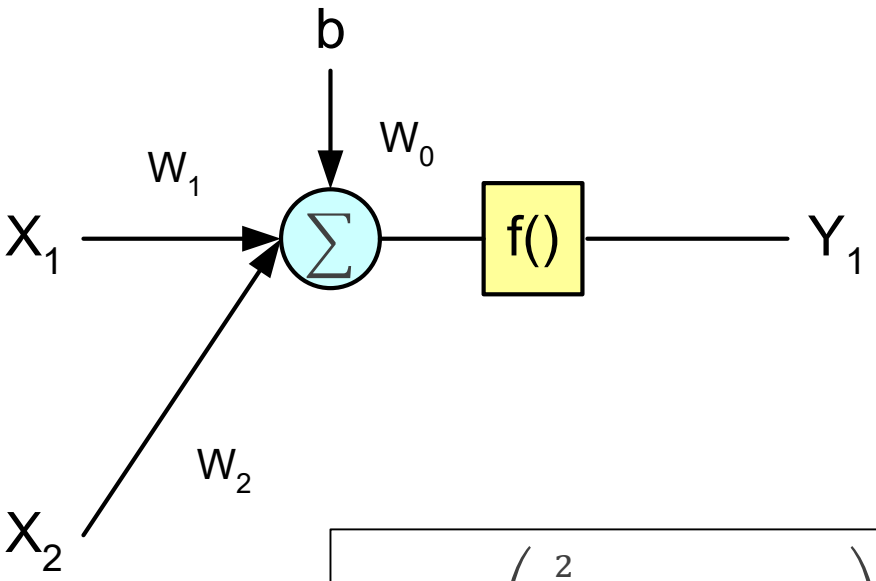
While techniques such as Principal Component Analysis (PCA) allow us to learn a complete set of basis vectors efficiently, we wish to learn an **over-complete** set of basis vectors to represent input vectors $\mathbf{x} \in \mathbb{R}^n$ (i.e. such that $k > n$). The advantage of having an over-complete basis is that our basis vectors are better able to capture structures and patterns inherent in the input data. However, with an over-complete basis, the coefficients a_i are no longer uniquely determined by the input vector $\mathbf{x}$. Therefore, in sparse coding, we introduce the additional criterion of **sparsity** to resolve the degeneracy introduced by over-completeness.

Topics

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

Recall: Neural network

Note: A **bias** is similar to intercept in regression model, it increases flexibility of the model to fit the data.
Reference: https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks

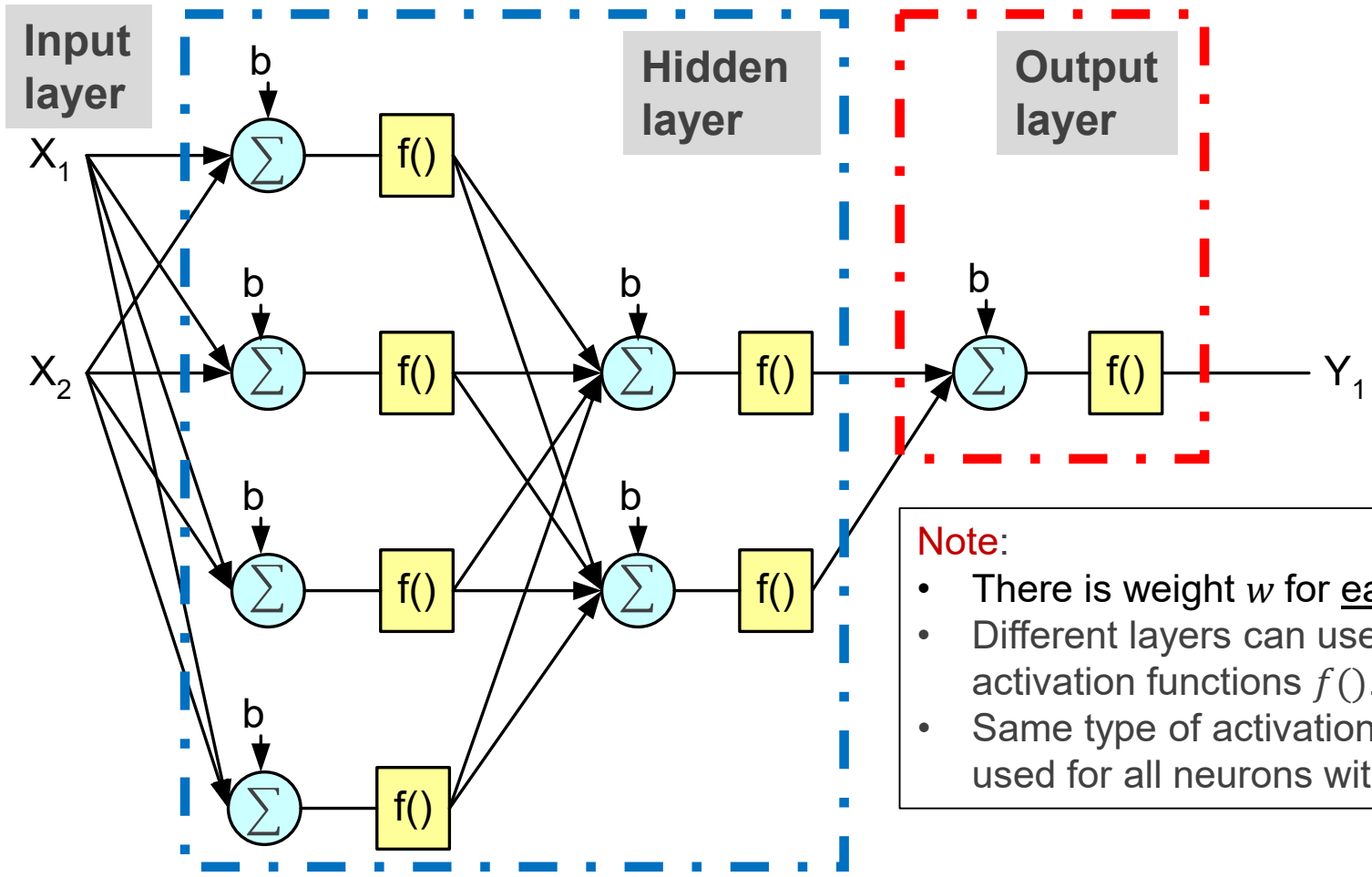


$$Y_1 = f\left(\sum_{i=1}^{2} W_i X_i + W_0 b\right)$$

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
A training data record: X_1, X_2, Y_1	

No. of Input	No. of Hidden		No. of Output
2	Layer	0	1
	Node at each hidden layer	0	

Recall: Neural network

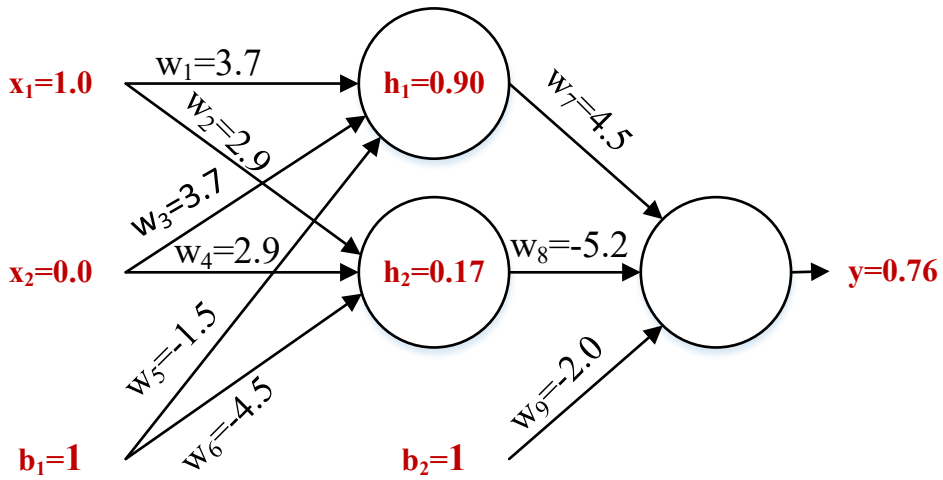


Note:

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

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

Example



- 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

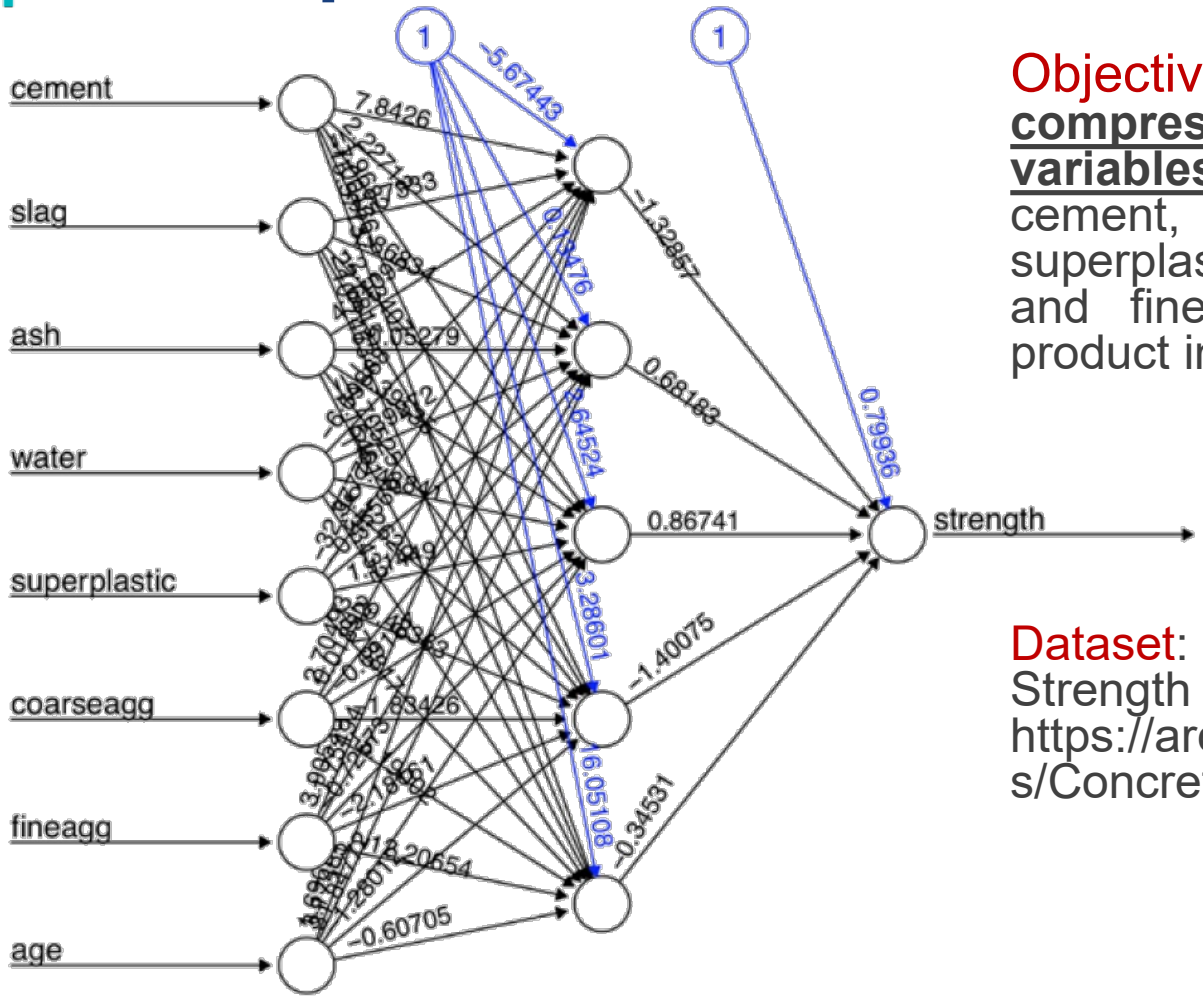
$$h_1 = \text{sigmoid}\big(1.0 \times 3.7 + 0.0 \times 3.7 + 1 \times (-1.5)\big) = \text{sigmoid}(2.2) = \frac{1}{1+e^{-2.2}} = 0.90$$

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

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

No. of Input	No. of Hidden		No. of Output
2	Layer	1	1
	Node at each hidden layer	2	

Example



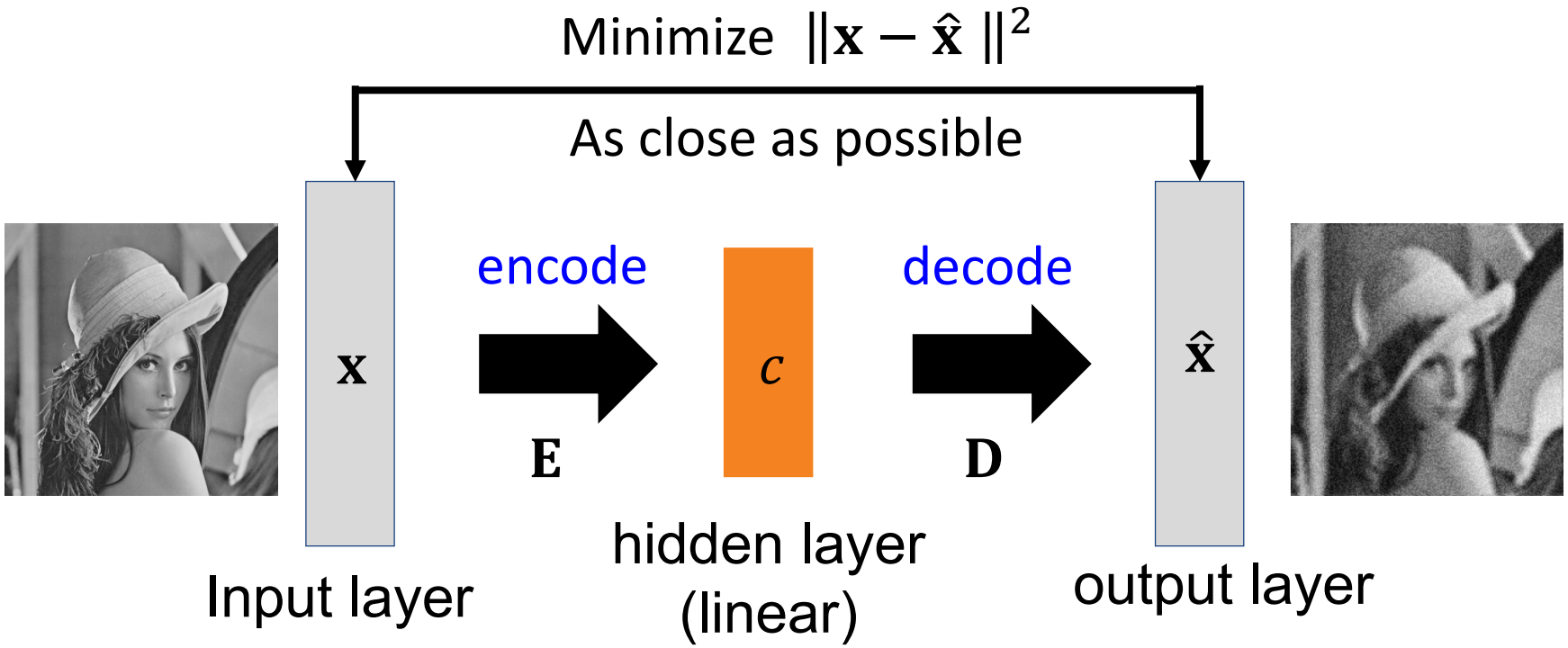
Objective: **Predict concrete compressive strength from input variables**, including the amount of cement, slag, ash, water, superplasticizer, coarse aggregate, and fine aggregate used in the product in addition to the aging time.

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
	Node at each hidden layer	5	

Signal representation learning

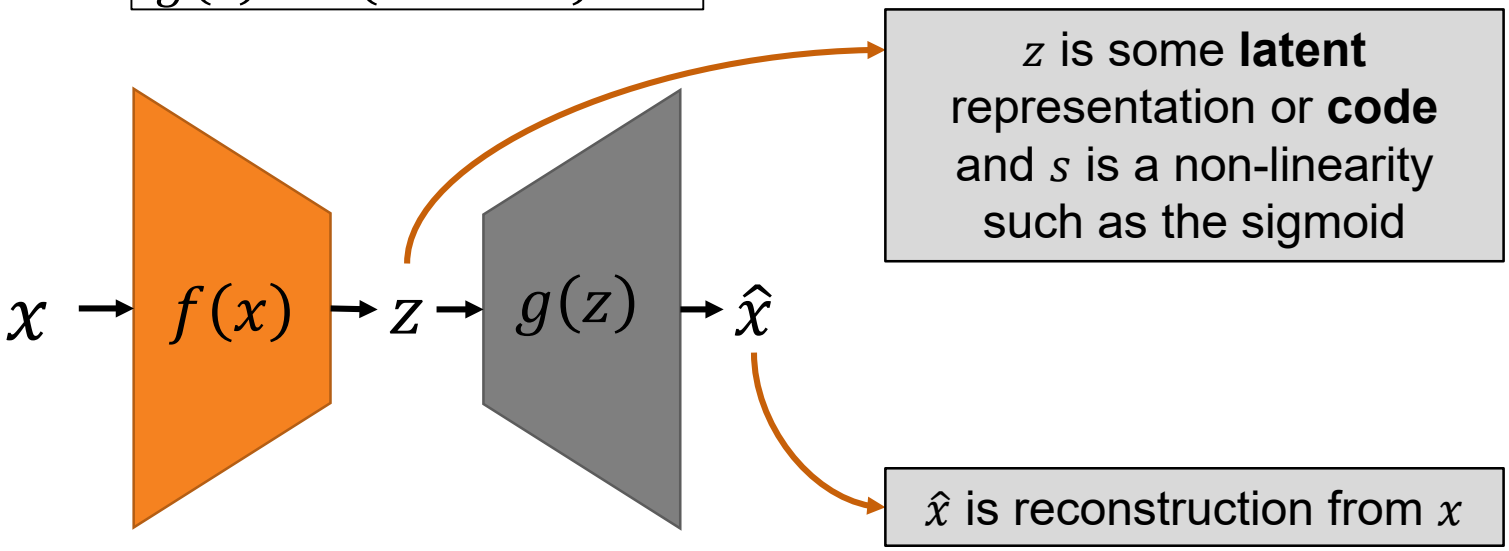
- Can we learn signal representation as neural network?



Signal representation learning

- Idea: Given data x (no labels) we would like to learn the functions f (encoder) and g (decoder) to minimize the reconstruction error between x and $\hat{x}$

$$f(x) = s(wx + b) = z$$
$$g(z) = s(w'z + b') = \hat{x}$$



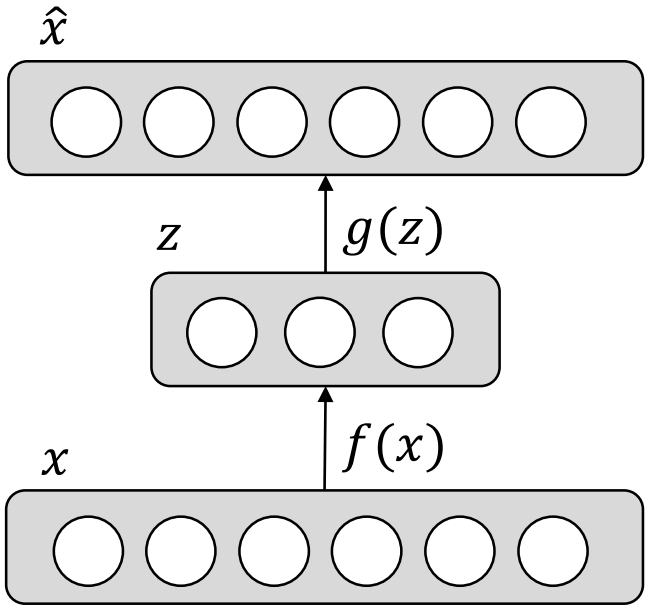
Signal representation: Undercomplete and overcomplete

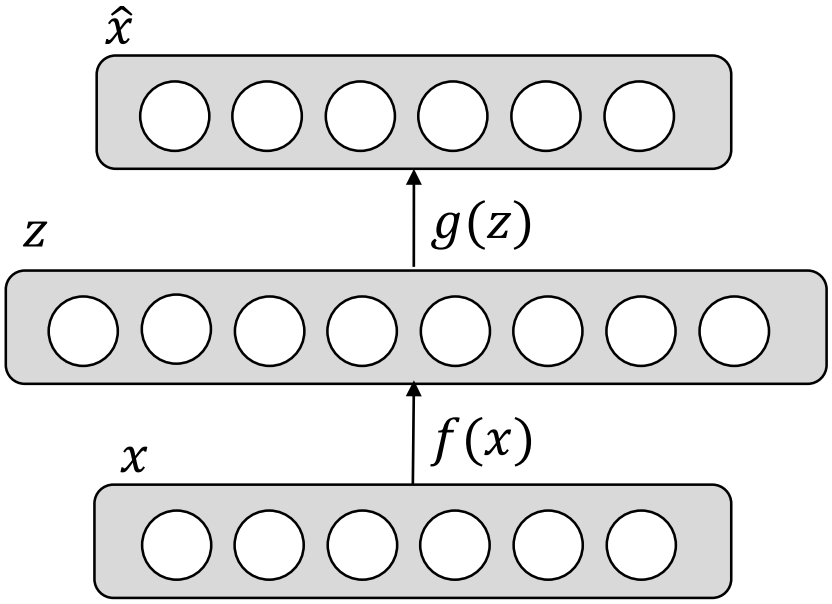
Undercomplete

- Hidden layer is smaller than the input layer
- Compresses the input
- Compresses well only for the training dist.
- Hidden nodes will be good features for the training distribution.

Overcomplete

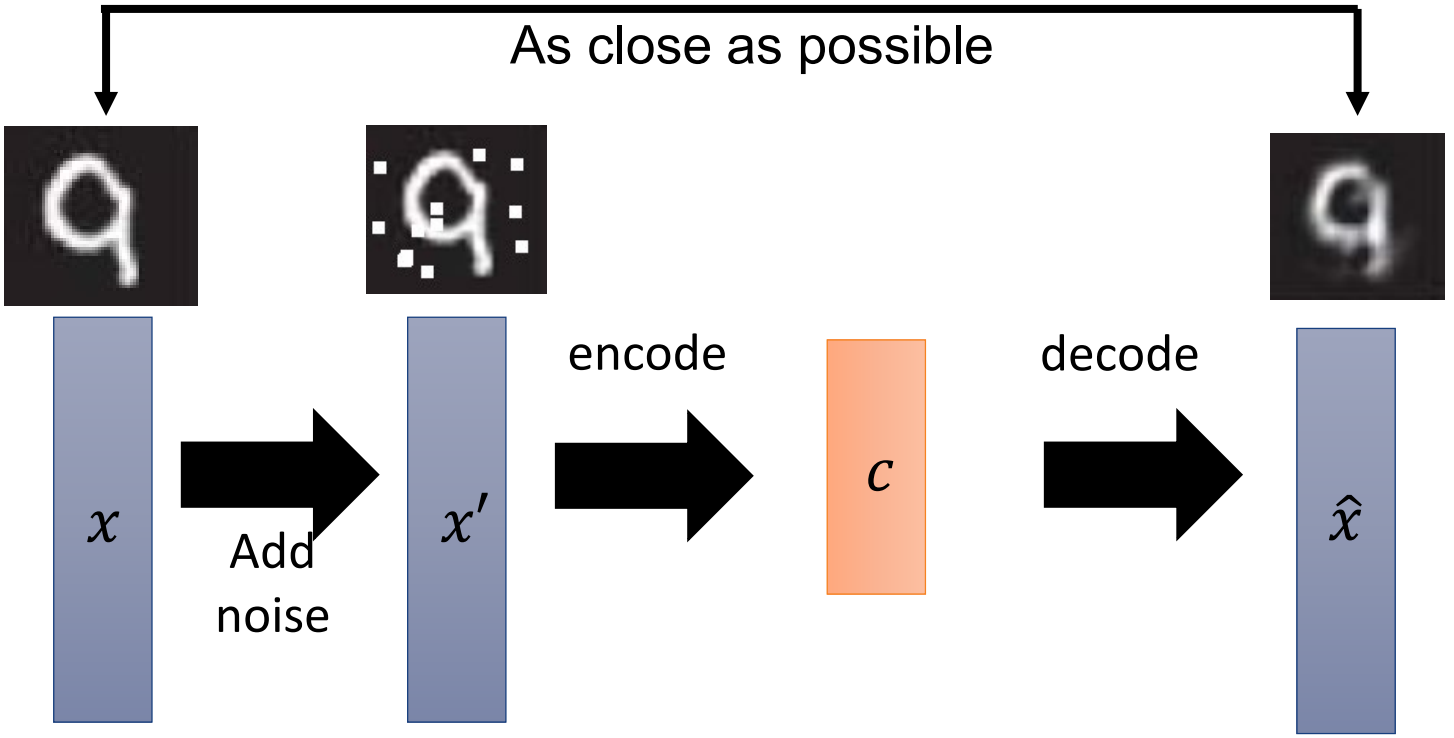
- Hidden layer is greater than the input layer
  - No compression in hidden layer.
  - No guarantee that the hidden units will extract meaningful structure.
  - A higher dimension code helps model a more complex distribution.





Signal representation learning

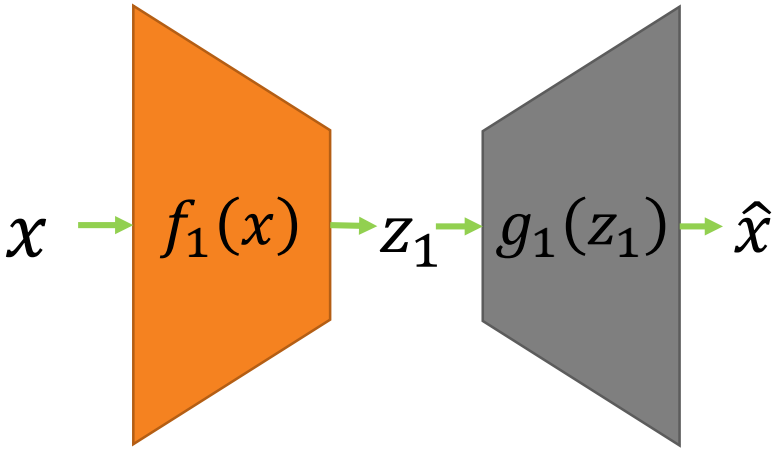
- De-noising auto-encoder: Corrupts input data by injecting Gaussian noise



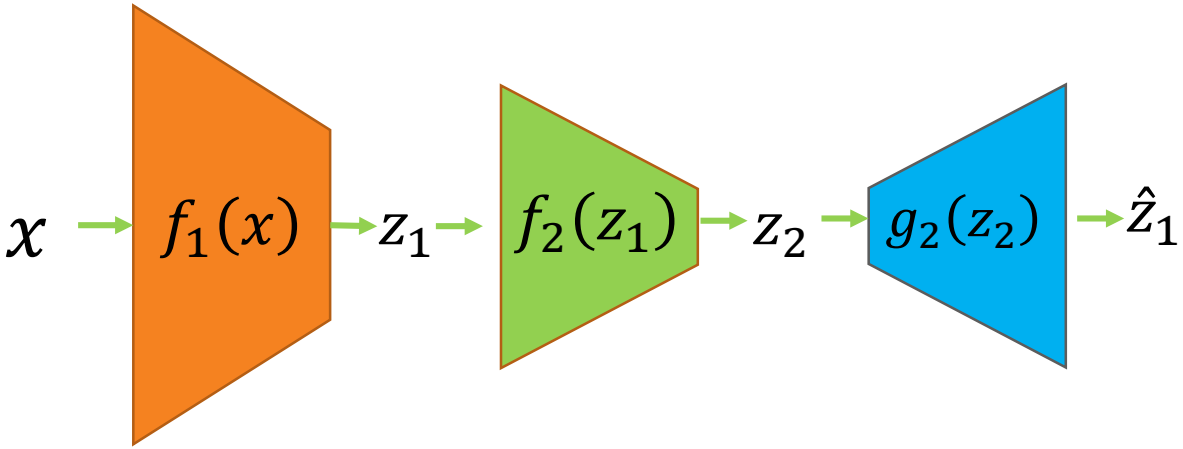
P. Vincent, H. Larochelle, Y. Bengio, P. Manzagol, "Extracting and composing robust features with denoising autoencoders," *Int. Conf. on Machine Learning*, Helsinki, Finland, Jul. 2008, pp. 1096-1103.

Stacked signal representation learning

First layer training



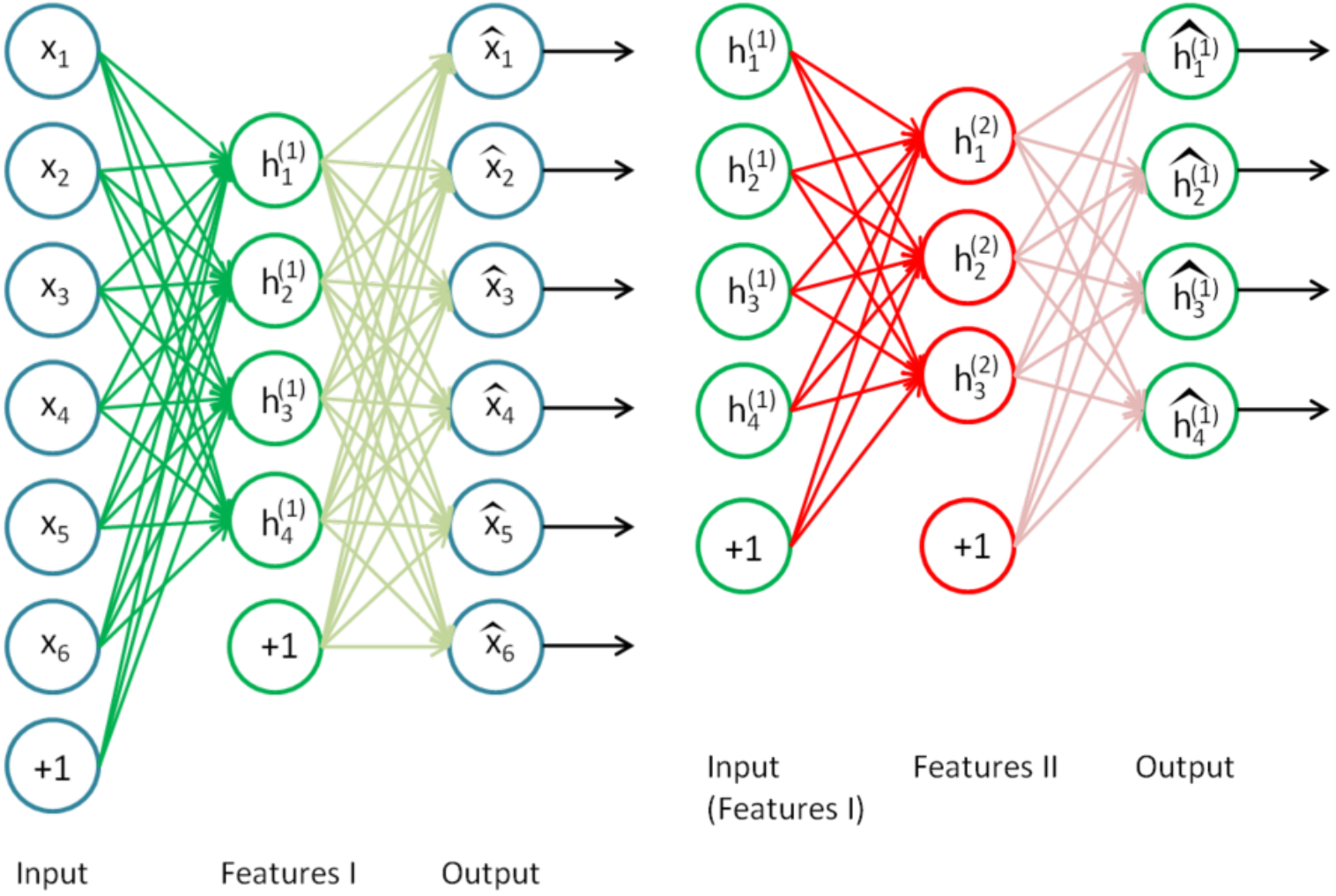
Second layer training



Layer-wise training of stacked autoencoders:

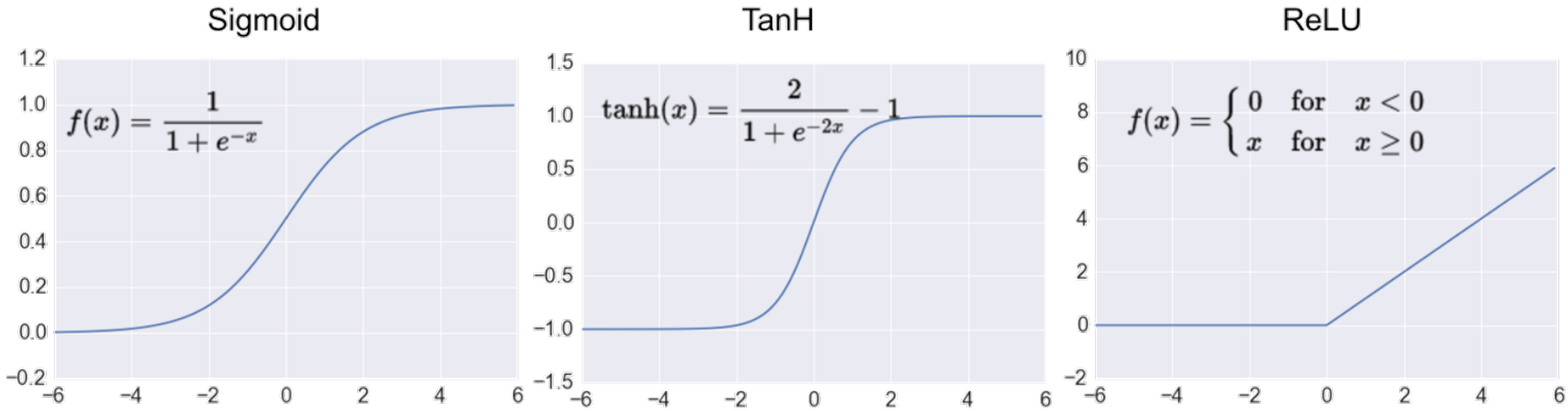
1. Train the bottom-most autoencoder.
2. After training, remove the decoder layer, construct a new autoencoder by taking the *latent representation* of the previous autoencoder as input.
3. Train the new autoencoder. Note the weights and bias of the encoder from the previously trained autoencoders are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

Stacked signal representation learning



Activation function

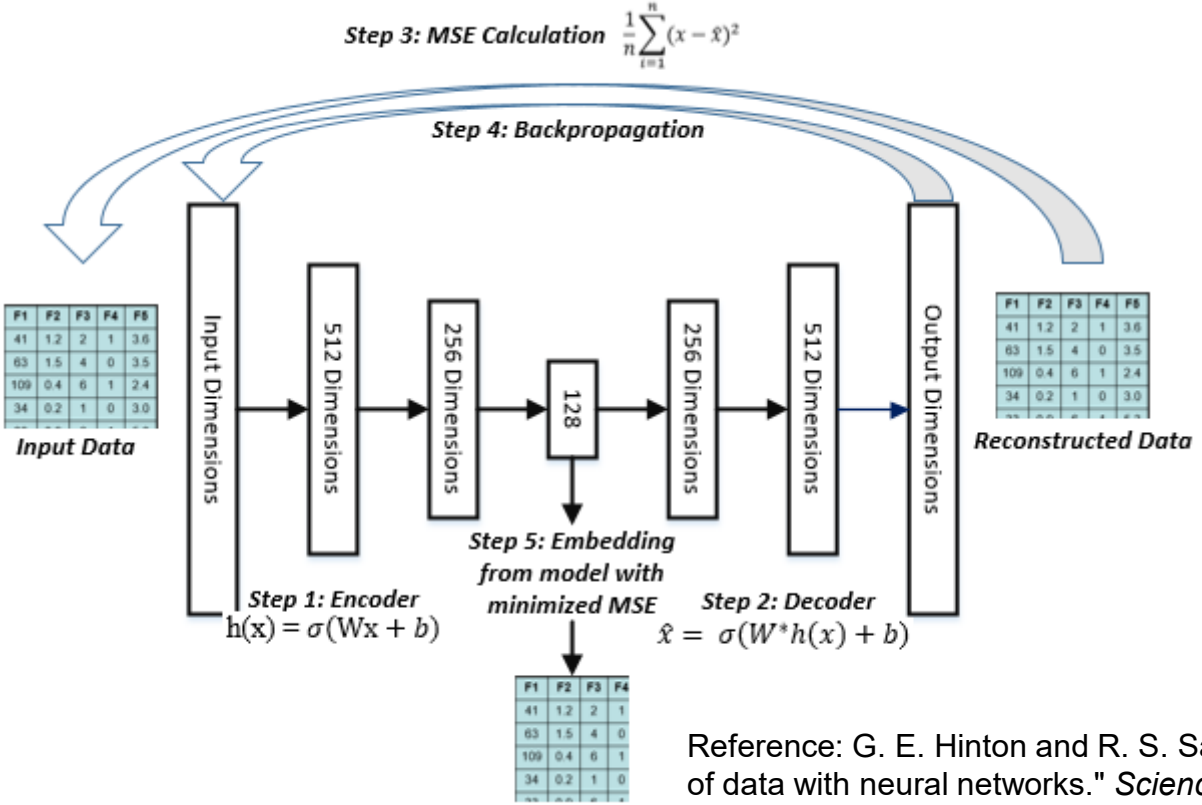
- *Sigmoid* functions generally work better in the case of classifiers. *Sigmoid* and *Tanh* functions are sometimes avoided due to the vanishing gradient problem.
- *ReLU* function is a general activation function and is used in most cases these days.
- As a rule of thumb, you can begin with using *ReLU* function and then move over to other activation functions in case *ReLU* doesn't provide with optimum results.



Reference: https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/

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.



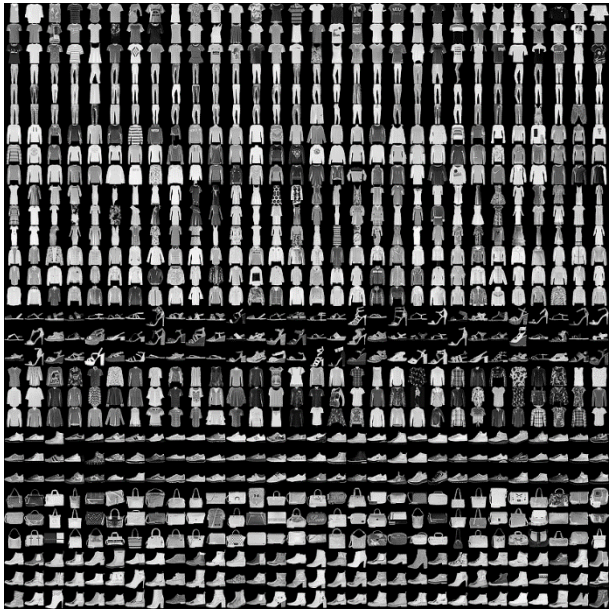
Reference: G. E. Hinton and R. S. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *Science* 313.5786 (2006): 504-507

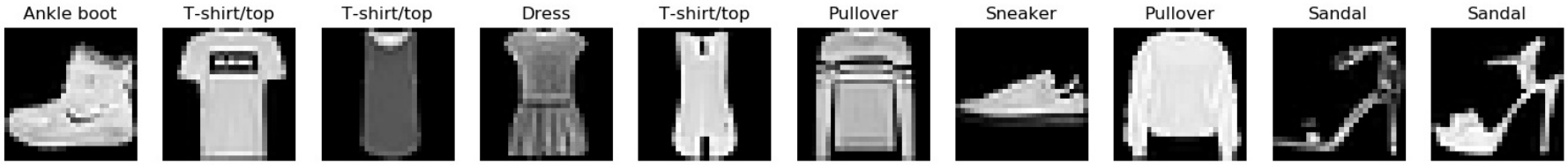
Topics

- 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

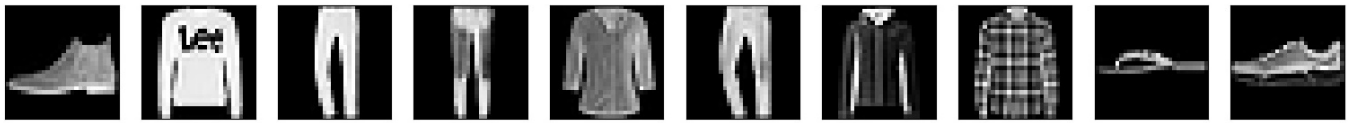
```
# input placeholder
input_image = Input(shape=(ENCODING_DIM_INPUT, ))

# 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

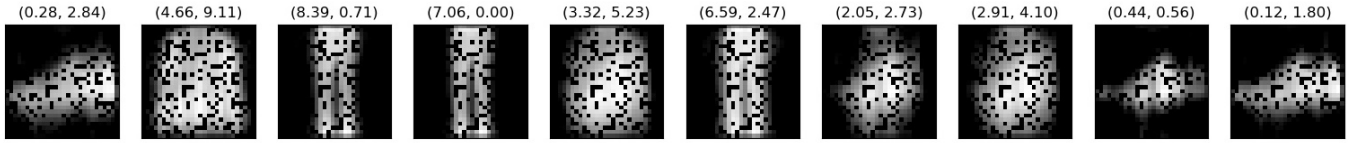
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 784)	0
dense_1 (Dense)	(None, 2)	1570
dense_2 (Dense)	(None, 784)	2352

Total params: 3,922
Trainable params: 3,922
Non-trainable params: 0

Original image

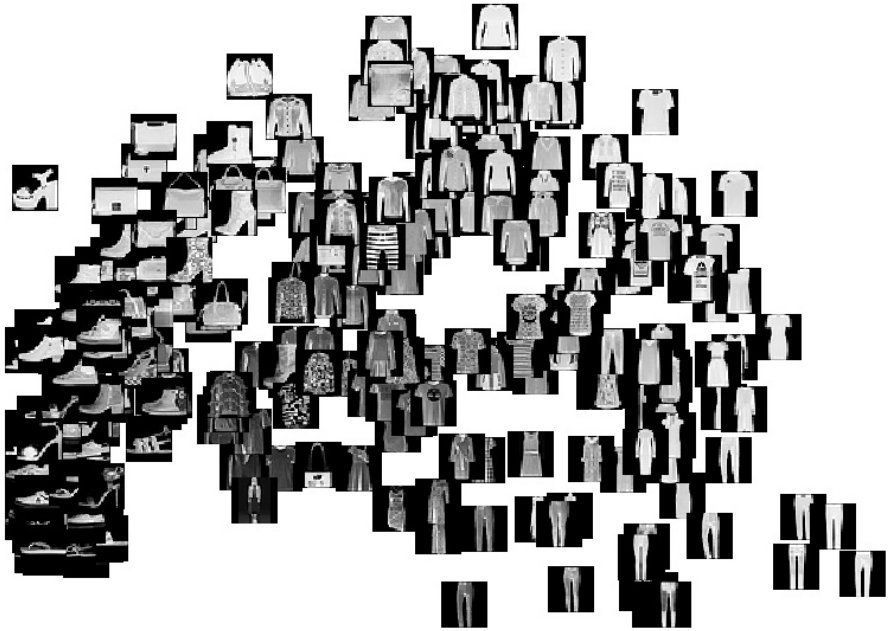


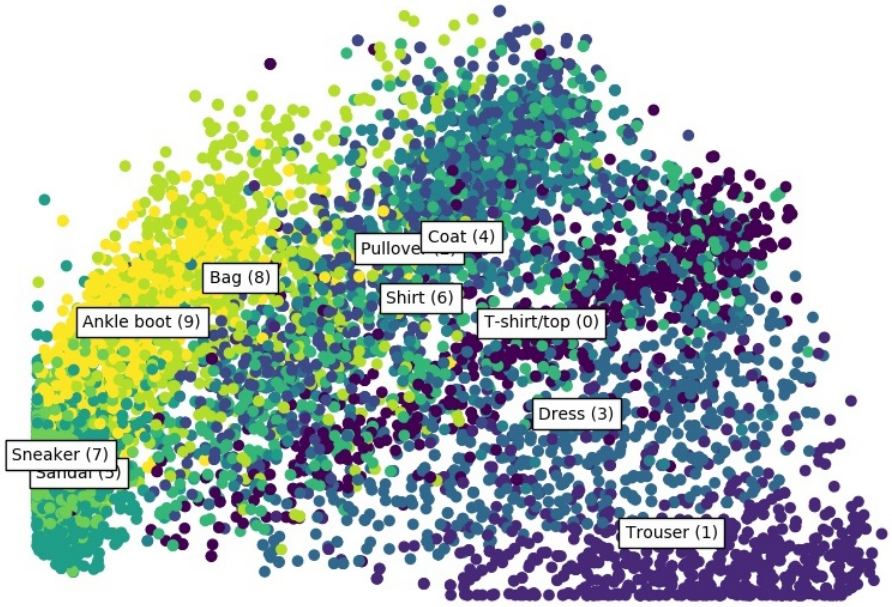
Reconstructed image and code



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

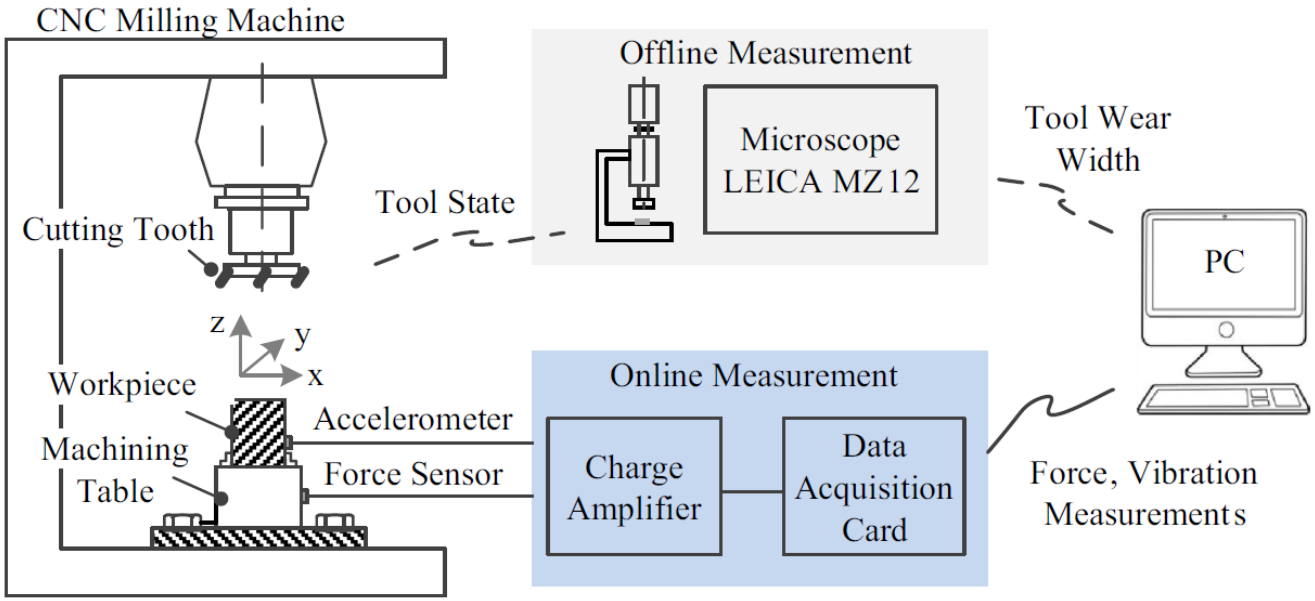
Reference: https://www.datacamp.com/community/tutorials/autoencoder-classifier-python

Case 3: 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

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 architecture

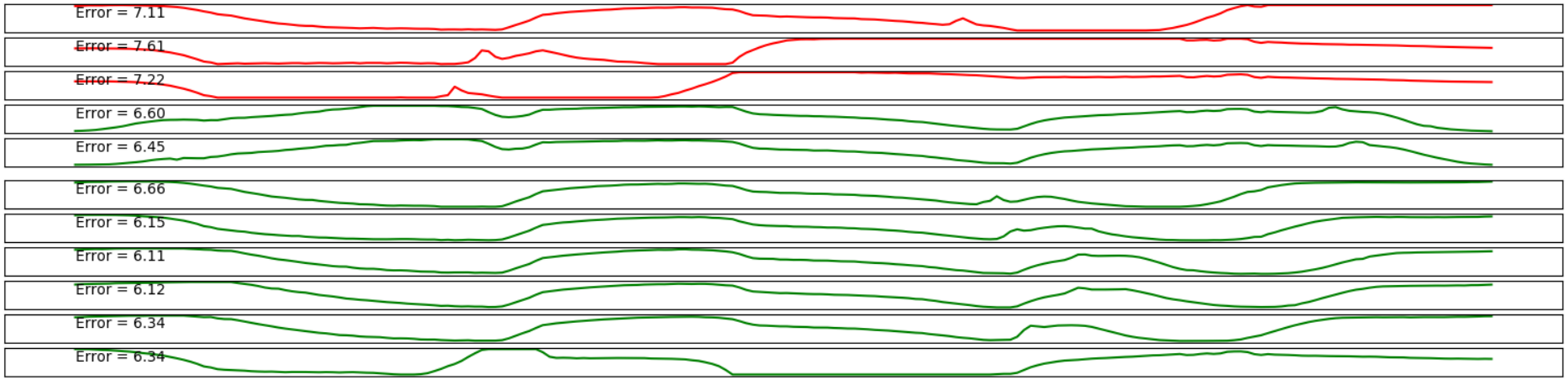
Model

```
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

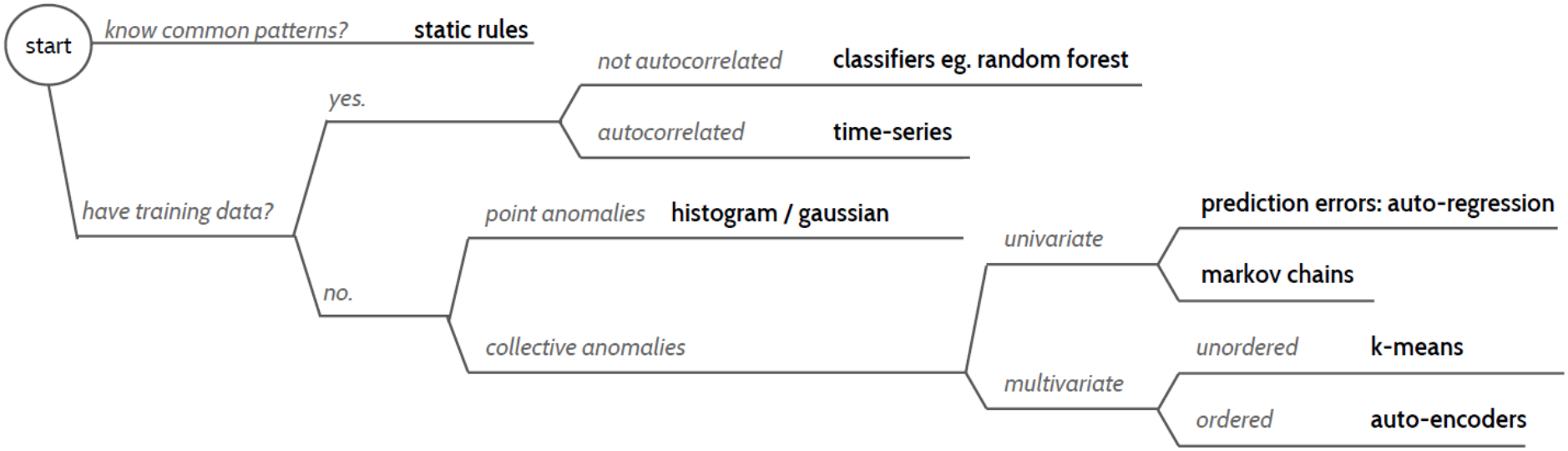
Case 4: Anomaly detection

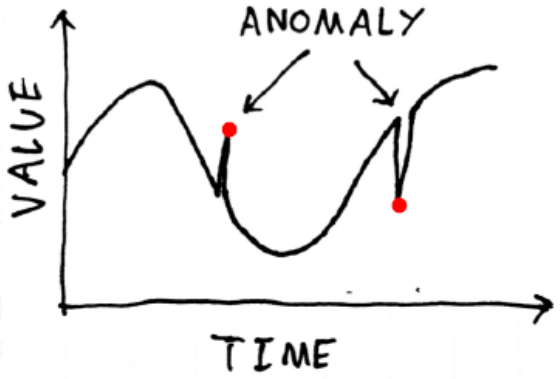
- Supervised
  - Requires labeled anomaly data
- Unsupervised
  - Train an auto-encoder on the training data.
  - Evaluate it on the validation data and the reconstructed error plot.
  - Choose a threshold, which determines whether a value is an outlier (anomalies) or not. This threshold can by dynamic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

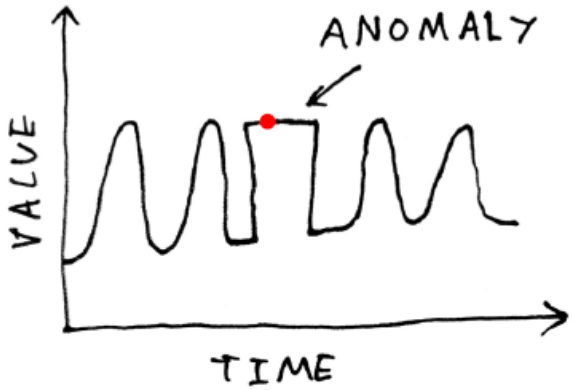


Reference: https://github.com/chen0040/keras-anomaly-detection

Case 4: Anomaly detection



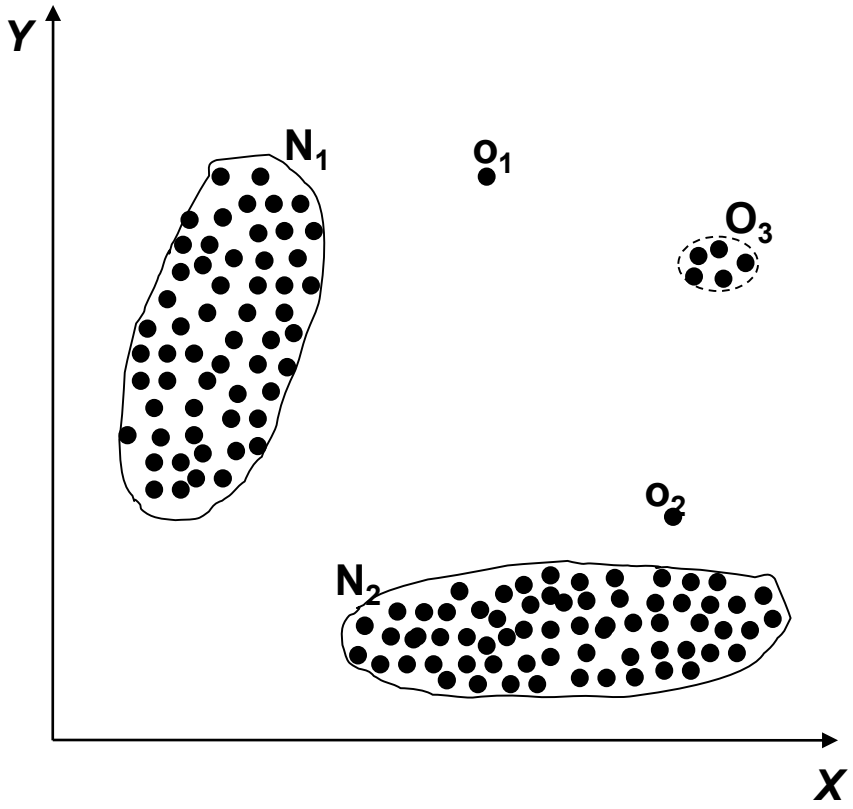




Reference: https://www.aisingapore.org/forums/forum-ai-meetups/the-science-of-anomaly-detection-meetup-slides/

Point anomaly

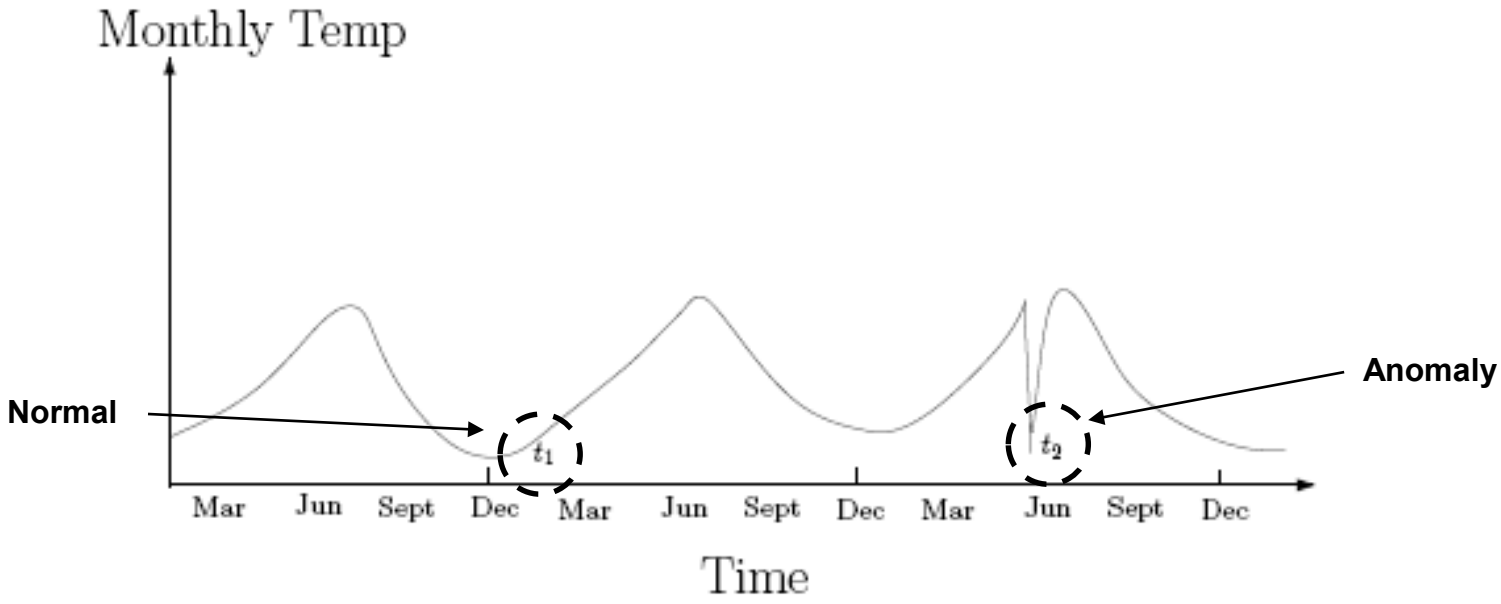
- An individual data instance is anomalous with respect to the data



Reference: Jeaf Howbert, Anomaly Detection, http://courses.washington.edu/css581/lecture_slides/18_anomaly_detection.pdf

Contextual anomaly

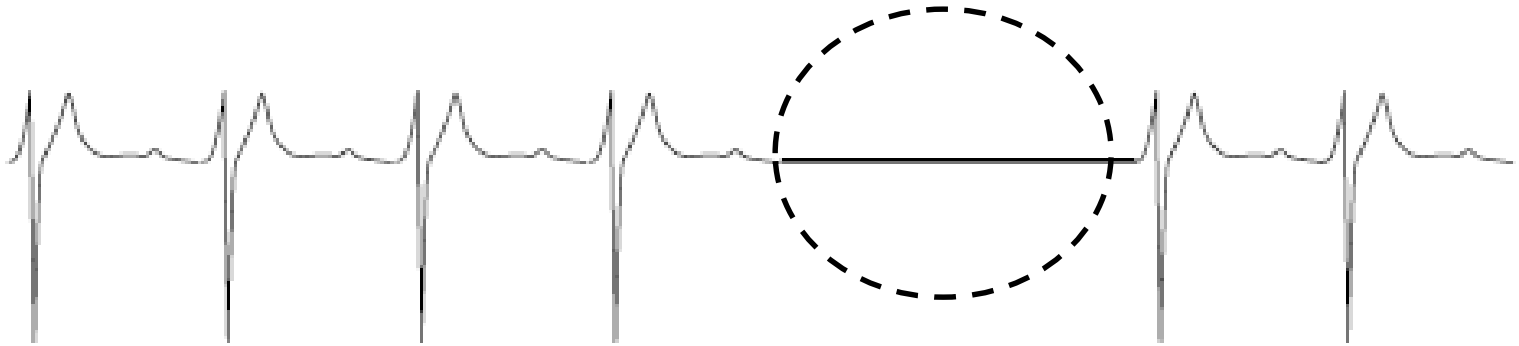
- An individual data instance is anomalous within a context
- Requires a notion of context
- Also referred to as conditional anomalies



Reference: Jeaf Howbert, Anomaly Detection, http://courses.washington.edu/css581/lecture_slides/18_anomaly_detection.pdf

Collective anomalies

- A collection of related data instances is anomalous
- Requires a relationship among data instances
  - Sequential data
  - Spatial data
  - Graph data
- The individual instances within a collective anomaly are not anomalous by themselves



Reference: Jeaf Howbert, Anomaly Detection, http://courses.washington.edu/css581/lecture_slides/18_anomaly_detection.pdf

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
- Semi-supervised anomaly detection
  - Labels available only for normal data
- 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

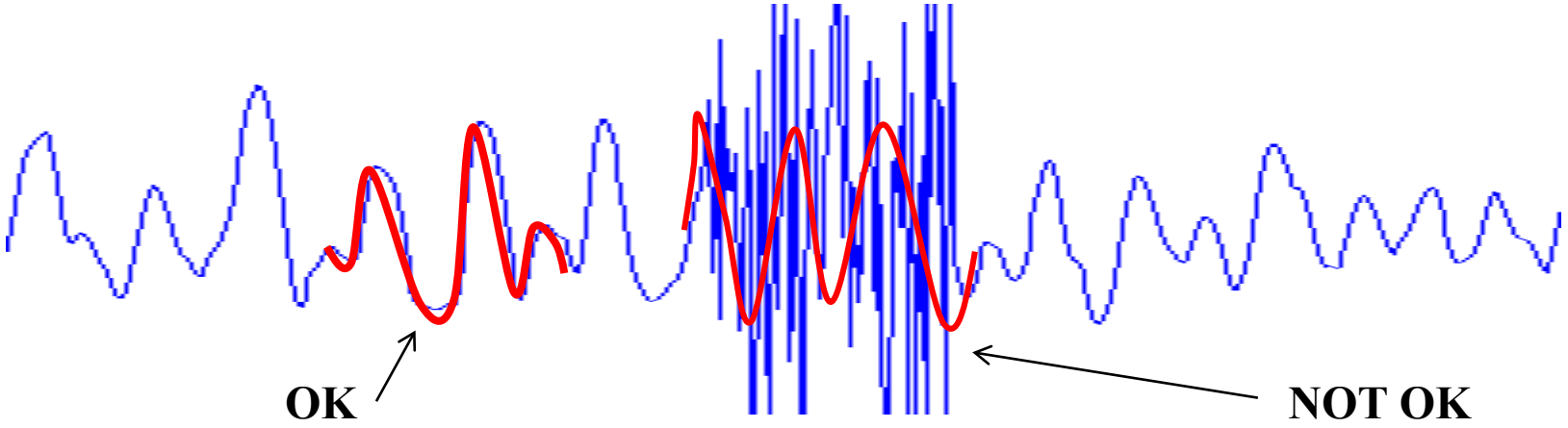
Anomaly detection problem

- Given a dataset D, find all the data points $\boldsymbol{x} \in D$ with anomaly scores greater than some threshold t.
- Given a dataset D, find all the data points $\boldsymbol{x} \in D$ having the top-n largest anomaly scores.
- Given a dataset D, containing mostly normal data points, and a test point $\boldsymbol{x}$, compute the anomaly score of $\boldsymbol{x}$ with respect to D.

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.

A special case: Anomaly in time



- Detection
  - Regression-based reconstruction can be done anywhere
  - Reconstructed value will not match actual value, large error of reconstruction identifies anomaly
- Prediction
  - *Forward* prediction*:* "Extend" the curve on the left to "predict" the values in the "blank" region
  - *Backward* prediction*:* Extend the blue curve on the right leftwards to predict the blank region

Anomaly in time

How?
- ARIMA
- Neural network

Detection

- At each time, learn a "forward" predictor $\boldsymbol{a}_t$
- At each time, predict next sample $x_t^{\text{est}} = \sum_i \alpha_{t,i} x_{t-i}$
- Compute error: $ferr_t = \left| x_t - x_t^{\text{est}} \right|^2$
- Also can learn a "backward" predict and compute backward error
- Compute average prediction error over window. If the error exceeds a threshold, identify burst

Prediction

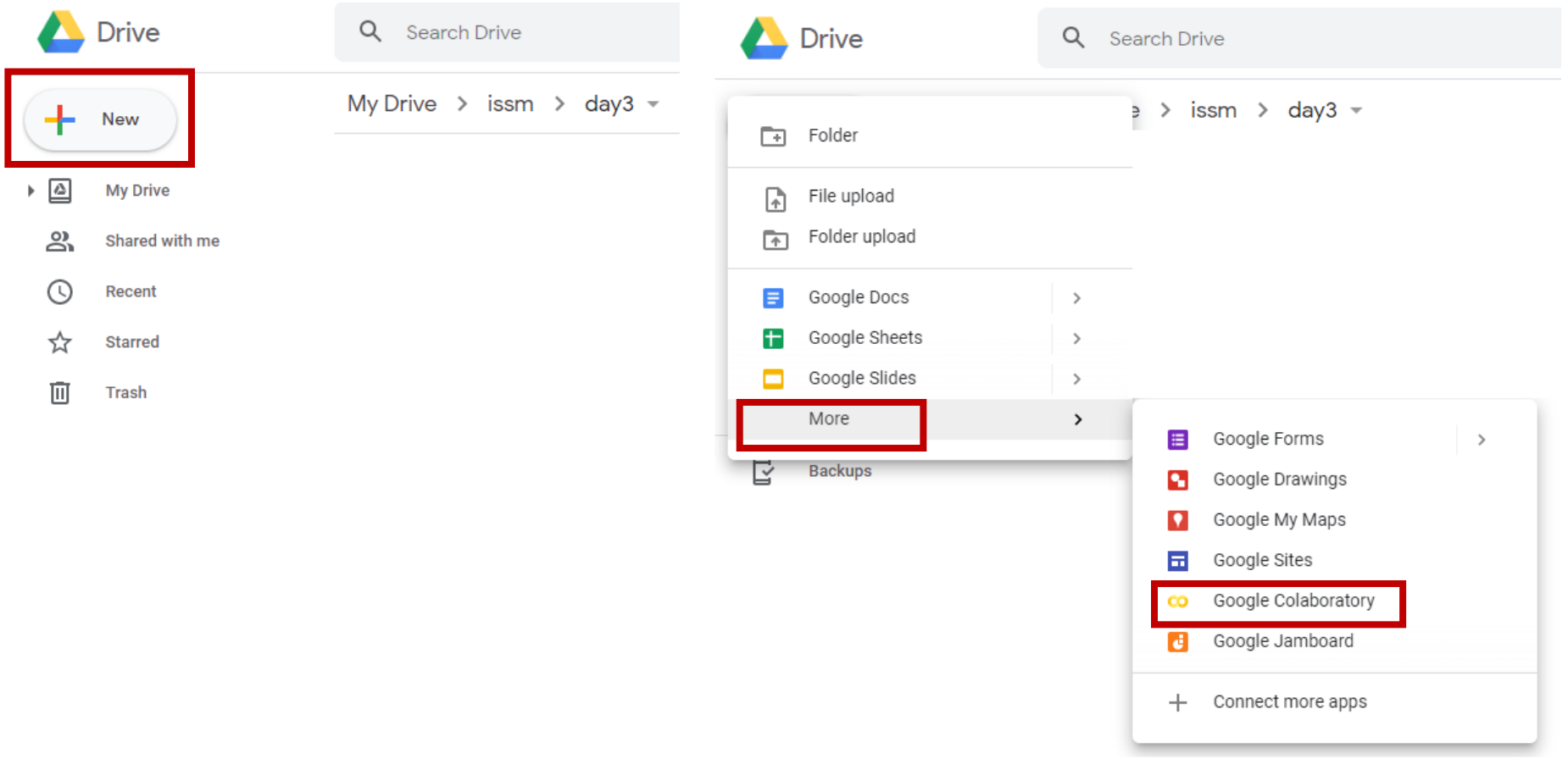
- Learn "forward" predictor at left edge of "hole". For each missing sample, at each time, predict next sample $x_t^{\text{est}} = \sum_i \alpha_{t,i} x_{t-i}$
- Learn "backward" predictor at left edge of "hole". For each missing sample, at each time, predict next sample $x_t^{\text{est}} = \sum_i b_{t,i} x_{t+i}$
- Average forward and backward predictions

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

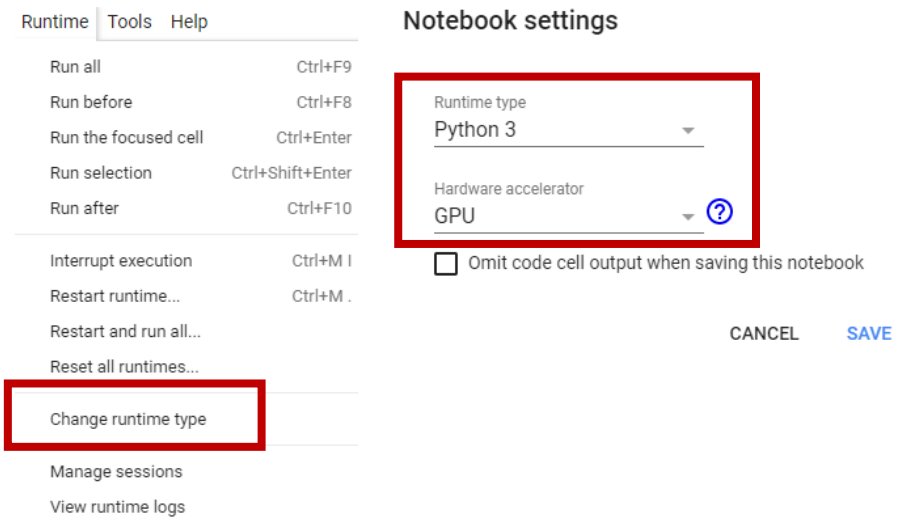
Colab

- Login your Google account in Google drive
- Create a folder for the workshop
- New – More – Google Colaboratory



Colab

- Change runtime type to be GPU

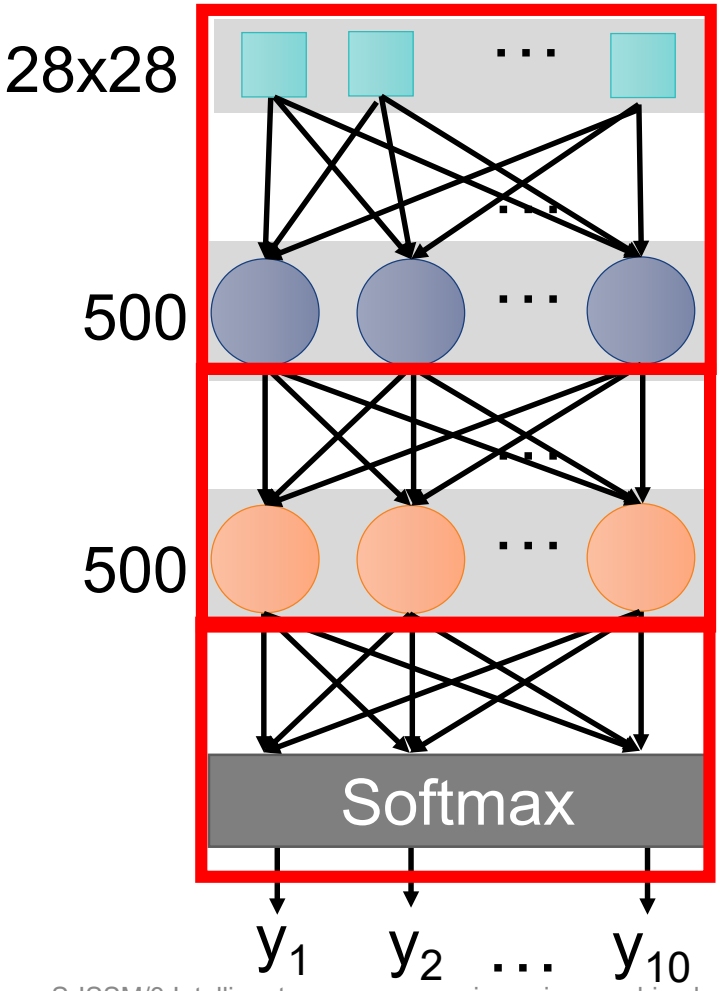


- Similar IDE with Jupyter Notebook

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

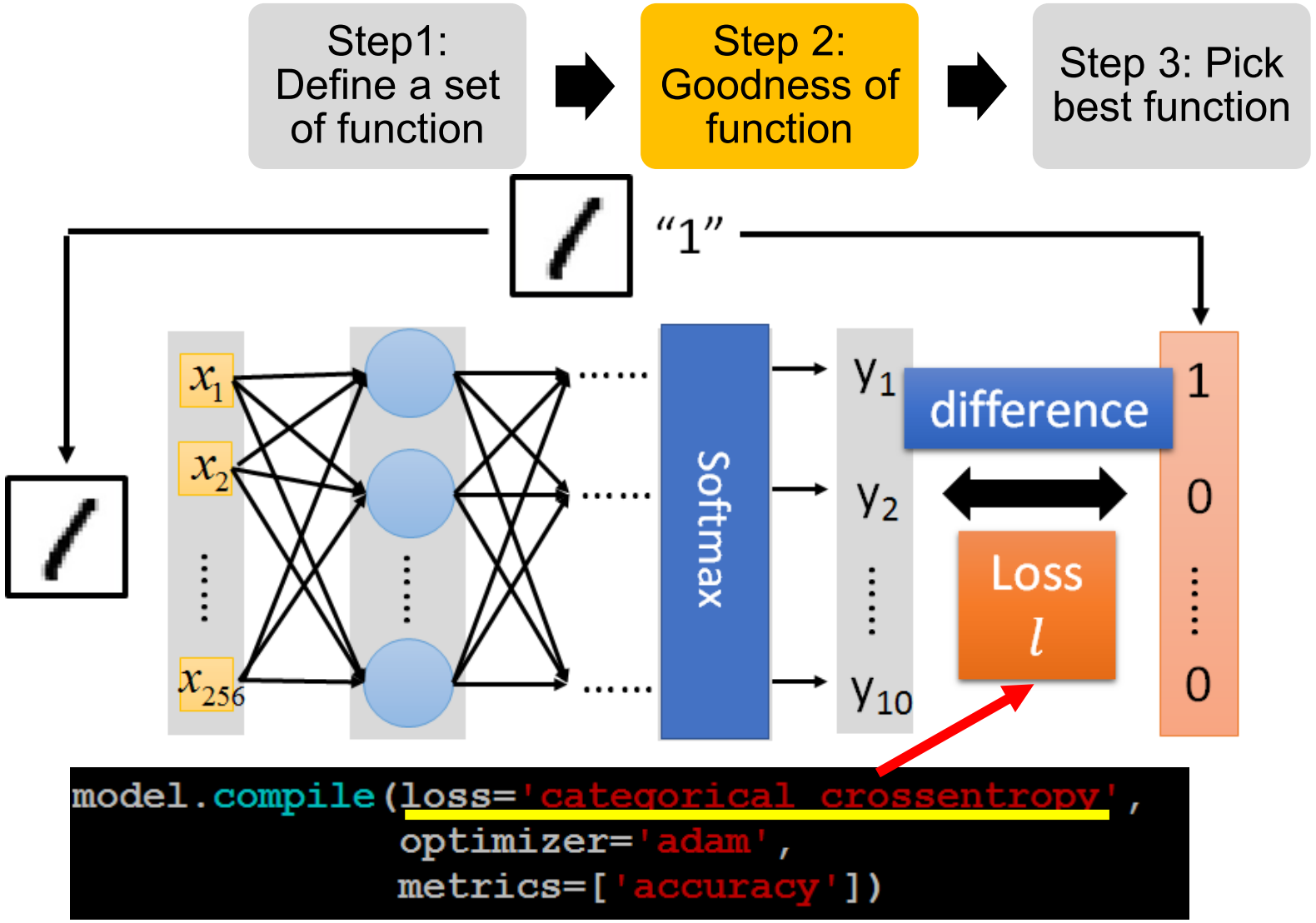
Step1: Define a set of function → Step 2: Goodness of function → Step 3: Pick best function

```
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



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)

Labels (digits)

Training configuration

How to use the neural network (testing)

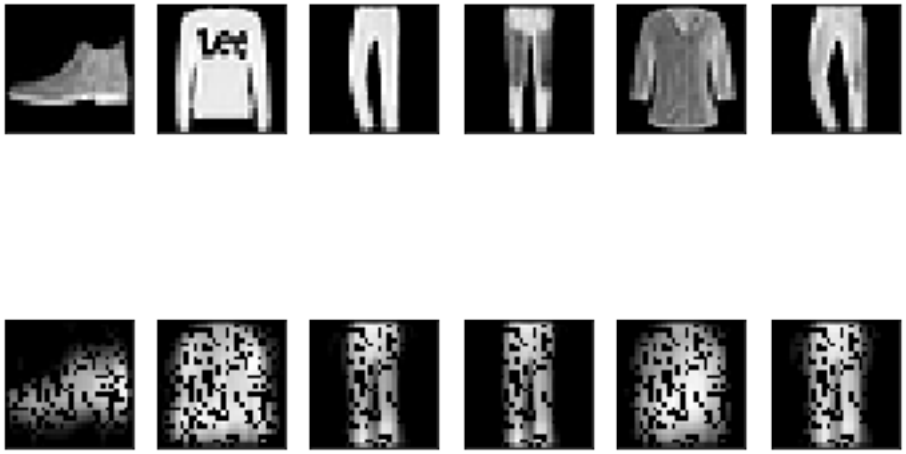
case 1:
```
score = model.evaluate(x_test,y_test)
print('Total loss on Testing Set:', score[0])
print('Accuracy of Testing Set:', score[1])
```

case 2:
```
result = model.predict(x_test)
```

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

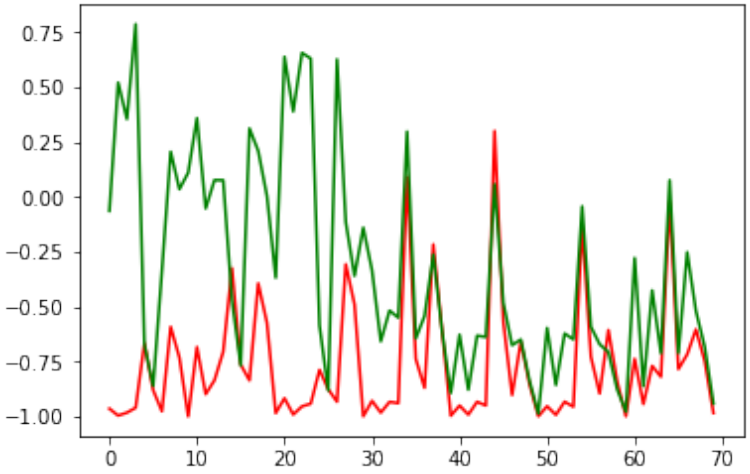
Workshop

- Exercise 1: Introduction to autoencoder model and Keras
- Exercise 2: Perform anomaly detection using autoencoder as reconstruction model



Machine health monitoring

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



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

Summary

- Data driven signal representation
- 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

APPENDIX

Appendix: Eigenvector

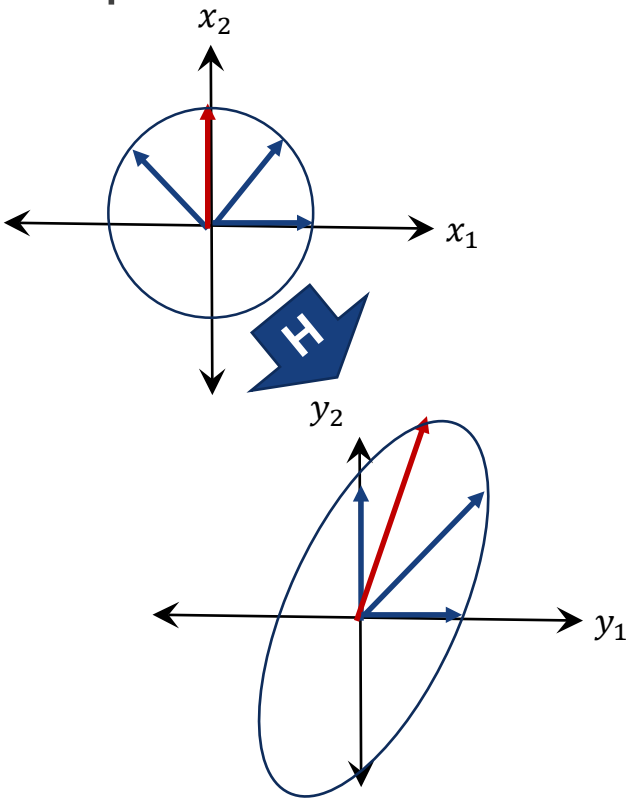
A linear transform $\vec{y} = \mathbf{H}\vec{x}$ maps vector space $\vec{x}$ onto vector space $\vec{y}$.

For example: the matrix $\mathbf{H} = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$ maps the vectors

$$\overrightarrow{x_1}, \overrightarrow{x_2}, \overrightarrow{x_3}, \overrightarrow{x_4} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

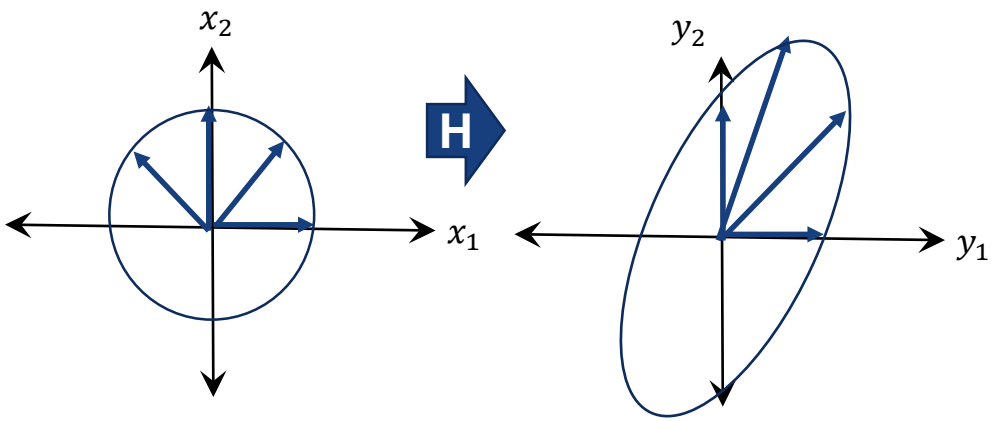
to the vectors

$$\overrightarrow{y_1}, \overrightarrow{y_2}, \overrightarrow{y_3}, \overrightarrow{y_4} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} \sqrt{2} \\ \sqrt{2} \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 0 \\ \sqrt{2} \end{bmatrix}$$



Appendix: Eigenvector

- For a D-dimensional square matrix, there may be up to D different directions $\vec{x} = \overrightarrow{v_d}$ such that, for some scalar λ_d, $\mathbf{H}\overrightarrow{v_d} = \lambda_d \overrightarrow{v_d}$
- For example: if $\mathbf{H} = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$, then eigenvectors and eigenvalues are $\overrightarrow{v_1} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \overrightarrow{v_2} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, \lambda_1 = 1,\ \lambda_2 = 2$



- Multiply the transformation matrix with the eigenvector does not change its direction.

Appendix: Eigenvector

- Exercise: For the following square matrix

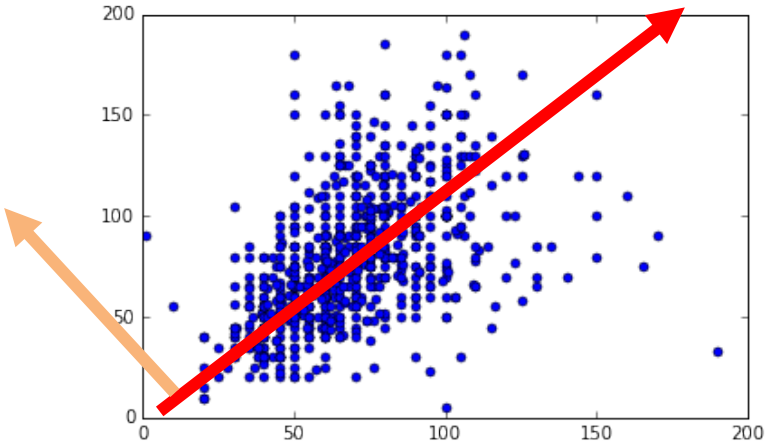
$$\begin{pmatrix} 3 & 0 & 1 \\ -4 & 1 & 2 \\ -6 & 0 & -2 \end{pmatrix}$$

- Decide which, if any of the following vectors are eigenvectors of the above matrix and give the corresponding eigenvalue.

$$\begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \\ 3 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$$

Appendix: PCA

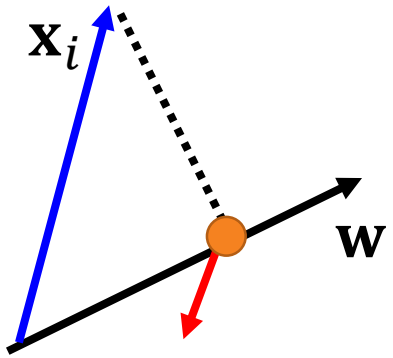
- Given a set of 2-D original data $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$, each data is a 2-D column vector $\mathbf{x}_i = [x_{i1}, x_{i2}]^T$
- A linear transformation matrix $\mathbf{W} = \begin{bmatrix} (\mathbf{w}_1)^T \\ (\mathbf{w}_2)^T \end{bmatrix}$
- A transformed data $\mathbf{Y} = \mathbf{WX} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n\}$, each data is a 2-D column vector $\mathbf{y}_i = [y_{i1}, y_{i2}]^T$



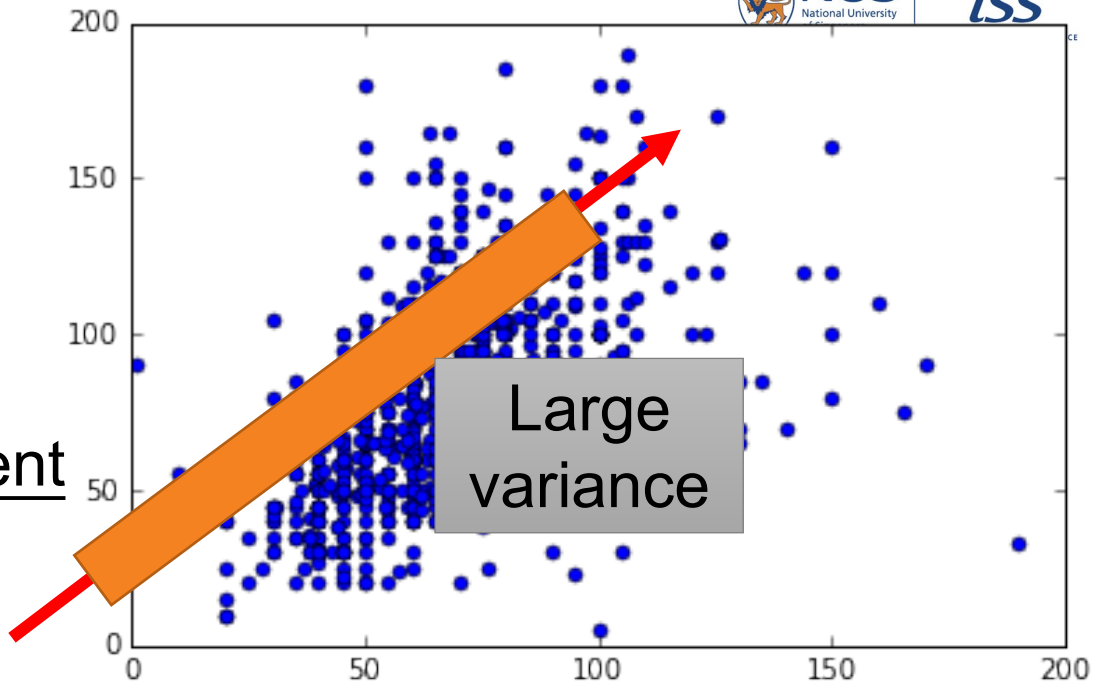
Appendix: PCA

$$\begin{bmatrix} y_{i1} \\ y_{i2} \end{bmatrix} = \begin{bmatrix} (\mathbf{w}_1)^T \\ (\mathbf{w}_2)^T \end{bmatrix} \times \begin{bmatrix} x_{i1} \\ x_{i2} \end{bmatrix}$$

We study the first component



$$y_{i1} = \mathbf{w}_1 \cdot \mathbf{x}_i$$



- Project all the data points $\mathbf{x}_i$ onto $\mathbf{w}_1$, and obtain a set of new data y_{i1}
- We want the variance of y_{i1} as large as possible

$$\text{Find } \mathbf{w}_1 \text{ to maximize } var(y_{i1}) = \frac{1}{N-1}\sum_{i=1}^{N}(y_{i1} - \overline{y_{i1}})^2 \text{ subject to } \|\mathbf{w}_1\|_2 = 1$$

Appendix: PCA

Find $\mathbf{w}_1$ to maximize $var(y_{i1}) = \frac{1}{N-1}\sum_{i=1}^{N}(y_{i1} - \overline{y_{i1}})^2$ subject to $\|\mathbf{w}_1\|_2 = 1$

$$var(y_{i1}) = \frac{1}{N-1}\sum_{i=1}^{N}(y_{i1} - \overline{y_{i1}})^2$$

$$= \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{w}_1 \cdot \mathbf{x}_i - \mathbf{w}_1 \cdot \bar{\mathbf{x}})^2$$

$$= \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{w}_1 \cdot (\mathbf{x}_i - \bar{\mathbf{x}}))^2$$

$$= \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{w}_1)^T(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T\mathbf{w}_1$$

$$= (\mathbf{w}_1)^T\left[\frac{1}{N-1}\sum(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T\right]\mathbf{w}_1 = (\mathbf{w}_1)^T cov(\mathbf{X})\mathbf{w}_1$$

$$\overline{y_{i1}} = \frac{1}{N}\sum y_{i1} = \frac{1}{N}\sum \mathbf{w}_1 \cdot \mathbf{x}_i = \mathbf{w}_1 \cdot \frac{1}{N}\sum \mathbf{x}_i = \mathbf{w}_1 \cdot \bar{\mathbf{x}}$$

$$y_{i1} = \mathbf{w}_1 \cdot \mathbf{x}_i$$

Covariance matrix

Find $\mathbf{w}_1$ to maximize $(\mathbf{w}_1)^T cov(\mathbf{X})\mathbf{w}_1$ subject to $\|\mathbf{w}_1\|_2 = 1$

Appendix: PCA

Find $\mathbf{w}_1$ to maximize $(\mathbf{w}_1)^T cov(\mathbf{X})\mathbf{w}_1$ subject to $\|\mathbf{w}_1\|_2 = 1$

Using Lagrange multiplier set derivative to be zero

$$\mathrm{L}(\mathbf{w}_1) = (\mathbf{w}_1)^T cov(\mathbf{X})\ \mathbf{w}_1 - \alpha\left((\mathbf{w}_1)^T\ \mathbf{w}_1 - 1\right)$$

$$cov(\mathbf{X})\mathbf{w}_1 - \alpha\mathbf{w}_1 = 0$$

$$cov(\mathbf{X})\mathbf{w}_1 = \alpha\mathbf{w}_1$$

$\mathbf{w}_1$: eigenvector of matrix $cov(\mathbf{X})$

$$(\mathbf{w}_1)^T cov(\mathbf{X})\mathbf{w}_1 = \alpha(\mathbf{w}_1)^T\mathbf{w}_1$$ (*Recall $(\mathbf{w}_1)^T\mathbf{w}_1$=1)

$$= \alpha$$ ⟸ Choose the largest eigenvalue

Solution: $\mathbf{w}_1$ is the eigenvector of the covariance matrix $cov(\mathbf{X})$
Corresponding to the 1st largest eigenvalue λ_1

Appendix: PCA

$$\begin{bmatrix} y_{i1} \\ y_{i2} \end{bmatrix} = \begin{bmatrix} (\mathbf{w}_1)^T \\ (\mathbf{w}_2)^T \end{bmatrix} \times \begin{bmatrix} x_{i1} \\ x_{i2} \end{bmatrix}$$

$$y_{i1} = \mathbf{w}_1 \cdot \mathbf{x}_i$$

- Project all the data points $\mathbf{x}_i$ onto $\mathbf{w}_1$, and obtain a set of new data y_{i1}, want the variance of y_{i1} as large as possible

1st component

$$\text{Find } \mathbf{w}_1 \text{ to maximize } var(y_{i1}) = \frac{1}{N-1}\sum_{i=1}^{N}(y_{i1} - \overline{y_{i1}})^2$$
$$\text{subject to } \|\mathbf{w}_1\|_2 = 1$$

2nd component

$$y_{i2} = \mathbf{w}_2 \cdot \mathbf{x}_i$$

- Project all the data points $\mathbf{x}_i$ onto $\mathbf{w}_2$, and obtain a set of new data y_{i2}, want the variance of y_{i2} as large as possible

$$\text{Find } \mathbf{w}_2 \text{ to maximize } var(y_{i2}) = \frac{1}{N-1}\sum_{i=1}^{N}(y_{i2} - \overline{y_{i2}})^2$$
$$\text{subject to } \|\mathbf{w}_2\|_2 = 1 \text{ and } \mathbf{w_1} \cdot \mathbf{w_2} = \mathbf{0}$$

Recall that $\mathbf{W}$ is an orthogonal matrix

Find $\mathbf{w}_2$ to maximize $var(y_{i2}) = \frac{1}{N-1}\sum_{i=1}^{N}(y_{i2} - \overline{y_{i2}})^2$

subject to $\|\mathbf{w}_2\|_2 = 1$ and $\mathbf{w_1} \cdot \mathbf{w_2} = \mathbf{0}$

Using Lagrange multiplier and set derivative to be zero

$$L(\mathbf{w}_2) = (\mathbf{w}_2)^T cov(\mathbf{X})\mathbf{w}_2 - \alpha\left((\mathbf{w}_2)^T\mathbf{w}_2 - 1\right) - \beta\left((\mathbf{w}_2)^T\mathbf{w}_1 - 0\right)$$

$$cov(\mathbf{X})\mathbf{w}_2 - \alpha\mathbf{w}_2 - \beta\mathbf{w}_1 = 0$$ (* Multiply $\mathbf{w}_1$ at both sides)

$$(\mathbf{w}_1)^T cov(\mathbf{X})\mathbf{w}_2 - \alpha(\mathbf{w}_1)^T\mathbf{w}_2 - \beta(\mathbf{w}_1)^T\mathbf{w}_1 = 0$$

$$(\mathbf{w}_1)^T cov(\mathbf{X})\mathbf{w}_2 = \left((\mathbf{w}_1)^T cov(\mathbf{X})\mathbf{w}_2\right)^T$$ (*Recall $(\mathbf{w}_1)^T\mathbf{w}_1$=1, $(\mathbf{w}_1)^T\mathbf{w}_2$=0)

(*Recall $cov(\mathbf{X})\mathbf{w}_1 = \lambda_1\mathbf{w}_1$)

$$= (\mathbf{w}_2)^T\left(cov(\mathbf{X})\right)^T\mathbf{w}_1 = (\mathbf{w}_2)^T cov(\mathbf{X})\mathbf{w}_1 = \lambda_1(\mathbf{w}_2)^T\mathbf{w}_1 = 0$$

So we have $\beta = 0$, then

$$cov(\mathbf{X})\mathbf{w}_2 = \alpha\mathbf{w}_2$$

$\mathbf{w}_2$: eigenvector of matrix $cov(\mathbf{X})$

Solution: $\mathbf{w}_2$ is the eigenvector of the covariance matrix $cov(\mathbf{X})$
Corresponding to the 2nd largest eigenvalue λ_2