QBUS6850
Lecture 6
Neural Networks IV
Representation Learning

BUSINESS SCHOOL

QBUS6850 Team

THE UNIVERSITY OF SYDNEY

Topics covered

- Feature Engineering
- Feature Extraction
- Representation Learning
- Principal Component Analysis
- Spectral Clustering
- Classification Criteria

THE UNIVERSITY OF SYDNEY

Learning Objectives

- Understand different types of features and feature extraction
- Be able to extract features for text data
- Be able to conduct feature engineering by deep learning
- Be able to use classic algorithms for feature engineering and visualisation

Feature Extraction and Representation

Processing Features

- All we have assumed so far is that data come to us in good shape and most of them are in numeric format and possibly in a categorical form
- In Python machine learning, we normally organise data into a matrix (or multidimensional arrays)
- However data coming from application domains could be in any forms or categories
- For business applications, we may have data in the form of text (or natural language), or in media such as audio and videos
- It is easy to deal with numeric data which can be sent to a machine learning straightaway

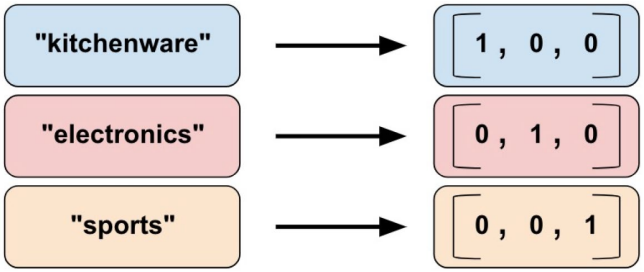
How the real data look like?

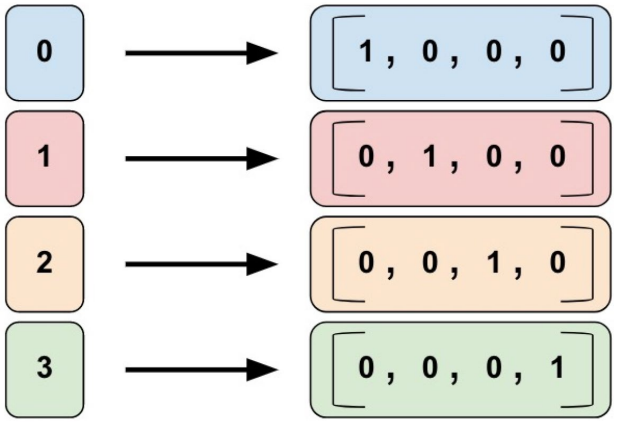
default	student	balance	income
0	Yes	951.0729	18601.53
0	Yes	1292.211	23065.85
0	No	582.1887	24760.8
0	Yes	339.4246	19307.98
0	No	409.9823	44641.09
0	No	1216.452	53639.85
0	No	598.4614	44124.27
1	Yes	1486.998	17854.4
0	No	943.7963	59976.84
0	No	0	49525.75
0	Yes	996.2761	20883.24
0	Yes	748.3013	11248.67
0	No	324.7379	15411.52
0	Yes	575.3822	16005.68
0	No	441.7383	36012.24
0	No	1188.642	39526.56
0	No	95.14768	51371.2
0	No	1015.615	43218.79
0	No	1258.567	44931.67
0	Yes	1178.244	7750.289
0	No	731.5327	43956.06

id	description
5506	This is a private guest room with private bath, It is an independent space, You do not need to cut through anyone elses space to get to guest room. No shared space. Private entrance off main hall/stairs of the building. Has a mini fridge, micro and coffee maker. **THE BEST Value in BOSTON!!*** PRIVATE GUEST ROOM WITH PRIVATE BATH. $99 special!!!! all remaining nights this month! 3 night minimum. Ask for the monthly special. (Special Does not include cleaning or airbnb fee) **Super Value on a Really Nice, Comfortable, Tastefully decorated guest room, Clean, w/ full private bath available for your long or short term stay! Why stay in an overpriced hotel???!! **Excellent Boston location, a 5 minute walk to the Orange Line Train, 4 Minute ride to Copley Plaza, the Center of all Boston attractions! Back Bay/South End, Downtown. **Located in a quiet Boston residential neighborhood, In a Historic Boston Victorian Rowhouse, Circa 1860. We offer the comfort of a Inn-like setting com
6695	** WELCOME *** FULL PRIVATE APARTMENT In a Historic Victorian Brick Row House, Circa 1860. **EXCELLENT SUNNY!! ***HOME AWAY!**. SPECIAL!! ***All Remaining Nights this Month!!! $125 nt, (Special is for up to 2 guests, Does not include cleaning or airbnb fee.) 3 night minimum.. Perfect for Vacation and Extended Stays! Ask for this months promotion! Does not include holidays and special events. Super value on a very nice, comfortable, tastefully decorated, clean, private 1 Bedroom Duplex Condo/Apartment with full kitchen and full bath available for your long or short term stay! Excellent Boston location, 5 minute walk to the Orange Line Train, 4 Minute ride to the Center of all Boston attractions! 3 short stops(1.5miles) to Copley Plaza/Back Bay/South End, Downtown. Walk to Museums from here. We are in the geographical center of Boston. Located in the quiet Boston residential neighborhood of Fort Hill, On a quiet private way, we offer the comfort of a Inn-like setting combined wit
6976	Come stay with me in Boston's Roslindale neighborhood. It is a very safe and suburban part of the city, where most of the houses have driveways and backyards. Your room is fully furnished with an 8 x 10 wool rug, and a nice bureau, a wood-frame, full-size bed, cable TV, WI-FI, a desk to work at, and a new chair. I am an importer of Mexican Folk Art, and run an online store out of the apartment. You will see some wonderful examples of handmade wood, tin and pottery figures on display here. This is a well-maintained, two-family house built in the 1920s. My apartment is on the second floor. This is a pet and smoke-free apartment. PRICE: Price includes ALL utilities (heat, electricity, Wi-Fi, cable TV, air conditioner), parking in street, and use of back yard. NO SMOKING indoors or outside. Note that the bed is a size "Full" mattress, not a Queen or a King. I offer discounted rates for stays of one week or longer. Guests get free coffee and a slice or two of my homemade banana bre

Categorical Features

- In raw data, they are represented by strings. For example “Red”, “Green”, “Blue”, “Yellow”, and “White”.
- They are not suitable to machine learning. We need engineer them into numeric numbers.
  - Label Representation: For example “Red” to 4, “Green” to 3, “Blue” to 2, “Yellow” to 1, and “White” to 0.
  - One-hot Encoding:





Use scikit-learn to make this transform or pandas’ `get_dummies`:

`Lecture06_Example01.py`

Transforming Categorical Features in scikit-learn

❑ Encoding categorical features

- ❖ Converting a categorical feature to one-hot coding by OneHotEncoder

```
>>> enc = preprocessing.OneHotEncoder()
>>> enc.fit([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]])
OneHotEncoder(categorical_features='all', dtype=<... 'numpy.float64'>,
       handle_unknown='error', n_values='auto', sparse=True)
>>> enc.transform([[0, 1, 3]]).toarray()
array([[ 1.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  1.]])
```

3 to 9

❑ Loading features from dicts

The number of features increased

```
>>> measurements = [
...     {'city': 'Dubai', 'temperature': 33.},
...     {'city': 'London', 'temperature': 12.},
...     {'city': 'San Francisco', 'temperature': 18.},
... ]

>>> from sklearn.feature_extraction import DictVectorizer
>>> vec = DictVectorizer()

>>> vec.fit_transform(measurements).toarray()
array([[  1.,   0.,   0.,  33.],
       [  0.,   1.,   0.,  12.],
       [  0.,   0.,   1.,  18.]])

>>> vec.get_feature_names()
['city=Dubai', 'city=London', 'city=San Francisco', 'temperature']
```

2 to 4

from scikit-learn documentation

Ordinal Features

- In raw data, they are represented by strings. For example “Strongly Agree”, “Agree”, “Neutral”, “Disagree”, and “Strongly Disagree”.

- The order information is important for modelling. Most time, we can encode such feature in terms of integers, such as “Strongly Agree” to 5, “Agree” to 4, “Neutral” to 3, “Disagree” to 2, and “Strongly Disagree” to 1.

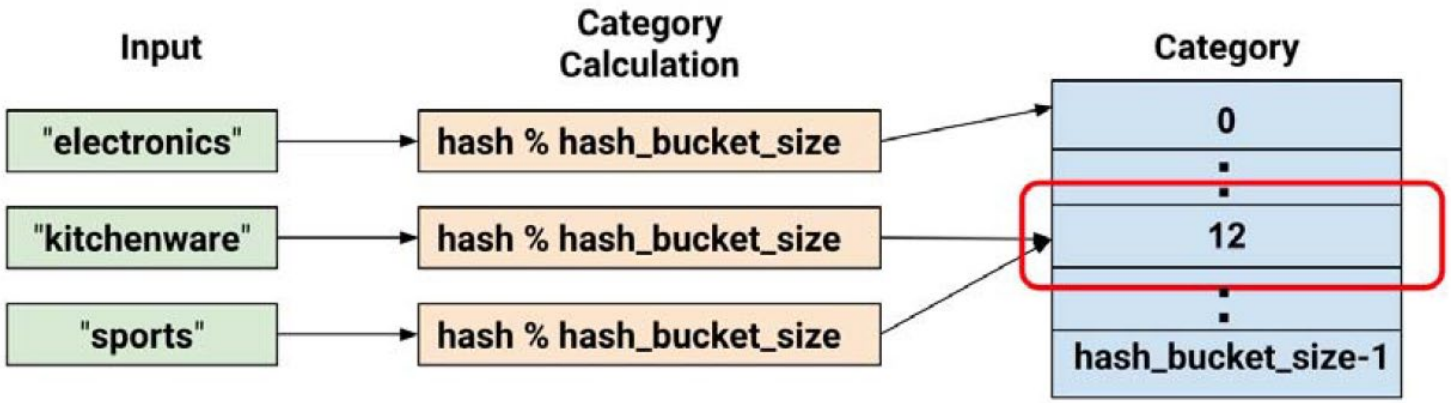
- OrdinalEncoder in scikit-learn can be used to transform such non-numerical labels to numerical labels

```
>>> from sklearn.preprocessing import OrdinalEncoder
>>> enc = OrdinalEncoder()
>>> X = [['Male', 1], ['Female', 3], ['Female', 2]]
>>> enc.fit(X)
OrdinalEncoder()
>>> enc.categories_
[array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
>>> enc.transform([['Female', 3], ['Male', 1]])
array([[0., 2.],
       [1., 0.]])
```

from scikit-learn documentation

Feature Hashing

- If a categorical feature has huge of different values, then the one-hot coding is a long (sparse) vector
- Instead of using a long 0-1 vector, we use a hash function which calculates a hash code. We are forcing the different input values to a smaller set of categories



- Collision: Both “kitchenware” and “sports” may be mapped to the same values

```
hasher = FeatureHasher(input_type='string')
X = hasher.transform(raw_X)
```

Hashed Label Encoder

- `keras` package offers a very sophisticated one hot label encoder:

```
from keras.preprocessing.text import one_hot
```

- It is useful to convert a chunk of text into a set of unique integer label. For example:

```
doc = "This is a demo for one hot encoder. # The text (one document) can be converted to integer labels. "
```

- With a larger dictionary size, we may convert it to

```
label = [927, 367, 59, 18, 858, 864, 405, 481, 875, 451, 864, 334, 307, 574, 817, 85, 263, 629]
```

where we have

```
This -> 927
is   -> 367
.. .. ..
```

- See `Lecture06_Example02.py`

Bag-of-Words

- In Business Intelligence, we analyse texts such business plan, business report, even news.
- Machine learning algorithms cannot work with raw text directly and the text must be converted in to numbers.
- The BoW representation of text describes the occurrence of words within a document
- For example, suppose we have a vocabulary of 1000 words. We have a document in which ``ours'' appears 3 times, ``competition'' appears 1 time and ``managers'' 10 times.
- We will represent this document as a vector of dimension 1000 (such as each component in the vector corresponds to a word in the vocabulary) such that 3 will be in the position of the vector corresponding to ``ours'', 1 at the position corresponding to `` competition'' and 15 at the position corresponding ``managers''.
- All other positions have values of 0. So the vector looks like

$$x = (0, \dots, 0, 3, 0, \dots, 0, 1, 0, \dots, 15, 0, \dots, 0)^T \in \mathbb{R}^{1000}$$

which is sparse. Why? Poll

Bag-of-Words

- Sciki-learn can extract numerical features from text content such as counting the occurrence of tokens in each document

- A corpus of documents can thus be represented by a matrix with one row per document and one column per token (e.g. word) occurring in the corpus

- See `Lecture06_Example03.py`

Text Feature Extraction

There are other definitions for these two "frequencies"

- **tf-idf Term Weighting**
  - tf-idf(t, d) is defined as

$$\text{tf-idf}(t,d) = \text{tf}(t,d) \times \text{idf}(t)$$

  - tf(t,d) (term frequency): is the frequency of a term *t* in a document *d*, i.e., the ratio of the occurring count number of *t* in *d* to the number of different words in *d*
  - idf(t) **inverse document frequency:** a measure of how much information the word *t* provides, that is, whether the term is common or rare across all documents

$$\text{idf}(t) = \log\left(\frac{\text{the total of number of documents}}{\text{the number of documents in corpus containing term } t}\right)$$

- How is this done in scikit-learn? `Lecture06_Example04.py`

Representation Learning

Representation Learning

- Both One-hot Encoding and BoW may produce feature representations which are of large dimension and sparse.
- High dimension will result in the so-called curse of dimensionality problem in many machine learning algorithms.
- Each of these has its own drawback. For example, if we use one-hot encoding for Colour names like Red, Green and Madder, there is no way for us to tell Red and Madder is much more similar than Red and Green. Although BoW can tell frequency information, it does not tell any "semantic" information of text.
- Is there a way for us to do better?
- The so-called Representation Learning (*extension of dimensionality reduction*) can achieve this goal.

Representation Learning

- Representation learning automatically discovers the feature patterns in the data according to certain task goals.
- We will provide the (raw) data to the machine, it learns the representation itself without any human intervention.
- The goal of representation learning is to train machine learning algorithms to learn useful representations, such as those that are interpretable, incorporate latent features, or can be used for transfer learning.
- We will discuss the concept of representation learning along with its need and different approaches through demonstration.
- Representation learning should not be “new” to us, indeed when after train a deep neural network, you can use the hidden layer as a new representation of data.

https://analyticsindiamag.com/a-comprehensive-guide-to-representation-learning-for-beginners/

Embedding Layer

- An embedding layer can be used to conduct representation learning, which convert categorical index (hot-one code) to vectorial representation in much lower dimension
- You can imagine that an embedding layer is a normal linear layer which connects the input layer and a hidden layer to a hidden layer with only given number of neurons.
- The inputs are normally the hot-one features or categorical indices
- This is particularly useful in feature engineering words.

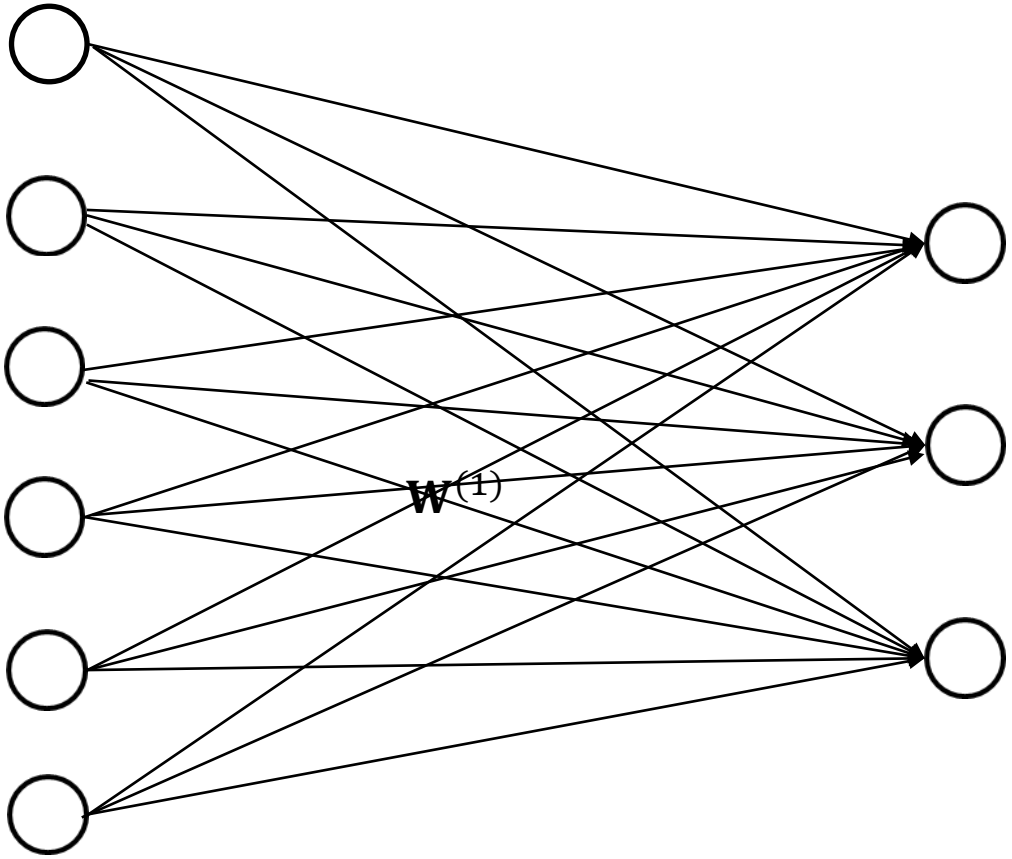
Understanding Embedding Layer

- Both Pytorch and Keras implement their Embedding layers.
- Let us explain it from a Linear layer view in this example:
- consider a color bank ={red, limegreen, yellow, blue, peru, magenta} and their one-hot coding

```
red       = (1, 0, 0, 0, 0, 0)
limegreen = (0, 1, 0, 0, 0, 0)
yellow    = (0, 0, 1, 0, 0, 0)
blue      = (0, 0, 0, 1, 0, 0)
peru      = (0, 0, 0, 0, 1, 0)
magenta   = (0, 0, 0, 0, 0, 1)
```

Understanding Embedding Layer

- Consider the following linear layer with their connecting weights.

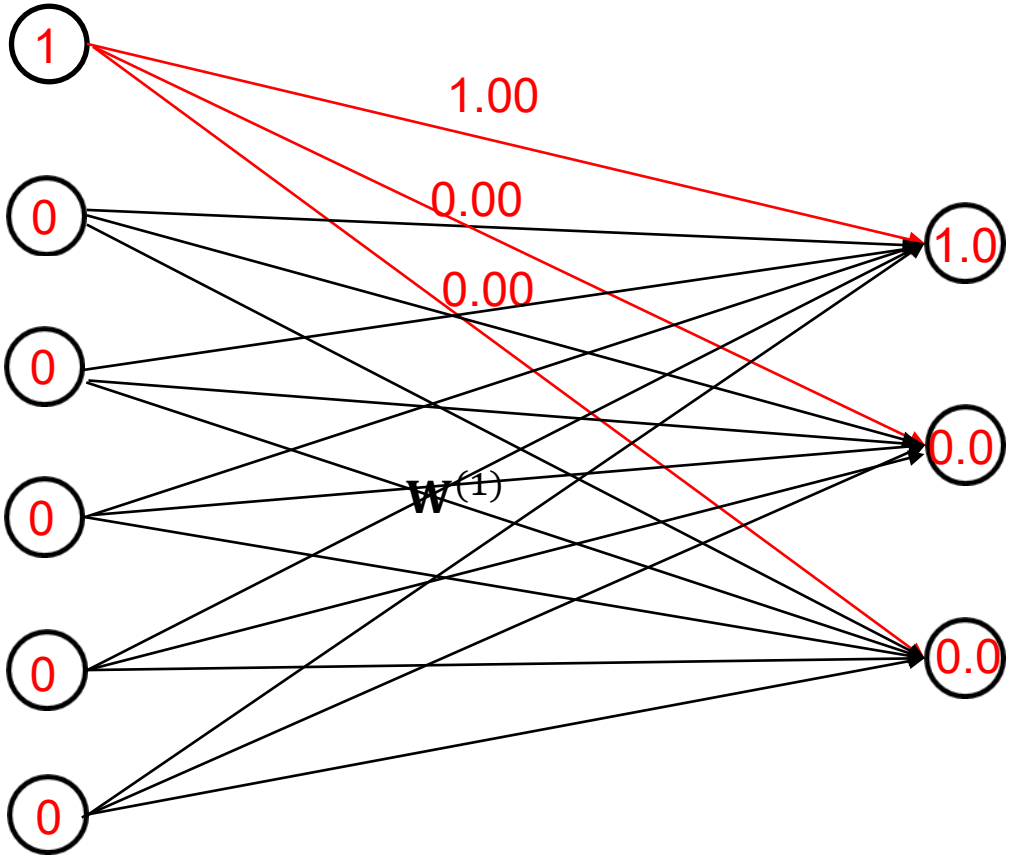


$$\mathbf{W}^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} \\ w_{41}^{(1)} & w_{42}^{(1)} & w_{43}^{(1)} \\ w_{51}^{(1)} & w_{52}^{(1)} & w_{53}^{(1)} \\ w_{61}^{(1)} & w_{62}^{(1)} & w_{63}^{(1)} \end{bmatrix}$$

$$= \begin{bmatrix} 1.00 & 0.00 & 0.00 \\ 0.13 & 0.55 & 0.13 \\ 1.00 & 0.75 & 0.00 \\ 0.00 & 0.00 & 1.00 \\ 0.80 & 0.52 & 0.25 \\ 0.54 & 0.00 & 0.54 \end{bmatrix}$$

Understanding Embedding Layer

- Take input **red** = (1, 0, 0, 0, 0, 0)

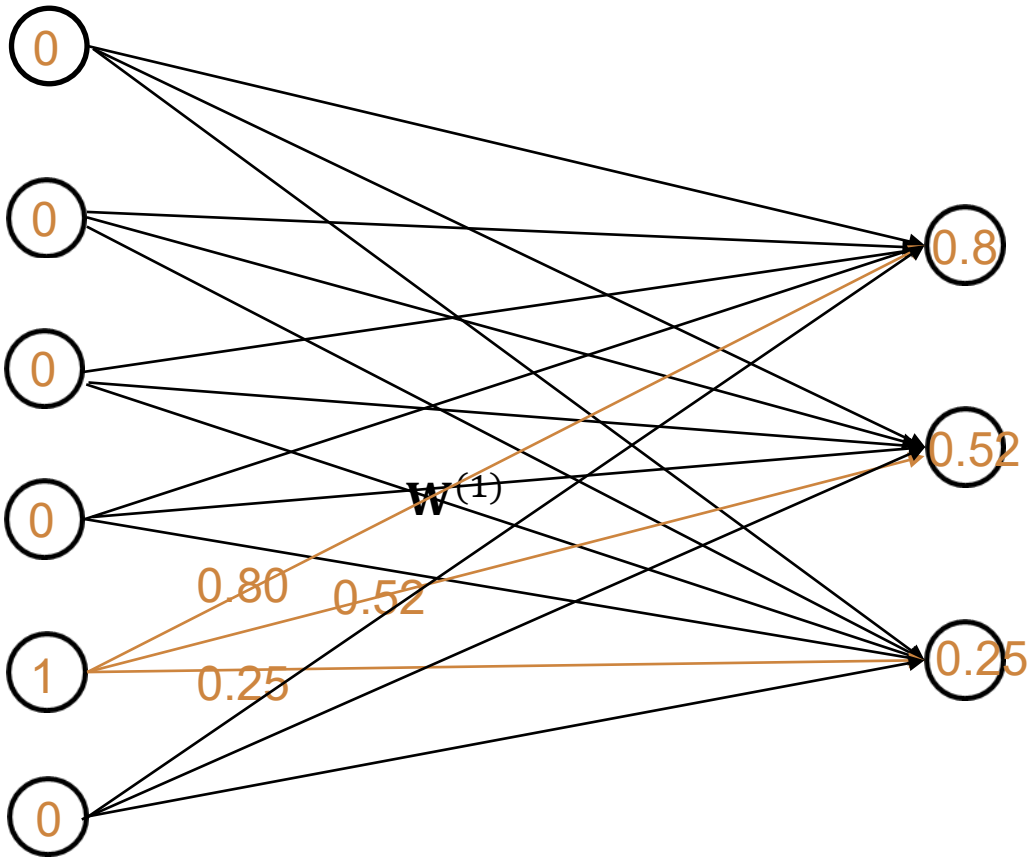


$$\mathbf{W}^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} \\ w_{41}^{(1)} & w_{42}^{(1)} & w_{43}^{(1)} \\ w_{51}^{(1)} & w_{52}^{(1)} & w_{53}^{(1)} \\ w_{61}^{(1)} & w_{62}^{(1)} & w_{63}^{(1)} \end{bmatrix}$$

$$= \begin{bmatrix} 1.00 & 0.00 & 0.00 \\ 0.13 & 0.55 & 0.13 \\ 1.00 & 0.75 & 0.00 \\ 0.00 & 0.00 & 1.00 \\ 0.80 & 0.52 & 0.25 \\ 0.54 & 0.00 & 0.54 \end{bmatrix}$$

Understanding Embedding Layer

- Take input `peru = (0, 0, 0, 0, 1, 0)`



$$\mathbf{W}^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} \\ w_{41}^{(1)} & w_{42}^{(1)} & w_{43}^{(1)} \\ w_{51}^{(1)} & w_{52}^{(1)} & w_{53}^{(1)} \\ w_{61}^{(1)} & w_{62}^{(1)} & w_{63}^{(1)} \end{bmatrix}$$

$$= \begin{bmatrix} 1.00 & 0.00 & 0.00 \\ 0.13 & 0.55 & 0.13 \\ 1.00 & 0.75 & 0.00 \\ 0.00 & 0.00 & 1.00 \\ 0.80 & 0.52 & 0.25 \\ 0.54 & 0.00 & 0.54 \end{bmatrix}$$

Understanding Embedding Layer

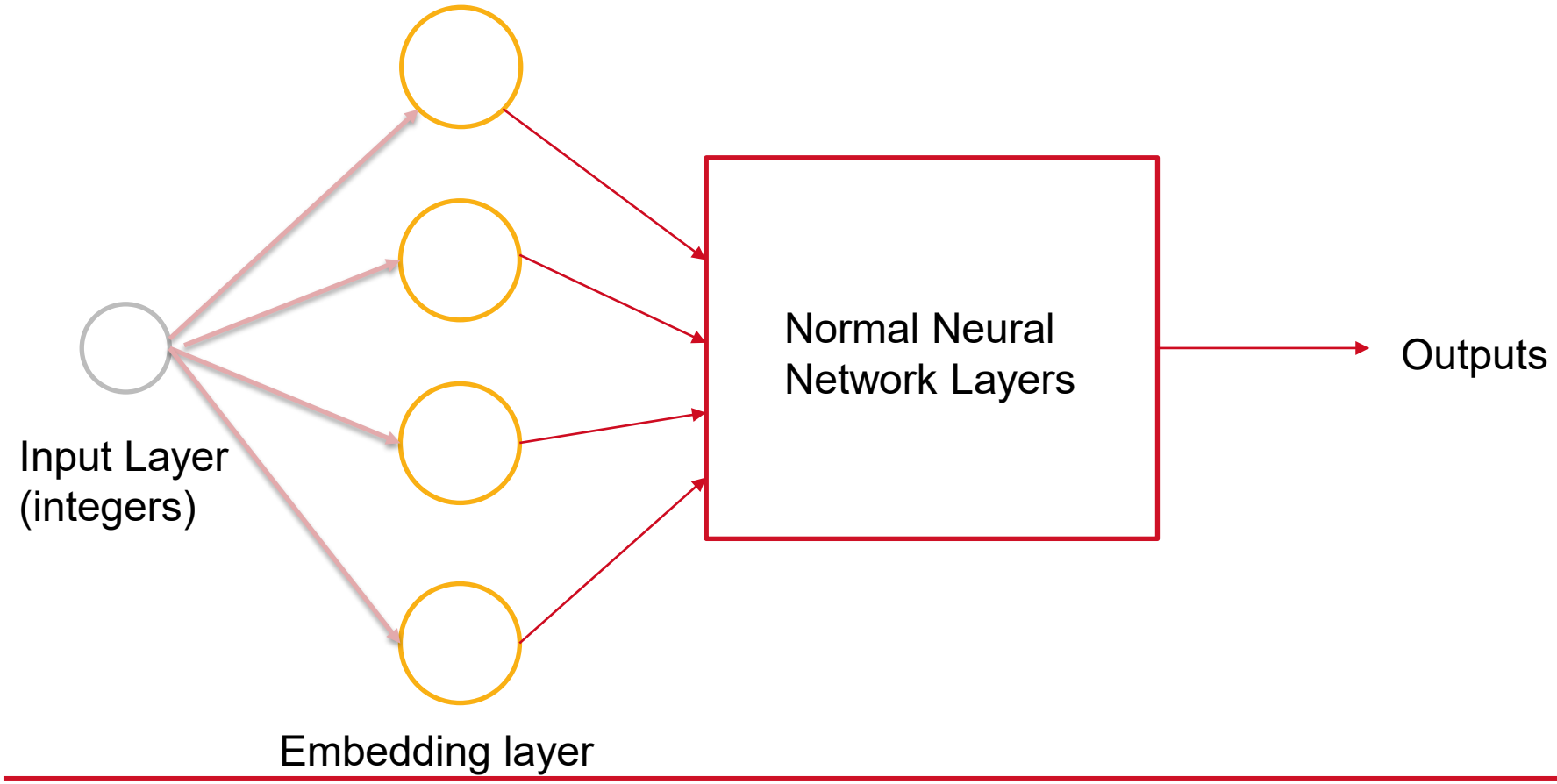
- This linear layer operation is equivalent to pick up a row from the layer weight matrix $\mathbf{W}^{(1)}$
- By this equivalence, we don't need explicitly use one-hot coding and the linear layer, instead we only need the index (which color) and the linear weight matrix.
- This is how pytorch or keras implements such a so-called embedding layer, in which a weight matrix is maintained, and forward operation is to pick a row from the matrix according to the index. This is a look-up table way.
- The matrix row now becomes the feature of the index (e.g a colour name)

Using Embedding Layer

- In the previous example, the weight matrix values are given, indeed they are color RGB values
- However we can turn this matrix into parameter matrix, and put this Embedding layer in our neural networks (as the first hidden layer), then we can learn embeddings for all the index (e.g. for color names)
- This is particularly useful in coding words in business applications (or Natural Language Processing – NLP)
- In tutorial, we will learn how to build a neural network model to learn our own color name embedding, rather than RGB

Using Embedding Layer

- A typical neural network architecture with embedding layer(s)

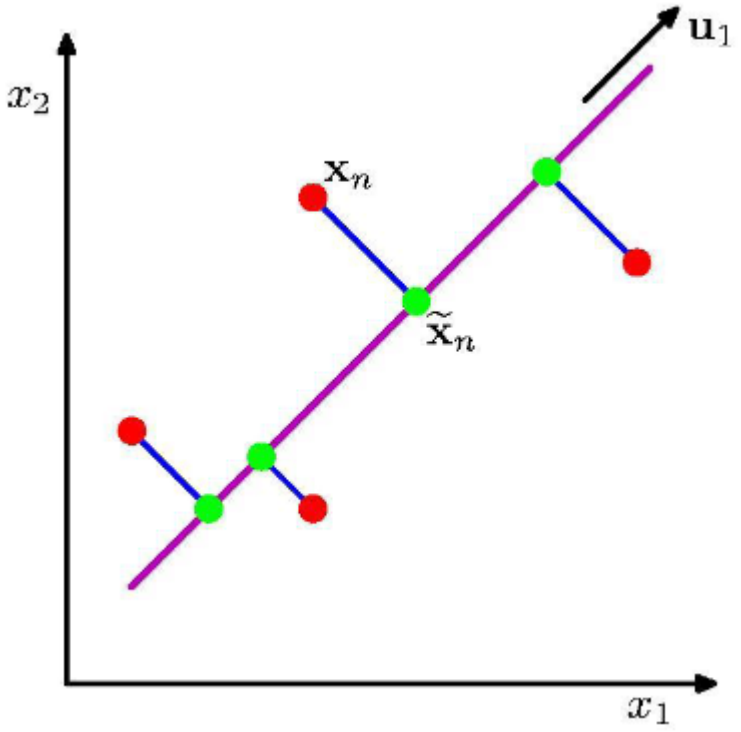


Principal Component Analysis (PCA)

Poll 1

Principle Component Analysis (PCA)

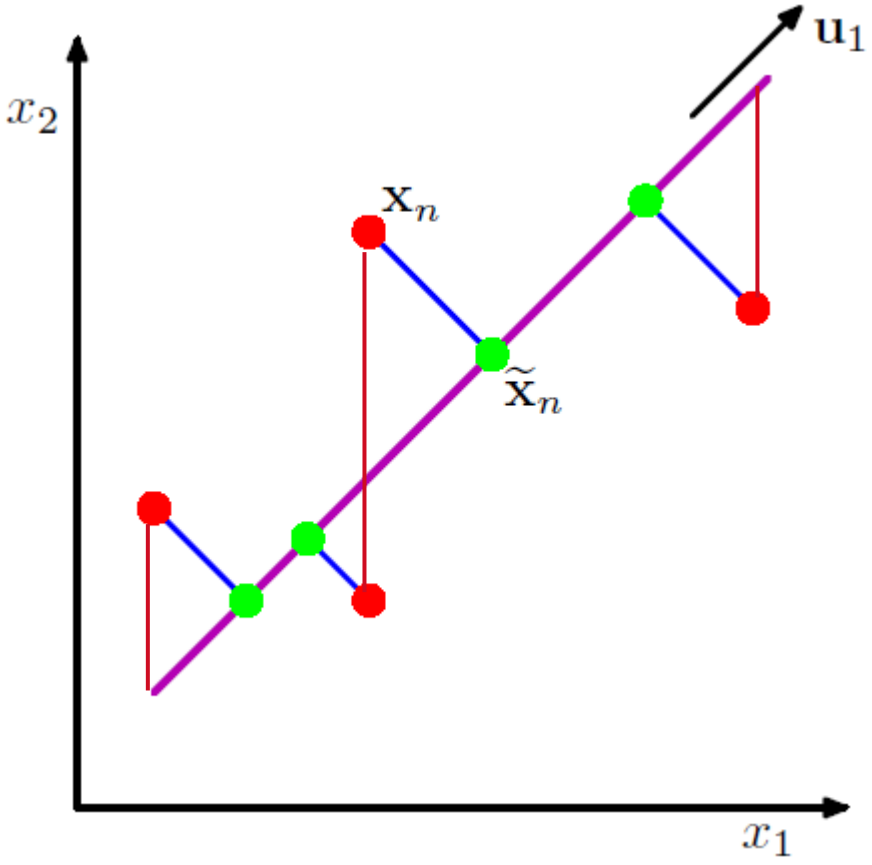
- Principal component analysis seeks a space of lower dimensionality, known as the principal subspace and denoted by the magenta line, such that the orthogonal projection of the data points (red dots) onto this subspace maximizes the variance of the projected points (green dots).
- How can we find this line? Can we do this by linear regression?



Poll 2

Principle Component Analysis (PCA)

- Principal component analysis seeks a space of lower dimensionality, known as the principal subspace and denoted by the magenta line, such that the orthogonal projection of the data points (red dots) onto this subspace maximizes the variance of the projected points (green dots).
- How can we find this line? Can we do this by linear regression?



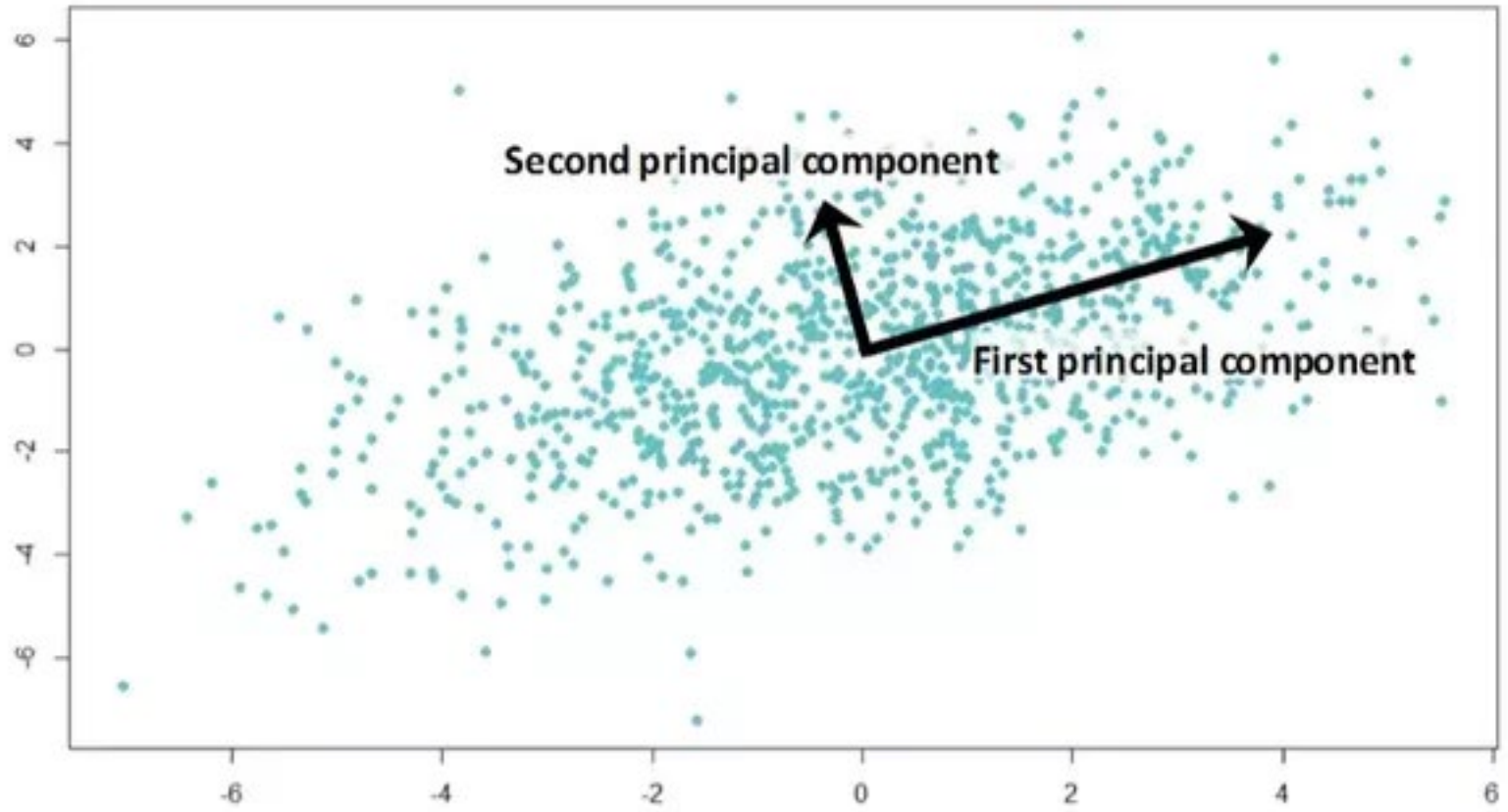
Principle Component Analysis (PCA)

- Objective: given a set of *d* measurements on *N* individuals, we aim at determining $r \leq d$ orthogonal (uncorrelated) variables, called principal components, defined as linear combinations of the original ones

- The PCs are uncorrelated and have decreasing variance
  - Synthesis: information dimensionality reduction
  - Interpretation: express the original data in terms of a reduced number of underlying variables (factors)
  - Score the individual proles, with a summary score
  - Obtain multivariate displays (scatterplot) of the units in two or three dimensions

- The first component is designed to capture as much of the variability in the data as possible, and the succeeding components in turn extract as much of residual variability as possible

2D Gaussian Dataset



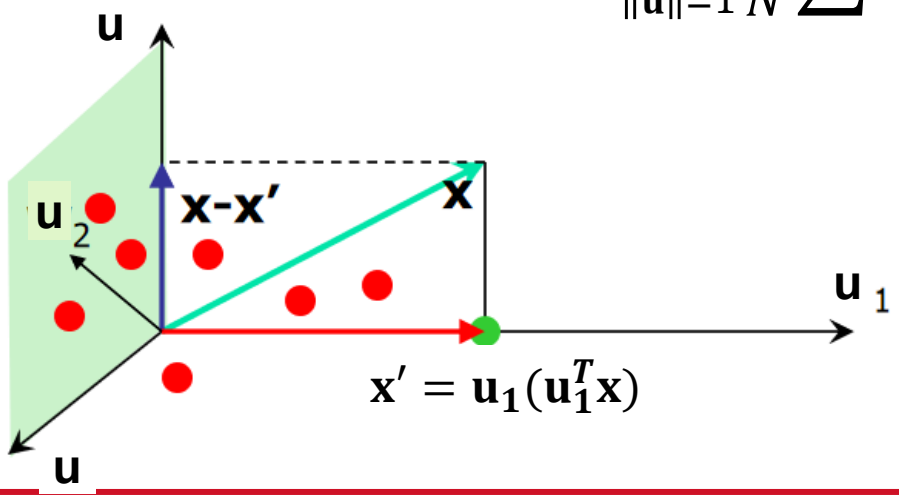
Mathematical Problem of PCA

- Given the centred d-dimensional data $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, compute the first PCA vector $\mathbf{u}_1$ of dimension d as solving for

$$\mathbf{u}_1 = \arg\max_{\|\mathbf{u}\|=1} \frac{1}{N} \sum_{n=1}^{N} (\mathbf{u}^T \mathbf{x}_n)^2$$

- To find $\mathbf{u}_1$, we are maximizing the variance of project of $\mathbf{x}$
- The second PCA vector is solved by

$$\mathbf{u_2} = \arg\max_{\|\mathbf{u}\|=1} \frac{1}{N} \sum^{N} (\mathbf{u}^T (\mathbf{x}_n - \mathbf{u}_1 \mathbf{u}_1^T \mathbf{x}_n))^2$$



Mathematical Problem of PCA*

- Given the centred d-dimensional data $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, the first r $(1 \leq r \leq d)$ PCA components $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_r]$ are defined as the solution of the following optimization problem:

$$\mathbf{U}_r = \arg\min_{\mathbf{U}^T\mathbf{U}=\mathbf{I}} \frac{1}{N} \|\mathbf{X} - \mathbf{X}\mathbf{U}\,\mathbf{U}^T\|_2^2$$

- There is a very nice closed-form solution for this problem. This is the classic PCA algorithm. See the next slide

PCA: The Algorithm

Size $N \times d$

- Given a set of data $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$. Suppose they have been centralised, i.e., removing the mean from them. Collect them in a data matrix $\mathbf{X}$

Size $d \times d$

- Calculate the variance matrix $\mathbf{S} = \frac{1}{N}\mathbf{X}^T\mathbf{X}$
- Conduct the eigen-decomposition of $\mathbf{S}$ such that

$$\mathbf{S} = \mathbf{U}\Lambda\mathbf{U}^T$$

where $\mathbf{U^T U} = \mathbf{I_d}$ and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_d)$ where λ_i in descending order, ie. $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$.

- The first r $(r \leq d)$ principal components of $\mathbf{X}$ are given by

Size $N \times r$

$$\mathbf{Z}_r = \mathbf{X}\mathbf{U}_r$$

where $\mathbf{U}_r$ is the matrix of the first r columns of $\mathbf{U}$.

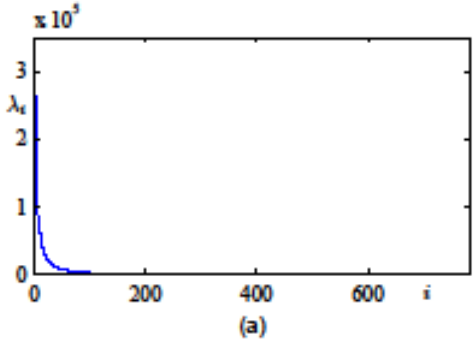
- Each row of $\mathbf{Z}_r$ (in r new factors/features) is a new representation of the given data, i.e., the corresponding row in $\mathbf{X}$ (in d attributes/features)

PCA: Selecting r

- Consider the share of the total variance absorbed by the first r components $\mathbf{Z}_r$

$$Q_r = \frac{\sum_{h=1}^{r} \lambda_h}{\sum_{h=1}^{d} \lambda_h}$$

Select r so that $Q_r \geq 0.95$ for example



- Kaiser criterion: compute the average eigenvalues

$$\bar{\lambda} = \frac{1}{d} \sum_{h=1}^{d} \lambda_h$$

- Select the first r components for which $\lambda_h > \bar{\lambda}$. Note: if the variables are standardised $\bar{\lambda}$ = 1.

THE UNIVERSITY OF SYDNEY

Python Example

(Lecture06_Example05.py)

Autoencoder

PCA as a Neural Network

- PCA definition: Given the centred d-dimensional data $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, the first r $(1 \leq r \leq d)$ PCA components $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_r]$ are defined as the solution of the following optimization problem:

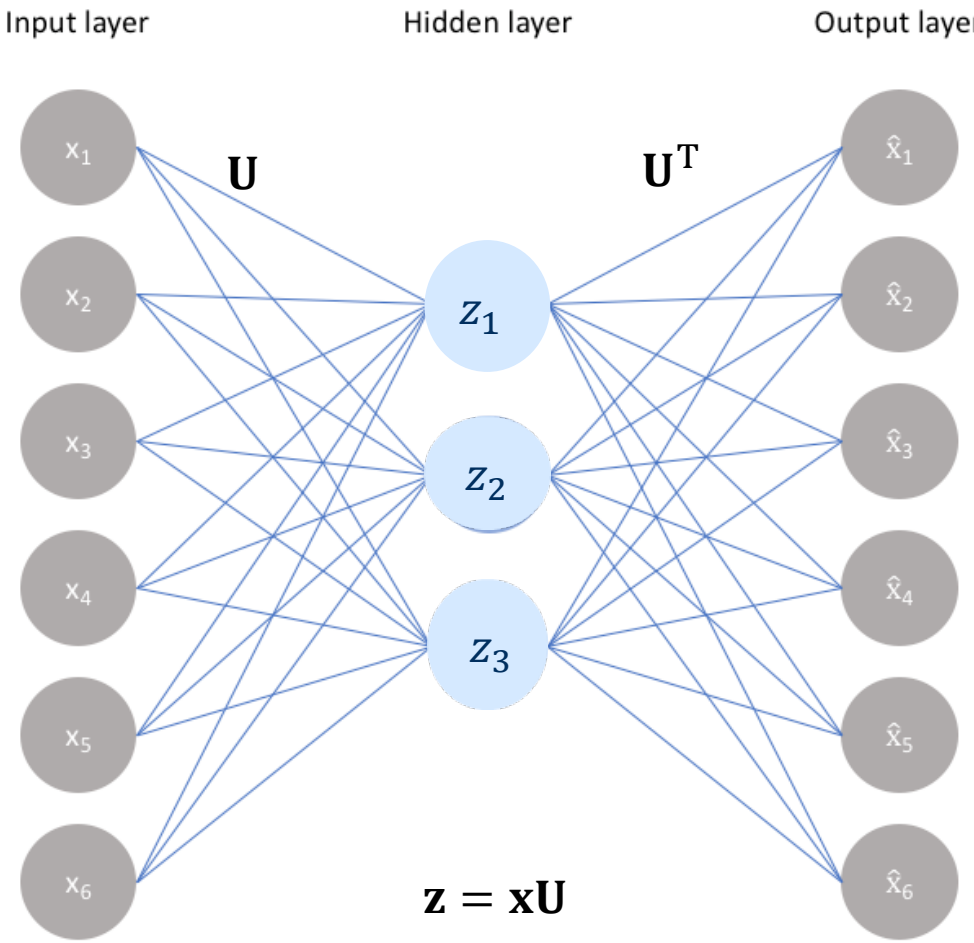
$$\mathbf{U}_r = \arg \min_{\mathbf{U}^T\mathbf{U}=\mathbf{I}} \frac{1}{N} \|\mathbf{X} - \mathbf{X}\mathbf{U}\,\mathbf{U}^T\|_2^2$$

- We can use a 3-layer neural network to unfold the PCA problem. The network consists of two linear layers, one with connection matrix $\mathbf{U}$ and the other with $\mathbf{U}^T$.

- Please note that in general neural networks, connection weight matrices $\mathbf{W}^{(l)}$ are different in each layer. However here we take a special requirement.

PCA as a Neural Network

No activation

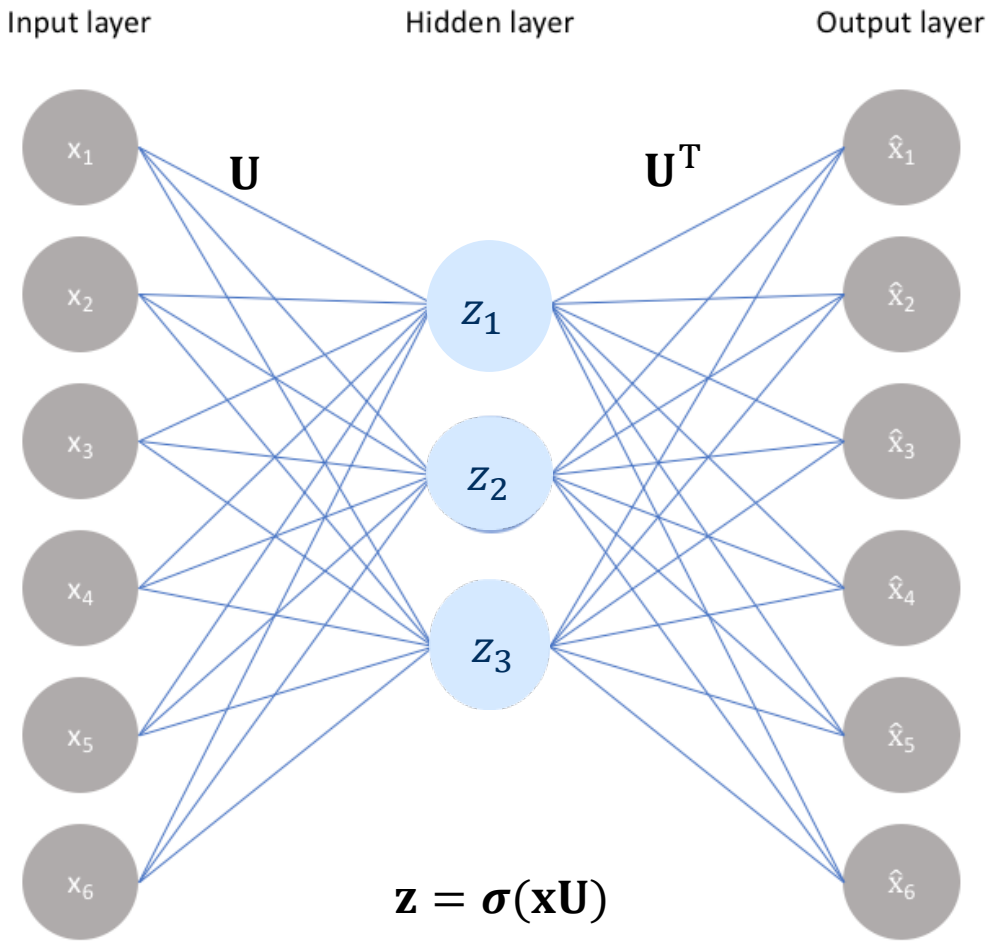


$$\hat{\mathbf{x}} = \mathbf{z}\mathbf{U}^{\mathrm{T}} = \mathbf{x}\mathbf{U}\mathbf{U}^{\mathrm{T}}$$

Image source: https://www.jeremyjordan.me/autoencoders/

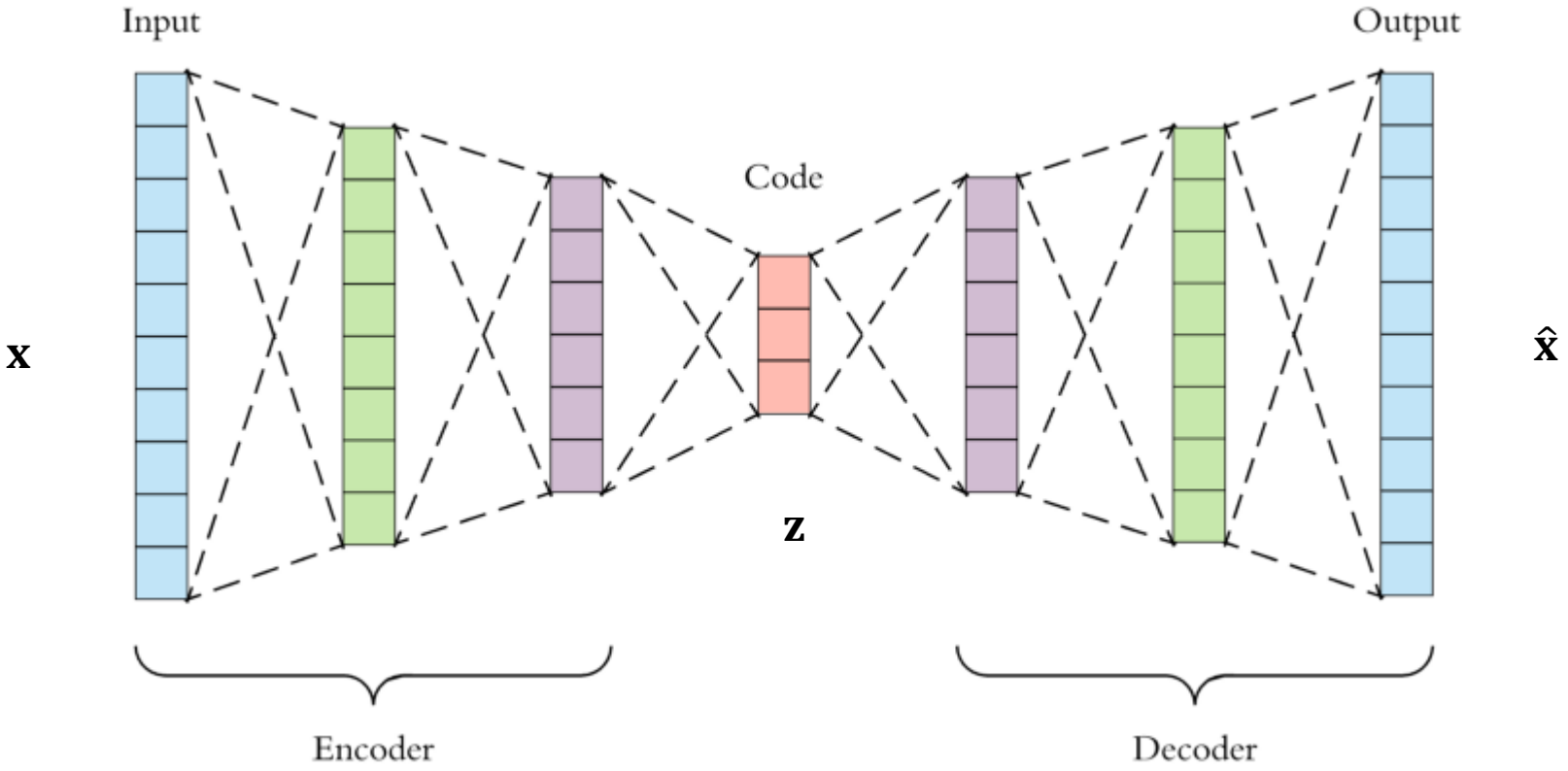
Autoencoders

1. Add activation
2. Remove the condition $\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}$
3. Increase the number of layers



$$\hat{\mathbf{x}} = \mathbf{z}\mathbf{U}^{\mathrm{T}}$$
$$= \sigma(\mathbf{xU})\mathbf{U}^{\mathrm{T}}$$

Autoencoders



Objective: $\min ||\mathbf{x} - \hat{\mathbf{x}}||^2$

https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Summary (Representation Learning)

- Representation learning can improve the model's performance in three learning frameworks – supervised learning, unsupervised learning, and reinforcement learning (not covered in this unit)

- Advantages include:
  - Improved performance of model using representation learning.
  - Interpretability gets improved when we are using featured representation which is a prime business constraint for any ML task
  - Automatic feature learning can be done through neural networks which eliminates the need of domain expert while solving a real world problem as model learns in an incremental manner. No human designed feature engineering is required
  - We can get better results using unlabelled data.

Spectral Clustering*

Manifold Learning*

- PCA is a linear model. It is not good for data on a lower dimensional “manifold” space.
- For example, if the data cloud is around a sphere in 3D space, which is indeed of two freedom. How could we find its 2D representation
- This is particularly useful for reducing the dimensionality for data corrupted in a much higher dimension space.
- The way of nonlinearly inferring low dimensional representation for high dimensional data is called “manifold learning”
- https://en.wikipedia.org/wiki/Nonlinear_dimensionality_reduction
- Among those state-of-the-art manifold learning method, t-SNE has been demonstrated most powerful
- You can use t-SNE to reduce dimensional of data (e.g., Bag-of-Words vectors) to lower dimensional, e.g., 2D or 3D, and visualize your data.
- See `Lecture06_Example06.py`

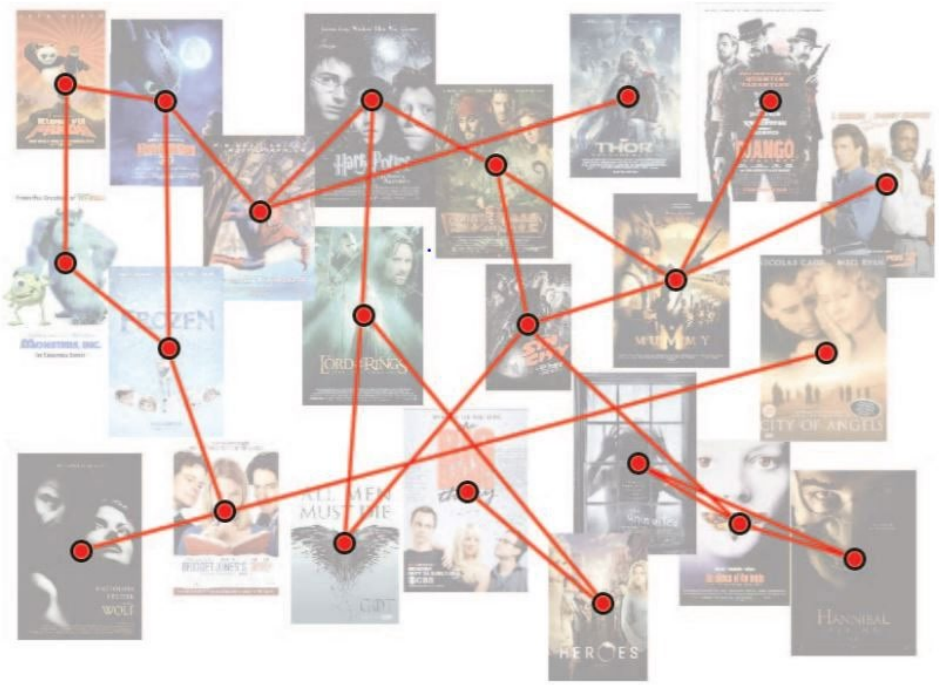
Spectral Clustering*

- Clustering is a typical type of unsupervised learning techniques.
- k-Means is a representative clustering method in machine learning (covered by QBUS6810)
- Another widely used clustering method is the so-called Spectral Clustering which take the relation information among data into account.
- Spectral clustering has become increasingly popular due to its simple implementation and promising performance in many graph-based clustering

Poll 4

Spectral Clustering*

- Step 1: The dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ is presented in the form of a similarity graph G = (V, E) where V is the node set defined by the data $\mathcal{D}$ while E is the relation information between each pair of data $(\mathbf{x}_i, \mathbf{x}_j)$ https://towardsdatascience.com/spectral-clustering-for-beginners-d08b7d25b4d8

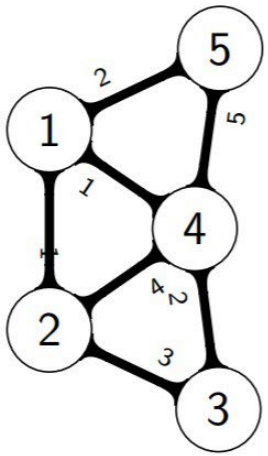


Spectral Clustering*

- Step 2: From the graph, we can form the so-called Laplacian matrix $\mathbf{L}$ from the adjacency matrix $\mathbf{A}$ and/or weight matrix $\mathbf{W}$

$\mathbf{A}$	adjacency matrix
$\mathbf{W}$	weight matrix
$\mathbf{D}$	(diagonal) degree matrix
$\mathbf{L} = \mathbf{D} - \mathbf{W}$	graph **Laplacian** matrix

$$\mathbf{L} = \begin{pmatrix} 4 & -1 & 0 & -1 & -2 \\ -1 & 8 & -3 & -4 & 0 \\ 0 & -3 & 5 & -2 & 0 \\ -1 & -4 & -2 & 12 & -5 \\ -2 & 0 & 0 & -5 & 7 \end{pmatrix}$$



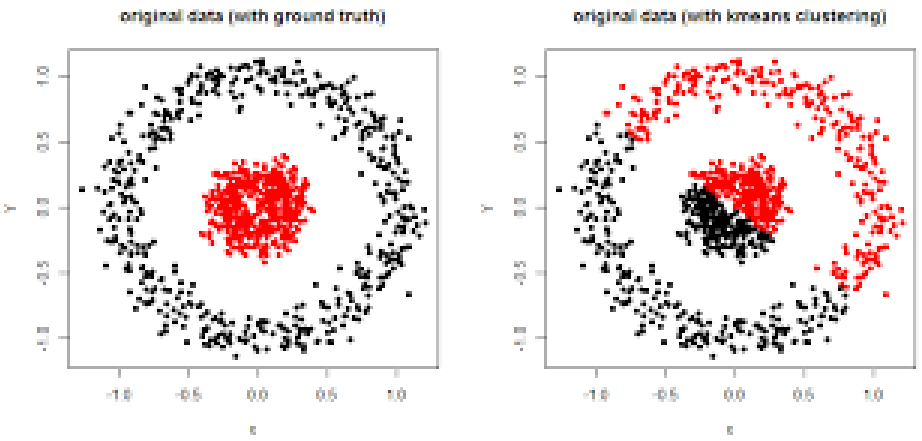
Spectral Clustering*

- Step 3: Construct the eigen-decomposition of $\mathbf{L}$ such that

$$\mathbf{L} = \mathbf{U}\Lambda\mathbf{U}^T$$

where $\mathbf{U^T U} = \mathbf{U U^T} = \mathbf{I_N}$ and $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N)$ where λ_i in increasing order, ie. $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N$.[How different from PCA?]

- Step 4: Choose the first r columns $\mathbf{U}_r$ from $\mathbf{U}$, here $\mathbf{U}_r$ is $N \times r$ matrix
- Step 5: Run k-Means on the data from all the rows of $\mathbf{U}_r$



Example: Spectral Clustering*

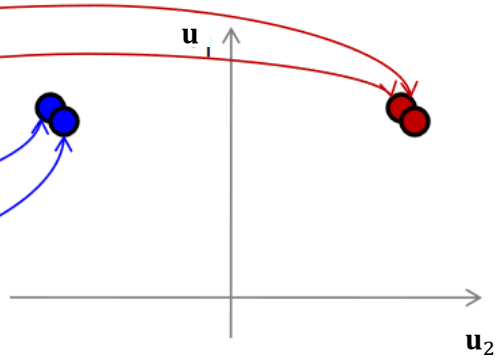
- Embed data points into blocks using eigenvectors:

1	1	.2	0
1	1	0	.1
.2	0	1	1
0	.1	1	1

W

.50	.47
.50	.52
.50	-.47
.50	-.52

$\mathbf{U}_2$



- Embedding is same regardless of data ordering:

1	.2	1	0
.2	0	1	1
1	1	0	.1
0	1	.1	1

W

.50	.47
.50	-.47
.50	.52
.50	-.52

$\mathbf{U}_2$