SIRCSS – Machine Learning for Social Science

Lecture 2

Martin Arvidsson | Institute of Analytical Sociology, Linköping University

2025-12-02

Recap of yesterday (pt.1)

- Supervised learning
  - Learn a function $\hat{f}$ that maps input X to output Y

Recap of yesterday (pt.1)

- Supervised learning
  - Learn a function $\hat{f}$ that maps input X to output Y
- Two types of objectives: prediction vs. inference
  - $\hat{y}$ focus—predict as good as possible
  - $\hat{\beta}$ focus—understand the relation between X and y

Recap of yesterday (pt.1)

- Supervised learning
  - Learn a function $\hat{f}$ that maps input X to output Y
- Two types of objectives: prediction vs. inference
  - $\hat{y}$ focus—predict as good as possible
  - $\hat{\beta}$ focus—understand the relation between X and y
- Limitations of standard linear paradigm
  - *Non-linearity*: standard linear models *underfit* data
  - *High-dimensionality*: standard linear models *overfit* data

Recap of yesterday (pt.1)

- Supervised learning
  - Learn a function $\hat{f}$ that maps input X to output Y
- Two types of objectives: prediction vs. inference
  - $\hat{y}$ focus—predict as good as possible
  - $\hat{\beta}$ focus—understand the relation between X and y
- Limitations of standard linear paradigm
  - *Non-linearity*: standard linear models *underfit* data
  - *High-dimensionality*: standard linear models *overfit* data
- This can be understood in terms of *Bias-Variance Trade-off*
  - *High bias*: $\hat{f}$ not capturing the underlying patterns in the data
  - *High variance*: $\hat{f}$ capturing noise in data and therefore being sensitive to the data it is estimated on

Recap of yesterday (pt.1)

- Supervised learning
  - Learn a function $\hat{f}$ that maps input X to output Y
- Two types of objectives: prediction vs. inference
  - $\hat{y}$ focus—predict as good as possible
  - $\hat{\beta}$ focus—understand the relation between X and y
- Limitations of standard linear paradigm
  - *Non-linearity*: standard linear models *underfit* data
  - *High-dimensionality*: standard linear models *overfit* data
- This can be understood in terms of *Bias-Variance Trade-off*
  - *High bias*: $\hat{f}$ not capturing the underlying patterns in the data
  - *High variance*: $\hat{f}$ capturing noise in data and therefore being sensitive to the data it is estimated on
- Beyond the standard linear model
  - *Polynomials*—enable $\hat{f}$ to pick up more patterns in data ($\downarrow$bias)
  - *Ridge*—penalizes $\hat{f}$ that picks up too much patterns ($\downarrow$variance)
  - *Cross-validation*—helps find a good level of complexity; balancing bias- and variance trade-off

Recap of yesterday (pt.2)

- Non-parameteric supervised learning
  - Parametric: makes strong assumptions about $\hat{f}$ (e.g., additive)
  - Non-parametric: does not make explicit assumptions about parametric form or $\hat{f}$

Recap of yesterday (pt.2)

- Non-parameteric supervised learning
  - Parametric: makes strong assumptions about $\hat{f}$ (e.g., additive)
  - Non-parametric: does not make explicit assumptions about parametric form or $\hat{f}$
- KNN — Predict $\hat{y}_i$ based on closest k training obs to i.

Recap of yesterday (pt.2)

- Non-parameteric supervised learning
  - Parametric: makes strong assumptions about $\hat{f}$ (e.g., additive)
  - Non-parametric: does not make explicit assumptions about parametric form or $\hat{f}$
- KNN — Predict $\hat{y}_i$ based on closest k training obs to i.
- Decision trees
  - Partition X space to create homogeneous y regions.
  - *Recursive binary splitting*: nested if-else statements using variables that, at each step, increase homogeneity in y the most.
  - Predict $\hat{y}_i$ based on which region/leaf i belongs.

Recap of yesterday (pt.2)

- Non-parameteric supervised learning
  - Parametric: makes strong assumptions about $\hat{f}$ (e.g., additive)
  - Non-parametric: does not make explicit assumptions about parametric form or $\hat{f}$
- KNN — Predict $\hat{y}_i$ based on closest k training obs to i.
- Decision trees
  - Partition X space to create homogeneous y regions.
  - *Recursive binary splitting*: nested if-else statements using variables that, at each step, increase homogeneity in y the most.
  - Predict $\hat{y}_i$ based on which region/leaf i belongs.
- Bagging, Random forest
  - Bootstrap original data, estimate decision trees in each bootstrap sample, and average predictions across trees.
  - Random forest extends bagging by decorrelating the trees.

Recap of yesterday (pt.2)

- Non-parameteric supervised learning
  - Parametric: makes strong assumptions about $\hat{f}$ (e.g., additive)
  - Non-parametric: does not make explicit assumptions about parametric form or $\hat{f}$
- KNN — Predict $\hat{y}_i$ based on closest k training obs to i.
- Decision trees
  - Partition X space to create homogeneous y regions.
  - *Recursive binary splitting*: nested if-else statements using variables that, at each step, increase homogeneity in y the most.
  - Predict $\hat{y}_i$ based on which region/leaf i belongs.
- Bagging, Random forest
  - Bootstrap original data, estimate decision trees in each bootstrap sample, and average predictions across trees.
  - Random forest extends bagging by decorrelating the trees.
- Interpretable machine learning

Today

- Deep learning
  - The basics
  - Convolutional neural networks
- If time permits, a brief detour: unsupervised learning

Deep learning

Background: deep learning

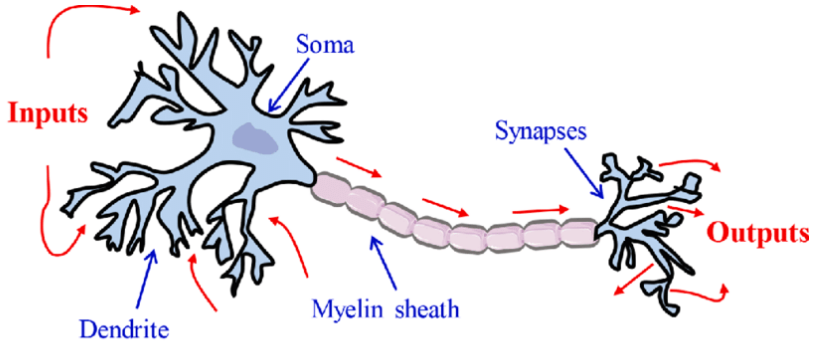
- Responsible for much of the recent sucess and hype about AI.
  - Key technology underlying *chat bots*, *self-driving*, etc.

Background: deep learning

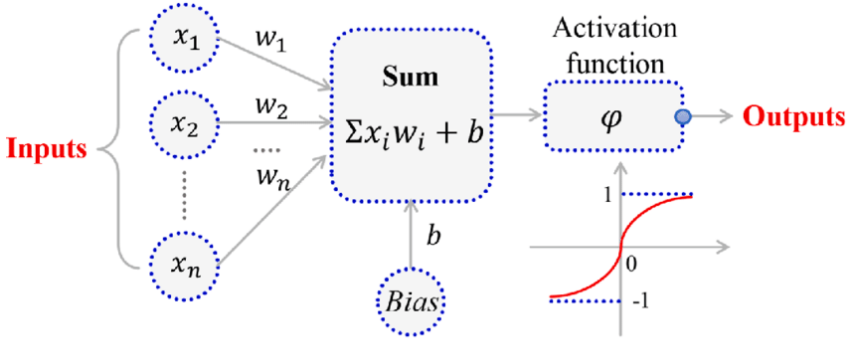
- Responsible for much of the recent sucess and hype about AI.
  - Key technology underlying *chat bots*, *self-driving*, etc.
- New term, old tech? 1950-2010, we called them *"neural networks"*.

Background: deep learning

- Responsible for much of the recent sucess and hype about AI.
  - Key technology underlying *chat bots*, *self-driving*, etc.
- New term, old tech? 1950-2010, we called them *"neural networks"*.
- Early research was inspired by *biological neurons*



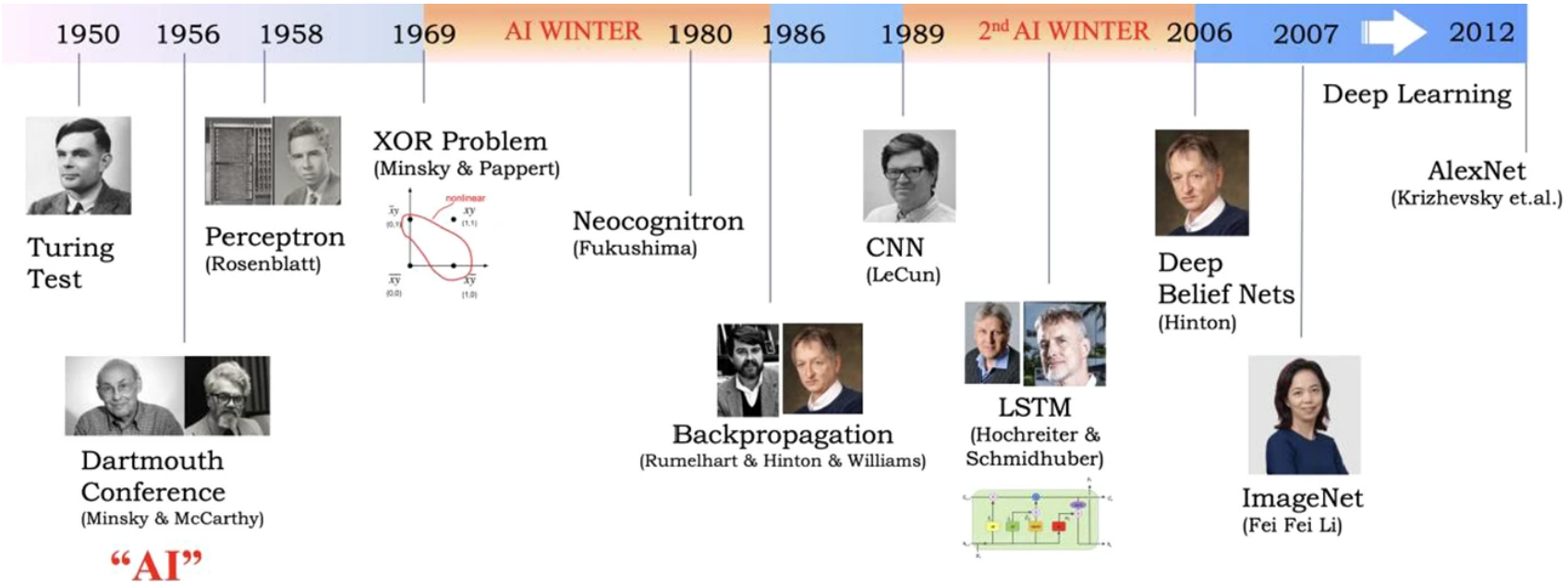
(a) Biological neuron



(b) Artificial neuron

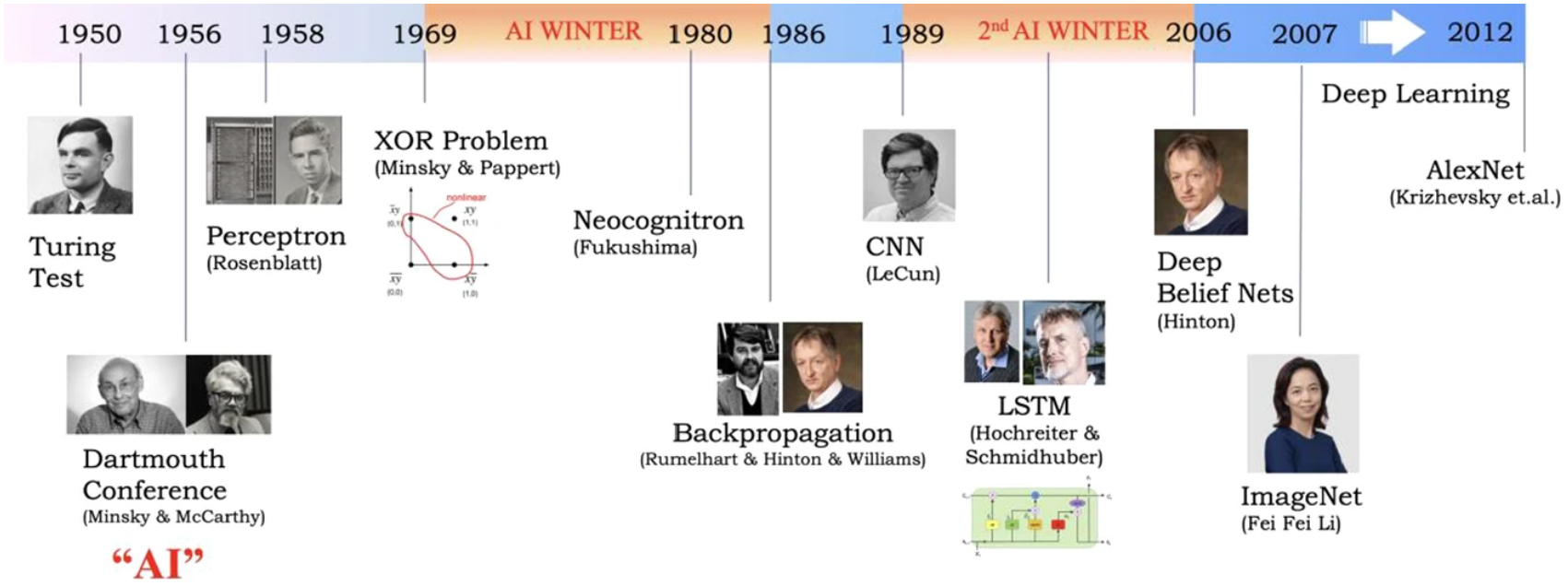
Background: deep learning

- The neural networks timeline (people have been hyped before)



Background: deep learning

- The neural networks timeline (people have been hyped before)



What's different now?

- Massive data and computational power.

Preview: what makes neural networks so powerful?

"Deep learning is a particular kind of machine learning that achieves great power and flexibility by representing the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones." — Ian Goodfellow and Yoshua Bengio (2016)

The importance of representation

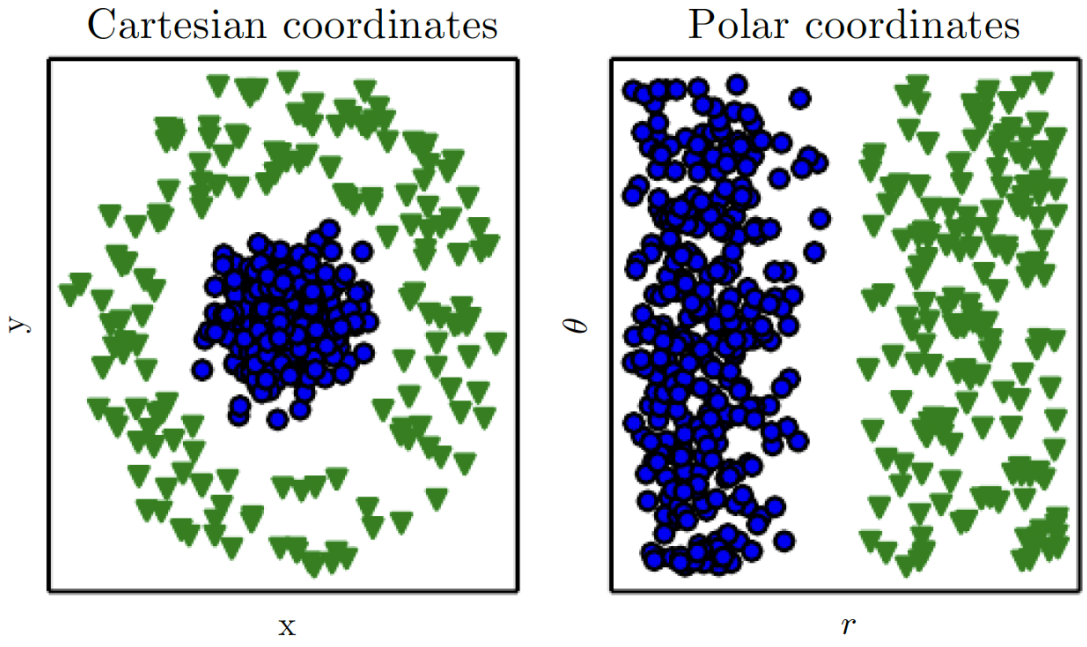


Figure 1.1: Example of different representations: suppose we want to separate two categories of data by drawing a line between them in a scatterplot. In the plot on the left, we represent some data using Cartesian coordinates, and the task is impossible. In the plot on the right, we represent the data with polar coordinates and the task becomes simple to solve with a vertical line. (Figure produced in collaboration with David Warde-Farley.)

Figure 1: Goodfellow and Bengio (2016)

Preview: what makes neural networks so powerful?

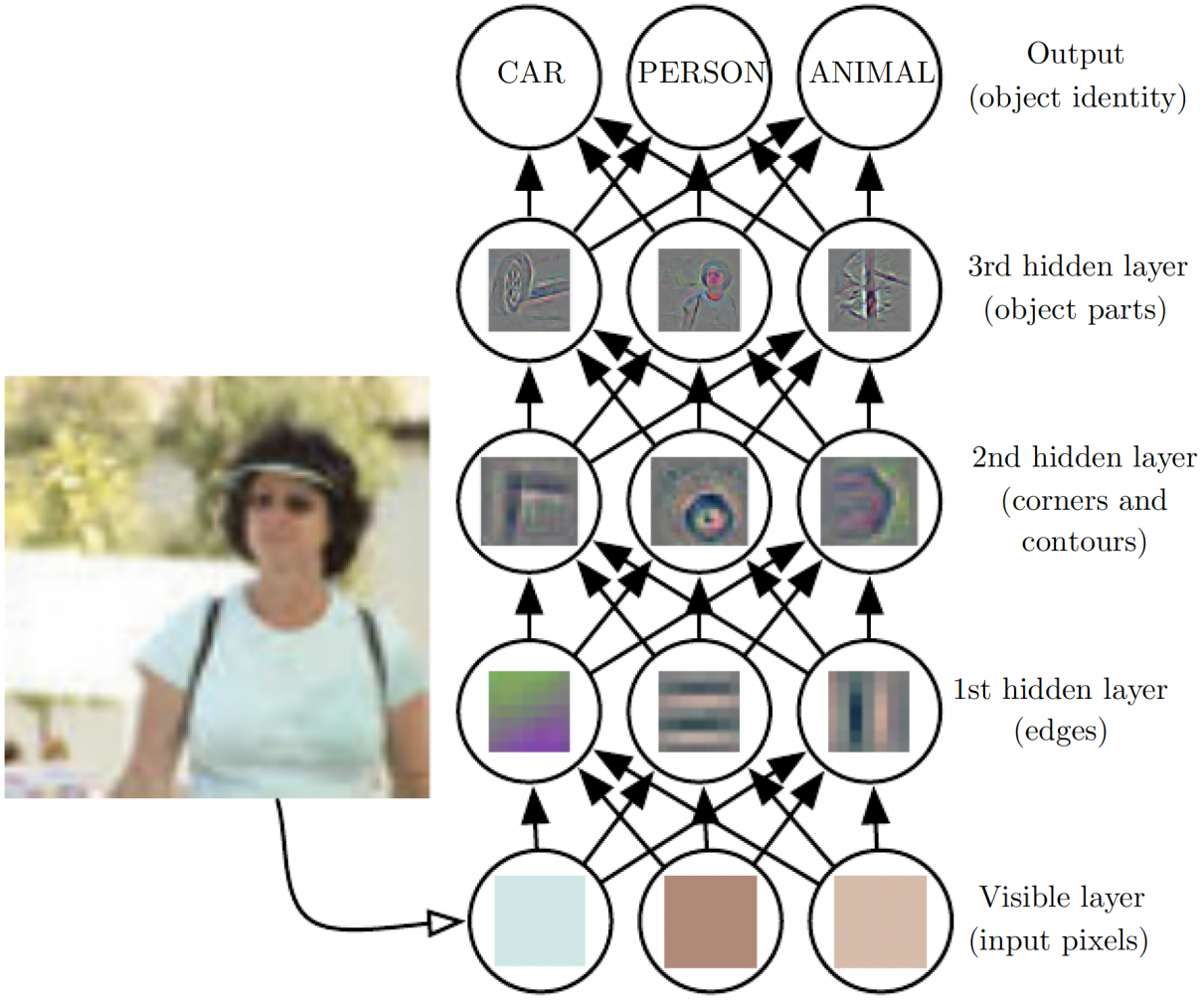


Figure 2: Goodfellow and Bengio (2016)

Contrasting with other stats/ml methods

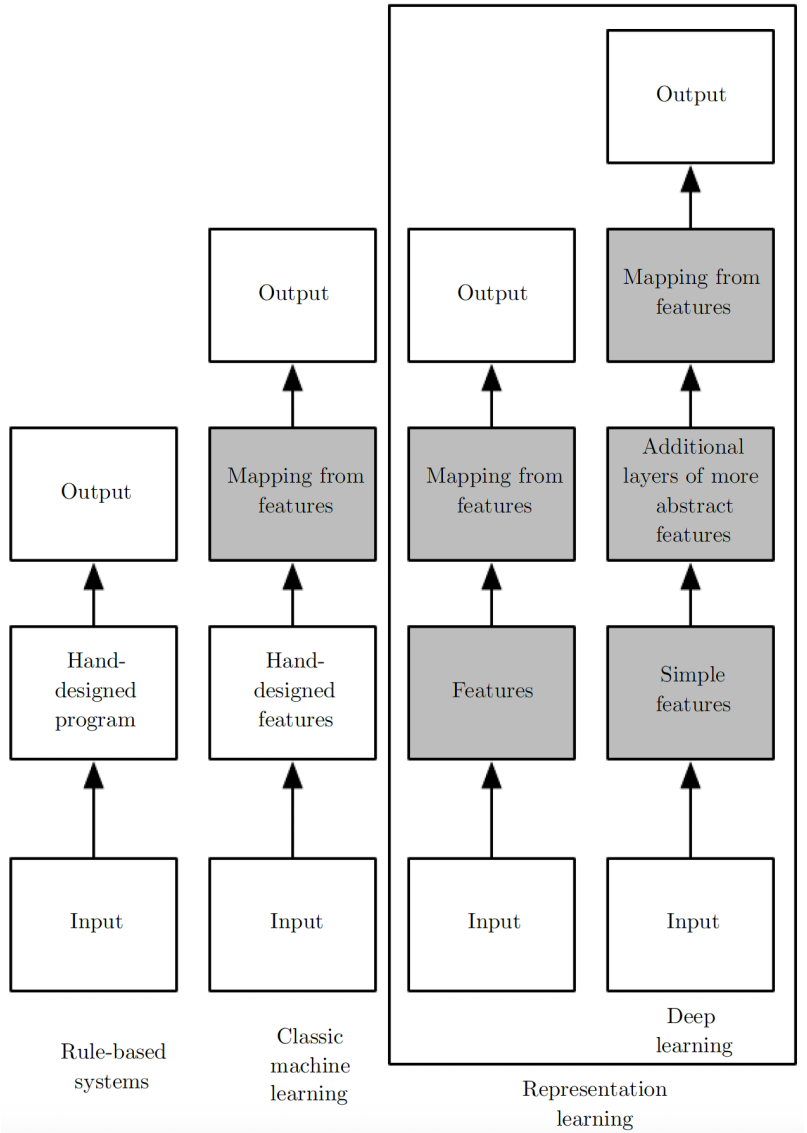


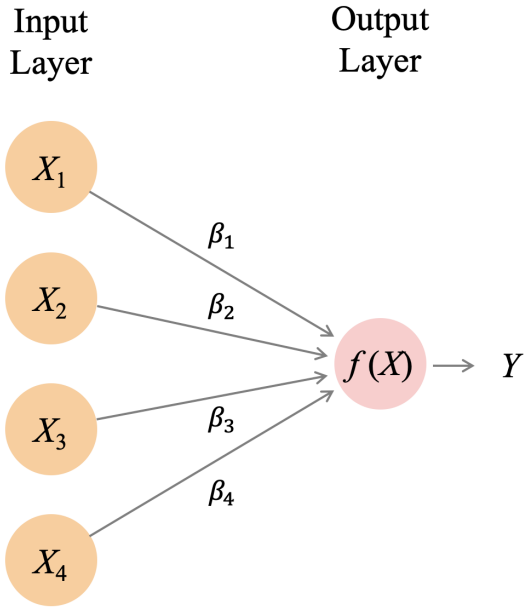
Figure 3: Goodfellow and Bengio (2016)

The basics

The simplest neural network

- Consists of two 'layers'
  - Input layer: X
  - Output layer: Y
- The output layer combines the inputs in a linear combination

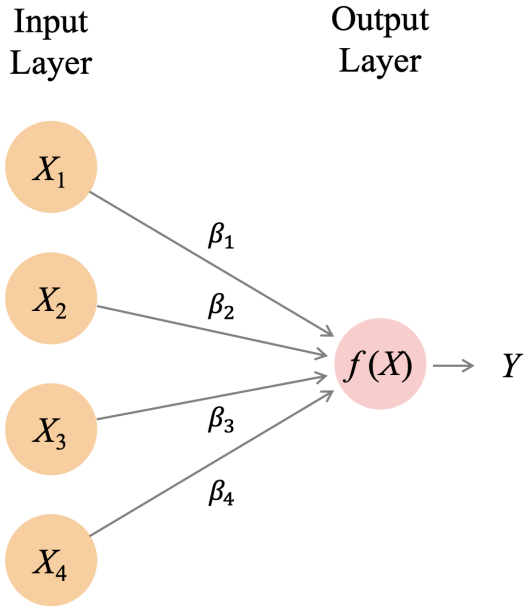
$$f(X) = \sum_{j}^{P} X_j \beta_j$$



The simplest neural network

- Consists of two 'layers'
  - Input layer: X
  - Output layer: Y
- The output layer combines the inputs in a linear combination

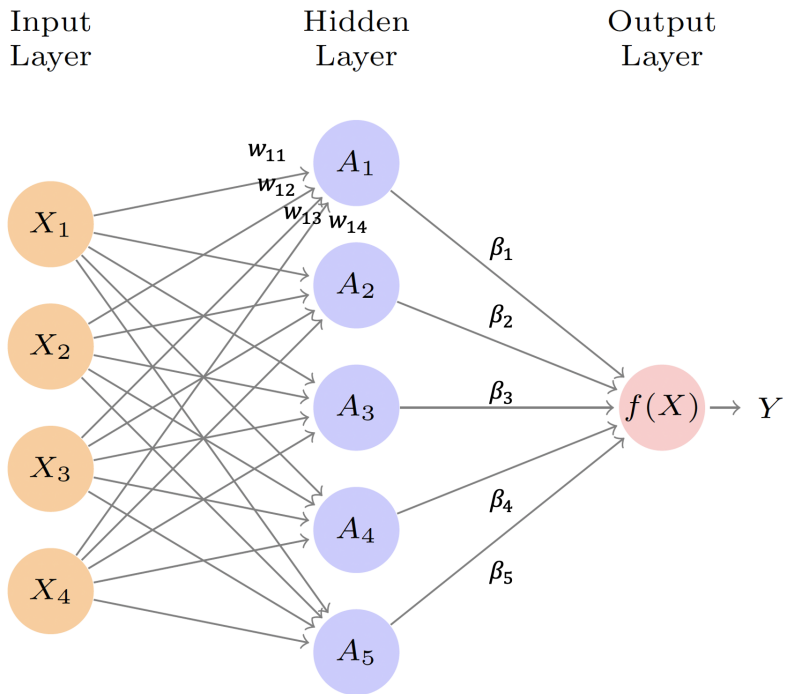
$$f(X) = \sum_{j}^{P} X_j \beta_j$$



Note — this is just a standard linear regression!

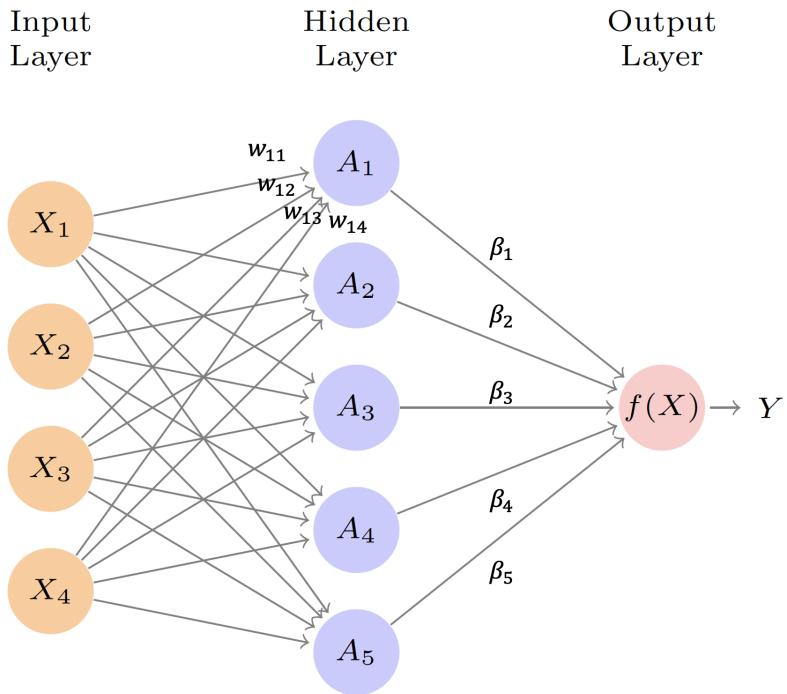
A more complex neural network: adds a 'hidden' layer

- X now feed into the hidden layer



A more complex neural network: adds a 'hidden' layer

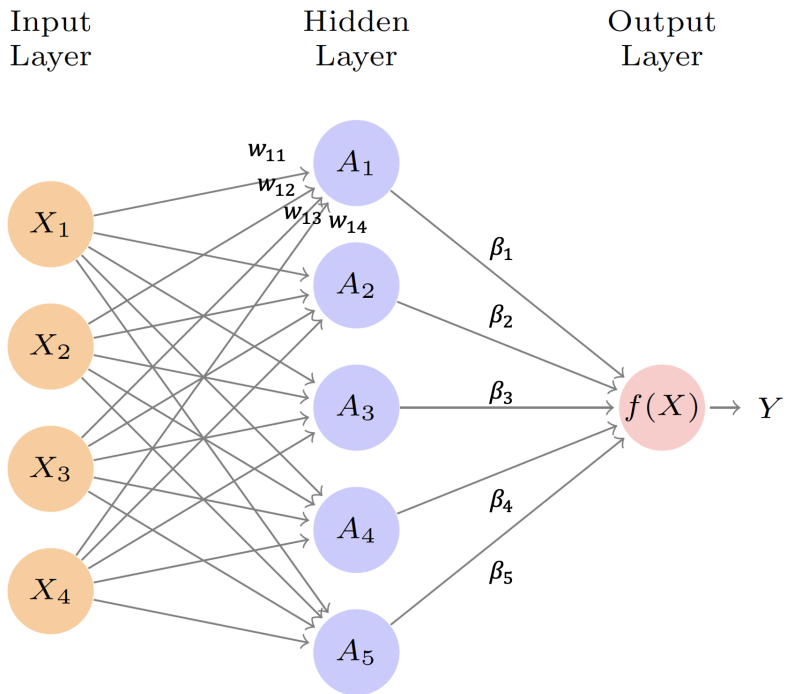
- X now feed into the hidden layer
  - K hidden units A_k



A more complex neural network: adds a 'hidden' layer

- X now feed into the hidden layer
  - K hidden units A_k
  - Each computing a non-linear transformation of a linear combination of X

$$A_k(X) = g(\sum_{j=1}^{p} w_{kj} X_j)$$



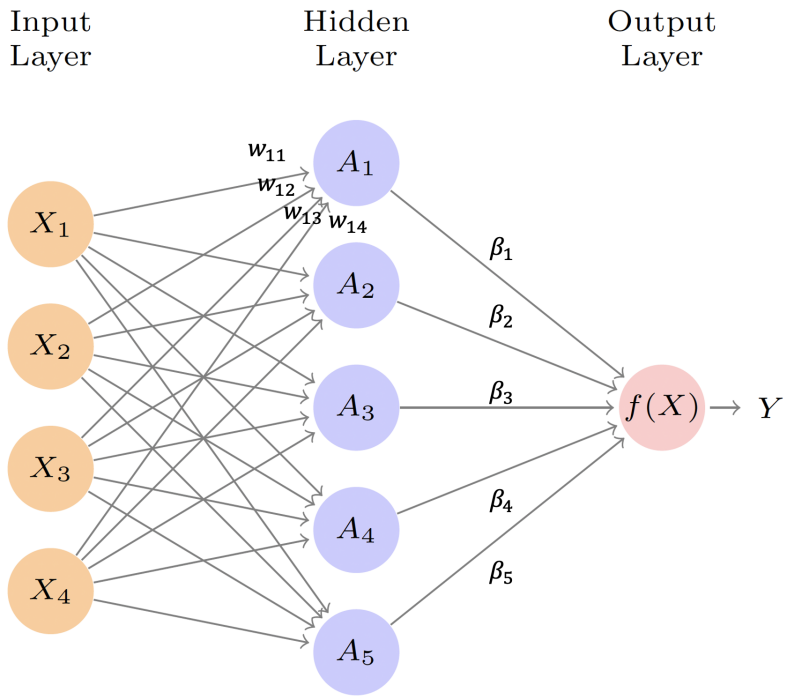
A more complex neural network: adds a 'hidden' layer

- X now feed into the hidden layer
  - K hidden units A_k
  - Each computing a non-linear transformation of a linear combination of X

$$A_k(X) = g\left(\sum_{j=1}^{p} w_{kj} X_j\right)$$

- Output layer then aggregates $A_k(X)$ in linear combination:

$$f(X) = \sum_{k}^{K} \beta_k A_k(X)$$



A more complex neural network: adds a 'hidden' layer

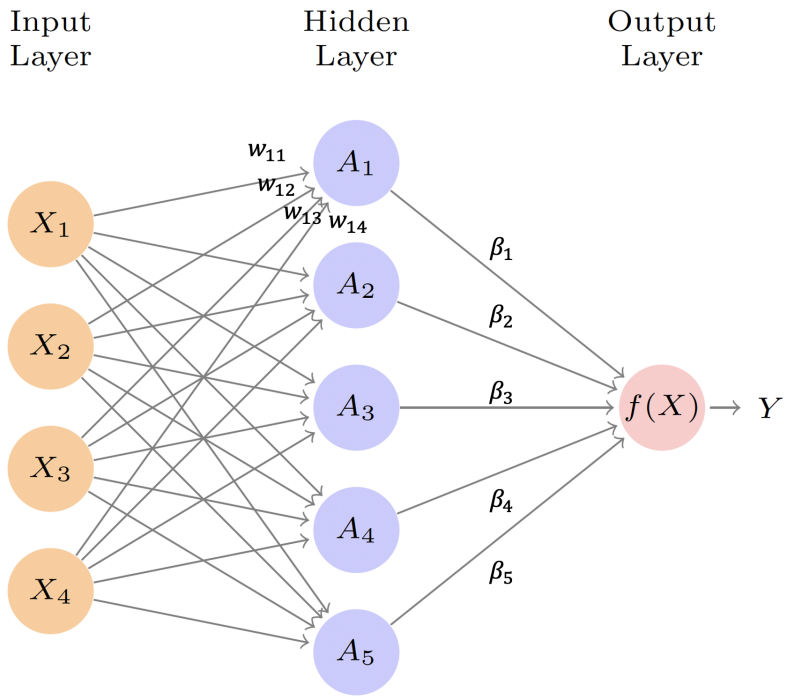
- X now feed into the hidden layer
  - K hidden units A_k
  - Each computing a non-linear transformation of a linear combination of X

$$A_k(X) = g\left(\sum_{j=1}^{p} w_{kj} X_j\right)$$

- Output layer then aggregates $A_k(X)$ in linear combination:

$$f(X) = \sum_{k}^{K} \beta_k A_k(X)$$

- **Goal**: learn β, w minimizing the *loss function* (eg squared loss).



Why is this powerful?

Recall **polynomial functions**

- Extend standard linear model by including non-linear transformations of individual variables X_j of choice: X_j^2, X_j^3 etc.
- Limitations:
  - Transformations (polynomials) of X created prior to learning $\hat{f}$.
  - Does not consider *interactions* between variables.

Why is this powerful?

Recall **polynomial functions**

- Extend standard linear model by including non-linear transformations of individual variables X_j of choice: X_j^2, X_j^3 etc.
- Limitations:
  - Transformations (polynomials) of X created prior to learning $\hat{f}$.
  - Does not consider *interactions* between variables.

Neural networks address both these limitations:

- Learns non-linearity and interactions of X automatically through the hidden layer.

A note on the non-linearity of $g()$

- When combining linear transformations, all weights merge into one slope per input $\rightarrow$ no curves, no interactions.

$$\underbrace{(X_1 + X_2)}_{A_1} + \underbrace{(X_1 + 2X_2)}_{A_2} = 2X_1 + 3X_2$$

A note on the non-linearity of $g()$

- When combining linear transformations, all weights merge into one slope per input $\rightarrow$ no curves, no interactions.

$$\underbrace{(X_1 + X_2)}_{A_1} + \underbrace{(X_1 + 2X_2)}_{A_2} = 2X_1 + 3X_2$$

- When combining non-linear transformations (e.g., squaring) $\rightarrow$ curves and interactions:

$$\underbrace{(X_1 + X_2)}_{A_1}^2 + \underbrace{(X_1 + 2X_2)}_{A_2}^2 = 2X_1^2 + 6X_1X_2 + 5X_2^2$$

A note on the non-linearity of $g()$

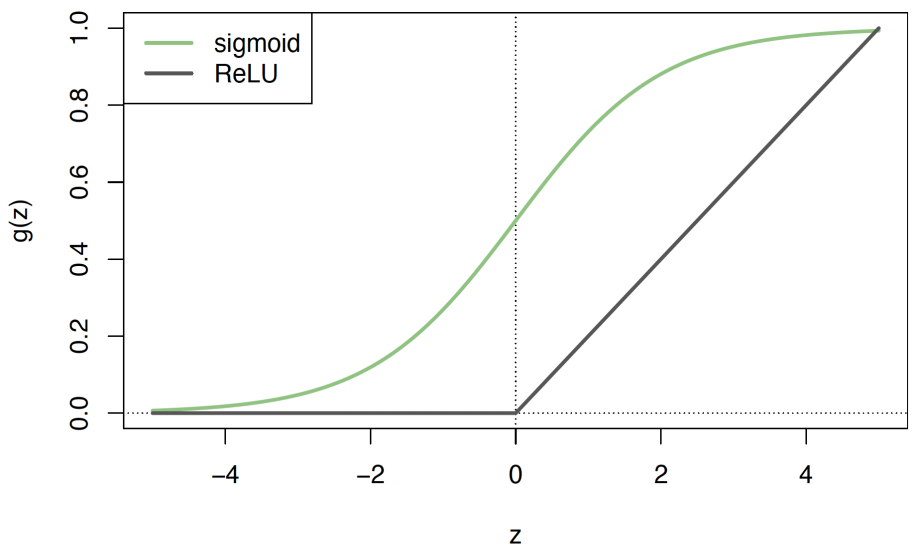
- When combining linear transformations, all weights merge into one slope per input $\rightarrow$ no curves, no interactions.

$$\underbrace{(X_1 + X_2)}_{A_1} + \underbrace{(X_1 + 2X_2)}_{A_2} = 2X_1 + 3X_2$$

- When combining non-linear transformations (e.g., squaring) $\rightarrow$ curves and interactions:

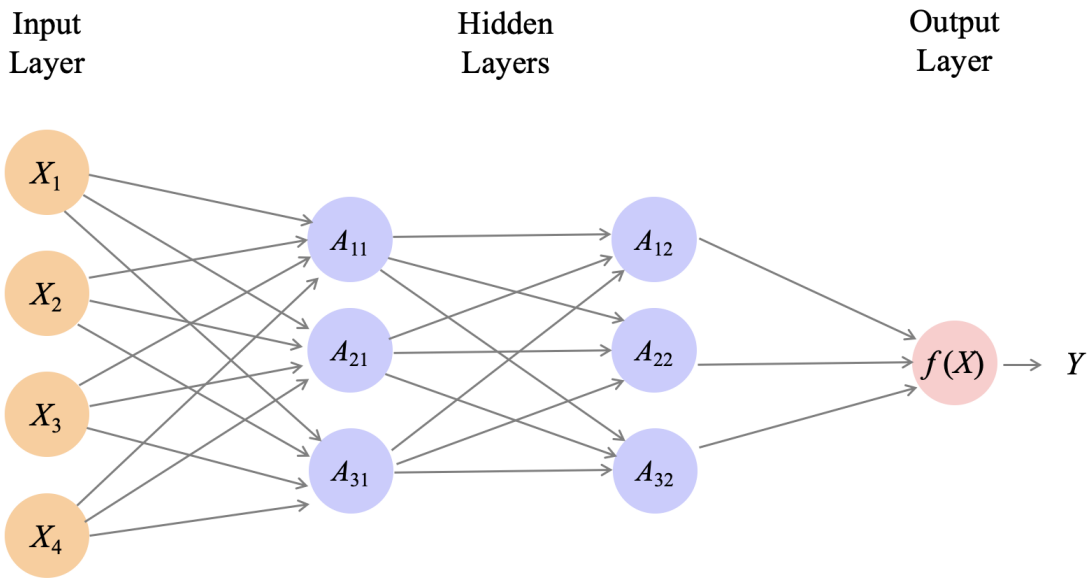
$$\underbrace{(X_1 + X_2)}_{A_1}^2 + \underbrace{(X_1 + 2X_2)}_{A_2}^2 = 2X_1^2 + 6X_1X_2 + 5X_2^2$$

Popular transformations:



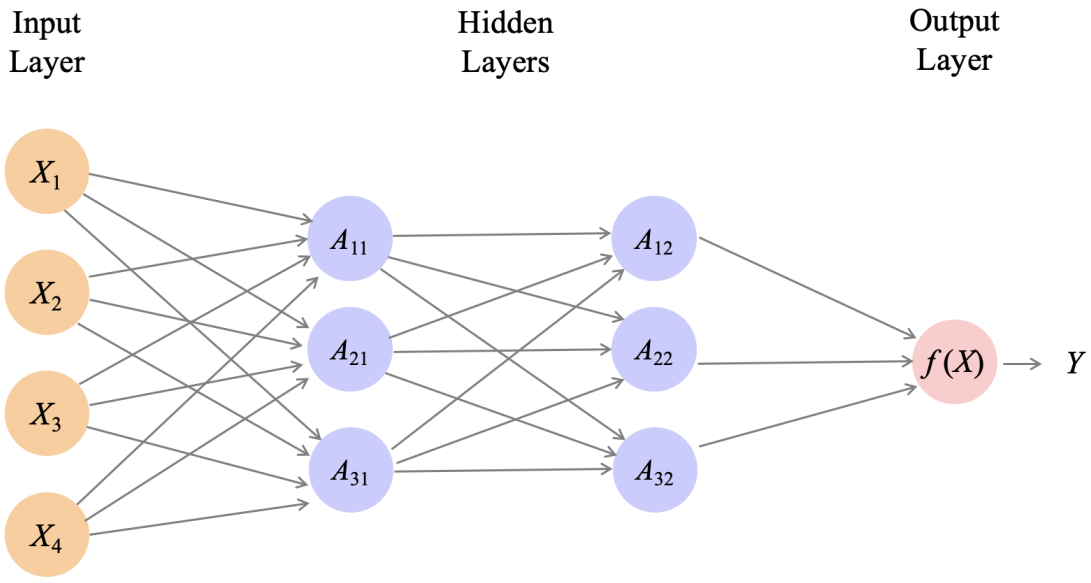
Can we have multiple hidden layers? Yes!

- Modern neural networks typically have *more than one hidden layer*.
- Can learn richer representations, more effectively.



Can we have multiple hidden layers? Yes!

- Modern neural networks typically have *more than one hidden layer*.
- Can learn richer representations, more effectively.

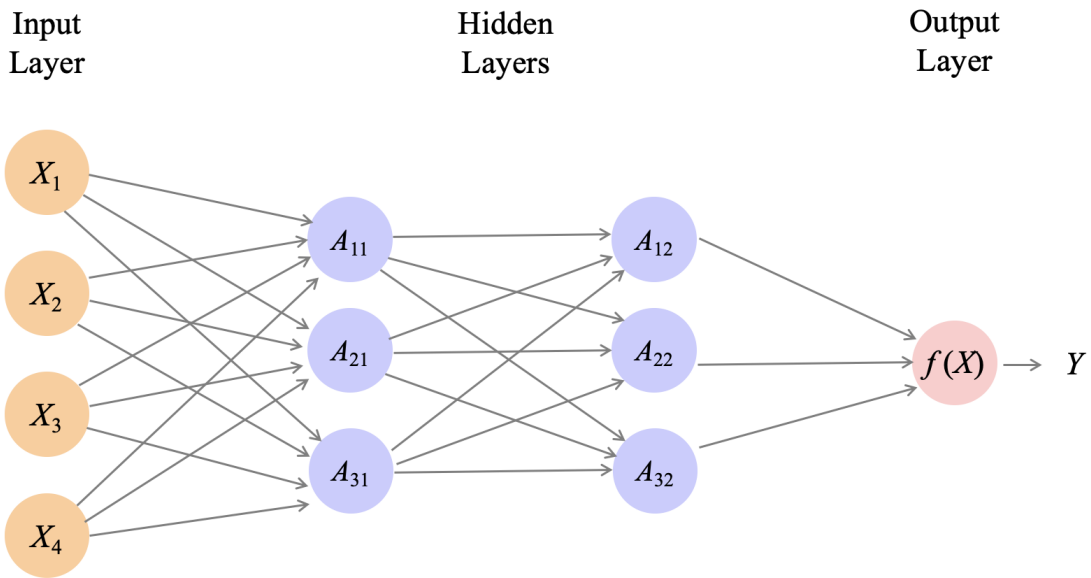


1st hidden layer:

$$A_k^{(1)} = g(\sum_j^p w_{kj}^{(1)} X_j)$$

Can we have multiple hidden layers? Yes!

- Modern neural networks typically have *more than one hidden layer*.
- Can learn richer representations, more effectively.



1st hidden layer:

$$A_k^{(1)} = g(\sum_j^p w_{kj}^{(1)} X_j)$$

2nd hidden layer:

$$A_k^{(2)} = g(\sum_j^p w_{kj}^{(2)} A_k^{(1)})$$

But... many, many parameters?

- In a **standard linear model** — one parameter per variable.
- In a **neural network** — one parameter per variable per hidden unit.

But... many, many parameters?

- In a **standard linear model** — one parameter per variable.
- In a **neural network** — one parameter per variable per hidden unit.

For example, suppose we have a dataset with 100 variables:

- *Standard linear model*: 100 parameters
- *Neural network* (2 hidden layers;50 units): 100*50 + 50*50 = 7,500

But... many, many parameters?

- In a **standard linear model** — one parameter per variable.
- In a **neural network** — one parameter per variable per hidden unit.

For example, suppose we have a dataset with 100 variables:

- *Standard linear model*: 100 parameters
- *Neural network* (2 hidden layers;50 units): 100*50 + 50*50 = 7,500

Recall: more parameters → more *flexibility*, but also risk of *overfitting*.

But... many, many parameters?

- In a **standard linear model** — one parameter per variable.
- In a **neural network** — one parameter per variable per hidden unit.

For example, suppose we have a dataset with 100 variables:

- *Standard linear model*: 100 parameters
- *Neural network* (2 hidden layers;50 units): 100*50 + 50*50 = 7,500

Recall: more parameters → more *flexibility*, but also risk of *overfitting*.

For neural networks — several approaches for addressing overfitting.

Three common ways to regularize neural networks

Ridge penalty

- Add cost for large weights $\Rightarrow$ shrinks many towards 0.

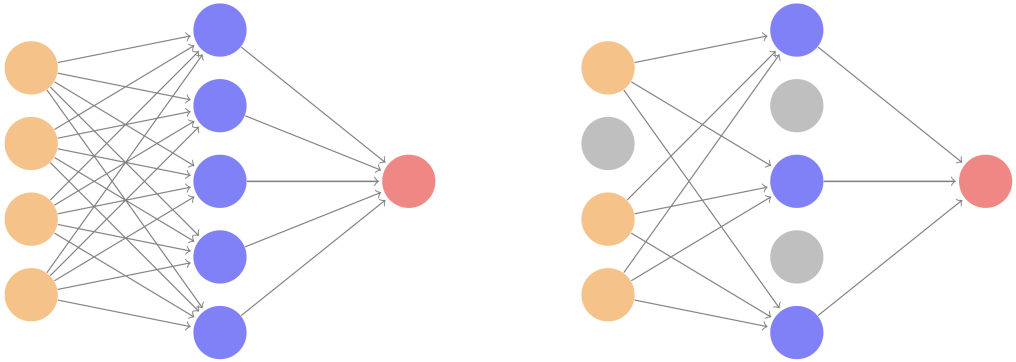
Three common ways to regularize neural networks

Ridge penalty

- Add cost for large weights $\Rightarrow$ shrinks many towards 0.

Dropout learning

- Randomly inactivate some % of nodes *each training step* $\Rightarrow$ forces robustness (not rely on fine-tuned combination of weights).



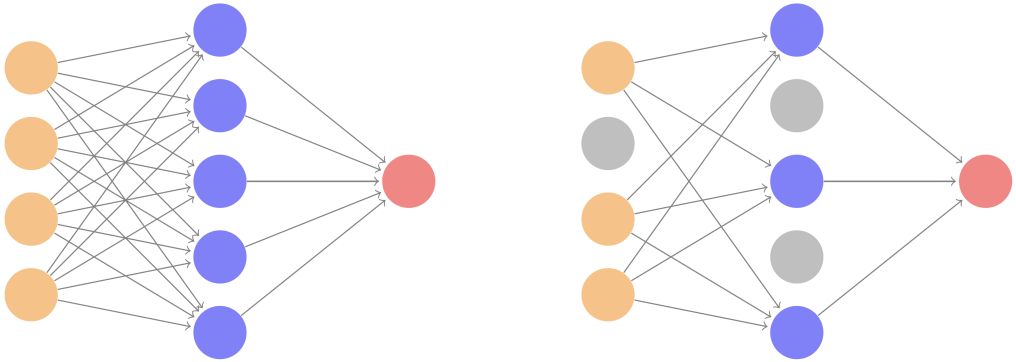
Three common ways to regularize neural networks

Ridge penalty

- Add cost for large weights $\Rightarrow$ shrinks many towards 0.

Dropout learning

- Randomly inactivate some % of nodes *each training step* $\Rightarrow$ forces robustness (not rely on fine-tuned combination of weights).



Early stopping

- Training error improves with the number of optimization rounds.
- Monitor validation loss and stop when it starts to increase.

Example application: predict handwritten digits

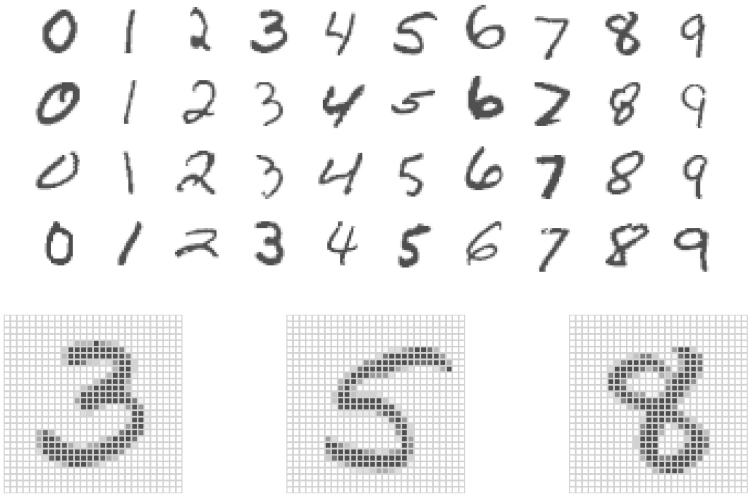


Figure 4: ISL, Fig 10.3

Example application: predict handwritten digits

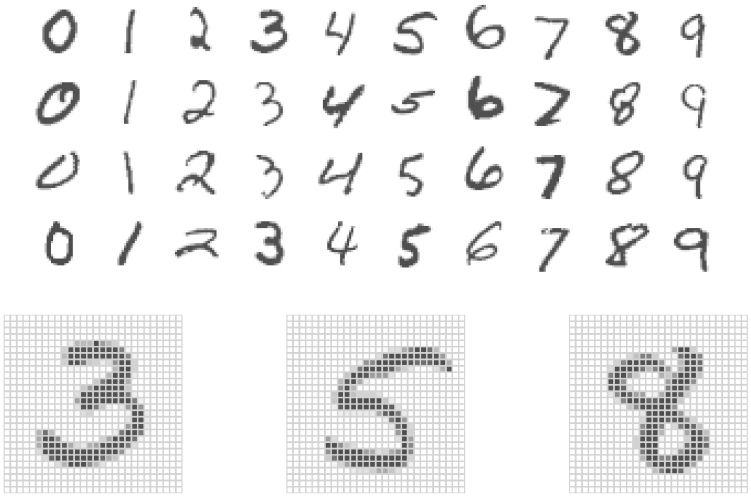


Figure 4: ISL, Fig 10.3

- **Goal**: build model to classify images into their correct class (0–9).

Example application: predict handwritten digits

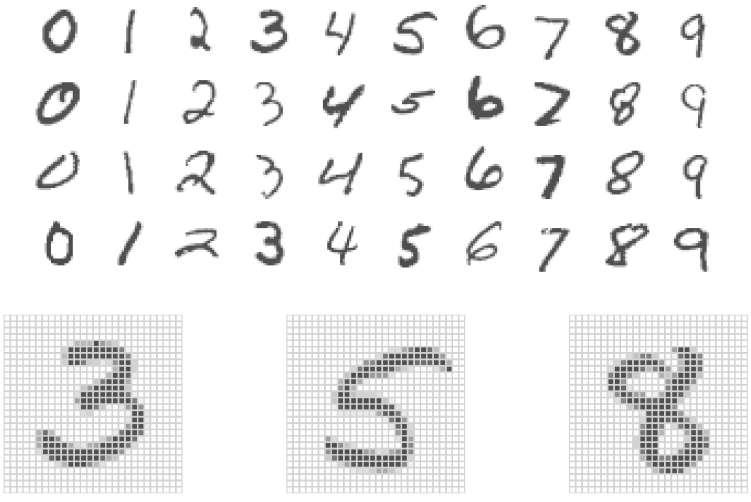


Figure 4: ISL, Fig 10.3

- **Goal**: build model to classify images into their correct class (0–9).
- Every image has 28*28=784 pixels.
- Each pixel is an 8-bit gray scale value between 0-255.

Example application: predict handwritten digits

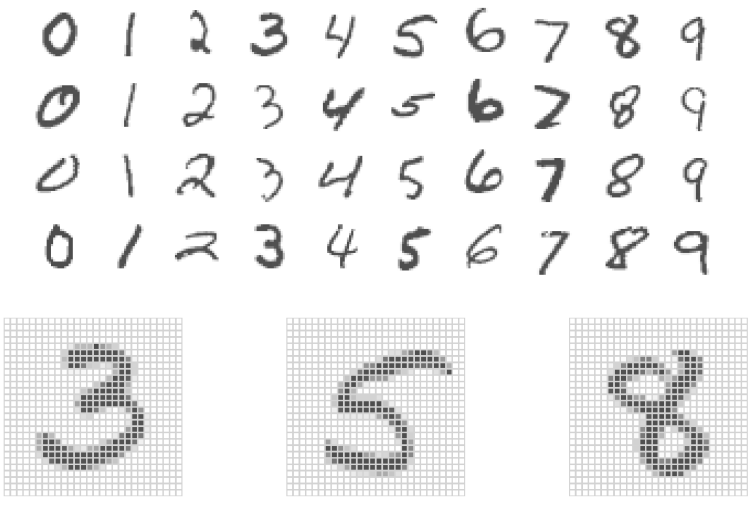


Figure 4: ISL, Fig 10.3

- **Goal**: build model to classify images into their correct class (0–9).
- Every image has 28*28=784 pixels.
- Each pixel is an 8-bit gray scale value between 0-255.
- Thus, X, consist of 784 variables, one for each pixel value.

Example application: predict handwritten digits

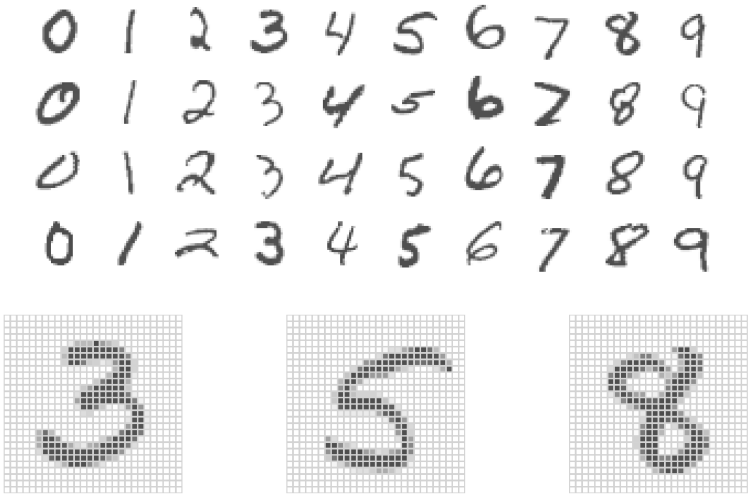


Figure 4: ISL, Fig 10.3

- **Goal**: build model to classify images into their correct class (0–9).
- Every image has 28*28=784 pixels.
- Each pixel is an 8-bit gray scale value between 0-255.
- Thus, X, consist of 784 variables, one for each pixel value.
- Outcome, y, is the class label: represented by 10 dummy variables.

Example application: predict handwritten digits

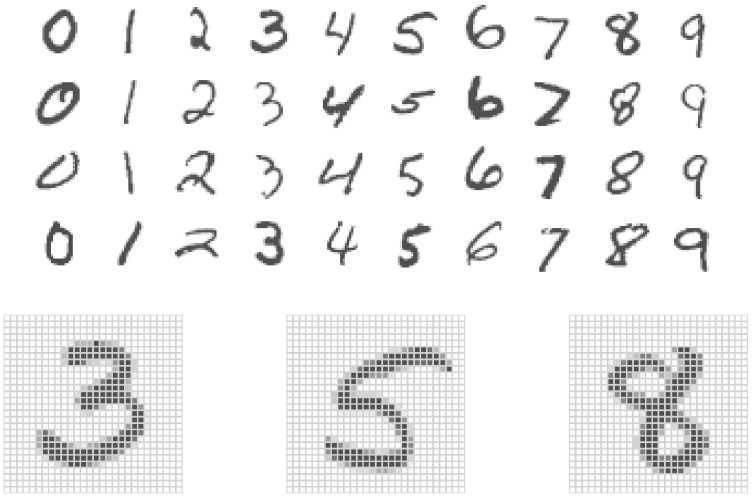


Figure 4: ISL, Fig 10.3

- **Goal**: build model to classify images into their correct class (0–9).
- Every image has 28*28=784 pixels.
- Each pixel is an 8-bit gray scale value between 0-255.
- Thus, X, consist of 784 variables, one for each pixel value.
- Outcome, y, is the class label: represented by 10 dummy variables.
- *Training* set: 50,000 images. *Test* set: 10,000 images.

Example application: predict handwritten digits

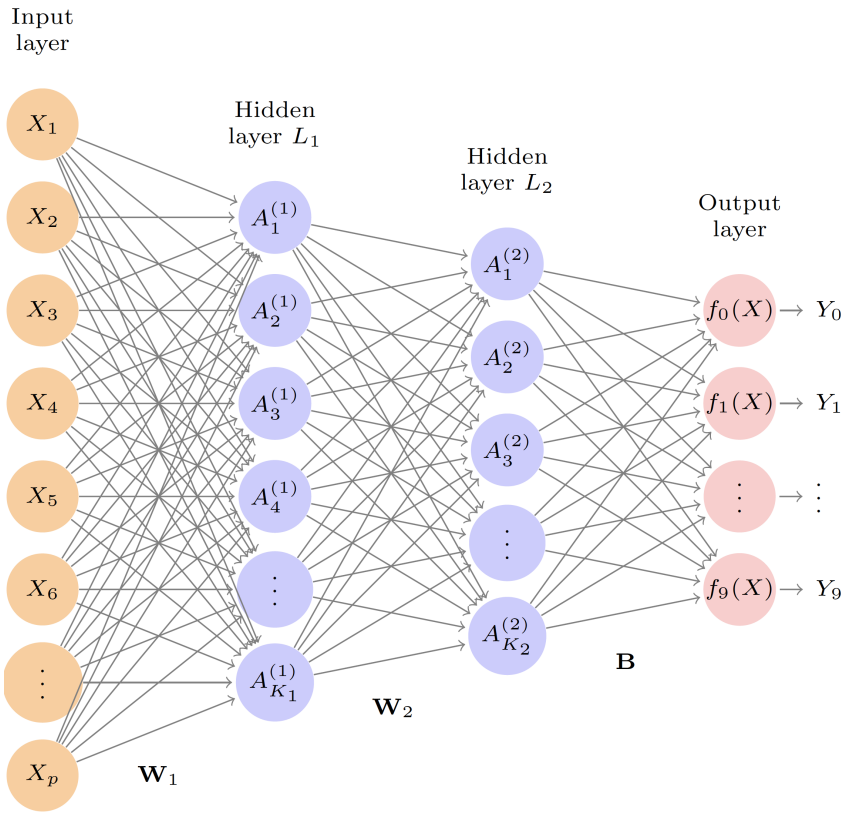


Figure 5: Model estimated in ISL (Fig 10.4)

- Two hidden layers (256, 128 units).
- Implies: (784*256) + (256*128) + (128*10) $\approx$ 235,000 parameters.

Results

Method	Test Error
Neural Network + Ridge Regularization	2.3%
Neural Network + Dropout Regularization	1.8%
Multinomial Logistic Regression	7.2%
Linear Discriminant Analysis	12.7%

Figure 6: ISL, Table 10.1

- Significant improvement over standard linear model(s).
- And — impressive accuracy ($\approx$ 98%)!

Example II: predicting salaries of baseball players

Data: 322 observations of major league players (20 input variables).

Comparing three models:

- Standard linear model: 20 parameters.
- Lasso regression (λ selected via CV): 12 non-zero parameters.
- Neural network with 1 hidden layer (64 units): 1,345 parameters.

Example II: predicting salaries of baseball players

Data: 322 observations of major league players (20 input variables).

Comparing three models:

- Standard linear model: 20 parameters.
- Lasso regression (λ selected via CV): 12 non-zero parameters.
- Neural network with 1 hidden layer (64 units): 1,345 parameters.

Results

Model	# Parameters	Mean Abs. Error	Test Set R^2
Linear Regression	20	254.7	0.56
Lasso	12	252.3	0.51
Neural Network	1345	257.4	0.54

Figure 7: ISL, Table 10.2

- On this (small N, small p) data, we see no improvement.

Considerations when applying neural nets

- **Architecture**
  - Thus far — generic, fully-connected hidden layers.
  - But there exists specialized architectures (e.g., CNNs, RNNs)

Considerations when applying neural nets

- **Architecture**
  - Thus far — generic, fully-connected hidden layers.
  - But there exists specialized architectures (e.g., CNNs, RNNs)
    Specialized architectures is where NN shines.

Considerations when applying neural nets

- **Architecture**
  - Thus far — generic, fully-connected hidden layers.
  - But there exists specialized architectures (e.g., CNNs, RNNs)
    Specialized architectures is where NN shines.
- **Structure of neural net**
  - Number of hidden layers
  - Number of units in each hidden layer
  - Activation function (type of non-linear transformation)

Considerations when applying neural nets

- **Architecture**
  - Thus far — generic, fully-connected hidden layers.
  - But there exists specialized architectures (e.g., CNNs, RNNs)
    Specialized architectures is where NN shines.
- **Structure of neural net**
  - Number of hidden layers
  - Number of units in each hidden layer
  - Activation function (type of non-linear transformation)

    #Hidden layers/units: archiceture-dependent heuristics + grid-search
    Activation-function: ReLU is the standard

Considerations when applying neural nets

- **Architecture**
  - Thus far — generic, fully-connected hidden layers.
  - But there exists specialized architectures (e.g., CNNs, RNNs)
    Specialized architectures is where NN shines.
- **Structure of neural net**
  - Number of hidden layers
  - Number of units in each hidden layer
  - Activation function (type of non-linear transformation)

    #Hidden layers/units: archiceture-dependent heuristics + grid-search
    Activation-function: ReLU is the standard
- **Regularization**
  - Penalization (L1/L2)?
  - Dropout?
  - Early stopping?

Considerations when applying neural nets

- **Architecture**
  - Thus far — generic, fully-connected hidden layers.
  - But there exists specialized architectures (e.g., CNNs, RNNs)
    Specialized architectures is where NN shines.
- **Structure of neural net**
  - Number of hidden layers
  - Number of units in each hidden layer
  - Activation function (type of non-linear transformation)

    #Hidden layers/units: archiceture-dependent heuristics + grid-search
    Activation-function: ReLU is the standard
- **Regularization**
  - Penalization (L1/L2)?
  - Dropout?
  - Early stopping?

    Early stopping almost always implemented

    Other: grid search (in conjunction with #hidden layers/units etc.)

Summary, thus far.

Pros

- Neural networks can learn highly *non-linear* and *complex interactions*
- Especially powerful for 'complex' and 'unstructured' data — where *representation learning* is key.
  - E.g., text, images, video, sound...
  - Rich library of architectures developed for such data.

Summary, thus far.

Pros

- Neural networks can learn highly *non-linear* and *complex interactions*
- Especially powerful for 'complex' and 'unstructured' data — where *representation learning* is key.
  - E.g., text, images, video, sound...
  - Rich library of architectures developed for such data.

Cons

- Require *a lot of data* to learn complex patterns.[1]
- *Many hyperparameters* that the analyst much choose.
- Lacking in *inference* and *interpretability*.

[1] When data is smaller / patterns simpler — usually not competitive.

Convolutional neural networks

Convolutional neural networks

- Class of neural networks developed for image classification.

Convolutional neural networks

- Class of neural networks developed for image classification.
- **Basic idea**:
  1. Identify low-level features in images (edges, patches of colors)
  2. Learn how such features combine into high-level features (ears, eyes)
  3. Learn association between high-level features and outcome variable.

Convolutional neural networks

- Class of neural networks developed for image classification.
- **Basic idea**:
  1. Identify low-level features in images (edges, patches of colors)
  2. Learn how such features combine into high-level features (ears, eyes)
  3. Learn association between high-level features and outcome variable.

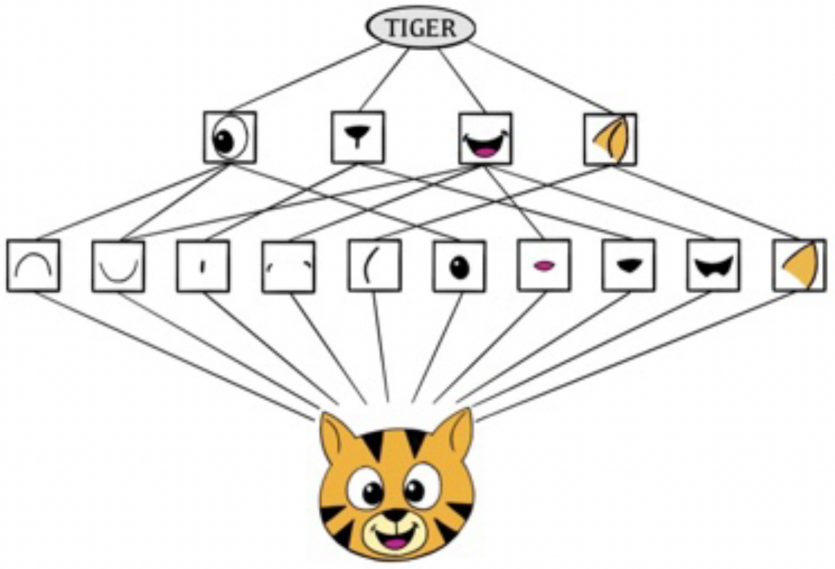


Figure 8: ISL, Fig 10.6

Convolutional neural networks

- Class of neural networks developed for image classification.
- **Basic idea**:
  1. Identify low-level features in images (edges, patches of colors)
  2. Learn how such features combine into high-level features (ears, eyes)
  3. Learn association between high-level features and outcome variable.

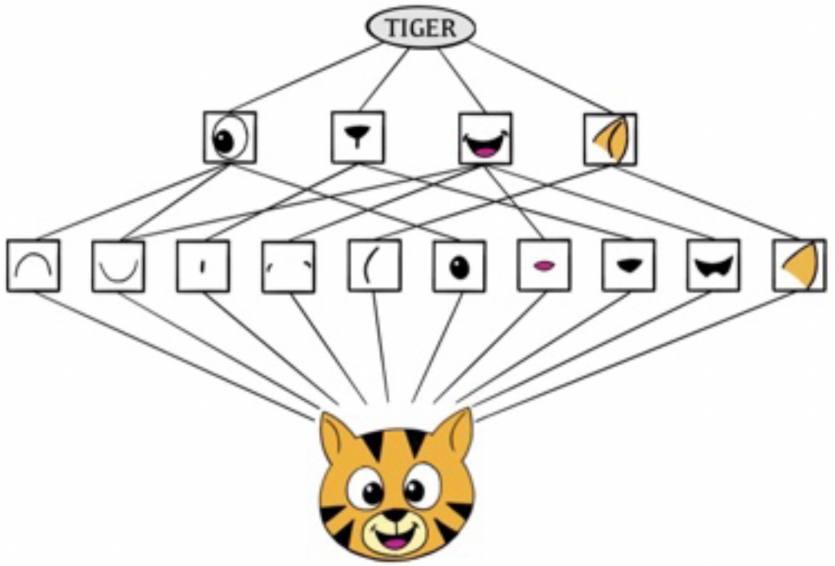


Figure 8: ISL, Fig 10.6

This is achieved by adding a specialized type of hidden layer to the network.

Convolutional layers

- A convolutional layer contains a set of smaller matrices ('filters'), each representing a *particular visual feature*.[2]

[2] 1st layer: simple (eg lines); subsequent layers: combine → complex (eg objects).

Convolutional layers

- A convolutional layer contains a set of smaller matrices ('filters'), each representing a *particular visual feature*.[2]

[2] 1st layer: simple (eg lines); subsequent layers: combine → complex (eg objects).

Convolutional layers

- A convolutional layer contains a set of smaller matrices ('filters'), each representing a *particular visual feature*.[2]
- Filters are slid across the image — at every step, recording correspondence between subregion of image and filter-pattern.[3]

[2] 1st layer: simple (eg lines); subsequent layers: combine → complex (eg objects).
[3] This is achieved by calculating their dot-product.

Convolutional layers

- A convolutional layer contains a set of smaller matrices ('filters'), each representing a *particular visual feature*.[2]
- Filters are slid across the image — at every step, recording correspondence between subregion of image and filter-pattern.[3]

[2] 1st layer: simple (eg lines); subsequent layers: combine → complex (eg objects).
[3] This is achieved by calculating their dot-product.

Convolutional layers

- A convolutional layer contains a set of smaller matrices ('filters'), each representing a *particular visual feature*.[2]
- Filters are slid across the image — at every step, recording correspondence between subregion of image and filter-pattern.[3]
- **Output**: a 'feature map' — identifying regions of image where filter-pattern is salient.

[2] 1st layer: simple (eg lines); subsequent layers: combine → complex (eg objects).
[3] This is achieved by calculating their dot-product.

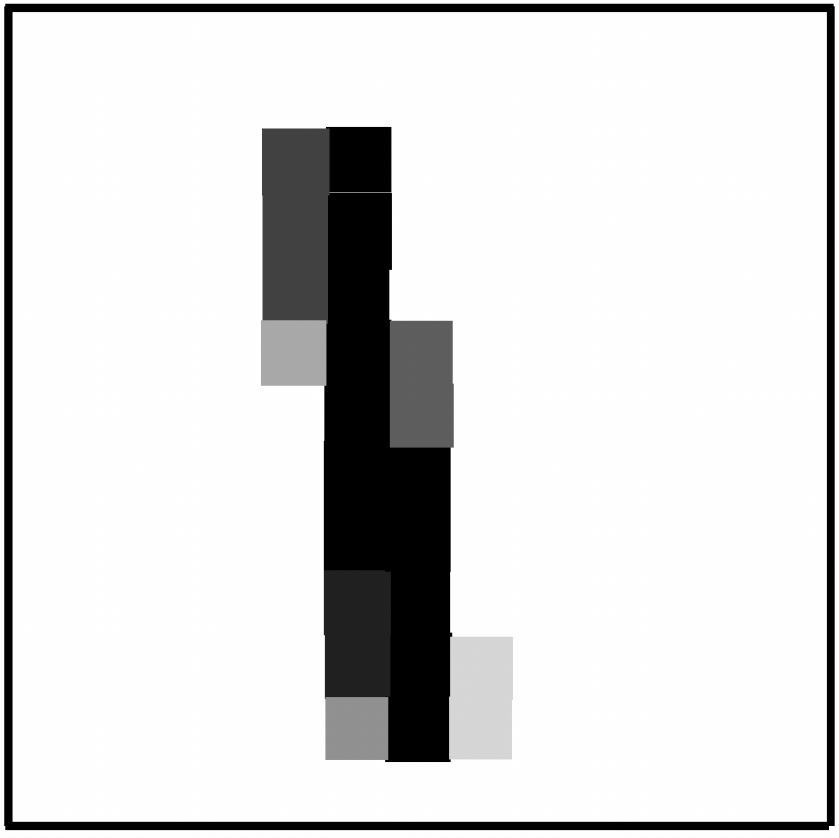
Convolutional layers

- A convolutional layer contains a set of smaller matrices ('filters'), each representing a *particular visual feature*.[2]
- Filters are slid across the image — at every step, recording correspondence between subregion of image and filter-pattern.[3]
- **Output**: a 'feature map' — identifying regions of image where filter-pattern is salient.
- This is repeated for all k filters, producing k feature maps.

[2] 1st layer: simple (eg lines); subsequent layers: combine $\rightarrow$ complex (eg objects).
[3] This is achieved by calculating their dot-product.

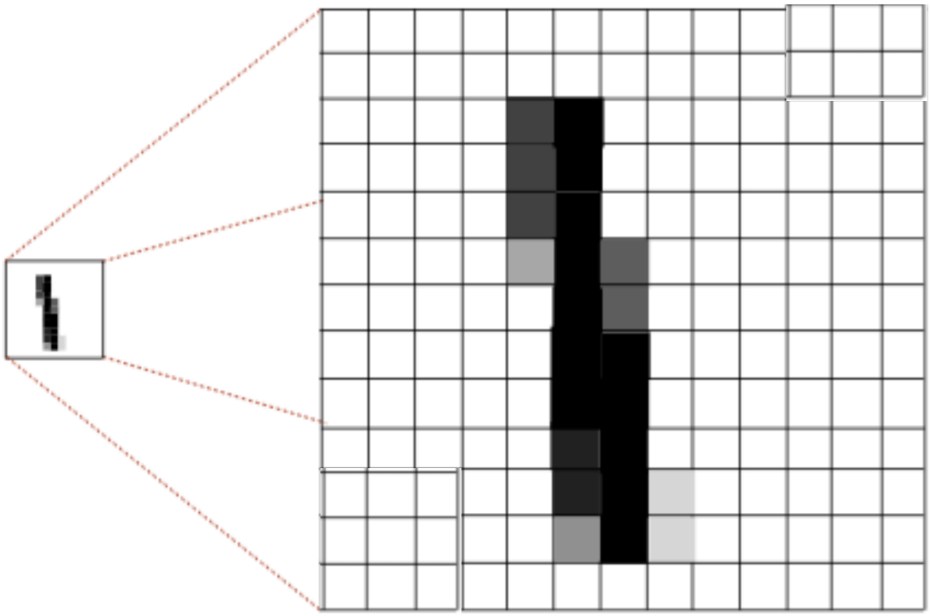
Illustration

Suppose this is our image:



Illustration

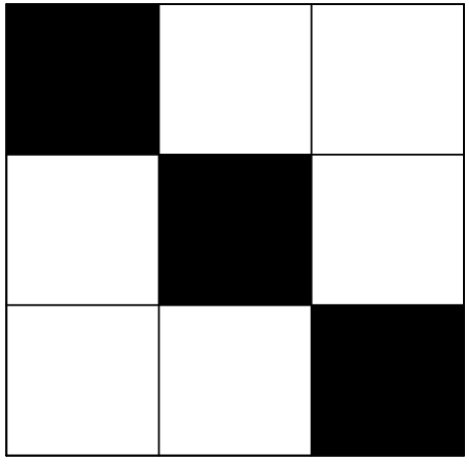
Which we map to a matrix of numbers (gray scale of each pixel)



0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	.6	1	0	0	0	0	0	0	0
0	0	0	0	.6	1	0	0	0	0	0	0	0
0	0	0	0	.6	1	0	0	0	0	0	0	0
0	0	0	0	.3	1	.5	0	0	0	0	0	0
0	0	0	0	0	1	.5	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	0	0
0	0	0	0	.8	.8	1	0	0	0	0	0	0
0	0	0	0	0	.8	1	.2	0	0	0	0	0
0	0	0	0	0	.3	1	.2	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0

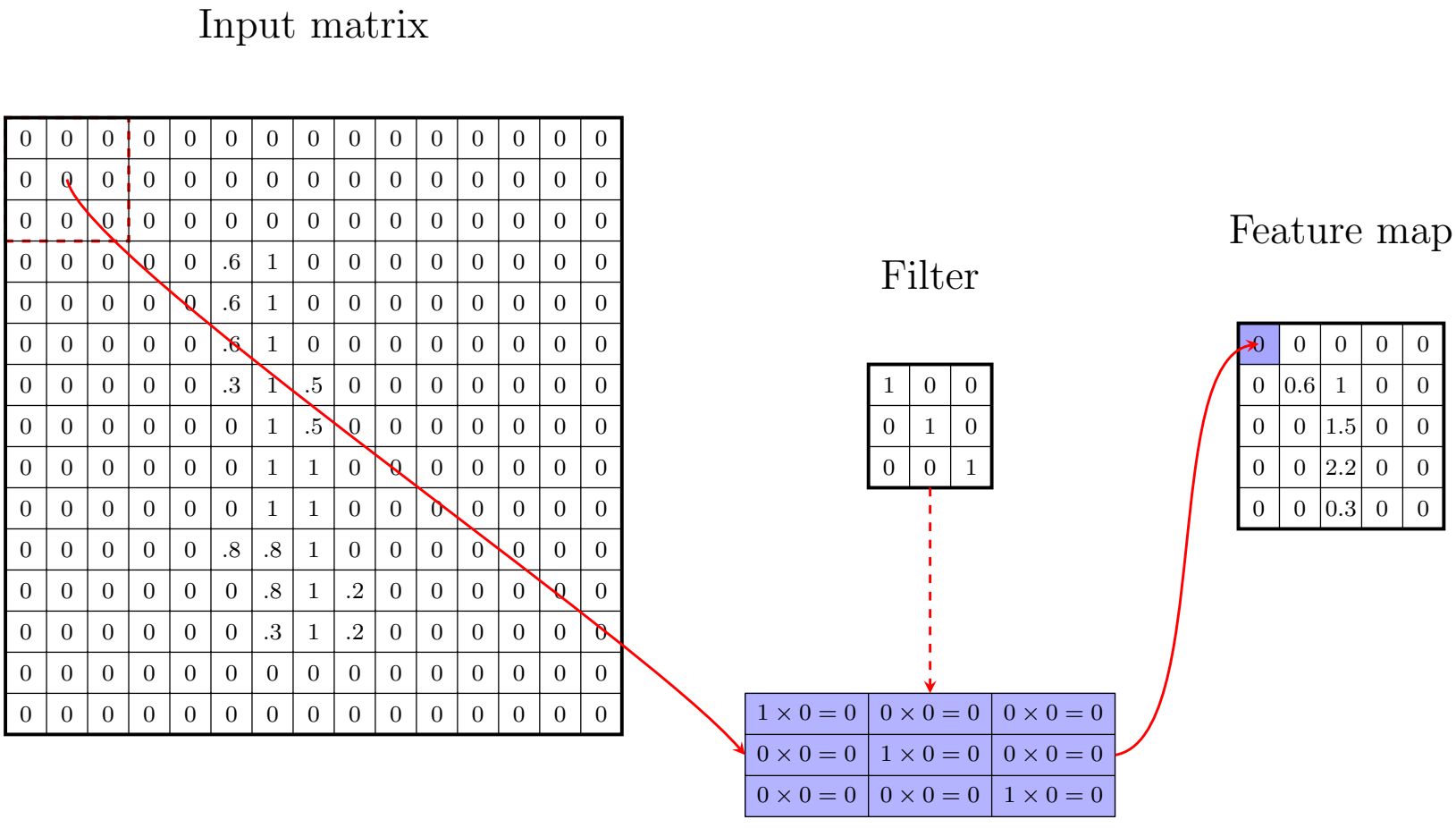
Illustration

And suppose this is our (3*3 sized) filter:



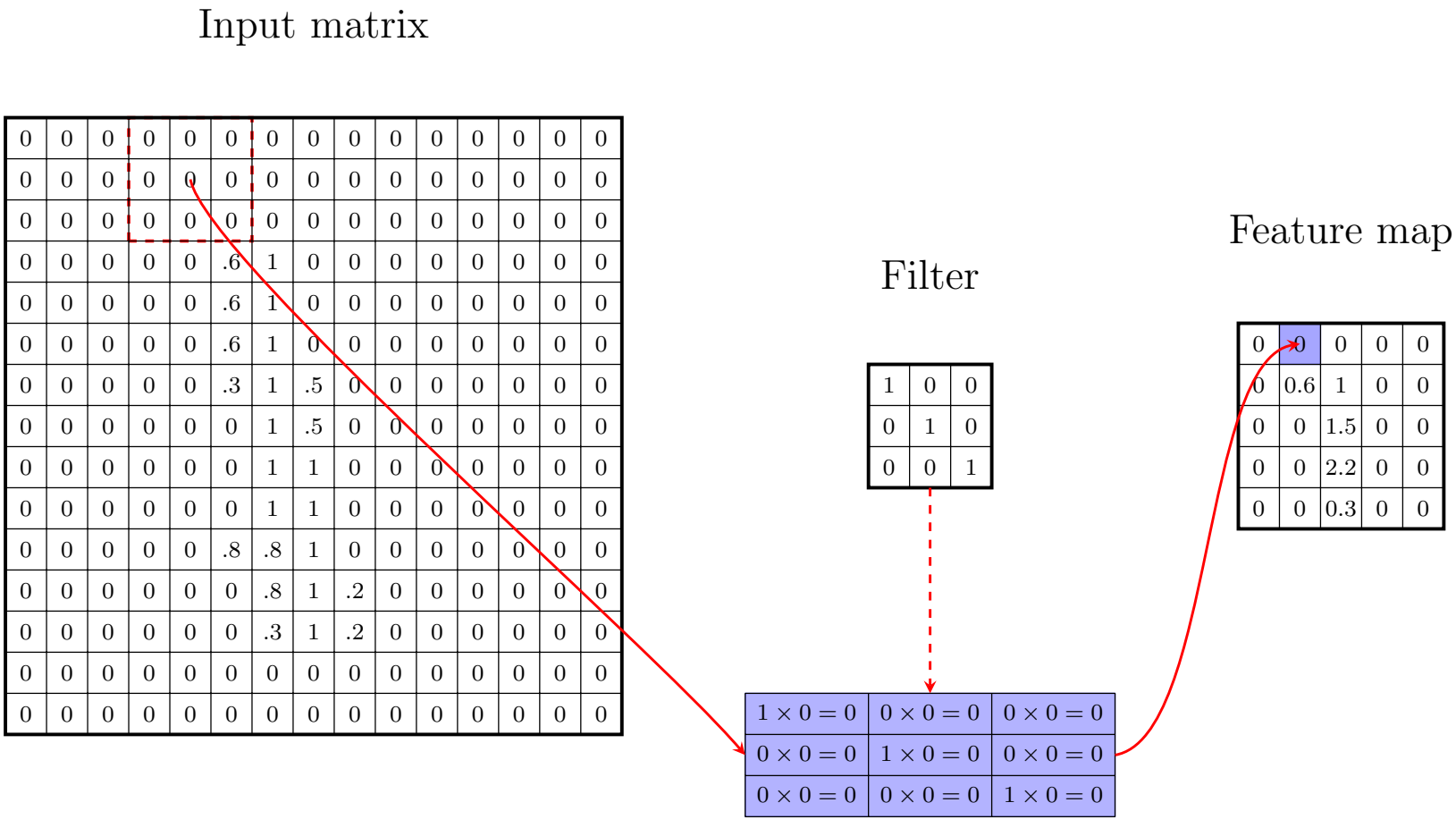
Illustration

We slide the filter through each 3x3 patch of the image — recording the presence of the filter-pattern:



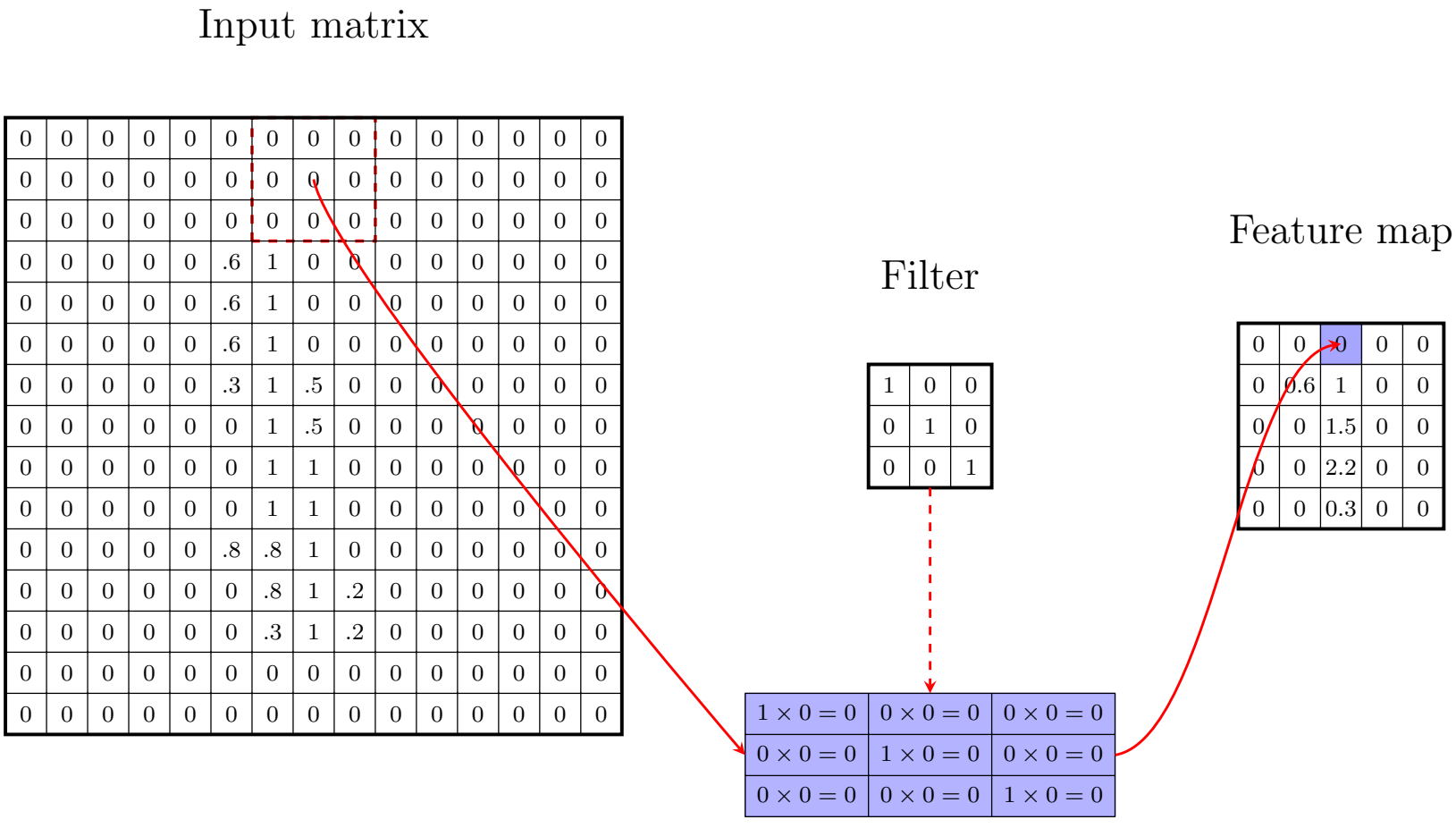
Illustration

We slide the filter through each 3x3 patch of the image — recording the presence of the filter-pattern:



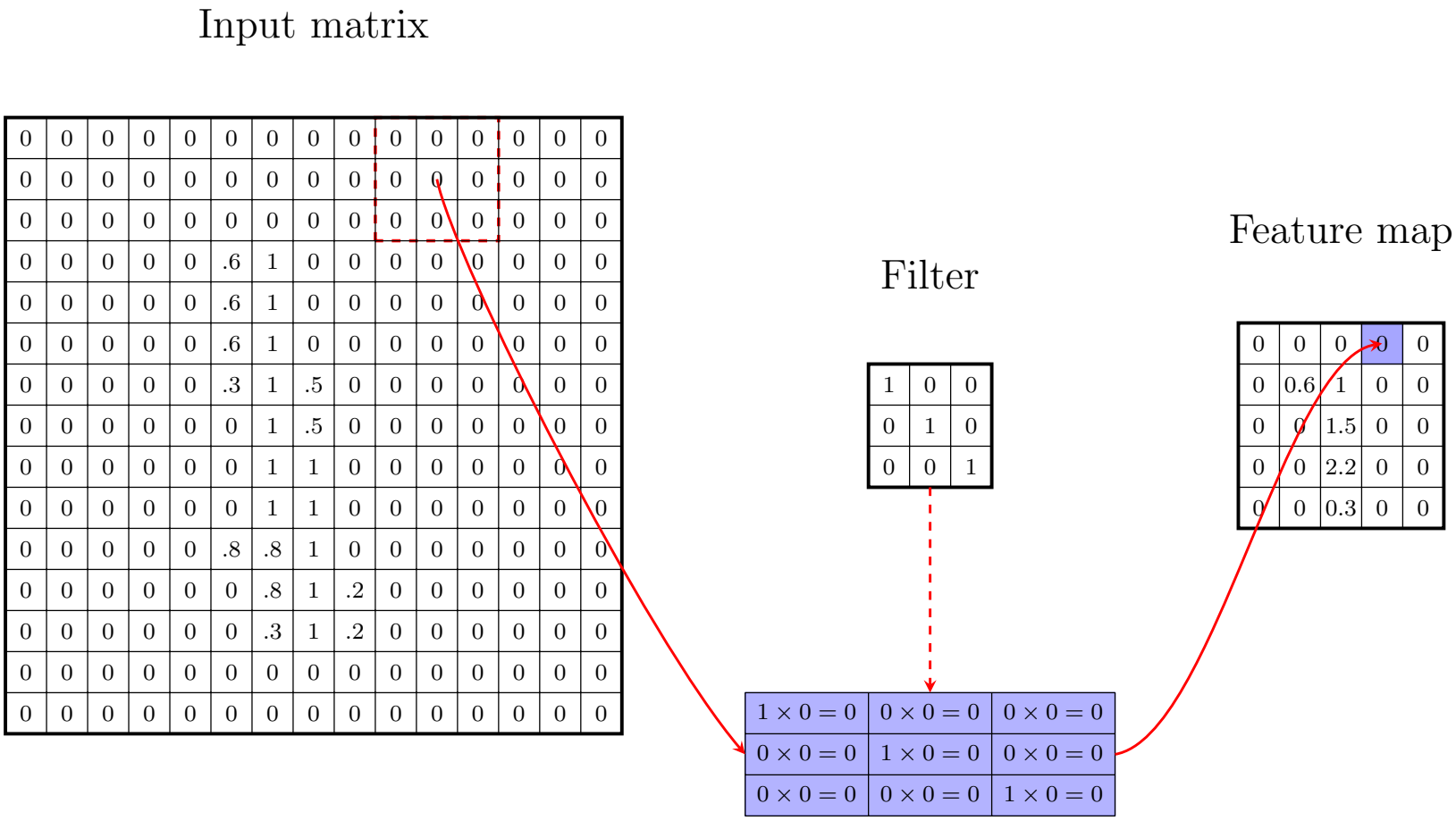
Illustration

We slide the filter through each 3x3 patch of the image — recording the presence of the filter-pattern:



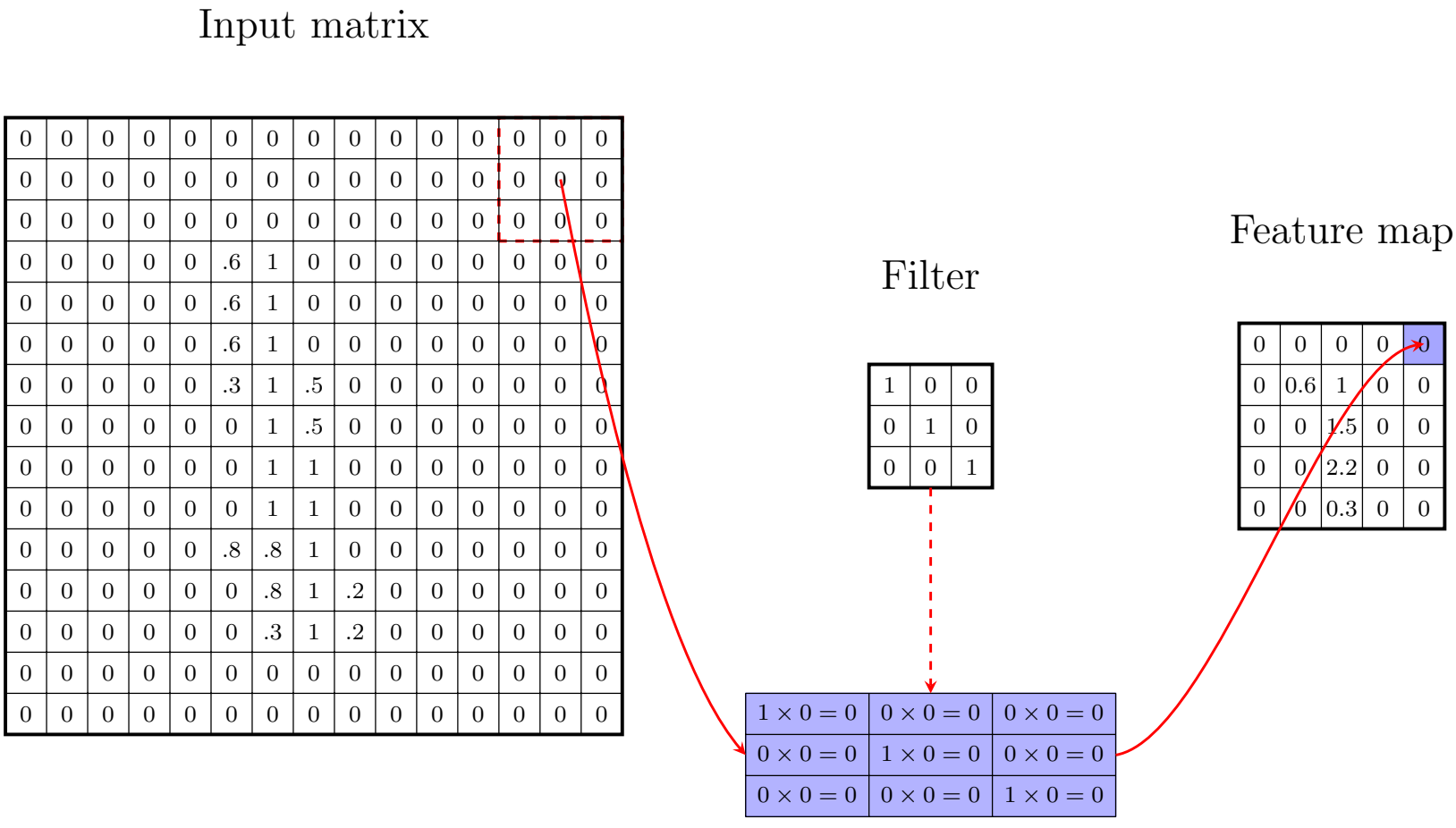
Illustration

We slide the filter through each 3x3 patch of the image — recording the presence of the filter-pattern:



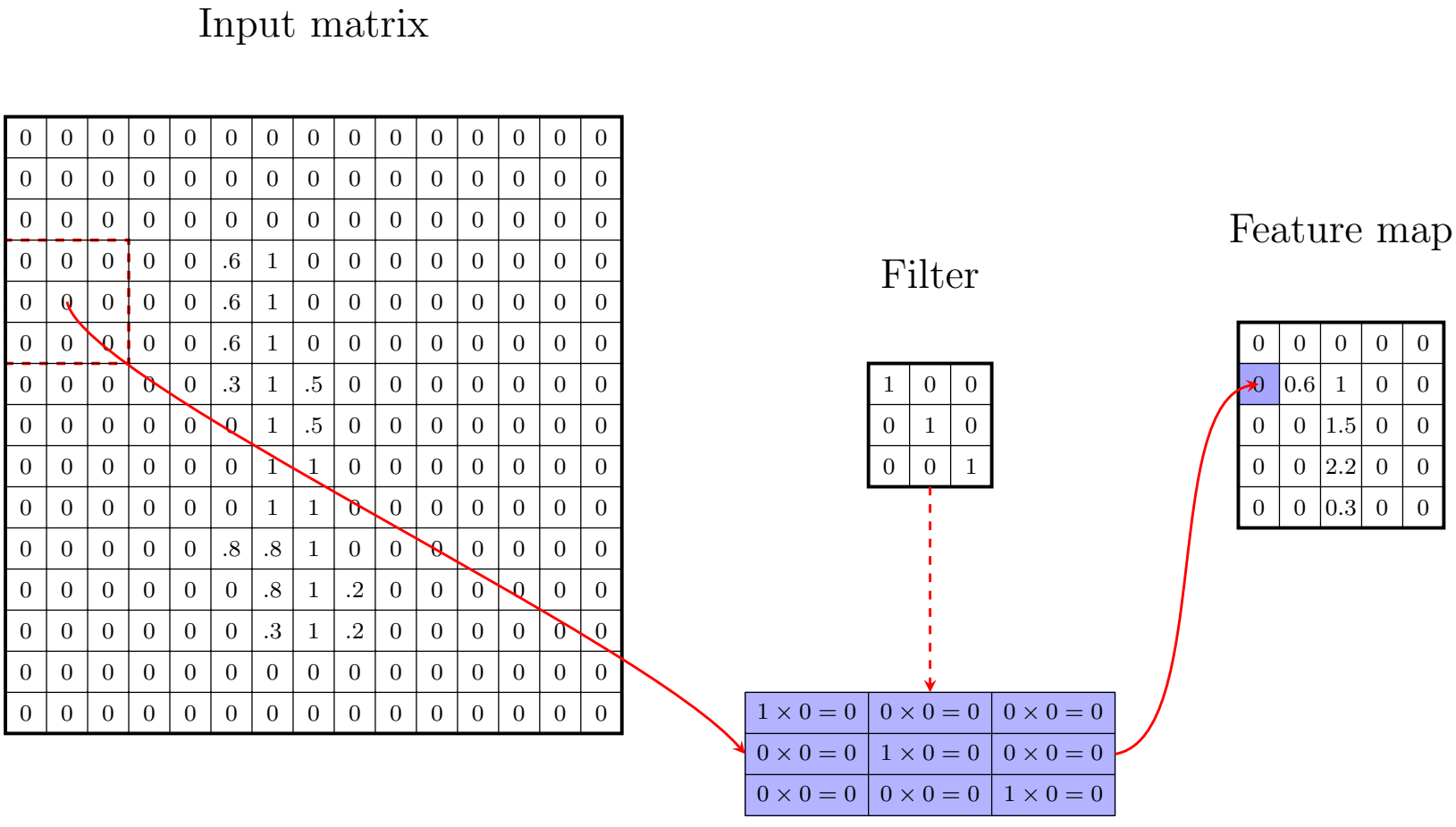
Illustration

We slide the filter through each 3x3 patch of the image — recording the presence of the filter-pattern:



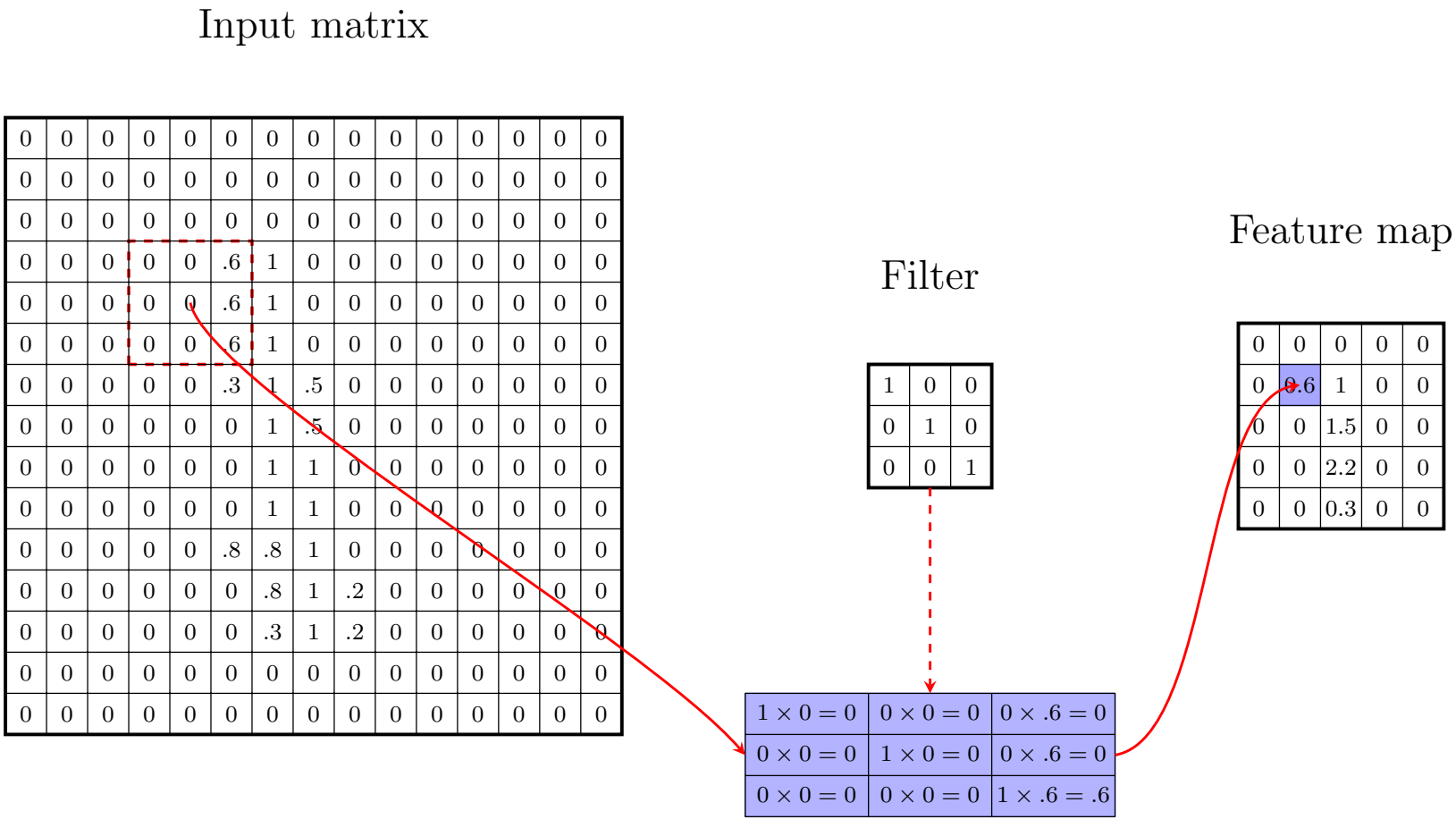
Illustration

We slide the filter through each 3x3 patch of the image — recording the presence of the filter-pattern:



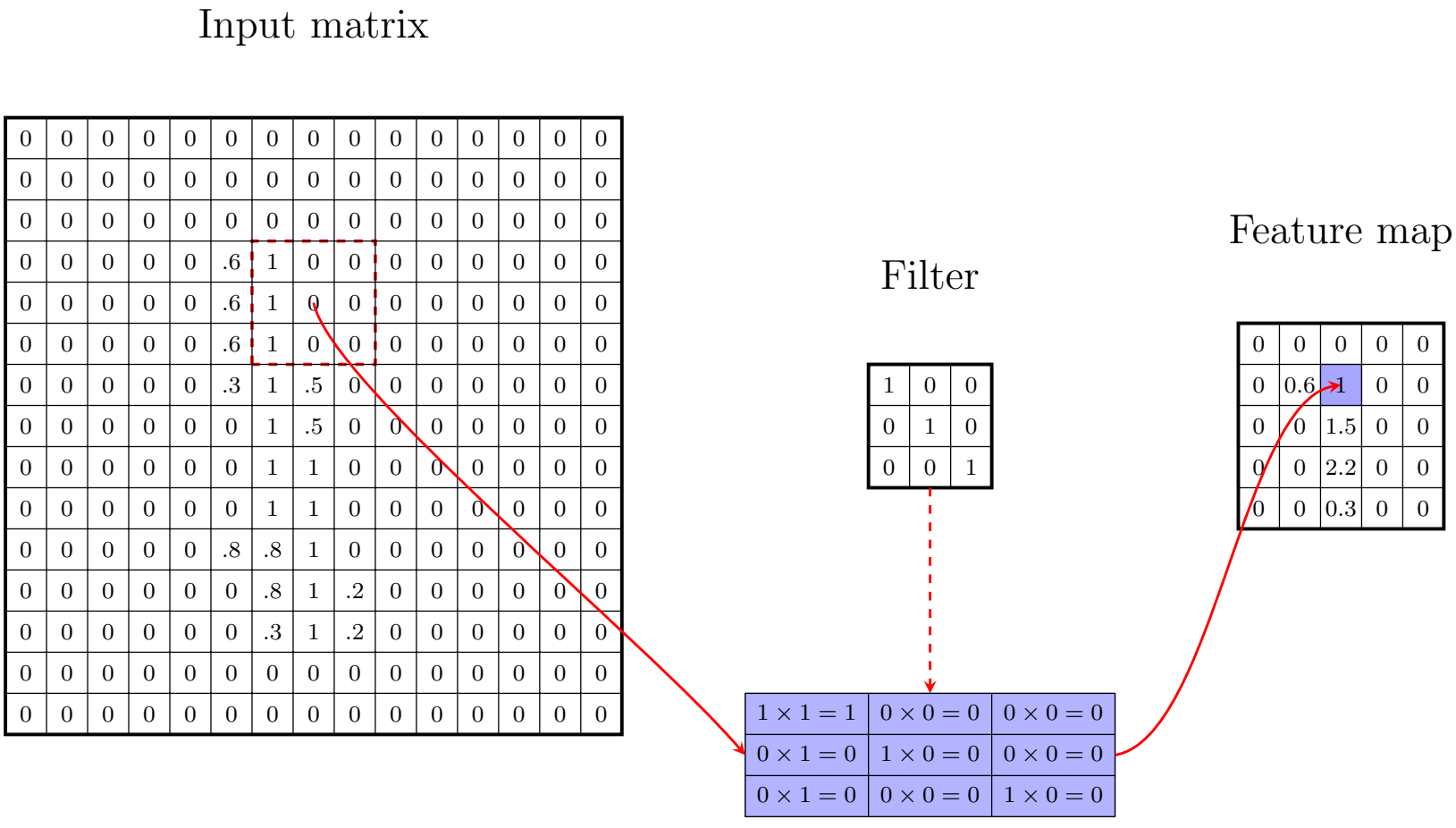
Illustration

We slide the filter through each 3x3 patch of the image — recording the presence of the filter-pattern:



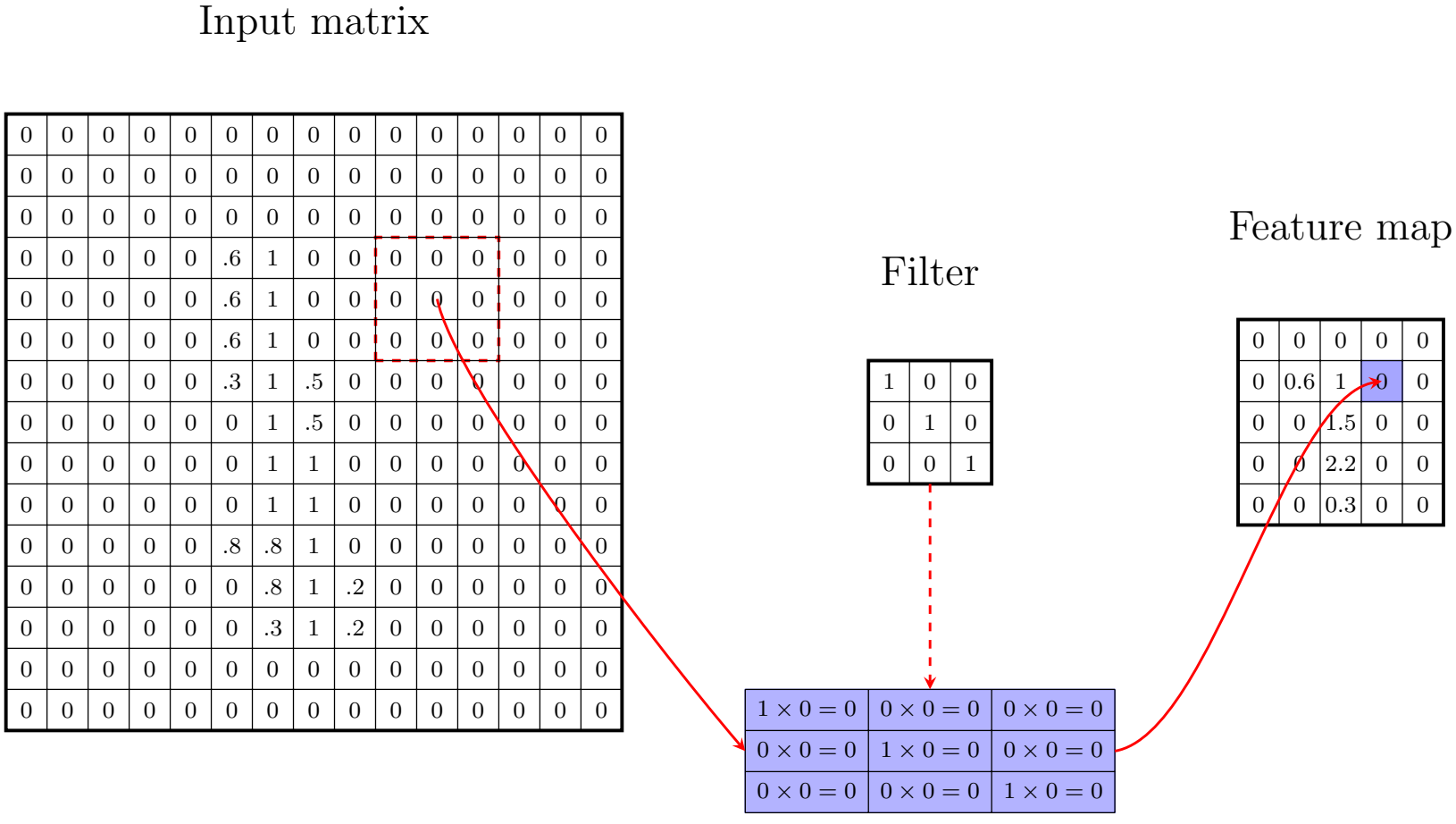
Illustration

We slide the filter through each 3x3 patch of the image — recording the presence of the filter-pattern:



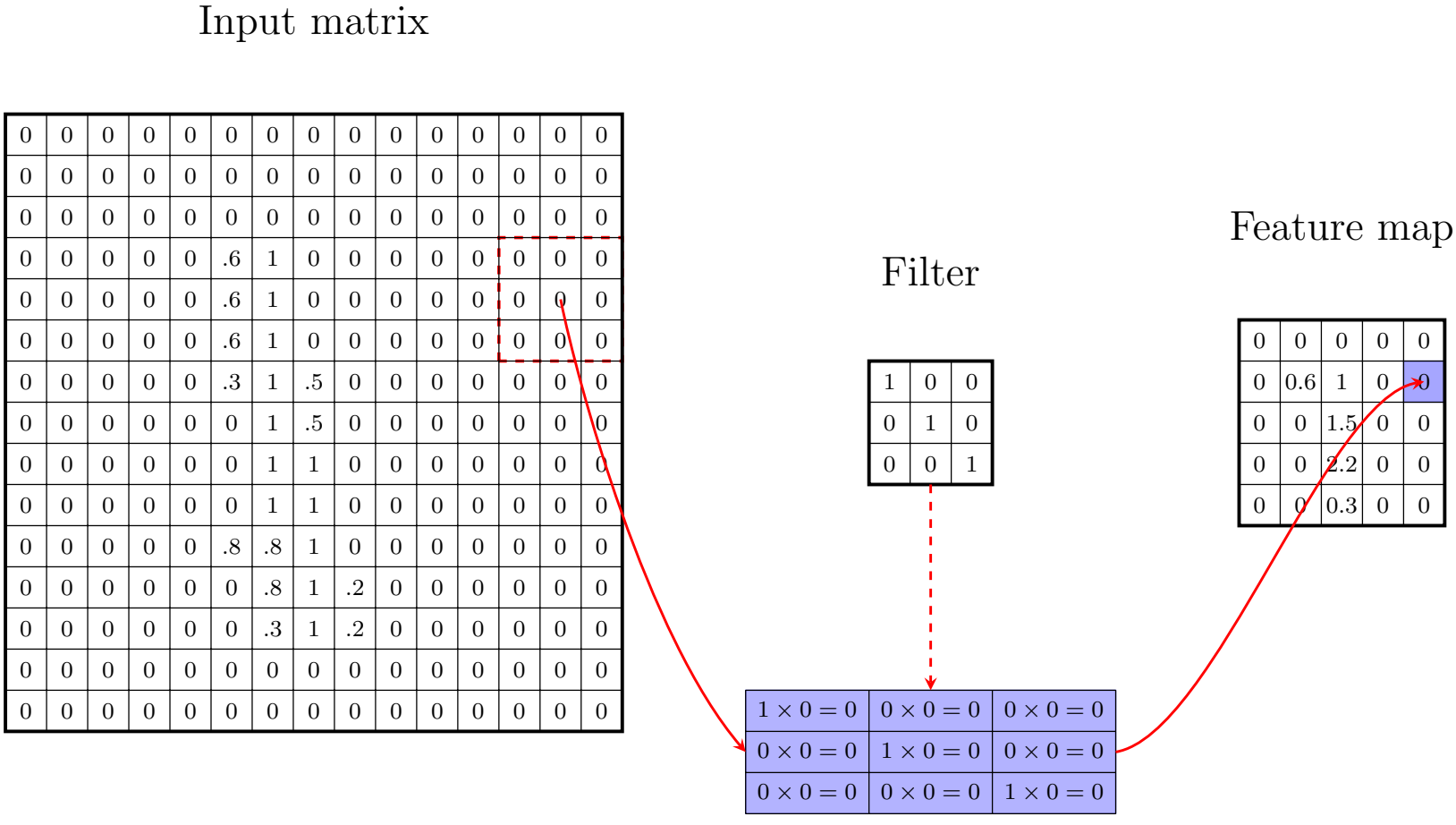
Illustration

We slide the filter through each 3x3 patch of the image — recording the presence of the filter-pattern:



Illustration

We slide the filter through each 3x3 patch of the image — recording the presence of the filter-pattern:



Processing of feature maps

1. **Non-linear transformation** — ensures non-redundancy of convolutional layers (extraction of more info).

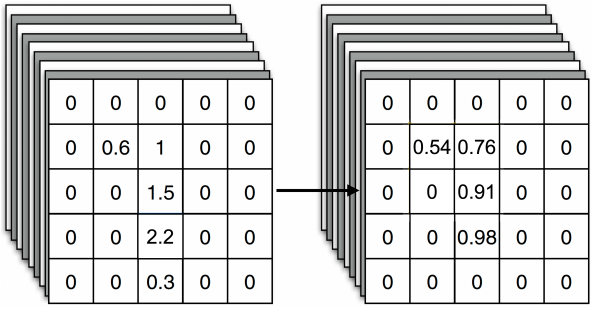


Figure 9: Tanh transformation (0-centered, -1/1 bounded sigmoid)

Processing of feature maps

1. **Non-linear transformation** — ensures non-redundancy of convolutional layers (extraction of more info).

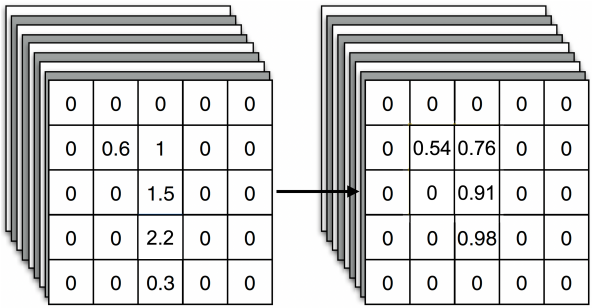
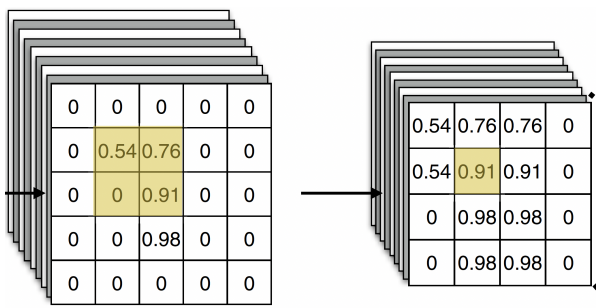


Figure 9: Tanh transformation (0-centered, -1/1 bounded sigmoid)

2. **Pooling** — reduces dimensionality of feature maps, and provides some *location invariance*.[4]. Here: 'max pooling'.<.->Keeps largest value within $k \times k$ patches (here: k=2)



[4] An emphasis on presence rather than location of pattern

Processing of feature maps

1. **Non-linear transformation** — ensures non-redundancy of convolutional layers (extraction of more info).

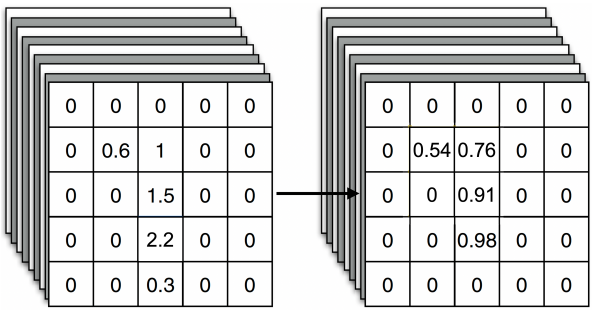
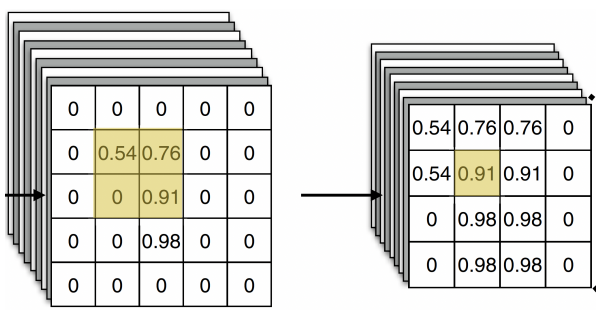


Figure 9: Tanh transformation (0-centered, -1/1 bounded sigmoid)

2. **Pooling** — reduces dimensionality of feature maps, and provides some *location invariance*.[4]. Here: 'max pooling'.[5]



[4] An emphasis on presence rather than location of pattern

[5] Keeps largest value within $k \times k$ patches (here: k=2)

Feed *feature maps* into regular hidden layers & predict

- **Thus far**: considered how convolutional layers extract feature maps
- **Last step**: regular hidden layers use them as input for classification

Feed *feature maps* into regular hidden layers & predict

- **Thus far**: considered how convolutional layers extract feature maps
- **Last step**: regular hidden layers use them as input for classification

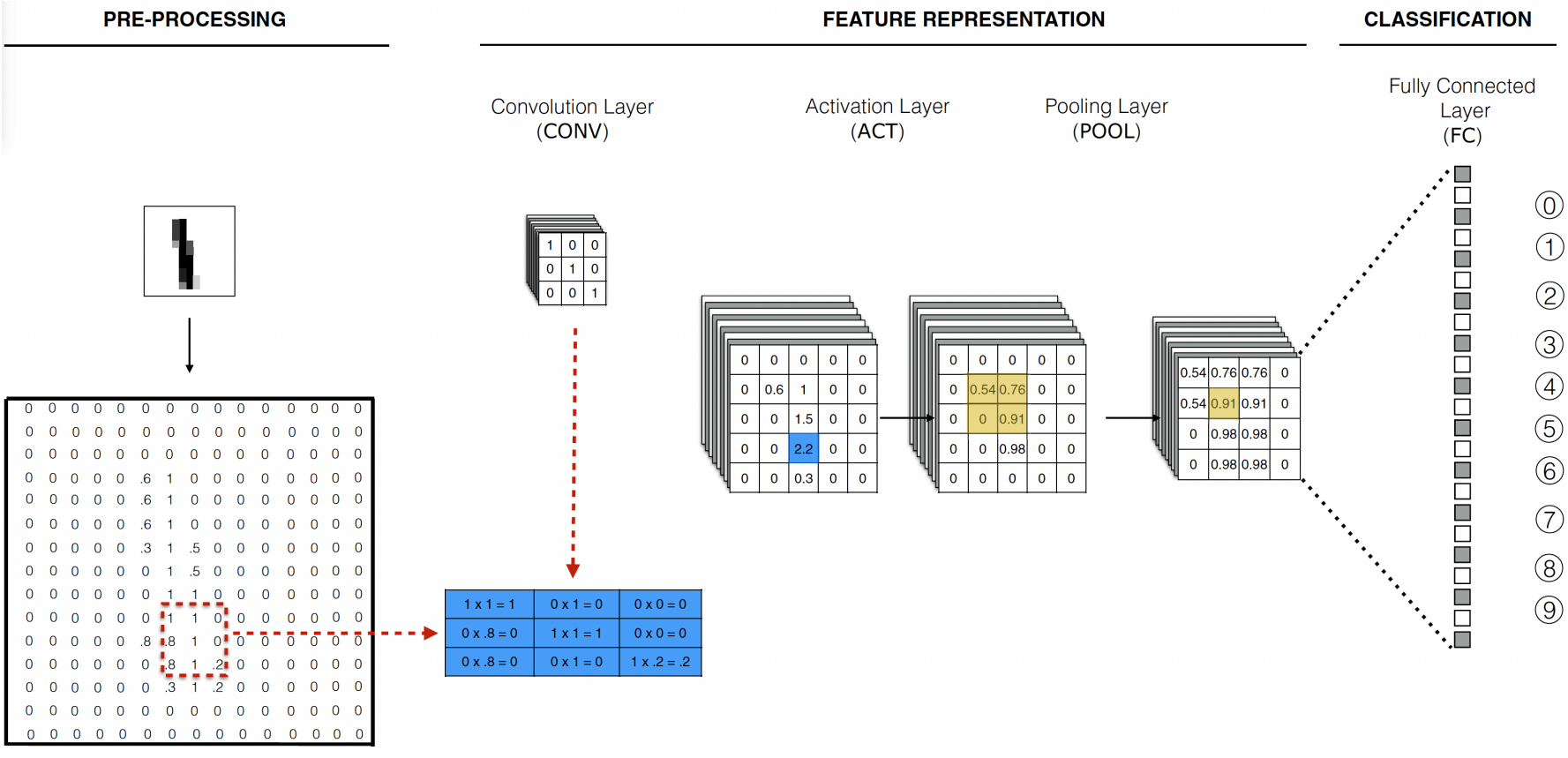


Figure 10: Overview of convolutional network structure (Torres & Cantu '22)

Feed *feature maps* into regular hidden layers & predict

- **Thus far**: considered how convolutional layers extract feature maps
- **Last step**: regular hidden layers use them as input for classification

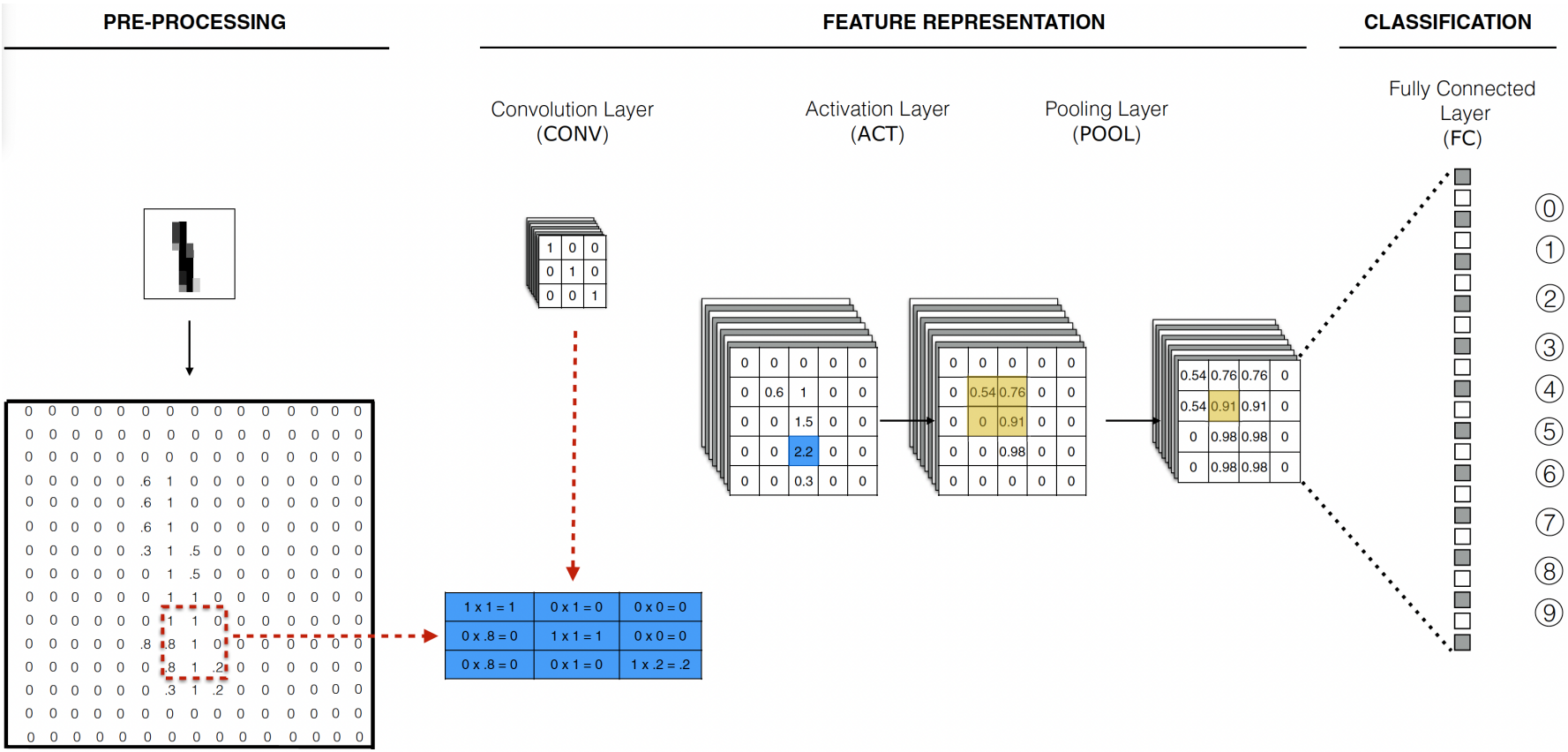


Figure 10: Overview of convolutional network structure (Torres & Cantu '22)

Goal: learn weights in the CONV, FC that minimizes loss function.

Some practical considerations

- Choices to be made about the CNN architecture
- Data augmentation
- Transfer learning
- Validation

Choices to be made about the CNN architecture

The basic structure of a CNN:

Figure 11: Torres & Cantu (2022)

- M *convolutional* layers
- K *regular* hidden layers

Choices to be made about the CNN architecture

The basic structure of a CNN:

Figure 11: Torres & Cantu (2022)

- M *convolutional* layers
- K *regular* hidden layers

How to set them?

Choices to be made about the CNN architecture

The basic structure of a CNN:

Figure 11: Torres & Cantu (2022)

- M *convolutional* layers
- K *regular* hidden layers

How to set them?

- No unique recipie — depends on data and problem.[6]
- Larger M: more *complex features* of image extracted.
- Larger K: more *complex classifier*.

[6] e.g., size of data, number classes to predict, similarity between classes.

Choices to be made about the CNN architecture

The basic structure of a CNN:

Figure 11: Torres & Cantu (2022)

- M *convolutional* layers
- K *regular* hidden layers

How to set them?

- No unique recipie — depends on data and problem.[6]
- Larger M: more *complex features* of image extracted.
- Larger K: more *complex classifier*.

[6] e.g., size of data, number classes to predict, similarity between classes.

Choices to be made about the CNN architecture

The basic structure of a CNN:

Figure 11: Torres & Cantu (2022)

- M *convolutional* layers
- K *regular* hidden layers

How to set them?

- No unique recipie — depends on data and problem.[6]
- Larger M: more *complex features* of image extracted.
- Larger K: more *complex classifier*.
- Starting point (Rosebrock 2017): $1 \leq M \leq 3$ and $K \geq 1$.
- Experimenting using grid search encouraged (Torres & Cantu 2022)

[6] e.g., size of data, number classes to predict, similarity between classes.

Choices to be made about the CNN architecture

Additionally

- *Number of filters*:
  - Large # $\rightarrow$ network looks for more features of similar complexity.
  - Common procedure: increase # filters gradually through network.[7]
- *Size of filters*:
  - If goal is to find *large-scale features* in image $\rightarrow$ larger filters.
  - If goal is to find *more fine-grained features* $\rightarrow$ smaller filters.

[7] x2 the no. filters every time we add an additional conv-layer (Shang 2016).

Choices to be made about the CNN architecture

Additionally

- *Number of filters*:
  - Large # $\rightarrow$ network looks for more features of similar complexity.
  - Common procedure: increase # filters gradually through network.[7]
- *Size of filters*:
  - If goal is to find *large-scale features* in image $\rightarrow$ larger filters.
  - If goal is to find *more fine-grained features* $\rightarrow$ smaller filters.

[7] x2 the no. filters every time we add an additional conv-layer (Shang 2016).

Choices to be made about the CNN architecture

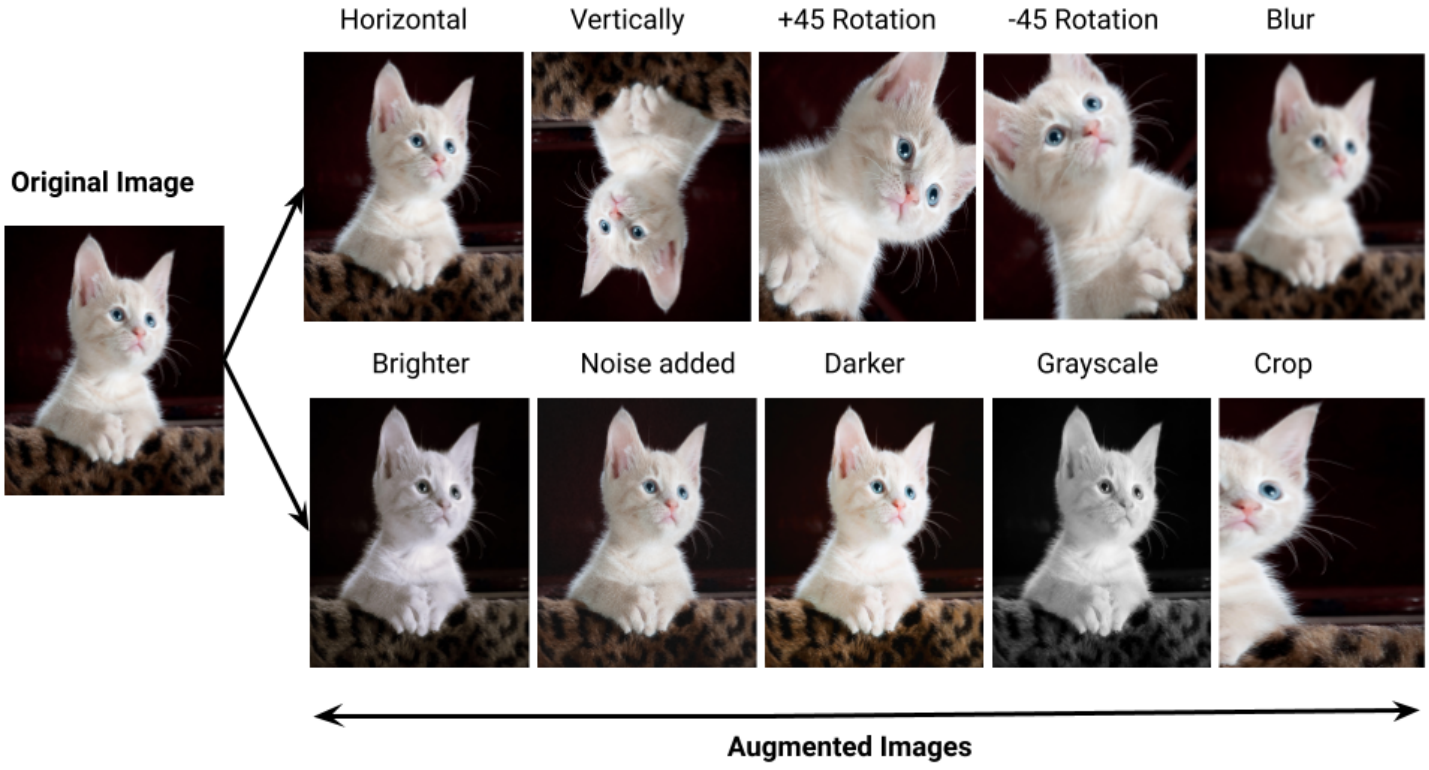
Additionally

- *Number of filters*:
  - Large # → network looks for more features of similar complexity.
  - Common procedure: increase # filters gradually through network.[7]
- *Size of filters*:
  - If goal is to find *large-scale features* in image → larger filters.
  - If goal is to find *more fine-grained features* → smaller filters.
- *Pooling*
  - Most common is max-pooling.
  - If very complex network: 3*3 in early stages and then 2*2 in later.
  - Else, 2*2 throughout.[8]

[7] x2 the no. filters every time we add an additional conv-layer (Shang 2016).
[8] With a stride of 2: ie, slide window 2 pixels each time → non-overlapping patches.

Data augmentation

- Produces random variations of the original training images by, e.g., *flipping*, *rotating*, *zooming out*.
- Forces model to pay less attention to location and orientation of a feature — instead: grasp relationship to other features.
- Generally leads to better generalizability.



Transfer learning

- Training a CNN *from scratch* can be computationally expensive and *require large data*.
- Alternative: *transfer learning* ; use (large) model trained on a *similar but different* data.
- How it works:
  - Freeze *initial k* layers of pre-trained model.
  - Re-train *last j* layers.

Validation

- When used for measurement purposes — '*validate*, *validate*, *validate*' (Grimmer & Stewart 2013; Torres & Cantu 2022)
- Recommended ways to improve model:
  - Review missclassified images in order to spot problems.
  - Visualize most important features (apsects of images) for predictions.

Application: coding electoral vote from vote tallies.

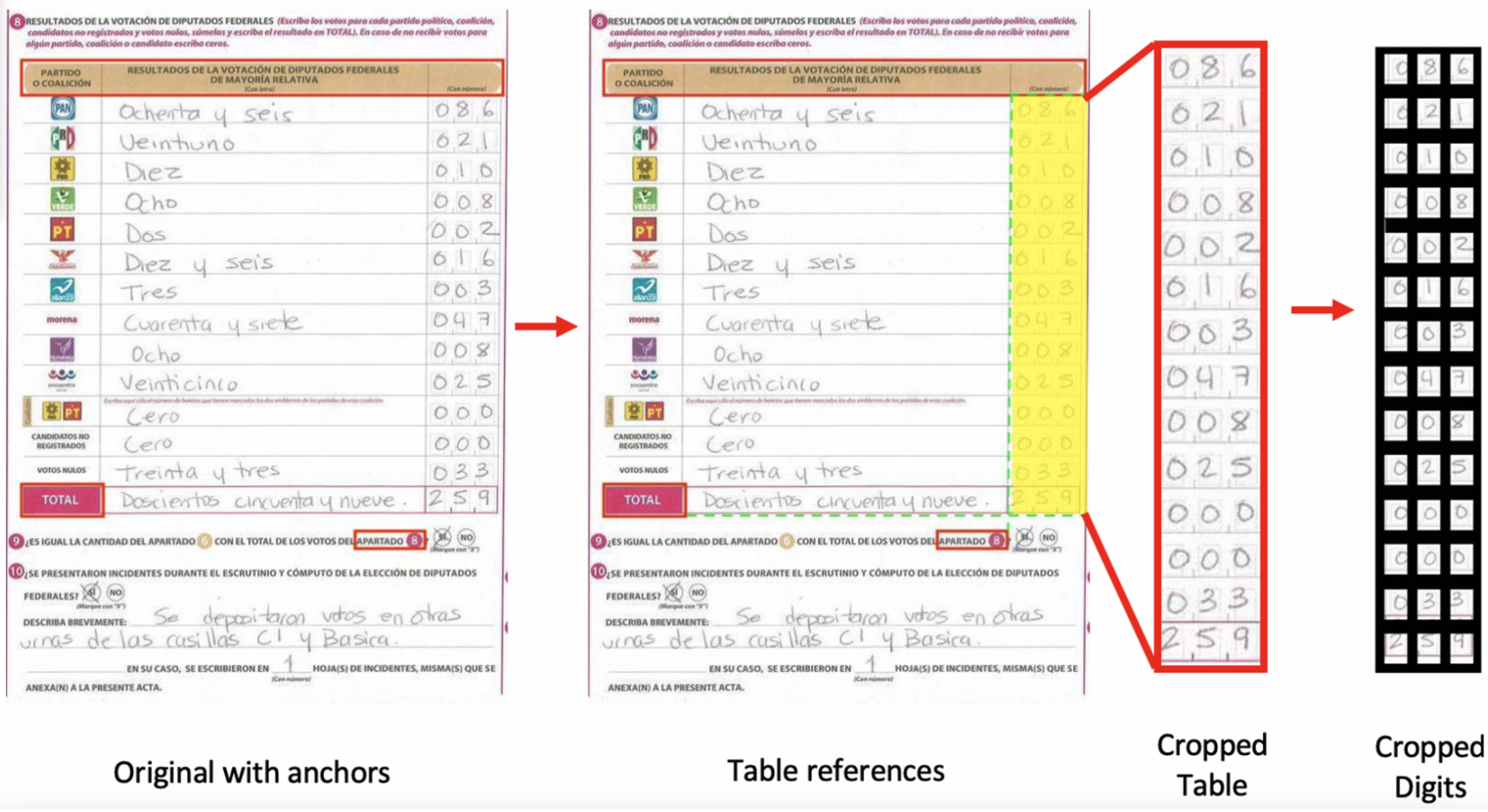


Figure 12: Vote-tally extraction, Mexico 2015 (Cantu and Torres 22)

Goal

- Correctly identify the handwritten digits.
- Sum predicted digits to arrive at electoral outcomes.

Application: coding electoral vote from vote tallies.

Basic setup

1. Train 'base' model on large *external data* (MNIST; 60K handwritten digits).
   - CNN with 2 CONV-layers, max-pooling, and 3 FC-layers.[9]
   - Select (via grid search) parameters to achieve *low validation error on Mexican sample*.

[9] Regular hidden layers

Application: coding electoral vote from vote tallies.

Basic setup

1. Train 'base' model on large *external data* (MNIST; 60K handwritten digits).
   - CNN with 2 CONV-layers, max-pooling, and 3 FC-layers.[9]
   - Select (via grid search) parameters to achieve *low validation error on Mexican sample*.

[9] Regular hidden layers

Application: coding electoral vote from vote tallies.

Basic setup

1. Train 'base' model on large *external data* (MNIST; 60K handwritten digits).
   - CNN with 2 CONV-layers, max-pooling, and 3 FC-layers.[9]
   - Select (via grid search) parameters to achieve *low validation error on Mexican sample*.
2. *Transfer learning*
   - Freeze 1st CONV-layer.
   - Retrain remaining layers based on Mexican digits.

[9] Regular hidden layers

Application: coding electoral vote from vote tallies.

Basic setup

1. Train 'base' model on large *external data* (MNIST; 60K handwritten digits).
   - CNN with 2 CONV-layers, max-pooling, and 3 FC-layers.[9]
   - Select (via grid search) parameters to achieve *low validation error on Mexican sample*.
2. *Transfer learning*
   - Freeze 1st CONV-layer.
   - Retrain remaining layers based on Mexican digits.
3. Validate

[9] Regular hidden layers

Application: coding electoral vote from vote tallies.

Table A.2: Hyperparameter grid search set-up

Hyperparameter	Default	Min. value	Max. value	Interval	Selected
Number `CONV` blocks	2	1	3	1	3
Filters Layer 1	-	16	128	16	64
Filters Layer 2	-	16	128	16	96
Filters Layer 3	-	16	128	16	112
Dropout	0.5	0.1	0.5	0.1	0.1
Size `FC 1`	128	50	200	20	150
Size `FC 2`	50	10	100	10	70
Learning rate	0.0001	0.0001	0.01	-	$\approx$0.0001
Epochs	-	-	45	-	20

Figure 13: Grid search (Cantu and Torres 22)

Application: coding electoral vote from vote tallies.

Table A.3: Results from the hyperparameter search grid (selected)

Number CONV layers	Filters Layer 1	Filters Layer 2	Filters Layer 3	Dropout rate	Size FC 1	Size FC 2	Number Epochs	Validity Accuracy
3	64	32	32	0.200	110	20	45	0.332
3	96	32	112	0.300	70	50	5	0.335
3	112	80	80	0.200	70	50	45	0.338
3	80	64	96	0.200	50	20	5	0.340
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
3	96	112	48	0.300	130	50	45	0.352
3	80	112	112	0.500	170	30	15	0.350
3	32	32	112	0.200	110	30	5	0.346
3	96	32	112	0.300	70	50	15	0.343
3	32	32	112	0.200	110	30	15	0.342

Figure 14: Results: grid search (Cantu and Torres 22)

Application: coding electoral vote from vote tallies.

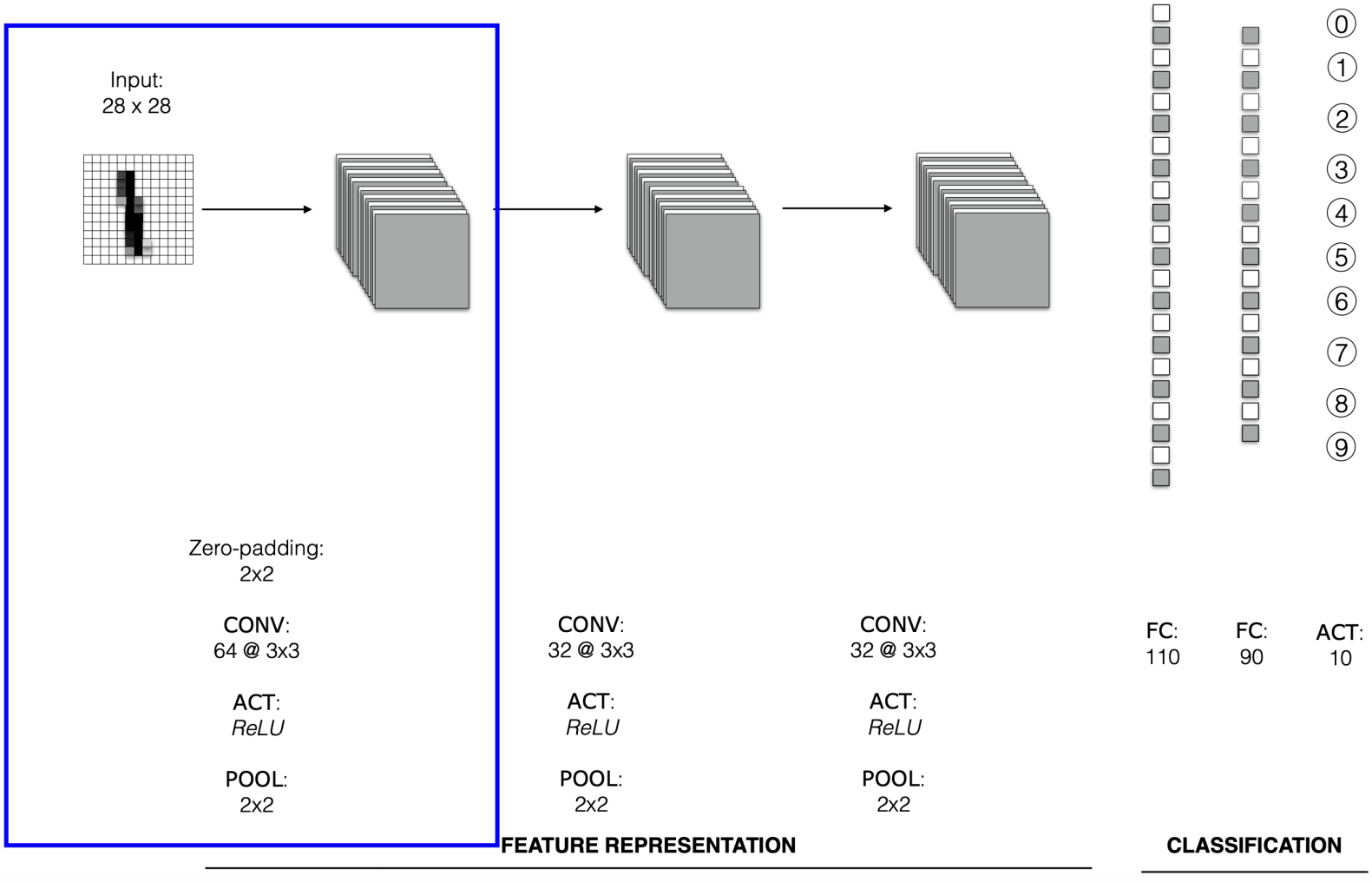
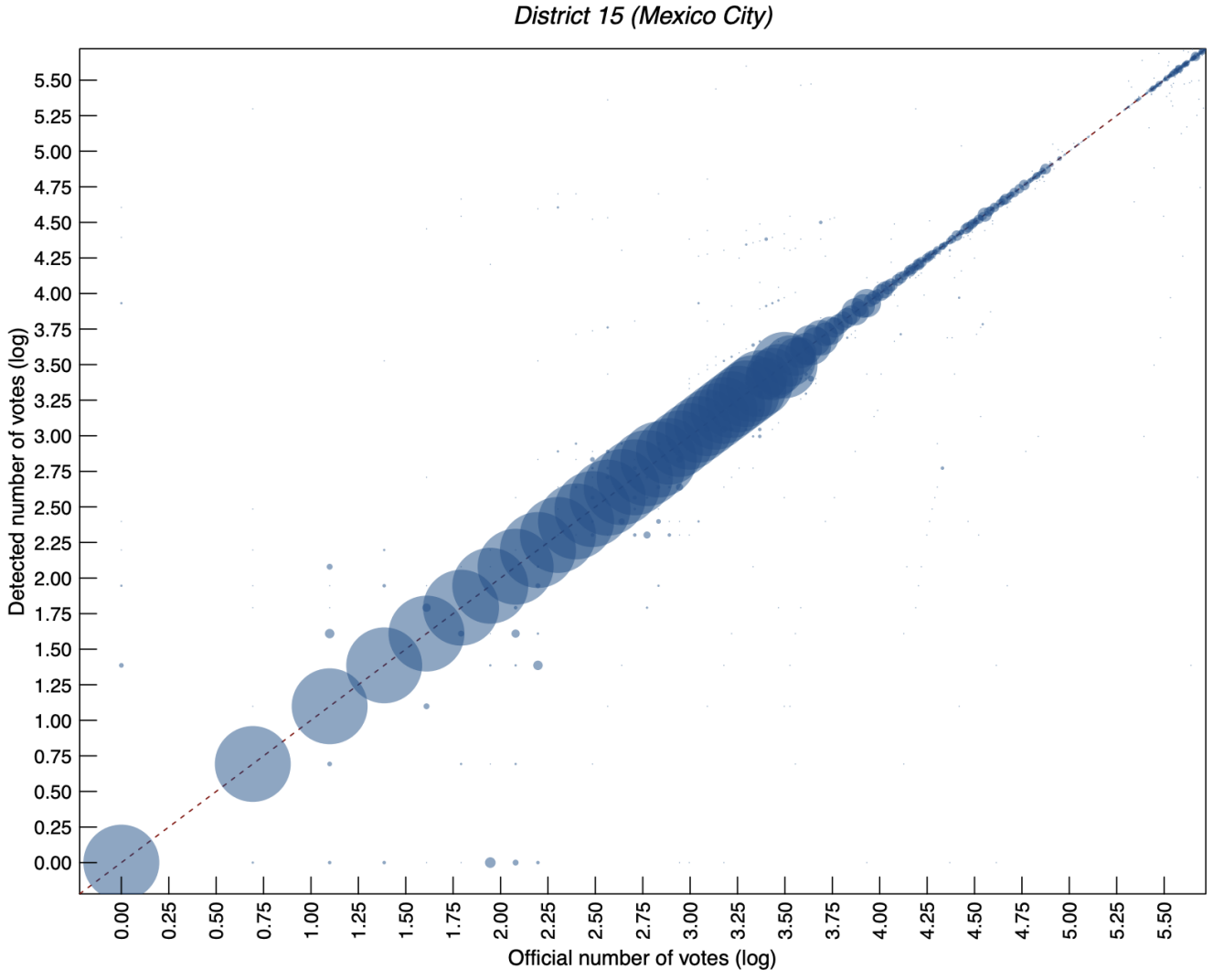


Figure 15: CNN-structure, Mexico 2015 (Cantu and Torres 22)

Application: coding electoral vote from vote tallies.



Application: coding electoral vote from vote tallies.

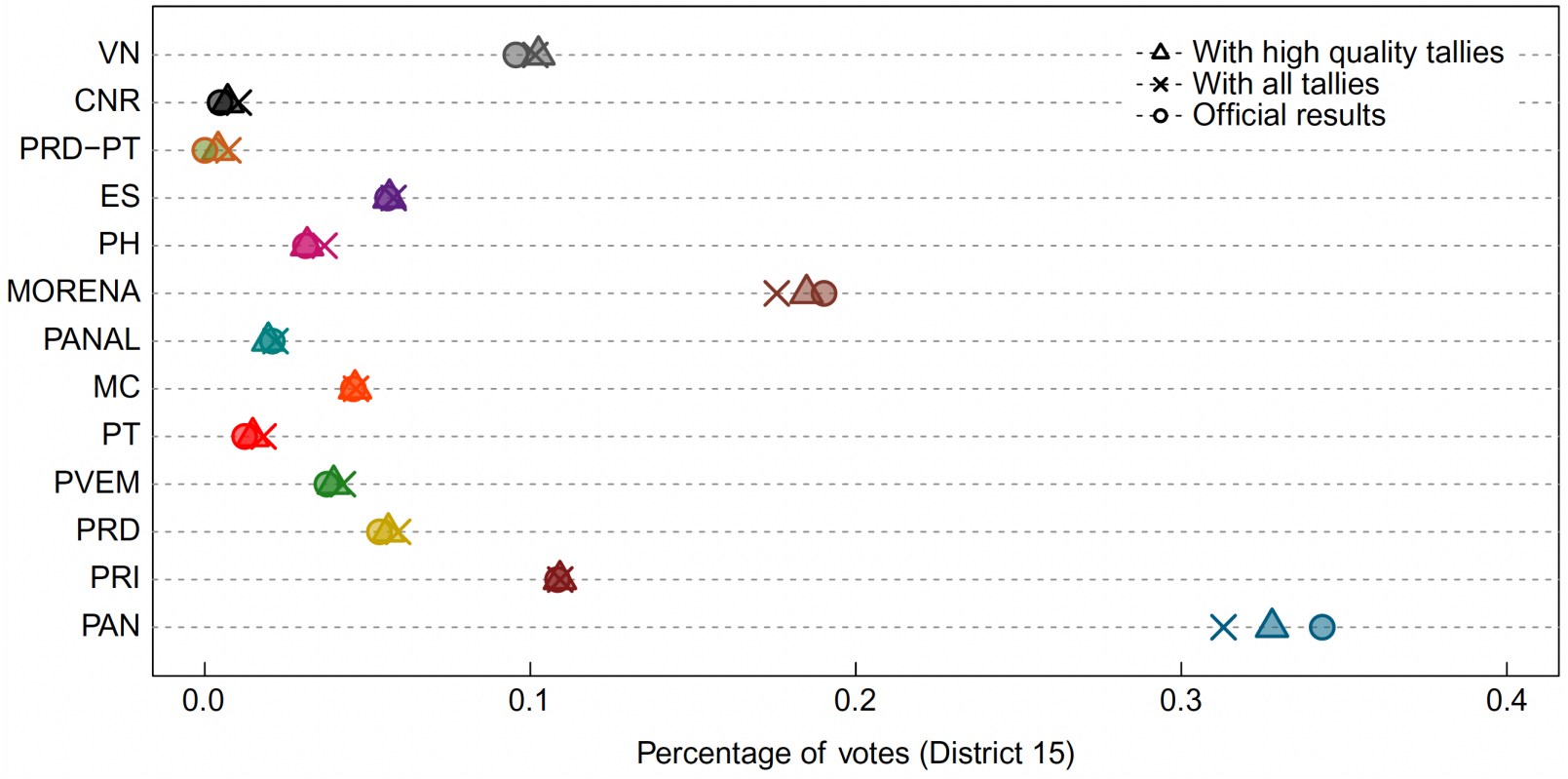


Figure 17: Final results (Cantu and Torres 22)

Misc

Summary

- CNNs are a powerful type of neural network, with an architecture attuned to image data.
- Learns simple and complex features of images (in *convolutional layers*) and then learns how to weight/combine them (in *regular hidden layers*) to effectively predict image classes.
- Like for regular networks:
  - Require lots of data to clearn complex patterns.
  - Even more hyperparameters to choose.
  - Lacks in inference and interpretability.

Further reading

- ISL, Ch. 10 (10.1-10.7; minus 10.5 and 10.7.1)
- Torres, M., & Cantú, F. (2022). Learning to see: Convolutional neural networks for the analysis of social science data. Political Analysis, 30(1), 113-131.
- Zhang, H., & Pan, J. (2019). Casm: A deep-learning approach for identifying collective action events with text and image data from social media. Sociological Methodology, 49(1), 1-57.

Unsupervised learning

Types of learning

Main focus of this course: branch of machine learning called **supervised learning**.

- For such problems, we have a *priviliged variabled* Y that we want to *predict* based on some other set of variables X—by estimating a function $\hat{f}(X)$.

Types of learning

Main focus of this course: branch of machine learning called **supervised learning**.

- For such problems, we have a *priviliged variabled* Y that we want to *predict* based on some other set of variables X—by estimating a function $\hat{f}(X)$.

Now, a brief detour: **unsupervised learning**

Types of learning

Main focus of this course: branch of machine learning called **supervised learning**.

- For such problems, we have a *priviliged variabled* Y that we want to *predict* based on some other set of variables X—by estimating a function $\hat{f}(X)$.

Now, a brief detour: **unsupervised learning**

- Do *not* have a privileged variarble Y, just X variables.

Types of learning

Main focus of this course: branch of machine learning called **supervised learning**.

- For such problems, we have a *priviliged variabled* Y that we want to *predict* based on some other set of variables X—by estimating a function $\hat{f}(X)$.

Now, a brief detour: **unsupervised learning**

- Do *not* have a privileged variarble Y, just X variables.
- What sort of analysis is possible without Y?

Types of learning

Main focus of this course: branch of machine learning called **supervised learning**.

- For such problems, we have a *priviliged variabled* Y that we want to *predict* based on some other set of variables X—by estimating a function $\hat{f}(X)$.

Now, a brief detour: **unsupervised learning**

- Do *not* have a privileged variarble Y, just X variables.
- What sort of analysis is possible without Y?
- What will be our loss function if we cannot minimize the distance between $\hat{f}$ and some true Y??

Types of learning

Main focus of this course: branch of machine learning called **supervised learning**.

- For such problems, we have a *priviliged variabled* Y that we want to *predict* based on some other set of variables X—by estimating a function $\hat{f}(X)$.

Now, a brief detour: **unsupervised learning**

- Do *not* have a privileged variarble Y, just X variables.
- What sort of analysis is possible without Y?
- What will be our loss function if we cannot minimize the distance between $\hat{f}$ and some true Y??

There exists two main—broad—types unsupervised learning:

Types of learning

Main focus of this course: branch of machine learning called **supervised learning**.

- For such problems, we have a *priviliged variabled* Y that we want to *predict* based on some other set of variables X—by estimating a function $\hat{f}(X)$.

Now, a brief detour: **unsupervised learning**

- Do *not* have a privileged variarble Y, just X variables.
- What sort of analysis is possible without Y?
- What will be our loss function if we cannot minimize the distance between $\hat{f}$ and some true Y??

There exists two main—broad—types unsupervised learning:

- *Dimensionality reduction*: analyzes relationship between variables

Types of learning

Main focus of this course: branch of machine learning called **supervised learning**.

- For such problems, we have a *priviliged variabled* Y that we want to *predict* based on some other set of variables X—by estimating a function $\hat{f}(X)$.

Now, a brief detour: **unsupervised learning**

- Do *not* have a privileged variarble Y, just X variables.
- What sort of analysis is possible without Y?
- What will be our loss function if we cannot minimize the distance between $\hat{f}$ and some true Y??

There exists two main—broad—types unsupervised learning:

- *Dimensionality reduction*: analyzes relationship between variables
- *Clustering*: analyzes relationship between observations.

Illustration—toy data






User A	0	0	0	1	1	1
User B	1	1	0	0	0	0
User C	0	0	0	1	1	0
User D	1	1	1	0	0	0
User E	0	0	0	0	1	1
User F	0	1	1	0	0	0
User G	1	1	0	0	0	0
User G	0	0	0	1	1	0

Can we spot (*a*) similar rows and/or (*b*) similar columns here?

Dimensionality reduction: *column patterns*



User A	0	0	0	1	1	1
User B	1	1	0	0	0	0
User C	0	0	0	1	1	0
User D	1	1	1	0	0	0
User E	0	0	0	0	1	1
User F	0	1	1	0	0	0
User G	1	1	0	0	0	0
User G	0	0	0	1	1	0

We can relatively easy spot two **subgroups of columns** (artists). They are deemed similar because they have high correlation / co-occurrence.

Clustering: *row patterns*






User A	0	0	0	1	1	1
User B	1	1	0	0	0	0
User C	0	0	0	1	1	0
User D	1	1	1	0	0	0
User E	0	0	0	0	1	1
User F	0	1	1	0	0	0
User G	1	1	0	0	0	0
User G	0	0	0	1	1	0

We can also spot **subgroups of rows** (users). They are deemed similar because they have adopted similar artists (i.e., have similar values on each column).

Clustering: *row patterns*






User A	0	0	0	1	1	1
User B	1	1	0	0	0	0
User C	0	0	0	1	1	0
User D	1	1	1	0	0	0
User E	0	0	0	0	1	1
User F	0	1	1	0	0	0
User G	1	1	0	0	0	0
User G	0	0	0	1	1	0

We can also spot **subgroups of rows** (users). They are deemed similar because they have adopted similar artists (i.e., have similar values on each column).

Seems simple enough—why do we need ML?

When...

- We have just a couple of observations (rows) and just a couple of variables (columns),
- And, the data is made up of just a small number of simple & non-noisy patterns.

Seems simple enough—why do we need ML?

When. . .

- We have just a couple of observations (rows) and just a couple of variables (columns),
- And, the data is made up of just a small number of simple & non-noisy patterns.

. . . Then we likely can manage ourselves, without ML.

Seems simple enough—why do we need ML?

When. . .

- We have just a couple of observations (rows) and just a couple of variables (columns),
- And, the data is made up of just a small number of simple & non-noisy patterns.

. . . Then we likely can manage ourselves, without ML.

Problem is—in reality, this is not how our data generally looks like.

Seems simple enough—why do we need ML?

When...

- We have just a couple of observations (rows) and just a couple of variables (columns),
- And, the data is made up of just a small number of simple & non-noisy patterns.

... Then we likely can manage ourselves, without ML.

Problem is—in reality, this is not how our data generally looks like.

- Many rows
- Many columns
- No simple separation between subgroups
- Complex patterns

A challenge of unsupervised learning

OK—I might now have convinced you that we need computers for *unsupervised learning*. However, a challenge of unsupervised learning is the following:

A challenge of unsupervised learning

OK—I might now have convinced you that we need computers for *unsupervised learning*. However, a challenge of unsupervised learning is the following:

While the *goal(s)* of **supervised learning**—prediction, inference—are well-defined and can be *assessed* using highly standardized procedures (prediction accuracy, p-values)...

A challenge of unsupervised learning

OK—I might now have convinced you that we need computers for *unsupervised learning*. However, a challenge of unsupervised learning is the following:

While the *goal(s)* of **supervised learning**—prediction, inference—are well-defined and can be *assessed* using highly standardized procedures (prediction accuracy, p-values)...

...The goal of **unsupervised learning** is a bit more *fuzzy* and *subjective*. Broadly speaking: find patterns in data that in some way are "useful" or "interesting" in relation to some user-defined purpose.

A challenge of unsupervised learning

OK—I might now have convinced you that we need computers for *unsupervised learning*. However, a challenge of unsupervised learning is the following:

While the *goal(s)* of **supervised learning**—prediction, inference—are well-defined and can be *assessed* using highly standardized procedures (prediction accuracy, p-values)...

...The goal of **unsupervised learning** is a bit more *fuzzy* and *subjective*. Broadly speaking: find patterns in data that in some way are "useful" or "interesting" in relation to some user-defined purpose.

- This fuzziness can—in large part—be attributed to a **lack of a ground-truth** (which Y provides for supervised learning).

A challenge of unsupervised learning

OK—I might now have convinced you that we need computers for *unsupervised learning*. However, a challenge of unsupervised learning is the following:

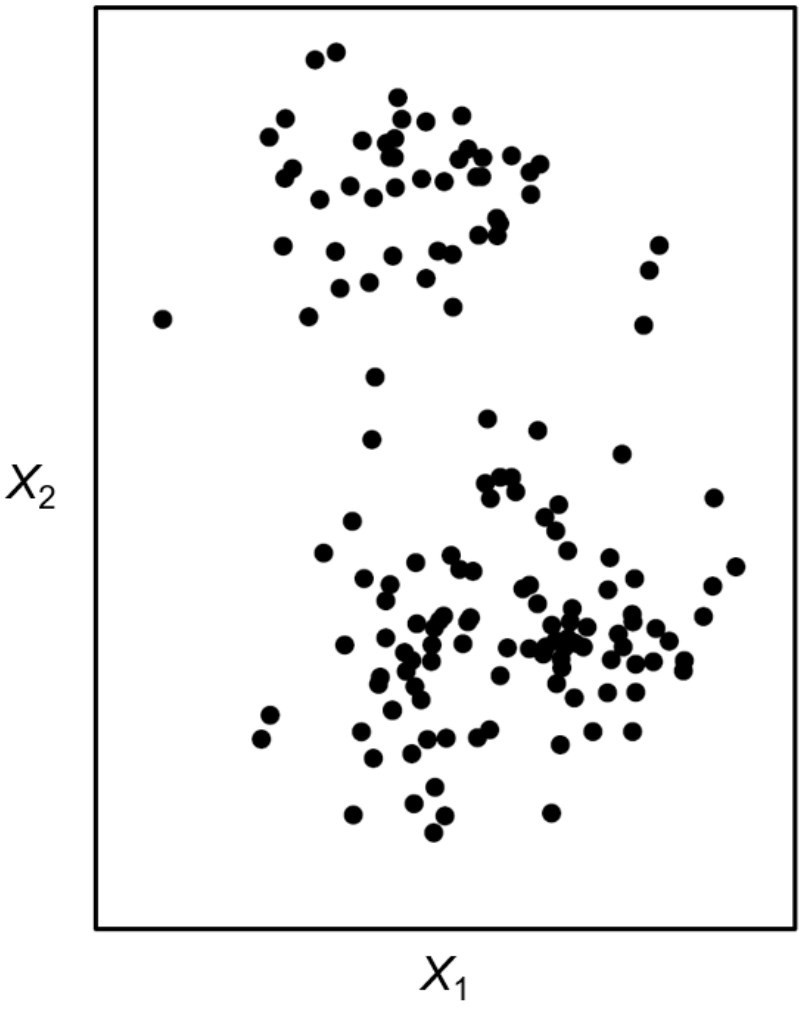
While the *goal(s)* of **supervised learning**—prediction, inference—are well-defined and can be *assessed* using highly standardized procedures (prediction accuracy, p-values)...

...The goal of **unsupervised learning** is a bit more *fuzzy* and *subjective*. Broadly speaking: find patterns in data that in some way are "useful" or "interesting" in relation to some user-defined purpose.

- This fuzziness can—in large part—be attributed to a **lack of a ground-truth** (which Y provides for supervised learning).
- We cannot simply check if our prediction was correct or not.

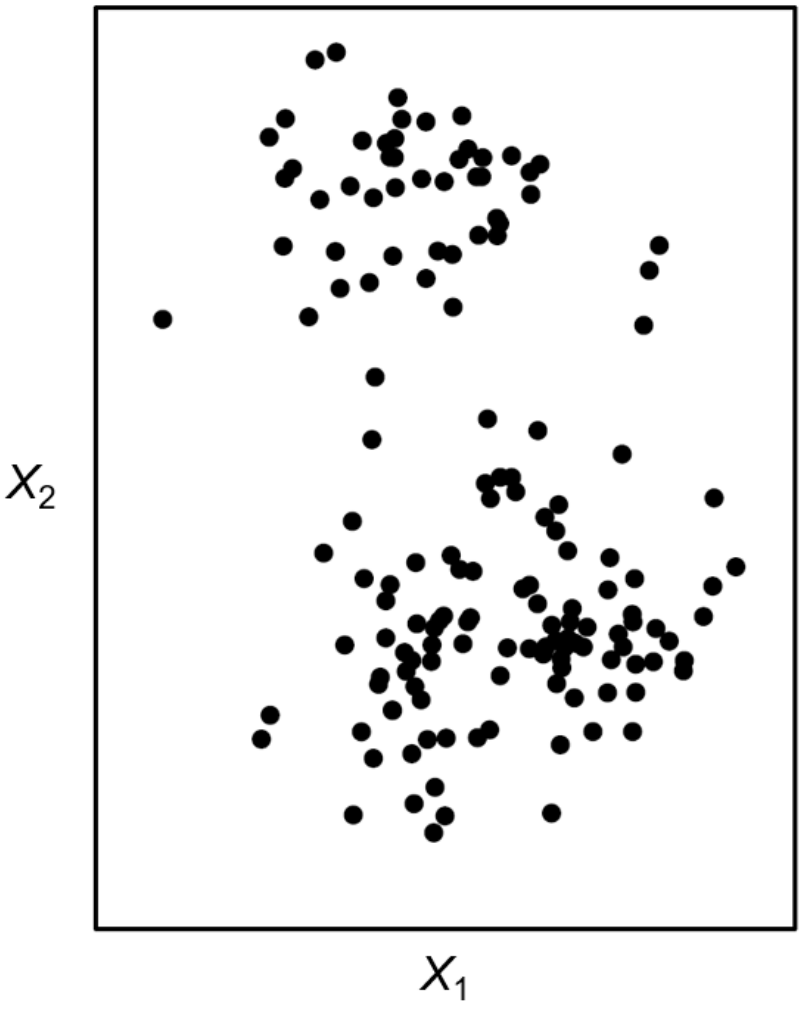
Clustering

Clustering



Objective of clustering: partition observations into distinct groups according to some *distance function* (e.g., euclidean distance) such that:

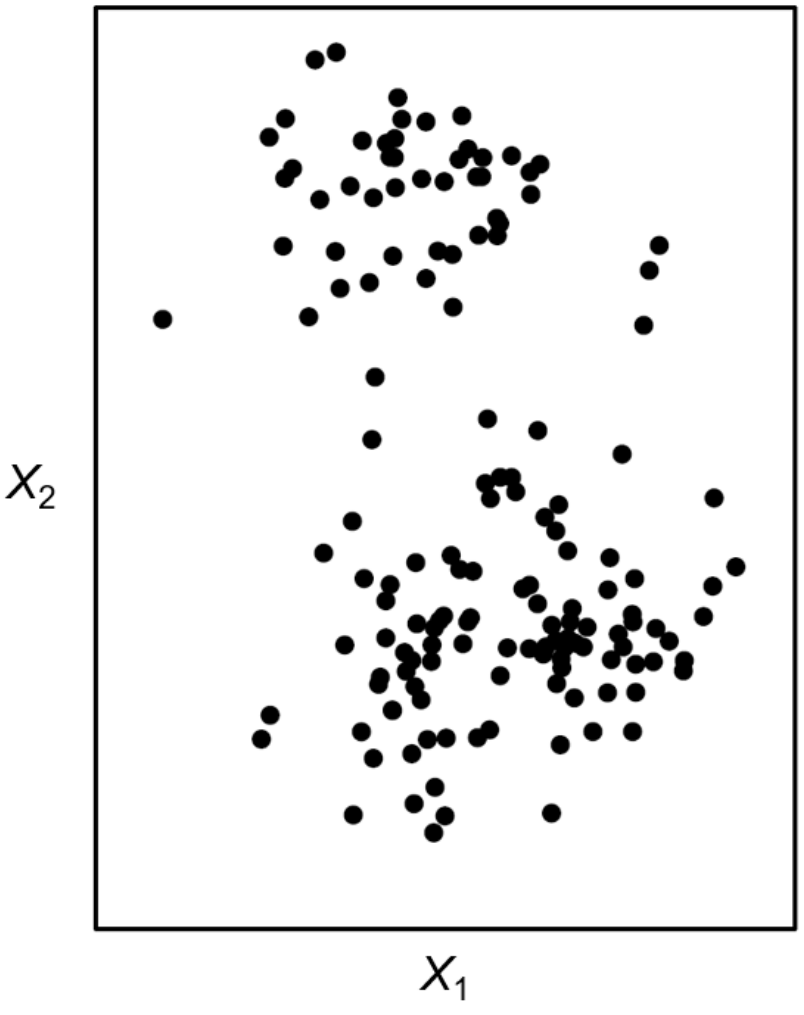
Clustering



Objective of clustering: partition observations into distinct groups according to some *distance function* (e.g., euclidean distance) such that:

- within-group similarity is *maximized*,

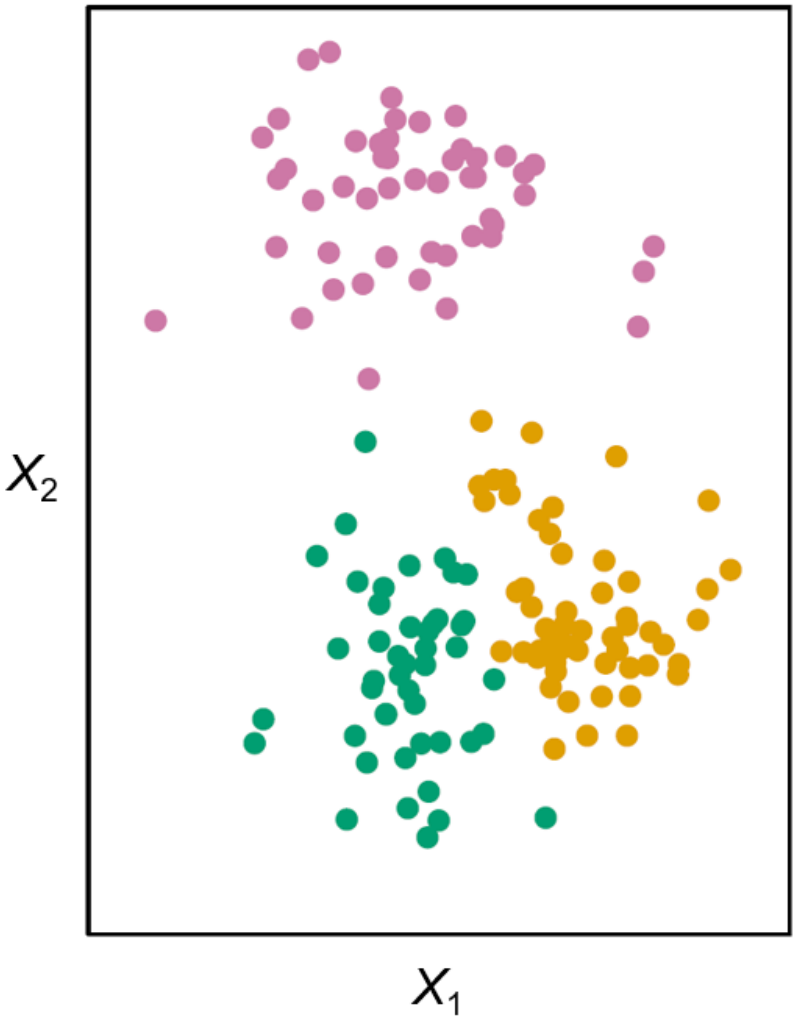
Clustering



Objective of clustering: partition observations into distinct groups according to some *distance function* (e.g., euclidean distance) such that:

- within-group similarity is *maximized*,
- between-group similarity is *minimized*.

Clustering

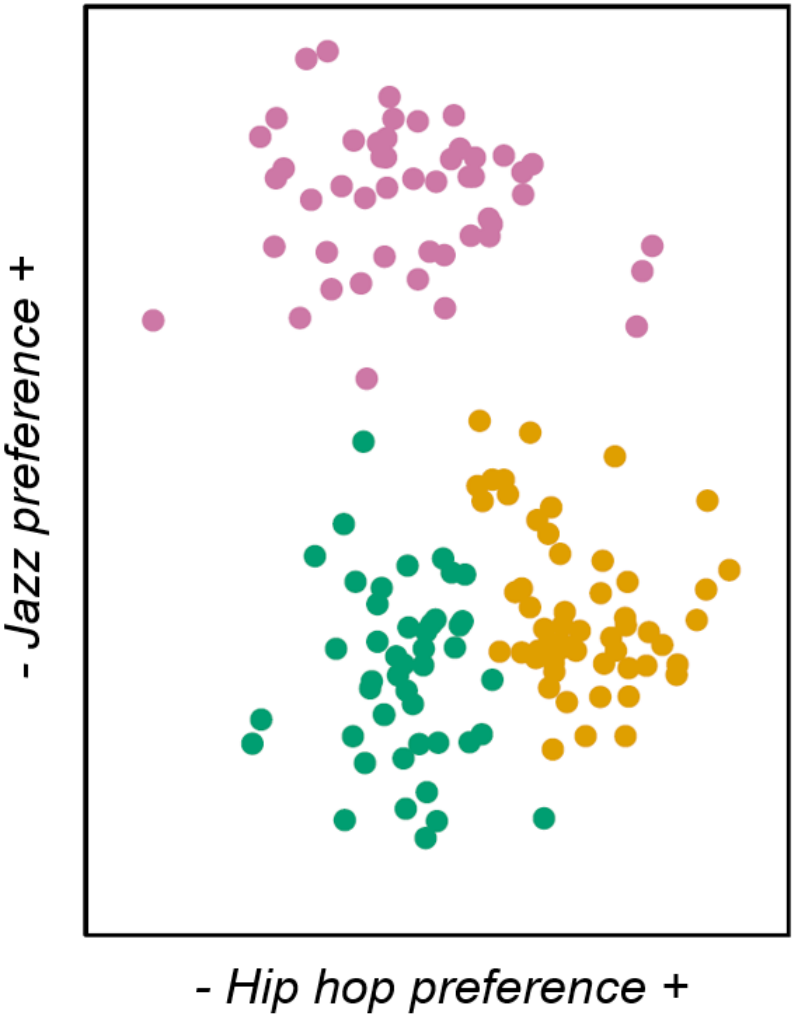


Objective of clustering: partition observations into distinct groups according to some *distance function* (e.g., euclidean distance) such that:

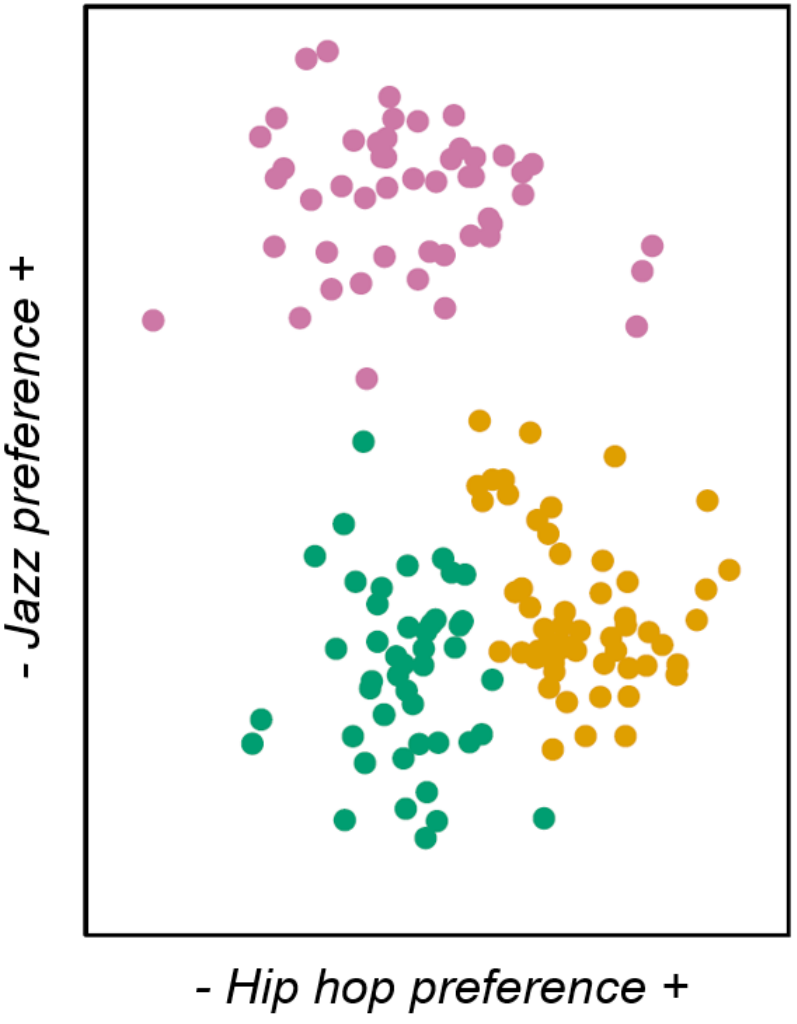
- within-group similarity is *maximized*,
- between-group similarity is *minimized*.

Here is one such partitioning of distinct groups.

How is this relevant for social scientists?

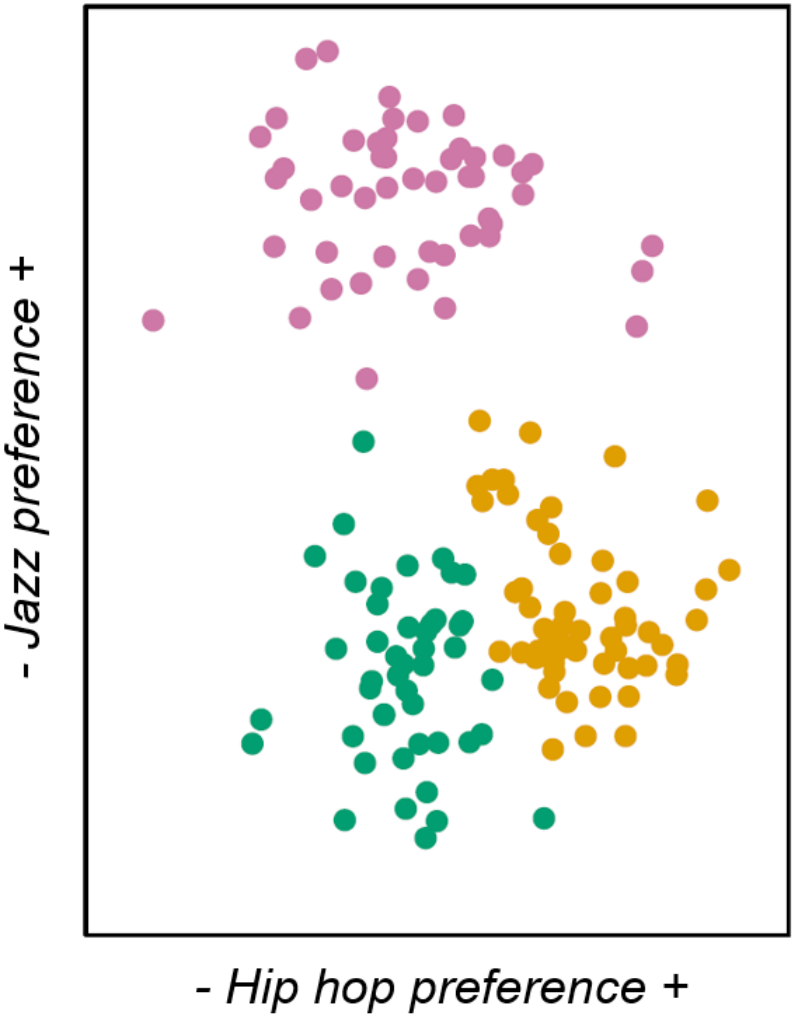


How is this relevant for social scientists?



Can help us discover/measure heterogeneity in terms of known or unknown subpopulations that may be of social/theoretical relevance.

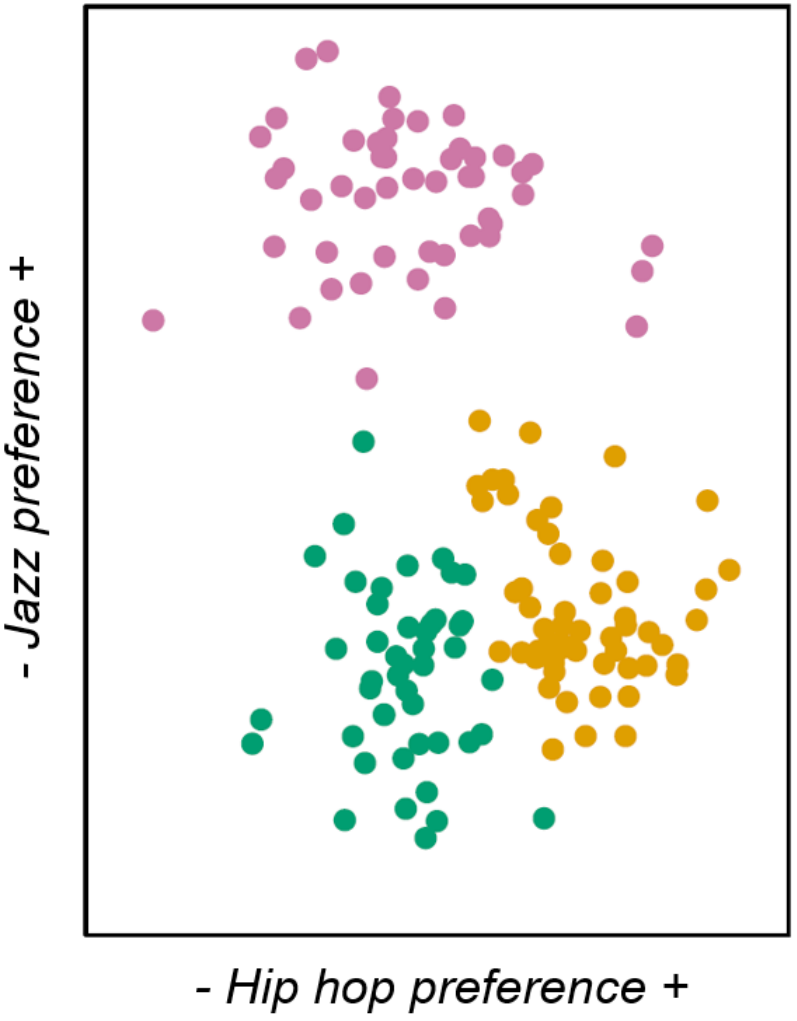
How is this relevant for social scientists?



Can help us discover/measure heterogeneity in terms of known or unknown subpopulations that may be of social/theoretical relevance.

Example—Heterogeneity in music tastes.

How is this relevant for social scientists?

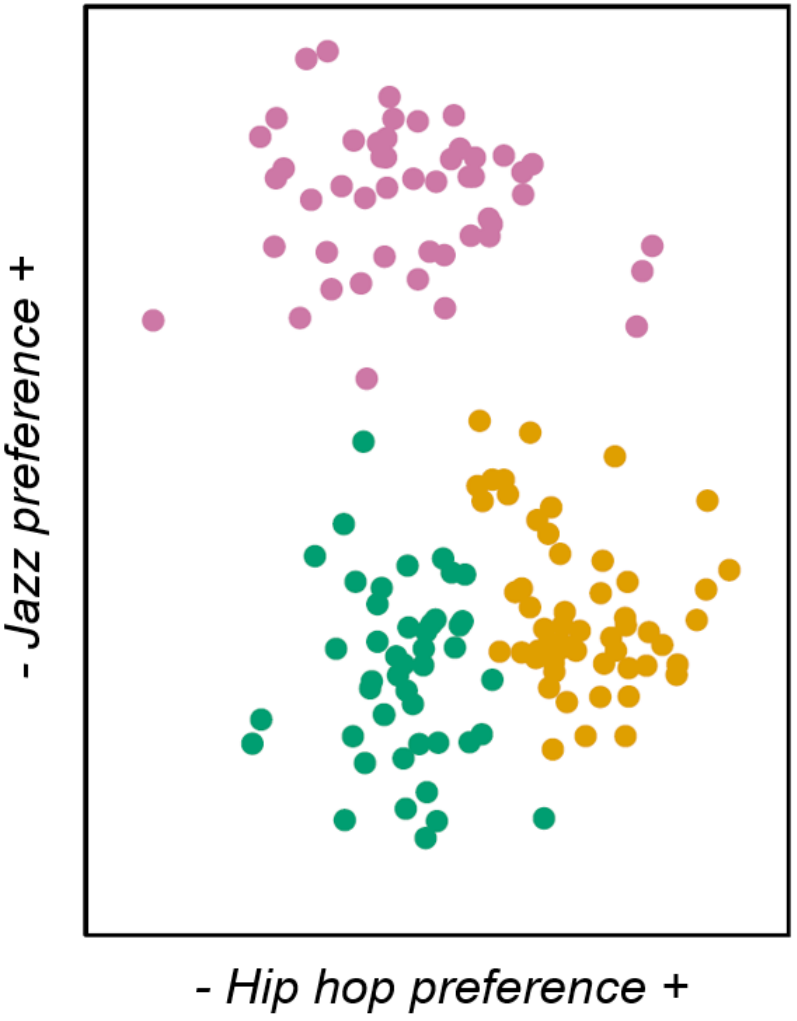


Can help us discover/measure heterogeneity in terms of known or unknown subpopulations that may be of social/theoretical relevance.

Example—Heterogeneity in music tastes.

- One group likes hip hop alot, but not jazz

How is this relevant for social scientists?

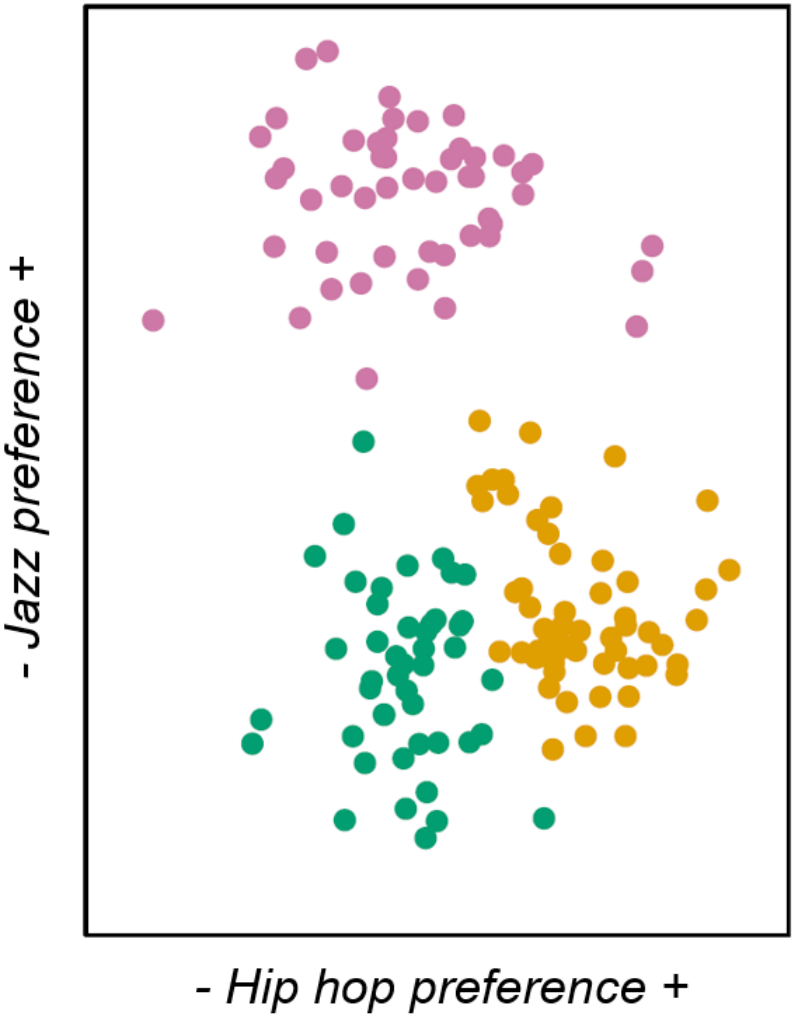


Can help us discover/measure heterogeneity in terms of known or unknown subpopulations that may be of social/theoretical relevance.

Example—Heterogeneity in music tastes.

- One group likes hip hop alot, but not jazz
- Another enoys hip hop moderately, but not jazz

How is this relevant for social scientists?



Can help us discover/measure heterogeneity in terms of known or unknown subpopulations that may be of social/theoretical relevance.

Example—Heterogeneity in music tastes.

- One group likes hip hop alot, but not jazz
- Another enoys hip hop moderately, but not jazz
- A third enjoys hip hop moderadely, and likes jazz a lot.

Types of clustering

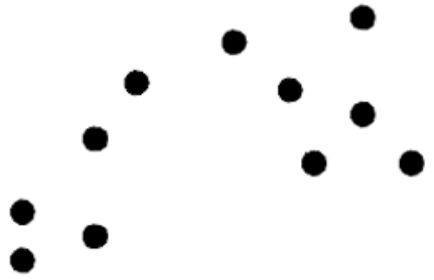
We can fruitfully categorize clustering according to three properties:

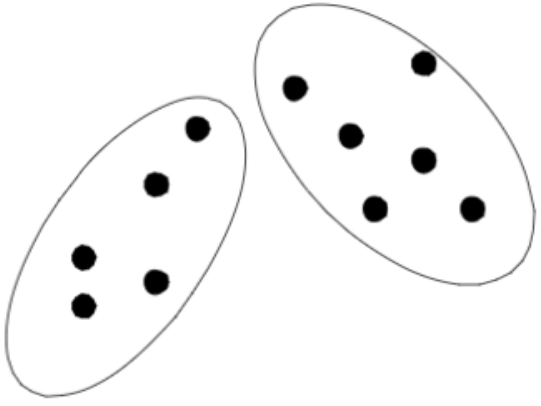
- *Flat* vs. *hiearchical*
- *Hard* vs. *soft*.
- *Model-based*[10] vs. *algorithmic*

[10] Also called *"Probabilistic"*.

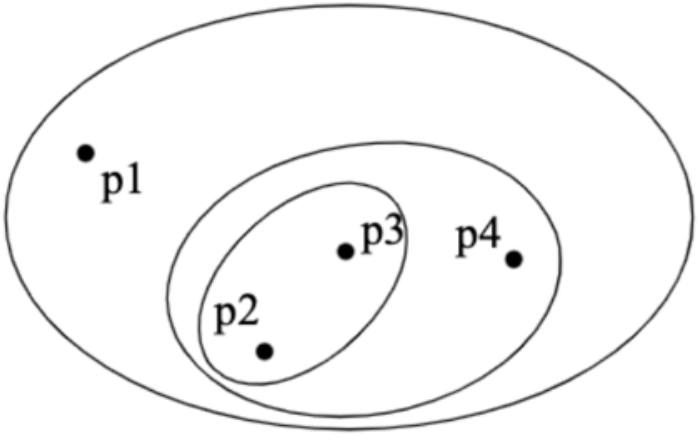
Flat vs. hiearchical

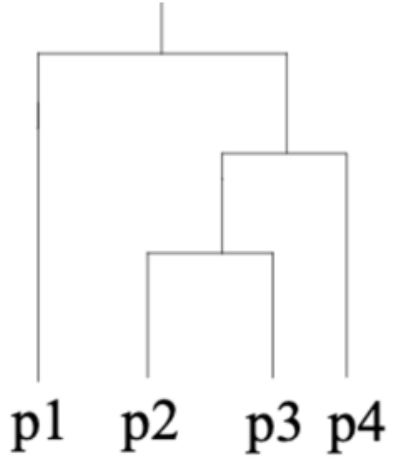
FLAT



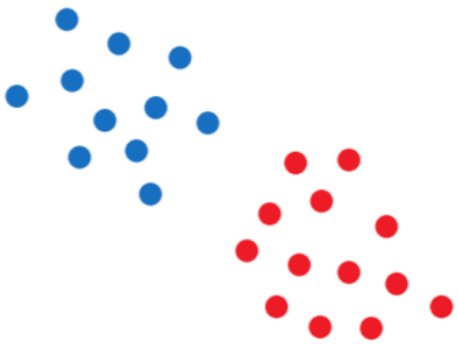


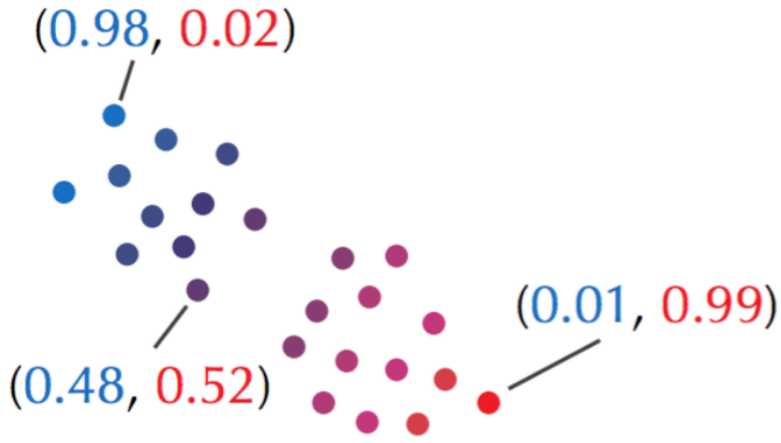
HIERARCHICAL





Hard vs. soft

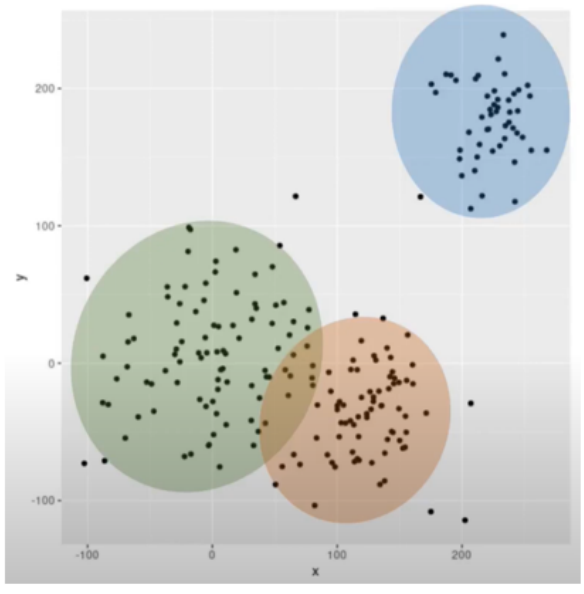




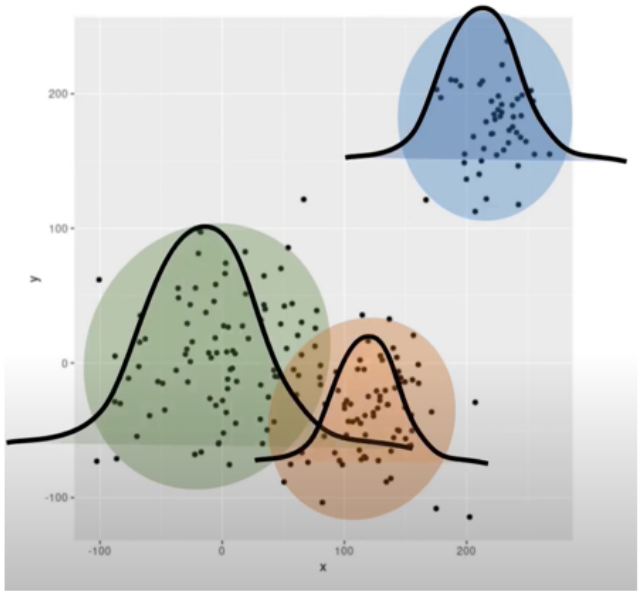
Model-based vs. algorithmic

Do we assume that data comes from some distribution?

NO



YES



What we will consider today

K-means

- Flat
- Hard
- Algorithmic

What we will consider today

K-means

- Flat
- Hard
- Algorithmic

In bonus slides: **Mixture models**

- Flat
- Soft
- Model-based

K-means

Basic idea:

- We take the starting-point that—a good clustering is one where the within-cluster variation is as *small as possible*.

K-means

Basic idea:

- We take the starting-point that—a good clustering is one where the within-cluster variation is as *small as possible*.
- *Within-cluster variation* $= W(C_k) =$ measure of how much observations within a cluster C_k *differ from each other*.

K-means

Basic idea:

- We take the starting-point that—a good clustering is one where the within-cluster variation is as *small as possible*.
- *Within-cluster variation* $= W(C_k) =$ measure of how much observations within a cluster C_k *differ from each other*.
- Thus, we—formally—want to:

$$\text{minimize} \quad \sum_{k=1}^{K} W(C_k)$$

K-means

Basic idea:

- We take the starting-point that—a good clustering is one where the within-cluster variation is as *small as possible*.
- *Within-cluster variation* $= W(C_k) =$ measure of how much observations within a cluster C_k *differ from each other*.
- Thus, we—formally—want to:

$$\text{minimize} \quad \sum_{k=1}^{K} W(C_k)$$

- **In words**: we want to partition obs. into K clusters such that the amount by which observations *differ within clusters is minimized*.

K-means

Basic idea:

- We take the starting-point that—a good clustering is one where the within-cluster variation is as *small as possible*.
- *Within-cluster variation* $= W(C_k) =$ measure of how much observations within a cluster C_k *differ from each other*.
- Thus, we—formally—want to:

$$\text{minimize} \quad \sum_{k=1}^{K} W(C_k)$$

- **In words**: we want to partition obs. into K clusters such that the amount by which observations *differ within clusters is minimized*.
- How to measure *distance/variation*? Euclidian distance

$$\text{minimize} \quad \sum_{k=1}^{K} \underbrace{\frac{1}{|C_k|}}_{C_k \text{ size}} \underbrace{\sum_{i,i' \in C_k}}_{\text{All pairs } i,i' \text{ in } C_k} \underbrace{\sum_{j=1}^{p} (x_{ij} - x_{i',j})^2}_{\text{Euclidian dist.}}$$

How do we *minimize* this expression?

K-means algorithm:

1. Randomly initialize cluster assignments.

How do we *minimize* this expression?

K-means algorithm:

1. Randomly initialize cluster assignments.
2. Iterate until the cluster assignments stop changing:

How do we *minimize* this expression?

K-means algorithm:

1. Randomly initialize cluster assignments.
2. Iterate until the cluster assignments stop changing:
    a. Holding *cluster assignments fixed*—calculate cluster centroid (vector of *feature means*) for each cluster.

How do we *minimize* this expression?

K-means algorithm:

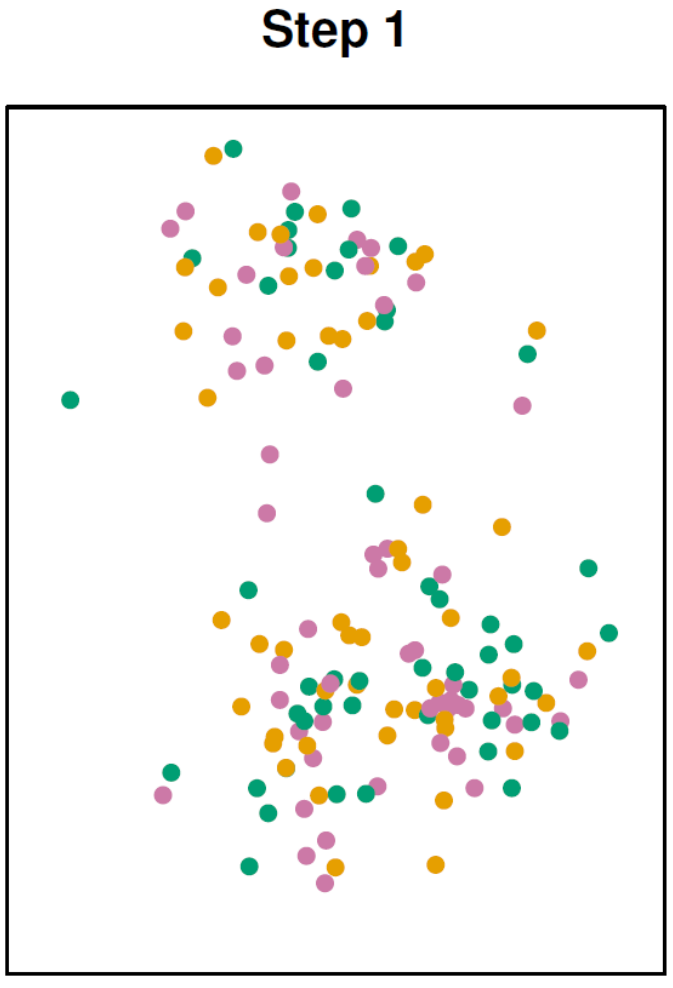
1. Randomly initialize cluster assignments.
2. Iterate until the cluster assignments stop changing:
    a. Holding *cluster assignments fixed*—calculate cluster centroid (vector of *feature means*) for each cluster.
    b. Holding *cluster centroids fixed*—assign each observation to the cluster whose centroid is closest.

Illustration

The original data

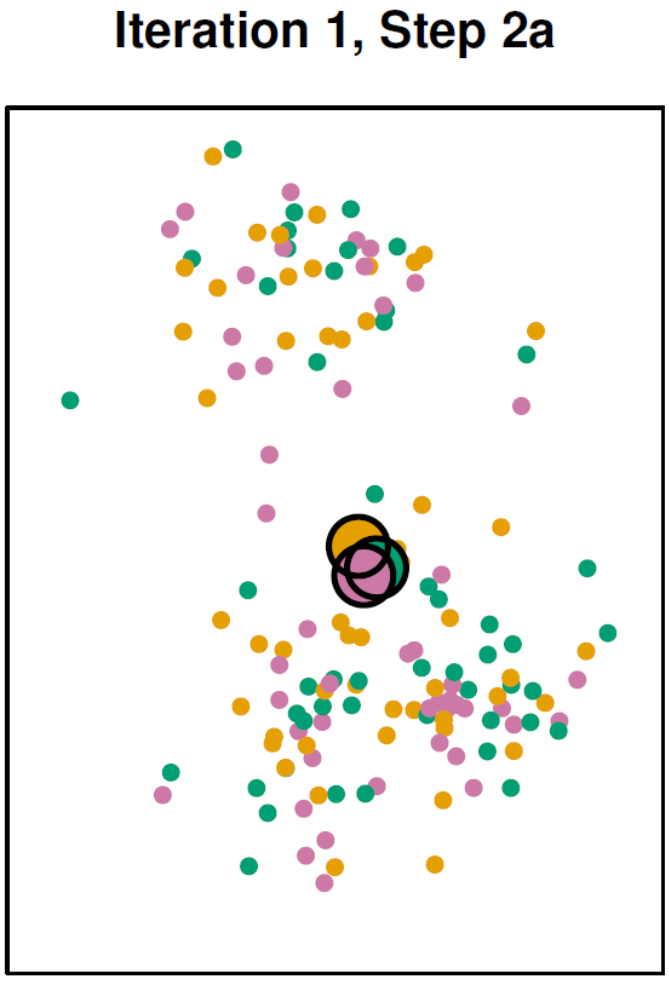
Illustration

Step 1—Random initialization of cluster assignments.



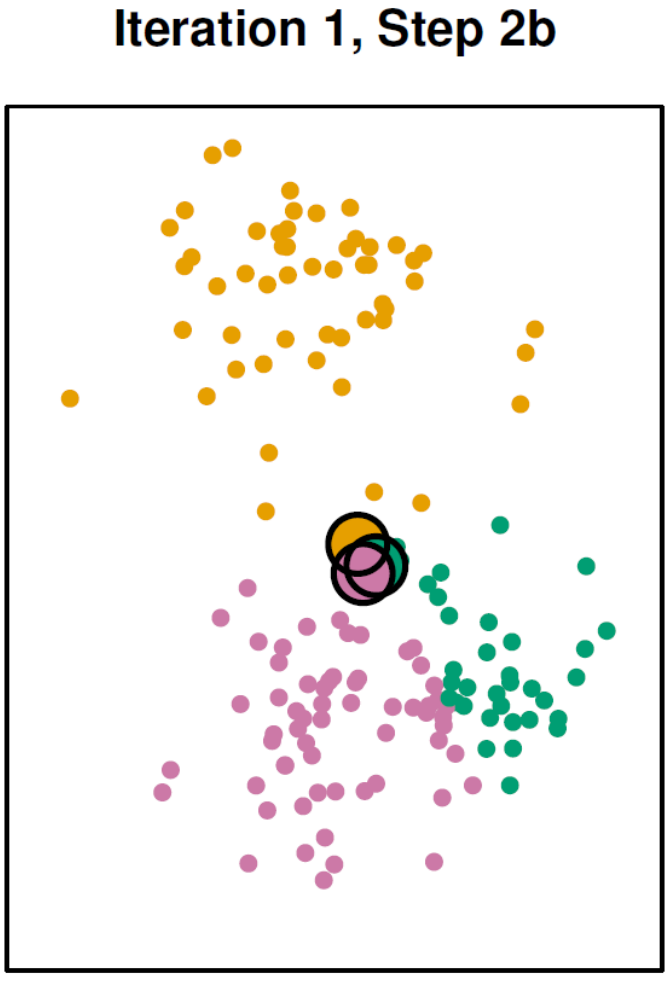
Illustration

Step 2a—Calculate centroids for each cluster.



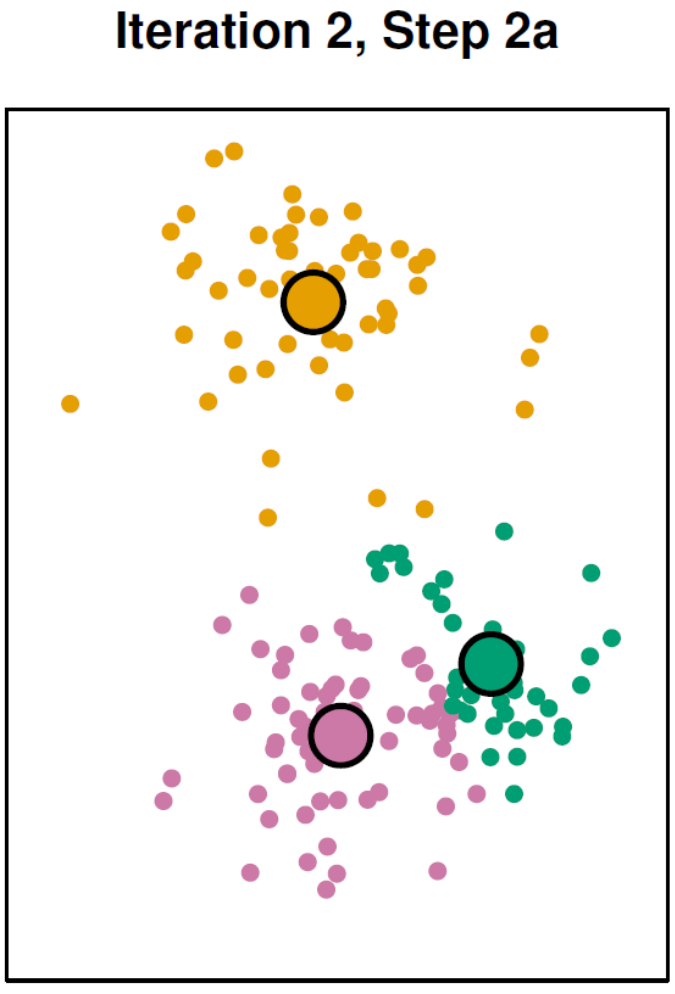
Illustration

Step 2b—Assign each observation to the nearest centroid.



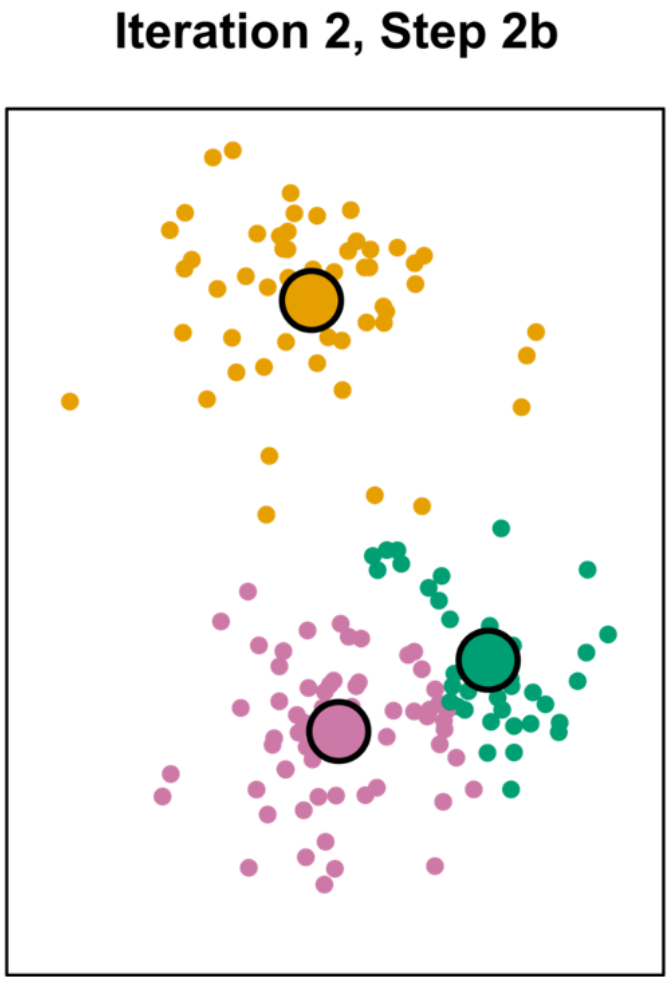
Illustration

Step 2a—Update position of cluster centroids.



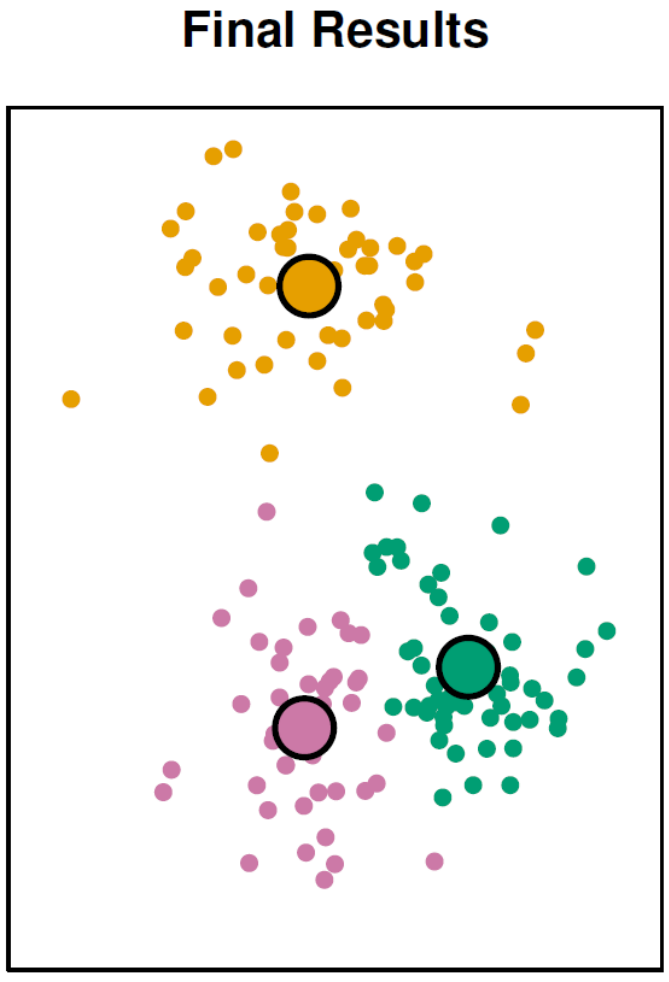
Illustration

Step 2b—Re-assign observations to nearest centroid.



Final results

After 10 iterations of 2a–2b, we get the following:



Sensitivity to starting values values

K-means is sensitive to the *initial random assignment*.

Sensitivity to starting values values

K-means is sensitive to the *initial random assignment*. Example from ISL:

Sensitivity to starting values values

K-means is sensitive to the *initial random assignment*. Example from ISL:

- Three different local optima across 6 runs.
- 3/6 reached the same final (and best) solution.

Sensitivity to starting values values

K-means is sensitive to the *initial random assignment*. Example from ISL:

- Three different local optima across 6 runs.
- 3/6 reached the same final (and best) solution.

$\rightarrow$ R re-runs several times with different initial values; then keep best.

How to select K?

K—dictating the number of clusters to obtain—is a parameter which the researcher must *select prior* to performing the clustering.

How to select K?

K—dictating the number of clusters to obtain—is a parameter which the researcher must *select prior* to performing the clustering.

Quantitative basis

How to select K?

K—dictating the number of clusters to obtain—is a parameter which the researcher must *select prior* to performing the clustering.

Quantitative basis

- We want to optimize two aspects: (*i*) max. the amount variance accounted for by clusters, (*ii*) min. the complexity degree (number of clusters).

How to select K?

K—dictating the number of clusters to obtain—is a parameter which the researcher must *select prior* to performing the clustering.

Quantitative basis

- We want to optimize two aspects: (*i*) max. the amount variance accounted for by clusters, (*ii*) min. the complexity degree (number of clusters).
- Popular approach: the *"elbow"* method:
  1. Fit *k-means* for different values of k (e.g. 1–20).
  2. For each k, calculate the *total within-cluster sum of squares* (WSS).
  3. Plot WSS against the number of clusters k
  4. Identify the location of a "bend" (i.e., elbow) in the plot—point at which adding extra complexity only bring marginal returns.

How to select K?

K—dictating the number of clusters to obtain—is a parameter which the researcher must *select prior* to performing the clustering.

Quantitative basis

- We want to optimize two aspects: (*i*) max. the amount variance accounted for by clusters, (*ii*) min. the complexity degree (number of clusters).
- Popular approach: the *"elbow"* method:
  1. Fit *k-means* for different values of k (e.g. 1–20).
  2. For each k, calculate the *total within-cluster sum of squares* (WSS).
  3. Plot WSS against the number of clusters k
  4. Identify the location of a "bend" (i.e., elbow) in the plot—point at which adding extra complexity only bring marginal returns.

Qualitative basis

- Of course, ultimately, the goal of clustering is to provide a "useful" partitioning of the data. The above procedure does not ensure this.

How to select K?

K—dictating the number of clusters to obtain—is a parameter which the researcher must *select prior* to performing the clustering.

Quantitative basis

- We want to optimize two aspects: (*i*) max. the amount variance accounted for by clusters, (*ii*) min. the complexity degree (number of clusters).
- Popular approach: the *"elbow"* method:
  1. Fit *k-means* for different values of k (e.g. 1–20).
  2. For each k, calculate the *total within-cluster sum of squares* (WSS).
  3. Plot WSS against the number of clusters k
  4. Identify the location of a "bend" (i.e., elbow) in the plot—point at which adding extra complexity only bring marginal returns.

Qualitative basis

- Of course, ultimately, the goal of clustering is to provide a "useful" partitioning of the data. The above procedure does not ensure this.
- Hence, it is also recommended that the analyst performs *qualitative* checks of the clusters—and let that guide her choice *jointly* with the *quantitative* insights.

How to select K?

K—dictating the number of clusters to obtain—is a parameter which the researcher must *select prior* to performing the clustering.

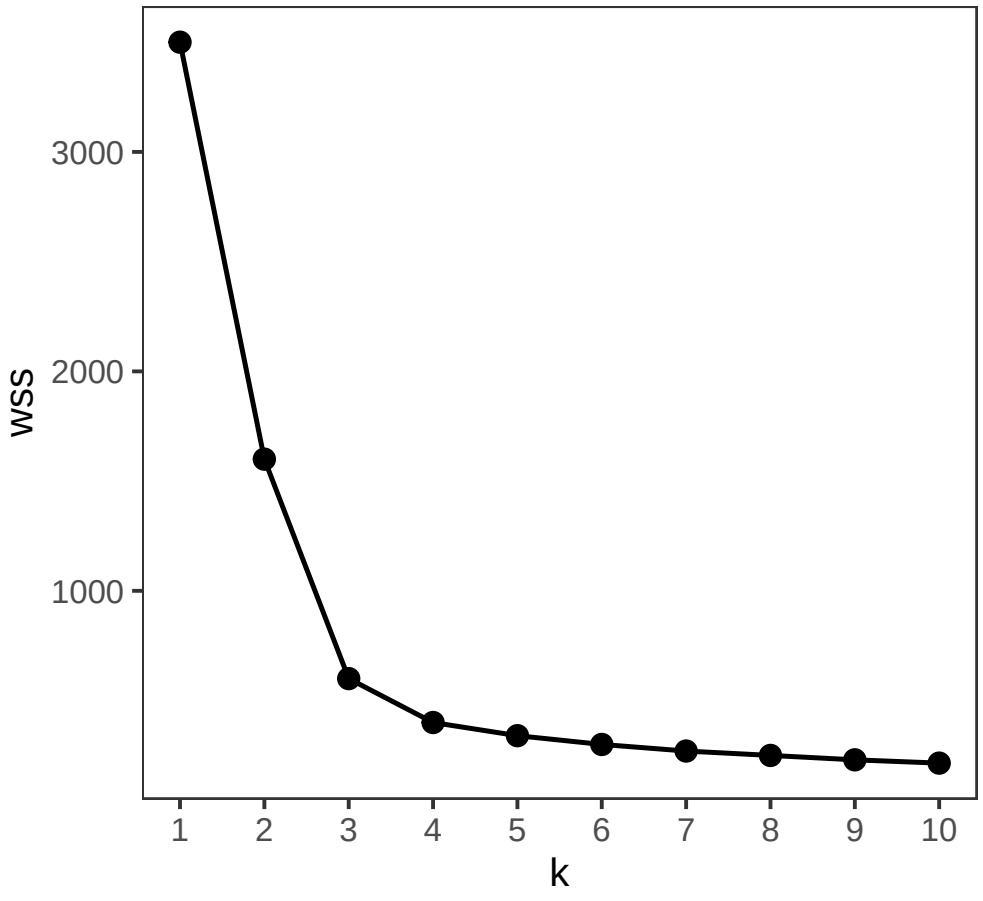
Quantitative basis

- We want to optimize two aspects: (*i*) max. the amount variance accounted for by clusters, (*ii*) min. the complexity degree (number of clusters).
- Popular approach: the *"elbow"* method:
  1. Fit *k-means* for different values of k (e.g. 1–20).
  2. For each k, calculate the *total within-cluster sum of squares* (WSS).
  3. Plot WSS against the number of clusters k
  4. Identify the location of a "bend" (i.e., elbow) in the plot—point at which adding extra complexity only bring marginal returns.

Qualitative basis

- Of course, ultimately, the goal of clustering is to provide a "useful" partitioning of the data. The above procedure does not ensure this.
- Hence, it is also recommended that the analyst performs *qualitative* checks of the clusters—and let that guide her choice *jointly* with the *quantitative* insights. We will return to this.

Identifying the elbow



A note on *interpretation* of clusters

Two common approaches for interpreting clustering results (Grimmer et al. 2022):

A note on *interpretation* of clusters

Two common approaches for interpreting clustering results (Grimmer et al. 2022):

1. **Identifying distinctive features**—variables which have larger positive/negative values in focal cluster compared to other clusters.

A note on *interpretation* of clusters

Two common approaches for interpreting clustering results (Grimmer et al. 2022):

1. **Identifying distinctive features**—variables which have larger positive/negative values in focal cluster compared to other clusters.
2. **Sampling observations** assigned to each cluster—and inspect them. If observations are text; read original texts.

Pros and cons of *K-means*

Pros

- Fast.
- Simple, intuitive.
- Often works quite well.

Pros and cons of *K-means*

Pros

- Fast.
- Simple, intuitive.
- Often works quite well.

Cons

- Assigns observations to a given cluster with 100% certainity
- Limited ability to model different cluster properties (e.g., spread/variance).
- Sensitive to outliers.

Pros and cons of *K-means*

Pros

- Fast.
- Simple, intuitive.
- Often works quite well.

Cons

- Assigns observations to a given cluster with 100% certainity
- Limited ability to model different cluster properties (e.g., spread/variance).
- Sensitive to outliers.

In the computational text analysis course (in the spring?) you will learn more sophisticated ways to cluster data.