Lecture 8. Deep Learning. Convolutional ANNs. Autoencoders

COMP90051 Statistical Machine Learning

Semester 2, 2018
Lecturer: Ben Rubinstein

THE UNIVERSITY OF
MELBOURNE

This lecture

- Deep learning
  - Representation capacity
  - Deep models and representation learning
- Convolutional Neural Networks
  - Convolution operator
  - Elements of a convolution-based network
- Autoencoders
  - Learning efficient coding

Deep Learning and Representation Learning

Hidden layers viewed as
feature space transformation

Representational capacity

- ANNs with a single hidden layer are universal approximators
- For example, such ANNs can represent any Boolean function

$$OR(x_1, x_2) \qquad u = g(x_1 + x_2 - 0.5)$$

$$AND(x_1, x_2) \qquad u = g(x_1 + x_2 - 1.5)$$

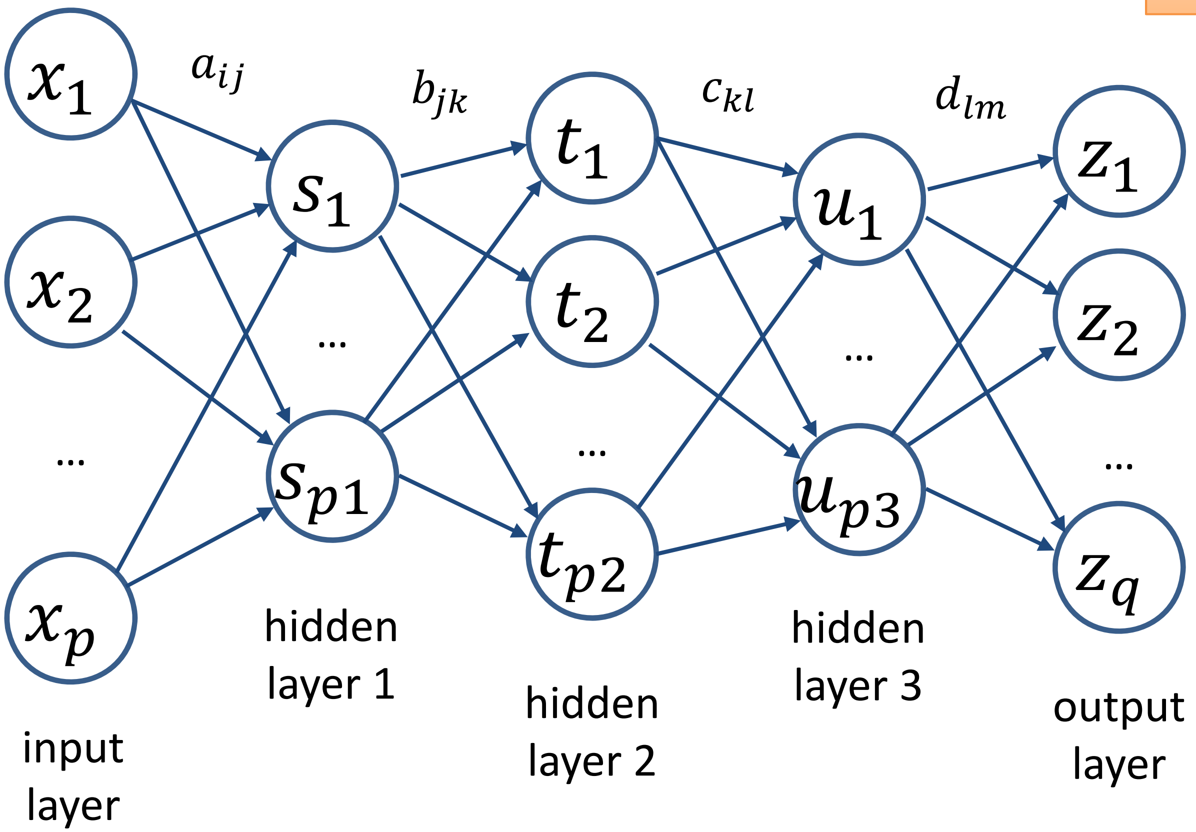
$$NOT(x_1) \qquad u = g(-x_1)$$

$g(r) = 1$ if $r \geq 0$ and $g(r) = 0$ otherwise

- Any Boolean function over m variables can be implemented using a hidden layer with up to 2^m elements
- More *efficient* to stack several hidden layers

Deep networks

"Depth" refers to number of hidden layers



$$\boldsymbol{s} = \tanh(\boldsymbol{A}'\boldsymbol{x}) \quad \boldsymbol{t} = \tanh(\boldsymbol{B}'\boldsymbol{s}) \quad \boldsymbol{u} = \tanh(\boldsymbol{C}'\boldsymbol{t}) \quad \boldsymbol{z} = \tanh(\boldsymbol{D}'\boldsymbol{u})$$

Deep ANNs as representation learning

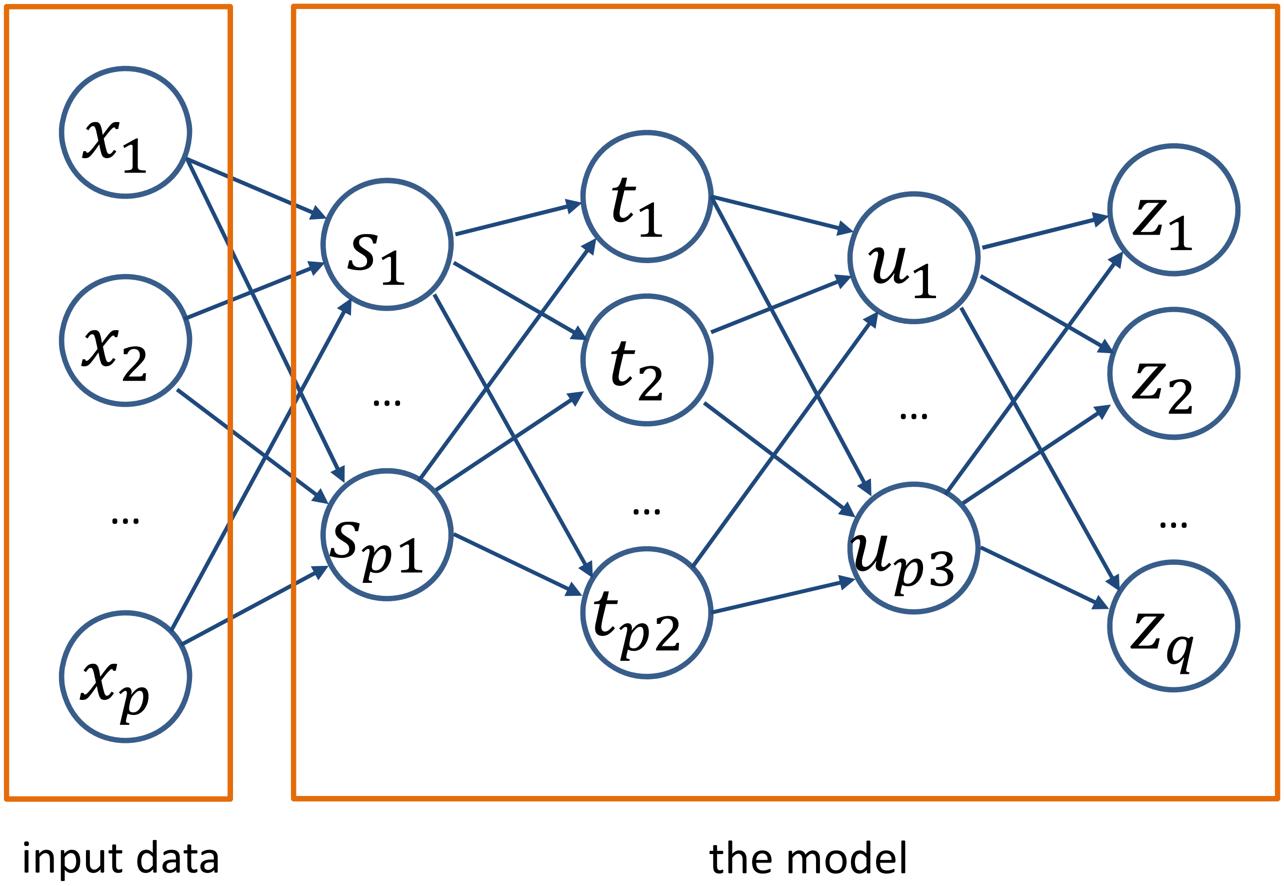
- Consecutive layers form *representations* of the input of increasing complexity
- An ANN can have a simple *linear* output layer, but using complex *non-linear* representation

$$\boldsymbol{z} = \tanh\left(\boldsymbol{D}'\left(\tanh\left(\boldsymbol{C}'(\tanh(\boldsymbol{B}'(\tanh(\boldsymbol{A}'\boldsymbol{x}))))\right)\right)\right)$$

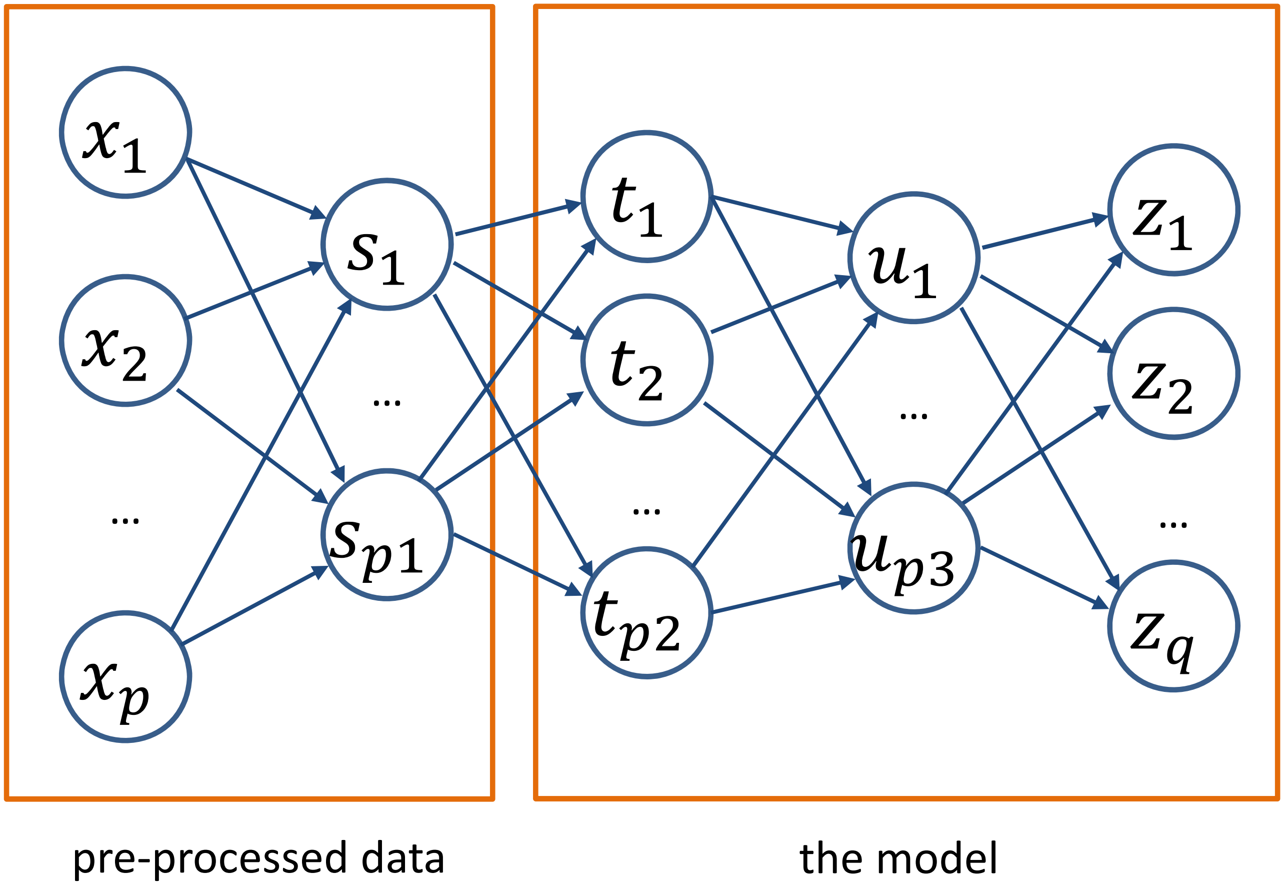
- Equivalently, a hidden layer can be thought of as the transformed feature space, e.g., $\boldsymbol{u} = \varphi(\boldsymbol{x})$
- Parameters of such a transformation are learned from data

Bias terms are omitted for simplicity

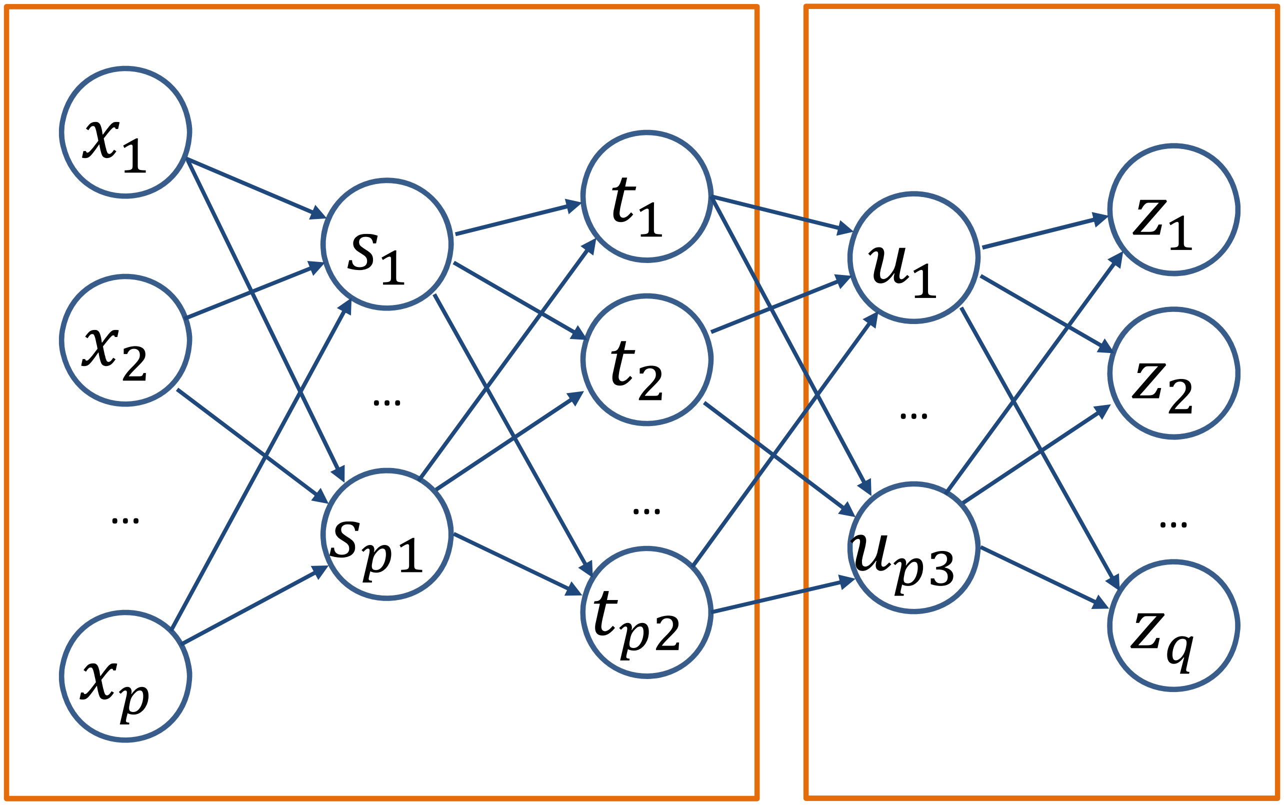
ANN layers as data transformation



ANN layers as data transformation



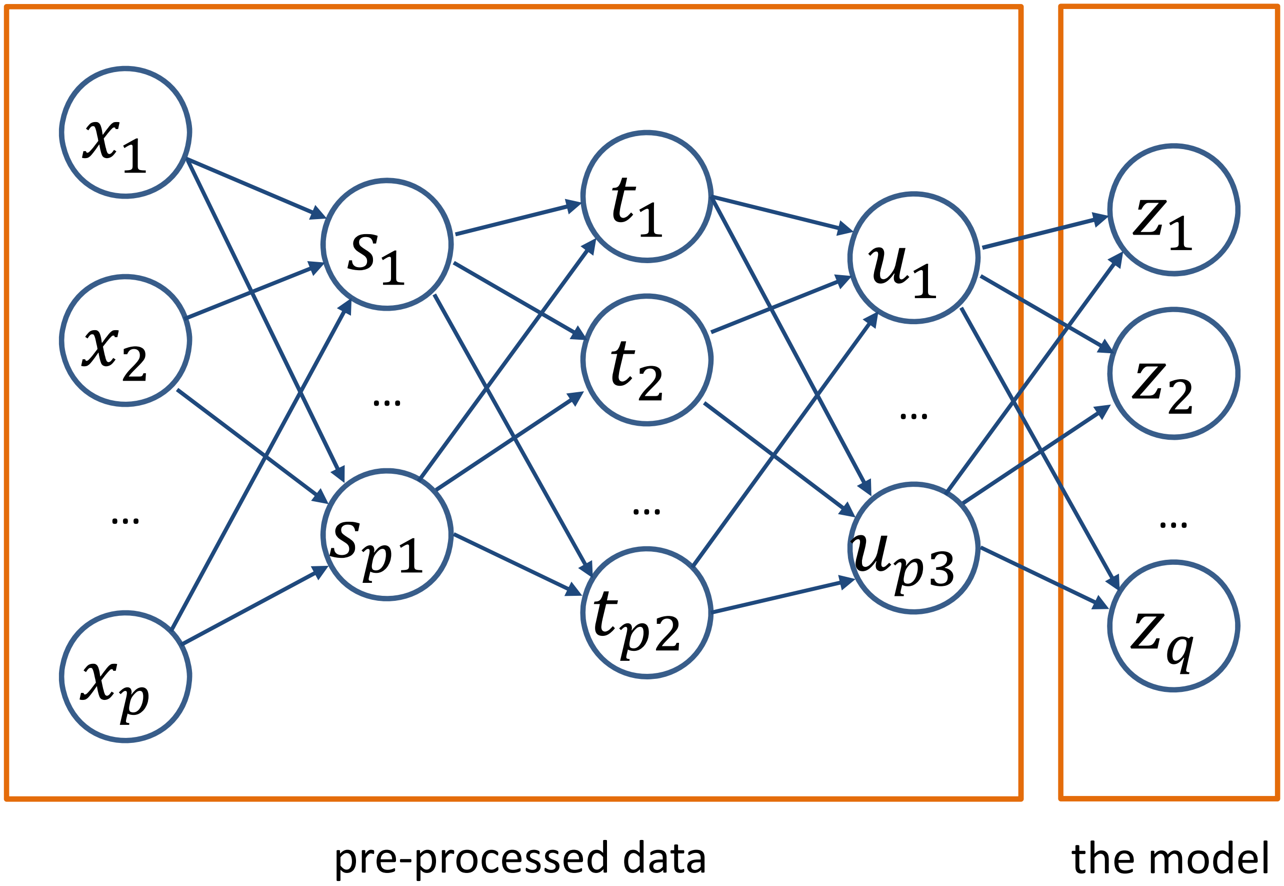
ANN layers as data transformation



pre-processed data

the model

ANN layers as data transformation



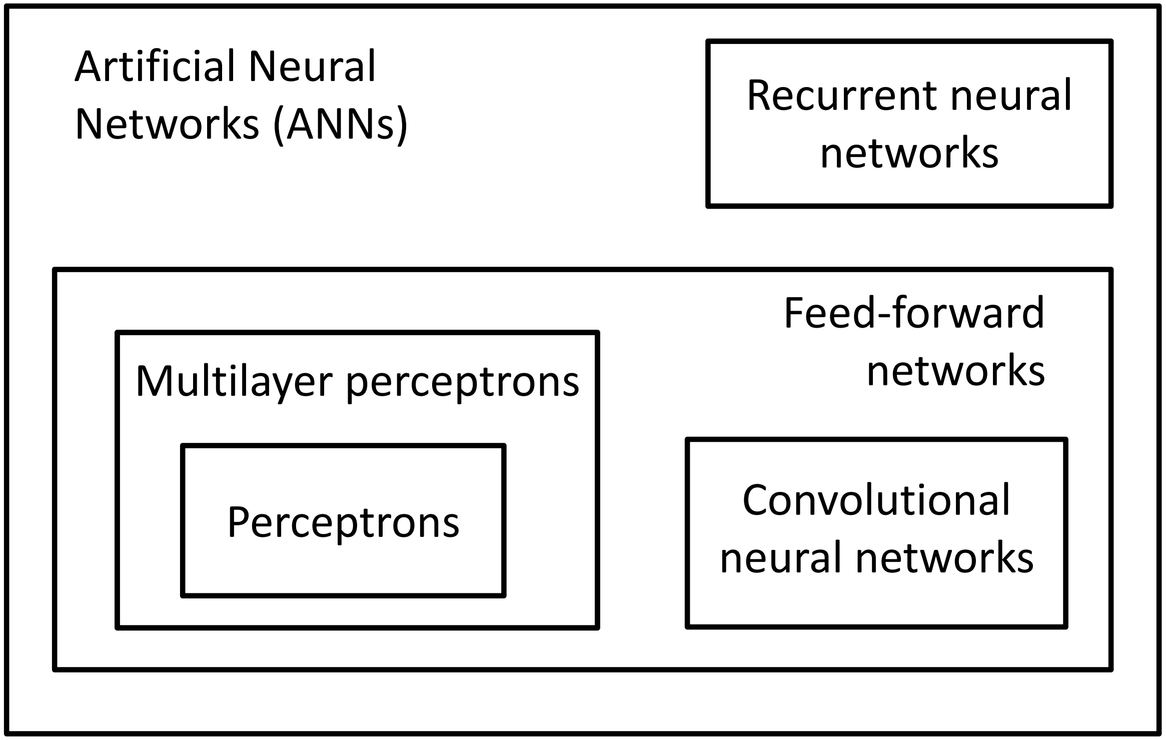
Depth vs width

- A single infinitely wide layer in theory gives a universal approximator
- However (empirically) depth yields more accurate models

  Biological inspiration from the eye:

  * first detect small edges and color patches;
  * compose these into smaller shapes;
  * building to more complex detectors, of e.g. textures, faces, etc.
- Seek to mimic layered complexity in a network
- However *vanishing gradient problem* affects learning with very deep models

Animals in the zoo


Artificial Neural Networks (ANNs)

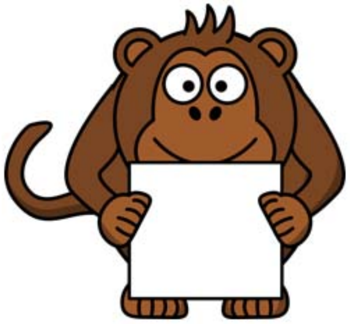
Recurrent neural networks

Feed-forward networks

Multilayer perceptrons

Perceptrons

Convolutional neural networks



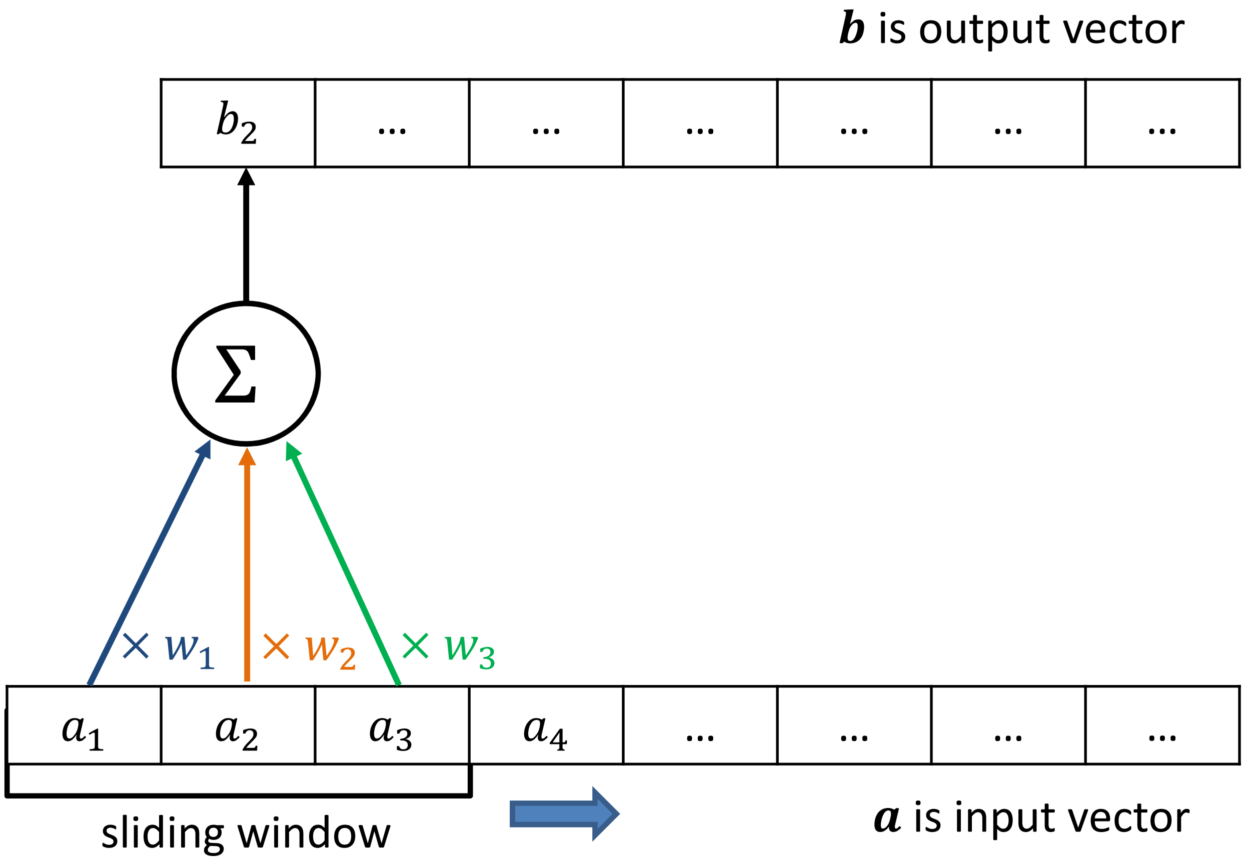
art: OpenClipartVectors at pixabay.com (CC0)

- Recurrent neural networks are not covered in this subject
- An autoencoder is an ANN trained in a specific way.
  - E.g., a multilayer perceptron can be trained as an autoencoder, or a recurrent neural network can be trained as an autoencoder.

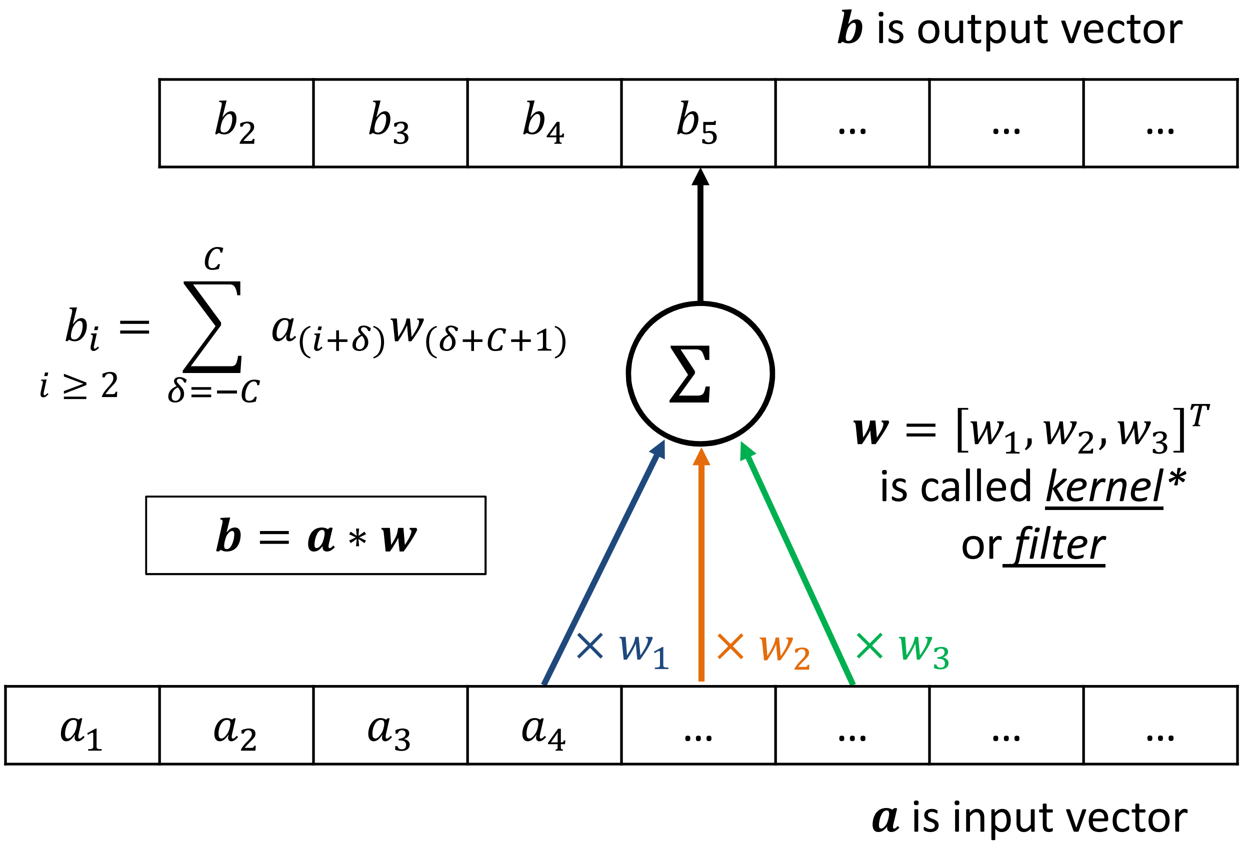
Convolutional
Neural Networks (CNN)

Based on repeated application of small filters to patches of a 2D image or range of a 1D input

Convolution



Convolution



$\boldsymbol{b}$ is output vector

$$\underset{i \geq 2}{b_i} = \sum_{\delta=-C}^{C} a_{(i+\delta)} w_{(\delta+C+1)}$$

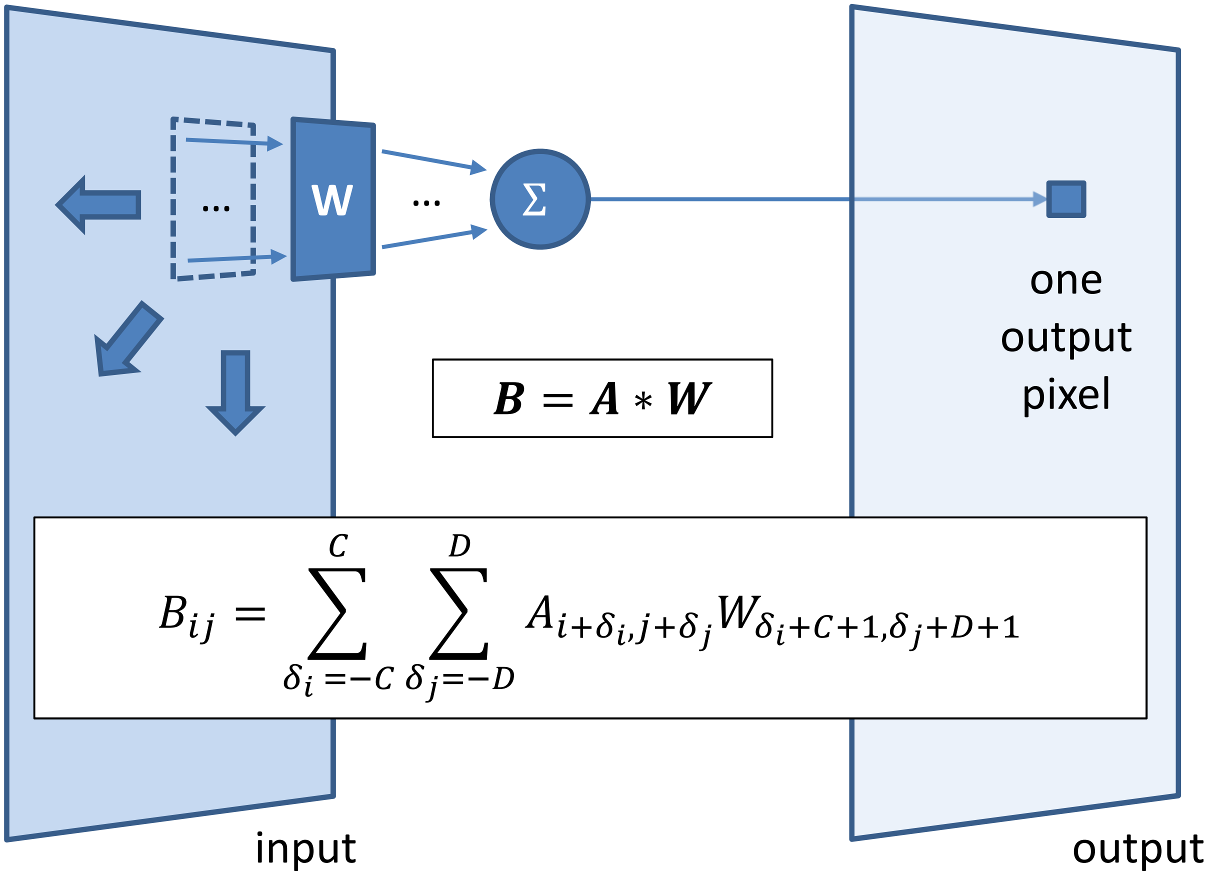
$$\boxed{\boldsymbol{b} = \boldsymbol{a} * \boldsymbol{w}}$$

$\boldsymbol{w} = [w_1, w_2, w_3]^T$ is called *kernel** or *filter*

$\boldsymbol{a}$ is input vector

*Later in the subject, we will also use an unrelated definition of kernel as a function representing a dot product

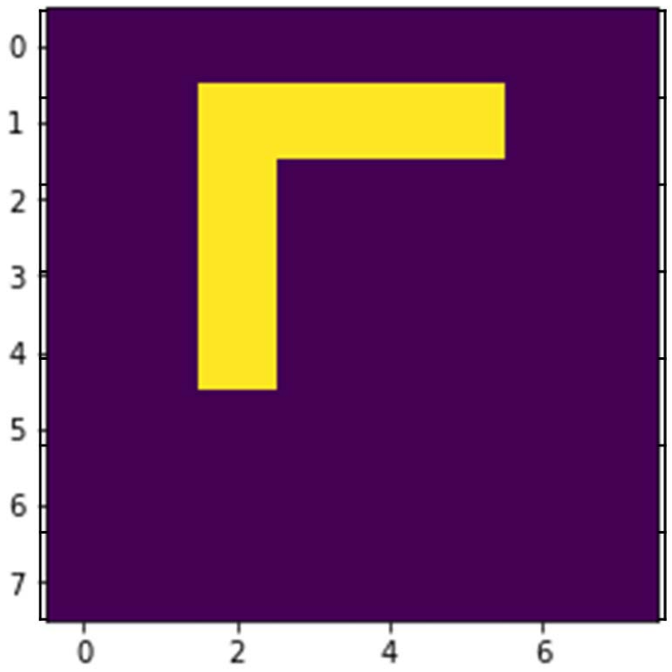
Convolution on 2D images



$$\boldsymbol{B} = \boldsymbol{A} * \boldsymbol{W}$$

$$B_{ij} = \sum_{\delta_i=-C}^{C} \sum_{\delta_j=-D}^{D} A_{i+\delta_i,j+\delta_j} W_{\delta_i+C+1,\delta_j+D+1}$$

Filters as feature detectors

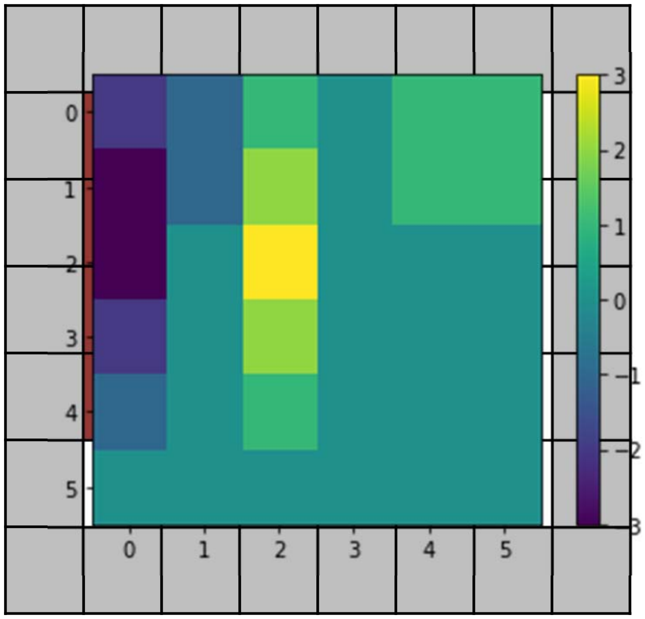


$\boldsymbol{A}$ is input image

convolve with a vertical edge filter

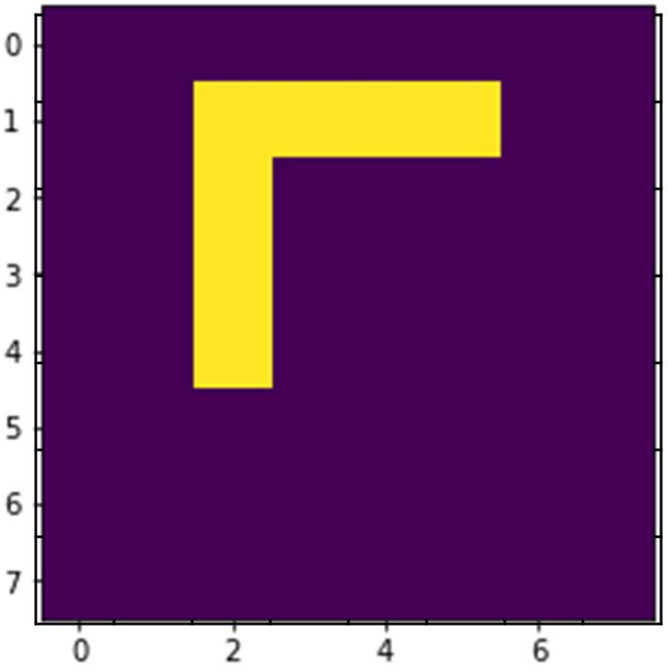
-1	0	1
-1	0	1
-1	0	1

activation function



filtered image

Filters as feature detectors

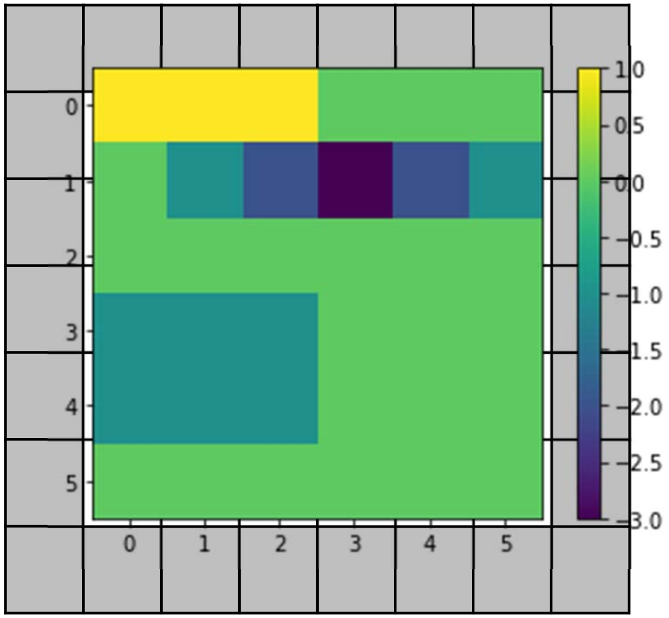


$\boldsymbol{A}$ is input image

convolve with a vertical edge filter

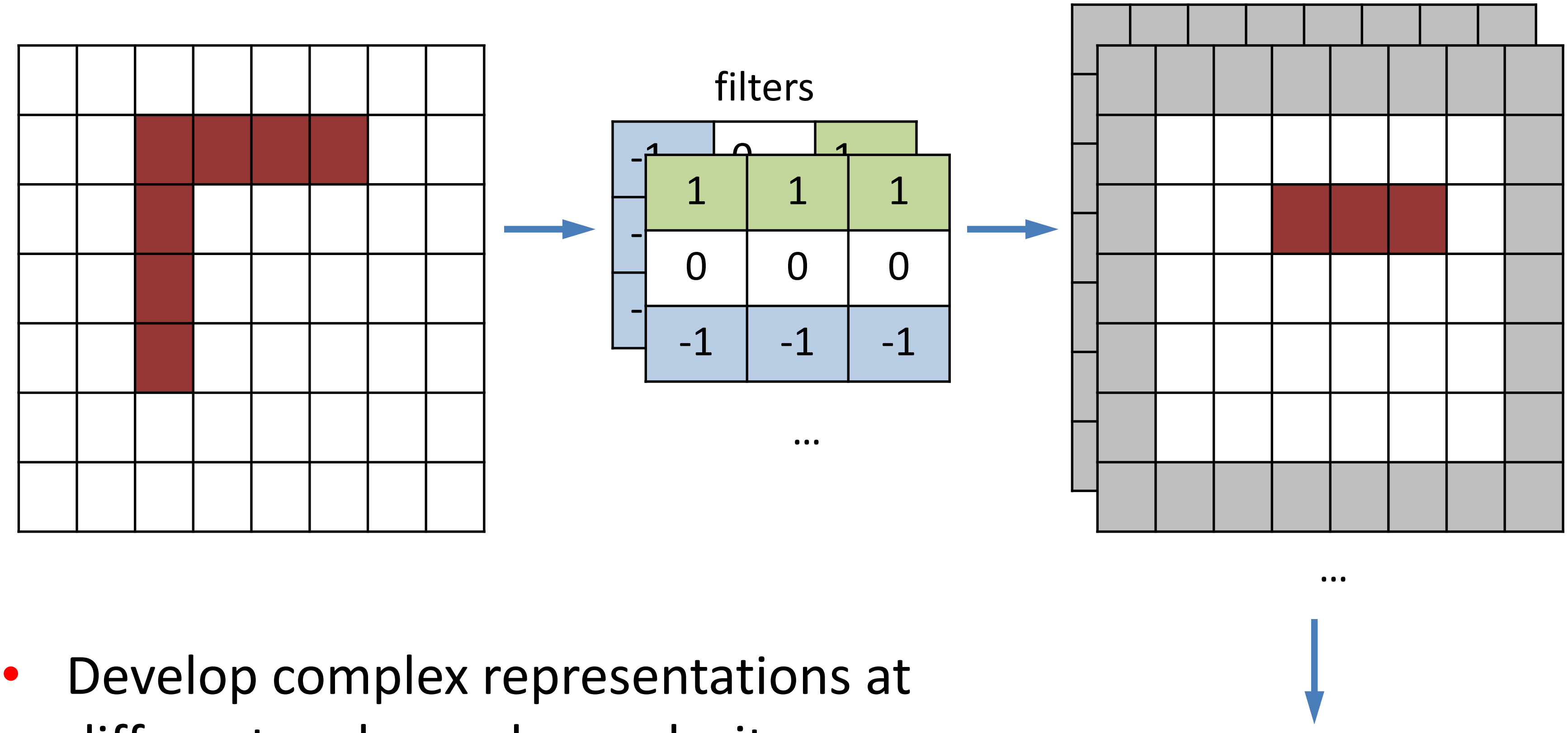
1	1	1
0	0	0
-1	-1	-1

activation function



filtered image

Stacking convolutions



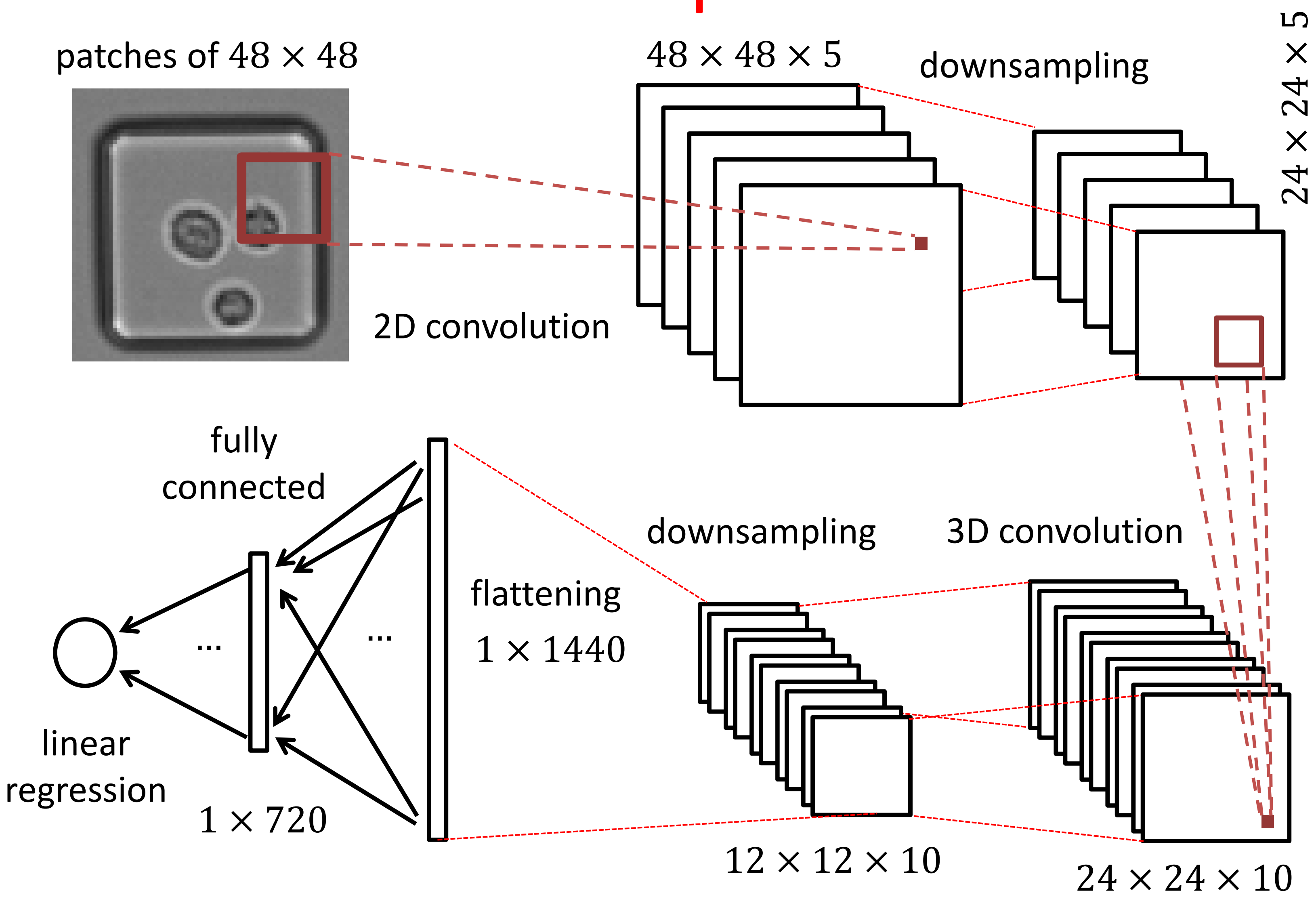
filters

1	1	1
0	0	0
-1	-1	-1

- Develop complex representations at different scales and complexity
- Filters are learned from training data!

downsampling and further convolutions

CNN for computer vision



Implemented by Jizhizi Li
based on LeNet5: http://deeplearning.net/tutorial/lenet.html

Components of a CNN

- Convolutional layers
  - Complex input representations based on convolution operation
  - Filter weights are learned from training data
- Downsampling, usually via Max Pooling
  - Re-scales to smaller resolution, limits parameter explosion
- Fully connected parts and output layer
  - Merges representations together

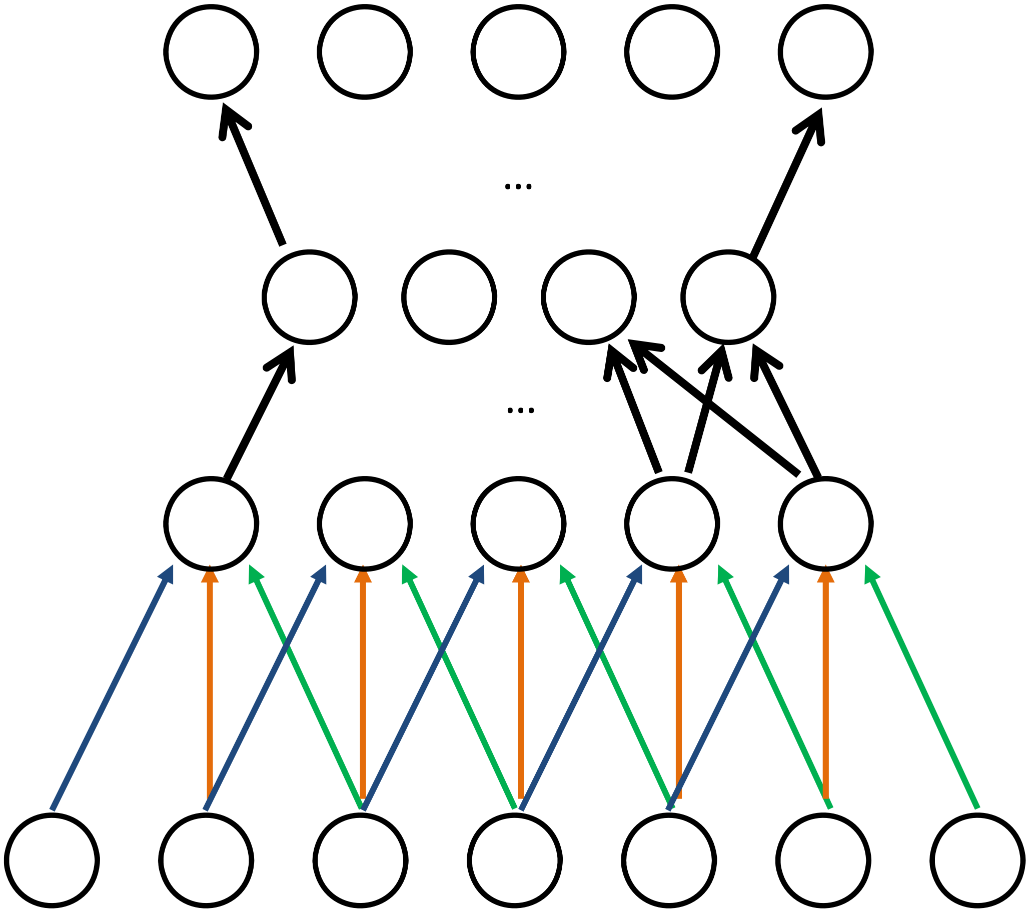
Downsampling via max pooling

- Special type of processing layer. For an $m \times m$ patch
$$v = \max(u_{11}, u_{12}, \dots, u_{mm})$$

- Strictly speaking, not everywhere differentiable. Instead, gradient is defined according to "sub-gradient"
    - Tiny changes in values of u_{ij} that is not max do not change v
    - If u_{ij} is max value, tiny changes in that value change v linearly
    - Use $\frac{\partial v}{\partial u_{ij}} = 1$ if $u_{ij} = v$, and $\frac{\partial v}{\partial u_{ij}} = 0$ otherwise

- Forward pass records maximising element, which is then used in the backward pass during back-propagation

Convolution as a regulariser



Fully connected, unrestricted

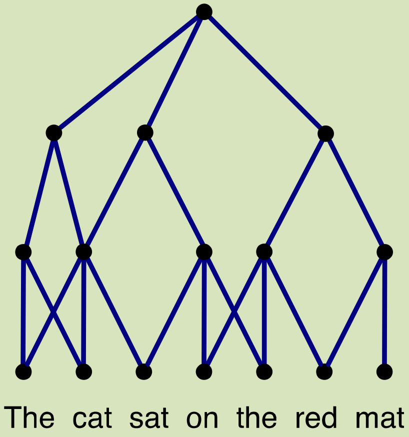
Fully connected, unrestricted

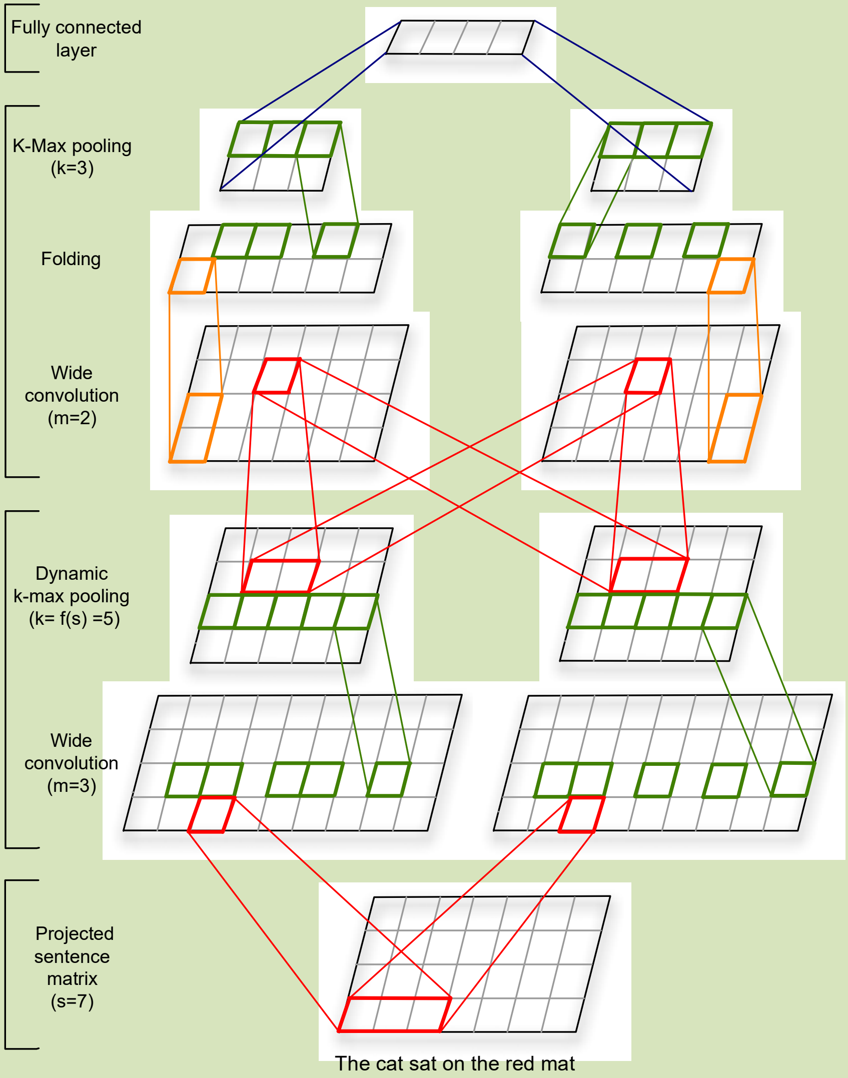
Restriction: same color – same weight

Document classification (Kalchbrenner et al, 2014)

Structure of text important for classifying documents

Capture patterns of nearby words using 1d convolutions




Fully connected layer
K-Max pooling (k=3)
Folding
Wide convolution (m=2)
Dynamic k-max pooling (k= f(s) =5)
Wide convolution (m=3)
Projected sentence matrix (s=7)
The cat sat on the red mat

Autoencoder

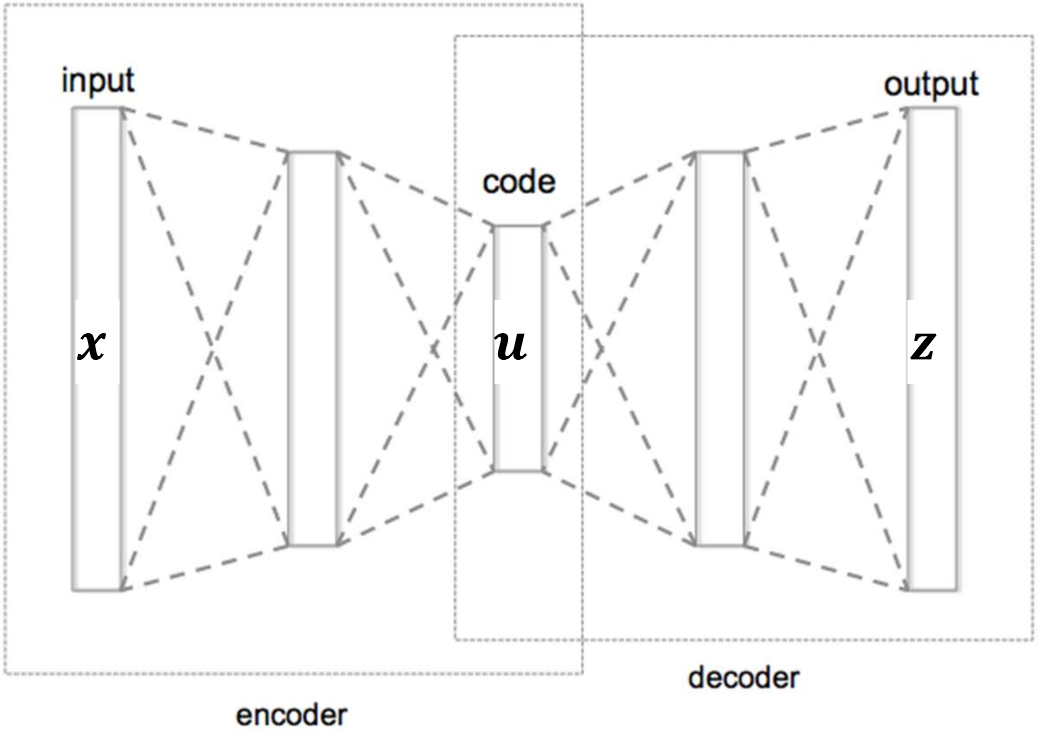
An ANN training setup that can be used for unsupervised learning, initialisation, or just efficient coding

Autoencoding idea

- Supervised learning:
  - Univariate regression: predict y from $\boldsymbol{x}$
  - Multivariate regression: predict $\boldsymbol{y}$ from $\boldsymbol{x}$
- Unsupervised learning: explore data $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$
  - No response variable
- For each $\boldsymbol{x}_i$ set $\boldsymbol{y}_i \equiv \boldsymbol{x}_i$
- Train an ANN to predict $\boldsymbol{y}_i$ from $\boldsymbol{x}_i$
- Pointless?

Autoencoder topology

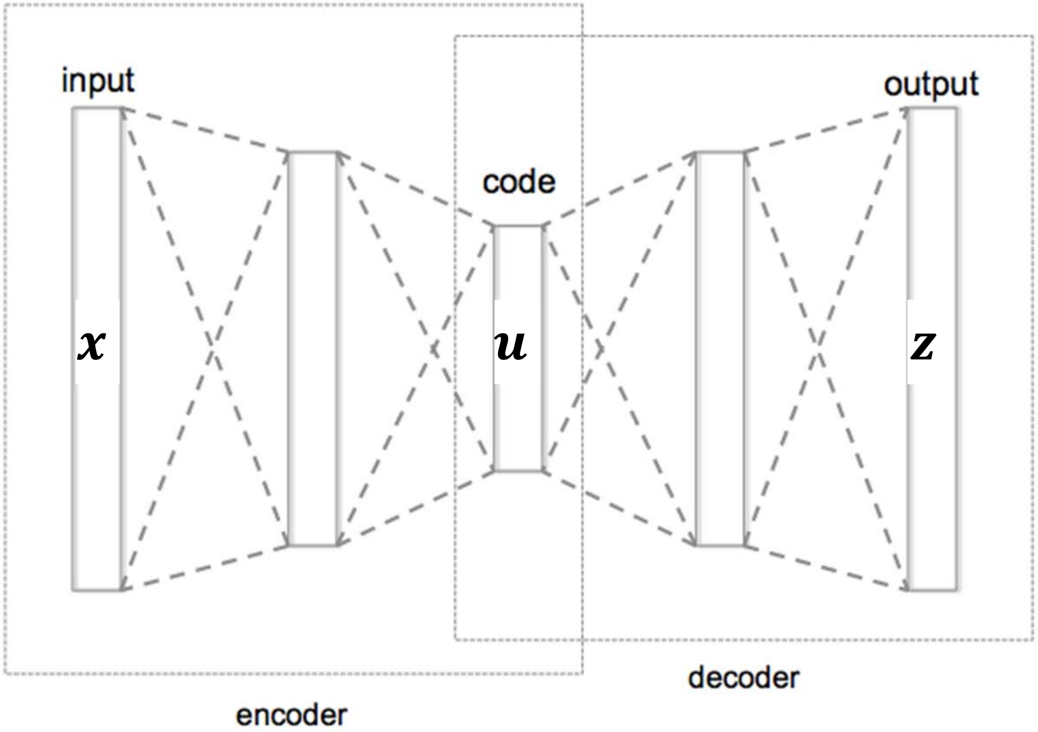
- Given data without labels $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$, set $\boldsymbol{y}_i \equiv \boldsymbol{x}_i$ and train an ANN to predict $\boldsymbol{z}(\boldsymbol{x}_i) \approx \boldsymbol{x}_i$
- Set hidden layer $\boldsymbol{u}$ in the middle "thinner" than input



adapted from: Chervinskii at Wikimedia Commons (CC4)

Introducing the bottleneck

- Suppose you managed to train a network that gives a good restoration of the original signal $\boldsymbol{z}(\boldsymbol{x}_i) \approx \boldsymbol{x}_i$
- This means that the data structure can be effectively described (encoded) by a lower dimensional representation $\boldsymbol{u}$



adapted from: Chervinskii at Wikimedia Commons (CC4)

Dimensionality reduction

- Autoencoders can be used for compression and dimensionality reduction via a non-linear transformation

- If you use linear activation functions and only one hidden layer, then the setup becomes almost that of Principal Component Analysis (stay tuned!)
  - ANN might find a different solution, doesn't use eigenvalues

Tools

- Tensorflow, Theano, Torch
  - python / lua toolkits for deep learning
  - symbolic or automatic differentiation
  - GPU support for fast compilation
  - Theano tutorials at http://deeplearning.net/tutorial/
- Various others
  - Caffe
  - CNTK
  - deeplearning4j …
- Keras: high-level Python API. Can run on top of TensorFlow, CNTK, or Theano

This lecture

- Deep learning
  - Representation capacity
  - Deep models and representation learning
- Convolutional Neural Networks
  - Convolution operator
  - Elements of a convolution-based network
- Autoencoders
  - Learning efficient coding

- Workshops Week #5: Neural nets
- Next lectures: Kernel methods