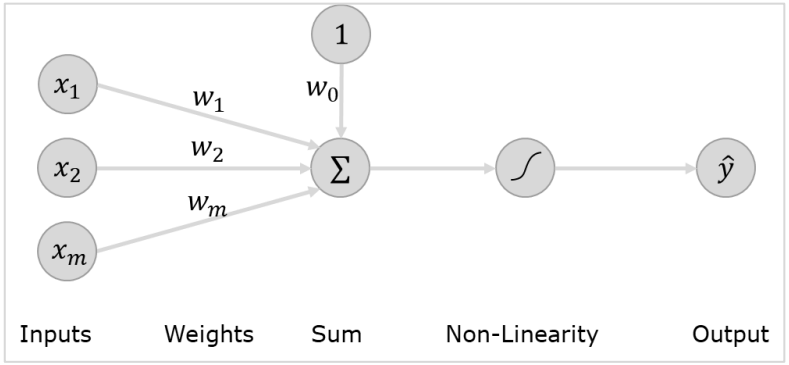
FH KREMS
UNIVERSITY OF APPLIED SCIENCES / AUSTRIA

Machine Learning (WK, 4th Semester)

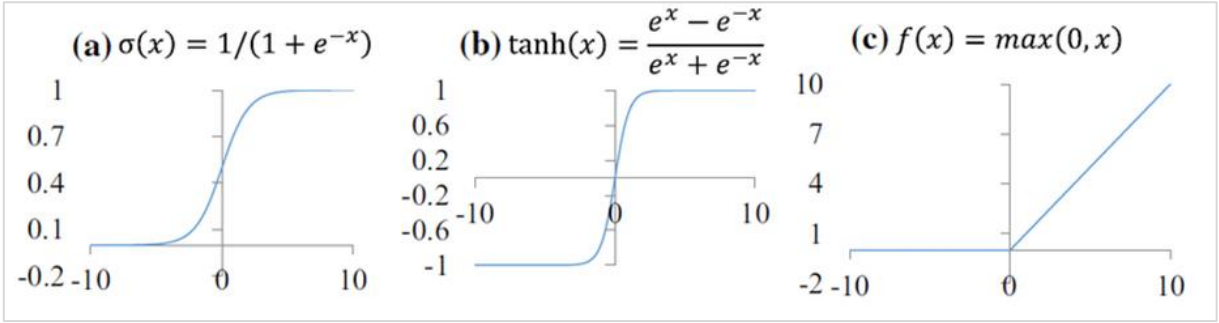
Lecture 11

Summary of previous lecture (1/4)

- The Perceptron



- Activation functions

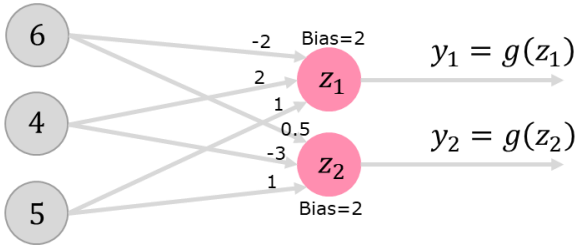


Summary of previous lecture (2/4)

- Computing the output of a perceptron given an input, a matrix of weights, and an activation function

$x_1 = 6$
$x_2 = 4$
$x_3 = 5$

$w_{0,1} = 2$ $\quad w_{0,2} = 2$
$w_{1,1} = -2$ $\quad w_{1,2} = 0.5$
$w_{2,1} = 2$ $\quad w_{2,2} = -3$
$w_{3,1} = 1$ $\quad w_{3,2} = 1$



$z_1 = 2 + (-2 * 6) + (2 * 4) + (1 * 5) = 3$

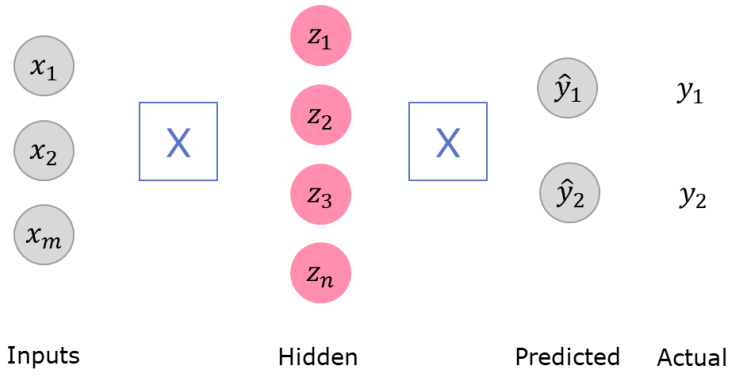
$z_2 = 2 + (0.5 * 6) + (-3 * 4) + (1 * 5) = -2$

$$y_1 = g(z_1) = \begin{cases} \sigma(3) = \sim 0.952 \\ \tanh(3) = \sim 0{,}995 \\ \mathrm{ReLU}(3) = 3 \end{cases}$$

$$y_2 = g(z_2) = \begin{cases} \sigma(-2) = \sim 0.119 \\ \tanh(-2) = \sim -0{,}964 \\ \mathrm{ReLU}(-2) = \sim 0 \end{cases}$$

Summary of previous lecture (3/4)

- Training a network:
  Identifying parameters (biases and weights) such that the output of the neural network is as close as possible to the ground-truth



Mean Squared Error Loss

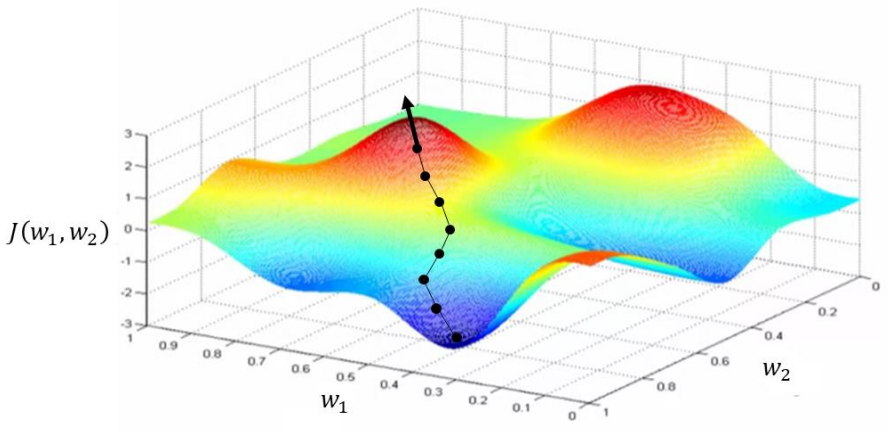
$$L(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y}_i)^2$$

Binary Cross-entropy loss (Log-loss)

$$L(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N} -(y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i))$$

Summary of previous lecture (4/4)

- The gradient descent algorithm



- The learning rate

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

Learning rate

gradient

- Backpropagation
- Drop-out and early-stop

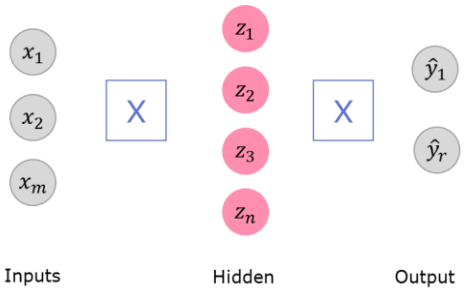
Agenda

- Why Deep Learning?
- CNN
- Transfer learning
- Sequence learning
- Autoencoder

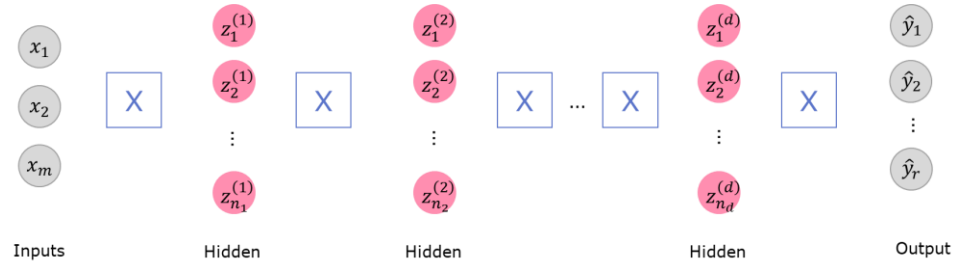
Why going deep?

- Universal approximation theorem:

  A multilayer feed-forward network with as few as one hidden layer can approximate any function to any desired level of accuracy provided that a sufficient number of units is available.



- In many applications, using multiple hidden layers instead of one large layer reduces the number of neurons needed and leads to faster convergence



[1] Goodfellow et al., 2016

Universal approximation theorem

Multilayer Feedforward Networks are Universal Approximators

KURT HORNIK

Technische Universität Wien

MAXWELL STINCHCOMBE AND HALBERT WHITE

University of California, San Diego

(*Received* 16 *September* 1988; *revised and accepted* 9 *March* 1989)

Abstract—*This paper rigorously establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.*

[2] Hornik et al., 1989

Convolutional neural network (CNN)

Motivation

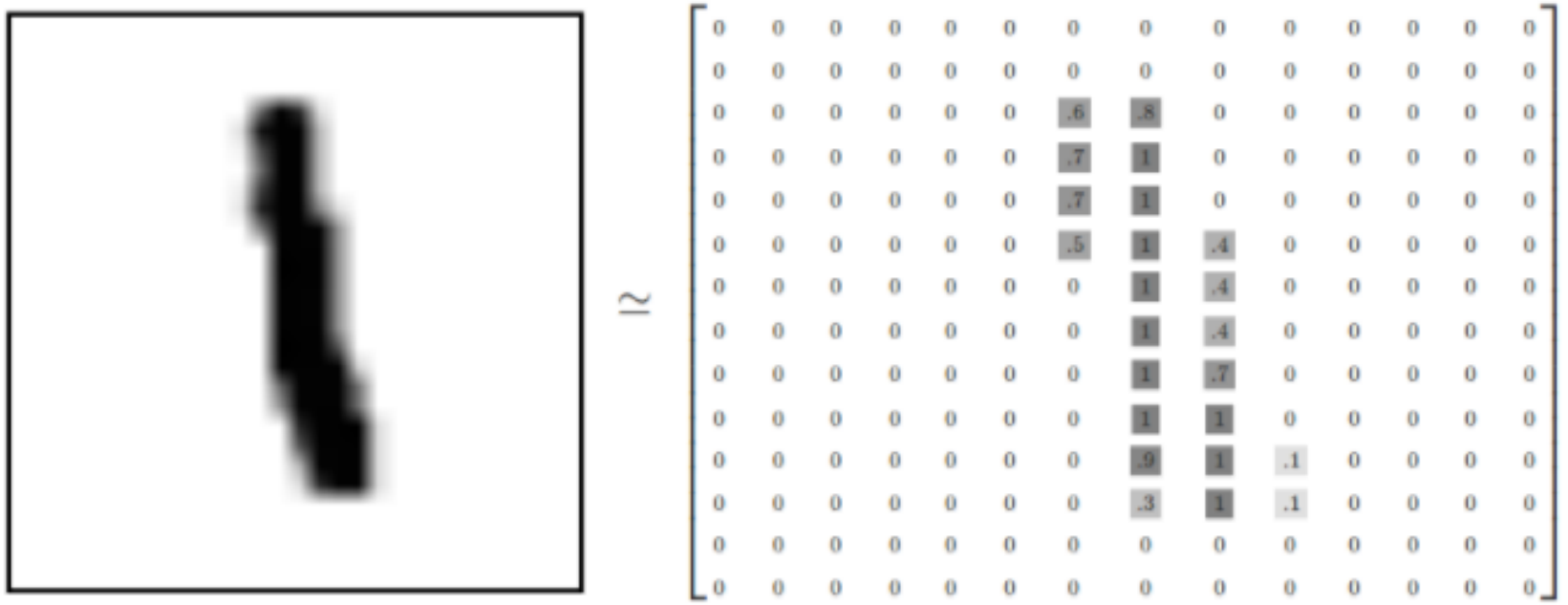
- CNNs proved superior in the field of image classification
- CNNs are specialized on processing data that has a known grid-like topology, e.g.
  - Image data
  - Equidistant time-series
  - Video data

- CNN are particularly well-suited to extract features from an original input

[1] Goodfellow et al., 2016

Convolutional neural network (CNN)

Image data - grayscale

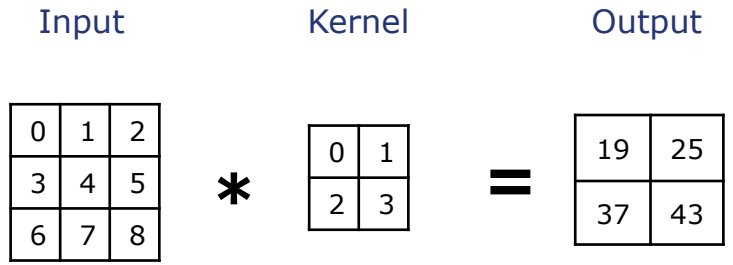


Source: https://www.tutorialexample.com/understand-and-read-tensorflow-mnist-dataset-for-beginners-tensorflow-tutorial/

Convolutional neural network (CNN)

Convolution

- Convolution: Sliding dot product (cross-correlation)



- Each kernel is defined by its width and height

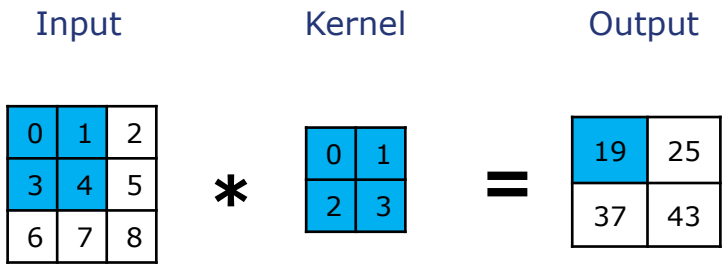
  $n\_parameters = (height \times width + 1) \times n\_kernels$

- These parameters are then subject to the optimization
- → Sparse number of parameters

Convolutional neural network (CNN)

Convolution

- Convolution: Sliding dot product (cross-correlation)



- Each kernel is defined by its width and height

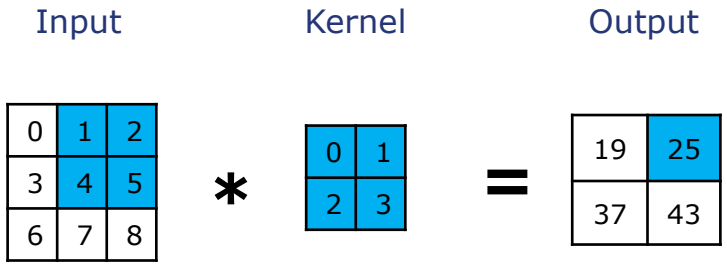
  $n\_parameters = (height \times width + 1) \times n\_kernels$

- These parameters are then subject to the optimization
- → Sparse number of parameters

Convolutional neural network (CNN)

Convolution

- Convolution: Sliding dot product (cross-correlation)



- Each kernel is defined by its width and height

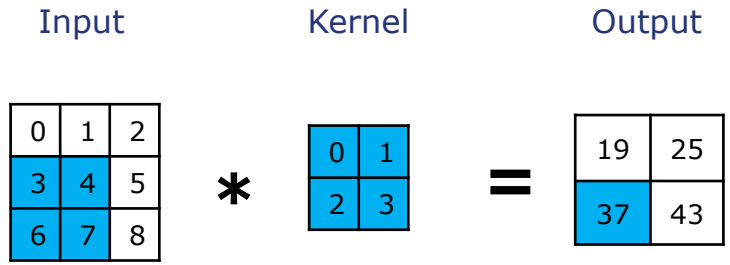
  $n\_parameters = (height \times width + 1) \times n\_kernels$

- These parameters are then subject to the optimization
- → Sparse number of parameters

Convolutional neural network (CNN)

Convolution

- Convolution: Sliding dot product (cross-correlation)



- Each kernel is defined by its width and height

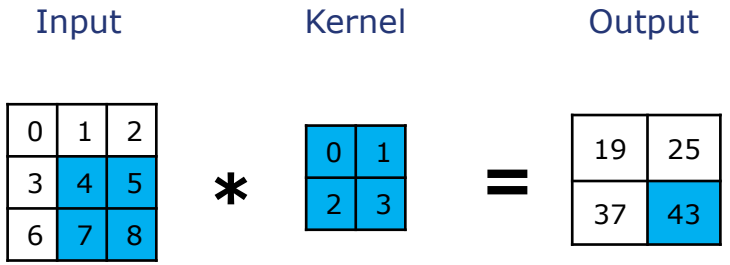
  $n\_parameters = (height \times width + 1) \times n\_kernels$

- These parameters are then subject to the optimization
- → Sparse number of parameters

Convolutional neural network (CNN)

Convolution

- Convolution: Sliding dot product (cross-correlation)



- Each kernel is defined by its width and height

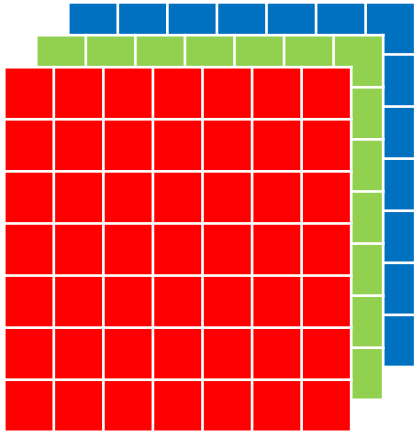
  $n\_parameters = (height \times width + 1) \times n\_kernels$

- These parameters are then subject to the optimization
- → Sparse number of parameters

Convolutional neural network (CNN)

Image data - RGB

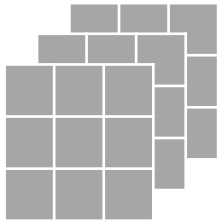
Input



7x7x3

*

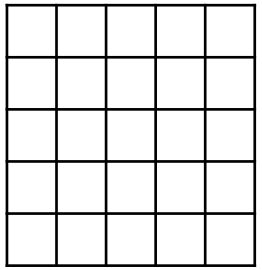
Kernel



3x3x3

=

Output

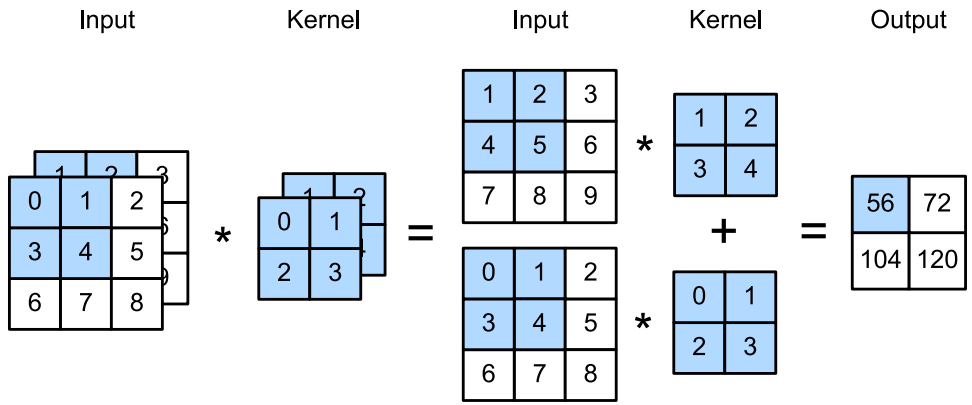


5x5

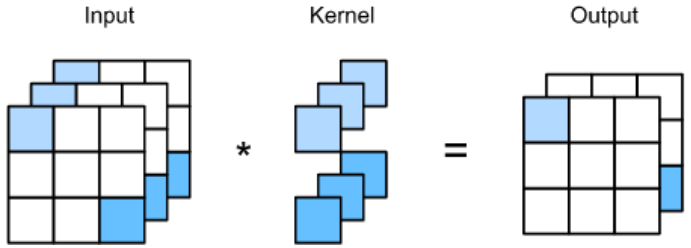
Convolutional neural network (CNN)

Multi-input channel convolution

- Each channel needs a corresponding 2D kernel
- The cross-correlation is calculated by summing over the two-dimensional cross-correlations of each channel



- For multiple output channels:
  - Multi-input channel convolution is applied several times
    --> hyper-parameter: out_channels
  - See *Conv2D()*



[4] Zhang et al., (2021)

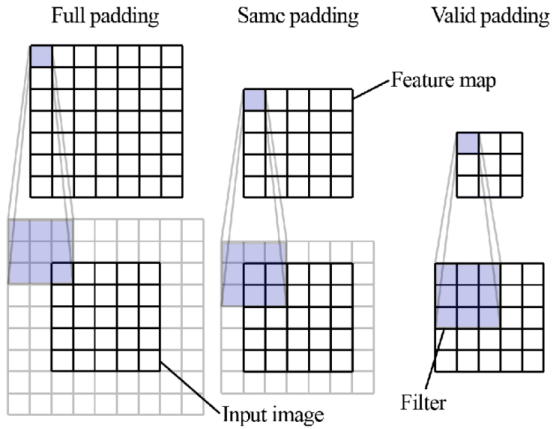
Convolutional neural network (CNN)

Kernel hyper-parameters

- **Kernel size**:
  - single integer for same height and width
  - Tuple: (height, width)

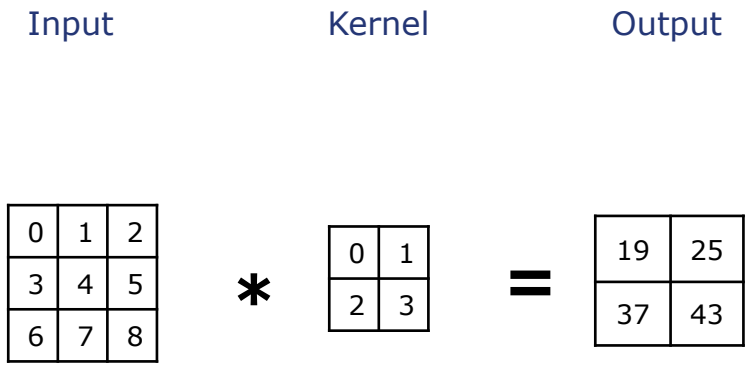
- **Padding**:
  - Number of pixels to add to border of the image
    - In pytorch and mxnet: single or tuple integer
    - In Keras: “full”, “same”, or “valid”

- **Stride**:
  - Number of pixel we skip when moving the convolutional window
  - Tuple: (x-axis, y-axis)



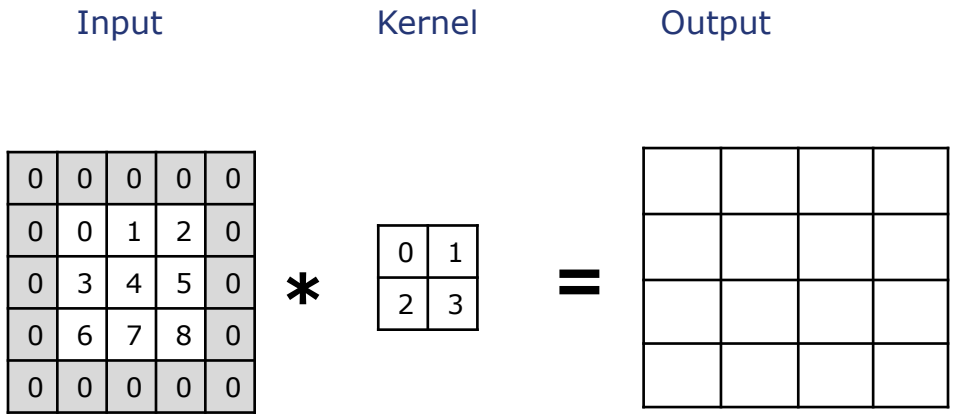
[5] Nowell, Tyrone. (2019)

Example



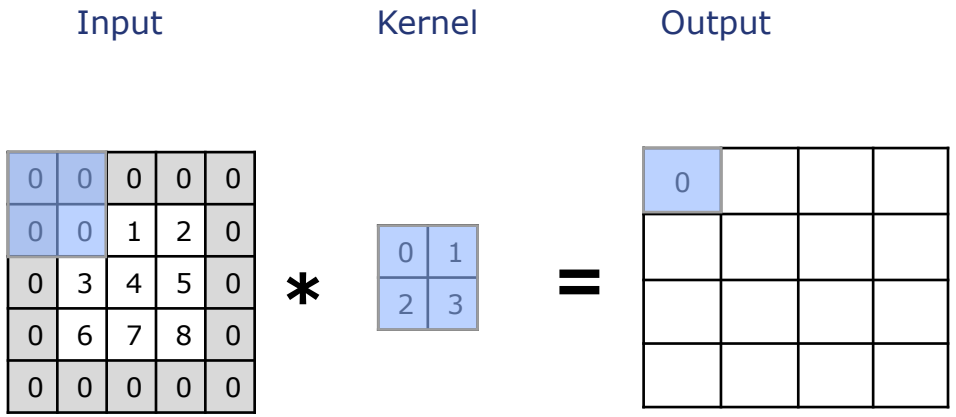
Example

1. Padding = 1



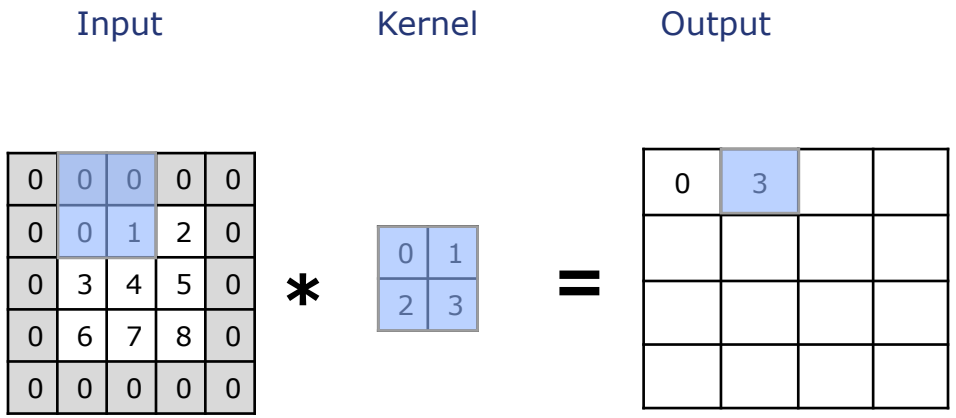
Example

1. Padding = 1



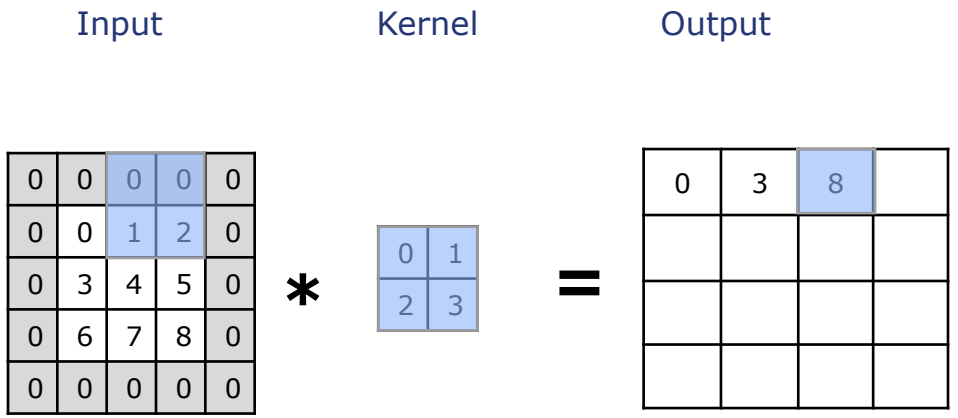
Example

1. Padding = 1



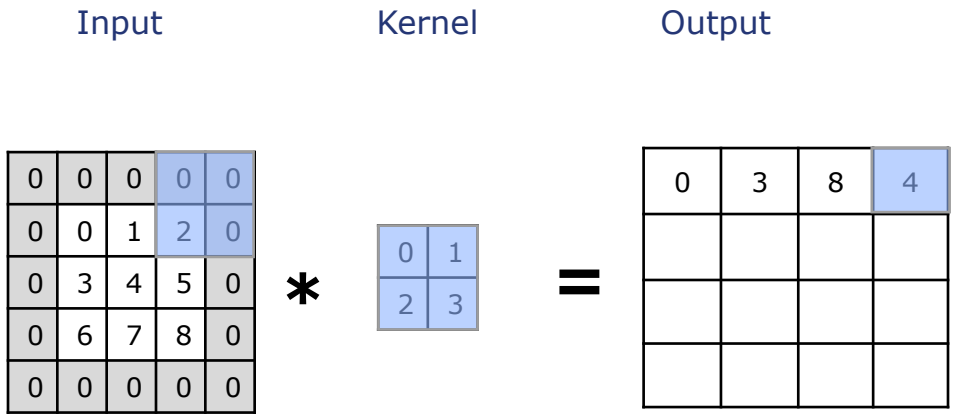
Example

1. Padding = 1



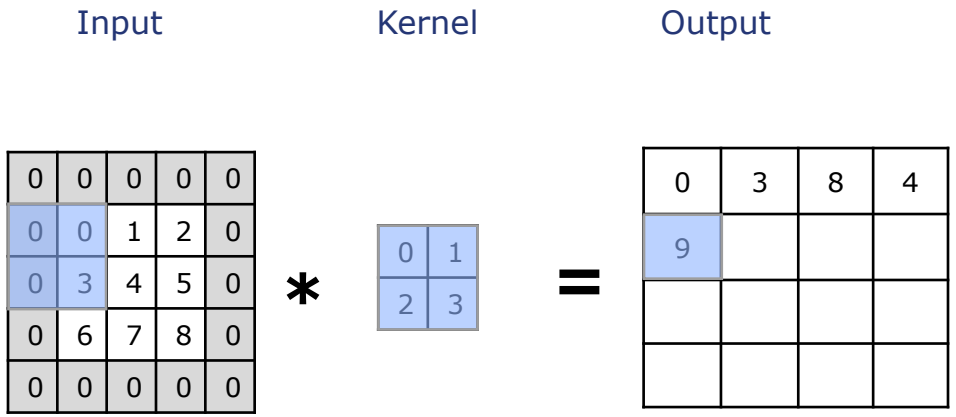
Example

1. Padding = 1



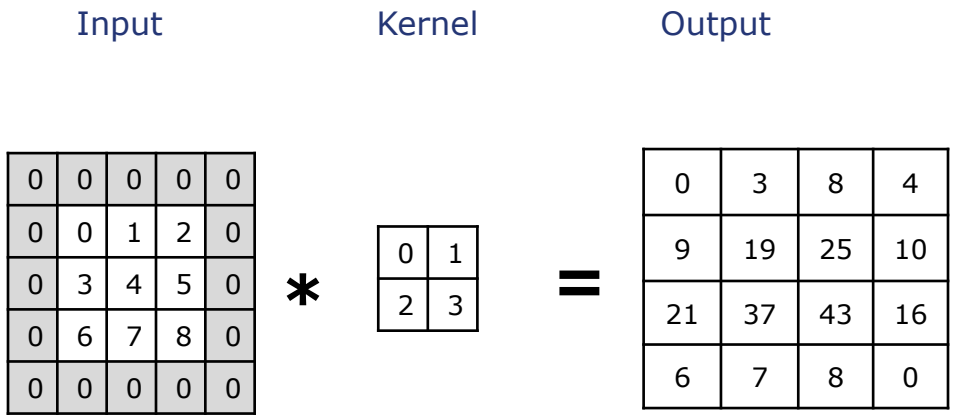
Example

1. Padding = 1



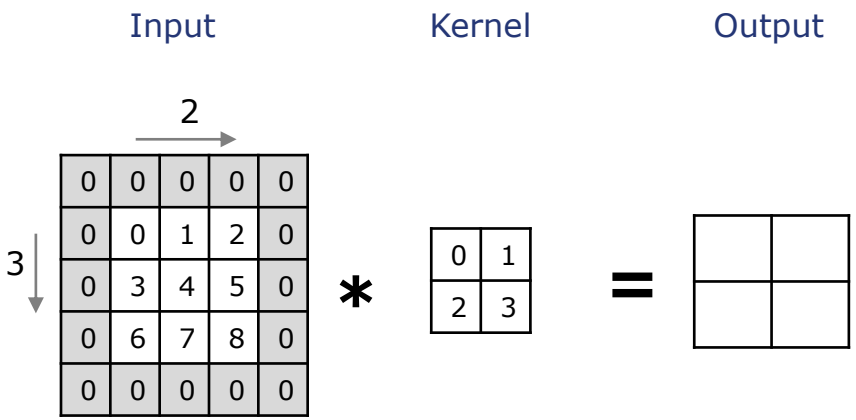
Example

1. Padding = 1



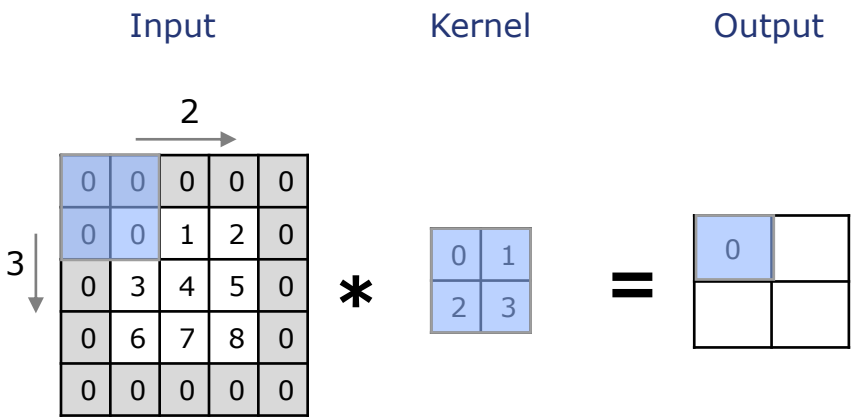
Example

1. Padding = 1
2. Stride
   1. Width=2
   2. Height=3



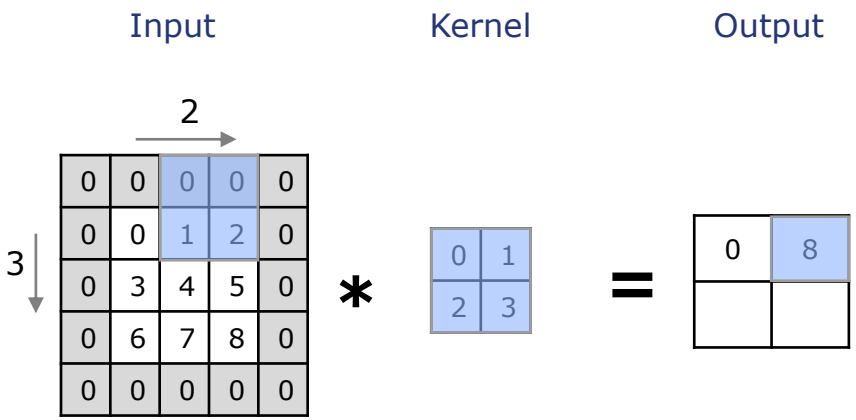
Example

1. Padding = 1
2. Stride
   1. Width=2
   2. Height=3



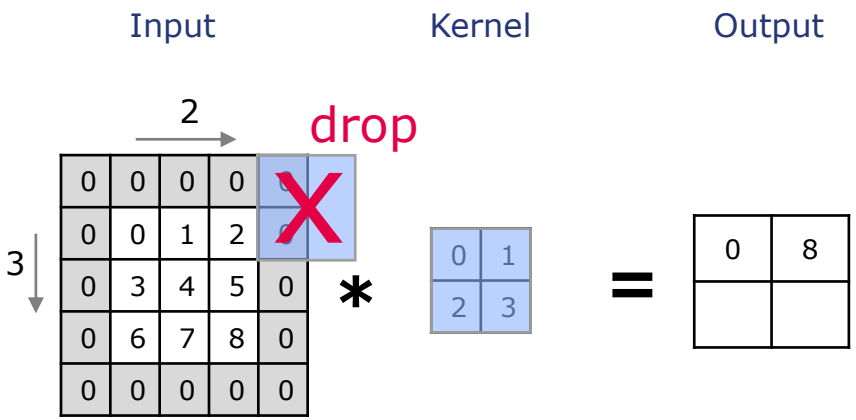
Example

1. Padding = 1
2. Stride
   1. Width=2
   2. Height=3



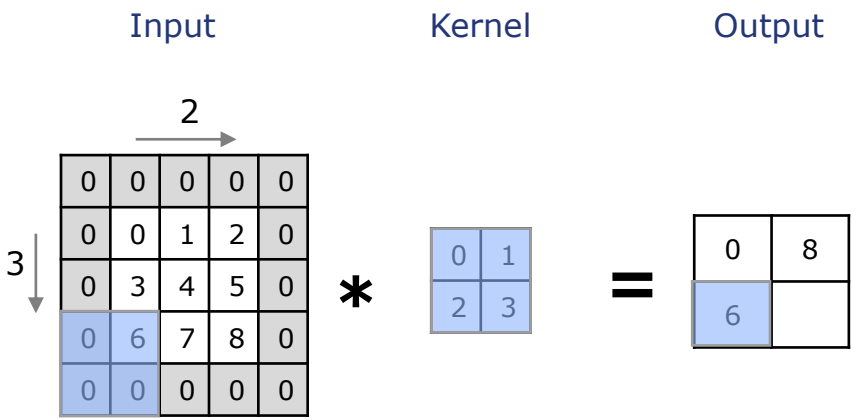
Example

1. Padding = 1
2. Stride
   1. Width=2
   2. Height=3



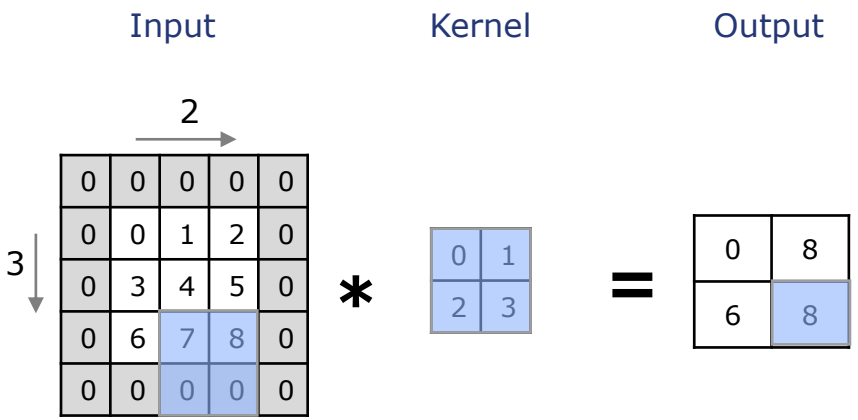
Example

1. Padding = 1
2. Stride
   1. Width=2
   2. Height=3



Example

1. Padding = 1
2. Stride
   1. Width=2
   2. Height=3

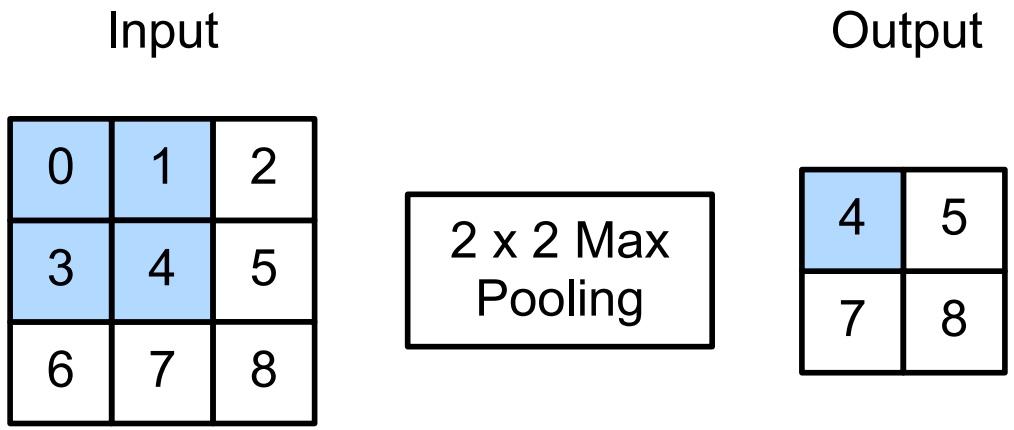


Convolutional neural network (CNN)

Pooling

- Pooling is used to further reduce the number of parameters

- Common pooling operations
  - Max
  - Min
  - Average

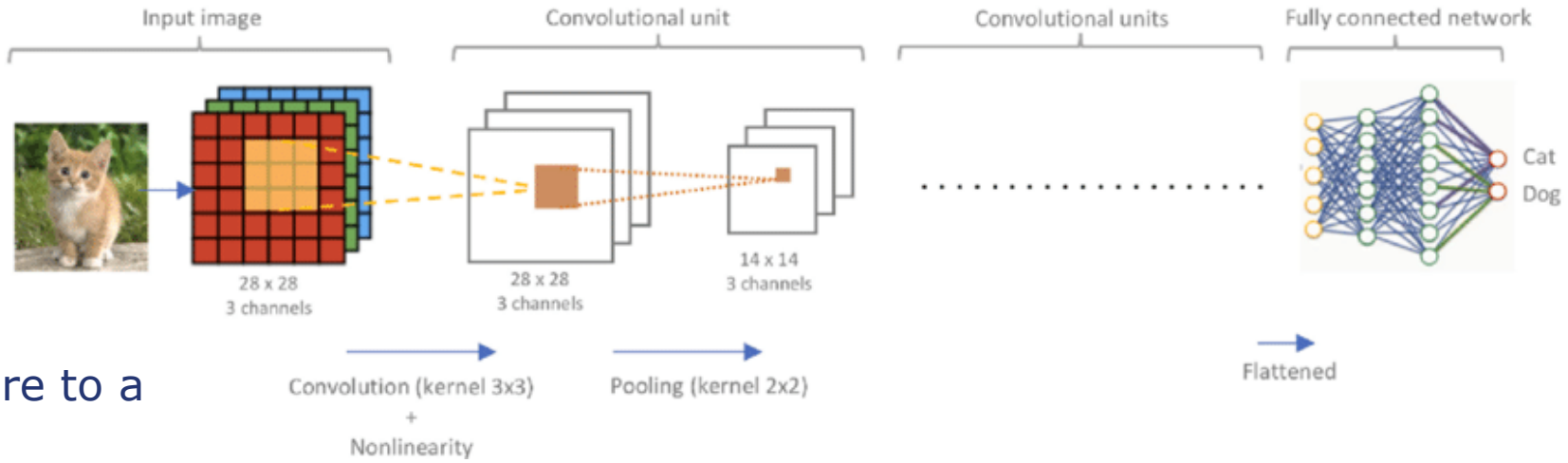


[4] Zhang et al., (2021)

Convolutional neural network (CNN)

Convolutional layers example

- Convolutional layer usually consists of:
  - Convolution layer
  - Activation function
  - Pooling layer
- Flattening
  - Transform pooled feature to a one-dimensional array



→ feed forward network

[7] Shrestha et al. (2020)

Convolutional neural network (CNN)

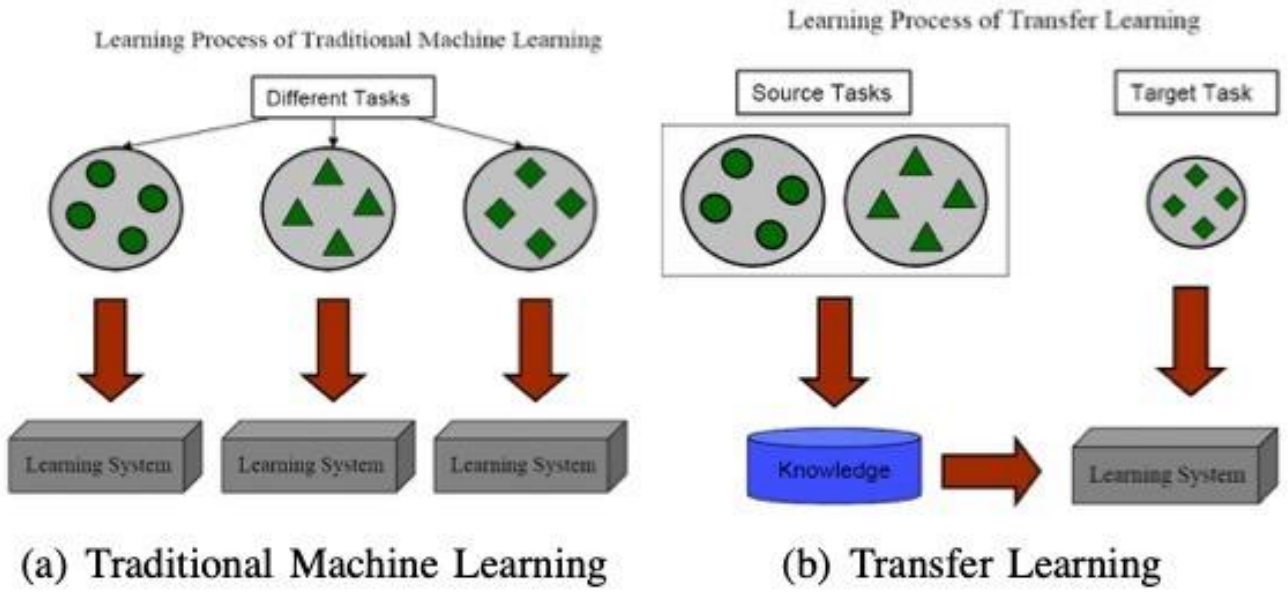
Conclusion

- Pros
  - Leading method in the field of computer vision
  - No human supervision needed
  - Less parameters needed compared to deep feedforward neural networks
- Cons
  - Flattening → loose information about the position of the object
  - Needs a large amount of training data

Transfer learning

Motivation

- Apply knowledge obtained from already solved problems on new similar problems
- Less training required
- Less data needed

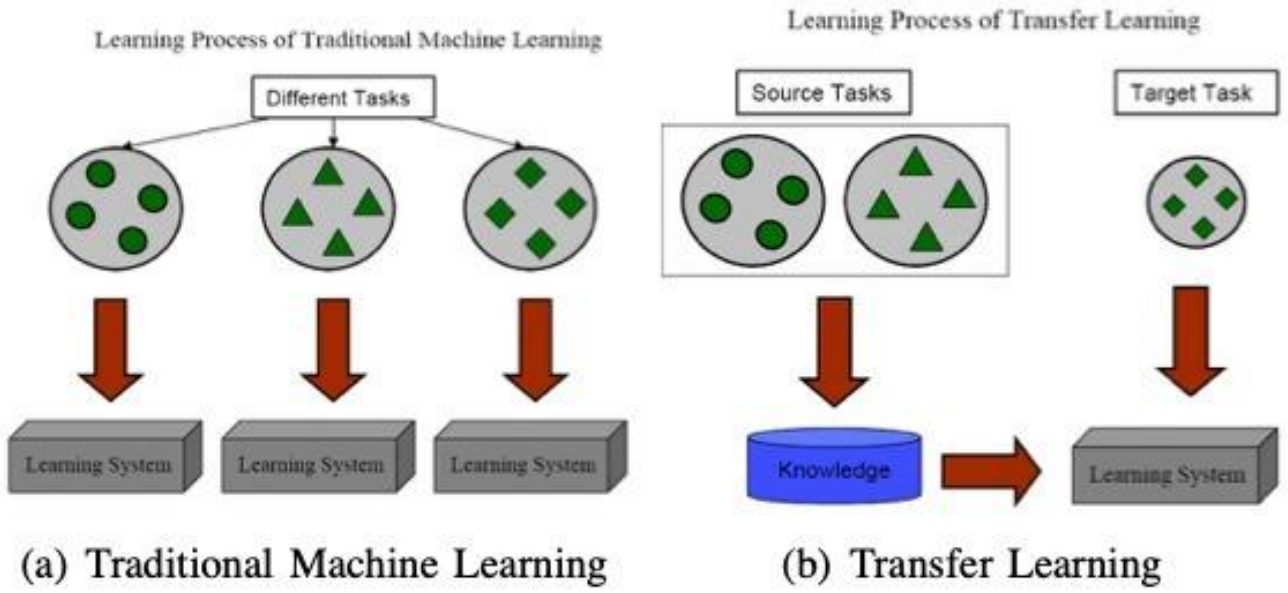


[8] Pan and Yang, (2009)

Transfer learning

Example CNN

- **Finetuning CNN**
  - Instead of randomly initializing the parameters we could use pre-trained parameters
  - Train these parameters to better fit the new problem
- **CNN as feature extractor**
  - Assume convolutional layers extracted helpful features for previous problems
  - Fix pre-trained parameters in the convolutional layers and train a feedforward network on the flattened output



[8] Pan and Yang, (2009)

Transfer learning

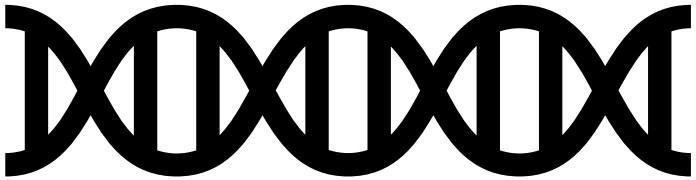
- Pros
  - Handle small sample sizes
  - Faster convergence
  - Take advantage of models trained on large data sets
- Cons
  - Model size increases drastically
  - Poor performance if initial and target problem are very dissimilar
  - Constrains in terms of architecture

Well known pre trained models:

- Pytorch: https://pytorch.org/vision/stable/models.html
- TensorFlowHub: https://tfhub.dev/

Sequence learning

- ML for sequential data e.g.:
  - Audio signals
  - Video classification
  - NLP - text classification
  - DNA/Gene classification



- Classical deep feedforward networks and CNNs
  - need a fixed input and output size
  - ignore temporal dependency
  - assume independent observations

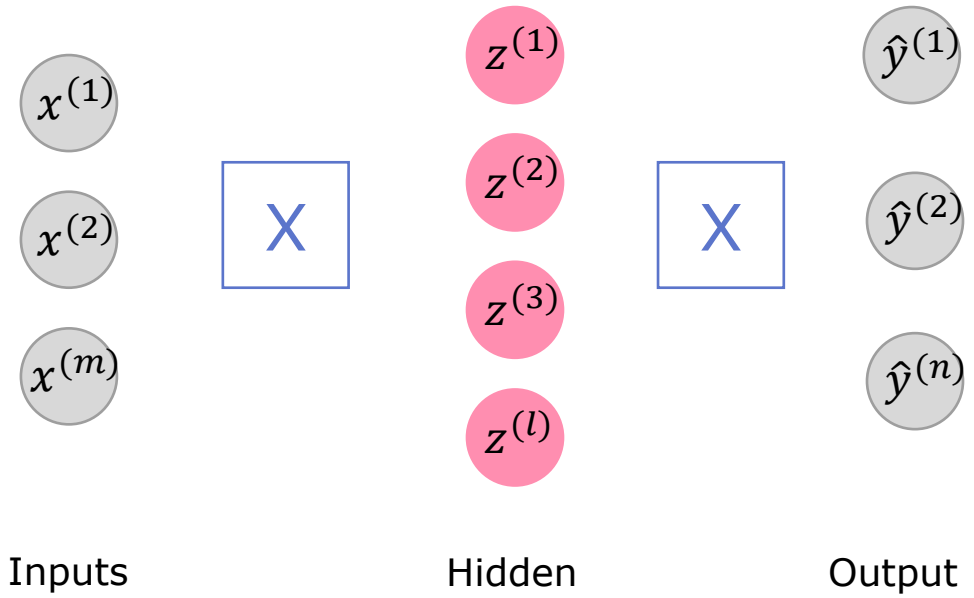
Recurrent neural networks (RNN)

Motivation

- Class of artificial neural networks that exhibit a temporal dynamic behaviour
- Applications:
  - Time series data
  - NLP – text generator
  - Speech recognition
  - Stock price prediction

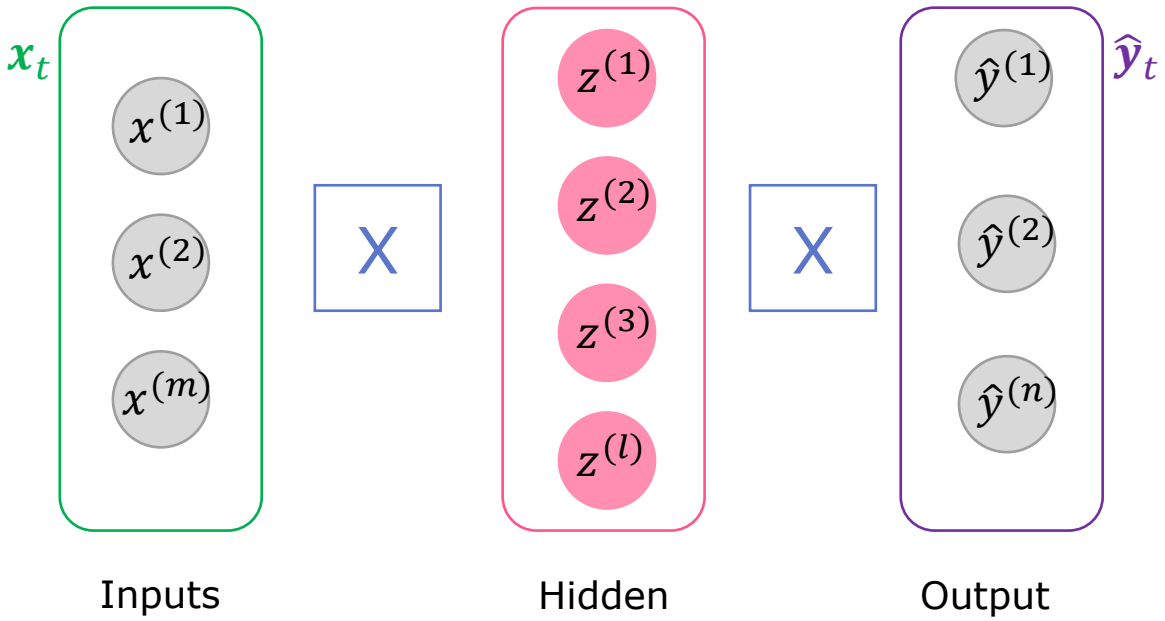
Recurrent neural networks (RNN)

Revisiting the perceptron



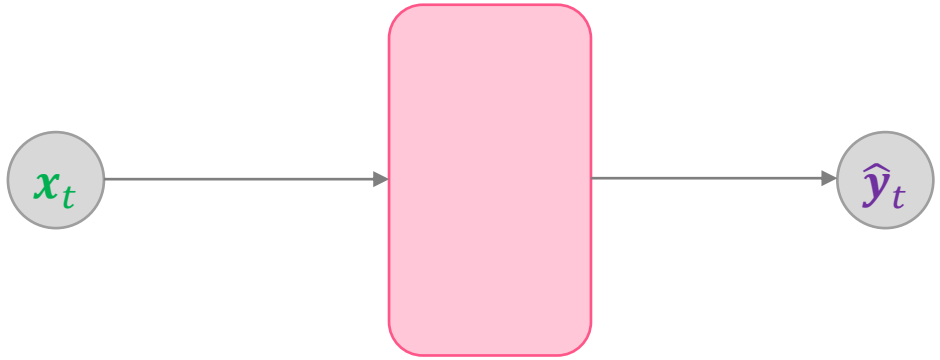
Recurrent neural networks (RNN)

Revisiting the perceptron



Recurrent neural networks (RNN)

Revisiting the perceptron

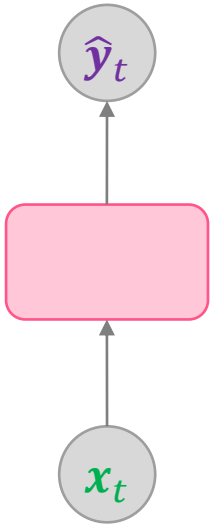


Inputs

Output

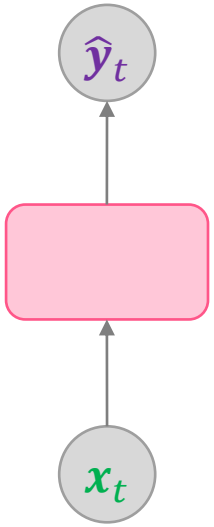
Recurrent neural networks (RNN)

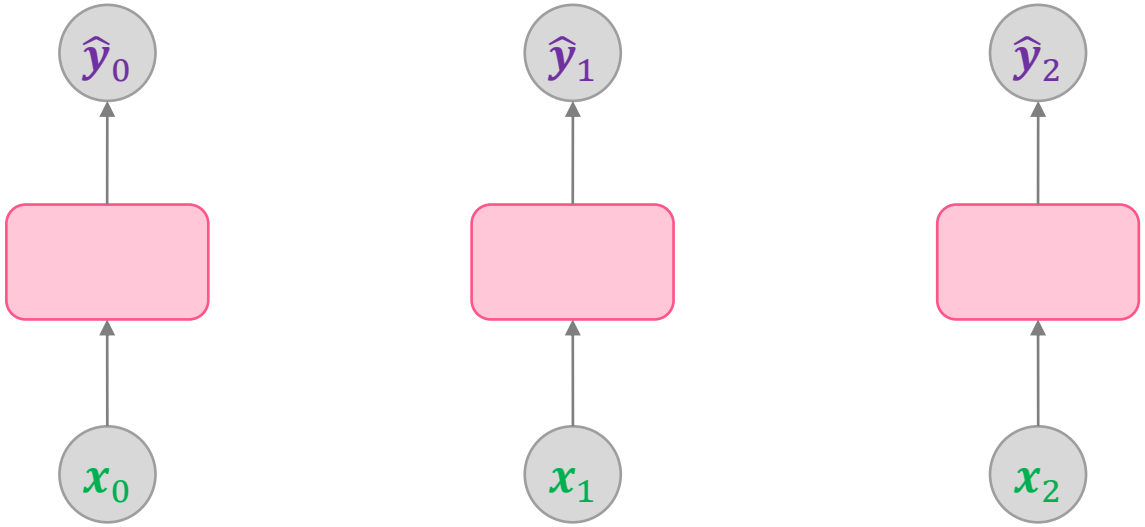
Revisiting the perceptron



Recurrent neural networks (RNN)

Revisiting the perceptron



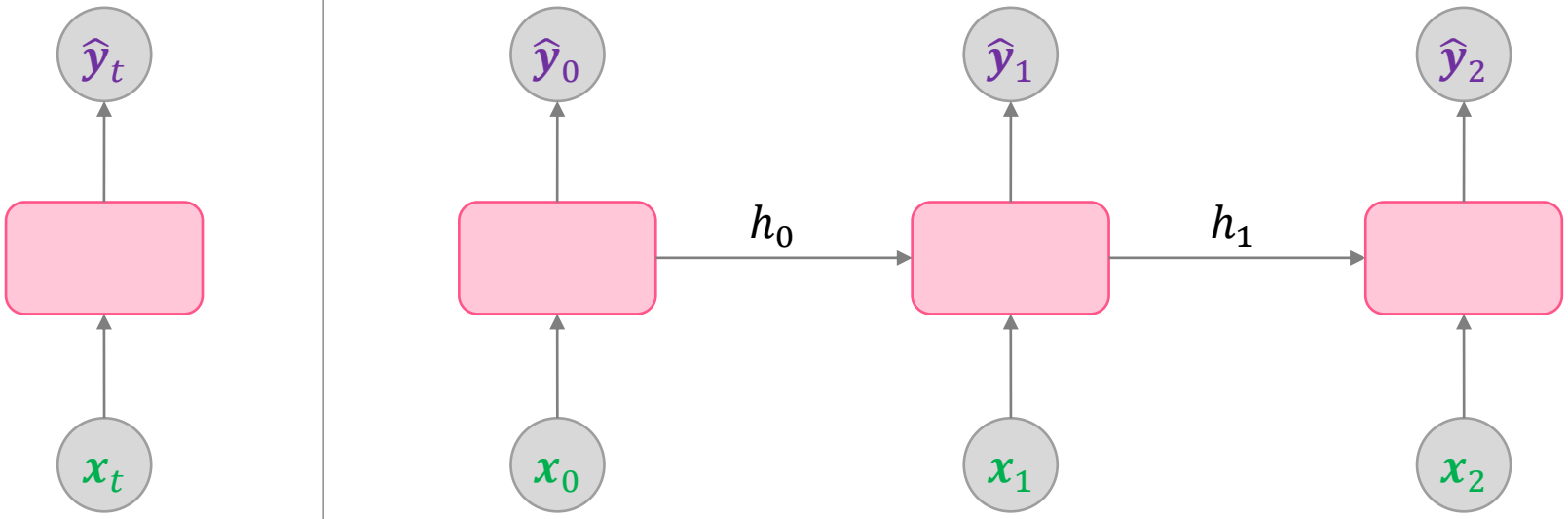


$$\hat{y}_t = f(x_t)$$

There is no dependency between observation. We cannot capture sequence properties

Recurrent neural networks (RNN)

Revisiting the perceptron

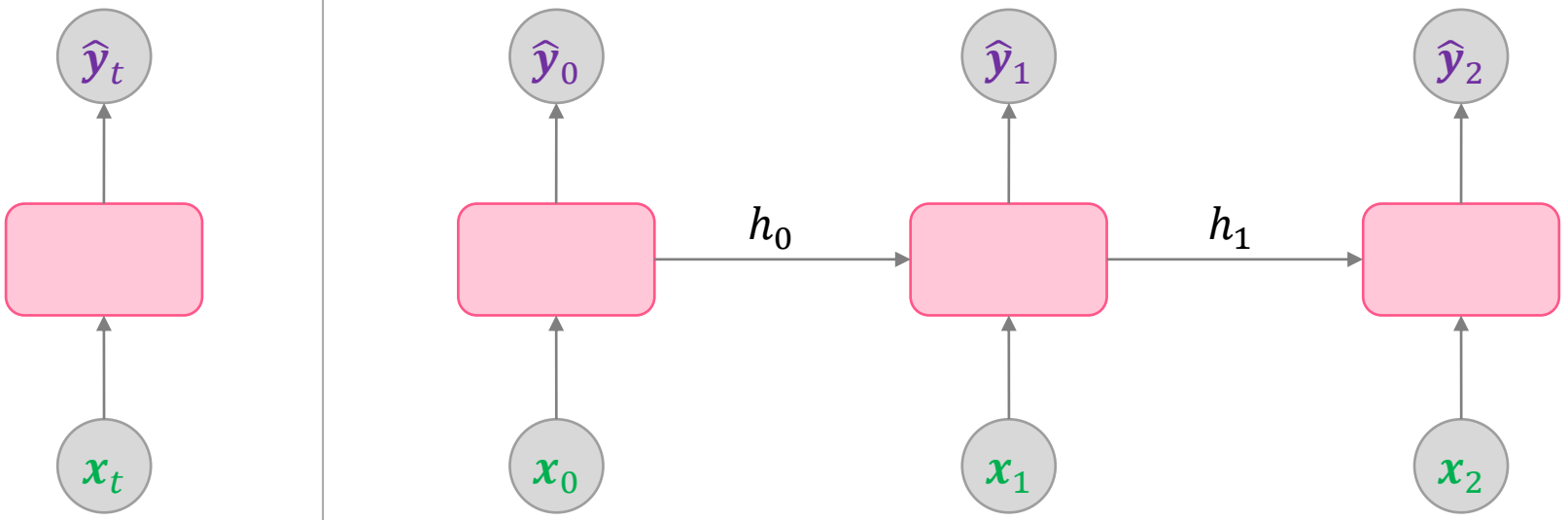


$$\hat{y}_t = f(x_t)$$

There is no dependency between observation. We cannot capture sequence properties

Recurrent neural networks (RNN)

Revisiting the perceptron

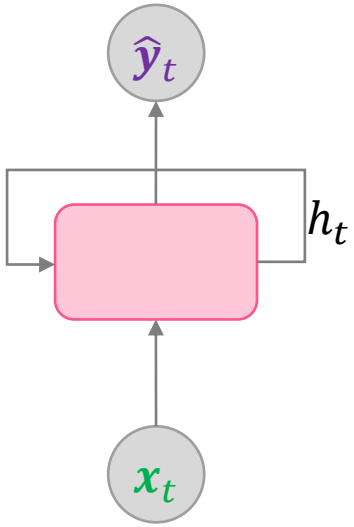


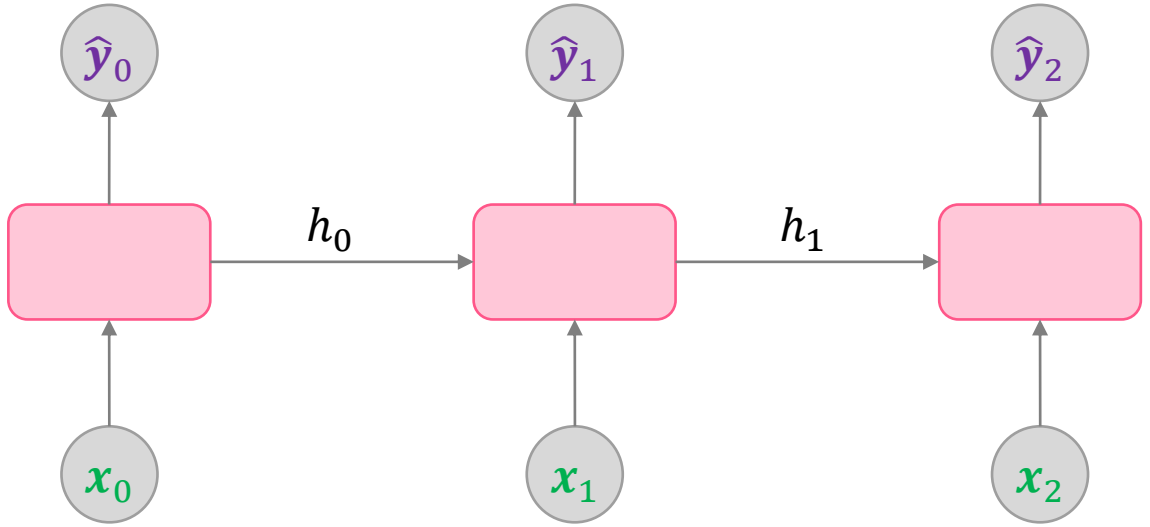
$$\hat{\boldsymbol{y}}_t = f(\boldsymbol{x}_t, h_{t-1})$$

output input past

Recurrent neural networks (RNN)

Revisiting the perceptron



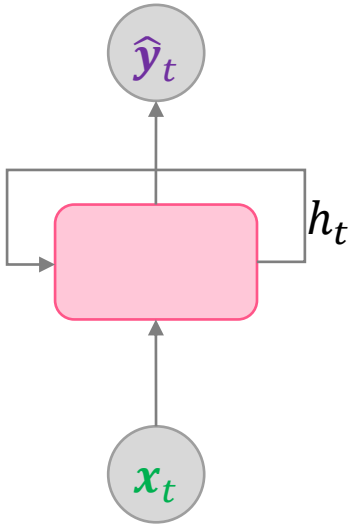


$$\hat{y}_t = f(x_t, h_{t-1})$$

output input past

Recurrent neural networks (RNN)

Revisiting the perceptron



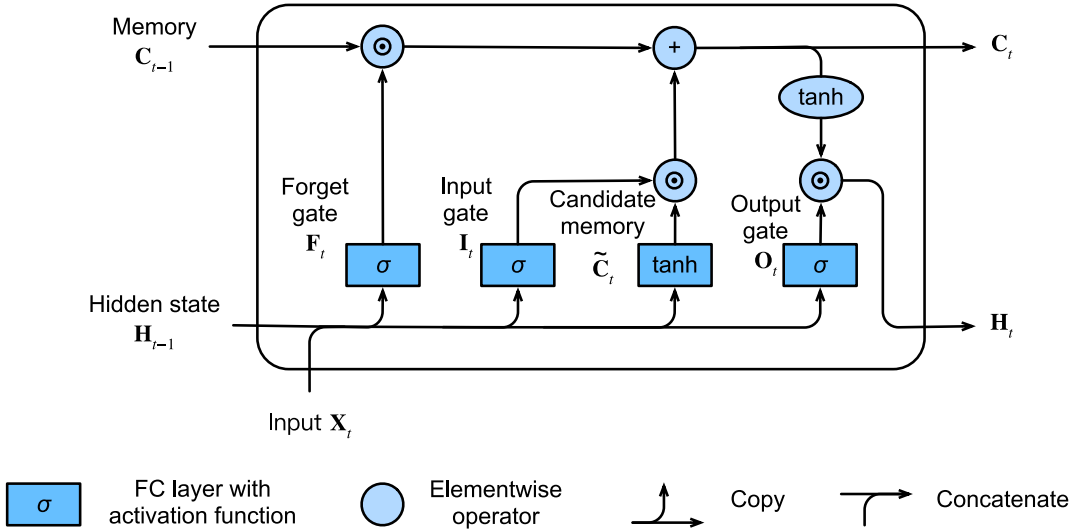
$h_t = f_W(x_t,\ h_{t-1})$ Applies a recurrence relation at every time step

- Problem:
  - Backpropagation through time: Vanishing/exploding gradient
- But we need to handle
  - short-term memory
  - long-term memory
- Example: Sentence completion
- Introducing the concept of gate

Recurrent neural networks (RNN)

Long Short-Term Memory (LSTM)

- 3 types of gates:
  - input gates
  - forget gates
  - output gates
- The hidden layer output of LSTM includes the hidden state and the memory cell. Only the hidden state is passed into the output layer. The memory cell is entirely internal.
- LSTM is an approach to avoid vanishing and exploding gradients



[4] Zhang et al., (2021)

Recurrent neural networks (RNN)

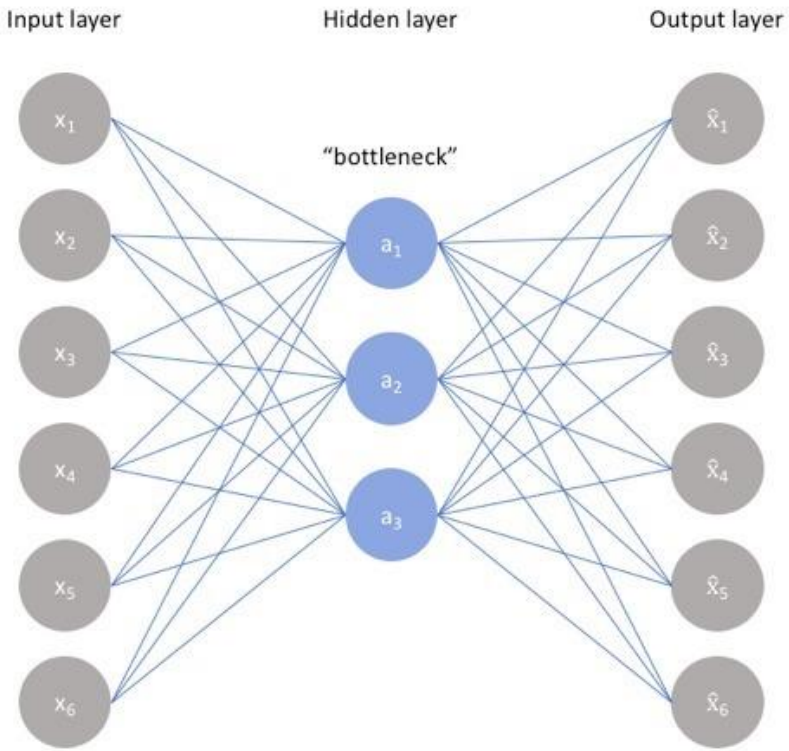
LSTM

- Pros
  - No fixed input/output size needed
  - Memories previous observations
  - Can be used to model time-series/sequence data
  - Models long AND short time behavior
- Cons
  - Slow in training
  - Easy overfitting
  - Sensitive to parameter initializations

Autoencoder

Motivation

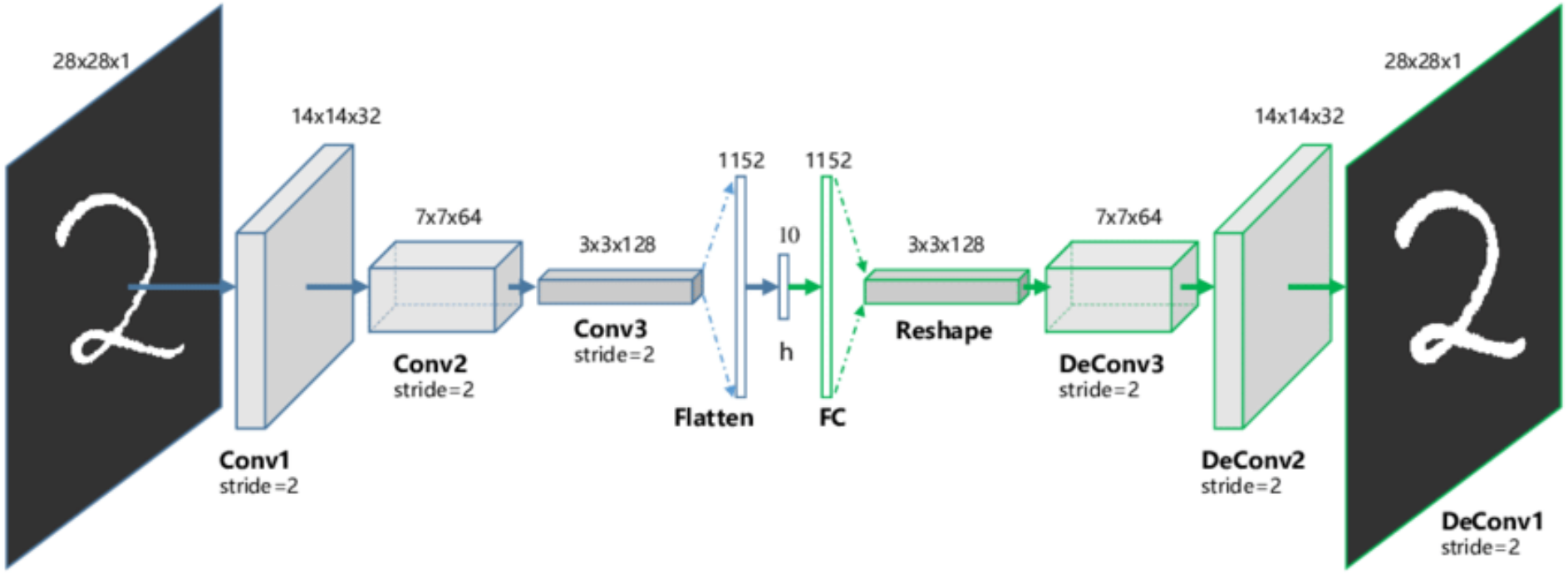
- Unsupervised learning approach
- Predict x given observation x
- Learns a lower dimensional representation of higher dimensional data
- Applications:
  - Information retrieval
  - Dimensionality reduction
  - Anomaly detection



Source: https://www.jeremyjordan.me/autoencoders/

Autoencoder

Example



Source: https://analyticsindiamag.com/how-to-implement-convolutional-autoencoder-in-pytorch-with-cuda/

References

- [1] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.
- [2] Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, *2*(5), 359-366.
- [3] Hanifah, Anis & Gunawan, Teddy & Kartiwi, Mira. (2018). Speech Emotion Recognition Using Deep Feedforward Neural Network. Indonesian Journal of Electrical Engineering and Computer Science. 10. 554-561. 10.11591/ijeecs.v10.i2.pp554-561.
- [4] Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into deep learning. *arXiv preprint arXiv:2106.11342*.
- [5] Nowell, Tyrone. (2019). Detection and Quantification of Rot in Harvested Trees using Convolutional Neural Networks. 10.13140/RG.2.2.28653.74722.

References

- [6] Hristov, Anton & Nisheva-Pavlova, Maria & Dimov, Dimo. (2019). Filters in Convolutional Neural Networks as Independent Detectors of Visual Concepts. 110-117. 10.1145/3345252.3345294.
- [7] Shrestha, Yash & Krishna, Vaibhav & Krogh, Georg. (2020). Augmenting Organizational Decision-Making with Deep Learning Algorithms: Principles, Promises, and Challenges (Forthcoming at Journal of Business Research). Journal of Business Research.
- [8] Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, *22*(10), 1345-1359.

If you want to learn more

- Excellent book (open and available online): http://d2l.ai/index.html
- Excellent course: http://introtodeeplearning.com/ (MIT – Videos on youtube)