Neural Networks 2 - Sequences and RNNs

18NES2 - Week 9, Winter semester 2025/26

Zuzana Petříčková

November 25, 2025

What We Covered Last Time

Practical examples of CNN design patterns

- residual connections, bottleneck blocks, depthwise separable convolutions
- training a small Xception-like model from scratch

Applications of CNNs

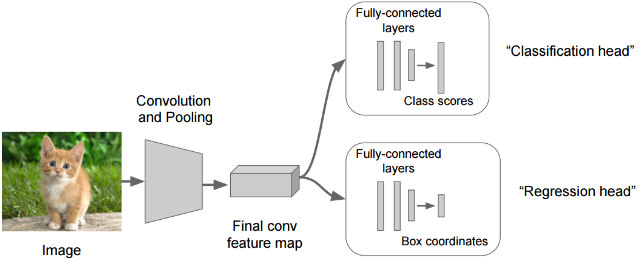
- Semantic segmentation + encoder-decoder architecture + practical example
- Object detection

This Week

1. Applications of Convolutional Neural Networks
   - Object Detection
   - Instance segmentation
   - Autoencoders
   - Other Applications
2. Sequential data
   - Tasks
   - Time Series Forecating
3. Recurrent Neural Network (RNN)
   - Vanilla RNN
4. Graded Homework

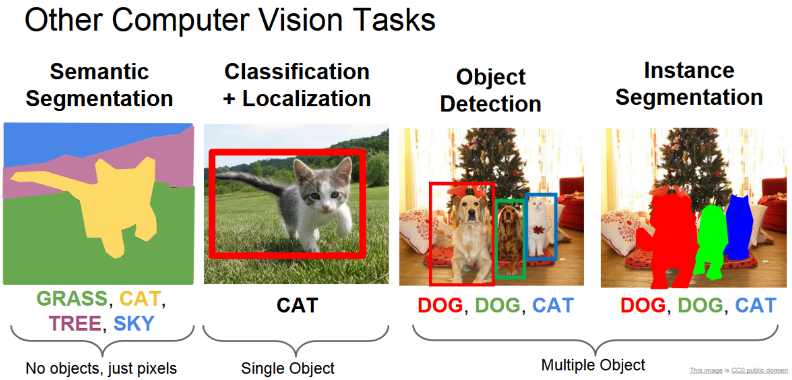
Applications of Convolutional Neural Networks

- The classification head of a neural network can be replaced by a different head to solve a different task on the same (or similar) data
  - Different classification tasks — classification head
  - Regression tasks — regression head
  - Semantic segmentation — encoder-decoder architeture
  - ...



Source: https://i.stack.imgur.com/FGrD1.png

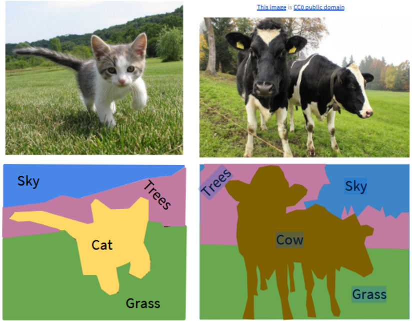
Applications of Convolutional Neural Networks



Source: `https://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf`

Encoder–Decoder for Semantic Segmentation

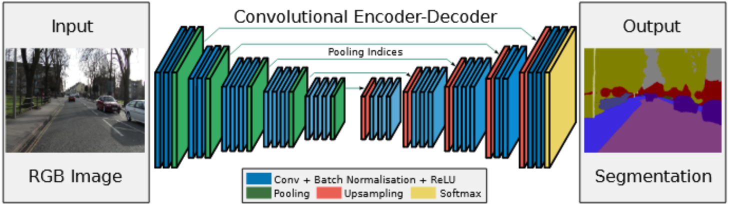
- Goal: assign a **class label to each pixel** in the image.
- Each pixel belongs to one of the predefined object categories.
- Typical architecture: **encoder–decoder CNN** (e.g., SegNet, U-Net).



Source: `https://cs231n.stanford.edu/slides/2024/lecture_9.pdf`

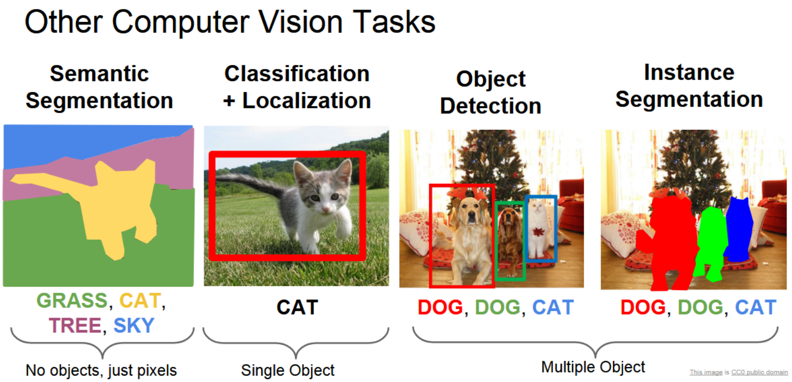
Encoder–Decoder for Semantic Segmentation

- **Encoder:** classic CNN that extracts multi-scale features (**downsampling using greater stride rather than poolng**).
- **Latent space:** compact feature representation.
- **Decoder:** reconstructs spatial resolution using **upsampling:**
  - **Transposed convolutions (a.k.a. deconvolutions)**
  - **Unpooling** (less common in segmentation)



Source: SegNet — A Deep Convolutional Encoder–Decoder Architecture for Image Segmentation `https://arxiv.org/pdf/1511.00561`

Applications of Convolutional Neural Networks

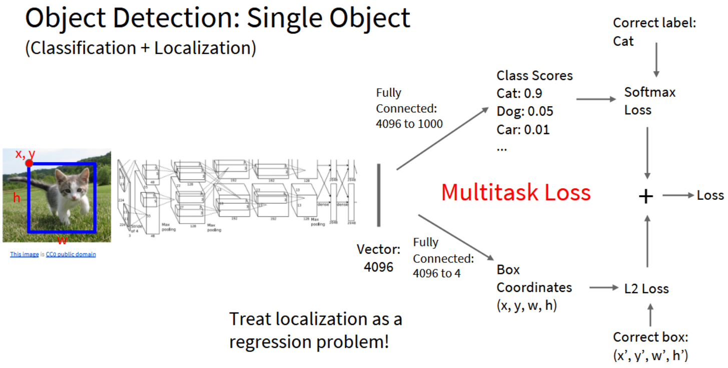


Source: `https://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf`

CNNs and Object Detection

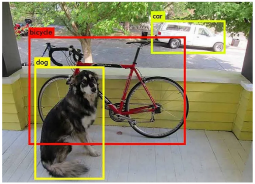
Single object – two heads:

- **Classification head** – predicts the object class.
- **Regression head** – predicts the bounding box coordinates.

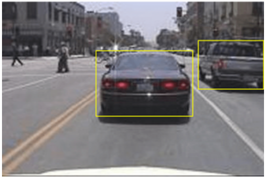


Source: `https://cs231n.stanford.edu/slides/2024/lecture_9.pdf`

CNNs and Multi-Object Detection



Source: Blog by Matthijs Hollemans



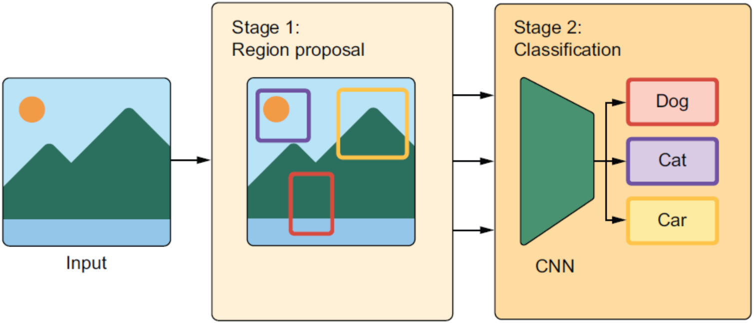
Sources: `https://matlabacademy.mathworks.com`

Triantafyllidou, D. et al.: *A Fast Deep Convolutional Neural Network for Face Detection in Big Visual Data.*

R-CNN: Region-based Convolutional Neural Network

Two-stage detector

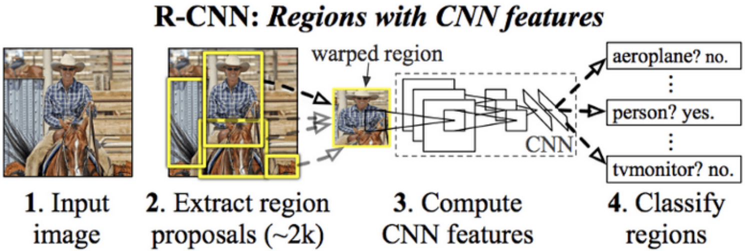
1. Extract candidate regions (ROIs, regions of interest)
2. Run classification on each region



Source: F. Chollet, "Deep Learning with Python," Figure 12.2

R-CNN: Region-based Convolutional Neural Network

- The input image is first divided into candidate regions (ROIs) using **selective search** heuristics.
- Each region is resized to a fixed size and passed through a pretrained CNN (e.g., VGG-16 trained on ImageNet in the original paper).
- Two heads are used:
  - a classifier (originally SVM) to classify each region,
  - a regressor to refine the bounding box.



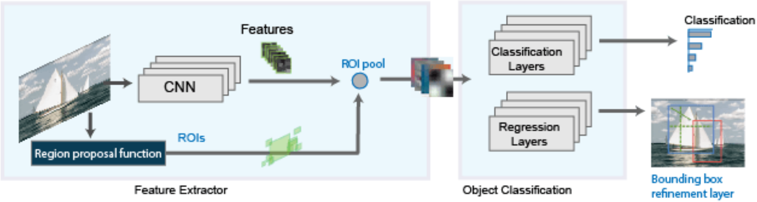
Fast R-CNN and Faster R-CNN

The original R-CNN

- Very computationally expensive (many independent forward passes for each image)

Fast R-CNN

- Speeds up computation by **sharing convolutional features** between all regions of interest (ROIs).
- The whole image is passed through a CNN base once; ROIs are pooled from the feature map.

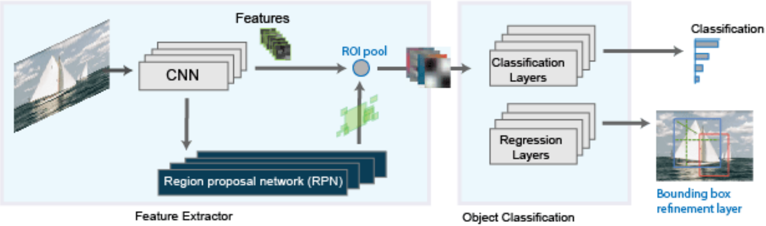


Source: *https://www.mathworks.com/help/vision/ug/getting-started-with-r-cnn-fast-*

Fast R-CNN and Faster R-CNN

Faster R-CNN

- Introduces a **Region Proposal Network (RPN)** that learns to propose ROIs directly inside the model.
- Replaces the slow selective search step.



Source: *https://www.mathworks.com/help/vision/ug/getting-started-with-r-cnn-fast-r-cnn-and-faster-r-cnn.html*

One-Stage Detectors: YOLO, RetinaNET, after 2015

Key idea:

- Predict **bounding boxes and classes in a single forward pass**, without a separate region proposal stage.

YOLO (You Only Look Once)

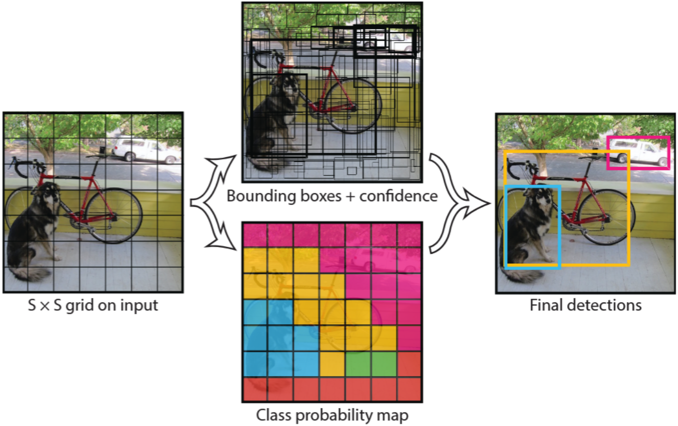
- Divides the image into a grid and predicts bounding boxes + class probabilities for each cell.
- Extremely fast – suitable for real-time applications.

SSD (Single Shot MultiBox Detector)

- Uses multiple feature maps at different scales to detect objects of various sizes.
- Also a one-stage detector, balancing accuracy and speed.

Modern detectors (e.g. YOLOv5/8, RetinaNet, DETR) further improve speed–accuracy trade-offs.

YOLO model



Source: Redmon et al. You only look once: unified, real-time object detection, 2015

Practical example: Object detection

Original source of the example

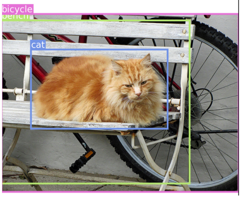
- **Cholett: Deep learning with Python**

coco_object_detection.ipynb

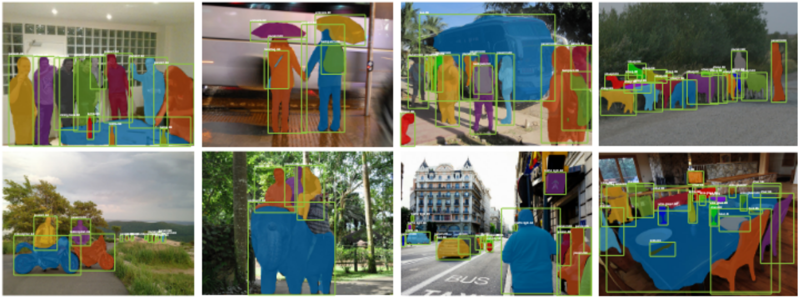
- Two examples:
  - Training a YOLO model from scratch on the COCO dataset
  - Using a pretrained RetinaNet detector

COCO (Common Objects in Context)

- Dataset designed for object detection, segmentation, and image captioning



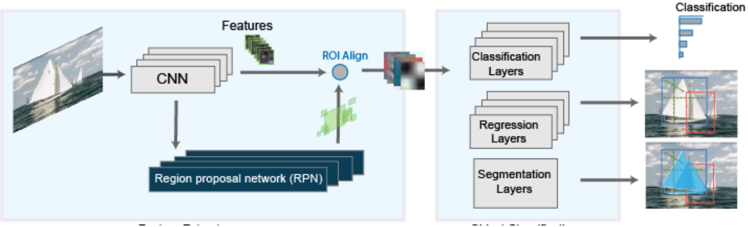
CNN and Instance Segmentation



Source: He et al., Mask R-CNN, 2017, Figure 2,
`https://arxiv.org/abs/1703.06870`

CNN and Instance Segmentation

- **Mask R-CNN**: an extension of Faster R-CNN that adds a parallel head to predict a **pixel-level segmentation mask** for each detected instance.



Source: `https://www.mathworks.com/help/vision/ug/getting-started-with-mask-r-cnn-for-instance-segmentation.html`

CNN and Human Pose Estimation

- **Mask R-CNN**: can also be extended to predict human keypoints (e.g., COCO Keypoints dataset).

Source: He et al., Mask R-CNN, 2017, Figure 7,
https://arxiv.org/abs/1703.06870

Practical Example: Instance Segmentation with SAM

Segment Anything Model (SAM, Meta AI, 2023)

- Zero-shot segmentation model — no retraining needed.
- Predicts object masks based on:
  - points,
  - bounding boxes,
  - or automatically (mask proposal mode).
- Works on any domain without fine-tuning.

instance_segmentation_using_a_pretrained_model.ipynb

Practical Example: Instance Segmentation with SAM

Segment Anything Model (SAM, Meta AI, 2023)

- Predicts object masks based on:
  - points,
  - bounding boxes,
  - or automatically (mask proposal mode).

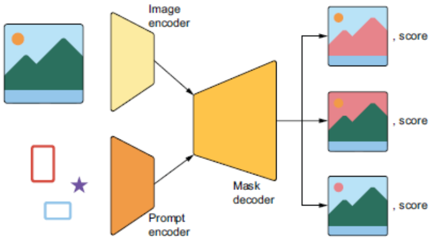
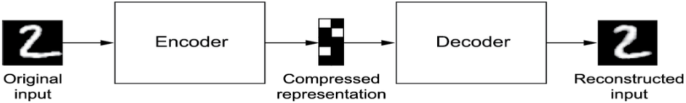


Image source: F. Chollet, *Deep Learning with Python*, Fig. 11.8

Applications of Convolutional Neural Networks

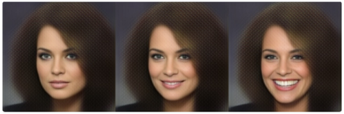
Autoencoders



F. Chollet: Deep Learning with Python, Fig. 12.4

Variational Autoencoders

- Encode data as a probability distribution
- Capable of generating new data samples or creating interpolations



F. Chollet: Deep learning v jazyku Python, obr. 5.7

Source: Tom White, "Sampling Generative

Applications of Convolutional Neural Networks

Processing 2D Images

- Image classification — classification head
- Regression tasks — regression head
- Object detection — detection head
- Image segmentation — segmentation head (encoder-decoder architecture)
- Image restoration, style transfer, image generation — autoencoder architecture
- Image captioning, pose estimation, facial feature detection, image similarity evaluation, . . .

Other Data Types

- Video analysis (3D convolutions) — action recognition (e.g., in sports recordings)
- Sequential data (1D convolutions) — time series, audio data, limited use for natural language

Advantages and Disadvantages of Convolutional Neural Networks

- Well-suited for grid-like data (e.g., images)
- Invariance to translation, scale, and color changes
- Robust to noise in the data
- Computationally intensive training; requires large datasets and GPUs
- Risk of overfitting, especially with small datasets
- Vulnerable to adversarial examples — still an open problem *(small invisible perturbations causing incorrect classifications)* https://www.tensorflow.org/tutorials/generative/adversarial_fgsm

This Week

1 Applications of Convolutional Neural Networks
- Object Detection
- Instance segmentation
- Autoencoders
- Other Applications

2 Sequential data
- Tasks
- Time Series Forecating

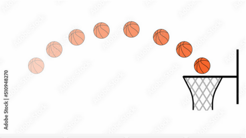
3 Recurrent Neural Network (RNN)
- Vanilla RNN

4 Graded Homework

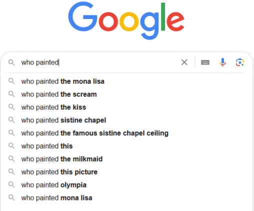
Sequential Data

Example tasks:

- Where will a thrown ball be in the next moment?



- Complete the sentence:



- Who is speaking?

Sequential Data

Time series

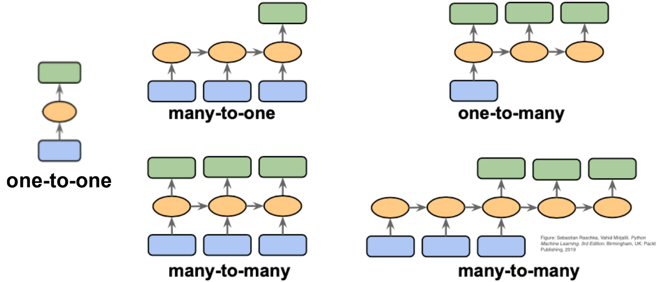
- Different granularity: daily stock prices, hourly electricity consumption, weekly store sales, ...
- Different dynamics: website traffic, credit-card transactions, seismic activity, weather evolution, ...

Sequential data are not only time series:

- **Audio:** speech recognition, speaker identification, emotion detection, acoustic localization, music analysis
- **Text:** sentiment analysis, machine translation, next-word prediction
- **Video:** action recognition, object tracking, trajectory prediction, video captioning
- **Biological signals:** DNA sequence analysis, heart-rate monitoring (ECG), . . .

Types of Tasks on Sequential Data

- **one-to-one** — standard classification
- **many-to-one** — sentiment analysis, action recognition
- **one-to-many** — image captioning, sentence tagging, music generation
- **many-to-many** (direct /delayed) — object tracking, machine translation,



Source: S. Raschka: Introduction to Recurrent Neural Networks

Time Series

Time series

- Sequential data with typical dynamics:
  - periodicity (daily, annual, ...)
  - trends in time: regular regime, sudden spikes,...

Typical tasks:

- **Forecasting** — predict the next value (many-to-one) or the next sequence (many-to-many)
- **Classification** — e.g., bot vs. human web visitor, heart-attack risk from ECG
- **Event detection** — seismic activity, keyword spotting ("OK Google")
- **Anomaly detection** — unusual behavior in network traffic
  - usually solved with unsupervised learning: clustering, autoencoders

Example: Time Series Forecasting

Example: Jena Climate Dataset time_series_jena.ipynb

- Meteorological data recorded at the Max Planck Institute in Jena (Germany), years 2009–2016.
- 15 features (timestamp, temperature, pressure, humidity, wind speed/direction, ...)
- Measurements taken every 10 minutes — roughly $\sim$ 400,000 samples.

Task

- Predict the temperature **24 hours into the future** using the previous 5 days of data.
- Use 14 input features (timestamp omitted).
- We downsample to hourly measurements $\rightarrow 24 \times 5 = 120$ time steps.
- Each training example has shape 120×14, and the output is a single value (temperature in 24 hours).

Example: Time Series Forecasting

Loading and preparing the data

- **Example:** reading the CSV file and creating time-series datasets using Keras utilities.
- **Important:** Validation and test sets must follow the training set **in time**.
  - Common mistake: randomly shuffling data before splitting into train/val/test.
  - Only the **training samples** may be shuffled.

Baseline level

- **Naive forecast:** "The temperature in 24 hours will be the same as now."
- Baseline: $MAE_{\text{test}} = 2.62^\circ\mathrm{C}$ — surprisingly strong for this dataset.

Example: Time Series Forecasting

Solution using an MLP

- The MAE remains close to the baseline.
- Why is it not better?
  - The MLP sees all 120×14 numbers at once $\rightarrow$ it must "find a needle in a haystack".
  - It has no notion of temporal structure.

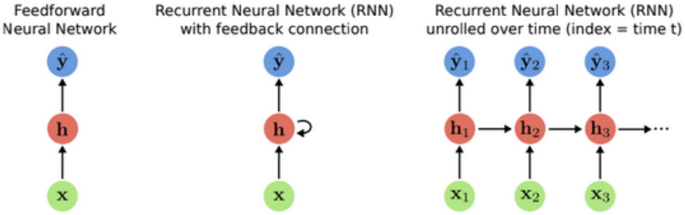
Solution using a 1D CNN

- Uses 1D convolution along the time axis.
- Performs even worse than the baseline.
- Why?
  - Time series are **not shift-invariant**.
  - The order matters — recent history is more important than distant history.

The MLP and CNN models fail. What to do next?

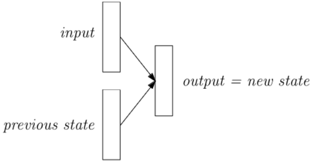
- Try a model dedicated to sequential data: a recurrent neural network (RNN)

Recurrent Neural Network (RNN)

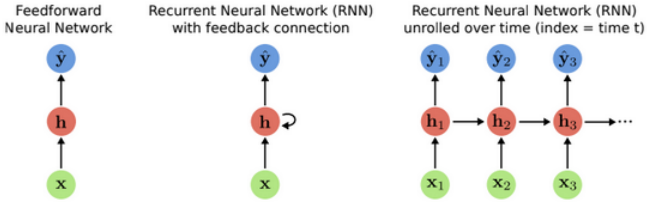


Recurrent NN model = a model with cycles

- **Idea:** Neurons maintain an internal state (**memory**).
- Output depends not only on the current input but also on the previous state.
- RNN neurons are sometimes called *memory cells*.

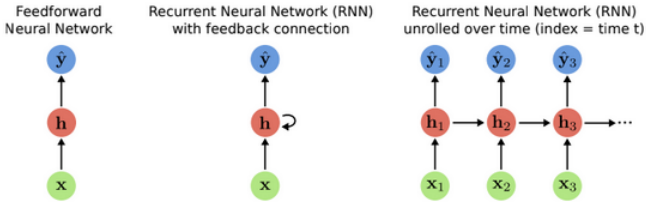


Vanilla (Elman) RNN



- The neuron has a hidden state: $h_t = f_h(h_{t-1}, x_t)$
- Output: $y_t = f_y(h_t)$
- The model processes the whole input sequence:
  - of arbitrary length,
  - can be "unrolled" over time (right figure),
  - functions f_h and f_y share parameters across time steps.
- The hidden state is reset for each new input sequence.

Vanilla (Elman) RNN: Equations



- Hidden state:

$$h_t = \tanh(W_h h_{t-1} + W_x x_t + b_h)$$

- Output:

$$y_t = W_y h_t + b_y$$

- Parameters W, b stay the same for all time steps.

Vanilla RNN – unrolled

- Examples of unrolled architecture for different sequence-to-sequence mappings:

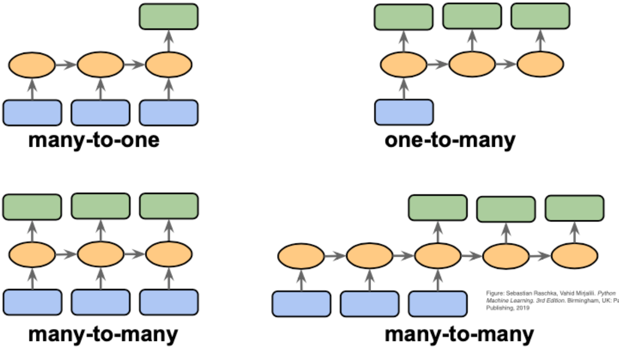
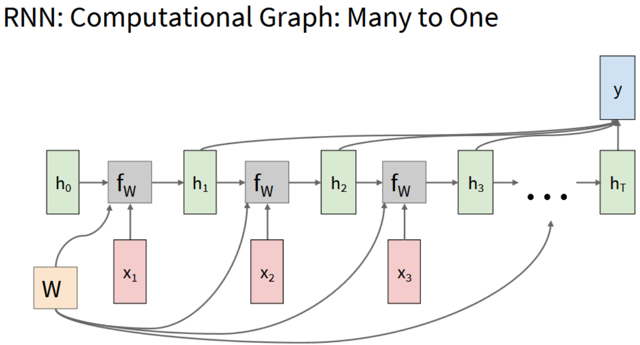


Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Birmingham, UK: Packt Publishing, 2019

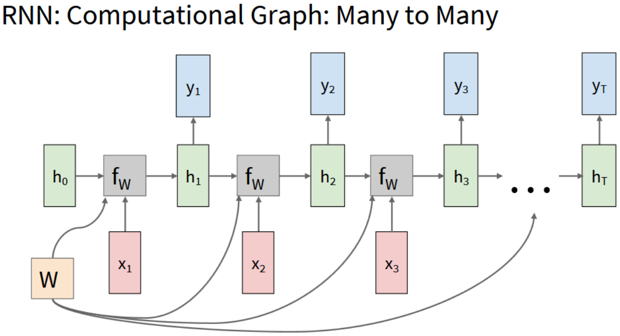
Vanilla RNN – unrolled



- RNN unrolled in time can be viewed as a deep feed-forward neural network
- The main difference: unrolled units share parameters (weights and biases)

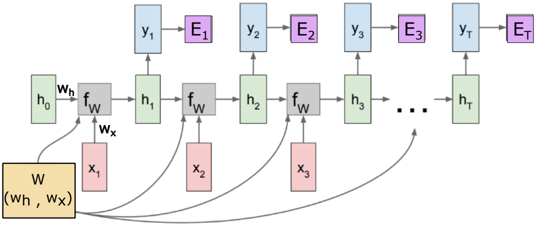
Source: S. Raschka: Introduction to Recurrent Neural Networks, https://sebastianraschka.com/blog/2021/dl-course.html

Vanilla RNN – unrolled



Source: S. Raschka: Introduction to Recurrent Neural Networks,
https://sebastianraschka.com/blog/2021/dl-course.html

Backpropagation Through Time (BPTT)



Source: S. Raschka: Introduction to Recurrent Neural Networks, 2021

- An extension of standard Backpropagation for RNN
- Shown for a many-to-many architecture.
- Gradients are backpropagated through all time steps at once (for a sequence or batch of sequences).

Practical examples

- Next week ...

Practical example: RNN Layers in Keras

- **RNN_layers.ipynb**

Practical example: Jena Climate Dataset

- **time_series_jena.ipynb**

7th Graded Homework: Time Series Prediction

Goal: Build and compare several NN models for **multistep forecasting** on the **Jena Climate** dataset.

Task:

- Use meteorological data sampled every 10 minutes.
- Predict the next **6 future steps** (next hour) of:
  - temperature,
  - atmospheric pressure ("p (mbar)")
- Input: last **3 days** of data (432 time steps $\times$ selected features).

Start from the example notebooks:

- **time_series_jena.ipynb**
- **RNN_layers.ipynb**

7th Graded Homework: Suggested Workflow

Hints

- When modifiing the code of **time_series_jena.ipynb**, think about input shape (batch, 432, features) and output shape (batch, 6, 2), and how to adapt the loss function accordingly

Models to compare:

- **Model 1:** Simple baseline (naive forecast: "next hour = keep last value").
- **Model 2:** MLP or 1D–CNN baseline
- **Model 3:** Simple recurrent model (LSTM or GRU).
- **Models 4–?:** Variants of an improved recurrent model (e.g., stacked LSTM/GRU, recurrent dropout, gradient clipping, layer normalization, bidirectional RNN).

7th Graded Homework: Requirements

Include in your notebook:

- Training curves (MAE, RMSE).
- Final metrics on the test set.
- Visualization of the predictions.
- Short discussion: Which architecture performs best and why?

Submission

- Submit the notebook by **Dec 8, 2025**.
- **Consultation required by Dec 12, 2025** to receive points (short discussion after lab or individually).
- **Points: 1–2**
  - +1 point for the four required models and evaluation
  - +1 point for discussing more variants of extended RNNs