Hands-on Session: Machine Learning

A (Very) Short Primer on the Basics of Deep Learning

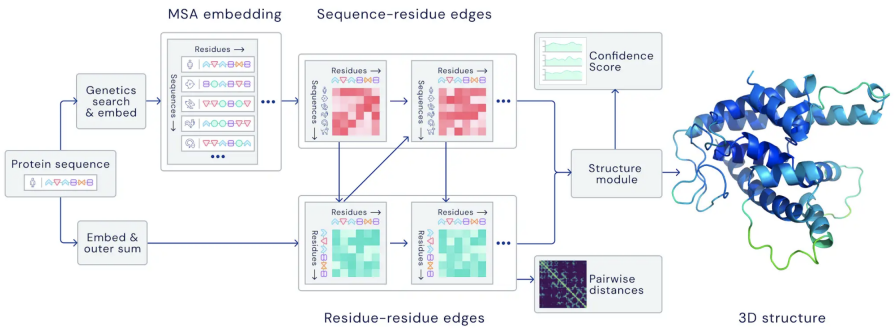
Tobias Kortus (tobias.kortus@rptu.de),
Alexander Schilling (alexander.schilling@rptu.de)

Scientific Comuting Group
NHR @ SW - Method Lab AI and ML (https://nhrsw.de/mls/)
University of Kaiserslautern-Landau (RPTU)

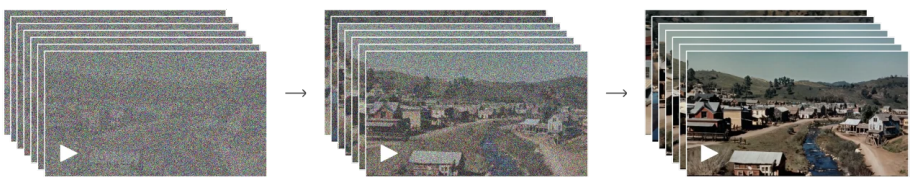
HighRR Lecture Week
10 June 2024 - 14 June 2024

NHR@SW

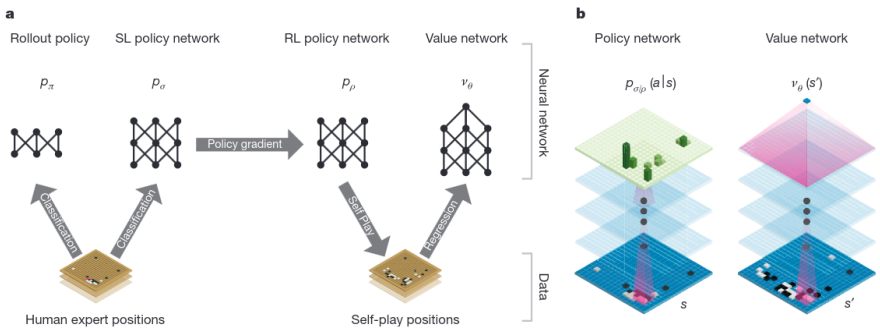
Motivation



https://www.nature.com/articles/s41586-021-03819-2



https://openai.com/index/video-generation-models-as-world-simulators/



https://www.nature.com/articles/nature16961

Machine Learning and Deep Learning

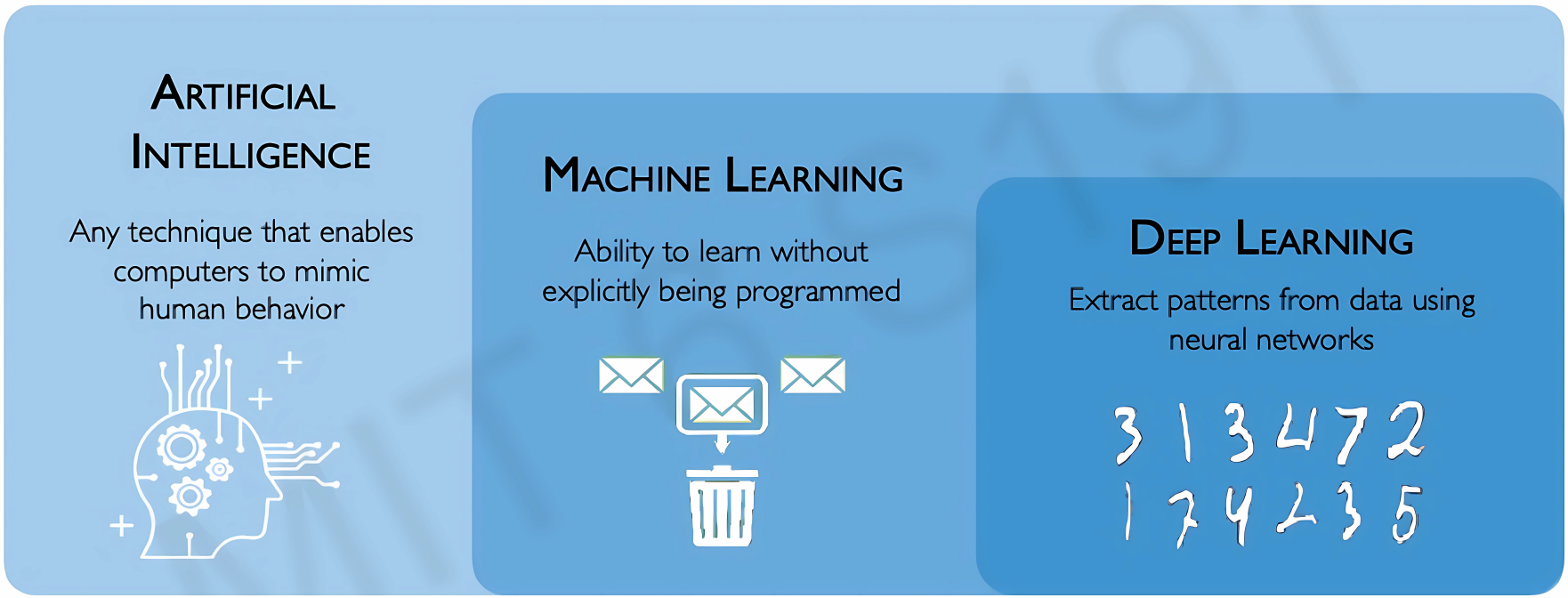


Figure from "MIT Introduction to Deep Learning" (`http://introtodeeplearning.com`)

From Linear Regression to Deep Learning

- We want to find parameters $\hat{\beta}$ for a linear model

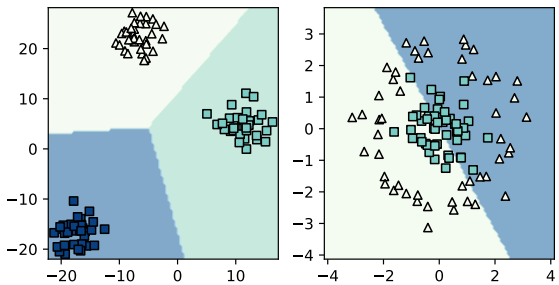
$$\hat{y} = \mathbf{X'}^T \hat{\beta}, \tag{1}$$

- that minimize the residual sum of squares, defined as

$$RSS(\hat{\beta}) = \left(y - \mathbf{X'}^T \hat{\beta}\right)^T \left(y - \mathbf{X'}^T \hat{\beta}\right). \tag{2}$$

- We can find the unique solution of the quadratic function by differentiating w.r.t. $\hat{\beta}$:

$$\hat{\beta} = \left(\mathbf{X'}^T \mathbf{X'}\right)^{-1} \mathbf{X'}^T y \tag{3}$$

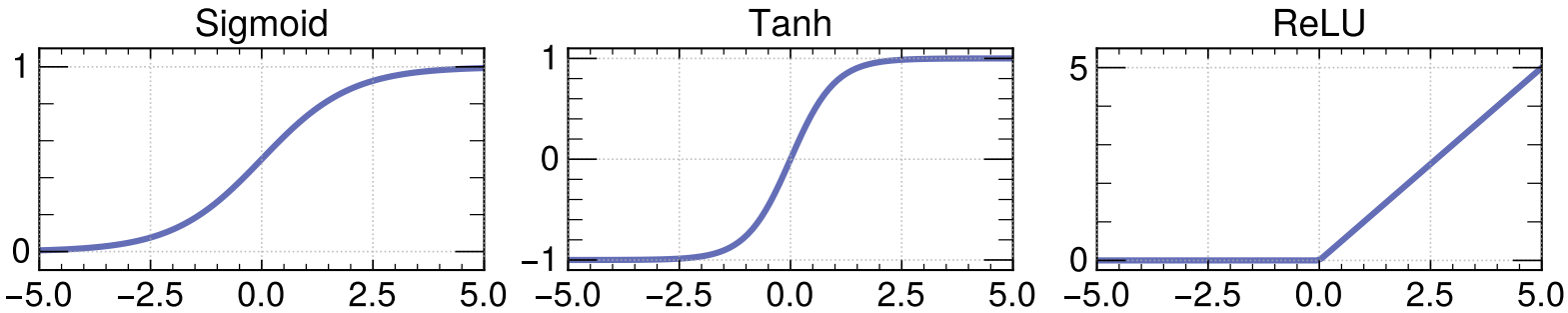


Linear regression/logistic regression fails if output is not linear/ linearly separable $\rightarrow$ requires additional **feature engineering**.

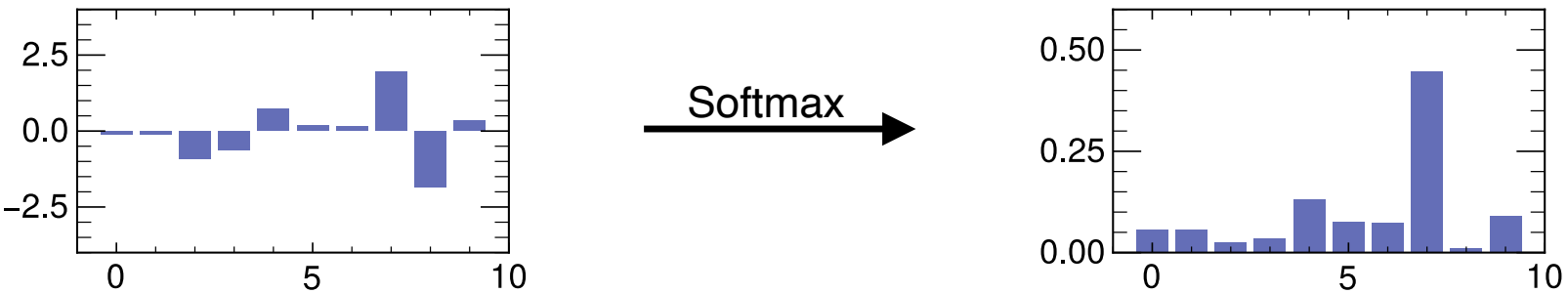
- Deep learning aims to **learn directly from raw data** without feature engineering.
- **Idea**: stack multiple network layers after each other with nonlinear activation functions.
- **Limitation**: optimization becomes non-convex & no closed form solution.

Activation Functions

- Activation functions introduce non-linearities into the network.
- Basic requirements to learn complex non-linear relations (e.g. XOR Problem).
- The choice of the activation function is a hyperparameter. The most common choice however is ReLU or one of its variants.

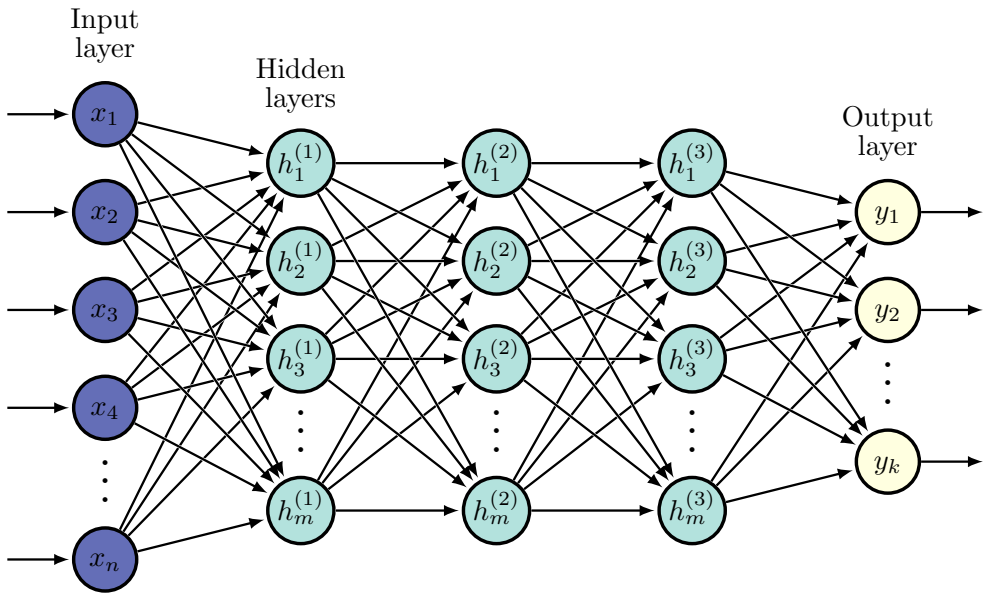


- **Softmax**: special activation for classification output → converts a vector of K real numbers into a probability distribution of K possible outcomes.



Feed-Forward Networks/ Multilayer Perceptron

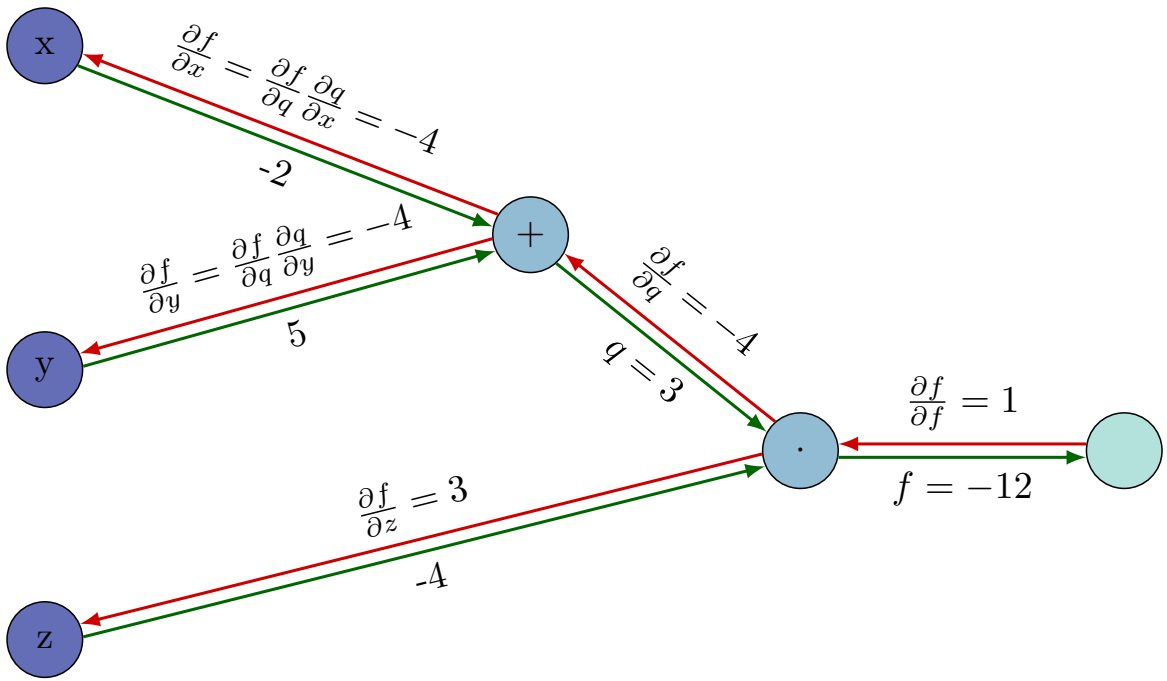
- Feed-Forward networks are composed in a multilayer structure with the following components:
  1. an **input layer**
  2. one or multiple **hidden layer**
  3. one **output layer**
- where each layer is connected with a non-linear activation function (e.g., ReLU). The activation function of the output layer (if required) is chosen based on the target value range.



Backpropagation

Deep neural networks are composed of a complicated set of functions, thus differentiating the whole expression is not trivial.

→ Split all computations of neural network into atomic operations (e.g., addition, subtraction, multiplication) and use chain rule to decompose derivatives:

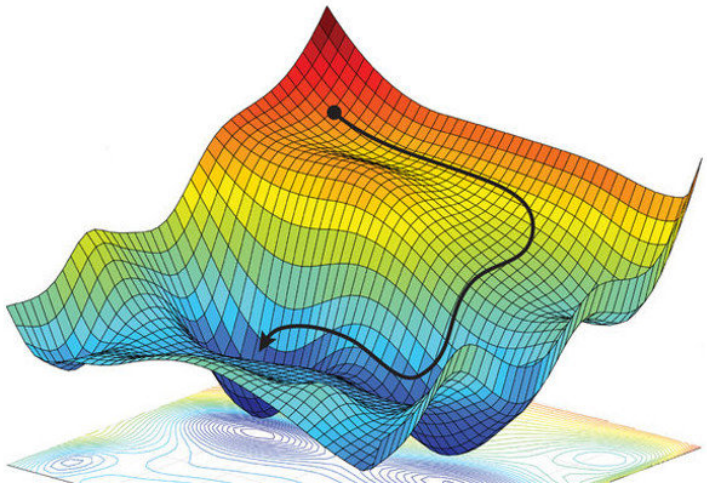


Gradient Descent

Algorithm 1: Gradient Descent

Input: Step size: η
Tolerance: ϵ

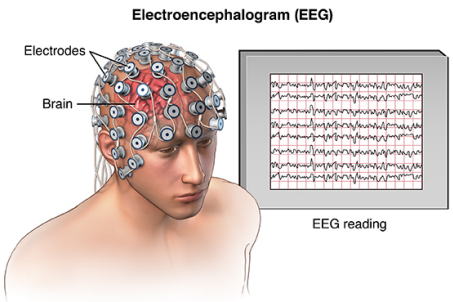
Initialize $\mathbf{w}^0$ randomly;
while $\|\mathbf{v}\| \geq \epsilon$ **do**
  for $i = 1 \ldots N$ **do**
    Forward pass $\rightarrow \mathcal{L}(\hat{y}_i = f_{\mathbf{w}}(\mathbf{x}_i), y_i)$
    Backward pass $\rightarrow \nabla_{\mathbf{w}} \mathcal{L}(\hat{y}_i, y_i)$
  end
  Gradient $\mathbf{v} = \sum_i^N \nabla_{\mathbf{w}} \mathcal{L}(\hat{y}_i, y_i)$
  Update $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \mathbf{v}$
end

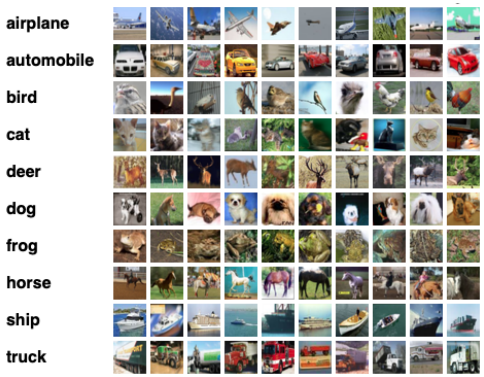


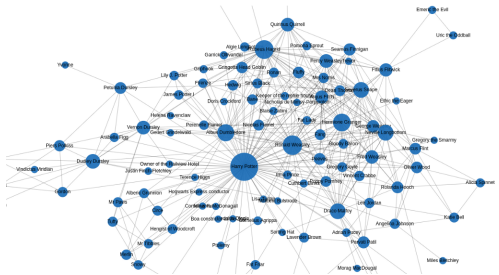
- **Challenges with Vanilla Gradient Descent**
  1. Choosing a proper learning rate can be difficult: slow convergence/ divergence
  2. Escaping flat regions such as plateaus or saddle points is notoriously hard
  3. Performs same gradient updates for all parameters
  4. ...
- Optimizer with, e.g., adaptive learning rate and/or momentum are often more robust to mentioned problems. $\rightarrow$ ADAM Optimizer

Going Beyond Simple Feed-forward Networks

- Data doesn't usually come in a tabular structure:







- Feed-fortward networks often struggle to efficiently learn from complex data:

Example: Parameter Explosion for Dense Networks

MNIST Dataset

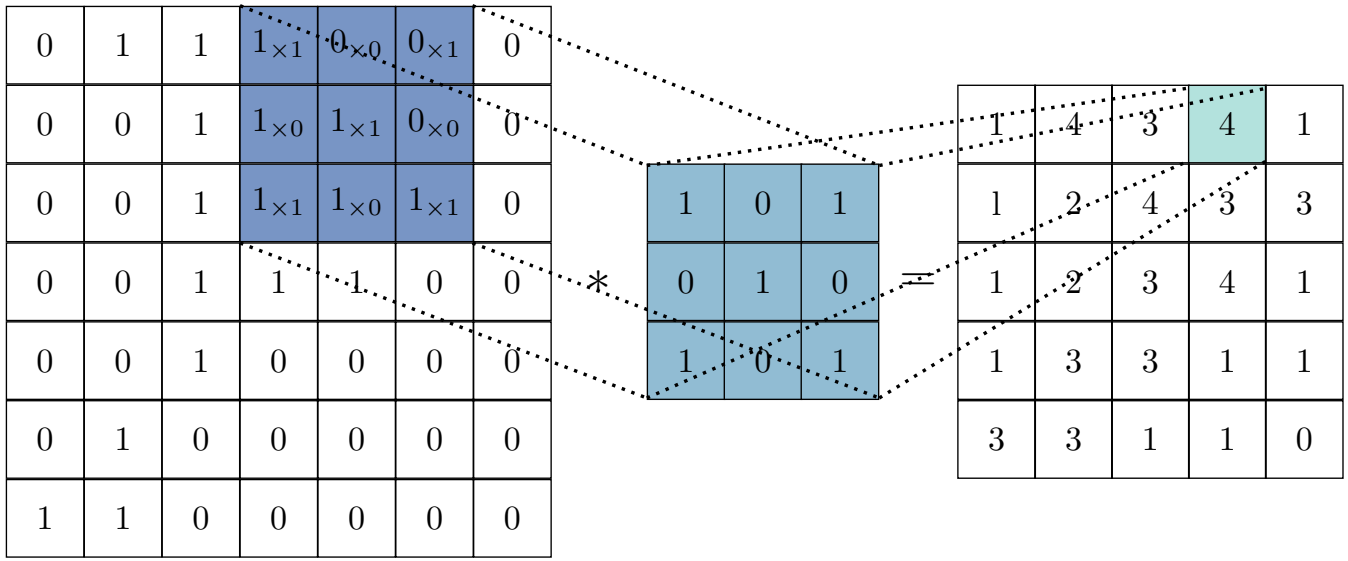
28×28 pixel $\rightarrow W_1 \in \mathbb{R}^{784 \times N}$
$N = 128 \approx 100k$ parameters

Cards Dataset (grayscale)

224×224 pixel $\rightarrow W_1 \in \mathbb{R}^{50176 \times N}$
$N = 128 \approx 6M$ Parameters

Convolutional Neural Networks

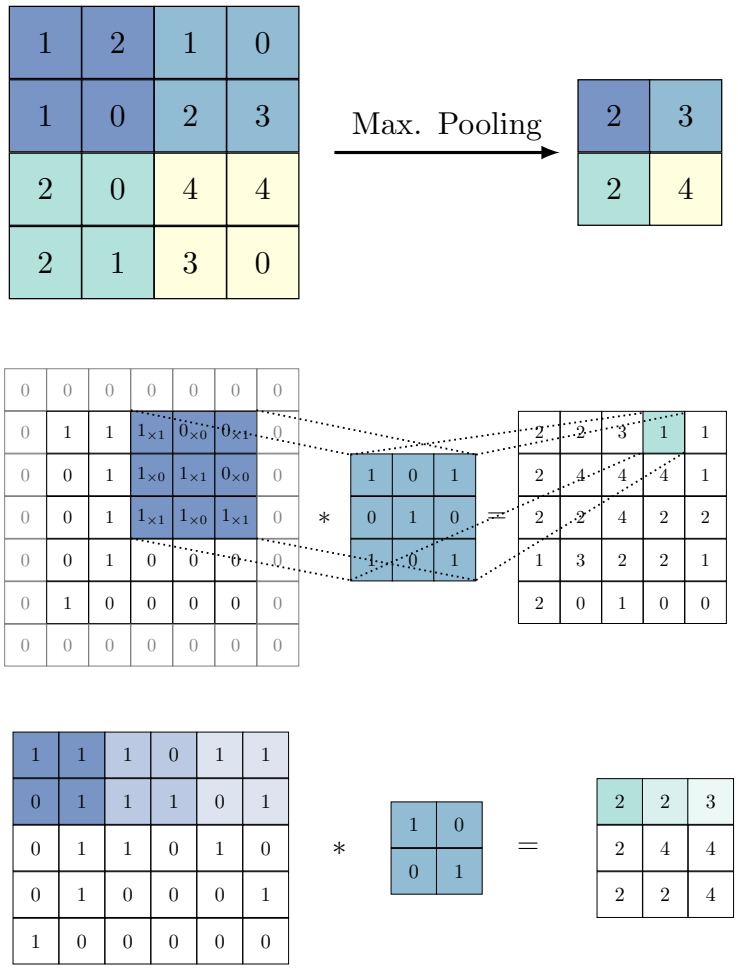
- For a long time (and still) fundamental component of DL architectures used for image, signal processing and natural language processing.
- Kernel matrix as a learnable network parameter.



- Convolutional layer introduces inductive bias in terms of:
  1. Sparse interaction/ parameter sharing
  2. Equivariance to translation

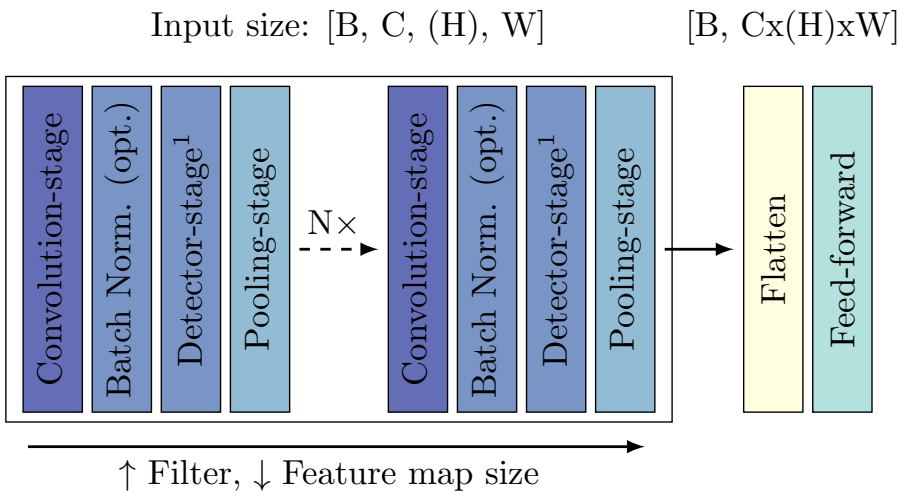
Padding, Pooling and Stride

- **Pooling**: Aggregate statistic over input neighborhood
  - output rep. is approximately invariant to small translations.
- **Padding**: Added zeros/copied values around grid topology
  - control the spatial size of the output volumes.
- **Stride**: Parameter controlling the amount of movement of kernel
  - Reduce size of output volume + improved comp. efficiency



Convolutional Neural Networks

- Given the introduced components, an entire convolutional network can be constructed using the following recipe:



- The output of each convolution block is defined as:

$$n_{out} = \frac{n_{in} + 2p - k}{s} + 1 \qquad (4)$$

- n_{in}: number of input features
- n_{out}: number of output features
- k: convolution kernel size
- p: convolution padding size
- s: convolution stride size

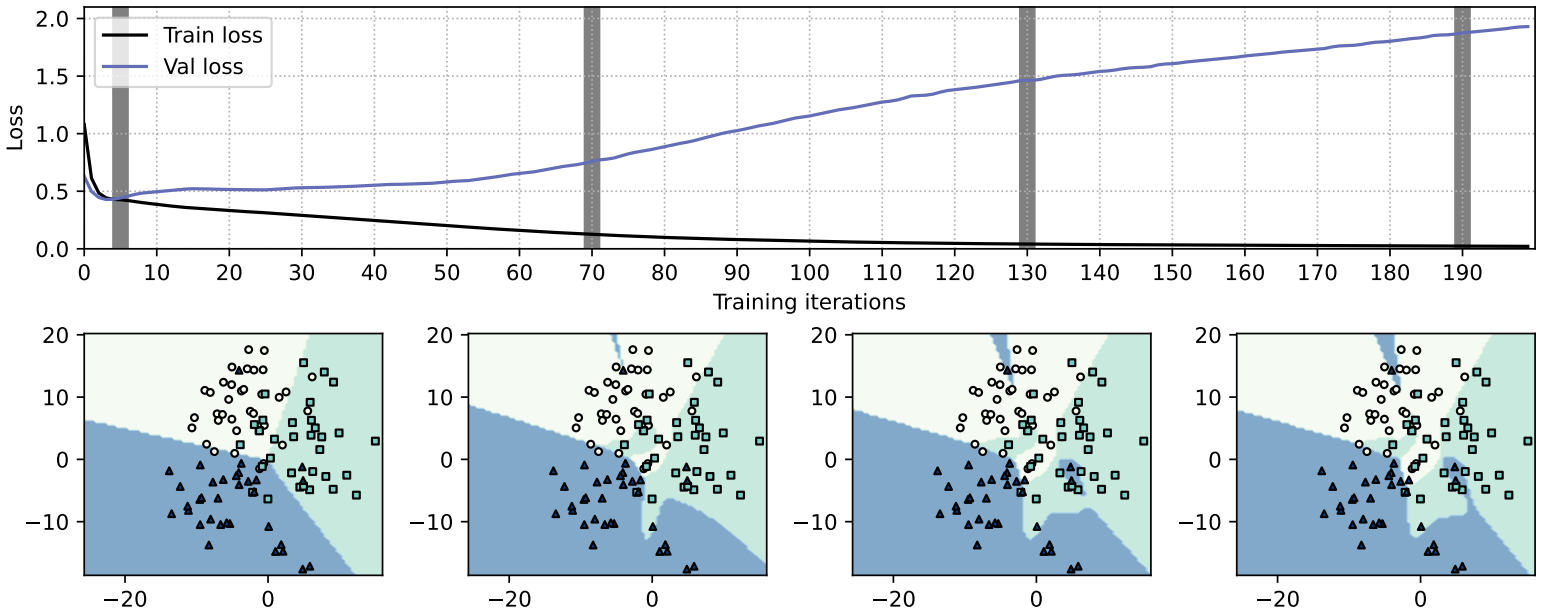
[1] Detector-stage = non-linear activation

Model Evaluation

- **Never evaluate performance with an example used for training**
  - Estimate the performance on *previously unseen* examples
  - Get a sense of how well the model *generalizes*
- Reserve a portion of the available data for testing
  - How much data we reserve depends on the problem
  - Often a reasonable split: 70% for training, 30% for evaluation
  - Training set: used for model learning
  - Test set: used for evaluation
- Problems
  - Potential waste of data: we want to use as much data for training as possible
  - High variance: performance is highly dependent on the data split
  - In cases where only little data is available $\rightarrow$ k-fold cross-validation

Model Capacity, Underfitting and Overfitting

- **Underfitting**: Model can not capture the complexity of the data → bigger network or more training iterations
- **Overfitting**: Model can not generalize and fits closely to training data → smaller network or regularization techniques



- Measuring the model performance during training to detect under and overfitting requires additional validation dataset.

Regularization Techniques

- An incomplete list of regularization techniques for deep learning:
  - **Parameter norm penalties** (L1 & L2 regularization)
    - Limit the model capacity by penalizing high parameter values with L1 $|w|$ or L2 norm $\|w\|$ (Also known as ridge-/ lasso regression in linear models).
  - **Dropout** [1]
    - Randomly "disable" network connections to avoid co-adaption of learned features (creates a new network architecture from the parent network).
  - **Batch normalization**
    - Intermediate layers may take values with widely varying magnitudes $\rightarrow$ normalize the inputs.
  - **Data augmentation**
    - Generate new training instances from a relatively small dataset by modifying existing samples (e.g. rotation, noise, blur, crop & resize, ...)
  - **Early stopping**
    - Stop training when validatation loss increases.

[1] Can be used during inference to obtain an approximate Bayesian network. (`https://arxiv.org/abs/1506.02142`)

Additional Literature[2]

- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. (2016). Deep Learning. MIT Press. (`https://www.deeplearningbook.org/`)
- Zhang, Aston Lipton, Zachary Li, Mu Smola, Alexander. (2021). Dive into Deep Learning. (`https://d2l.ai/index.html`)
- Hastie, T., Tibshirani, R.,, Friedman, J. (2001). The Elements of Statistical Learning. New York, NY, USA: Springer New York Inc.
- Kevin P. Murphy. (2022). Probabilistic Machine Learning: An Introduction. MIT Press. (`https://probml.github.io/pml-book/book1.html`)
- Kevin P. Murphy. (2023). Probabilistic Machine Learning: Advanced Topics. MIT Press. (`https://probml.github.io/pml-book/book2.html`)
- Christopher M. Bishop. (2006). Pattern Recognition and Machine Learning. Springer-Verlag, Berlin, Heidelberg.

[2] All additional references are listed without any particular ordering in mind.