Further Topics

TECHNISCHE UNIVERSITÄT DARMSTADT

Data Mining and Machine Learning: Techniques and Algorithms

Eneldo Loza Mencía

eneldo@ke.tu-darmstadt.de

Knowledge Engineering Group, TU Darmstadt

International Week 2019, 21.1. – 24.1.
University of Economics, Prague

What are Neural Networks?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Models of the brain and nervous system
- Highly parallel
  - Process information much more like the brain than a serial computer
- Learning

- Very simple principles
- Very complex behaviours

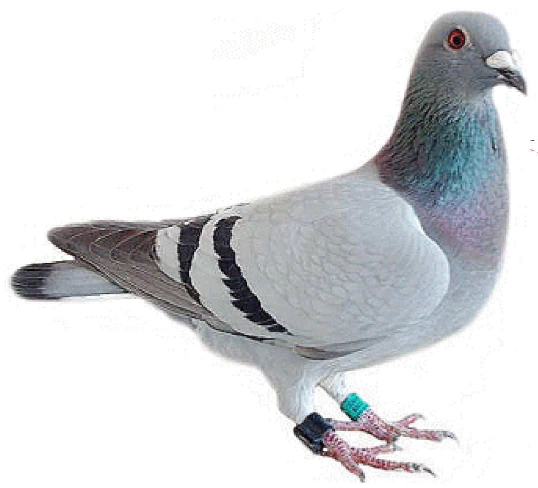
- Applications
  - As powerful problem solvers
  - As biological models

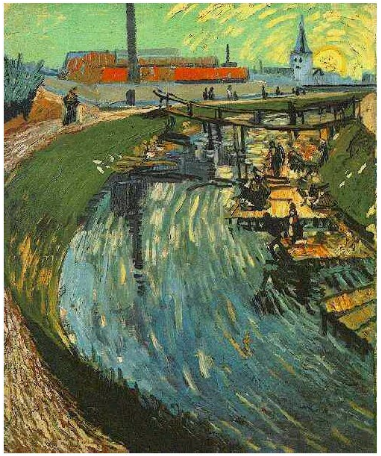
Pigeons as Art Experts

TECHNISCHE UNIVERSITÄT DARMSTADT

Famous experiment (Watanabe *et al.* 1995, 2001)

- Pigeon in Skinner box
- Present paintings of two different artists (e.g. Chagall / Van Gogh)
- Reward for pecking when presented a particular artist





Results

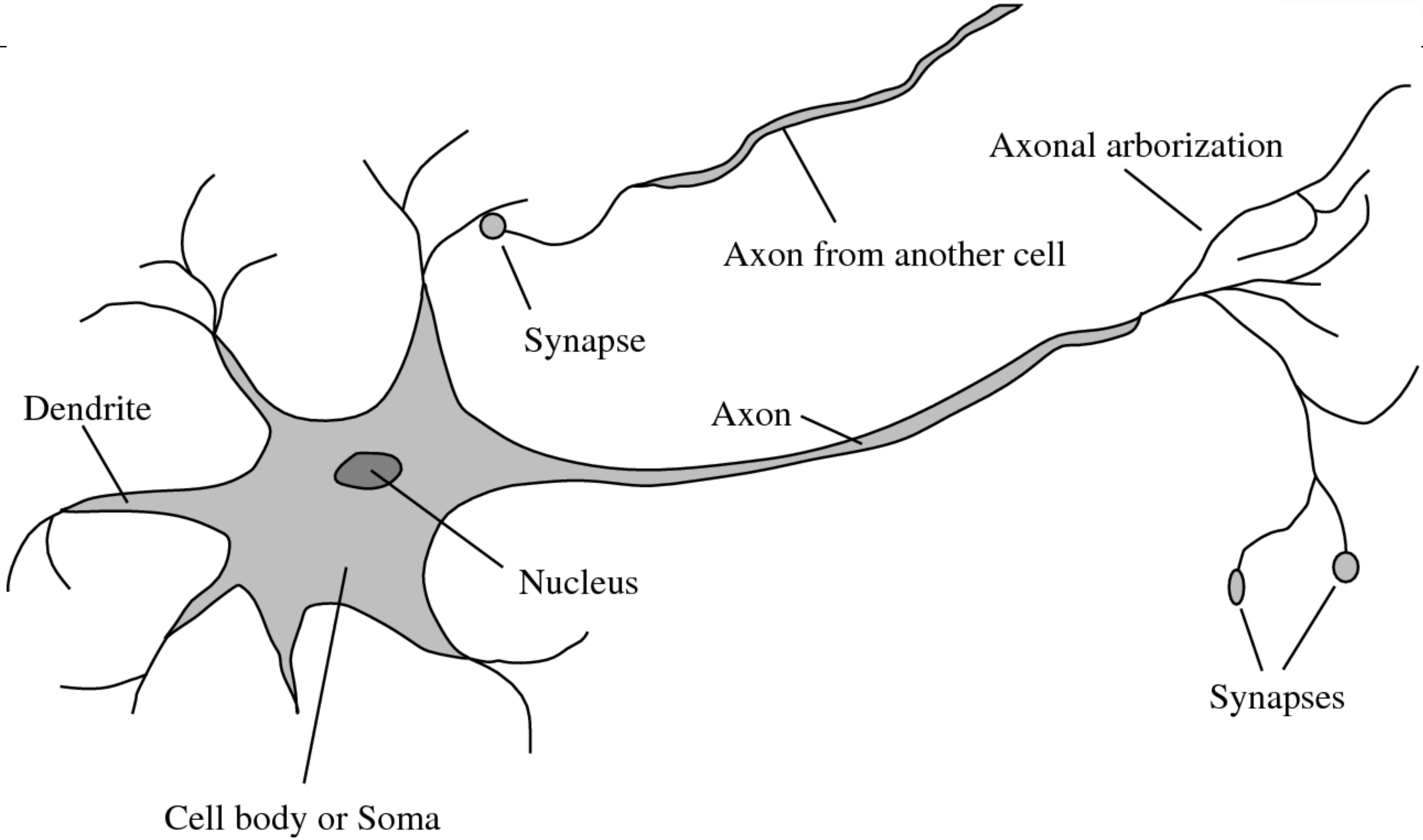
- Pigeons were able to discriminate between Van Gogh and Chagall with 95% accuracy
  - when presented with pictures they had been trained on

- Discrimination still 85% successful for previously unseen paintings of the artists

→ Pigeons do not simply memorise the pictures
  - They can extract and recognise patterns (the ‘style’)
  - They generalise from the already seen to make predictions

- This is what neural networks (biological and artificial) are good at (unlike conventional computer)

A Biological Neuron

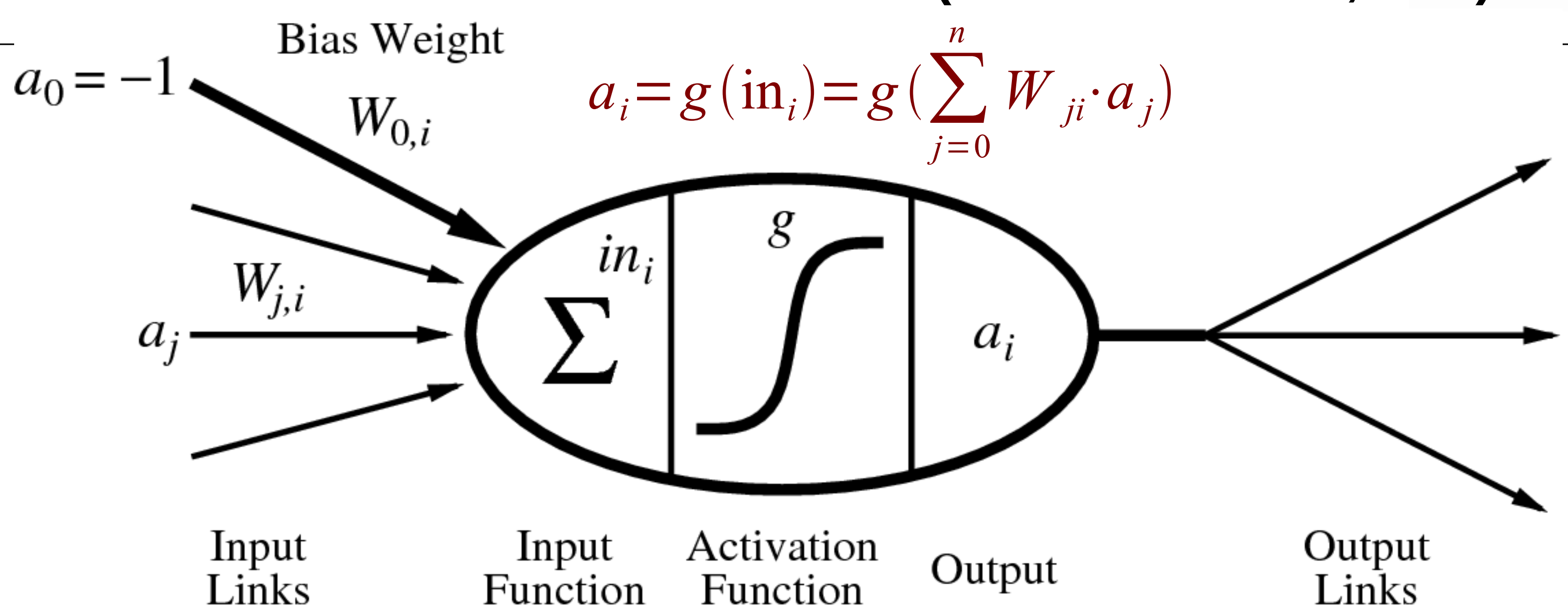


- Neurons are connected to each other via synapses
- If a neuron is activated, it spreads its activation to all connected neurons

An Artificial Neuron

(McCulloch-Pitts,1943)

$$a_i = g(\text{in}_i) = g\left(\sum_{j=0}^{n} W_{ji} \cdot a_j\right)$$



- Neurons correspond to nodes or units
- A link from unit j to unit i propagates activation a_j from j to i
- The weight $W_{j,i}$ of the link determines the strength and sign of the connection
- The total input activation is the sum of the input activations
- The output activation is determined by the activiation function g

Perceptron

(Rosenblatt 1957, 1960)

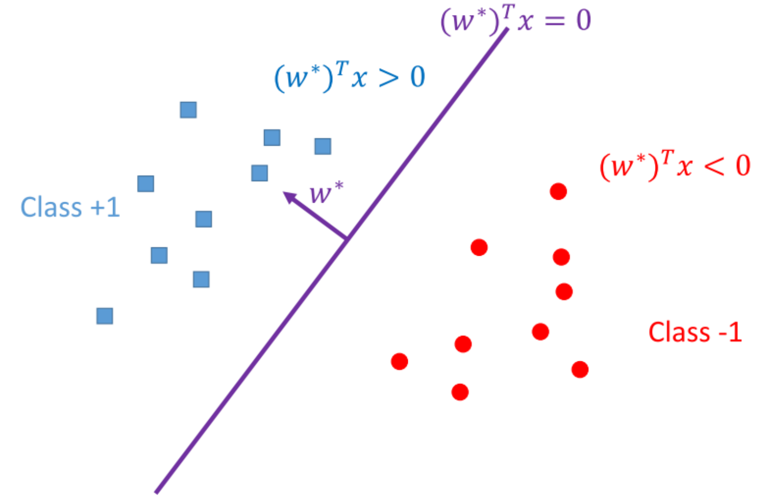
- A single node
  - connecting n input signals a_j with one output signal a
  - typically signals are -1 or $+1$

$$f_w(x) = w^T x$$

- Activation function
  - A simple threshold function:

$$y = \begin{cases} +1 \text{ if } w^T x > 0 \\ -1 \text{ if } w^T x < 0 \end{cases}$$

- Thus it implements a linear separator
  - i.e., a hyperplane that divides n-dimensional space into a region with output -1 and a region with output 1



Perceptron
Update rule

(Rosenblatt 1957, 1960)

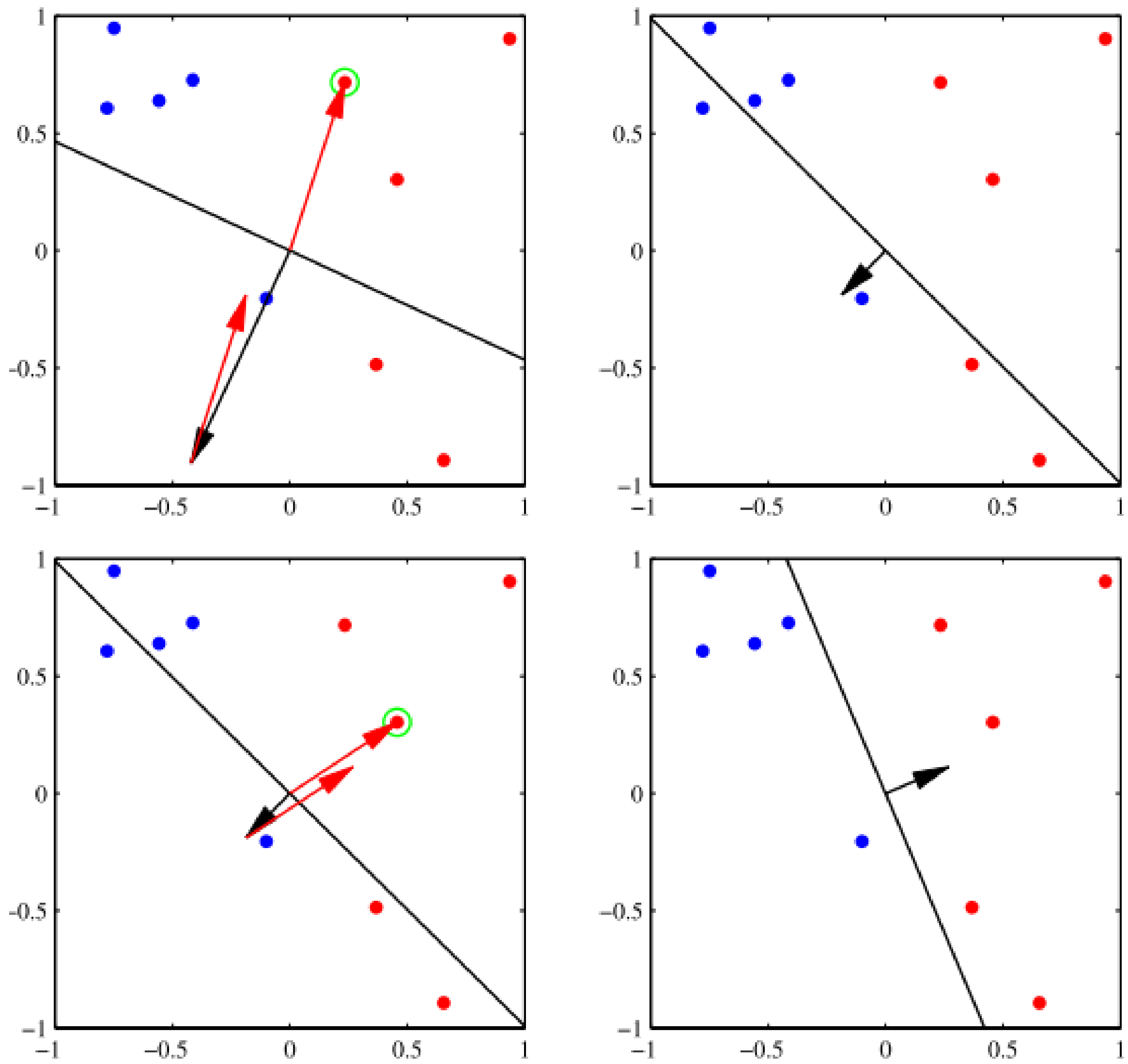
1. Start with the all-zeroes weight vector $\mathbf{w}_1 = \mathbf{0}$, and initialize t to 1.
2. Given example $\mathbf{x}$, predict positive iff $\mathbf{w}_t \cdot \mathbf{x} > 0$.
3. On a mistake, update as follows:
   - Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{x}$.
   - Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{x}$.

   $t \leftarrow t + 1$.

- Trained iteratively and incrementally (online learning)
- It is guaranteed to find separating hyperplane if existent
- Simple, but often competitive to state-of-the-art (SVM), especially for text classification (linear classifier work well there)

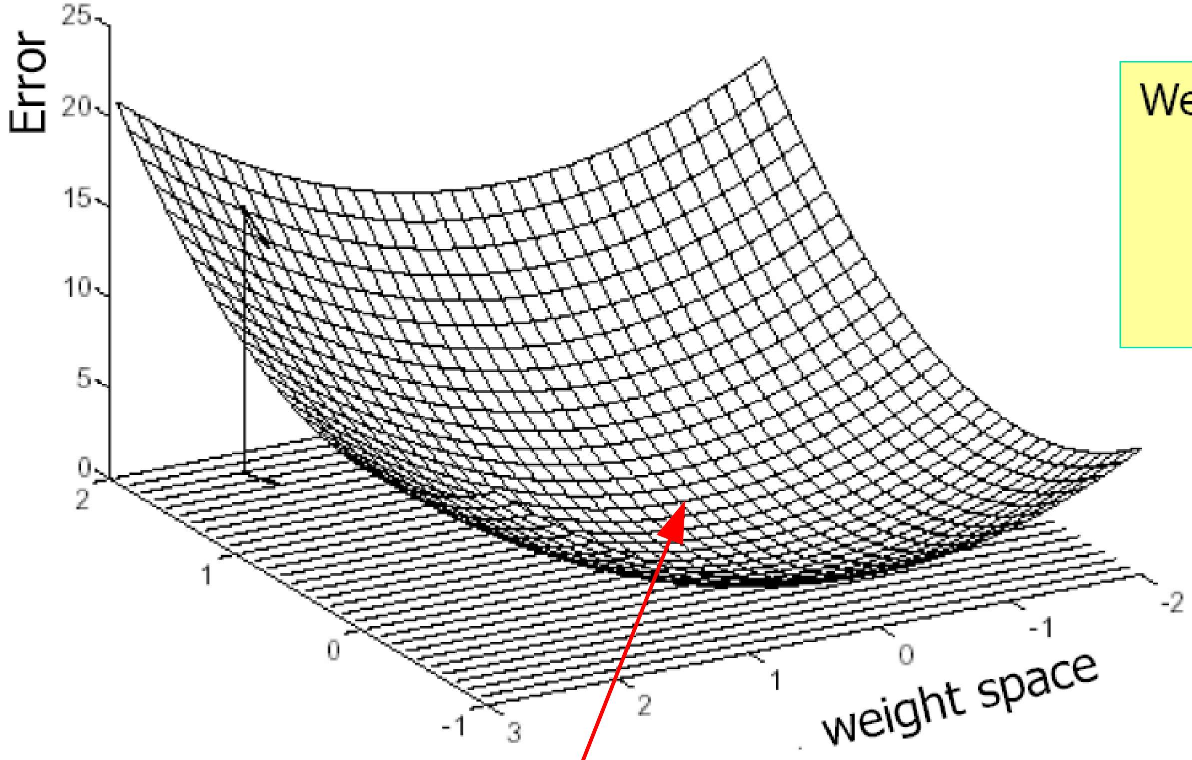
Perceptron
Update rule

(Rosenblatt 1957, 1960)



Error Landscape

- The error function for one training example may be considered as a function in a multi-dimensional weight space



Weight space is N-dimensional, where N is the total number of weights in the network

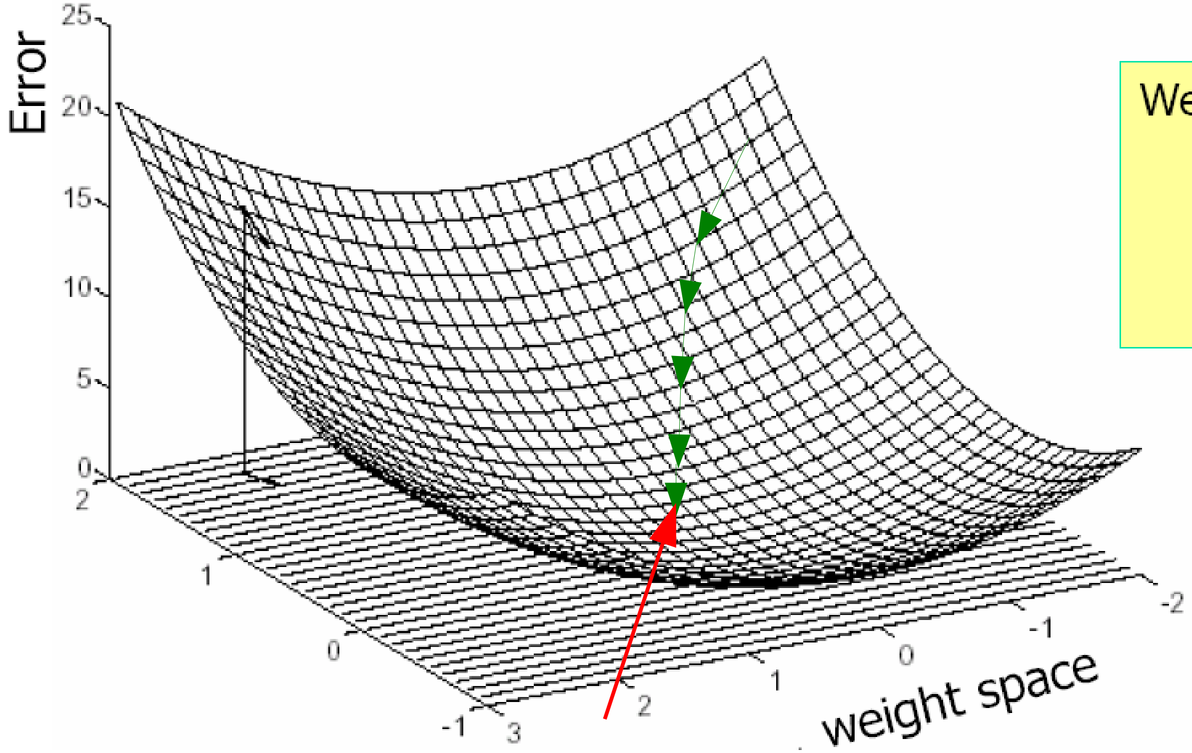
$$E(W)=\frac{1}{2}\left(f(\mathbf{x})-g\left(\sum_{j=0}^{n} W_j \cdot x_j\right)\right)^2$$

- The best weight setting for one example is where the error measure for this example is minimal

Error Minimization via Gradient Descent

TECHNISCHE
UNIVERSITÄT
DARMSTADT

- In order to find the point with the minimal error:
  - go downhill in the direction where it is steepest



Weight space is N-dimensional, where N is the total number of weights in the network

$$E(W)=\frac{1}{2}\left(f(\mathbf{x})-g\left(\sum_{j=0}^{n} W_j \cdot x_j\right)\right)^2$$

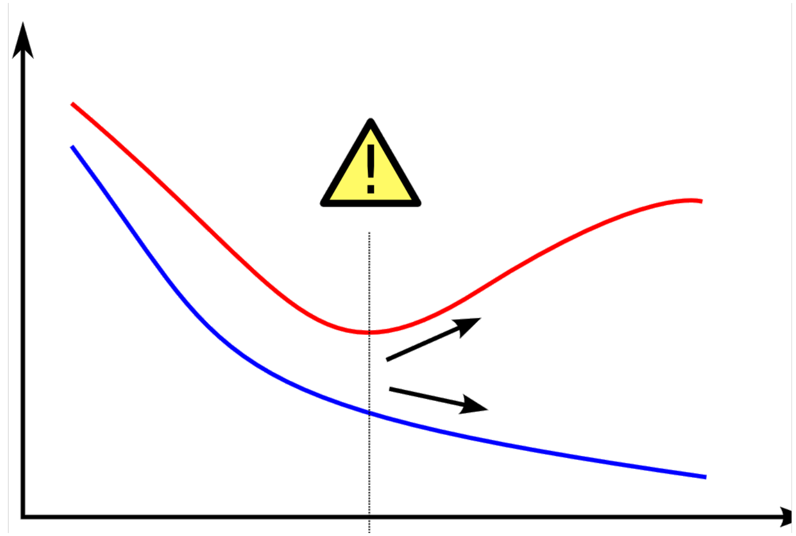
  - … but make small steps, or you might shoot over the target

Error Minimization via Gradient Descent

- Properly speaking, gradient descent is when you compute the gradient on the whole training set (batch), and then move your weights (=one epoch)
- Stochastic Gradient Descent (SGD) does a stochastic version of that: it approximates the gradient on only a few examples
  - Extreme case: presented perceptron algorithm, which only takes one example at the time $\rightarrow$ instable gradients, a lot of jumping around
  - Intermediate case: Mini-batches
    - take a small number of training examples randomly (e.g. n=32,128), compute the gradient, and descent
    - repeat N/n times for an epoch
    - has also computational advantages, since these batches fit into GPU memory and gradient can be computed in one step

Overfitting

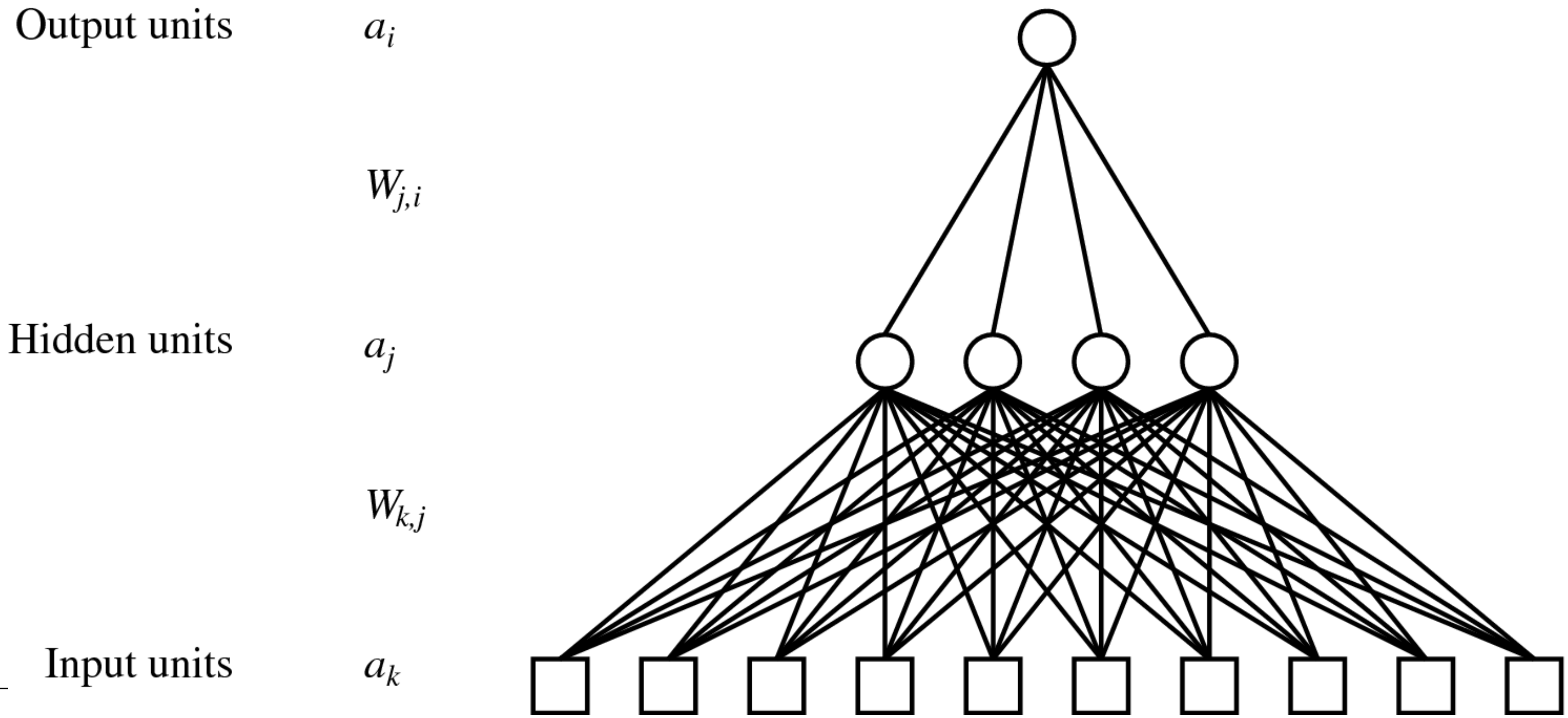
- Training Set Error continues to decrease with increasing number of training examples / number of epochs
  - an epoch is a complete pass through all training examples
- Test Set Error will start to increase because of overfitting



- Simple training protocol:
  - keep a separate validation set to watch the performance
    - validation set is different from training and test sets!
  - stop training if error on validation set gets down

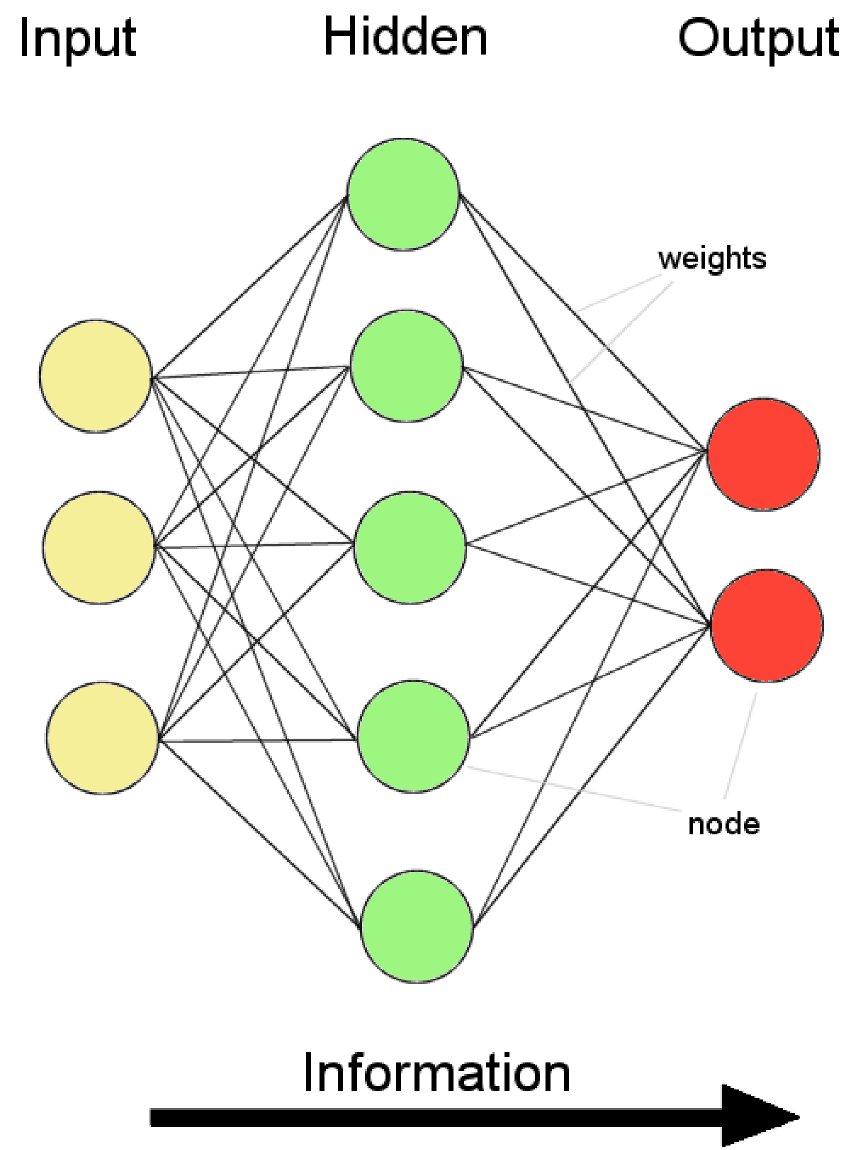
Multilayer Perceptrons

- Perceptrons may have multiple output nodes
  - may be viewed as multiple parallel perceptrons
- The output nodes may be combined with another perceptron
  - which may also have multiple output nodes
- The size of this hidden layer is determined manually



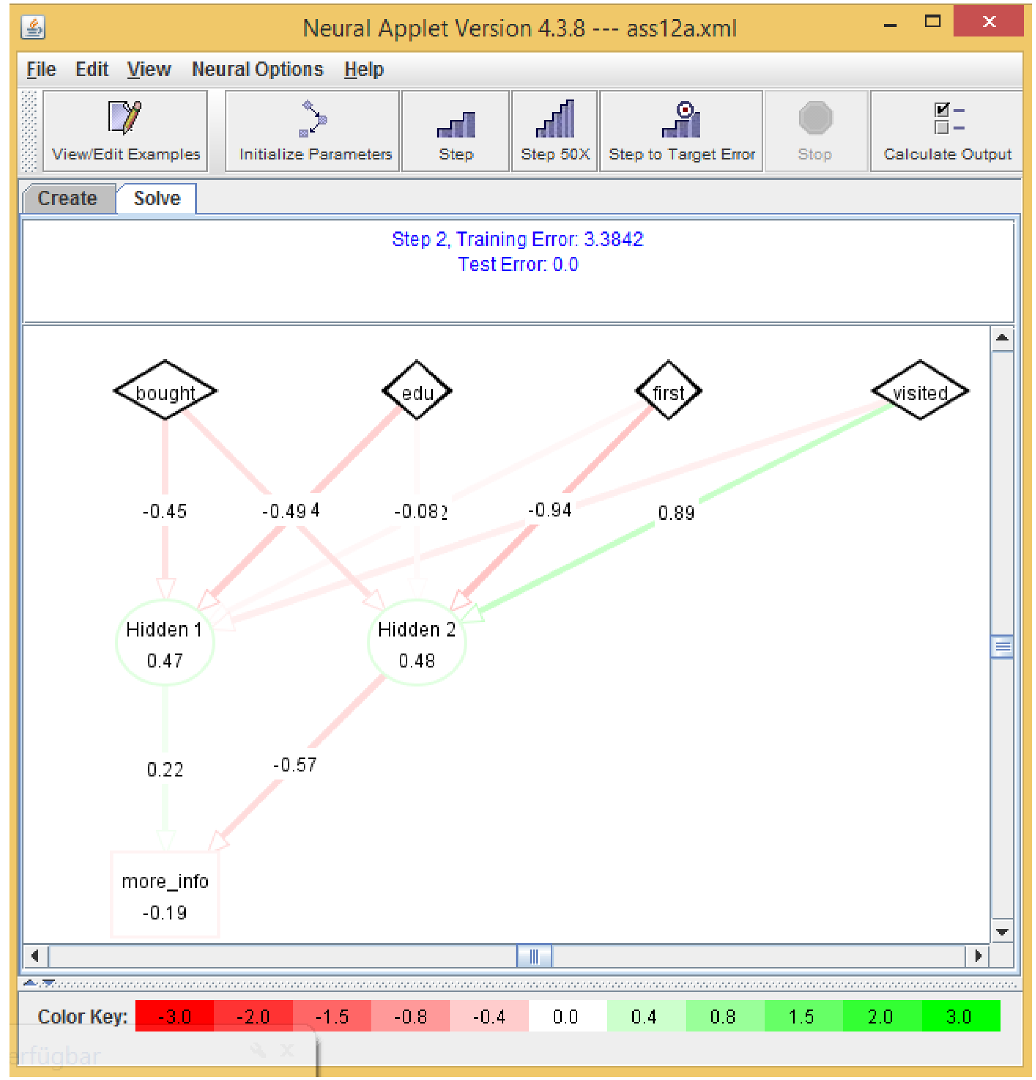
Multilayer Perceptrons

- Information flow is unidirectional
  - Data is presented to *Input layer*
  - Passed on to *Hidden Layer*
  - Passed on to *Output layer*
- Information is distributed
- Information processing is parallel

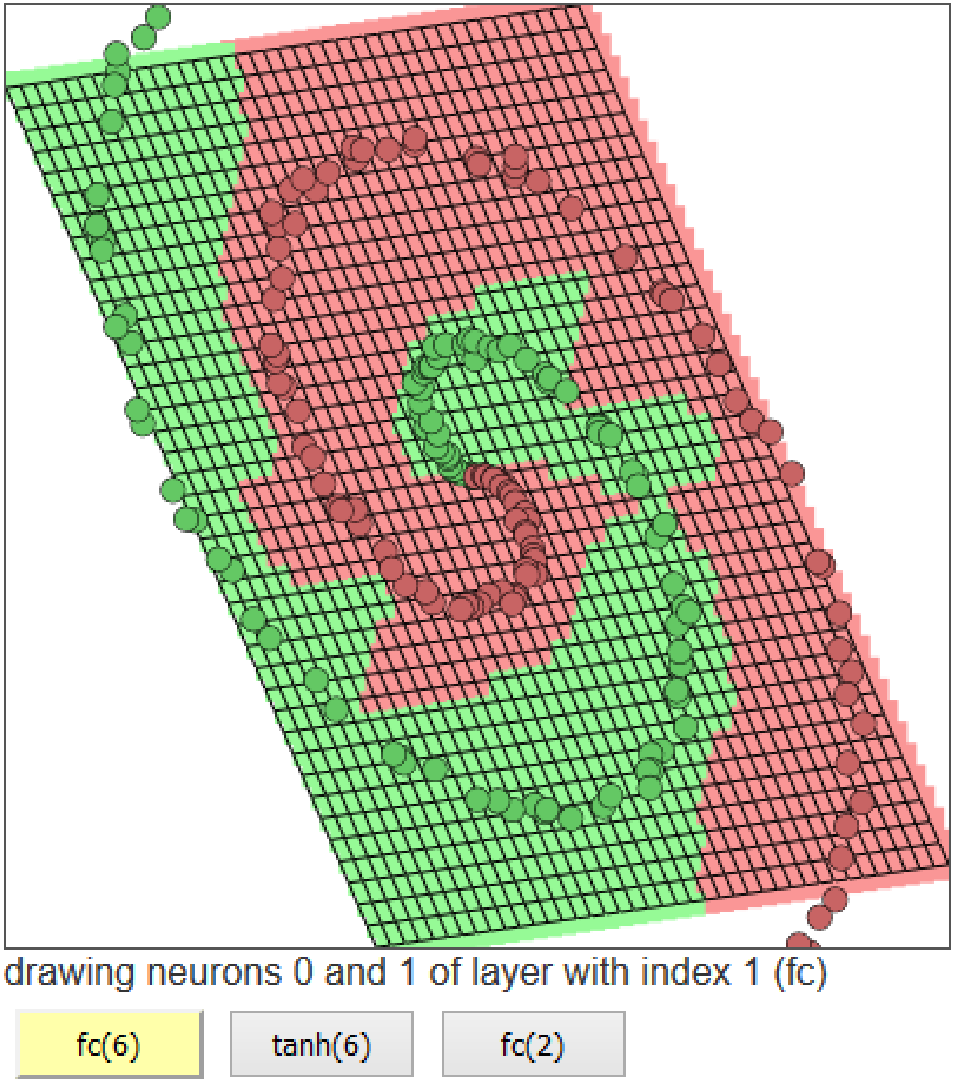


Multilayer Perceptrons

- Online tools for exercising
  - http://www.aispace.org/exercises/exercise7-b-1.shtml
  - https://cs.stanford.edu/people/karpathy/convnetjs/

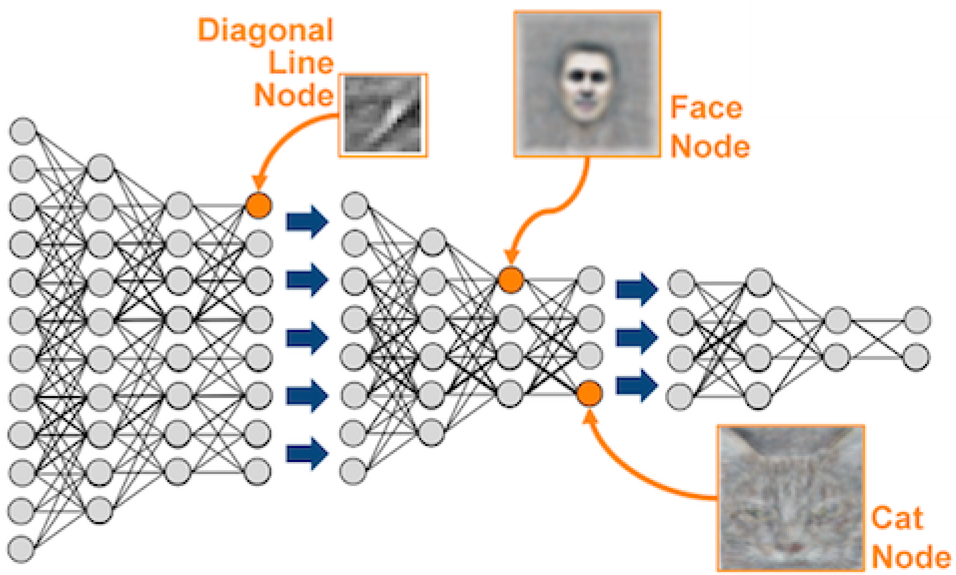


- Online tools for exercising
  - http://www.aispace.org/exercises/exercise7-b-1.shtml
  - https://cs.stanford.edu/people/karpathy/convnetjs/

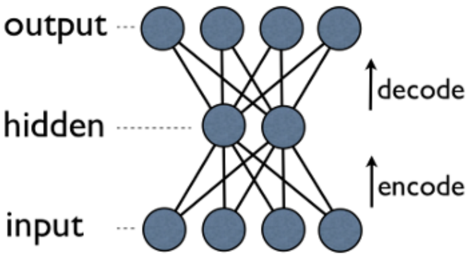


Deep Learning

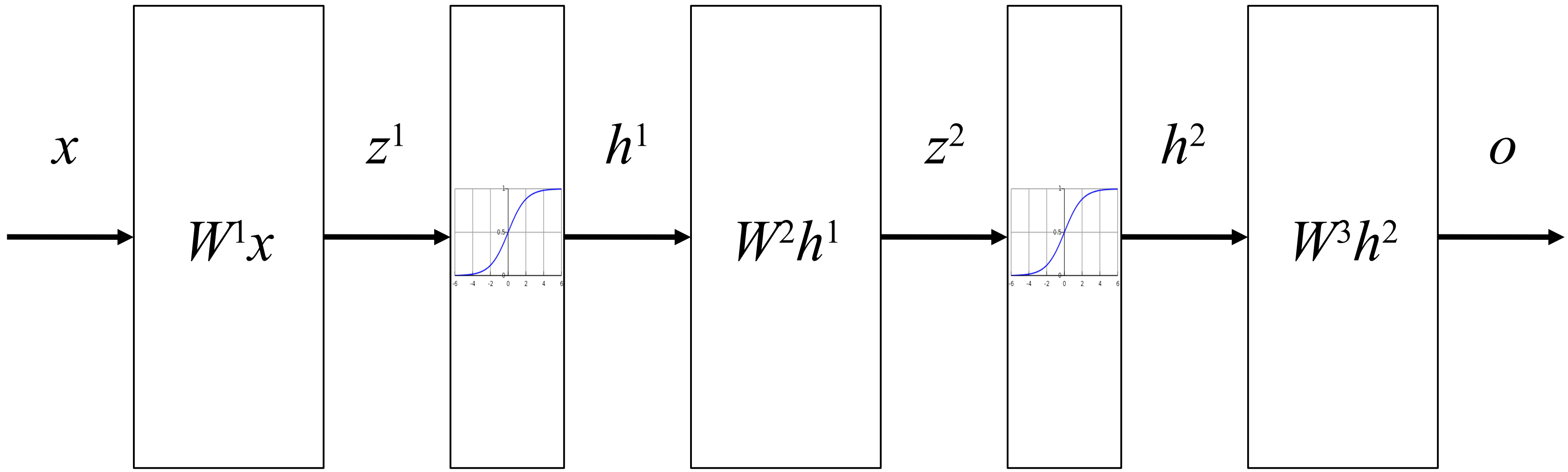
- In the last years, great success has been observed with training „deep“ neural networks
  - Deep networks are networks with multiple layers
- Successes in particular in image classification
  - Idea is that layers sequentially extract information from image
    - 1st layer → edges,
    - 2nd layer → corners, etc…



- Key ingredients:
  - A lot of training data are needed and available (big data)
  - Fast processing and a few new tricks made fast training for big data possible
  - Unsupervised pre-training of layers
    - **Autoencoder** use the previous layer as input and output for the next layer



Feed-forward Neural Network



x input

h^1 1st hidden activations

h^2 2nd hidden activations

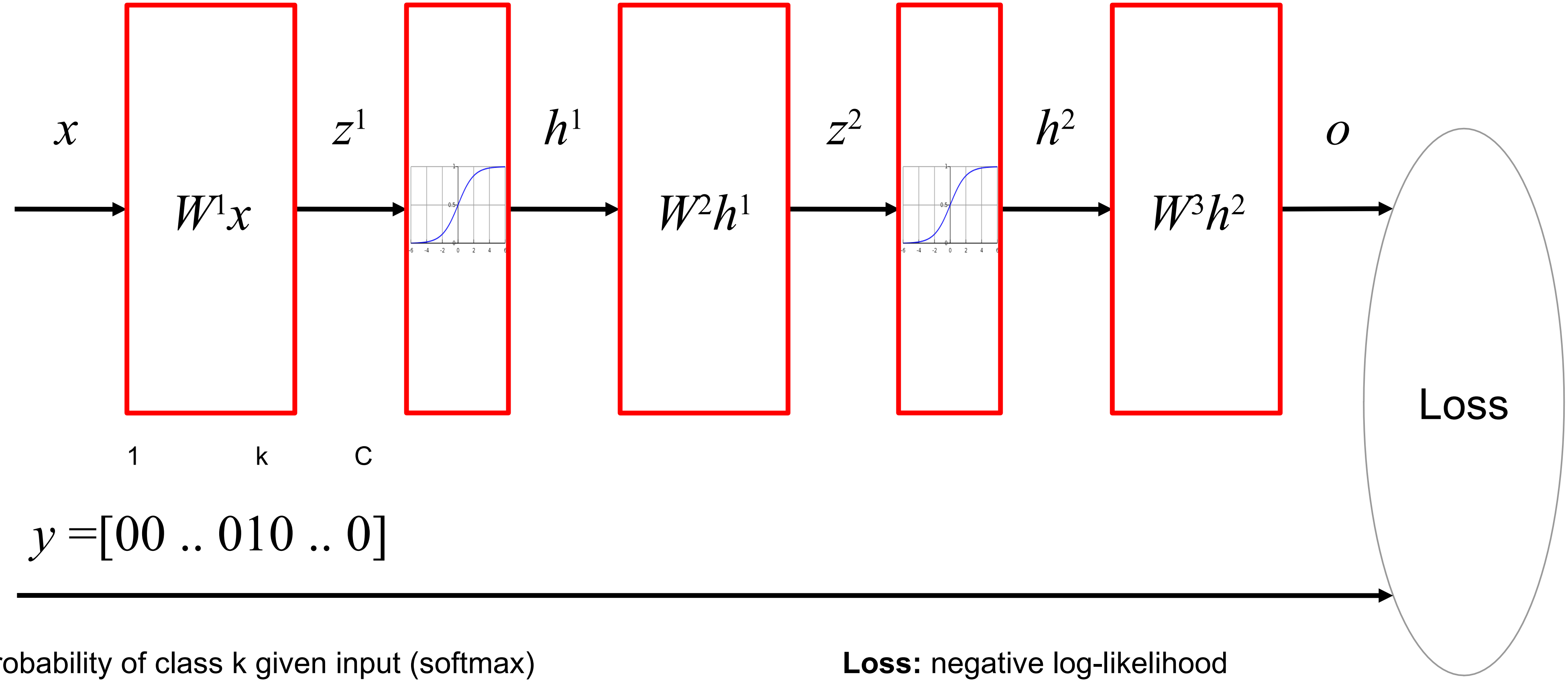
o output

z^1 1st linear projection

z^2 2nd linear projection

Ranzato, CVPR tutorial, 2014

How Good is a Network? (In Classification)



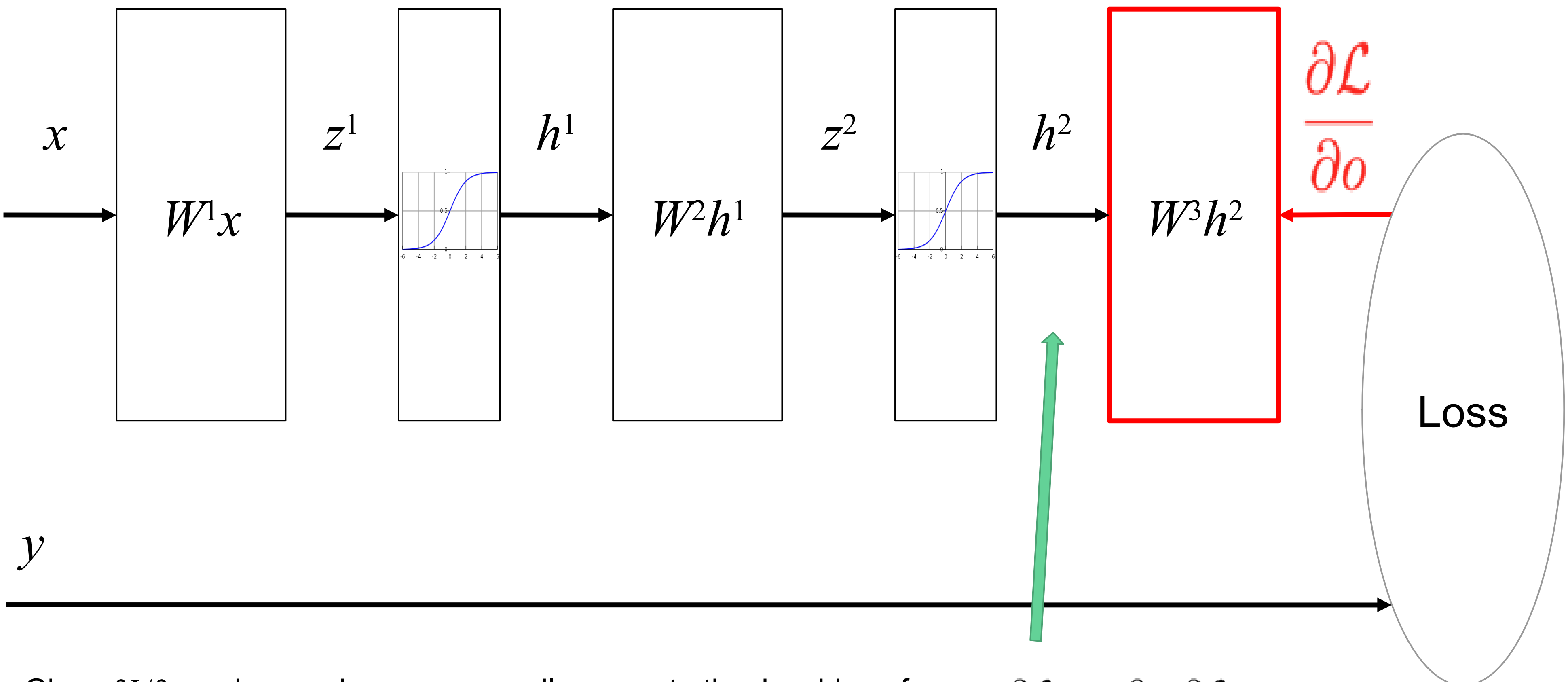
Probability of class k given input (softmax)

$$p(c_k = 1|x) = \frac{\exp(o_k)}{\sum_{j=1}^{C} \exp(o_j)}$$

Loss: negative log-likelihood

$$\mathcal{L}(\theta; x, y) = -\sum_j y_j \log p(c_j|x)$$

Propagating Errors Backwards



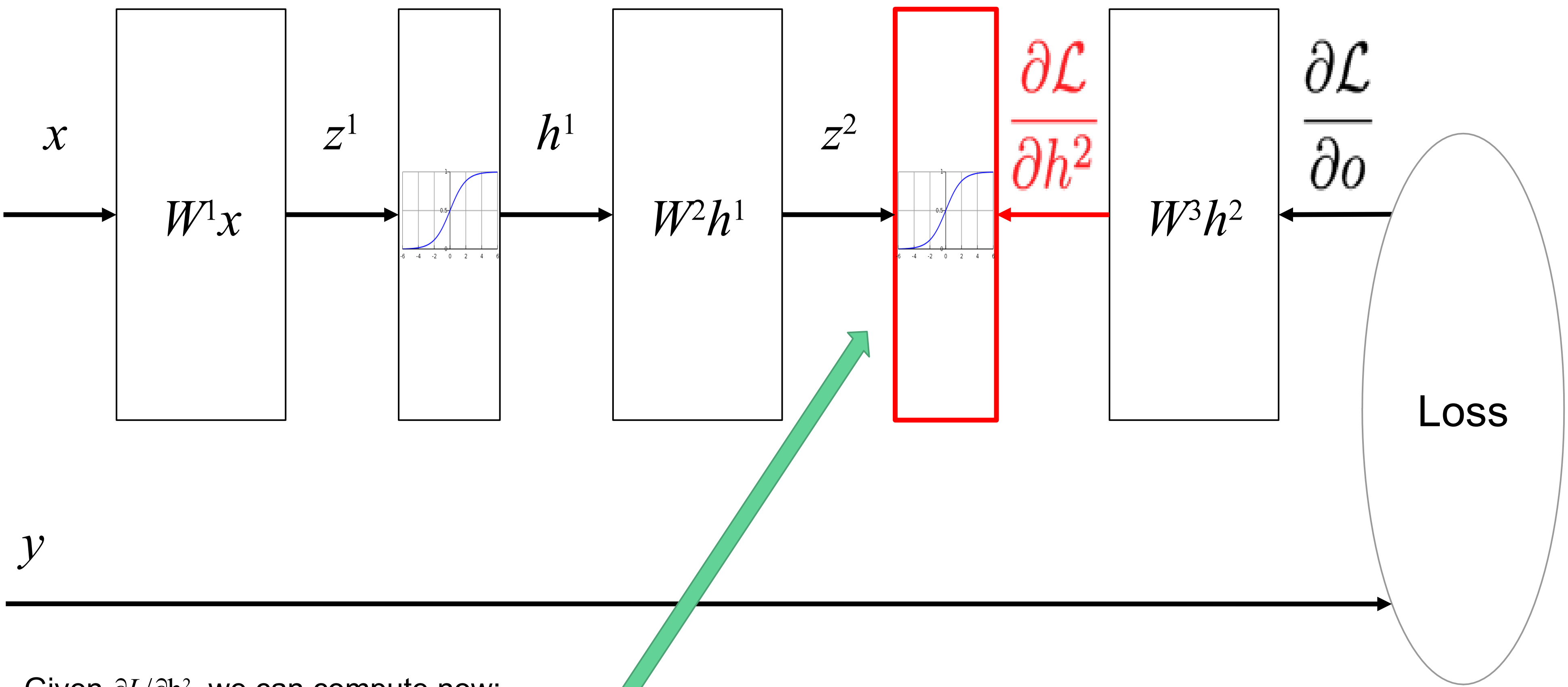
Given $\partial L/\partial o$ and assuming we can easily compute the Jacobian of each module, we have

$$\frac{\partial \mathcal{L}}{\partial h^2} = \frac{\partial o}{\partial h^2}\frac{\partial \mathcal{L}}{\partial o}$$

$$\frac{\partial \mathcal{L}}{\partial W^3} = \frac{\partial \mathcal{L}}{\partial o}\frac{\partial o}{\partial W^3} \Rightarrow \frac{\partial \mathcal{L}}{\partial W^3} = (p(c|x) - y)h^{2T}$$

$$\frac{\partial \mathcal{L}}{\partial h^2} = W^{3T}(p(c|x) - y)$$

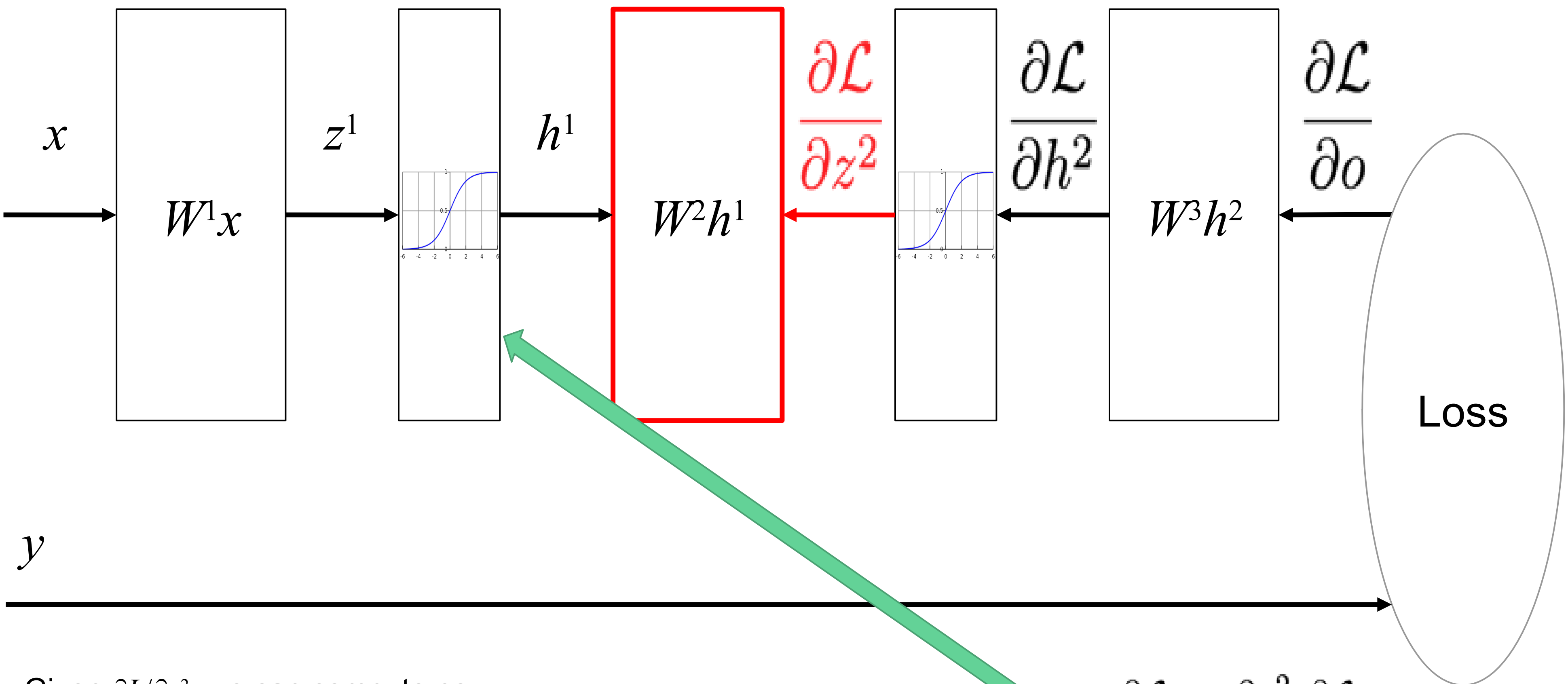
Propagating Errors Backwards



Given $\partial L/\partial h^2$, we can compute now:

$$\frac{\partial \mathcal{L}}{\partial z^2} = \frac{\partial \mathcal{L}}{\partial h^2}\frac{\partial h^2}{\partial z^2} \quad \Rightarrow \quad \frac{\partial \mathcal{L}}{\partial z^2} = \frac{\partial \mathcal{L}}{\partial h^2} \odot f'(z^2)$$

Propagating Errors Backwards

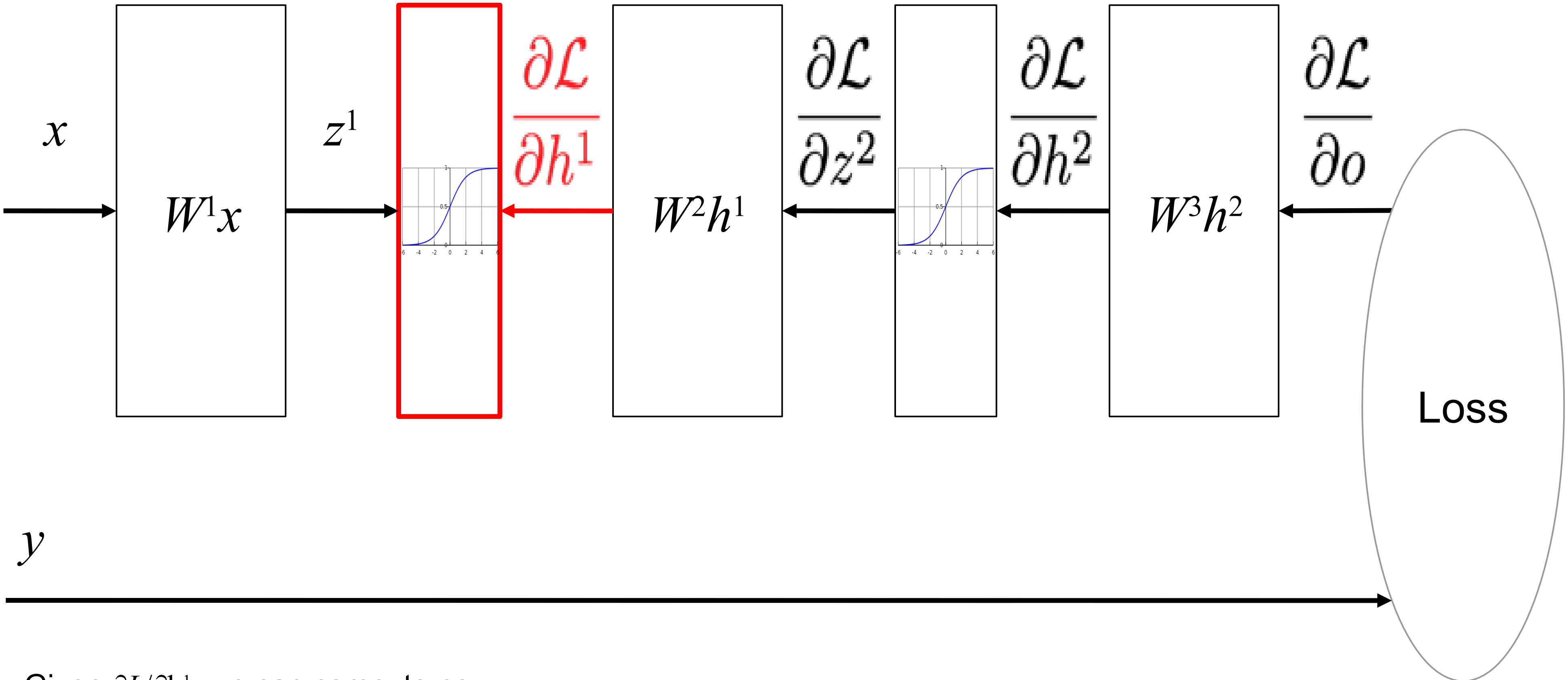


Given $\partial L/\partial z^2$, we can compute now:

$$\frac{\partial \mathcal{L}}{\partial h^1} = \frac{\partial z^2}{\partial h^1}\frac{\partial \mathcal{L}}{\partial z^2} = W^{2T}\frac{\partial \mathcal{L}}{\partial z^2}$$

$$\frac{\partial \mathcal{L}}{\partial W^2} = \frac{\partial \mathcal{L}}{\partial z^2}\frac{\partial z^2}{\partial W^2} \Rightarrow \frac{\partial \mathcal{L}}{\partial W^2} = \left(\frac{\partial \mathcal{L}}{\partial h^2} \odot f'(z^2)\right) h^{1T}$$

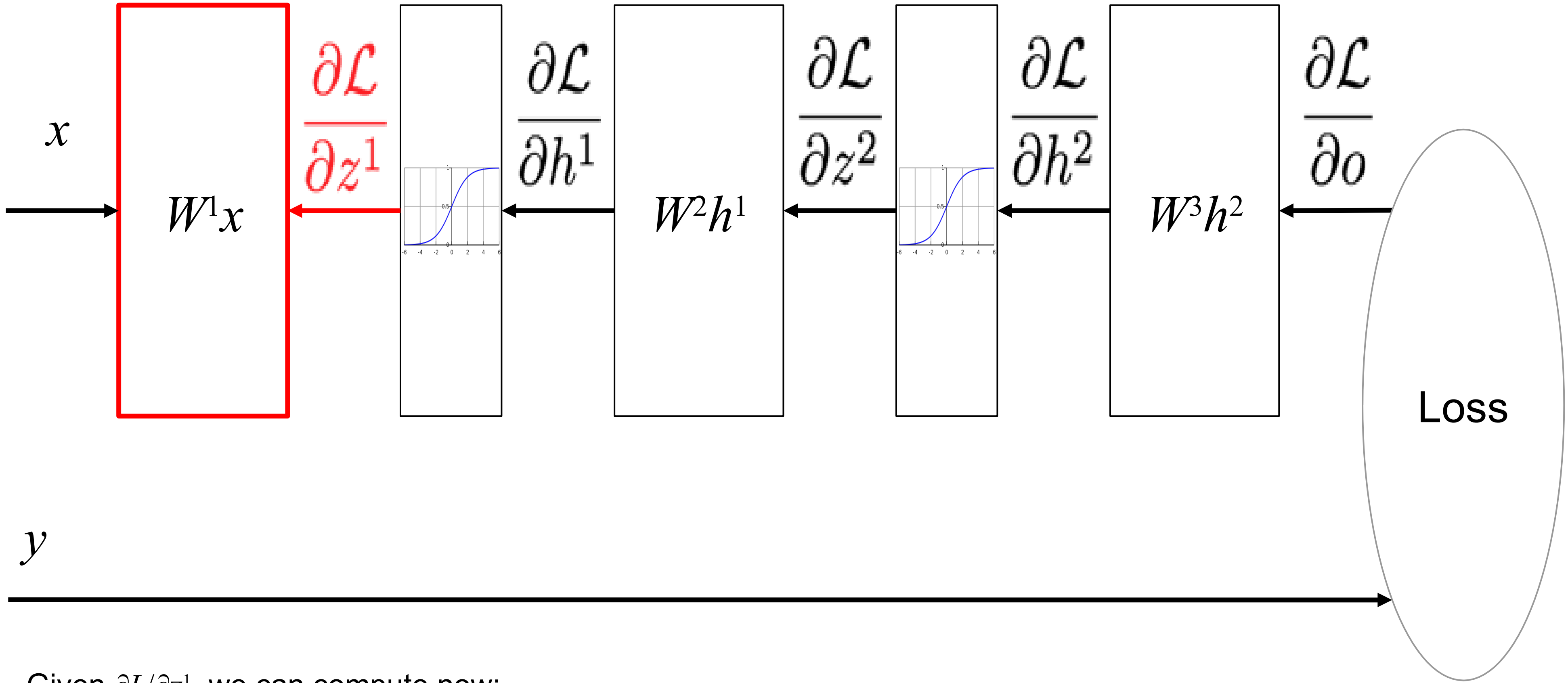
Propagating Errors Backwards



Given $\partial L/\partial h^1$, we can compute now:

$$\frac{\partial \mathcal{L}}{\partial z^1} = \frac{\partial \mathcal{L}}{\partial h^1}\frac{\partial h^1}{\partial z^1}$$

Propagating Errors Backwards



Given $\partial L/\partial z^1$, we can compute now:

$$\frac{\partial \mathcal{L}}{\partial W^1} = \frac{\partial \mathcal{L}}{\partial z^1}\frac{\partial z^1}{\partial W^1} \Rightarrow \frac{\partial \mathcal{L}}{\partial W^1} = \left(\frac{\partial \mathcal{L}}{\partial h^1} \odot f'(z^1)\right) x^T$$

Optimization

(Minibatch) Stochastic Gradient Descent

$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

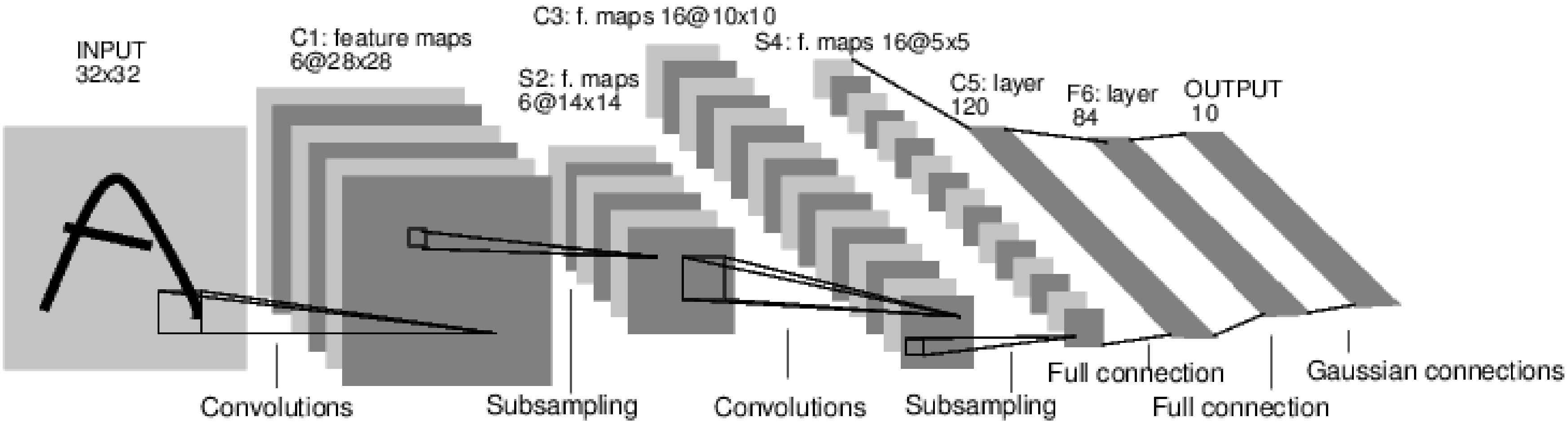
Parameters

Learning rate
(0, 1)

Gradients

Image Processing Networks

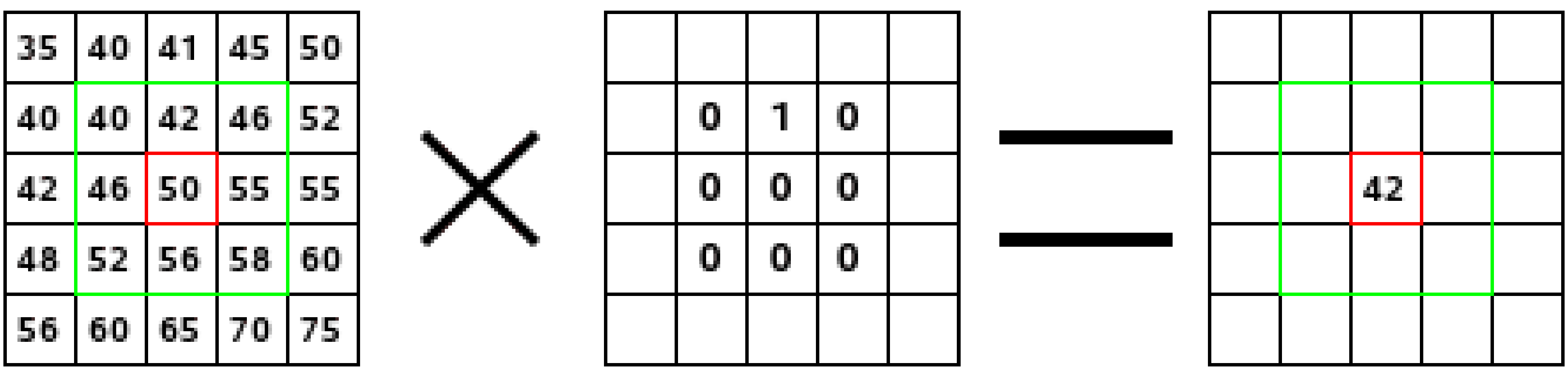
- Convolutions can be encoded as network layers
  - all possible 3x3 pixels of the input image are connected to the corresponding pixel in the next layer
- Convolutional Layers are at the heart of Image Recognition
  - Several stacked on top of each other and parallel to each other
- **Example:** LeNet (LeCun et al. 1989)



  - GoogLeNet is a modern variant of this architecture

Convolutional Neural Networks

- Convolution:
  - for each pixel of an image, a new feature is computed using a weighted combination of its nxn neighborhood



5x5 image

3x3 convolution
runs over all possible 3x3 subimages of picture

resulting image
only one pixel shown

Convolution - Blur

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

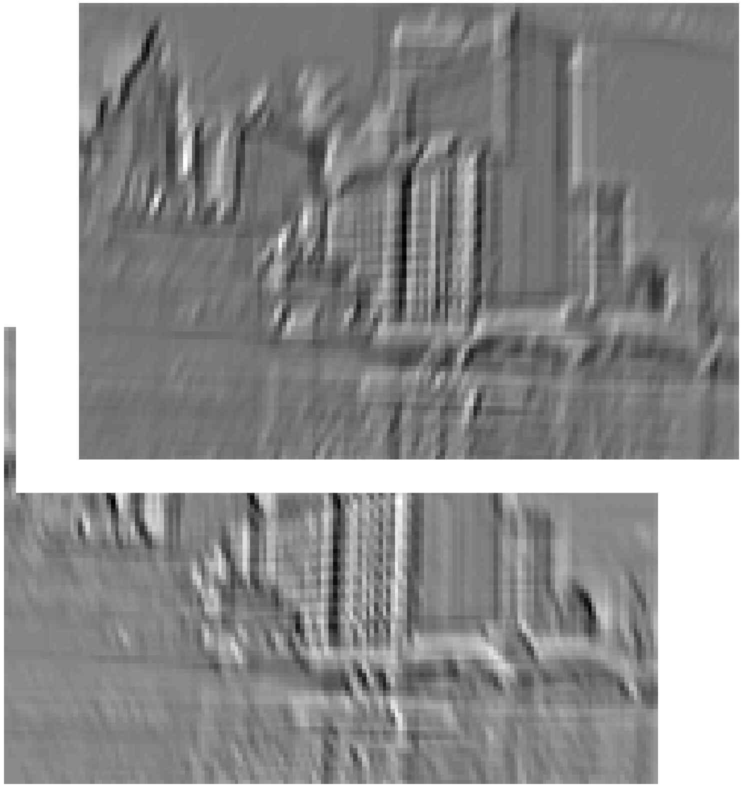
Convolution - Edge detection

	0	1	0	
	1	-4	1	
	0	1	0	

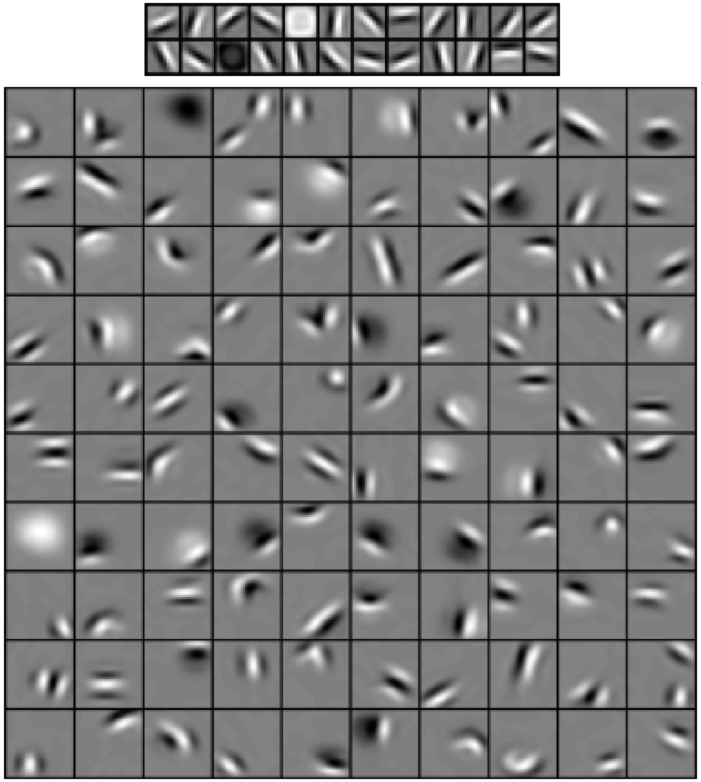
Outputs of Convolution

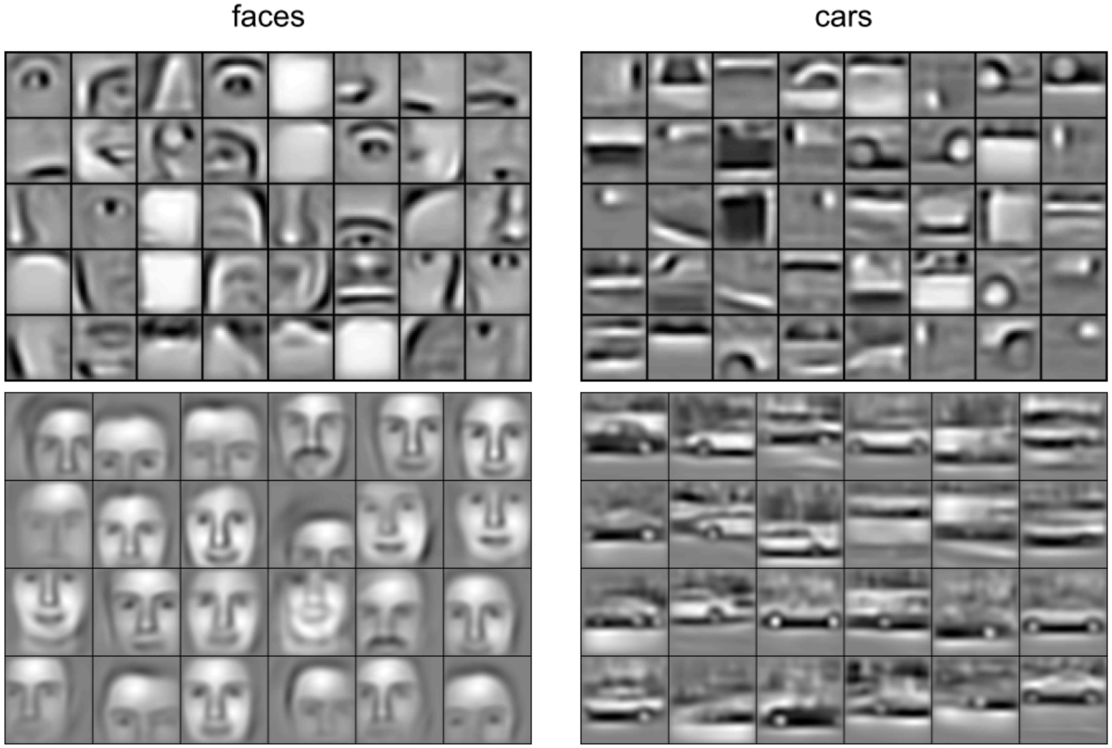
Outputs of Convolution

Outputs of Convolution

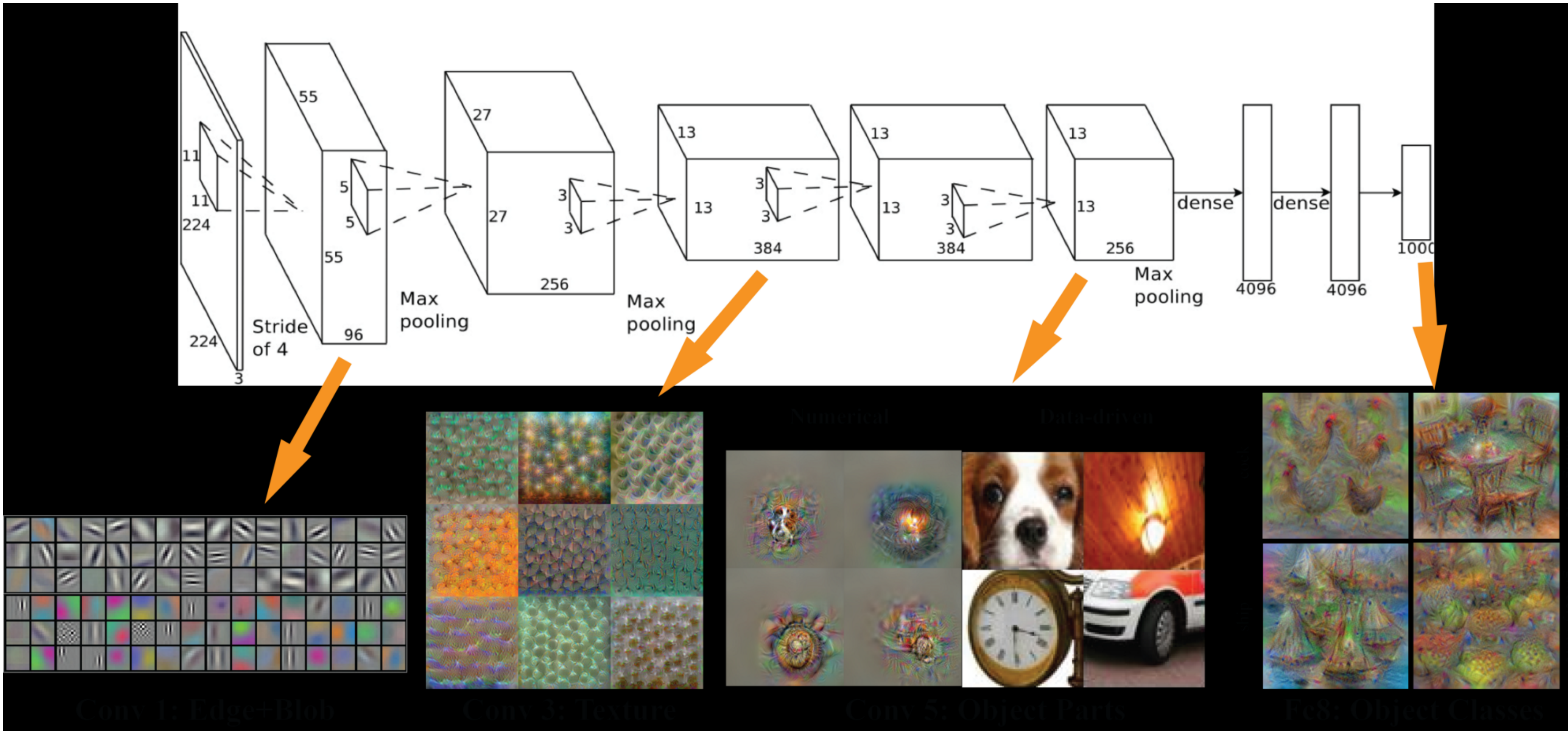


Visualizations of CNN networks





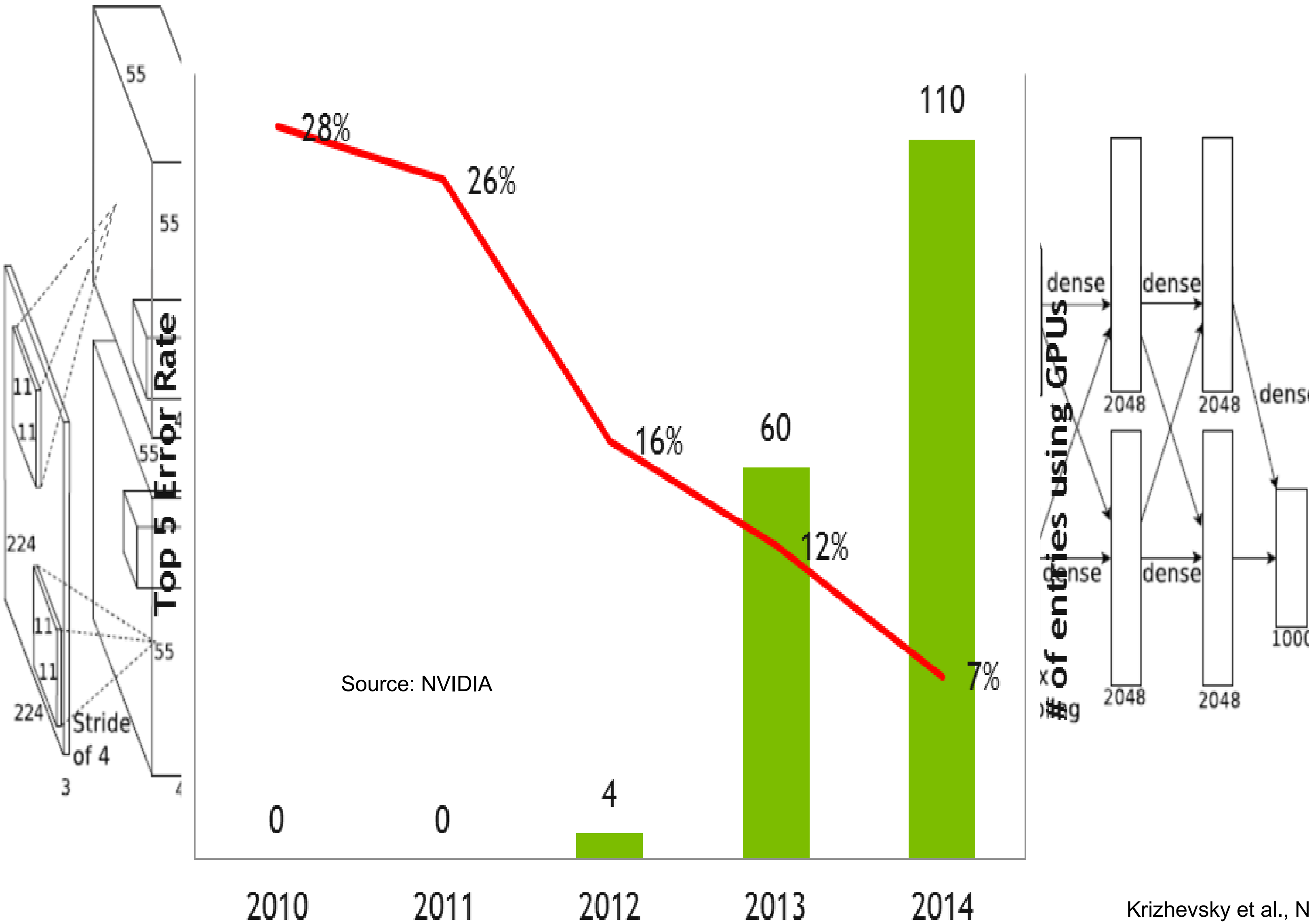
Visualizations of CNN networks



AlexNet, 2

IMAGENET

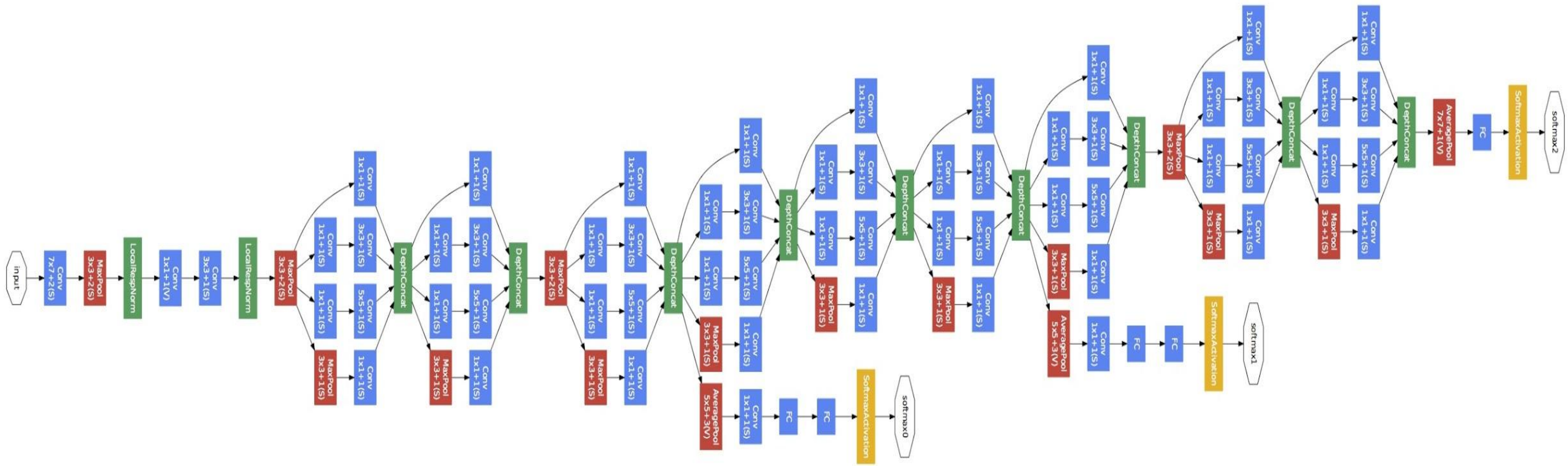
TECHNISCHE UNIVERSITÄT DARMSTADT



Krizhevsky et al., NIPS, 2012

GoogLeNet, 2014

TECHNISCHE UNIVERSITÄT DARMSTADT



22 layers
(only blue rectangular)

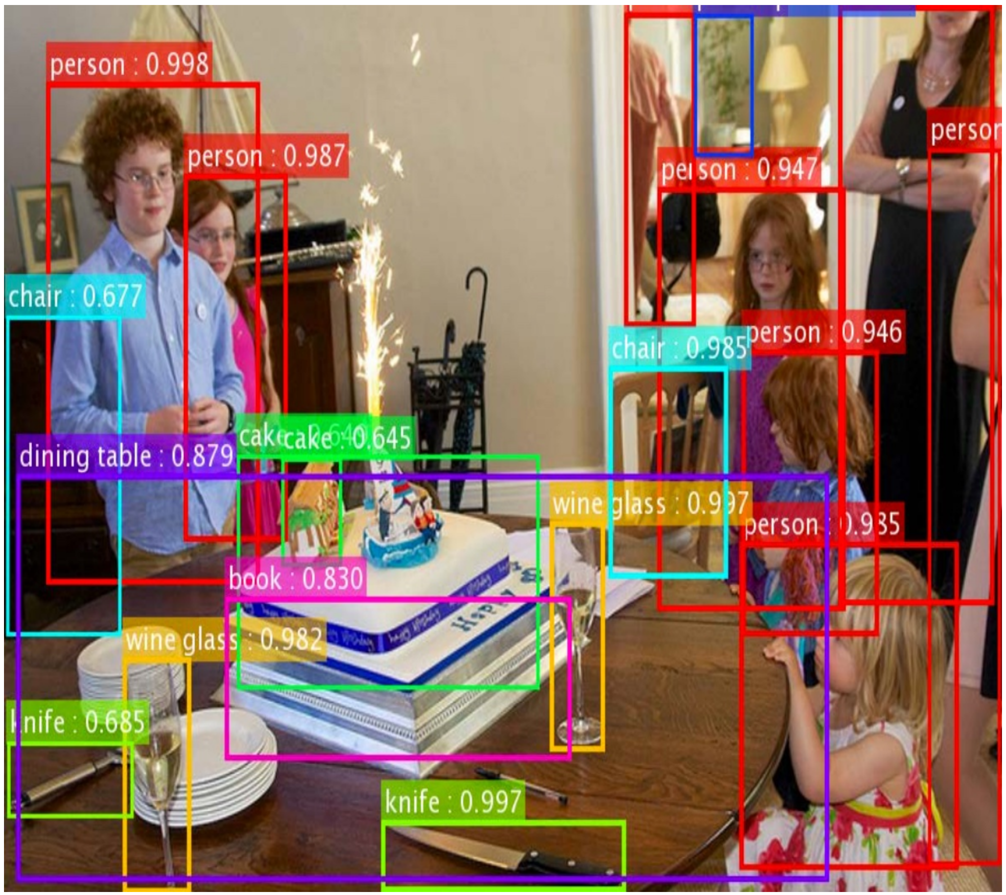
Szegedy et al., CVPR, 2015

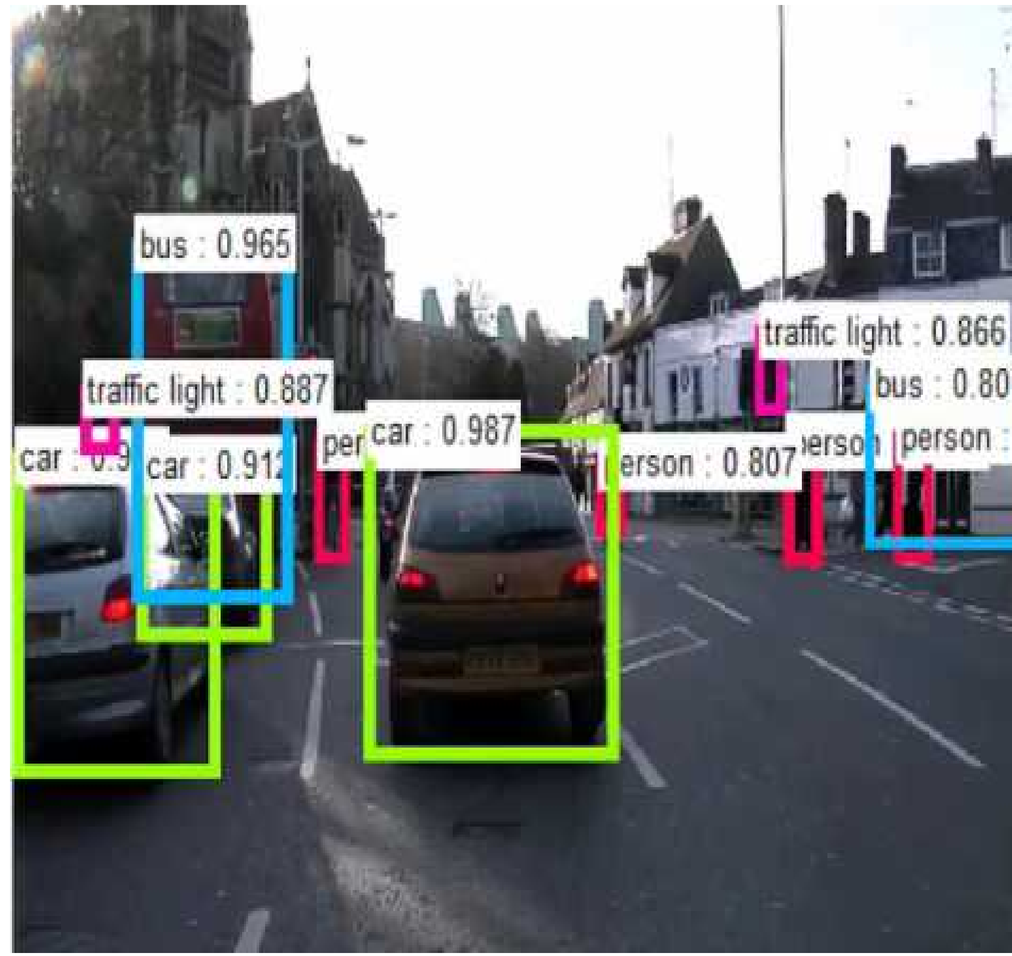
ResNet, 2015

152 layers

He et al., CVPR, 2016

Object Detection





He, ICML tutorial, 2016

Word2Vec

(Mikolov et al. 2013)

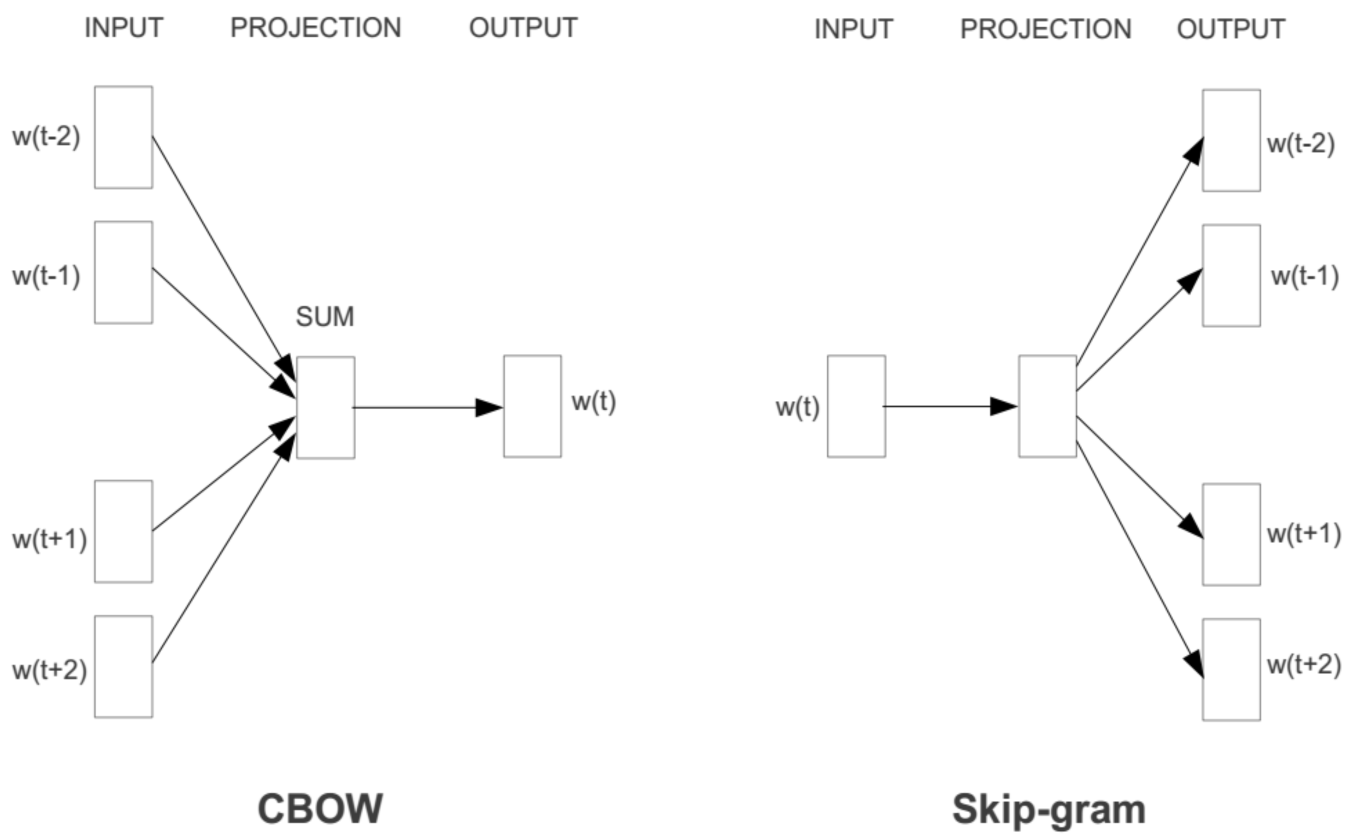
TECHNISCHE UNIVERSITÄT DARMSTADT

- Key Idea:
  - find a distributed word representation, i.e., each word is represented as a lower-dimensional, non-sparse vector
  - similar to PCA
  - allows, e.g., to compute cosine similarities between words
- General Approach:
  - train a (deep) neural network in a supervised way
  - using the context of a word as additional input
- Efficient Implementation available
  - https://radimrehurek.com/gensim/
  - processes the whole Wikipedia quite quickly

2 Variants of Word2Vec

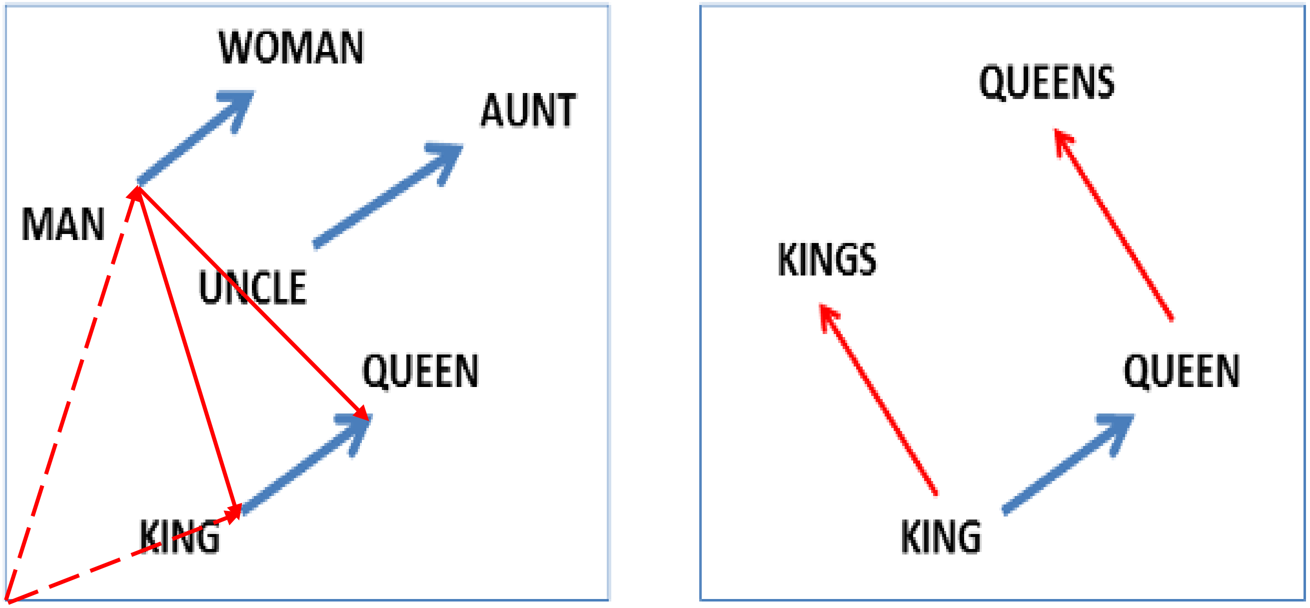
Continuous Bag of Words:
- predict the current word from a window of surrounding words

Skip-gram:
- use the current word to predict the context window



Word2Vec Representation allows Analogical Reasoning

vec(King) - vec(Man) + vec(Woman) ≈ vec(Queen)

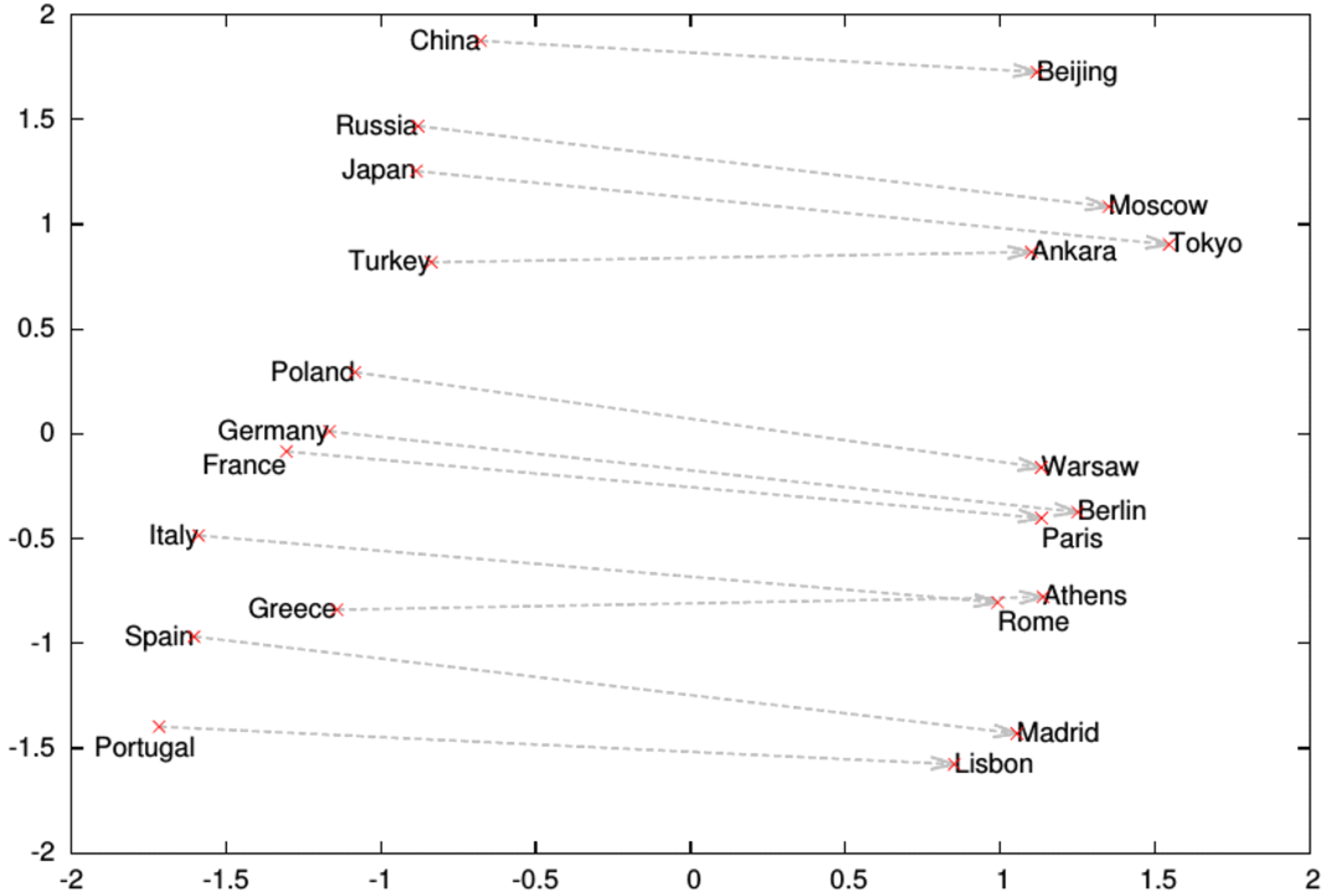


Mikolov et al., NAACL, 2013

Word2Vec Representation allows Analogical Reasoning

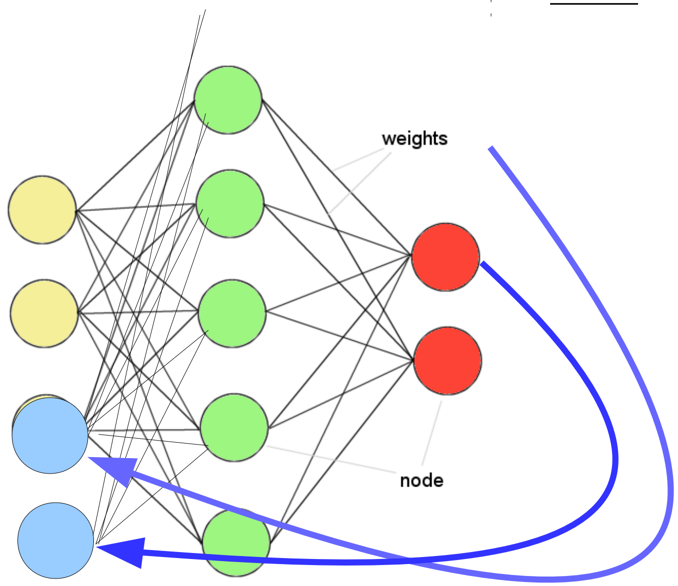
TECHNISCHE UNIVERSITÄT DARMSTADT

(Mikolov et al. 2014

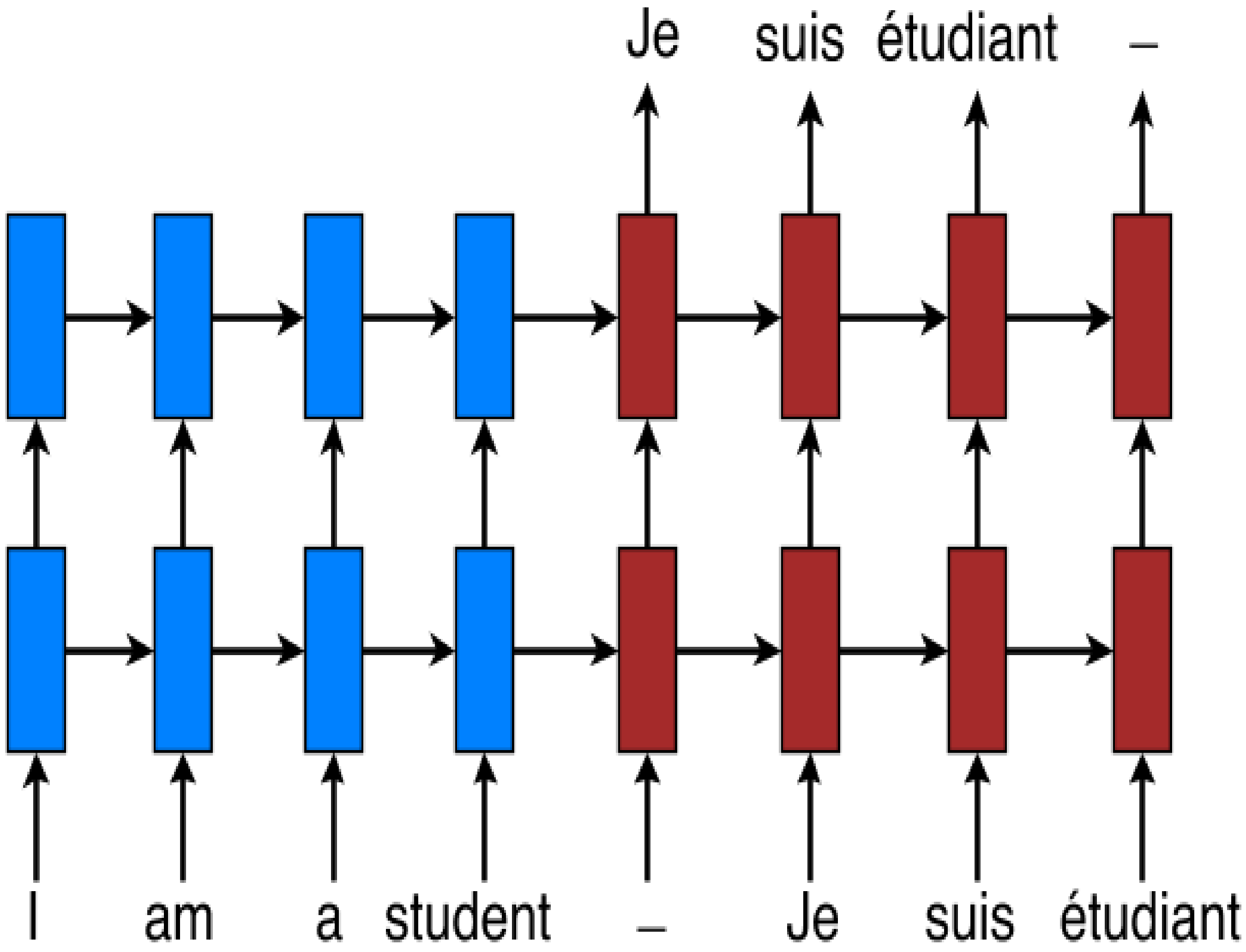


Recurrent Neural Networks

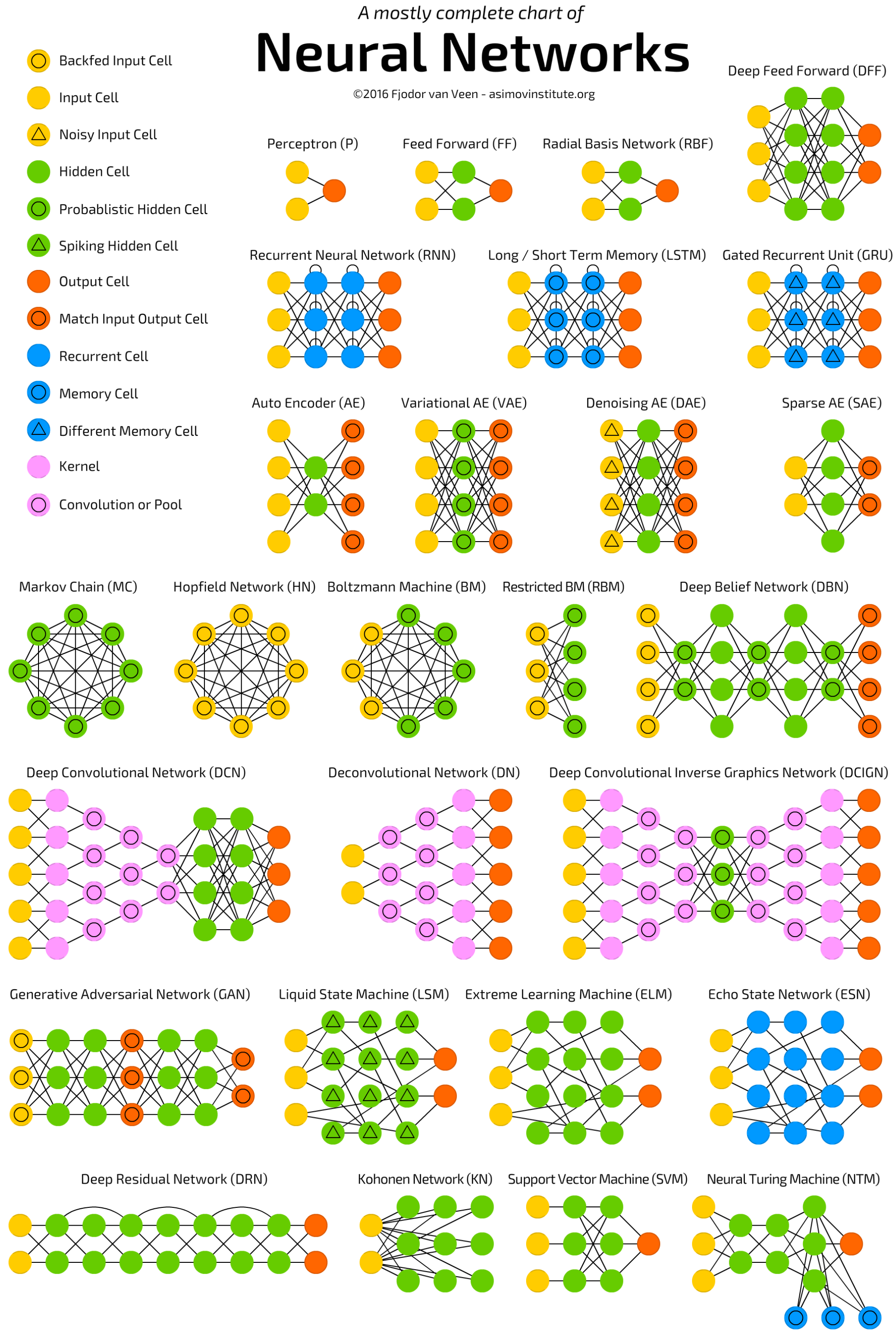
- Recurrent Neural Networks (RNN)
  - allow to process sequential data
  - by feeding back the output of the network into the next input
- Long-Short Term Memory (LSTM)
  - add „forgetting“ to RNNs
  - good for mapping sequential input data into sequential output data
    - e.g., text to text, or time series to time series
- Deep Learning often allows „end-to-end learning“
  - e.g., learn a network that does the complete translation of text in one language into another language
  - previously, learning often concentrated on individual components (e.g. word sense disambiguation)



Neural Machine Translation



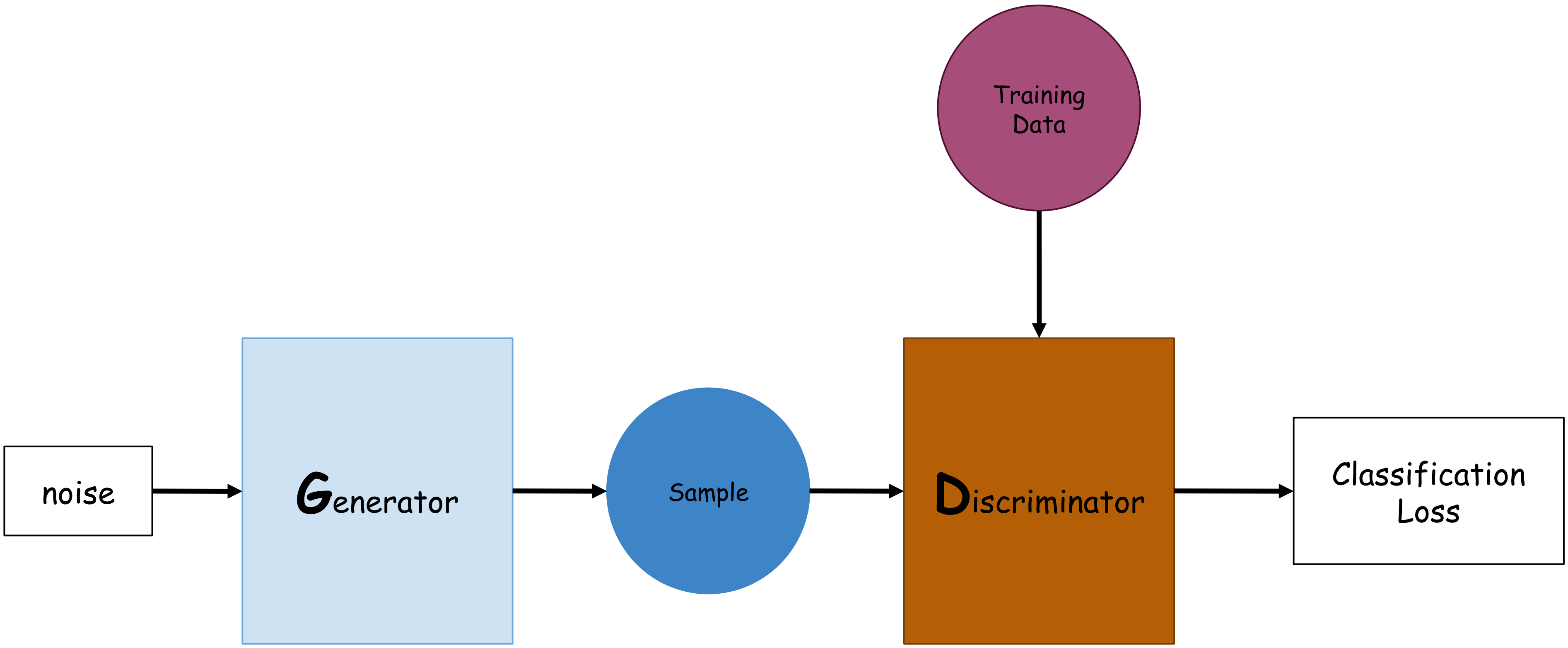
Sutskever et al., NIPS, 2014
Luong and Manning, EMNLP, 2015



https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-678c51b4b463

Generative Adversarial Network

Goodfellow et al., NIPS, 2014

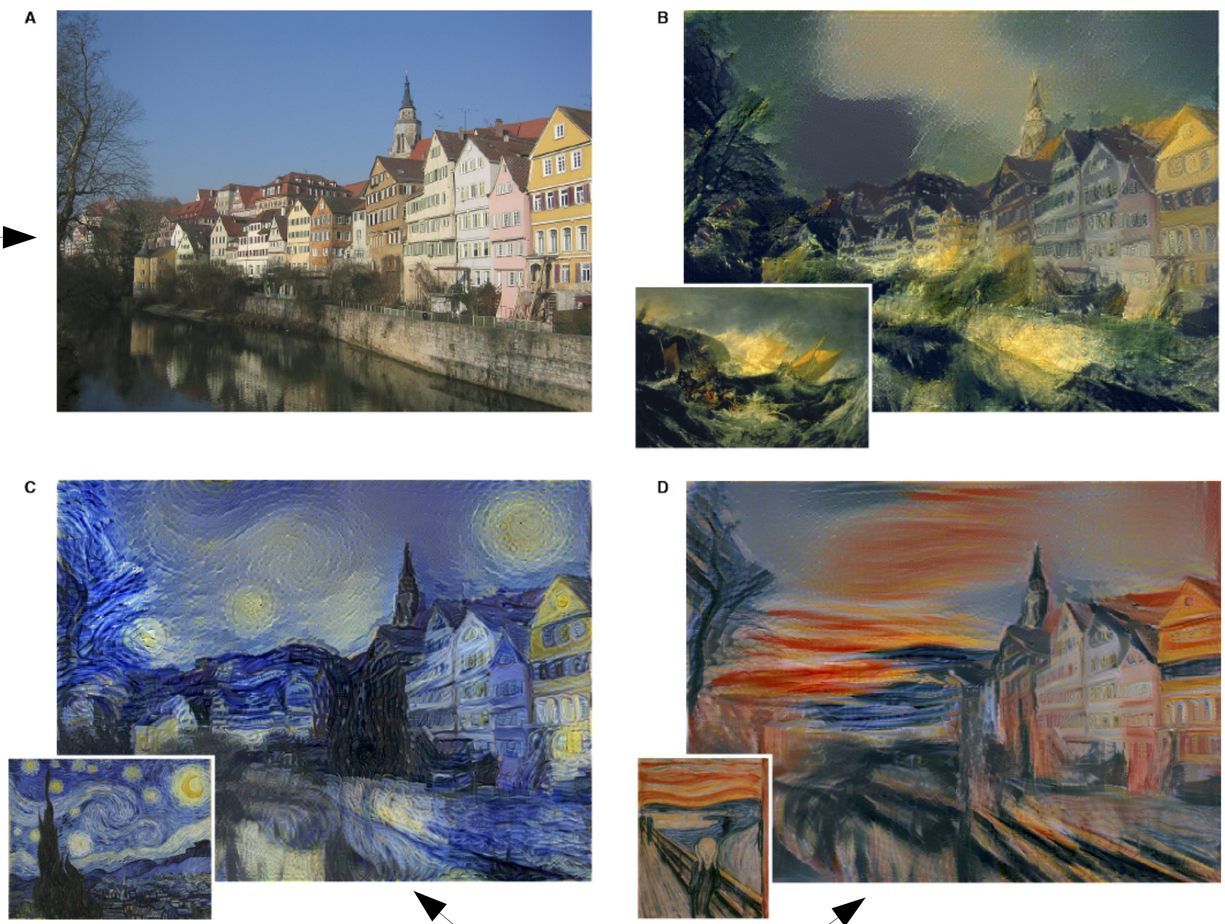


Learning to distinguish
real data from generated ones

Neural Artistic Art Transfer

(Gatys et al., 2016)

Content Image

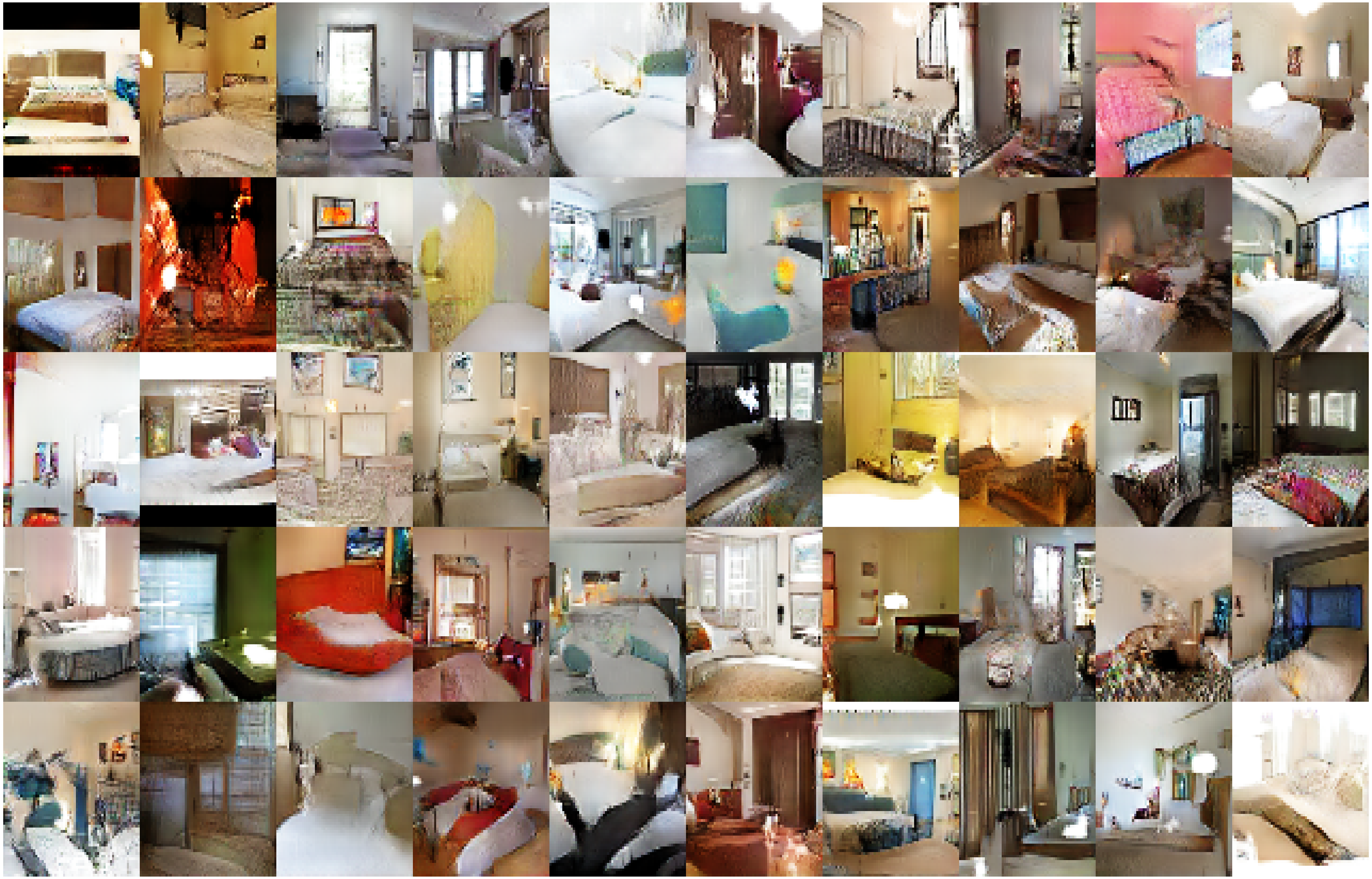


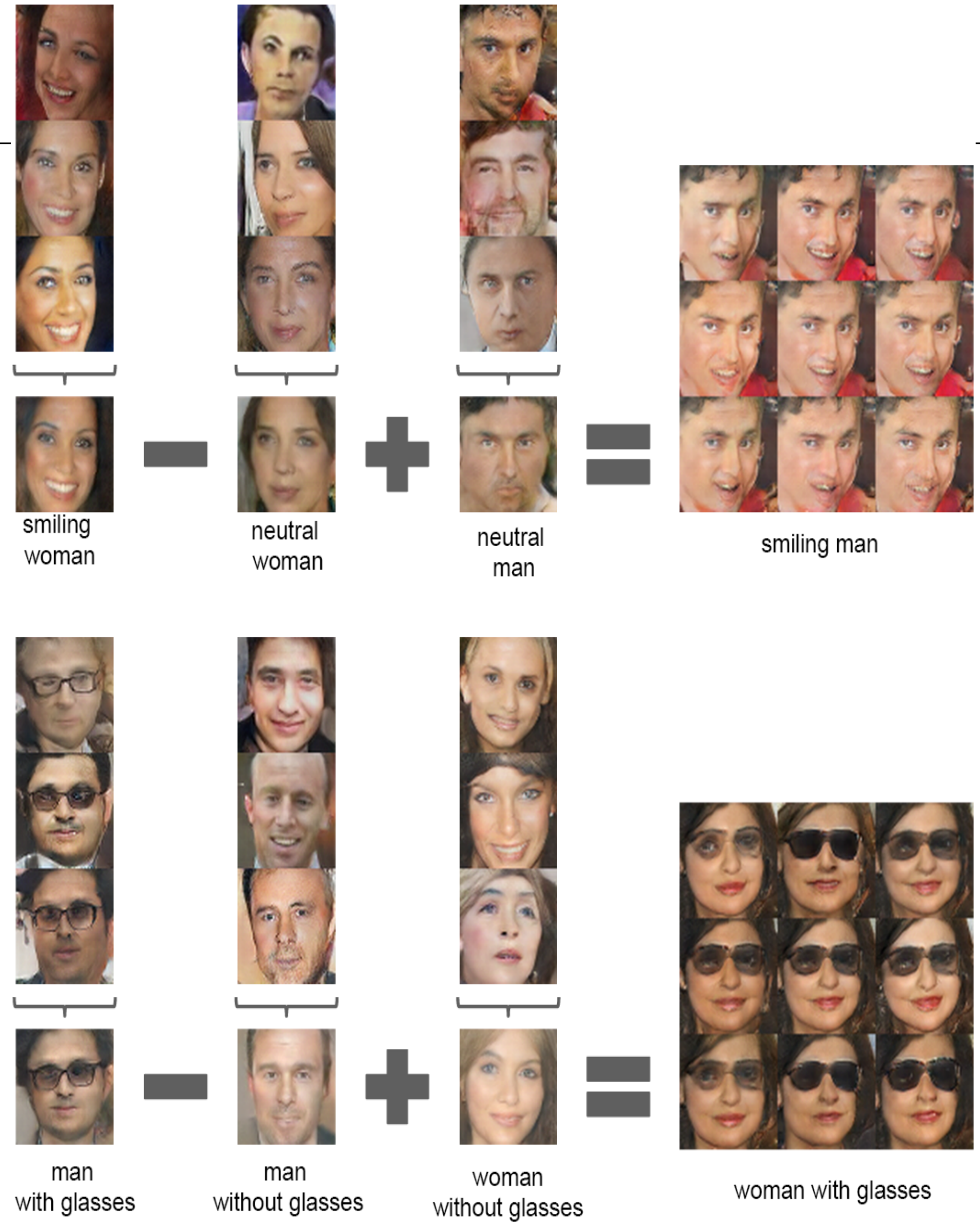
Style Image

Synthesized Image

Bedrooms generated by DCGAN

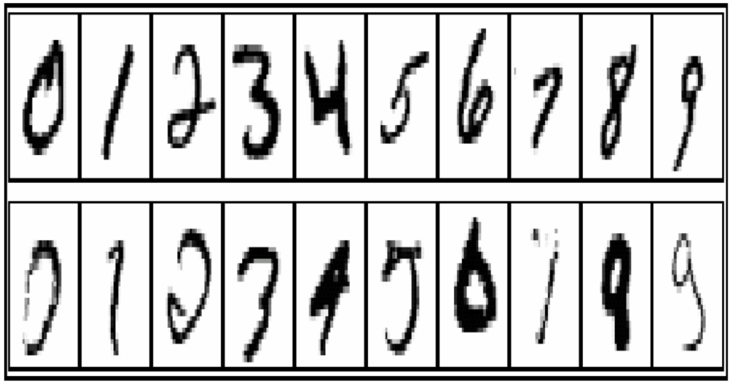
Radford et al., 2015




smiling woman
neutral woman
neutral man
smiling man
man with glasses
man without glasses
woman without glasses
woman with glasses

Wide Variety of Applications

- Speech Recognition
- Autonomous Driving
- Handwritten Digit Recognition
- Credit Approval
- Backgammon
- etc.



- Good for problems where the final output depends on combinations of many input features
  - rule learning is better when only a few features are relevant
- Bad if explicit representations of the learned concept are needed
  - takes some effort to interpret the concepts that form in the hidden layers

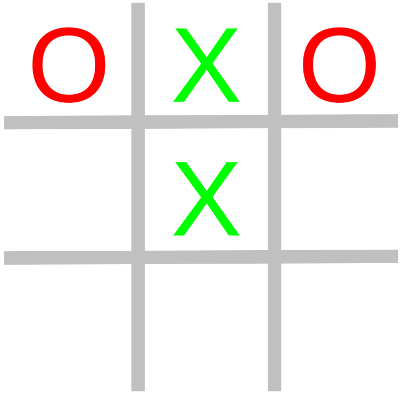
Reinforcement Learning

TECHNISCHE UNIVERSITÄT DARMSTADT

- Goal
  - Learning of policies (action selection strategies) based on feedback from the environment (reinforcement)
    - e.g., game won / game lost
- Applications
  - **Games**
    - Tic-Tac-Toe: MENACE (Michie 1963)
    - Backgammon: TD-Gammon (Tesauro 1995)
    - Schach: KnightCap (Baxter et al. 2000)
  - **Other**
    - Elevator Dispatching
    - Robot Control
    - Job-Shop Scheduling

MENACE (Michie, 1963)

- Learns to play Tic-Tac-Toe

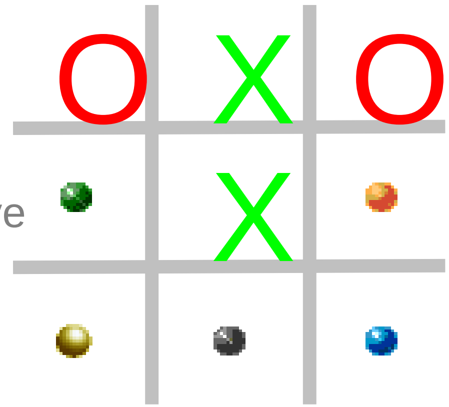


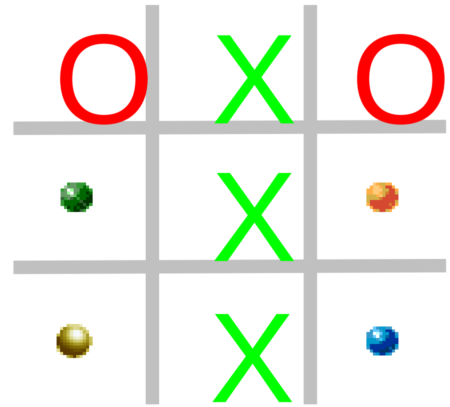
- Hardware:
  - 287 Matchboxes
    (1 for each position)
  - Beads in 9 different colors
    (1 color for each square)

- Playing algorithm:
  - Select the matchbox corresponding to the current position
  - Randomly draw a bead from this matchbox
  - Play the move corresponding to the color of the drawn bead

- Implementation: http://www.codeproject.com/KB/cpp/ccross.aspx

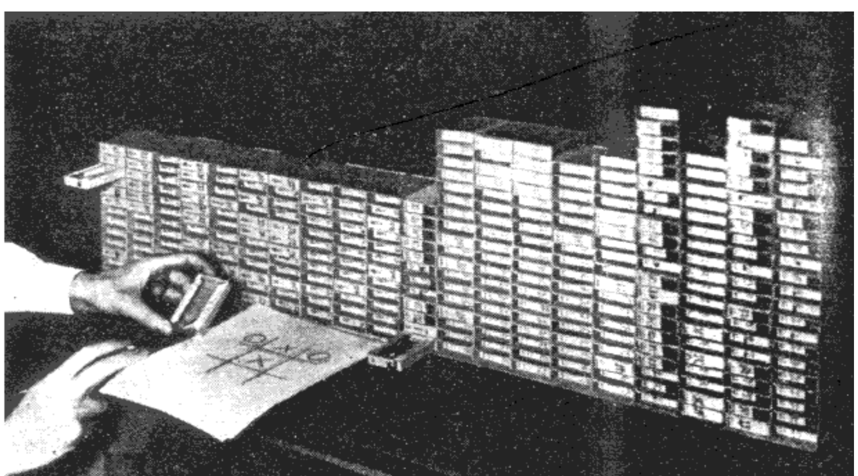
X to move





Select the matchbox corresponding to this position

Play the move that corresponds to the color of the drawn bead



Draw a bead from the matchbox

Reinforcement Learning in MENACE

- Initialisation
  - all moves are equally likely, i.e. every box contains an equal number of beads for each possible move / color
- Learning algorithm:
  - Game lost → drawn beads are kept (*negative reinforcement)*
  - Game won → put the drawn bead back and add another one in the same color to this box (*positive reinforcement*)
  - Game drawn → drawn beads are put back (no change)
- This results in
  - Increased likelihood that a successful move will be tried again
  - Decreased likelihood that an unsuccessful move will be repeated

Credit Assignment Problem

- Delayed Reward
  - The learner knows whether it has one or lost not before the end of the game
  - The learner does not know which move(s) are responsible for the win / loss
    - a crucial mistake may already have happened early in the game, and the remaining moves were not so bad (or vice versa)

- Solution in Reinforcement Learning:
  - All moves of the game are rewarded or penalized (adding or removing beads from a box)
  - Over many games, this procedure will converge
    - bad moves will rarely receive a positive feedback
    - good moves will be more likely to be positively reinforced

MENACE - Formalization

- Framework
  - states = matchboxes, discrete
  - actions = moves/beads, discrete
  - policy = prefer actions with higher number of beads, stochastic
  - reward = game won/ game lost
    - *delayed* reward: we don't know right away whether a move was good or bad+
  - transition function: choose next matchbox according to rules, deterministic

- Task
  - Find a policy that maximizes the sum of future rewards